CS Exit Ethiopia

Questions & Worked Answers

Computer Science Exit Examination

177 complete questions and 1 source variants

1 selected course | 18 November 2026

Selected courses

01. Computer Programming

https://csexitethiopia.com

abelasnake2003@gmail.com | Telegram: @Abechu7

Course contents

Each course begins on a new page. PDF bookmarks provide question, study-note section and revision-topic navigation.

Computer Programming 3

Reading the answers

Every question includes its answer, explanation, choice-by-choice contrasts and available evidence. Primary note proof quotes the study notes. Related reading and module passages are labelled separately; a module reference is not presented as primary note proof.

Computer Programming

Questions & Worked Answers

177 questions + 1 source variants

Question 1 | Q00017

In the conventional historical classification of programming-language generations, which generation is associated with declarative logic, constraints, AI and expert systems?

A. Fifth generation languages

B. Second generation languages

C. Fourth generation languages

D. Third generation languages

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: A. Fifth generation languages

The question uses the historical generation taxonomy, rather than asking which languages can implement AI. The note associates fifth-generation languages with logic, constraints and expert systems. Second generation refers to assembly, third to general high-level languages, and fourth to query languages and application generators. Those other languages can be used within AI applications, but that does not change their generation category.

Why each choice is right or wrong

A - Correct: Fifth generation is the historical logic/constraints and expert-system category.

B - Incorrect: Second generation denotes symbolic assembly.

C - Incorrect: Fourth generation denotes query languages and application generators.

D - Incorrect: Third generation includes general high-level languages such as C++ and Java.

Primary note proof

Study-note section CP03

> Machine instructions are conventionally first generation; symbolic assembly is second; general high-level languages such as C++ and Java are third. Database query and application generators such as SQL are conventionally fourth generation. Fifth generation is a loose historical category associated with declarative logic, constraints and AI/expert systems. Generations do not determine whether an implementation uses an interpreter.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

> Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Question sources

MOE Computer Science Exit Exam · 2016 · Question 17

Question 2 | Q00027

Which one of the following is correct ordering in language processing system?

A. Source program ->Preprocessor -> Assembler -> Compiler Linker/Loader -> target machine code

B. Source program -> Preprocessor ->Linker/Loader -> Compiler /Assembler ->target machine code

C. Source program Compiler -> Assembler -> Preprocessor > Linker/Loader -> target machine code

D. Source program _> Preprocessor -> Compiler -> Assembler -> Linker/Loader -> target machine code

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: Preprocessor → compiler → assembler → linker/loader.

CP04 explicitly places preprocessing before compilation proper and compilation before assembly. The assembler produces object files, which are linked into an executable and then loaded. D preserves that order, although it compresses linking/loading into one label and describes the final result loosely as target machine code.

Why each choice is right or wrong

A - Incorrect: Places the assembler before the compiler, reversing their stated order in CP04.

B - Incorrect: Attempts linking/loading before compilation and assembly have produced the required object files.

C - Incorrect: Places preprocessing after compilation and assembly, contrary to the conventional toolchain order.

D - Correct: Preserves preprocessing, compilation, assembly, linking and loading in the required sequence.

Primary note proof

Study-note section CP04

> In the conventional C++ toolchain: source → preprocessing → compilation proper → assembly → object files → linking → executable → loading → execution. Preprocessing handles directives such as #include; compilation may emit assembly; the assembler emits machine instructions in an object file. The linker resolves external symbols and combines object files and libraries. The loader places an executable and required components in memory and starts it.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 4 | lines 21-30

> The compilation process of C++ program consists of the following steps. • edit source code → compile → link → execute (Editor) (Compiler) (Linker) (Loader) I. Program source code is entered into a file using a text editor. II. After the code has been entered, a compiler program is started that translates the source into an object code file. III. The object code file is linked with other object code files that come with the compiler and an executable file (or program) is created. IV. In order to execute the program, a program called the loader copies the executable file into the memory of the computer and sends an execute command to the CPU.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 27

Question 3 | Q00042

What does this C++ fragment print? Assume cout is available.

```
int p,q,n;
p=q=n=8;
p+=n;
p/=q;
cout<<p;
```

A. 16

B. 2

C. 24

D. 1

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: B. 2

All three integers initially hold 8. The compound addition changes p to 16 without changing q or n. The compound division then divides the updated p by q, giving 16/8=2. Only the final value is inserted into cout. Both compound operators are single tokens and follow ordinary integer update rules.

Why each choice is right or wrong

A - Incorrect: 16 is the intermediate value before division.

B - Correct: After addition and division, p is 2.

C - Incorrect: 24 would require another addition that the fragment does not perform.

D - Incorrect: 1 would result from dividing the original 8 by 8 and ignoring the addition.

Primary note proof

Study-note section CP08

> x+=y, x-=y, x*=y, x/=y and x%=y update x using the operation and conversion to its type. For simple variables, trace them as x=x op y. Starting p=q=n=8, p+=n gives 16 and p/=q gives 2. Starting x=10, x+=5 then x/=2 gives 7. x=x+x doubles; it does not generally add one.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 42

Question 4 | Q00049

Look at the following fragment code :
int a = 6;
int b = 5;
while(a>b)
{
Cout<<"computer science program"<<endl;
}
Which algorithm property is not satisfied by the above C++ code algorithm?

A. Sequential

B. Feasibility

C. Finiteness

D. Definiteness

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer C: Finiteness

With a=6 and b=5, the while condition remains true because neither variable changes. Thus the intended algorithm never terminates and violates finiteness, explicitly required on PDF p.2. The steps are definite and executable; sequentiality is not the termination property. Interpret Cout as the intended cout. Module support: PDF p. 2, PDF p. 11.

Why each choice is right or wrong

A - Incorrect: "Sequential" does not match the rule or result derived here. With a=6 and b=5, the while condition remains true because neither variable changes. Thus the intended algorithm never terminates and violates finiteness, explicitly required on PDF p.2. The steps are definite and executable; sequentiality is not the termination property. Interpret Cout as the intended cout.

B - Incorrect: "Feasibility" does not match the rule or result derived here. With a=6 and b=5, the while condition remains true because neither variable changes. Thus the intended algorithm never terminates and violates finiteness, explicitly required on PDF p.2. The steps are definite and executable; sequentiality is not the termination property. Interpret Cout as the intended cout.

C - Correct: "Finiteness" matches the derived result. With a=6 and b=5, the while condition remains true because neither variable changes. Thus the intended algorithm never terminates and violates finiteness, explicitly required on PDF p.2. The steps are definite and executable; sequentiality is not the termination property. Interpret Cout as the intended cout.

D - Incorrect: "Definiteness" does not match the rule or result derived here. With a=6 and b=5, the while condition remains true because neither variable changes. Thus the intended algorithm never terminates and violates finiteness, explicitly required on PDF p.2. The steps are definite and executable; sequentiality is not the termination property. Interpret Cout as the intended cout.

Primary note proof

Study-note section CP02

> An algorithm gives precise, executable steps and terminates for every legal input.

Study-note section CP16

> A while(a>b) with fixed a=6,b=5 never changes its true condition.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 2 | lines 19-26

> Algorithm is an ordered set of unambiguous, executable steps, defining a terminating process. Important Properties of Algorithms - Correct: always returns the desired output for all legal instances of the problem. - Unambiguous: Each step must be clearly defined, having only one interpretation. - Precise: simple but efficient - Efficient: can be measured in terms of Time and Space - Finiteness: the execution of the algorithm must terminate in a finite amount of time for each possible set of input data/must have an end.

Computer Programming module | PDF page 11 | lines 5-13

> Iteration/Looping Statement: Iteration is the repetition of a statement or block of statements in a program. - Loops have as objective to repeat a statement a certain number of times or while a condition is fulfilled. - C++ has three iteration statements: the while statement, the do. While statement, and the for statement. The while loop: Its format is: while (expression) {statement;} First expression (called the loop condition) is evaluated. If the outcome is nonzero then statement (called the loop body) is executed and the whole process is repeated. Otherwise, the loop is terminated.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 49

Question 5 | Q00051

In the conventional C++ build process, which program resolves external symbols and combines object files and libraries into an executable before it is loaded into memory?

A. Editor

B. Debugger

C. Linker

D. Loader

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: C. Linker

Combining object files and resolving their external references is the linker's role in the conventional toolchain. This occurs after object-file generation and before the executable is loaded for execution. Loading has a separate responsibility: placing the executable and required components in memory and starting it. The question asks about the build operation, so an editor, debugger or loader does not satisfy it.

Why each choice is right or wrong

A - Incorrect: An editor changes source text.

B - Incorrect: A debugger investigates program execution and defects.

C - Correct: The linker combines objects/libraries and resolves external references.

D - Incorrect: The loader places an already linked executable into memory.

Primary note proof

Study-note section CP04

> The linker resolves external symbols and combines object files and libraries. The loader places an executable and required components in memory and starts it.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 4 | lines 21-30

> The compilation process of C++ program consists of the following steps. • edit source code → compile → link → execute (Editor) (Compiler) (Linker) (Loader) I. Program source code is entered into a file using a text editor. II. After the code has been entered, a compiler program is started that translates the source into an object code file. III. The object code file is linked with other object code files that come with the compiler and an executable file (or program) is created. IV. In order to execute the program, a program called the loader copies the executable file into the memory of the computer and sends an execute command to the CPU.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 51

Question 6 | Q00052

Machine language is grouped under_____ generation of programming language.

A. 2nd generation language

B. 3rd generation language

C. 4th generation language

D. 1st generation language

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: Machine language is first generation.

CP03 calls machine instructions first generation and symbolic assembly second generation. The question asks specifically about machine language, so the first-generation option follows directly. Third and fourth generations refer to higher-level languages and query/application generators, respectively; they do not classify native machine instructions.

Why each choice is right or wrong

A - Incorrect: Second generation identifies symbolic assembly, whose instructions still require assembly into machine code.

B - Incorrect: Third generation identifies general high-level languages such as C++ and Java.

C - Incorrect: Fourth generation identifies query languages and application generators, rather than machine instructions.

D - Correct: First generation is the note's explicit conventional classification for machine instructions.

Primary note proof

Study-note section CP03

> Machine instructions are conventionally first generation; symbolic assembly is second; general high-level languages such as C++ and Java are third. Database query and application generators such as SQL are conventionally fourth generation. Fifth generation is a loose historical category associated with declarative logic, constraints and AI/expert systems. Generations do not determine whether an implementation uses an interpreter.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

> Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Question sources

MOE Computer Science Exit Exam · 2016 · Question 52

Question 7 | Q00071

What words are printed, in order, when execution enters this C++ fragment at label a and ends after its final statement? Ignore padding spaces.
a: cout<<"Graduating ";
goto b;
c: cout<<" Year ";
goto a;
b: cout<<"fourth ";

A. GraduatingYear Fourth

B. Graduating fourth

C. YearFourth Graduating

D. GraduatingFourth Year

Hint

Fix the input case and cost model, then count dominant work or simultaneously live storage.

Answer B: B. Graduating fourth

Execution starts at a and prints Graduating. The unconditional goto b then transfers directly to the final output statement. That jump skips label c and its Year output, as well as the jump back to a. The statement at b prints lowercase fourth and the fragment ends. Ignoring padding spaces as specified, the two printed words are therefore Graduating followed by fourth.

Why each choice is right or wrong

A - Incorrect: Year is skipped, and Fourth changes the printed case.

B - Correct: The jump reaches b, giving Graduating followed by lowercase fourth.

C - Incorrect: Execution does not start at c, so Year cannot be the first word.

D - Incorrect: Year is never reached after b; the fragment ends there.

Primary note proof

Study-note section CP15

> A goto transfers to a label within its function; follow the label reached rather than reading lines in source order. In the sequence a: print Graduating; goto b; c: print Year; goto a; b: print fourth;, entry at a prints Graduating fourth and reaches the end. Year is not printed. Structured alternatives usually make such logic easier to reason about.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 14 | lines 15-20

> C++ goto Statement Syntax: goto label1; statements Label1: statements

Computer Programming module | PDF page 6 | lines 4-15

6. Communication through console cout (the standard output stream) o used to display information for the user Syntax: cout<<object; • Object can be variable or literals Example: int x=5; cout<<x cout<< “Hello there”; cin(the standard input stream): o used to accept information from the user. Syntax: cin>>variable name; Eg. cin>>x;

Question sources

MOE Computer Science Exit Exam · 2016 · Question 71

Question 8 | Q00074

Given the C++ declarations below, which statement correctly prints the car_model member through c1? Assume cout is available.
struct car { char car_model[25]; float price; };
car c{};
car* c1=&c;

A. Cout<<c1<<price;

B. Cin>>c1.car_model;

C. C1.price = 150000;

D. cout<<c1->car_model;

Hint

Apply the naming, type and syntax rules before deciding whether an option can be used as stated.

Answer D: D. cout<<c1->car_model;

The variable c1 is a pointer to a live car object. Accessing a member through that pointer uses ->, followed by the exact member name car_model. The corrected D therefore inserts the character array into cout; zero initialization supplies a valid empty string here. The other expressions fail to select the requested member, use dot directly on a pointer, or refer to an undeclared differently capitalized name.

Why each choice is right or wrong

A - Incorrect: This does not select price from c1 and uses the undeclared name Cout.

B - Incorrect: A pointer needs -> rather than dot; Cin also has the wrong case.

C - Incorrect: C1 differs from c1, and dot is not pointer-member access.

D - Correct: cout<<c1->car_model; selects the declared member through the live pointer.

Primary note proof

Study-note section CP27

A struct groups named members, possibly of different types. struct House{double price; int rooms;}; defines a type; House h{150000,3}; creates an object. h.rooms uses a dot. House* p=&h; p->rooms is equivalent to (*p).rooms. The pointer must refer to a live object before member access. A type definition alone does not create an ordinary instance.

Study-note section CP-S01

> For an aggregate with no default member initializers, empty braces initialize its members recursively. Numeric scalar members become zero, and the elements of a character-array member become zero characters. A zero character at the beginning of such an array makes it an empty, terminated C string. This differs from leaving an ordinary automatic scalar or array uninitialized.

Study-note section CP24

> cout<<a prints a terminated char sequence, not its numeric address.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 38 | lines 10-16

> Pointer to structure Pointers to structures may be declared as follow: struct staructuretage *pinternname; eg Employe *emp; When using a pointer to a struct, member access can be achieved with the `.' operator, but can not look graceful in expression Example: (*emp).id Equivalently, the ->'` operator can be used; for example: emp->id

Computer Programming module | PDF page 35 | lines 13-17

> Accessing members of a structure variable A structure member can be accessed using dot operator'.' notation syntax: structvariable.member; eg. st.id=4;

Question sources

MOE Computer Science Exit Exam · 2016 · Question 74

Question 9 | Q00085

Software Debugging is known as _____

A. identifying the task to be computerized

B. creating program code

C. creating the algorithm

D. finding and correcting errors in the program code

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: Finding and correcting errors in program code.

CP32 defines a bug as a defect and debugging as locating and correcting it. Defining the task, designing an algorithm and creating code are other development activities. They may expose defects, but none of them states the specific error-finding and correction purpose of debugging.

Why each choice is right or wrong

A - Incorrect: Identifying the task defines requirements; it does not locate and repair defects in code.

B - Incorrect: Creating code is implementation, whereas debugging investigates and corrects defects in that code.

C - Incorrect: Creating an algorithm is design; debugging concerns finding and correcting defects.

D - Correct: Finding and correcting program errors directly matches the definition of debugging in CP32.

Primary note proof

Study-note section CP32

> A bug is a defect; debugging locates and corrects it.

Study-note section CP01

> Programming includes writing, testing, debugging and maintaining instructions. A program can execute without solving its intended problem; compare its actual result with the required result.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 2 | lines 2-18

> 1. Introduction to programming Computer programming is the process of writing, testing, debugging/troubleshooting, and maintaining the source code of computer programs. Programs are the instructions that tells the computer what to do .so, writing computer programs means writing instructions, that will make the computer follow and run a program based on those instructions. We are going to write computer programs to solve problems. Problem Solving Problem: gap between the existing and the desired or proposed situation where problem solving will try to fill this gap. Problem solving: The process of transforming the description of a problem into the solution by using our knowledge of the problem-solving strategies, techniques and tools. Problem solving techniques • Analyze the problem: Carefully understand the problem • Outline the problem requirements • Does program require user interaction? • Does program manipulate data? • What is the output? • Design steps (algorithm) to solve the problem

Question sources

MOE Computer Science Exit Exam · 2016 · Question 85

MOE Computer Science Exit Exam · 2016 · Question 97

Question 10 | Q00099

How many times is the phrase "In the loop" printed when the following code is executed?
int i, j=8;
for(i=0; i<j; i++, j--)
3. {
if(i%2!=0) continue;
cout << "In the loop " << endl;
6. }

A. 20

B. 10

C. 3

D. 2

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer D: 2

Ignoring source line numbers, the loop body is reached at (i,j)=(0,8),(1,7),(2,6),(3,5). Even i values 0 and 2 print; odd values 1 and 3 continue to the update. The next pair (4,4) fails i<j. Therefore exactly two lines print, not 20, 10 or 3. Module support: PDF p. 12, PDF p. 13, PDF p. 7.

Why each choice is right or wrong

A - Incorrect: "20" does not match the rule or result derived here. Ignoring source line numbers, the loop body is reached at (i,j)=(0,8),(1,7),(2,6),(3,5). Even i values 0 and 2 print; odd values 1 and 3 continue to the update. The next pair (4,4) fails i<j. Therefore exactly two lines print, not 20, 10 or 3.

B - Incorrect: "10" does not match the rule or result derived here. Ignoring source line numbers, the loop body is reached at (i,j)=(0,8),(1,7),(2,6),(3,5). Even i values 0 and 2 print; odd values 1 and 3 continue to the update. The next pair (4,4) fails i<j. Therefore exactly two lines print, not 20, 10 or 3.

C - Incorrect: "3" does not match the rule or result derived here. Ignoring source line numbers, the loop body is reached at (i,j)=(0,8),(1,7),(2,6),(3,5). Even i values 0 and 2 print; odd values 1 and 3 continue to the update. The next pair (4,4) fails i<j. Therefore exactly two lines print, not 20, 10 or 3.

D - Correct: "2" matches the derived result. Ignoring source line numbers, the loop body is reached at (i,j)=(0,8),(1,7),(2,6),(3,5). Even i values 0 and 2 print; odd values 1 and 3 continue to the update. The next pair (4,4) fails i<j. Therefore exactly two lines print, not 20, 10 or 3.

Primary note proof

Study-note section CP16

Make a table with condition inputs, body output and updated values. For i=0,j=8; i<j; i++,j--, body entries are (0,8),(1,7),(2,6),(3,5). If odd i triggers continue, printing occurs twice. With outer i=1; i<5; i+=5, the outer loop runs once; an inner j=10; j>i; --j prints nine times.

Study-note section CP15

continue skips the rest of the current loop iteration: a for loop then runs its update; a while loop goes directly to its condition.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 12 | lines 19-28

The for loop. Its format is: for (initialization; condition; increase) statement; 1, initialization is executed. Generally, it is a initial value setting for a counter variable. This is executed only once 2, condition is checked, if it is true the loop continues, otherwise the loop finishes and statement is skipped. 3, statement is executed. As usual, it can be either a single instruction or a block of instructions enclosed within curly brackets { }. 4, finally, whatever is specified in the increase field is executed and the loop gets back to step 2.

Computer Programming module | PDF page 13 | lines 30-35

Using continue and break Statements - The break statement skips the rest of the statements in the loop‟s block, jumping immediately to the next statement outside of the loop. - The continue statement is similar. It also skips the rest of the statements in the loop‟s block, but instead of terminating the loop, it transfers execution to the next iteration of the loop. It continues the loop after skipping the remaining statements in its current iteration.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Addis Ababa University mock exam · Question 1

Question 11 | Q00100

What does this C++ fragment print? Assume the standard streams are available; line breaks shown in the options are part of the output.

```
int main(){
int a=5,x=1,y=0,z=4;
a=(x && y) || ++z;
std::cout<<z<<std::endl<<a;
return 0;
}
```

A. 5
1

B. 5
2

C. 4
1

D. 4
2

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer A: A. 5, then 1 on the next line.

The conjunction x&&y is false because y is zero. The OR therefore evaluates ++z, which changes z from 4 to 5 and supplies a true value. Assigning that logical result stores 1 in a. The output inserts z, then a line ending, then a, so it consists of 5 followed by 1 on the next line.

Why each choice is right or wrong

A - Correct: The false left operand triggers ++z, and the logical result is 1.

B - Incorrect: A logical OR produces 0 or 1, so a cannot print as 2.

C - Incorrect: The OR evaluates ++z, so z does not remain 4.

D - Incorrect: This both omits the increment and gives the logical result as 2.

Primary note proof

Study-note section CP10

> && evaluates its right operand only if the left is true; || only if the left is false. Thus false && dangerousCall() skips the call. Bitwise & operates on bits and does not short circuit. With x=1,y=0,z=4, (x&y)||++z evaluates ++z, leaving z=5 and the Boolean result 1. The precedence ladder is grouping/postfix, unary, multiplicative, additive, shifts, relations, equality, bitwise &, ^, |, &&, ||, conditional, assignment, comma.

Study-note section CP28

std::endl inserts a newline and flushes.

Study-note section CP34

With valid C++ syntax for the stated assignments, the trace yields z=5 then a=1.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Computer Programming module | PDF page 6 | lines 4-15

6. Communication through console cout (the standard output stream) o used to display information for the user Syntax: cout<<object; • Object can be variable or literals Example: int x=5; cout<<x cout<< “Hello there”; cin(the standard input stream): o used to accept information from the user. Syntax: cin>>variable name; Eg. cin>>x;

Question sources

Addis Ababa University mock exam · Question 2

Question 12 | Q00101

is a function that invokes itself?

A. Recursive function

B. Inline function

C. Built in function

D. User defined function

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: Recursive function

A recursive function calls itself, exactly the definition on p.18. Inline concerns implementation and built-in/user-defined concern origin; none of those categories alone entails self-invocation. Module support: PDF p. 18.

Why each choice is right or wrong

A - Correct: "Recursive function" matches the derived result. A recursive function calls itself, exactly the definition on p.18. Inline concerns implementation and built-in/user-defined concern origin; none of those categories alone entails self-invocation.

B - Incorrect: "Inline function" does not match the rule or result derived here. A recursive function calls itself, exactly the definition on p.18. Inline concerns implementation and built-in/user-defined concern origin; none of those categories alone entails self-invocation.

C - Incorrect: "Built in function" does not match the rule or result derived here. A recursive function calls itself, exactly the definition on p.18. Inline concerns implementation and built-in/user-defined concern origin; none of those categories alone entails self-invocation.

D - Incorrect: "User defined function" does not match the rule or result derived here. A recursive function calls itself, exactly the definition on p.18. Inline concerns implementation and built-in/user-defined concern origin; none of those categories alone entails self-invocation.

Primary note proof

Study-note section CP21

> A recursive function calls itself directly or indirectly.

Study-note section CP19

> inline does not compel the compiler to copy a function body at a call. Small frequently used functions may be good optimization candidates, but actual expansion is the optimizer’s decision. Inline also has language linkage/definition rules, so it is not simply a speed command.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 18 | lines 23-28

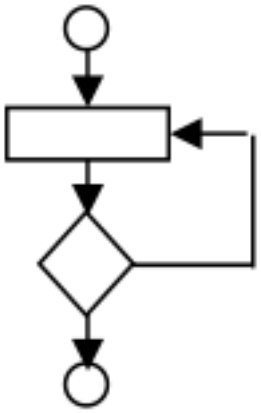
> Recursive Functions - Functions that call themselves - A function that is defined in terms of itself is called recursive - There must be a base case e.g. 0! = 1 - Slowly converges towards base case - For example, Recursive functions N! = N * (N-1)!

Question sources

Addis Ababa University mock exam · Question 3

Question 13 | Q00102

Which control flow statement is represented by the flowchart shown in the figure below?



A. while loop

B. do...while loop

C. if...else statement

D. for loop

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: do...while loop

The retained flowchart enters the rectangular body before reaching the decision diamond. Its backward arrow returns from the decision to the body. This is a post-test repetition with at least one execution, matching do-while (p.12). while and ordinary for test before the body; if-else does not loop back. Module support: PDF p. 12, PDF p. 11.

Why each choice is right or wrong

A - Incorrect: "while loop" does not match the rule or result derived here. The retained flowchart enters the rectangular body before reaching the decision diamond. Its backward arrow returns from the decision to the body. This is a post-test repetition with at least one execution, matching do-while (p.12). while and ordinary for test before the body; if-else does not loop back.

B - Correct: "do...while loop" matches the derived result. The retained flowchart enters the rectangular body before reaching the decision diamond. Its backward arrow returns from the decision to the body. This is a post-test repetition with at least one execution, matching do-while (p.12). while and ordinary for test before the body; if-else does not loop back.

C - Incorrect: "if...else statement" does not match the rule or result derived here. The retained flowchart enters the rectangular body before reaching the decision diamond. Its backward arrow returns from the decision to the body. This is a post-test repetition with at least one execution, matching do-while (p.12). while and ordinary for test before the body; if-else does not loop back.

D - Incorrect: "for loop" does not match the rule or result derived here. The retained flowchart enters the rectangular body before reaching the decision diamond. Its backward arrow returns from the decision to the body. This is a post-test repetition with at least one execution, matching do-while (p.12). while and ordinary for test before the body; if-else does not loop back.

Primary note proof

Study-note section CP14

> while(condition) checks before each body execution and may run zero times. do {body;} while(condition); runs the body once before checking; it repeats while the condition remains true. Use it when a menu or input attempt must occur at least once.

Study-note section CP34

> When entry reaches the body rectangle before the decision diamond and a decision branch returns to the body, the body executes before the condition is tested. This is a post-test do-while loop.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 12 | lines 2-7

> The do-while loop The condition in the do-while is evaluated after the execution of statement instead of before, granting at least one execution of statement even if condition is never fulfilled. do {statement; }while (condition); First statement is executed and then condition is evaluated. If the outcome of the latter is nonzero then the whole process is repeated. Otherwise, the loop is terminated.

Computer Programming module | PDF page 11 | lines 5-13

> Iteration/Looping Statement: Iteration is the repetition of a statement or block of statements in a program. - Loops have as objective to repeat a statement a certain number of times or while a condition is fulfilled. - C++ has three iteration statements: the while statement, the do. While statement, and the for statement. The while loop: Its format is: while (expression) {statement;} First expression (called the loop condition) is evaluated. If the outcome is nonzero then statement (called the loop body) is executed and the whole process is repeated. Otherwise, the loop is terminated.

Question sources

Addis Ababa University mock exam · Question 4

Question 14 | Q00103

What is the output of the following code snippet?
double ph=15.3;
if(ph<@ || ph>15)
3. {
cout<< "Invalid PH value. ";
5. }
if(ph<=6.5)
cout<< "Acidic";
else if(ph>6.5 && ph<7.5)
cout<< "Neutral";
else
cout<< "Alkaline";

A. Invalid PH Value

B. Neutral

C. Alkaline

D. Invalid PH Value. Alkaline

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer D: Invalid PH Value. Alkaline

The negative-value test is ph<0. Since 15.3>15, the first independent if prints Invalid PH value. The following separate if/else-if chain still executes: ph<=6.5 is false and the middle range test is false, so it prints Alkaline. Thus both messages appear. A and C omit one message, and B takes a false middle branch. Module support: PDF p. 9, PDF p. 7.

Why each choice is right or wrong

A - Incorrect: "Invalid PH Value" does not match the rule or result derived here. The negative-value test is ph<0. Since 15.3>15, the first independent if prints Invalid PH value. The following separate if/else-if chain still executes: ph<=6.5 is false and the middle range test is false, so it prints Alkaline. Thus both messages appear. A and C omit one message, and B takes a false middle branch.

B - Incorrect: "Neutral" does not match the rule or result derived here. The negative-value test is ph<0. Since 15.3>15, the first independent if prints Invalid PH value. The following separate if/else-if chain still executes: ph<=6.5 is false and the middle range test is false, so it prints Alkaline. Thus both messages appear. A and C omit one message, and B takes a false middle branch.

C - Incorrect: "Alkaline" does not match the rule or result derived here. The negative-value test is ph<0. Since 15.3>15, the first independent if prints Invalid PH value. The following separate if/else-if chain still executes: ph<=6.5 is false and the middle range test is false, so it prints

Alkaline. Thus both messages appear. A and C omit one message, and B takes a false middle branch.

D - Correct: "Invalid PH Value. Alkaline" matches the derived result. The negative-value test is ph<0. Since 15.3>15, the first independent if prints Invalid PH value. The following separate if/else-if chain still executes: ph<=6.5 is false and the middle range test is false, so it prints Alkaline. Thus both messages appear. A and C omit one message, and B takes a false middle branch.

Primary note proof

Study-note section CP12

> A standalone if may execute independently of the following if. An else binds to its nearest unmatched if; an else-if ladder executes the first true branch only. In nested if(c1){if(c2) S1; else S2;}, S2 requires c1 true and c2 false. Braces determine grouping.

Study-note section CP12

> With ph=15.3, an invalid-range if can print its warning, and a separate classification ladder can also print Alkaline.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 9 | lines 9-21

> Conditional Statements A. The if Statement Syntax: if (expression) statement; First expression is evaluated. If the outcome is nonzero (true) then statement is executed. Otherwise, nothing happens. if-else statement Syntax: if (expression) statement1; else statement2; First expression is evaluated. If the outcome is nonzero (true) then statement1 is executed. Otherwise, statement2 is executed.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Addis Ababa University mock exam · Question 5

Question 15 | Q00104

Which three numeric values are displayed, in order, by this C++ program? Ignore the prompt text and separator spaces; assume cout and endl are available.

```
int func(int a,int b){
a*=5;
b+=5;
return a+b;
}
int main(){
int x=9,y=3;
cout<<"Result = "<<func(x,y)<<", "<<x<<", "<<y<<endl;
return 0;
}
```

A. 12,9,3

B. 50,9,3

C. 53,9,3

D. 53,45,8

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: C. 53,9,3

The value parameters are separate objects initialized with 9 and 3. Inside func, a becomes 45 and b becomes 8, so the returned sum is 53. Neither update changes the caller's x or y, which remain 9 and 3. The requested numeric sequence is therefore 53,9,3. Standard int main() restores the valid enclosing program already required by CP34 without changing the parameter-passing problem.

Why each choice is right or wrong

A - Incorrect: 12 adds the original parameters instead of their updated local values.

B - Incorrect: The updated local values sum to 53, not 50.

C - Correct: The function returns 53 while the caller retains 9 and 3.

D - Incorrect: Value parameters do not change the caller's x and y to 45 and 8.

Primary note proof

Study-note section CP18

> A C++ value parameter is a separate object initialized from the argument. In func(int a,int b){a*=5;b+=5;return a+b;}, calling func(9,3) returns 53 and leaves the caller values 9 and 3 unchanged.

Study-note section CP34

Use <0 for a less-than-zero comparison and the standard int main() declaration for a C++ entry point.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 16 | lines 13-20

Passing by Value In call by value, the actual value that is passed as argument is not changed after performing some operation on it. it creates a copy of that variable into the stack section in memory. When the value is changed, it changes the value of that copy, the actual value remains the same Example: Passing by Reference: It allows a function to modify a variable without having to create a copy of it. We have to declare reference variables. The memory location of the passed variable and parameter is the same and therefore, any change to the parameter reflects in the variable as well.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, | |) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 | | 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Addis Ababa University mock exam · Question 6

Question 16 | Q00105

What do arrays do?

A. Hold address of a single value

B. Hold one value

C. Hold values of the same type under a single name

D. Hold addresses of values under a single name

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: Hold values of the same type under a single name

An array groups a fixed number of elements of the same type (p.19), with each element selected by an index. It is not a single-value storage definition or intrinsically a collection of addresses; pointer-valued arrays are only a particular element type. Module support: PDF p. 19.

Why each choice is right or wrong

A - Incorrect: "Hold address of a single value" does not match the rule or result derived here. An array groups a fixed number of elements of the same type (p.19), with each element selected by an index. It is not a single-value storage definition or intrinsically a collection of addresses; pointer-valued arrays are only a particular element type.

B - Incorrect: "Hold one value" does not match the rule or result derived here. An array groups a fixed number of elements of the same type (p.19), with each element selected by an index. It is not a single-value storage definition or intrinsically a collection of addresses; pointer-valued arrays are only a particular element type.

C - Correct: "Hold values of the same type under a single name" matches the derived result. An array groups a fixed number of elements of the same type (p.19), with each element selected by an index. It is not a single-value storage definition or intrinsically a collection of addresses; pointer-valued arrays are only a particular element type.

D - Incorrect: "Hold addresses of values under a single name" does not match the rule or result derived here. An array groups a fixed number of elements of the same type (p.19), with each element selected by an index. It is not a single-value storage definition or intrinsically a collection of addresses; pointer-valued arrays are only a particular element type.

Primary note proof

Study-note section CP22

> A built-in C++ array has a fixed number of same-type elements in contiguous storage: int a[4]={10,20,30,40};. Valid indices are 0 through 3. a[4] is outside the array; accessing it has undefined behavior. Java checks bounds and throws ArrayIndexOutOfBoundsException. Do not predict a C++ out-of-range value from nearby memory.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 19 | lines 4-29

> Arrays - an array is a collection of fixed-size number of elements of the same type stored sequentially in memory. - An individual element of an array is identified by its own unique index (or subscript). - The number of elements must be an integer. - The size of the array is referred to as its dimension - Array in C++ is Zero indexed. Declaring of one-dimensional Array Syntax: elementstype arrayname [size]; → You can use constants such as macro definitions or const ints as the sizes of arrays. Example: int num[5]; string fruit[3]; #define numstudents 100 or const int numstudents=100; double Score[numstudents] ; double Score[numstudents] ; Initialization of an Array - Arrays can be initialized at declaration time in two different ways: Syntax: elementstype arrayname[size] = {list each value separated by comma}; E.g int num[5]={1,2,3,4,5}; elementstype arrayname[ ] = { list with any number of values }; E.g int num[]={1,2,3,4,5}; string

fruit[]={“Apple”,“orange”, “Mango”}; Accessing Array Elements - Each element could be accessed using an index into the array - When accessing an array, the index given must be a positive integer from 0 to n-1, where n is the dimension of the array.

Question sources

Addis Ababa University mock exam · Question 7

Question 17 | Q00174

In recursion, the condition for which the function will stop calling itself is .

A. Best case

B. Worst case

C. Base case

D. There is no such condition

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: Base case

The recursion section explicitly requires a base case and convergence toward it. That case stops further self-calls. Best and worst case describe other analyses, while denying a stopping condition conflicts with the module requirement. Module support: PDF p. 18.

Why each choice is right or wrong

A - Incorrect: "Best case" does not match the rule or result derived here. The recursion section explicitly requires a base case and convergence toward it. That case stops further self-calls. Best and worst case describe other analyses, while denying a stopping condition conflicts with the module requirement.

B - Incorrect: "Worst case" does not match the rule or result derived here. The recursion section explicitly requires a base case and convergence toward it. That case stops further self-calls. Best and worst case describe other analyses, while denying a stopping condition conflicts with the module requirement.

C - Correct: "Base case" matches the derived result. The recursion section explicitly requires a base case and convergence toward it. That case stops further self-calls. Best and worst case describe other analyses, while denying a stopping condition conflicts with the module requirement.

D - Incorrect: "There is no such condition" does not match the rule or result derived here. The recursion section explicitly requires a base case and convergence toward it. That case stops further self-calls. Best and worst case describe other analyses, while denying a stopping condition conflicts with the module requirement.

Primary note proof

Study-note section CP21

> A recursive function calls itself directly or indirectly. It needs a base case and progress toward it for all supported inputs. factorial(0)=1 and factorial(n)=n*factorial(n-1) for positive n. factorial(3) expands to 3*2*1*1=6. Negative inputs require rejection or a separate definition; they do not reach zero under n-1.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 18 | lines 23-28

> Recursive Functions - Functions that call themselves - A function that is defined in terms of itself is called recursive - There must be a base case e.g. 0! = 1 - Slowly converges towards base case - For example, Recursive functions N! = N * (N-1)!

Question sources

Addis Ababa University mock exam · Question 76

Question 18 | Q00209

One of the following statements is true

A. The typedef command is used to create a new type.

B. If the function is declared inline the compiler must copy the code in the calling function

C. All of the choices are correct

D. It is recommended to use inline functions to simple and frequently used functions

E. None of the choices are correct

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: Small, frequently used functions are reasonable inline optimization candidates.

CP19 recommends small, frequently used functions as possible optimization candidates, while stressing that inline never forces expansion at the call site. CP06 says typedef creates an alias rather than a distinct new type. Thus D is defensible as a recommendation; the mandatory-expansion and distinct-type claims cannot make all choices correct.

Why each choice is right or wrong

A - Incorrect: A typedef supplies an alternative name for an existing type, not a distinct new type.

B - Incorrect: Declaring inline does not require the compiler to expand the body at each call.

C - Incorrect: The distinct-type and mandatory-expansion claims are false, so all choices cannot be correct.

D - Correct: Simple, frequently called functions can be reasonable candidates, although expansion remains an optimizer decision.

E - Incorrect: The recommendation in D is supported, so it is false that none is correct.

Primary note proof

Study-note section CP19

> inline does not compel the compiler to copy a function body at a call. Small frequently used functions may be good optimization candidates, but actual expansion is the optimizer’s decision. Inline also has language linkage/definition rules, so it is not simply a speed command.

Study-note section CP06

> typedef creates an alias, not a distinct new type.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 15 | lines 29-46

> • User-defined - Function that been created by the user. - These functions need to be declared and defined by the user. Declaration of function Function defines using the following syntax: Function_type function_name (Parameter List) { ...function body } Example: void Addition(int num1,int num2) #include<iostream> using namespace std; void Addition(int num1,int num2) { int sum=num1+num2; cout<<"The sum is:"<<sum; } int main() { int x=10,y=20; Addition(x,y);

Question sources

Mock exam 4 · Question 11

Question 19 | Q00210

Given the following lines of code, how many times will "hello" be displayed on the screen?

```
int i = 1;
while(i < 5){
int j = 10;
while (j > i){
j--;
cout << "hello" << endl;
}
i = i + 5;
}
```

A. 10

B. 9

C. 5

D. 25

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: 9

The outer loop starts with i=1. The inner loop decrements j from 10 through 1 and prints after each decrement: nine prints. Then i becomes 6, failing i<5, so no second outer iteration occurs. Ten counts the initial j without a successful iteration; five and 25 ignore the actual bounds. Module support: PDF p. 11, PDF p. 7.

Why each choice is right or wrong

A - Incorrect: "10" does not match the rule or result derived here. The outer loop starts with i=1. The inner loop decrements j from 10 through 1 and prints after each decrement: nine prints. Then i becomes 6, failing i<5, so no second outer iteration occurs. Ten counts the initial j without a successful iteration; five and 25 ignore the actual bounds.

B - Correct: "9" matches the derived result. The outer loop starts with i=1. The inner loop decrements j from 10 through 1 and prints after each decrement: nine prints. Then i becomes 6, failing i<5, so no second outer iteration occurs. Ten counts the initial j without a successful iteration; five and 25 ignore the actual bounds.

C - Incorrect: "5" does not match the rule or result derived here. The outer loop starts with i=1. The inner loop decrements j from 10 through 1 and prints after each decrement: nine prints. Then i becomes 6, failing i<5, so no second outer iteration occurs. Ten counts the initial j without a successful iteration; five and 25 ignore the actual bounds.

D - Incorrect: "25" does not match the rule or result derived here. The outer loop starts with i=1. The inner loop decrements j from 10 through 1 and prints after each decrement: nine prints. Then i becomes 6, failing i<5, so no second outer iteration occurs. Ten counts the initial j without a successful iteration; five and 25 ignore the actual bounds.

Primary note proof

Study-note section CP14

> while(condition) checks before each body execution and may run zero times. do {body;} while(condition); runs the body once before checking; it repeats while the condition remains true. Use it when a menu or input attempt must occur at least once.

Study-note section CP16

> With outer i=1; i<5; i+=5, the outer loop runs once; an inner j=10; j>i; --j prints nine times.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 11 | lines 5-13

> Iteration/Looping Statement: Iteration is the repetition of a statement or block of statements in a program. - Loops have as objective to repeat a statement a certain number of times or while a condition is fulfilled. - C++ has three iteration statements: the while statement, the do. While

statement, and the for statement. The while loop: Its format is: while (expression) {statement;} First expression (called the loop condition) is evaluated. If the outcome is nonzero then statement (called the loop body) is executed and the whole process is repeated. Otherwise, the loop is terminated.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, | |) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Mock exam 4 · Question 12

Question 20 | Q00211

What is the output of the below given program?

```
int figure_me_out() {
static int x = 0;
if (x % 2 == 0)
x = x + 1;
else
x = x + 2;
return x;
}
int main() {
int x = 0;
for(int i = 0; i < 5; i++)
x = x + figure_me_out();
cout << x;
return 0;
}
```

A. 1

B. 19

C. 2

D. 20

E. 25

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer E: 25.

The function's static x is initialized once, then persists across calls. Its five returns are 1, 3, 5, 7 and 9: after the first update it stays odd and increases by two. Main has a separate x and accumulates all five returns, giving 25.

Why each choice is right or wrong

A - Incorrect: One is only the first function return, not the sum accumulated by main.

B - Incorrect: The five return values sum to 25; a total of 19 omits or miscomputes updates.

C - Incorrect: The function does not reset or simply count calls; its persistent values are odd.

D - Incorrect: The five odd returns total 25, not the sum of five even values.

E - Correct: Adding the five persistent-state returns gives 1+3+5+7+9, which equals 25.

Primary note proof

Study-note section CP20

> An automatic local is created anew on entry. A static local is initialized once and retains its value across calls even though its name remains local. If static x=0 changes by +1 when even and +2 otherwise, five calls return 1,3,5,7,9, whose sum is 25. Namespace/static storage integers are zero initialized; an ordinary uninitialized local int has no usable assumed zero value.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 17 | lines 14-29

> Scope of variables Scope of variables defined as the extent of the program code within which the variable can be accessed or declared or worked with mainly two types of variable scopes: local variable: - Variables defined within a function or block are said to be local to those functions - Declared inside a block{} Global Variable - variable that can be accessed from any part of the program. - They are available throughout the life time of a program. - They are declared at the top of the program outside all of the functions or blocks. NB: - Usually when two variables with same name are defined then the compiler produces a compile time error. But if the variables are defined in different scopes, then the compiler allows it. - Whenever there is a local variable defined with same name as that of a global variable then the compiler will give precedence to the local variable

Computer Programming module | PDF page 11 | lines 5-13

> Iteration/Looping Statement: Iteration is the repetition of a statement or block of statements in a program. - Loops have as objective to repeat a statement a certain number of times or while a condition is fulfilled. - C++ has three iteration statements: the while statement, the do. While statement, and the for statement. The while loop: Its format is: while (expression) {statement;} First expression (called the loop condition) is evaluated. If the outcome is nonzero then

statement (called the loop body) is executed and the whole process is repeated. Otherwise, the loop is terminated.

Question sources

Mock exam 4 · Question 13

Question 21 | Q00212

What is the output of the following code fragment?

```
int f1(int n, int m){
if(n < m)
return 0;
else if(n == m)
return m + f1(n-1, m);
else
return n + f1(n-2, m-1);
}
int main(){
cout << f1(1, 4);
return 0;
}
```

A. 0

B. None of the choices are correct

C. 8

D. 4

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer A: 0

The initial call is f1(1,4). Its first condition 1<4 is true, so it returns 0 immediately. Neither recursive branch executes. Eight and four would require branches that are not entered; the none option is false because 0 is listed. Module support: PDF p. 18, PDF p. 9.

Why each choice is right or wrong

A - Correct: "0" matches the derived result. The initial call is f1(1,4). Its first condition 1<4 is true, so it returns 0 immediately. Neither recursive branch executes. Eight and four would require branches that are not entered; the none option is false because 0 is listed.

B - Incorrect: "None of the choices are correct" does not match the rule or result derived here. The initial call is f1(1,4). Its first condition 1<4 is true, so it returns 0 immediately. Neither recursive branch executes. Eight and four would require branches that are not entered; the none option is false because 0 is listed.

C - Incorrect: "8" does not match the rule or result derived here. The initial call is f1(1,4). Its first condition 1<4 is true, so it returns 0 immediately. Neither recursive branch executes. Eight and four would require branches that are not entered; the none option is false because 0 is listed.

D - Incorrect: "4" does not match the rule or result derived here. The initial call is f1(1,4). Its first condition 1<4 is true, so it returns 0 immediately. Neither recursive branch executes. Eight and four would require branches that are not entered; the none option is false because 0 is listed.

Primary note proof

Study-note section CP21

> Evaluate conditions before expanding a recursive call. For if(n<m) return 0;, f(1,4) returns immediately; unreachable recursive branches add nothing. Record each call's local parameters and pending operation, then unwind returned values. Recursion commonly uses smaller instances, but merely having a base-case branch does not prove it will be reached.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 18 | lines 23-28

> Recursive Functions - Functions that call themselves - A function that is defined in terms of itself is called recursive - There must be a base case e.g. 0! = 1 - Slowly converges towards base case - For example, Recursive functions N! = N * (N-1)!

Computer Programming module | PDF page 9 | lines 9-21

> Conditional Statements A. The if Statement Syntax: if (expression) statement; First expression is evaluated. If the outcome is nonzero (true) then statement is executed. Otherwise, nothing happens. if-else statement Syntax: if (expression) statement1; else statement2; First expression is evaluated. If the outcome is nonzero (true) then statement1 is executed. Otherwise, statement2 is executed.

Question sources

Mock exam 4 · Question 14

Question 22 | Q00270

Assume that p, x, y, and z are int variables with x = 1, y = 10, z= -3. Which of the following is true after the following statement is executed? p= (x++ * y--) + ++z;

A. p=22

B. p=17

C. p=8

D. p=16

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: p=8

Postfix x++ contributes old x=1 and y-- contributes old y=10; prefix ++z changes -3 to -2 before use. Thus p=1*10+(-2)=8. The new x=2 and y=9 are not the values multiplied. The other values mix old and updated operands. Module support: PDF p. 7.

Why each choice is right or wrong

A - Incorrect: "p=22" does not match the rule or result derived here. Postfix x++ contributes old x=1 and y-- contributes old y=10; prefix ++z changes -3 to -2 before use. Thus p=1*10+(-2)=8. The new x=2 and y=9 are not the values multiplied. The other values mix old and updated operands.

B - Incorrect: "p=17" does not match the rule or result derived here. Postfix x++ contributes old x=1 and y-- contributes old y=10; prefix ++z changes -3 to -2 before use. Thus p=1*10+(-2)=8. The new x=2 and y=9 are not the values multiplied. The other values mix old and updated operands.

C - Correct: "p=8" matches the derived result. Postfix x++ contributes old x=1 and y-- contributes old y=10; prefix ++z changes -3 to -2 before use. Thus p=1*10+(-2)=8. The new x=2 and y=9 are not the values multiplied. The other values mix old and updated operands.

D - Incorrect: "p=16" does not match the rule or result derived here. Postfix x++ contributes old x=1 and y-- contributes old y=10; prefix ++z changes -3 to -2 before use. Thus p=1*10+(-2)=8. The new x=2 and y=9 are not the values multiplied. The other values mix old and updated operands.

Primary note proof

Study-note section CP09

> Prefix ++x changes x and yields the new value; postfix x++ yields the previous value and also increments x. Separate the value used in the expression from the stored value after it. Starting a=8,b=5, a++ + --b uses 8+4, gives 12, and leaves a=9,b=4. Starting x=1,y=10,z=-3, (x++*y--)+ ++z gives 8 and leaves x=2,y=9,z=-2.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, | |) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 | | 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Computer Science model exam · Question 1

University of Gondar model exam · Question 1

Question 23 | Q00271

What will happen in the following C++ code snippet? int a = 100, b = 200; int *p = &a, *q = &b; p= q;

A. p now points to b

B. a is assigned to b

C. b is assigned to a

D. q now points to a

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: p now points to b

p initially stores &a and q stores &b. Assignment p=q copies the address stored in q into p, so both point to b. Neither pointee value is assigned because no dereference appears; q is not assigned at all. Module support: PDF p. 26, PDF p. 27.

Why each choice is right or wrong

A - Correct: "p now points to b" matches the derived result. p initially stores &a and q stores &b. Assignment p=q copies the address stored in q into p, so both point to b. Neither pointee value is assigned because no dereference appears; q is not assigned at all.

B - Incorrect: "a is assigned to b" does not match the rule or result derived here. p initially stores &a and q stores &b. Assignment p=q copies the address stored in q into p, so both point to b. Neither pointee value is assigned because no dereference appears; q is not assigned at all.

C - Incorrect: "b is assigned to a" does not match the rule or result derived here. p initially stores &a and q stores &b. Assignment p=q copies the address stored in q into p, so both point to b. Neither pointee value is assigned because no dereference appears; q is not assigned at all.

D - Incorrect: "q now points to a" does not match the rule or result derived here. p initially stores &a and q stores &b. Assignment p=q copies the address stored in q into p, so both point to b. Neither pointee value is assigned because no dereference appears; q is not assigned at all.

Primary note proof

Study-note section CP25

> int x=10; int* p=&x; stores x's address in p. *p reads or writes x; &p is the address of the pointer variable itself. With p=&a,q=&b, p=q redirects p to b and leaves a and b unchanged. By contrast *p=*q copies the pointed-to value. Never dereference an uninitialized, null or dangling pointer.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 26 | lines 2-22

> Pointer - A pointer is a variable that holds a memory address of another variable. - - Its content is a memory address - A pointer is simply the address of a memory location and provides an indirect way of accessing data in memory. - It is defined to „point to“ data of a specific type. - Because a pointer variable holds the address of another piece of data, it "points to" the data Reference operator (&) and Dereference operator (*) &: read as address of - Use address operator & to get address of a variable *: read as value pointed by Declaring variables of pointer types Syntax: type *variable_name; For example: int *ptr1; //pointertoanint double*number; char*ptr2; //pointertoachar The value of a pointer variable is the address to which it points. For example, given the definitions int num; wecanwrite: ptr1=#

Computer Programming module | PDF page 27 | lines 21-35

> int num1 = 5, num2 = 15; int *p1, *p2; p1 = &num1; p2 = &num2; cout<<*p1<<endl; *p1 = 10; cout<<"num1:"<<num1<<endl; *p2 = *p1; cout<<*p2<<endl; p1 = p2; cout<<*p1<<endl; *p1 = 20; cout<<"num1:"<< num1 << endl; cout<<"num2:"<<num2 << endl; cout<<*p2<<" "<<*p1;

Question sources

Computer Science model exam · Question 2

Computer Science model exam · Question 7

University of Gondar model exam · Question 2

University of Gondar model exam · Question 7

Question 24 | Q00272

Which of the following best describes the statement "C++ is a case sensitive programming language"?

A. All variables in C++ are identifiers

B. Every C++ statement should include the header file iostream.h

C. ExitExam and exitExam are distinct in C++

D. Every statement in C++ ends with semicolon

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: ExitExam and exitExam are distinct identifiers.

Case sensitivity means changing uppercase to lowercase can change the identifier. ExitExam and exitExam therefore name different entities in C++. Claims about headers, variable terminology or semicolon placement do not explain case sensitivity; the universal header and semicolon assertions also overstate C++ syntax requirements.

Why each choice is right or wrong

A - Incorrect: Variable names are identifiers, but this does not describe the effect of changing letter case.

B - Incorrect: Header inclusion is unrelated to case sensitivity, and headers are not included in every statement.

C - Correct: The differing initial letter case makes ExitExam and exitExam distinct C++ identifiers.

D - Incorrect: Semicolon rules concern grammar, not identifier case, and do not apply uniformly to every construct.

Primary note proof

Study-note section CP05

> In the basic ASCII identifier rules, start with a letter or underscore; continue with letters, digits or underscores. Do not use a keyword. C++ is case sensitive: exitExam and ExitExam differ; delete and float are keywords, but FLOAT is a legal identifier. variable_1234 is valid; 4thInput, T-3 and names containing spaces are not single identifiers. Avoid reserved underscore patterns in application code.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 2-10

> 1. Keywords (reserved words) - Unique meaning within a C++ program and must not be used for any other purposes. E.g., if, namespace for, switch, continue, float, new, default, break Etc. 2. Identifiers - Name associated with a function or data object and used to refer to that function or data object. An identifier must: o Start with a letter or underscore o Consist only of letters, the digits 0-9, or the underscore symbol _ o Not be a reserved word

Question sources

Computer Science model exam · Question 3

University of Gondar model exam · Question 3

Question 25 | Q00273

Assume a and b are distinct ordinary int variables and b has an initialized value. Immediately after a=b;, which statements about the assignment and subsequent changes to b are true? Select all that apply.

A. The value of a and b are equal

B. The value of b is assigned to variable a and the later change on variable b will affect the value of variable a

C. The value of variable a is assigned to variable b and the value of variable b is assigned to variable a

D. The value of b is assigned to variable a and the later change on variable b will not affect the value of variable a

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A, D: A. The value of a and b are equal; D. The value of b is assigned to variable a and the later change on variable b will not affect the value of variable a

Scalar assignment copies the value stored in b into the separate object a. Immediately afterward, the two values are equal. Their storage remains independent, however: changing b later does not automatically update a. Thus both the immediate-equality description and the copy-with-independence description are true. Assignment works in one direction and is neither a swap nor a reference binding between these two int variables.

Why each choice is right or wrong

A - Correct: The copied int values are equal immediately after assignment.

B - Incorrect: Separate scalar storage means later changes to b do not affect a.

C - Incorrect: Assignment copies one way; it does not swap values.

D - Correct: This correctly states both the copying direction and later independence.

Primary note proof

Study-note section CP06

> Write type name; or type name = value; for ordinary variables: int age = 20;. Declaration introduces a name and type; initialization gives its initial value; later assignment replaces a value. a=b copies a scalar value, so later changes to b do not change a. A pointer or Java object reference instead copies an address/reference.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Computer Programming module | PDF page 5 | lines 24-33

5. Data Types, Variables, and Constants Variable: - Variable is a symbolic name for a memory location in which data can be stored. - Attribute of Variables: o A type, which is, established when the variable is defined (e.g., integer, float, character) o A value, which can be changed by assigning a new value to the variable Variable Declaration - Declaring a variable means defining (creating) a variable. - Syntax: data type variable name; e.g. int num; - We can create More Than One Variable at a Time e.g., int num1, num2, num3;

Question sources

Computer Science model exam · Question 4

University of Gondar model exam · Question 4

Question 26 | Q00274

Which one of the following is true statement about an algorithm?

A. Ambiguous steps are allowed in an algorithm.

B. Complex problems cannot have an algorithm.

C. An algorithm may not have a beginning and an end.

D. An algorithm should take finite amount of time to compute.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: An algorithm should take finite amount of time to compute.

The module defines an algorithm as unambiguous executable steps forming a terminating process and explicitly requires finite execution for each input. A permits ambiguity, B imposes an unsupported complexity restriction, and C allows missing termination, contrary to that definition. Module support: PDF p. 2.

Why each choice is right or wrong

A - Incorrect: "Ambiguous steps are allowed in an algorithm." does not match the rule or result derived here. The module defines an algorithm as unambiguous executable steps forming a terminating process and explicitly requires finite execution for each input. A permits ambiguity, B imposes an unsupported complexity restriction, and C allows missing termination, contrary to that definition.

B - Incorrect: "Complex problems cannot have an algorithm." does not match the rule or result derived here. The module defines an algorithm as unambiguous executable steps forming a terminating process and explicitly requires finite execution for each input. A permits ambiguity, B imposes an unsupported complexity restriction, and C allows missing termination, contrary to that definition.

C - Incorrect: "An algorithm may not have a beginning and an end." does not match the rule or result derived here. The module defines an algorithm as unambiguous executable steps forming a terminating process and explicitly requires finite execution for each input. A permits ambiguity, B imposes an unsupported complexity restriction, and C allows missing termination, contrary to that definition.

D - Correct: "An algorithm should take finite amount of time to compute." matches the derived result. The module defines an algorithm as unambiguous executable steps forming a terminating process and explicitly requires finite execution for each input. A permits ambiguity, B imposes an unsupported complexity restriction, and C allows missing termination, contrary to that definition.

Primary note proof

Study-note section CP02

> An algorithm gives precise, executable steps and terminates for every legal input. Definiteness means each step has an unambiguous meaning; effectiveness means a step can actually be performed. Inputs may be absent, but a useful algorithm specifies a result. Correctness means the result meets the specification. Efficiency is desirable and measurable, but an inefficient terminating algorithm is still an algorithm. Efficiency is also a desirable property of an algorithm.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 2 | lines 19-26

> Algorithm is an ordered set of unambiguous, executable steps, defining a terminating process. Important Properties of Algorithms - Correct: always returns the desired output for all legal instances of the problem. - Unambiguous: Each step must be clearly defined, having only one interpretation. - Precise: simple but efficient - Efficient: can be measured in terms of Time and Space - Finiteness: the execution of the algorithm must terminate in a finite amount of time for each possible set of input data/must have an end.

Question sources

Computer Science model exam · Question 5

University of Gondar model exam · Question 5

Question 27 | Q00275

Which of the following will NOT increase an integer variable "i" by 1?

A. i+=1;

B. i=i+1; E. ++i;

C. i=i+i;

D. i++;

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: i=i+i;

i=i+i doubles i rather than adding a fixed one. i+=1, i=i+1, i++ and the embedded ++i all increment by one. At the special value i=1 doubling also adds one; the intended comparison is between operations for general i. Module support: PDF p. 7.

Why each choice is right or wrong

A - Incorrect: "i+=1;" does not match the rule or result derived here. i=i+i doubles i rather than adding a fixed one. i+=1, i=i+1, i++ and the embedded ++i all increment by one. At the special value i=1 doubling also adds one; the intended comparison is between operations for general i.

B - Incorrect: "i=i+1; E. ++i;" does not match the rule or result derived here. i=i+i doubles i rather than adding a fixed one. i+=1, i=i+1, i++ and the embedded ++i all increment by one. At the special value i=1 doubling also adds one; the intended comparison is between operations for general i.

C - Correct: "i=i+i;" matches the derived result. i=i+i doubles i rather than adding a fixed one. i+=1, i=i+1, i++ and the embedded ++i all increment by one. At the special value i=1 doubling also adds one; the intended comparison is between operations for general i.

D - Incorrect: "i++;" does not match the rule or result derived here. i=i+i doubles i rather than adding a fixed one. i+=1, i=i+1, i++ and the embedded ++i all increment by one. At the special value i=1 doubling also adds one; the intended comparison is between operations for general i.

Primary note proof

Study-note section CP08

> x+=y, x-=y, x*=y, x/=y and x%=y update x using the operation and conversion to its type. For simple variables, trace them as x=x op y. Starting p=q=n=8, p+=n gives 16 and p/=q gives 2. Starting x=10, x+=5 then x/=2 gives 7. x=x+x doubles; it does not generally add one.

Study-note section CP09

> Prefix ++x changes x and yields the new value; postfix x++ yields the previous value and also increments x.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Computer Science model exam · Question 6

University of Gondar model exam · Question 6

Question 28 | Q00336

Consider native C++ translation through assembly into an object file, stopping before linking. Which statement best describes the resulting object file’s instructions?

A. Machine-independent bytecode for a virtual machine

B. A complete linked executable

C. Both a and b

D. Native machine instructions in an object file

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: D. Native machine instructions in an object file

The specified native pipeline stops after the assembler produces an object file and before the linker combines objects and resolves external references. Its code is therefore native machine instructions in an object file. Machine-independent bytecode belongs to a different translation target, such as javac’s output, while a complete linked executable comes later in this C++ pipeline. Neither A nor B correctly names the specified result.

Why each choice is right or wrong

A - Incorrect: Machine-independent bytecode is a different target from the specified native object instructions.

B - Incorrect: The pipeline stops before linking produces a complete executable.

C - Incorrect: Both A and B describe different targets or later stages.

D - Correct: The assembler emits native machine instructions into an object file.

Primary note proof

Study-note section CP04

> In the conventional C++ toolchain: source → preprocessing → compilation proper → assembly → object files → linking → executable → loading → execution. Preprocessing handles directives such as #include; compilation may emit assembly; the assembler emits machine instructions in an object file. The linker resolves external symbols and combines object files and libraries. The loader places an executable and required components in memory and starts it.

Study-note section CP04

> Java javac produces bytecode, which a JVM interprets or compiles at runtime.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 4 | lines 21-30

> The compilation process of C++ program consists of the following steps. • edit source code → compile → link → execute (Editor) (Compiler) (Linker) (Loader) I. Program source code is entered into a file using a text editor. II. After the code has been entered, a compiler program is started that translates the source into an object code file. III. The object code file is linked with other object code files that come with the compiler and an executable file (or program) is created. IV. In order to execute the program, a program called the loader copies the executable file into the memory of the computer and sends an execute command to the CPU.

Question sources

Computer Science model exam · Question 67

University of Gondar model exam · Question 67

Question 29 | Q00455

A _____ provides us with named storage that our programs can manipulate.

A. data type

B. constants

C. operators

D. variable

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: variable

A variable is a symbolic name for a memory location holding data (p.5), so it supplies named manipulable storage. A data type classifies the value, constants have fixed values, and operators specify manipulations rather than storage. Module support: PDF p. 5.

Why each choice is right or wrong

A - Incorrect: "data type" does not match the rule or result derived here. A variable is a symbolic name for a memory location holding data (p.5), so it supplies named manipulable storage. A data type classifies the value, constants have fixed values, and operators specify manipulations rather than storage.

B - Incorrect: "constants" does not match the rule or result derived here. A variable is a symbolic name for a memory location holding data (p.5), so it supplies named manipulable storage. A data type classifies the value, constants have fixed values, and operators specify manipulations rather than storage.

C - Incorrect: "operators" does not match the rule or result derived here. A variable is a symbolic name for a memory location holding data (p.5), so it supplies named manipulable storage. A data type classifies the value, constants have fixed values, and operators specify manipulations rather than storage.

D - Correct: "variable" matches the derived result. A variable is a symbolic name for a memory location holding data (p.5), so it supplies named manipulable storage. A data type classifies the value, constants have fixed values, and operators specify manipulations rather than storage.

Primary note proof

Study-note section CP06

> Write type name; or type name = value; for ordinary variables: int age = 20;. Declaration introduces a name and type; initialization gives its initial value; later assignment replaces a value. a=b copies a scalar value, so later changes to b do not change a. A pointer or Java object reference instead copies an address/reference.

Study-note section CP25

> int x=10; int* p=&x; stores x’s address in p.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 24-33

> 5. Data Types, Variables, and Constants Variable: - Variable is a symbolic name for a memory location in which data can be stored. - Attribute of Variables: o A type, which is, established when the variable is defined (e.g., integer, float, character) o A value, which can be changed by assigning a new value to the variable Variable Declaration - Declaring a variable means defining (creating) a variable. - Syntax: data type variable name; e.g. int num; - We can create More Than One Variable at a Time e.g., int num1, num2, num3;

Question sources

Object-Oriented Programming practice questions · Question 25

Question 30 | Q00464

The && and || operators

A. Compare two boolean values

B. Compare two numeric values

C. Combine two boolean values

D. Combine two numeric values

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: Combine two boolean values

The logical operators combine truth-valued operands and produce 1 or 0 (p.7). Relational operators perform comparisons; arithmetic operators combine numeric quantities arithmetically. Numeric operands may be interpreted as truth values, but the logical role asked here is combination of boolean values. Module support: PDF p. 7.

Why each choice is right or wrong

A - Incorrect: "Compare two boolean values" does not match the rule or result derived here. The logical operators combine truth-valued operands and produce 1 or 0 (p.7). Relational operators perform comparisons; arithmetic operators combine numeric quantities arithmetically. Numeric operands may be interpreted as truth values, but the logical role asked here is combination of boolean values.

B - Incorrect: "Compare two numeric values" does not match the rule or result derived here. The logical operators combine truth-valued operands and produce 1 or 0 (p.7). Relational operators perform comparisons; arithmetic operators combine numeric quantities arithmetically. Numeric operands may be interpreted as truth values, but the logical role asked here is combination of boolean values.

C - Correct: "Combine two boolean values" matches the derived result. The logical operators combine truth-valued operands and produce 1 or 0 (p.7). Relational operators perform comparisons; arithmetic operators combine numeric quantities arithmetically. Numeric operands may be interpreted as truth values, but the logical role asked here is combination of boolean values.

D - Incorrect: "Combine two numeric values" does not match the rule or result derived here. The logical operators combine truth-valued operands and produce 1 or 0 (p.7). Relational operators perform comparisons; arithmetic operators combine numeric quantities arithmetically. Numeric operands may be interpreted as truth values, but the logical role asked here is combination of boolean values.

Primary note proof

Study-note section CP10

> Relational and equality operators test values; && and || combine truth values; ! negates. In C++, zero converts to false and a nonzero number to true. Streams display bool as 0 or 1 unless boolalpha is set. With a=3,b=5, cout<<(a==b) prints 0.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Object-Oriented Programming practice questions · Question 34

Question 31 | Q00470

Which of these have highest precedence?

A. ()

B. ++

C. *

D. >>

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: Parentheses ().

CP10 places grouping/postfix at the top of its precedence ladder. Parentheses explicitly group an expression before surrounding operations are applied. Increment, multiplication or dereference, and right shift do not override that grouping. The precise role of * or ++ in an expression does not make it outrank explicit parentheses.

Why each choice is right or wrong

A - Correct: Parentheses explicitly group expressions and occupy the highest grouping level in the note.

B - Incorrect: Increment does not override parentheses; prefix increment is unary and postfix increment follows grouping rules.

C - Incorrect: Multiplication is below grouping; unary dereference also does not override parenthesized grouping.

D - Incorrect: Right shift ranks below both grouping and arithmetic operations in the stated ladder.

Primary note proof

Study-note section CP10

> && evaluates its right operand only if the left is true; || only if the left is false. Thus false && dangerousCall() skips the call. Bitwise & operates on bits and does not short circuit. With x=1,y=0,z=4, (x&y)||++z evaluates ++z, leaving z=5 and the Boolean result 1. The precedence ladder is grouping/postfix, unary, multiplicative, additive, shifts, relations, equality, bitwise &, ^, |, &&, ||, conditional, assignment, comma.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Object-Oriented Programming practice questions · Question 40

Question 32 | Q00479

A _____ statement allows us to execute a statement or group of statements multiple times.

A. array

B. loop

C. function

D. exception

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: loop

Loop statements specify repeated computations (p.9), and iteration repeats a statement or block (p.11). Arrays group data, functions modularize code, and exceptions are not the module definition of repetition. Module support: PDF p. 9, PDF p. 11.

Why each choice is right or wrong

A - Incorrect: "array" does not match the rule or result derived here. Loop statements specify repeated computations (p.9), and iteration repeats a statement or block (p.11). Arrays group data, functions modularize code, and exceptions are not the module definition of repetition.

B - Correct: "loop" matches the derived result. Loop statements specify repeated computations (p.9), and iteration repeats a statement or block (p.11). Arrays group data, functions modularize code, and exceptions are not the module definition of repetition.

C - Incorrect: "function" does not match the rule or result derived here. Loop statements specify repeated computations (p.9), and iteration repeats a statement or block (p.11). Arrays group data, functions modularize code, and exceptions are not the module definition of repetition.

D - Incorrect: "exception" does not match the rule or result derived here. Loop statements specify repeated computations (p.9), and iteration repeats a statement or block (p.11). Arrays group data, functions modularize code, and exceptions are not the module definition of repetition.

Primary note proof

Study-note section CP14

> while(condition) checks before each body execution and may run zero times. do {body;} while(condition); runs the body once before checking; it repeats while the condition remains true. Use it when a menu or input attempt must occur at least once.

Study-note section CP32

> Exceptions transfer control to a matching handler for exceptional conditions; they do not make arbitrary undefined behavior recoverable.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 9 | lines 2-8

> Control Statements Control statements or flow control determines what is executed during a run and what is not, therefore affecting the overall outcome of the program. C++ provides different forms of statements for different purposes. Declaration statements are used for defining variables. Assignment-like statements are used for simple, algebraic computations. Branching statements are used for specifying alternate paths of execution, depending on the outcome of a

logical condition. Loop statements are used for specifying computations, which need to be repeated until a certain logical condition is satisfied

Computer Programming module | PDF page 11 | lines 5-13

Iteration/Looping Statement: Iteration is the repetition of a statement or block of statements in a program. - Loops have as objective to repeat a statement a certain number of times or while a condition is fulfilled. - C++ has three iteration statements: the while statement, the do. While statement, and the for statement. The while loop: Its format is: while (expression) {statement;} First expression (called the loop condition) is evaluated. If the outcome is nonzero then statement (called the loop body) is executed and the whole process is repeated. Otherwise, the loop is terminated.

Question sources

Object-Oriented Programming practice questions · Question 49

Question 33 | Q00482

A for loop is useful when you know how many times a task is to be repeated.

A. TRUE

B. FALSE

C. Can be true or false

D. Can not say

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: TRUE

A for loop has initialization, a condition and an update field (p.12). These directly express a known repetition count using a counter, so the stated usefulness is true; it does not claim that all for loops must have predetermined counts. Module support: PDF p. 12.

Why each choice is right or wrong

A - Correct: "TRUE" matches the derived result. A for loop has initialization, a condition and an update field (p.12). These directly express a known repetition count using a counter, so the stated usefulness is true; it does not claim that all for loops must have predetermined counts.

B - Incorrect: "FALSE" does not match the rule or result derived here. A for loop has initialization, a condition and an update field (p.12). These directly express a known repetition count using a counter, so the stated usefulness is true; it does not claim that all for loops must have predetermined counts.

C - Incorrect: "Can be true or false" does not match the rule or result derived here. A for loop has initialization, a condition and an update field (p.12). These directly express a known repetition

count using a counter, so the stated usefulness is true; it does not claim that all for loops must have predetermined counts.

D - Incorrect: "Can not say" does not match the rule or result derived here. A for loop has initialization, a condition and an update field (p.12). These directly express a known repetition count using a counter, so the stated usefulness is true; it does not claim that all for loops must have predetermined counts.

Primary note proof

Study-note section CP14

> for(initialization; condition; update) executes initialization once, then condition → body → update → condition. A missing condition in for(;;) acts as true. A for loop is convenient for a known count but can also be condition controlled. To print indices 0 through n-1, use i=0; i<n; ++i. Starting i=0 with i<5 and i+=2 prints 0 2 4.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 12 | lines 19-28

> The for loop. Its format is: for (initialization; condition; increase) statement; 1, initialization is executed. Generally, it is a initial value setting for a counter variable. This is executed only once 2, condition is checked, if it is true the loop continues, otherwise the loop finishes and statement is skipped. 3, statement is executed. As usual, it can be either a single instruction or a block of instructions enclosed within curly brackets { }. 4, finally, whatever is specified in the increase field is executed and the loop gets back to step 2.

Question sources

Object-Oriented Programming practice questions · Question 52

Question 34 | Q00627

Which one of the following are translate the whole source program into object code.

A. Compiler

B. Interprater

C. Linker

D. All

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: Compiler

The module says compilers translate the entire source before execution and produce object code (pp.3-4). Interpreters translate one line at a time. Linking combines existing object files rather than translating the source, so All cannot hold. Module support: PDF p. 4, PDF p. 3.

Why each choice is right or wrong

A - Correct: "Compiler" matches the derived result. The module says compilers translate the entire source before execution and produce object code (pp.3-4). Interpreters translate one line at a time. Linking combines existing object files rather than translating the source, so All cannot hold.

B - Incorrect: "Interprater" does not match the rule or result derived here. The module says compilers translate the entire source before execution and produce object code (pp.3-4). Interpreters translate one line at a time. Linking combines existing object files rather than translating the source, so All cannot hold.

C - Incorrect: "Linker" does not match the rule or result derived here. The module says compilers translate the entire source before execution and produce object code (pp.3-4). Interpreters translate one line at a time. Linking combines existing object files rather than translating the source, so All cannot hold.

D - Incorrect: "All" does not match the rule or result derived here. The module says compilers translate the entire source before execution and produce object code (pp.3-4). Interpreters translate one line at a time. Linking combines existing object files rather than translating the source, so All cannot hold.

Primary note proof

Study-note section CP04

> In the conventional C++ toolchain: source → preprocessing → compilation proper → assembly → object files → linking → executable → loading → execution. Preprocessing handles directives such as #include; compilation may emit assembly; the assembler emits machine instructions in an object file. The linker resolves external symbols and combines object files and libraries. The loader places an executable and required components in memory and starts it.

Study-note section CP04

> GCC documents the first four stages explicitly. A compiler need not emit a complete executable or even native machine code. Java javac produces bytecode, which a JVM interprets or compiles at runtime. Interpreters execute a program representation; “one source line at a time” is a beginner approximation. Compilation uses multiple passes, so “whole program in one step” is not literally universal.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 4 | lines 21-30

> The compilation process of C++ program consists of the following steps. • edit source code → compile → link → execute (Editor) (Compiler) (Linker) (Loader) I. Program source code is entered into a file using a text editor. II. After the code has been entered, a compiler program is started

that translates the source into an object code file. III. The object code file is linked with other object code files that come with the compiler and an executable file (or program) is created. IV. In order to execute the program, a program called the loader copies the executable file into the memory of the computer and sends an execute command to the CPU.

Computer Programming module | PDF page 3 | lines 8-17

Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Question sources

Computer Science model exam · 2015 · Question 96

Question 35 | Q00660

In C++ programming, a variable which stores a reference to another variable is called a____

A. Pointer

B. Constructor

C. Asterisks

D. Structure

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: Pointer

The module defines a pointer as a variable holding the memory address of another variable (p.26). Here reference is informal wording for an address, not the distinct C++ reference type. Constructor, the asterisk symbol and structure do not name that variable category. Module support: PDF p. 26.

Why each choice is right or wrong

A - Correct: "Pointer" matches the derived result. The module defines a pointer as a variable holding the memory address of another variable (p.26). Here reference is informal wording for an address, not the distinct C++ reference type. Constructor, the asterisk symbol and structure do not name that variable category.

B - Incorrect: "Constructor" does not match the rule or result derived here. The module defines a pointer as a variable holding the memory address of another variable (p.26). Here reference is

informal wording for an address, not the distinct C++ reference type. Constructor, the asterisk symbol and structure do not name that variable category.

C - Incorrect: "Asterisks" does not match the rule or result derived here. The module defines a pointer as a variable holding the memory address of another variable (p.26). Here reference is informal wording for an address, not the distinct C++ reference type. Constructor, the asterisk symbol and structure do not name that variable category.

D - Incorrect: "Structure" does not match the rule or result derived here. The module defines a pointer as a variable holding the memory address of another variable (p.26). Here reference is informal wording for an address, not the distinct C++ reference type. Constructor, the asterisk symbol and structure do not name that variable category.

Primary note proof

Study-note section CP25

int x=10; int* p=&x; stores x's address in p. *p reads or writes x; &p is the address of the pointer variable itself. With p=&a,q=&b, p=q redirects p to b and leaves a and b unchanged. By contrast *p=*q copies the pointed-to value. Never dereference an uninitialized, null or dangling pointer.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 26 | lines 2-22

Pointer - A pointer is a variable that holds a memory address of another variable. - - Its content is a memory address - A pointer is simply the address of a memory location and provides an indirect way of accessing data in memory. - It is defined to „point to" data of a specific type. - Because a pointer variable holds the address of another piece of data, it "points to" the data Reference operator (&) and Dereference operator (*) &: read as address of - Use address operator & to get address of a variable *: read as value pointed by Declaring variables of pointer types Syntax: type *variable_name; For example: int *ptr1; //pointertoanint double*number; char*ptr2; //pointertoachar The value of a pointer variable is the address to which it points. For example, given the definitions int num; wecanwrite: ptr1=#

Question sources

Computer Science model exam 2 · Question 1

Question 36 | Q00661

A program that translates a high-level language like C++ to a machine language is called a

A. Disassembler

B. Coder

C. Compiler

D. Assembler

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: Compiler

A compiler translates the high-level source program before execution (pp.3-4). A linker combines object files and an assembler is not named here as the high-level translator; disassembler/coder do not match the stated compilation role. Module support: PDF p. 3, PDF p. 4.

Why each choice is right or wrong

A - Incorrect: "Disassembler" does not match the rule or result derived here. A compiler translates the high-level source program before execution (pp.3-4). A linker combines object files and an assembler is not named here as the high-level translator; disassembler/coder do not match the stated compilation role.

B - Incorrect: "Coder" does not match the rule or result derived here. A compiler translates the high-level source program before execution (pp.3-4). A linker combines object files and an assembler is not named here as the high-level translator; disassembler/coder do not match the stated compilation role.

C - Correct: "Compiler" matches the derived result. A compiler translates the high-level source program before execution (pp.3-4). A linker combines object files and an assembler is not named here as the high-level translator; disassembler/coder do not match the stated compilation role.

D - Incorrect: "Assembler" does not match the rule or result derived here. A compiler translates the high-level source program before execution (pp.3-4). A linker combines object files and an assembler is not named here as the high-level translator; disassembler/coder do not match the stated compilation role.

Primary note proof

Study-note section CP04

> In the conventional C++ toolchain: source → preprocessing → compilation proper → assembly → object files → linking → executable → loading → execution. Preprocessing handles directives such as #include; compilation may emit assembly; the assembler emits machine instructions in an object file. The linker resolves external symbols and combines object files and libraries. The loader places an executable and required components in memory and starts it.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

> Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Computer Programming module | PDF page 4 | lines 21-30

> The compilation process of C++ program consists of the following steps. • edit source code → compile → link → execute (Editor) (Compiler) (Linker) (Loader) I. Program source code is entered into a file using a text editor. II. After the code has been entered, a compiler program is started that translates the source into an object code file. III. The object code file is linked with other object code files that come with the compiler and an executable file (or program) is created. IV. In order to execute the program, a program called the loader copies the executable file into the memory of the computer and sends an execute command to the CPU.

Question sources

Computer Science model exam 2 · Question 2

Question 37 | Q00662

Given int A=0, B=10, x=1, y=2;, which C++ Boolean expression evaluates to false?

A. (B == 10) || (x < y)

B. (A > 20) || (B >5)

C. (A > 10) && (x < y)

D. (A == 0) && (B < 20)

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: C. (A > 10) && (x < y)

Evaluate each comparison using the declared integers and apply short-circuit logic. B==10 is true, so the first disjunction is true. Although A>20 is false, B>5 makes the second disjunction true. In C, A>10 is false, so && yields false without needing its right operand. Both comparisons in D are true. Every offered expression is well formed under the stated declarations.

Why each choice is right or wrong

A - Incorrect: B==10 is true, so this disjunction is true.

B - Incorrect: B>5 is true, so this disjunction is true.

C - Correct: A>10 is false, making the conjunction false.

D - Incorrect: A==0 and B<20 are both true.

Primary note proof

Study-note section CP10

> Relational and equality operators test values; && and || combine truth values; ! negates. In C++, zero converts to false and a nonzero number to true. Streams display bool as 0 or 1 unless boolalpha is set. With a=3,b=5, cout<<(a==b) prints 0.

Study-note section CP10

&& evaluates its right operand only if the left is true; || only if the left is false.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Computer Science model exam 2 · Question 3

Question 38 | Q00663

Which one of the following format is write about functions in structural Programming:

A. Type name { parameter1, parameter2, ...} (statement)

B. Name type (parameter1, parameter2, ...) { statement }

C. Type name (parameter1, parameter2, ...) {statement }

D. Name type { parameter1, parameter2, ...} (statement)

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: Type name (parameter1, parameter2, ...) {statement }

The module function-definition pattern is Function_type function_name(Parameter List) followed by a braced function body (p.15). C preserves that order and delimiters. A/D interchange braces and parentheses, while B puts name before type. Module support: PDF p. 15.

Why each choice is right or wrong

A - Incorrect: "Type name { parameter1, parameter2, ...} (statement)" does not match the rule or result derived here. The module function-definition pattern is Function_type function_name(Parameter List) followed by a braced function body (p.15). C preserves that order and delimiters. A/D interchange braces and parentheses, while B puts name before type.

B - Incorrect: "Name type (parameter1, parameter2, ...) { statement }" does not match the rule or result derived here. The module function-definition pattern is Function_type function_name(Parameter List) followed by a braced function body (p.15). C preserves that order and delimiters. A/D interchange braces and parentheses, while B puts name before type.

C - Correct: "Type name (parameter1, parameter2, ...) {statement }" matches the derived result. The module function-definition pattern is Function_type function_name(Parameter List) followed by a braced function body (p.15). C preserves that order and delimiters. A/D interchange braces and parentheses, while B puts name before type.

D - Incorrect: "Name type { parameter1, parameter2, ...} (statement)" does not match the rule or result derived here. The module function-definition pattern is Function_type function_name(Parameter List) followed by a braced function body (p.15). C preserves that order and delimiters. A/D interchange braces and parentheses, while B puts name before type.

Primary note proof

Study-note section CP17

A function groups a named task. C++ definition: int add(int a,int b){return a+b;}. Call it with add(x,y);, without parameter type declarations at the call site. Parameters are names in the definition; arguments are supplied expressions or values. A void function produces no return value but can have side effects and return; to exit early.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 15 | lines 29-46

• User-defined - Function that been created by the user. - These functions need to be declared and defined by the user. Declaration of function Function defines using the following syntax: Function_type function_name (Parameter List) { ...function body } Example: void Addition(int num1,int num2) #include<iostream> using namespace std; void Addition(int num1,int num2) { int sum=num1+num2; cout<<"The sum is:"<<sum; } int main() { int x=10,y=20; Addition(x,y);

Question sources

Computer Science model exam 2 · Question 4

Question 39 | Q00664

Which of the following is a valid function call (assuming the function add exists & have two parameters)?

A. add;

B. add(a,b);

C. add a,b;

D. add(int a, int b);

Hint

Apply the naming, type and syntax rules before deciding whether an option can be used as stated.

Answer B: add(a,b);

The example Addition(x,y) on p.15 calls a function using its name and parenthesized argument expressions. add(a,b); has that form. A does not call it, C lacks parentheses, and D inserts declarations where argument expressions are required. Module support: PDF p. 15.

Why each choice is right or wrong

A - Incorrect: "add;" does not match the rule or result derived here. The example Addition(x,y) on p.15 calls a function using its name and parenthesized argument expressions. add(a,b); has that form. A does not call it, C lacks parentheses, and D inserts declarations where argument expressions are required.

B - Correct: "add(a,b);" matches the derived result. The example Addition(x,y) on p.15 calls a function using its name and parenthesized argument expressions. add(a,b); has that form. A does not call it, C lacks parentheses, and D inserts declarations where argument expressions are required.

C - Incorrect: "add a,b;" does not match the rule or result derived here. The example Addition(x,y) on p.15 calls a function using its name and parenthesized argument expressions. add(a,b); has that form. A does not call it, C lacks parentheses, and D inserts declarations where argument expressions are required.

D - Incorrect: "add(int a, int b);" does not match the rule or result derived here. The example Addition(x,y) on p.15 calls a function using its name and parenthesized argument expressions. add(a,b); has that form. A does not call it, C lacks parentheses, and D inserts declarations where argument expressions are required.

Primary note proof

Study-note section CP17

> A function groups a named task. C++ definition: int add(int a,int b){return a+b;}. Call it with add(x,y);, without parameter type declarations at the call site. Parameters are names in the definition; arguments are supplied expressions or values. A void function produces no return value but can have side effects and return; to exit early.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 15 | lines 29-46

> • User-defined - Function that been created by the user. - These functions need to be declared and defined by the user. Declaration of function Function defines using the following syntax: Function_type function_name (Parameter List) { ...function body } Example: void Addition(int num1,int num2) #include<iostream> using namespace std; void Addition(int num1,int num2) { int sum=num1+num2; cout<<"The sum is:"<<sum; } int main() { int x=10,y=20; Addition(x,y);

Question sources

Computer Science model exam 2 · Question 5

Question 40 | Q00665

A function is said to be recursive function_____?

A. If the function calls another function2 & Function3

B. If the main function calls many functions

C. If the main function calls another one function

D. If the function calls itself

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: If the function calls itself

A recursive function calls itself (p.18). Calling several other functions, or being called by main, does not itself constitute recursion. Module support: PDF p. 18.

Why each choice is right or wrong

A - Incorrect: "If the function calls another function2 & Function3" does not match the rule or result derived here. A recursive function calls itself (p.18). Calling several other functions, or being called by main, does not itself constitute recursion.

B - Incorrect: "If the main function calls many functions" does not match the rule or result derived here. A recursive function calls itself (p.18). Calling several other functions, or being called by main, does not itself constitute recursion.

C - Incorrect: "If the main function calls another one function" does not match the rule or result derived here. A recursive function calls itself (p.18). Calling several other functions, or being called by main, does not itself constitute recursion.

D - Correct: "If the function calls itself" matches the derived result. A recursive function calls itself (p.18). Calling several other functions, or being called by main, does not itself constitute recursion.

Primary note proof

Study-note section CP21

> A recursive function calls itself directly or indirectly. It needs a base case and progress toward it for all supported inputs. factorial(0)=1 and factorial(n)=n*factorial(n-1) for positive n. factorial(3) expands to 3*2*1*1=6. Negative inputs require rejection or a separate definition; they do not reach zero under n-1.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 18 | lines 23-28

> Recursive Functions - Functions that call themselves - A function that is defined in terms of itself is called recursive - There must be a base case e.g. 0! = 1 - Slowly converges towards base case - For example, Recursive functions N! = N * (N-1)!

Question sources

Computer Science model exam 2 · Question 6

Question 41 | Q00666

What will be the return type of a function, if the function does not have a return value?

A. Int

B. Float

C. Double

D. Void

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: Void

The module illustrates void Addition(...) whose body prints a result without returning a value (p.15). This matches void for no returned value; int, float and double denote value-bearing return types. Capitalization in the options is treated as prose. Module support: PDF p. 15.

Why each choice is right or wrong

A - Incorrect: "Int" does not match the rule or result derived here. The module illustrates void Addition(...) whose body prints a result without returning a value (p.15). This matches void for no returned value; int, float and double denote value-bearing return types. Capitalization in the options is treated as prose.

B - Incorrect: "Float" does not match the rule or result derived here. The module illustrates void Addition(...) whose body prints a result without returning a value (p.15). This matches void for no returned value; int, float and double denote value-bearing return types. Capitalization in the options is treated as prose.

C - Incorrect: "Double" does not match the rule or result derived here. The module illustrates void Addition(...) whose body prints a result without returning a value (p.15). This matches void for no returned value; int, float and double denote value-bearing return types. Capitalization in the options is treated as prose.

D - Correct: "Void" matches the derived result. The module illustrates void Addition(...) whose body prints a result without returning a value (p.15). This matches void for no returned value; int, float and double denote value-bearing return types. Capitalization in the options is treated as prose.

Primary note proof

Study-note section CP17

A function groups a named task. C++ definition: int add(int a,int b){return a+b;}. Call it with add(x,y);, without parameter type declarations at the call site. Parameters are names in the definition; arguments are supplied expressions or values. A void function produces no return value but can have side effects and return; to exit early.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 15 | lines 29-46

• User-defined - Function that been created by the user. - These functions need to be declared and defined by the user. Declaration of function Function defines using the following syntax: Function_type function_name (Parameter List) { ...function body } Example: void Addition(int num1,int num2) #include<iostream> using namespace std; void Addition(int num1,int num2) { int sum=num1+num2; cout<<"The sum is:"<<sum; } int main() { int x=10,y=20; Addition(x,y);

Question sources

Computer Science model exam 2 · Question 7

Question 42 | Q00667

____is a header file which permits writing of data to a file.

A. ofstream

B. ifstream

C. istream

D. fstream

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: fstream

The module explicitly instructs including the <fstream> header to work with files (p.40). ofstream is the class for writing, ifstream the reading class, and istream is not the named file header. The question asks for the header, so fstream is the applicable choice. Module support: PDF p. 40.

Why each choice is right or wrong

A - Incorrect: "ofstream" does not match the rule or result derived here. The module explicitly instructs including the <fstream> header to work with files (p.40). ofstream is the class for writing, ifstream the reading class, and istream is not the named file header. The question asks for the header, so fstream is the applicable choice.

B - Incorrect: "ifstream" does not match the rule or result derived here. The module explicitly instructs including the <fstream> header to work with files (p.40). ofstream is the class for writing, ifstream the reading class, and istream is not the named file header. The question asks for the header, so fstream is the applicable choice.

C - Incorrect: "istream" does not match the rule or result derived here. The module explicitly instructs including the <fstream> header to work with files (p.40). ofstream is the class for writing, ifstream the reading class, and istream is not the named file header. The question asks for the header, so fstream is the applicable choice.

D - Correct: "fstream" matches the derived result. The module explicitly instructs including the <fstream> header to work with files (p.40). ofstream is the class for writing, ifstream the reading class, and istream is not the named file header. The question asks for the header, so fstream is the applicable choice.

Primary note proof

Study-note section CP29

> Include <fstream>. ofstream writes, ifstream reads, and fstream can do both. The constructor's default ofstream mode argument is ios::out; its ordinary effect is to create/truncate the file. Distinguish the default mode argument from effective out|trunc behavior. ios::app places each write at the end; ios::ate initially seeks to the end; ios::binary requests binary mode.

Study-note section CP29

> "ofstream" is a class, whereas <fstream> is the header.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 40 | lines 16-22

> File and File Management Files are used to store data in storage devices permanently. The fstream library allows us to work with files. To use the fstream library, include both the standard <iostream> AND the <fstream> header file: Input/Output with files • ofstream: Stream class to write on files • ifstream: Stream class to read from files

Question sources

Computer Science model exam 2 · Question 8

Question 43 | Q00668

Which statements about C++ structures are true? Select all that apply. In C, initialization refers to defining a structure object with initial member values.

A. A structure must contain only homogeneous members.

B. The named individual components of a structure are called members.

C. A structure object can be initialized when it is defined.

D. All are correct

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B, C: B. The named individual components of a structure are called members.; C. A structure object can be initialized when it is defined.

A structure groups named members and can combine different member types, so it need not be homogeneous. Those named components are called members. Defining an object with an aggregate initializer supplies initial values in member declaration order, making the initialization statement true. Since A wrongly requires homogeneity, the aggregate claim that all statements are correct is false even though B and C hold.

Why each choice is right or wrong

A - Incorrect: Members can have different types; homogeneity is not required.

B - Correct: The named individual components of a structure are its members.

C - Correct: A structure object can be initialized when the object is defined.

D - Incorrect: A is false, so the aggregate all-correct claim fails.

Primary note proof

Study-note section CP27

> A struct groups named members, possibly of different types. struct House{double price; int rooms;}; defines a type; House h{150000,3}; creates an object. h.rooms uses a dot. House* p=&h; p->rooms is equivalent to (*p).rooms. The pointer must refer to a live object before member access. A type definition alone does not create an ordinary instance.

Study-note section CP27

> Aggregate initialization assigns members in declaration order.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 35 | lines 13-17

> Accessing members of a structure variable A structure member can be accessed using dot operator'.' notation syntax: structvariable.member; eg. st.id=4;

Computer Programming module | PDF page 35 | lines 37-44

> Initialize value #include<iostream> using namespace std; struct student{ string name; int age; double score; }st={"Abebe",19,24};

Question sources

Computer Science model exam 2 · Question 9

Question 44 | Q00669

Which scope category describes a variable declared outside functions, usable where its declaration is visible and potentially hidden by a local variable of the same name?

A. Global Variable

B. Limited Variables

C. Local Variable

D. Partial Variable

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: A. Global Variable

A namespace or global variable is declared outside a function. Its declaration must be visible at the point of use, so global does not mean automatically accessible in every file or scope. A local name can shadow it, and C++ scope resolution can select a hidden global name. This makes Global Variable the appropriate category while avoiding the earlier overstatement about unrestricted program-wide accessibility.

Why each choice is right or wrong

A - Correct: Global variables have the stated outside-function scope, subject to declaration visibility.

B - Incorrect: Limited Variables is not the scope category specified by the declarations.

C - Incorrect: A local variable is declared within a function or block.

D - Incorrect: Partial Variable is not the stated C++ scope category.

Primary note proof

Study-note section CP20

> Scope is where a name can be used; lifetime is when its object exists. A local name is restricted to its block; an outer name may be shadowed. A namespace/global variable is available where its declaration is visible, not magically in every file. C++ ::name selects a global name hidden by a local one.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 17 | lines 14-29

> Scope of variables Scope of variables defined as the extent of the program code within which the variable can be accessed or declared or worked with mainly two types of variable scopes: local variable: - Variables defined within a function or block are said to be local to those functions - Declared inside a block{} Global Variable - variable that can be accessed from any part of the program. - They are available throughout the life time of a program. - They are declared at the

top of the program outside all of the functions or blocks. NB: - Usually when two variables with same name are defined then the compiler produces a compile time error. But if the variables are defined in different scopes, then the compiler allows it. - Whenever there is a local variable defined with same name as that of a global variable then the compiler will give precedence to the local variable

Question sources

Computer Science model exam 2 · Question 10

Question 45 | Q00906

What is the output of the following C++ fragment code? int a = 6, b = 8; int x = 2, y = 4; int c = (x > y ? (a--, x) : (b--, y)); cout << "a=" << a; cout << " b=" << b; cout << " c=" << c;

A. a=5 b=7 c=2

B. a=5 b=8 c=4

C. a=6 b=8 c=4

D. a=6 b=7 c=4

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer D: a=6 b=7 c=4

x>y is 2>4, false, so only the third operand (b--,y) is evaluated. It decrements b from 8 to 7 and yields y=4; a remains 6. Thus a=6 b=7 c=4. A/B incorrectly decrement a; C omits the selected decrement. Module support: PDF p. 8, PDF p. 7.

Why each choice is right or wrong

A - Incorrect: "a=5 b=7 c=2" does not match the rule or result derived here. x>y is 2>4, false, so only the third operand (b--,y) is evaluated. It decrements b from 8 to 7 and yields y=4; a remains 6. Thus a=6 b=7 c=4. A/B incorrectly decrement a; C omits the selected decrement.

B - Incorrect: "a=5 b=8 c=4" does not match the rule or result derived here. x>y is 2>4, false, so only the third operand (b--,y) is evaluated. It decrements b from 8 to 7 and yields y=4; a remains 6. Thus a=6 b=7 c=4. A/B incorrectly decrement a; C omits the selected decrement.

C - Incorrect: "a=6 b=8 c=4" does not match the rule or result derived here. x>y is 2>4, false, so only the third operand (b--,y) is evaluated. It decrements b from 8 to 7 and yields y=4; a remains 6. Thus a=6 b=7 c=4. A/B incorrectly decrement a; C omits the selected decrement.

D - Correct: "a=6 b=7 c=4" matches the derived result. x>y is 2>4, false, so only the third operand (b--,y) is evaluated. It decrements b from 8 to 7 and yields y=4; a remains 6. Thus a=6 b=7 c=4. A/B incorrectly decrement a; C omits the selected decrement.

Primary note proof

Study-note section CP11

> condition ? yes : no evaluates the condition and exactly one alternative. The comma operator evaluates its left expression, then its right, and yields the right value. Do not confuse it with commas separating function arguments.

Study-note section CP09

> Prefix ++x changes x and yields the new value; postfix x++ yields the previous value and also increments x.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 8 | lines 2-6

> Ternary /Conditional operator (? :) A ternary operator evaluates the test condition and executes a block of code based on the result of the condition. Syntax: Condition? Expression 1: Expression2 If the condition is true expression 1 will be evaluated otherwise Expression 2 will evaluated.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Computer Science Exit Exam · 2018 · Question 19

MOE Computer Science Exit Exam · Question 25-2

National Computer Science Exit Exam · Question 23

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 77

Question 46 | Q00936

Which of the following is NOT property of an algorithm?

A. Finiteness

B. Definiteness

C. Correctness

D. Platform dependence

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: Platform dependence

The module properties include correctness, unambiguity and finiteness (p.2). Platform dependence is not a property demanded of an abstract algorithm. Definiteness corresponds to clearly defined unambiguous steps; D is the extraneous platform restriction. Module support: PDF p. 2.

Why each choice is right or wrong

A - Incorrect: "Finiteness" does not match the rule or result derived here. The module properties include correctness, unambiguity and finiteness (p.2). Platform dependence is not a property demanded of an abstract algorithm. Definiteness corresponds to clearly defined unambiguous steps; D is the extraneous platform restriction.

B - Incorrect: "Definiteness" does not match the rule or result derived here. The module properties include correctness, unambiguity and finiteness (p.2). Platform dependence is not a property demanded of an abstract algorithm. Definiteness corresponds to clearly defined unambiguous steps; D is the extraneous platform restriction.

C - Incorrect: "Correctness" does not match the rule or result derived here. The module properties include correctness, unambiguity and finiteness (p.2). Platform dependence is not a property demanded of an abstract algorithm. Definiteness corresponds to clearly defined unambiguous steps; D is the extraneous platform restriction.

D - Correct: "Platform dependence" matches the derived result. The module properties include correctness, unambiguity and finiteness (p.2). Platform dependence is not a property demanded of an abstract algorithm. Definiteness corresponds to clearly defined unambiguous steps; D is the extraneous platform restriction.

Primary note proof

Study-note section CP02

> An algorithm gives precise, executable steps and terminates for every legal input. Definiteness means each step has an unambiguous meaning; effectiveness means a step can actually be performed. Inputs may be absent, but a useful algorithm specifies a result. Correctness means the result meets the specification. Efficiency is desirable and measurable, but an inefficient

terminating algorithm is still an algorithm. Efficiency is also a desirable property of an algorithm.

Study-note section CP01

Solve one small case manually, choose an algorithm, express it as pseudocode or a flowchart, implement, test and maintain.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 2 | lines 19-26

Algorithm is an ordered set of unambiguous, executable steps, defining a terminating process. Important Properties of Algorithms - Correct: always returns the desired output for all legal instances of the problem. - Unambiguous: Each step must be clearly defined, having only one interpretation. - Precise: simple but efficient - Efficient: can be measured in terms of Time and Space - Finiteness: the execution of the algorithm must terminate in a finite amount of time for each possible set of input data/must have an end.

Question sources

Computer Science Exit Exam · 2018 · Question 49

Question 47 | Q00938

Assume that you are computer science expert and you need to develop large programs within short period of time. Finally, you have planned to provide portable program to the concerned body. Which type of programming language is preferable for the above scenario?

A. Low level

B. Machine level

C. .
High level

D. Assembly

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: .
High level

The decisive requirement is portability: the module says high-level languages are machine-independent and run on a variety of computers (p.3). Machine and assembly are low-level and machine-dependent. Thus C meets portability; the module does not quantify development time. Module support: PDF p. 3.

Why each choice is right or wrong

A - Incorrect: "Low level" does not match the rule or result derived here. The decisive requirement is portability: the module says high-level languages are machine-independent and run on a variety of computers (p.3). Machine and assembly are low-level and machine-dependent. Thus C meets portability; the module does not quantify development time.

B - Incorrect: "Machine level" does not match the rule or result derived here. The decisive requirement is portability: the module says high-level languages are machine-independent and run on a variety of computers (p.3). Machine and assembly are low-level and machine-dependent. Thus C meets portability; the module does not quantify development time.

C - Correct: ".
High level" matches the derived result. The decisive requirement is portability: the module says high-level languages are machine-independent and run on a variety of computers (p.3). Machine and assembly are low-level and machine-dependent. Thus C meets portability; the module does not quantify development time.

D - Incorrect: "Assembly" does not match the rule or result derived here. The decisive requirement is portability: the module says high-level languages are machine-independent and run on a variety of computers (p.3). Machine and assembly are low-level and machine-dependent. Thus C meets portability; the module does not quantify development time.

Primary note proof

Study-note section CP03

> Machine and assembly code depend on an instruction set. High-level source is easier to read and often portable after recompilation, provided it avoids platform assumptions. People can write both assembly and high-level programs. Python, Perl and PHP are commonly called scripting languages; C++ is conventionally compiled. These labels describe common use, not an absolute technical prohibition.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

> Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Question sources

Computer Science Exit Exam · 2018 · Question 51

Question 48 | Q00966

Which identifier is invalid under the basic ASCII C++ identifier rules? Treat the stray leading dot before jan2025 as an OCR artifact.

A. delete

B. for_cpp

C. jan2025

D. Hello

Hint

Apply the naming, type and syntax rules before deciding whether an option can be used as stated.

Answer A: A. delete

An ordinary identifier starts with a letter or underscore and continues with letters, digits or underscores; it must also avoid keywords. delete fails because it is a C++ keyword. for_cpp is a single identifier distinct from the keyword for, while jan2025 and Hello follow the character rules. Removing the documented stray dot makes the intended identifier comparison explicit without changing the keyword concept.

Why each choice is right or wrong

A - Correct: delete is a C++ keyword and cannot be an ordinary identifier.

B - Incorrect: for_cpp contains valid identifier characters and is not the keyword for.

C - Incorrect: jan2025 starts with a letter and continues with allowed letters and digits.

D - Incorrect: Hello contains only letters and is not a keyword.

Primary note proof

Study-note section CP05

> In the basic ASCII identifier rules, start with a letter or underscore; continue with letters, digits or underscores. Do not use a keyword. C++ is case sensitive: exitExam and ExitExam differ; delete and float are keywords, but FLOAT is a legal identifier. variable_1234 is valid; 4thInput, T-3 and names containing spaces are not single identifiers. Avoid reserved underscore patterns in application code.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 2-10

> 1. Keywords (reserved words) - Unique meaning within a C++ program and must not be used for any other purposes. E.g., if, namespace for, switch, continue, float, new, default, break Etc. 2. Identifiers - Name associated with a function or data object and used to refer to that function or data object. An identifier must: o Start with a letter or underscore o Consist only of letters, the digits 0-9, or the underscore symbol _ o Not be a reserved word

Question sources

Computer Science Exit Exam · 2018 · Question 79

Question 49 | Q00989

Which C++ declarations denote the same signed short integer type as short n;? Select all that apply.

A. int short n;

B. unsigned short n;

C. float short n;

D. short n;

Hint

Apply the naming, type and syntax rules before deciding whether an option can be used as stated.

Answer A, D: A. int short n;; D. short n;

The ordinary short type is signed unless unsigned is explicitly requested. The note identifies short n, short int n and int short n as equivalent spellings. Consequently, A is equivalent despite its reordered specifiers, and D is equivalent because it repeats the original declaration. B changes signedness, while C combines incompatible floating and integer type specifiers rather than naming the requested type.

Why each choice is right or wrong

A - Correct: int short is an equivalent spelling of signed short.

B - Incorrect: unsigned short is a different type with different signedness.

C - Incorrect: float short is not a valid combination of these type specifiers.

D - Correct: This repeats the original signed short declaration.

Primary note proof

Study-note section CP06

> Use int for whole numbers, float/double for fractional approximations, char for a character and bool for logical values in C++. Type sizes depend on the implementation; do not import Java widths into C++. short n, short int n and int short n denote the same signed short type. const int n=5 and int const n=5 are equivalent; an ordinary const int needs initialization. typedef creates an alias, not a distinct new type.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 24-33

> 5. Data Types, Variables, and Constants Variable: - Variable is a symbolic name for a memory location in which data can be stored. - Attribute of Variables: o A type, which is, established

when the variable is defined (e.g., integer, float, character) o A value, which can be changed by assigning a new value to the variable Variable Declaration - Declaring a variable means defining (creating) a variable. - Syntax: data type variable name; e.g. int num; - We can create More Than One Variable at a Time e.g., int num1, num2, num3;

Question sources

National Exit Exam practice questions · Question 3

Computer Science Exit Exam practice collection · Question 82

Question 50 | Q00990

Which one of the following is NOT a high-level language?

A. C++

B. Java

C. Python

D. Assembly

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: D. Assembly

Machine and assembly languages are low-level because their instructions are tied to an instruction set. C++ and Java are general high-level languages in the note's conventional classification, and Python is also a high-level language even though it is commonly used for scripting. Assembly is therefore the exception requested here. Classification by language level is separate from whether an implementation compiles or interprets a program.

Why each choice is right or wrong

A - Incorrect: C++ is a general high-level language, so it is not the requested exception.

B - Incorrect: Java is a general high-level language, so it is not the requested exception.

C - Incorrect: Python is high-level; scripting usage does not make it assembly or machine code.

D - Correct: Assembly is an instruction-set-dependent low-level language.

Primary note proof

Study-note section CP03

Machine instructions are conventionally first generation; symbolic assembly is second; general high-level languages such as C++ and Java are third. Database query and application generators such as SQL are conventionally fourth generation. Fifth generation is a loose historical category associated with declarative logic, constraints and AI/expert systems. Generations do not determine whether an implementation uses an interpreter.

Study-note section CP-S02

Python is a high-level programming language with built-in high-level data types. Calling a language a scripting language describes a common way it is used; it does not make that language machine code or assembly. Language level and execution style are different classifications.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Question sources

National Exit Exam practice questions · Question 4

Computer Science Exit Exam practice collection · Question 83

Question 51 | Q00998

What does this C++ fragment print? Assume cout is available.
int x=15,y=2;
int z=x/y;
cout<<z;

A. 7.5

B. 7

C. 3

D. 12.5

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: B. 7

Both division operands are int, so their quotient is computed using integer division. Fifteen divided by two truncates toward zero to 7, which is stored in z and printed. The lowercase identifier matches the declared variable. A fractional quotient would require conversion before division, and none occurs here.

Why each choice is right or wrong

A - Incorrect: The fractional mathematical quotient is discarded by integer division.

B - Correct: The int quotient is 7 and is printed from z.

C - Incorrect: 3 is not the quotient of the given operands.

D - Incorrect: 12.5 follows neither integer division nor the mathematical quotient.

Primary note proof

Study-note section CP08

> Multiplication, division and remainder have higher precedence than addition and subtraction; equal-precedence arithmetic groups left to right. Integer division truncates toward zero. For integers, a == (a/b)*b + a%b when the division is valid. % requires integral operands in C++; (8.0*3)%5 is invalid. Division by zero and signed integer overflow cannot be given ordinary numeric outputs.

Study-note section CP34

> Printing Z when only z is declared refers to a different, undeclared identifier.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, | |) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Computer Programming module | PDF page 6 | lines 4-15

> 6. Communication through console cout (the standard output stream) o used to display information for the user Syntax: cout<<object; • Object can be variable or literals Example: int x=5; cout<<x cout<< “Hello there”; cin(the standard input stream): o used to accept information from the user. Syntax: cin>>variable name; Eg. cin>>x;

Question sources

National Exit Exam practice questions · Question 12

Question 52 | Q01001

Which generation of programming languages stated for interpreted languages such as SQL?

A. 4th generation

B. 2nd generation

C. 3rd generation

D. All of the above

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: SQL is conventionally fourth generation.

CP03 explicitly classifies SQL with fourth-generation query languages and application generators. The question's description of interpreted languages is imprecise: interpretation is not what determines a language generation. Nevertheless, the named example SQL has a clear conventional classification, making fourth generation the supported answer.

Why each choice is right or wrong

A - Correct: Fourth generation is the note's explicit conventional classification for SQL and query languages.

B - Incorrect: Second generation refers to assembly; interpretation alone does not move SQL into this category.

C - Incorrect: Third generation describes general high-level languages, rather than SQL's stated query-language category.

D - Incorrect: SQL is not classified into all these generations merely because implementations can interpret it.

Primary note proof

Study-note section CP03

> Machine instructions are conventionally first generation; symbolic assembly is second; general high-level languages such as C++ and Java are third. Database query and application generators such as SQL are conventionally fourth generation. Fifth generation is a loose historical category associated with declarative logic, constraints and AI/expert systems. Generations do not determine whether an implementation uses an interpreter.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

> Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of

computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Question sources

National Exit Exam practice questions · Question 15

Computer Science Exit Exam practice collection · Question 85

Question 53 | Q01130

A variable name can only contain _____?

A. alphanumeric characters

B. underscores

C. Both A and B

D. None of the above

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: Both A and B

The permitted identifier characters are letters, digits and underscore (p.5), so both A and B describe allowed contents and C combines them. This is not permission to begin with a digit or to use a reserved word: those separate restrictions still apply. Module support: PDF p. 5.

Why each choice is right or wrong

A - Incorrect: "alphanumeric characters" does not match the rule or result derived here. The permitted identifier characters are letters, digits and underscore (p.5), so both A and B describe allowed contents and C combines them. This is not permission to begin with a digit or to use a reserved word: those separate restrictions still apply.

B - Incorrect: "underscores" does not match the rule or result derived here. The permitted identifier characters are letters, digits and underscore (p.5), so both A and B describe allowed contents and C combines them. This is not permission to begin with a digit or to use a reserved word: those separate restrictions still apply.

C - Correct: "Both A and B" matches the derived result. The permitted identifier characters are letters, digits and underscore (p.5), so both A and B describe allowed contents and C combines them. This is not permission to begin with a digit or to use a reserved word: those separate restrictions still apply.

D - Incorrect: "None of the above" does not match the rule or result derived here. The permitted identifier characters are letters, digits and underscore (p.5), so both A and B describe allowed contents and C combines them. This is not permission to begin with a digit or to use a reserved word: those separate restrictions still apply.

Primary note proof

Study-note section CP05

In the basic ASCII identifier rules, start with a letter or underscore; continue with letters, digits or underscores. Do not use a keyword. C++ is case sensitive: exitExam and ExitExam differ; delete and float are keywords, but FLOAT is a legal identifier. variable_1234 is valid; 4thInput, T-3 and names containing spaces are not single identifiers. Avoid reserved underscore patterns in application code.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 2-10

1. Keywords (reserved words) - Unique meaning within a C++ program and must not be used for any other purposes. E.g., if, namespace for, switch, continue, float, new, default, break Etc. 2. Identifiers - Name associated with a function or data object and used to refer to that function or data object. An identifier must: o Start with a letter or underscore o Consist only of letters, the digits 0-9, or the underscore symbol _ o Not be a reserved word

Question sources

Web Programming practice questions · Question 63

Question 54 | Q01140

What does "cout" represent?

A. Output statement

B. Input statement

C. Compiler

D. None of the above

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: Output statement

The module names cout the standard output stream and shows cout<<object displaying information. Output statement is the closest offered description; cout itself is a stream object. cin handles input, while a compiler translates source. Module support: PDF p. 6.

Why each choice is right or wrong

A - Correct: "Output statement" matches the derived result. The module names cout the standard output stream and shows cout<<object displaying information. Output statement is the closest offered description; cout itself is a stream object. cin handles input, while a compiler translates source.

B - Incorrect: "Input statement" does not match the rule or result derived here. The module names cout the standard output stream and shows cout<<object displaying information. Output statement is the closest offered description; cout itself is a stream object. cin handles input, while a compiler translates source.

C - Incorrect: "Compiler" does not match the rule or result derived here. The module names cout the standard output stream and shows cout<<object displaying information. Output statement is the closest offered description; cout itself is a stream object. cin handles input, while a compiler translates source.

D - Incorrect: "None of the above" does not match the rule or result derived here. The module names cout the standard output stream and shows cout<<object displaying information. Output statement is the closest offered description; cout itself is a stream object. cin handles input, while a compiler translates source.

Primary note proof

Study-note section CP28

> Include <iostream> and qualify std::cin/std::cout or introduce the names explicitly. cout is the standard output stream; << inserts values. cin is standard input; >> extracts formatted values. std::endl inserts a newline and flushes. C++ names are case sensitive: Cout, Int and std: :cout are not equivalent valid spellings of cout, int and std::cout.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 6 | lines 4-15

> 6. Communication through console cout (the standard output stream) o used to display information for the user Syntax: cout<<object; • Object can be variable or literals Example: int x=5; cout<<x cout<< “Hello there”; cin(the standard input stream): o used to accept information from the user. Syntax: cin>>variable name; Eg. cin>>x;

Question sources

Computer Science Exit Exam practice questions · Question 1

Question 55 | Q01141

Which terms can denote a defect in a program, rather than a debugging tool or the act of correcting defects? Select all that apply.

A. Bug

B. Debugger

C. Error

D. Debug

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A, C: A. Bug; C. Error

A bug is a defect in a program, and error can also name such a defect in this question's terminology. Both nouns therefore fit the requested meaning. A debugger instead names a tool used to investigate execution, and debug names the activity of locating or correcting defects.

Why each choice is right or wrong

A - Correct: Bug denotes a program defect.

B - Incorrect: A debugger is a tool for investigating defects.

C - Correct: Error can denote the defect itself.

D - Incorrect: Debug describes the correction/investigation activity rather than the defect.

Primary note proof

Study-note section CP32

> A syntax error violates grammar; a type error can be syntactically well formed but rejected during semantic checking. A runtime error occurs during execution; a logic error gives the wrong behavior. A bug is a defect; debugging locates and corrects it. Read the first useful compiler diagnostic, inspect nearby tokens and declarations, repair and rebuild; a debugger is useful after a runnable program exists.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 2 | lines 2-18

> 1. Introduction to programming Computer programming is the process of writing, testing, debugging/troubleshooting, and maintaining the source code of computer programs. Programs are the instructions that tells the computer what to do .so, writing computer programs means writing instructions, that will make the computer follow and run a program based on those instructions. We are going to write computer programs to solve problems. Problem Solving Problem: gap between the existing and the desired or proposed situation where problem solving will try to fill this gap. Problem solving: The process of transforming the description of a problem into the solution by using our knowledge of the problem-solving strategies, techniques and tools. Problem solving techniques • Analyze the problem: Carefully understand the problem • Outline the problem requirements • Does program require user interaction? • Does program manipulate data? • What is the output? • Design steps (algorithm) to solve the problem

Question sources

Computer Science Exit Exam practice questions · Question 2

Question 56 | Q01142

is a language translators required to translate the assembly languages into machine code.

A. | a | Compiler x

B. ? | Assembler 4

C. |@| Interpreter x

D. | None of the above x

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: Assembler.

Compilation may emit assembly, and the assembler emits machine instructions in an object file. The translator named by the question is the assembler.

Why each choice is right or wrong

A - Incorrect: The compiler can produce assembly from higher-level source; that is the preceding stage in CP04.

B - Correct: The assembler translates assembly instructions into machine instructions contained in an object file.

C - Incorrect: An interpreter executes a program representation instead of performing the stated assembly-to-object translation.

D - Incorrect: Assembler is present and directly matches the translator role, so none of the above is false.

Primary note proof

Study-note section CP04

> In the conventional C++ toolchain: source → preprocessing → compilation proper → assembly → object files → linking → executable → loading → execution. Preprocessing handles directives such as #include; compilation may emit assembly; the assembler emits machine instructions in an object file. The linker resolves external symbols and combines object files and libraries. The loader places an executable and required components in memory and starts it.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

> Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before

execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Computer Programming module | PDF page 4 | lines 21-30

The compilation process of C++ program consists of the following steps. • edit source code → compile → link → execute (Editor) (Compiler) (Linker) (Loader) I. Program source code is entered into a file using a text editor. II. After the code has been entered, a compiler program is started that translates the source into an object code file. III. The object code file is linked with other object code files that come with the compiler and an executable file (or program) is created. IV. In order to execute the program, a program called the loader copies the executable file into the memory of the computer and sends an execute command to the CPU.

Question sources

Computer Science Exit Exam practice questions · Question 3

Question 57 | Q01144

An error in the statement in a computer program that violates the rules of the programming.
a | Logical error x

A. ? | Syntax error ov

B. '@| Semantic error x

C. | All of the above x

D. 6 - True or false

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: ? | Syntax error ov

Syntax is the set of rules governing use of language symbols, while semantics concerns the meanings of syntactically correct constructs (p.3). Violating the written language rules is therefore a syntax error. Module support: PDF p. 3.

Why each choice is right or wrong

A - Correct: Syntax error matches violation of the language rules defined on p.3.

B - Incorrect: Semantic error concerns meaning rather than the syntactic rule violation described (p.3).

C - Incorrect: All would include semantic/logical alternatives, which are not equivalent to violating syntax.

D - Incorrect: This is an unrelated extracted True/false heading, not an error category.

Primary note proof

Study-note section CP32

A syntax error violates grammar; a type error can be syntactically well formed but rejected during semantic checking. A runtime error occurs during execution; a logic error gives the wrong behavior. A bug is a defect; debugging locates and corrects it. Read the first useful compiler diagnostic, inspect nearby tokens and declarations, repair and rebuild; a debugger is useful after a runnable program exists.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Question sources

Computer Science Exit Exam practice questions · Question 5

Question 58 | Q01145

Keywords are words whose meanings are defined to the compiler.

A. True

B. False

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: True

The module calls keywords reserved words with a unique meaning in a C++ program and forbids other uses (p.5). This supports True; False contradicts their predefined role. Module support: PDF p. 5.

Why each choice is right or wrong

A - Correct: "True" matches the derived result. The module calls keywords reserved words with a unique meaning in a C++ program and forbids other uses (p.5). This supports True; False contradicts their predefined role.

B - Incorrect: "False" does not match the rule or result derived here. The module calls keywords reserved words with a unique meaning in a C++ program and forbids other uses (p.5). This supports True; False contradicts their predefined role.

Primary note proof

Study-note section CP-S03

> A keyword is a reserved word with a predefined meaning in the language grammar. It cannot be repurposed as an ordinary variable or function name. An identifier is a name used to refer to a data object or function; its spelling must obey the identifier-character rules and avoid keywords.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 2-10

> 1. Keywords (reserved words) - Unique meaning within a C++ program and must not be used for any other purposes. E.g., if, namespace for, switch, continue, float, new, default, break Etc. 2. Identifiers - Name associated with a function or data object and used to refer to that function or data object. An identifier must: o Start with a letter or underscore o Consist only of letters, the digits 0-9, or the underscore symbol _ o Not be a reserved word

Question sources

Computer Science Exit Exam practice questions · Question 6

Question 59 | Q01146

In which constant that can be written in a single quotes?

A. | a | String constant x

B. i? Character constant vA

C. /@| All of the above x

D. | None of the above x

Hint

Apply the naming, type and syntax rules before deciding whether an option can be used as stated.

Answer B: i? Character constant vA

The module examples use single quotes for a character such as A and double quotes for a string such as Hello There (pp.8,22). Thus Character constant is the correct category. String constant uses double quotes, so All and None fail. Module support: PDF p. 8, PDF p. 22.

Why each choice is right or wrong

A - Incorrect: "| a | String constant x" does not match the rule or result derived here. The module examples use single quotes for a character such as A and double quotes for a string such as Hello There (pp.8,22). Thus Character constant is the correct category. String constant uses double quotes, so All and None fail.

B - Correct: "i? Character constant vA" matches the derived result. The module examples use single quotes for a character such as A and double quotes for a string such as Hello There (pp.8,22). Thus Character constant is the correct category. String constant uses double quotes, so All and None fail.

C - Incorrect: "/@| All of the above x" does not match the rule or result derived here. The module examples use single quotes for a character such as A and double quotes for a string such as Hello There (pp.8,22). Thus Character constant is the correct category. String constant uses double quotes, so All and None fail.

D - Incorrect: "| None of the above x" does not match the rule or result derived here. The module examples use single quotes for a character such as A and double quotes for a string such as Hello There (pp.8,22). Thus Character constant is the correct category. String constant uses double quotes, so All and None fail.

Primary note proof

Study-note section CP07

> Single quotes denote a character literal, such as 'A'; double quotes denote a string literal, such as "A". A string includes more storage than just its visible characters when a terminator is required. sizeof reports size in bytes, with sizeof(char)==1; the byte width and other sizes are implementation dependent. sizeof(array) gives total array storage before conversion to a pointer.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 8 | lines 27-33

> Constant: A constant is any expression that has a fixed value like integer, float, character and string Eg. 120, 15.2, „A‟, “Hello There” Generally, we have two ways to define constant: 1. Using Const Key word Syntax: const type constant name=value; e.g. const float PI=3.14; 2. Using #define Keyword Syntax: #define constant name value e.g. #define PI 3.14

Computer Programming module | PDF page 22 | lines 22-38

> String and pointer String: - is an ordered sequence of characters - Sequence of character in which the last character is the null character „\0‟ - The null character indicates the end of the string - C++ provides following two types of string representations: o The C-style character string. o The string class type introduced with Standard C++. NB: →If a string has n characters, then it requires an n+1 element array (at least) to store it. →In order to use the string data type, the C++ string header <string> must be included at the top of the program The C-Style Character String - originated within the C language and continues to be supported within C++ - array of characters which is terminated by a null character '\0' Example: char status[5]={„G‟,‟O‟,‟O‟,‟D‟,‟\0‟};//char status[]=”Good”; cout<<status;

Question sources

Computer Science Exit Exam practice questions · Question 7

Question 60 | Q01147

In which constant that can be written in a double quotes?

A. a | String constants oS

B. i? Character constants x

C. |@| All of the above x

D. | None of the above x

Hint

Apply the naming, type and syntax rules before deciding whether an option can be used as stated.

Answer A: a | String constants oS

The examples enclose strings in double quotes and characters in single quotes (pp.8,22). Therefore String constants is the intended category; Character, All and None do not fit the shown notation. Module support: PDF p. 8, PDF p. 22.

Why each choice is right or wrong

A - Correct: "a | String constants oS" matches the derived result. The examples enclose strings in double quotes and characters in single quotes (pp.8,22). Therefore String constants is the intended category; Character, All and None do not fit the shown notation.

B - Incorrect: "i? Character constants x" does not match the rule or result derived here. The examples enclose strings in double quotes and characters in single quotes (pp.8,22). Therefore String constants is the intended category; Character, All and None do not fit the shown notation.

C - Incorrect: "|@| All of the above x" does not match the rule or result derived here. The examples enclose strings in double quotes and characters in single quotes (pp.8,22). Therefore String constants is the intended category; Character, All and None do not fit the shown notation.

D - Incorrect: "| None of the above x" does not match the rule or result derived here. The examples enclose strings in double quotes and characters in single quotes (pp.8,22). Therefore String constants is the intended category; Character, All and None do not fit the shown notation.

Primary note proof

Study-note section CP07

Single quotes denote a character literal, such as 'A'; double quotes denote a string literal, such as "A". A string includes more storage than just its visible characters when a terminator is required. sizeof reports size in bytes, with sizeof(char)==1; the byte width and other sizes are implementation dependent. sizeof(array) gives total array storage before conversion to a pointer.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 8 | lines 27-33

Constant: A constant is any expression that has a fixed value like integer, float, character and string Eg. 120, 15.2, „A‟, “Hello There” Generally, we have two ways to define constant: 1. Using Const Key word Syntax: const type constant name=value; e.g. const float PI=3.14; 2. Using #define Keyword Syntax: #define constant name value e.g. #define PI 3.14

Computer Programming module | PDF page 22 | lines 22-38

String and pointer String: - is an ordered sequence of characters - Sequence of character in which the last character is the null character „\0‟ - The null character indicates the end of the string - C++ provides following two types of string representations: o The C-style character string. o The string class type introduced with Standard C++. NB: →If a string has n characters, then it requires an n+1 element array (at least) to store it. →In order to use the string data type, the C++ string header <string> must be included at the top of the program The C-Style Character String - originated within the C language and continues to be supported within C++ - array of characters which is terminated by a null character '\0' Example: char status[5]={„G‟,‟O‟,‟O‟,‟D‟,‟\0‟};//char status[]=”Good”; cout<<status;

Question sources

Computer Science Exit Exam practice questions · Question 8

Question 61 | Q01148

What is the output of the following C++ fragment?

```
#include <iostream>
using namespace std;
int main()
{
int a = 8;
int b = 5;
int c;
c = a++ + --b;
cout << c << endl;
return 0;
}
```

A. 13

B. 15

C. 11

D. 12

Hint

Fix the input case and cost model, then count dominant work or simultaneously live storage.

Answer D: 12

Postfix a++ contributes 8 to the expression and then makes a=9. Prefix --b changes 5 to 4 before use. Thus c=8+4=12. Thirteen uses old b; the other sums do not follow the two operator timings. Module support: PDF p. 7.

Why each choice is right or wrong

A - Incorrect: "13" does not match the rule or result derived here. Postfix a++ contributes 8 to the expression and then makes a=9. Prefix --b changes 5 to 4 before use. Thus c=8+4=12. Thirteen uses old b; the other sums do not follow the two operator timings.

B - Incorrect: "15" does not match the rule or result derived here. Postfix a++ contributes 8 to the expression and then makes a=9. Prefix --b changes 5 to 4 before use. Thus c=8+4=12. Thirteen uses old b; the other sums do not follow the two operator timings.

C - Incorrect: "11" does not match the rule or result derived here. Postfix a++ contributes 8 to the expression and then makes a=9. Prefix --b changes 5 to 4 before use. Thus c=8+4=12. Thirteen uses old b; the other sums do not follow the two operator timings.

D - Correct: "12" matches the derived result. Postfix a++ contributes 8 to the expression and then makes a=9. Prefix --b changes 5 to 4 before use. Thus c=8+4=12. Thirteen uses old b; the other sums do not follow the two operator timings.

Primary note proof

Study-note section CP09

Prefix ++x changes x and yields the new value; postfix x++ yields the previous value and also increments x. Separate the value used in the expression from the stored value after it. Starting a=8,b=5, a++ + --b uses 8+4, gives 12, and leaves a=9,b=4. Starting x=1,y=10,z=-3, (x++*y--)+ ++z gives 8 and leaves x=2,y=9,z=-2.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Computer Science Exit Exam practice questions · Question 9

Question 62 | Q01149

Which data type represents logical values?

A. Integer

B. Boolean

C. Double

D. Float

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: Boolean.

CP06 assigns bool to logical values, whereas int represents whole numbers and float/double represent fractional approximations. Boolean is therefore the intended type category. Numeric values can participate in truth conversion, but that does not make their numeric types the dedicated representation of logical true/false values.

Why each choice is right or wrong

A - Incorrect: Integer represents whole numbers; possible conversion to truth does not make it the logical type.

B - Correct: Boolean represents logical true/false values, matching bool in the note's C++ type mapping.

C - Incorrect: Double represents fractional numeric approximations rather than the dedicated logical type.

D - Incorrect: Float represents fractional numeric approximations rather than the dedicated logical type.

Primary note proof

Study-note section CP12

> A standalone if may execute independently of the following if. An else binds to its nearest unmatched if; an else-if ladder executes the first true branch only. In nested if(c1){if(c2) S1; else S2;}, S2 requires c1 true and c2 false. Braces determine grouping.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Computer Programming module | PDF page 5 | lines 24-33

> 5. Data Types, Variables, and Constants Variable: - Variable is a symbolic name for a memory location in which data can be stored. - Attribute of Variables: o A type, which is, established when the variable is defined (e.g., integer, float, character) o A value, which can be changed by assigning a new value to the variable Variable Declaration - Declaring a variable means defining (creating) a variable. - Syntax: data type variable name; e.g. int num; - We can create More Than One Variable at a Time e.g., int num1, num2, num3;

Question sources

Computer Science Exit Exam practice questions · Question 10

Question 63 | Q01150

What is the output of the following C++ fragment?

```
#include <iostream>
using namespace std;
int main()
{
int x = 25;
if (x < 20)
cout << "Low Quality" << endl;
else
cout << "High Quality" << endl;
return 0;
}
```

A. Low Quality

B. High Quality

C. No output

D. None of the above

Hint

Fix the input case and cost model, then count dominant work or simultaneously live storage.

Answer B: High Quality

The condition 25<20 is false, so the else branch prints High Quality. The then message is skipped; a defined else branch means the no-output alternative is false, and B supplies a matching option. Module support: PDF p. 9.

Why each choice is right or wrong

A - Incorrect: "Low Quality" does not match the rule or result derived here. The condition 25<20 is false, so the else branch prints High Quality. The then message is skipped; a defined else branch means the no-output alternative is false, and B supplies a matching option.

B - Correct: "High Quality" matches the derived result. The condition 25<20 is false, so the else branch prints High Quality. The then message is skipped; a defined else branch means the no-output alternative is false, and B supplies a matching option.

C - Incorrect: "No output" does not match the rule or result derived here. The condition 25<20 is false, so the else branch prints High Quality. The then message is skipped; a defined else branch means the no-output alternative is false, and B supplies a matching option.

D - Incorrect: "None of the above" does not match the rule or result derived here. The condition 25<20 is false, so the else branch prints High Quality. The then message is skipped; a defined else branch means the no-output alternative is false, and B supplies a matching option.

Primary note proof

Study-note section CP06

Use int for whole numbers, float/double for fractional approximations, char for a character and bool for logical values in C++. Type sizes depend on the implementation; do not import Java widths into C++. short n, short int n and int short n denote the same signed short type. const int n=5 and int const n=5 are equivalent; an ordinary const int needs initialization. typedef creates an alias, not a distinct new type.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 9 | lines 9-21

Conditional Statements A. The if Statement Syntax: if (expression) statement; First expression is evaluated. If the outcome is nonzero (true) then statement is executed. Otherwise, nothing happens. if-else statement Syntax: if (expression) statement1; else statement2; First expression is evaluated. If the outcome is nonzero (true) then statement1 is executed. Otherwise, statement2 is executed.

Question sources

Computer Science Exit Exam practice questions · Question 11

Question 64 | Q01166

Loading executable code, which is the output of a linker, into main memory is called:

A. Linker

B. Assembler

C. Compiler

D. Loader

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer D: Loader

The module says the loader copies the linked executable into computer memory and sends execution to the CPU (p.4). Linking builds that executable; compilation translates source, so they do not name the memory-loading role. Module support: PDF p. 4.

Why each choice is right or wrong

A - Incorrect: "Linker" does not match the rule or result derived here. The module says the loader copies the linked executable into computer memory and sends execution to the CPU (p.4). Linking builds that executable; compilation translates source, so they do not name the memory-loading role.

B - Incorrect: "Assembler" does not match the rule or result derived here. The module says the loader copies the linked executable into computer memory and sends execution to the CPU (p.4).

Linking builds that executable; compilation translates source, so they do not name the memory-loading role.

C - Incorrect: "Compiler" does not match the rule or result derived here. The module says the loader copies the linked executable into computer memory and sends execution to the CPU (p.4). Linking builds that executable; compilation translates source, so they do not name the memory-loading role.

D - Correct: "Loader" matches the derived result. The module says the loader copies the linked executable into computer memory and sends execution to the CPU (p.4). Linking builds that executable; compilation translates source, so they do not name the memory-loading role.

Primary note proof

Study-note section CP04

> In the conventional C++ toolchain: source → preprocessing → compilation proper → assembly → object files → linking → executable → loading → execution. Preprocessing handles directives such as #include; compilation may emit assembly; the assembler emits machine instructions in an object file. The linker resolves external symbols and combines object files and libraries. The loader places an executable and required components in memory and starts it.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 4 | lines 21-30

> The compilation process of C++ program consists of the following steps. • edit source code → compile → link → execute (Editor) (Compiler) (Linker) (Loader) I. Program source code is entered into a file using a text editor. II. After the code has been entered, a compiler program is started that translates the source into an object code file. III. The object code file is linked with other object code files that come with the compiler and an executable file (or program) is created. IV. In order to execute the program, a program called the loader copies the executable file into the memory of the computer and sends an execute command to the CPU.

Question sources

Computer Science Exit Exam practice questions · Question 27

Question 65 | Q01174

When control reaches a loop normally, which loop form executes its body before its first condition test, guaranteeing at least one body execution?

A. while loop

B. for loop

C. nested loop

D. do-while loop

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: D. do-while loop

A do-while loop performs its body first and tests the continuation condition afterward. Even when that first test is false, one body execution has already occurred. A while loop and a conventional for loop can reject entry on their first condition test. Nesting describes one loop inside another and supplies no independent guarantee that either body runs. The option names the post-test form.

Why each choice is right or wrong

A - Incorrect: while checks its condition before entering the body.

B - Incorrect: for can fail its initial condition and execute zero times.

C - Incorrect: A nested loop can have an outer condition that is false initially.

D - Correct: do-while performs its body before the first condition test.

Primary note proof

Study-note section CP14

> while(condition) checks before each body execution and may run zero times. do {body;} while(condition); runs the body once before checking; it repeats while the condition remains true. Use it when a menu or input attempt must occur at least once.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 12 | lines 2-7

> The do-while loop The condition in the do-while is evaluated after the execution of statement instead of before, granting at least one execution of statement even if condition is never fulfilled. do {statement; }while (condition); First statement is executed and then condition is evaluated. If the outcome of the latter is nonzero then the whole process is repeated. Otherwise, the loop is terminated.

Computer Programming module | PDF page 11 | lines 5-13

> Iteration/Looping Statement: Iteration is the repetition of a statement or block of statements in a program. - Loops have as objective to repeat a statement a certain number of times or while a condition is fulfilled. - C++ has three iteration statements: the while statement, the do. While statement, and the for statement. The while loop: Its format is: while (expression) {statement;} First expression (called the loop condition) is evaluated. If the outcome is nonzero then statement (called the loop body) is executed and the whole process is repeated. Otherwise, the loop is terminated.

Question sources

Computer Science Exit Exam practice questions · Question 35

Question 66 | Q01196

The _____ statement causes a loop to terminate immediately.

A. Continue

B. Goto

C. Break

D. Jump

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: Break

break immediately leaves the loop for the statement outside it (p.13). continue skips the remainder of the current iteration but keeps looping. Goto names a separate label transfer, and Jump is not the module statement that directly terminates a loop. Module support: PDF p. 13.

Why each choice is right or wrong

A - Incorrect: "Continue" does not match the rule or result derived here. break immediately leaves the loop for the statement outside it (p.13). continue skips the remainder of the current iteration but keeps looping. Goto names a separate label transfer, and Jump is not the module statement that directly terminates a loop.

B - Incorrect: "Goto" does not match the rule or result derived here. break immediately leaves the loop for the statement outside it (p.13). continue skips the remainder of the current iteration but keeps looping. Goto names a separate label transfer, and Jump is not the module statement that directly terminates a loop.

C - Correct: "Break" matches the derived result. break immediately leaves the loop for the statement outside it (p.13). continue skips the remainder of the current iteration but keeps looping. Goto names a separate label transfer, and Jump is not the module statement that directly terminates a loop.

D - Incorrect: "Jump" does not match the rule or result derived here. break immediately leaves the loop for the statement outside it (p.13). continue skips the remainder of the current iteration but keeps looping. Goto names a separate label transfer, and Jump is not the module statement that directly terminates a loop.

Primary note proof

Study-note section CP15

> break exits the nearest enclosing loop or switch. continue skips the rest of the current loop iteration: a for loop then runs its update; a while loop goes directly to its condition. return leaves the current function and may supply a result. In main it ends the program normally.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 13 | lines 30-35

> Using continue and break Statements - The break statement skips the rest of the statements in the loop‟s block, jumping immediately to the next statement outside of the loop. - The continue statement is similar. It also skips the rest of the statements in the loop‟s block, but instead of terminating the loop, it transfers execution to the next iteration of the loop. It continues the loop after skipping the remaining statements in its current iteration.

Question sources

Computer Science model exam 2 · Question 17

Question 67 | Q01214

A translator that translates a single-line instruction at a time is:

A. Compiler

B. Assembler

C. Interpreter

D. Linker

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: Interpreter

The module explicitly contrasts whole-program compilation with an interpreter translating one line at a time across pp.3-4. Therefore Interpreter fits; linking combines object files, and the other named translators are not the one-line interpreter role. Module support: PDF p. 3.

Why each choice is right or wrong

A - Incorrect: "Compiler" does not match the rule or result derived here. The module explicitly contrasts whole-program compilation with an interpreter translating one line at a time across pp.3-4. Therefore Interpreter fits; linking combines object files, and the other named translators are not the one-line interpreter role.

B - Incorrect: "Assembler" does not match the rule or result derived here. The module explicitly contrasts whole-program compilation with an interpreter translating one line at a time across pp.3-4. Therefore Interpreter fits; linking combines object files, and the other named translators are not the one-line interpreter role.

C - Correct: "Interpreter" matches the derived result. The module explicitly contrasts whole-program compilation with an interpreter translating one line at a time across pp.3-4. Therefore Interpreter fits; linking combines object files, and the other named translators are not the one-line interpreter role.

D - Incorrect: "Linker" does not match the rule or result derived here. The module explicitly contrasts whole-program compilation with an interpreter translating one line at a time across pp.3-4. Therefore Interpreter fits; linking combines object files, and the other named translators are not the one-line interpreter role.

Primary note proof

Study-note section CP04

> GCC documents the first four stages explicitly. A compiler need not emit a complete executable or even native machine code. Java javac produces bytecode, which a JVM interprets or compiles at runtime. Interpreters execute a program representation; “one source line at a time” is a beginner approximation. Compilation uses multiple passes, so “whole program in one step” is not literally universal.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

> Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Question sources

Computer Science model exam 2 · Question 35

Question 68 | Q01231

Consider the following outline of a nested if-else structure:

```
if (condition1)
{
if (condition2)
statement1;
else
statement2;
}
```

Which of the statements below is true regarding this structure?

A. Syntactically, it is invalid to have more if clauses than else clauses.

B. Statement2 will only execute if condition1 is false and condition2 is false.

C. Statement2 will only execute if condition1 is true and condition2 is false.

D. Statement2 will only execute if condition1 is false, it does not matter what condition2 is.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: Statement2 will only execute if condition1 is true and condition2 is false.

The outer body is entered only when condition1 is true. Inside that body, condition2 false selects its else and statement2. If condition1 is false neither nested branch executes. An if may omit else, so A imposes an unsupported syntactic requirement. Module support: PDF p. 9.

Why each choice is right or wrong

A - Incorrect: "Syntactically, it is invalid to have more if clauses than else clauses." does not match the rule or result derived here. The outer body is entered only when condition1 is true. Inside that body, condition2 false selects its else and statement2. If condition1 is false neither nested branch executes. An if may omit else, so A imposes an unsupported syntactic requirement.

B - Incorrect: "Statement2 will only execute if condition1 is false and condition2 is false." does not match the rule or result derived here. The outer body is entered only when condition1 is true. Inside that body, condition2 false selects its else and statement2. If condition1 is false neither nested branch executes. An if may omit else, so A imposes an unsupported syntactic requirement.

C - Correct: "Statement2 will only execute if condition1 is true and condition2 is false." matches the derived result. The outer body is entered only when condition1 is true. Inside that body, condition2 false selects its else and statement2. If condition1 is false neither nested branch executes. An if may omit else, so A imposes an unsupported syntactic requirement.

D - Incorrect: "Statement2 will only execute if condition1 is false, it does not matter what condition2 is." does not match the rule or result derived here. The outer body is entered only when condition1 is true. Inside that body, condition2 false selects its else and statement2. If condition1 is false neither nested branch executes. An if may omit else, so A imposes an unsupported syntactic requirement.

Primary note proof

Study-note section CP12

> A standalone if may execute independently of the following if. An else binds to its nearest unmatched if; an else-if ladder executes the first true branch only. In nested if(c1){if(c2) S1; else S2;}, S2 requires c1 true and c2 false. Braces determine grouping.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 9 | lines 9-21

> Conditional Statements A. The if Statement Syntax: if (expression) statement; First expression is evaluated. If the outcome is nonzero (true) then statement is executed. Otherwise, nothing happens. if-else statement Syntax: if (expression) statement1; else statement2; First expression is evaluated. If the outcome is nonzero (true) then statement1 is executed. Otherwise, statement2 is executed.

Question sources

Computer Science model exam 2 · Question 52

Question 69 | Q01256

Which of the following is a valid variable name?

A. T-3

B. GGGGG

C. float

D. 4thInput

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: GGGGG

GGGGG starts with a letter, uses only letters, and is not one of the given reserved keywords. T-3 contains a hyphen, float is explicitly reserved, and 4thInput starts with a digit; all violate p.5 rules. Module support: PDF p. 5.

Why each choice is right or wrong

A - Incorrect: "T-3" does not match the rule or result derived here. GGGGG starts with a letter, uses only letters, and is not one of the given reserved keywords. T-3 contains a hyphen, float is explicitly reserved, and 4thInput starts with a digit; all violate p.5 rules.

B - Correct: "GGGGG" matches the derived result. GGGGG starts with a letter, uses only letters, and is not one of the given reserved keywords. T-3 contains a hyphen, float is explicitly reserved, and 4thInput starts with a digit; all violate p.5 rules.

C - Incorrect: "float" does not match the rule or result derived here. GGGGG starts with a letter, uses only letters, and is not one of the given reserved keywords. T-3 contains a hyphen, float is explicitly reserved, and 4thInput starts with a digit; all violate p.5 rules.

D - Incorrect: "4thInput" does not match the rule or result derived here. GGGGG starts with a letter, uses only letters, and is not one of the given reserved keywords. T-3 contains a hyphen, float is explicitly reserved, and 4thInput starts with a digit; all violate p.5 rules.

Primary note proof

Study-note section CP05

In the basic ASCII identifier rules, start with a letter or underscore; continue with letters, digits or underscores. Do not use a keyword. C++ is case sensitive: exitExam and ExitExam differ; delete and float are keywords, but FLOAT is a legal identifier. variable_1234 is valid; 4thInput, T-3 and names containing spaces are not single identifiers. Avoid reserved underscore patterns in application code.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 2-10

> 1. Keywords (reserved words) - Unique meaning within a C++ program and must not be used for any other purposes. E.g., if, namespace for, switch, continue, float, new, default, break Etc. 2. Identifiers - Name associated with a function or data object and used to refer to that function or data object. An identifier must: o Start with a letter or underscore o Consist only of letters, the digits 0-9, or the underscore symbol _ o Not be a reserved word

Question sources

Computer Science model exam 2 · Question 77

Question 70 | Q01259

An array is declared and initialized as follows:

int A[4][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 8, 7, 6}, {5, 4, 3, 2}};

What are the values of `A[1][1]`, `A[2][3]`, and `A[3][3]`, respectively?

A. 7, 6, 3

B. 6, 6, 2

C. 6, 2, 6

D. 2, 6, 6

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: 6, 6, 2

Arrays are zero-indexed (p.19). Row 1 is {5,6,7,8}, hence A[1][1]=6; row 2 is {9,8,7,6}, hence A[2][3]=6; row 3 ends with 2, hence A[3][3]=2. Only B preserves all three positions. Module support: PDF p. 19, PDF p. 21.

Why each choice is right or wrong

A - Incorrect: "7, 6, 3" does not match the rule or result derived here. Arrays are zero-indexed (p.19). Row 1 is {5,6,7,8}, hence A[1][1]=6; row 2 is {9,8,7,6}, hence A[2][3]=6; row 3 ends with 2, hence A[3][3]=2. Only B preserves all three positions.

B - Correct: "6, 6, 2" matches the derived result. Arrays are zero-indexed (p.19). Row 1 is {5,6,7,8}, hence A[1][1]=6; row 2 is {9,8,7,6}, hence A[2][3]=6; row 3 ends with 2, hence A[3][3]=2. Only B preserves all three positions.

C - Incorrect: "6, 2, 6" does not match the rule or result derived here. Arrays are zero-indexed (p.19). Row 1 is {5,6,7,8}, hence A[1][1]=6; row 2 is {9,8,7,6}, hence A[2][3]=6; row 3 ends with 2, hence A[3][3]=2. Only B preserves all three positions.

D - Incorrect: "2, 6, 6" does not match the rule or result derived here. Arrays are zero-indexed (p.19). Row 1 is {5,6,7,8}, hence A[1][1]=6; row 2 is {9,8,7,6}, hence A[2][3]=6; row 3 ends with 2, hence A[3][3]=2. Only B preserves all three positions.

Primary note proof

Study-note section CP23

int a[R][C] has R rows, each with C elements. a[r][c] uses zero-based row then column. Its element offset is r*C+c in row-major storage. In the 4x4 matrix example, a[1][1]=6, a[2][3]=6 and a[3][3]=2. In Java, length belongs to each array; a traversal ending at arr.length-2 visits fewer elements than a full traversal.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 19 | lines 4-29

Arrays - an array is a collection of fixed-size number of elements of the same type stored sequentially in memory. - An individual element of an array is identified by its own unique index (or subscript). - The number of elements must be an integer. - The size of the array is referred to as its dimension - Array in C++ is Zero indexed. Declaring of one-dimensional Array Syntax: elementstype arrayname [size]; → You can use constants such as macro definitions or const ints as the sizes of arrays. Example: int num[5]; string fruit[3]; #define numstudents 100 or const int numstudents=100; double Score[numstudents] ; double Score[numstudents] ; Initialization of an Array - Arrays can be initialized at declaration time in two different ways: Syntax: elementstype arrayname[size] = {list each value separated by comma}; E.g int num[5]={1,2,3,4,5}; elementstype arrayname[ ] = { list with any number of values }; E.g int num[]={1,2,3,4,5}; string fruit[]={“Apple”,“orange”, “Mango”}; Accessing Array Elements - Each element could be accessed using an index into the array - When accessing an array, the index given must be a positive integer from 0 to n-1, where n is the dimension of the array.

Computer Programming module | PDF page 21 | lines 28-36

Multidimensional Array C++ support the creation of multidimensional array, through the addition of more than one set of brackets. Syntax: elementstype arrayname [size1] [size2] ... [sizen]; Two-Dimensional Array Declaration: elementstype arrayname [size1] [size2]; Eg. int num[2][3];//declare 2 row, 3 columns to store 6 elements Initialization of Two-Dimensional Array E.g. int num[2][3]={1,2,3,4,5,6};//2 row 3 columns Or int num[2][3]={{1,2},{3,4},{5,6}};

Question sources

Computer Science model exam 2 · Question 80

Question 71 | Q01275

Before a program is written, the programmer must ensure to:

A. Clear and understand what data are to be used

B. Clear and understand the desired result

C. Clear and understand the procedure to be used

D. All of the above

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: All of the above

The problem-solving checklist requires understanding the problem, its data and output, and designing algorithm steps (p.2). Thus data, desired result and procedure all belong to preparation, making All the most complete choice. A/B/C are individually necessary but incomplete as a combined checklist. Module support: PDF p. 2.

Why each choice is right or wrong

A - Incorrect: Supported component: understand the data to be manipulated (p.2); less complete than All.

B - Incorrect: Supported component: identify expected output (p.2); less complete than All.

C - Incorrect: Supported component: design the algorithm (p.2); less complete than All.

D - Correct: Combines all three preparation requirements on p.2.

Primary note proof

Study-note section CP01

> Identify inputs, outputs, legal ranges and edge cases before coding. Solve one small case manually, choose an algorithm, express it as pseudocode or a flowchart, implement, test and maintain. For the largest of three numbers, initialize best to the first input, compare the second and third, then report best. Initializing best to zero fails when all inputs are negative.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 2 | lines 2-18

> 1. Introduction to programming Computer programming is the process of writing, testing, debugging/troubleshooting, and maintaining the source code of computer programs. Programs are the instructions that tells the computer what to do .so, writing computer programs means writing instructions, that will make the computer follow and run a program based on those instructions. We are going to write computer programs to solve problems. Problem Solving Problem: gap between the existing and the desired or proposed situation where problem solving will try to fill this gap. Problem solving: The process of transforming the description of a problem

into the solution by using our knowledge of the problem-solving strategies, techniques and tools. Problem solving techniques • Analyze the problem: Carefully understand the problem • Outline the problem requirements • Does program require user interaction? • Does program manipulate data? • What is the output? • Design steps (algorithm) to solve the problem

Question sources

Computer Science model exam 2 · Question 96

Question 72 | Q01393

In the conventional language-level taxonomy, which category generally offers source that is easier to read and often portable after recompilation when platform assumptions are avoided?

A. Low-level Language

B. High-Level Language

C. Decimal-Format

D. Middle-Level Language

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: B. High-Level Language

High-level source is easier to read and often portable after recompilation when it avoids platform assumptions. Machine and assembly languages depend more directly on an instruction set. The question asks about this language-level property, rather than claiming that people write programs in only one category. Decimal format describes numeric notation, and middle-level is not a separate category in the specified taxonomy.

Why each choice is right or wrong

A - Incorrect: Low-level machine and assembly source depends on an instruction set.

B - Correct: High-level source has the readability and conditional portability described.

C - Incorrect: Decimal format is a numeric representation, not this language category.

D - Incorrect: The stated taxonomy assigns this property to high-level languages.

Primary note proof

Study-note section CP03

Machine and assembly code depend on an instruction set. High-level source is easier to read and often portable after recompilation, provided it avoids platform assumptions. People can write both assembly and high-level programs. Python, Perl and PHP are commonly called scripting languages; C++ is conventionally compiled. These labels describe common use, not an absolute technical prohibition.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

> Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Question sources

Compiler Design review questions · Question 1

Question 73 | Q01394

Which computer program accepts the high-level language and converts it into assembly language?

A. Interpreter

B. Linker

C. Assembler

D. Compiler

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: Compiler.

CP04 describes compilation as a stage that may emit assembly from source. The assembler then translates that assembly into object-file machine instructions, and linking combines objects and libraries. Thus the question asks for the compiler, without implying that every compiler implementation must produce assembly as its output.

Why each choice is right or wrong

A - Incorrect: An interpreter executes a representation rather than being the stated high-level-to-assembly translation stage.

B - Incorrect: The linker resolves external symbols and combines objects and libraries after their translation.

C - Incorrect: The assembler consumes assembly and emits machine instructions, rather than consuming high-level source.

D - Correct: A compiler can translate high-level source into assembly, as described in the conventional toolchain.

Primary note proof

Study-note section CP04

> In the conventional C++ toolchain: source → preprocessing → compilation proper → assembly → object files → linking → executable → loading → execution. Preprocessing handles directives such as #include; compilation may emit assembly; the assembler emits machine instructions in an object file. The linker resolves external symbols and combines object files and libraries. The loader places an executable and required components in memory and starts it.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 4 | lines 21-30

> The compilation process of C++ program consists of the following steps. • edit source code → compile → link → execute (Editor) (Compiler) (Linker) (Loader) I. Program source code is entered into a file using a text editor. II. After the code has been entered, a compiler program is started that translates the source into an object code file. III. The object code file is linked with other object code files that come with the compiler and an executable file (or program) is created. IV. In order to execute the program, a program called the loader copies the executable file into the memory of the computer and sends an execute command to the CPU.

Question sources

Compiler Design review questions · Question 2

Question 74 | Q01395

In the conventional ahead-of-time C++ toolchain, is the source translation unit compiled before its resulting program is executed, even though compilation may use several internal passes?

A. No

B. Depends on the Compiler

C. Don't Know

D. Yes

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: D. Yes

The conventional C++ toolchain preprocesses and compiles source, assembles object files, links an executable and then loads it for execution. Thus the specified source translation occurs before running the resulting program. This does not mean the compiler has only one internal pass or must directly emit a complete native executable. Under the explicit conventional ahead-of-time model, Yes is the definite answer.

Why each choice is right or wrong

A - Incorrect: No contradicts the stipulated ahead-of-time build sequence.

B - Incorrect: The question fixes the conventional ahead-of-time model rather than leaving the execution strategy unspecified.

C - Incorrect: The stated model supplies enough information to decide.

D - Correct: The source unit is compiled before the resulting program executes.

Primary note proof

Study-note section CP04

> GCC documents the first four stages explicitly. A compiler need not emit a complete executable or even native machine code. Java javac produces bytecode, which a JVM interprets or compiles at runtime. Interpreters execute a program representation; “one source line at a time” is a beginner approximation. Compilation uses multiple passes, so “whole program in one step” is not literally universal.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

> Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Computer Programming module | PDF page 4 | lines 21-30

> The compilation process of C++ program consists of the following steps. • edit source code → compile → link → execute (Editor) (Compiler) (Linker) (Loader) I. Program source code is entered into a file using a text editor. II. After the code has been entered, a compiler program is started that translates the source into an object code file. III. The object code file is linked with other object code files that come with the compiler and an executable file (or program) is created. IV. In order to execute the program, a program called the loader copies the executable file into the memory of the computer and sends an execute command to the CPU.

Question sources

Compiler Design review questions · Question 3

Question 75 | Q01396

Which of the following file is an output of the assembler?

A. Program file

B. Object file

C. Data File

D. Task File

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: Object file.

In CP04's toolchain the assembler emits machine instructions in an object file. Linking subsequently combines object files and libraries into an executable. The question therefore asks for the object file, rather than an unspecified program, data or task file; those other names do not identify the assembler's defined output stage.

Why each choice is right or wrong

A - Incorrect: Program file is a vague description and does not identify the assembler's specified object-file output.

B - Correct: The assembler emits an object file containing translated machine instructions for later linking.

C - Incorrect: Data file does not identify the translated instruction object produced at the assembly stage.

D - Incorrect: Task file is not a stage output identified for the assembler in CP04.

Primary note proof

Study-note section CP04

> In the conventional C++ toolchain: source → preprocessing → compilation proper → assembly → object files → linking → executable → loading → execution. Preprocessing handles directives such as #include; compilation may emit assembly; the assembler emits machine instructions in an object file. The linker resolves external symbols and combines object files and libraries. The loader places an executable and required components in memory and starts it.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 4 | lines 21-30

> The compilation process of C++ program consists of the following steps. • edit source code → compile → link → execute (Editor) (Compiler) (Linker) (Loader) I. Program source code is entered into a file using a text editor. II. After the code has been entered, a compiler program is started that translates the source into an object code file. III. The object code file is linked with other

object code files that come with the compiler and an executable file (or program) is created. IV. In order to execute the program, a program called the loader copies the executable file into the memory of the computer and sends an execute command to the CPU.

Question sources

Compiler Design review questions · Question 4

Question 76 | Q01398

What is the linker's role in the conventional C++ build process?

A. It translates assembly source into object files.

B. It combines object files and libraries and resolves external symbols to create an executable module.

C. It is the same as the loader

D. None of the above

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: B. It combines object files and libraries and resolves external symbols to create an executable module.

The assembler has already produced object files when linking begins. The linker combines those files and libraries and resolves their external symbol references to create the executable module. The loader subsequently places that executable and required components in memory. These distinct responsibilities make option B precise: A describes assembly instead, C conflates linking with loading, and D is false because B supplies the correct role.

Why each choice is right or wrong

A - Incorrect: Translating assembly into an object file is the assembler's role.

B - Correct: The linker combines objects/libraries and resolves external references to make the executable.

C - Incorrect: The loader's memory-placement role is separate from linking.

D - Incorrect: B gives the correct linker responsibility.

Primary note proof

Study-note section CP04

In the conventional C++ toolchain: source → preprocessing → compilation proper → assembly → object files → linking → executable → loading → execution. Preprocessing handles directives such as #include; compilation may emit assembly; the assembler emits machine instructions in an object file. The linker resolves external symbols and combines object files and libraries. The loader places an executable and required components in memory and starts it.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 4 | lines 21-30

> The compilation process of C++ program consists of the following steps. • edit source code → compile → link → execute (Editor) (Compiler) (Linker) (Loader) I. Program source code is entered into a file using a text editor. II. After the code has been entered, a compiler program is started that translates the source into an object code file. III. The object code file is linked with other object code files that come with the compiler and an executable file (or program) is created. IV. In order to execute the program, a program called the loader copies the executable file into the memory of the computer and sends an execute command to the CPU.

Question sources

Compiler Design review questions · Question 6

Question 77 | Q01424

Which statement is false about the conventional C++ toolchain described in the note?

A. Every compiler directly emits a complete executable without a separate linking stage.

B. A debugger is useful after a runnable program exists.

C. Source syntax errors must be repaired for successful translation.

D. Compilation may use multiple passes and may emit assembly.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: A. Every compiler directly emits a complete executable without a separate linking stage.

The note separates several translation and build stages. Compilation can emit assembly, assembly produces object files, and linking resolves external symbols and combines those objects. It therefore does not follow that every compiler directly produces a complete executable. Source grammar errors must be corrected for successful translation, and a debugger is useful after a runnable program exists. These stage-based claims avoid unspecified execution-time and debugging-speed comparisons.

Why each choice is right or wrong

A - Correct: A compiler need not emit a complete executable; linking is a separate stage.

B - Incorrect: The note places debugger use after a runnable program exists.

C - Incorrect: A successfully translated C++ program has passed its source syntax checks.

D - Incorrect: Compilation can use multiple passes and can emit assembly.

Primary note proof

Study-note section CP04

In the conventional C++ toolchain: source → preprocessing → compilation proper → assembly → object files → linking → executable → loading → execution. Preprocessing handles directives such as #include; compilation may emit assembly; the assembler emits machine instructions in an object file. The linker resolves external symbols and combines object files and libraries. The loader places an executable and required components in memory and starts it.

Study-note section CP32

A syntax error violates grammar; a type error can be syntactically well formed but rejected during semantic checking. A runtime error occurs during execution; a logic error gives the wrong behavior. A bug is a defect; debugging locates and corrects it. Read the first useful compiler diagnostic, inspect nearby tokens and declarations, repair and rebuild; a debugger is useful after a runnable program exists.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 4 | lines 21-30

The compilation process of C++ program consists of the following steps. • edit source code → compile → link → execute (Editor) (Compiler) (Linker) (Loader) I. Program source code is entered into a file using a text editor. II. After the code has been entered, a compiler program is started that translates the source into an object code file. III. The object code file is linked with other object code files that come with the compiler and an executable file (or program) is created. IV. In order to execute the program, a program called the loader copies the executable file into the memory of the computer and sends an execute command to the CPU.

Computer Programming module | PDF page 3 | lines 8-17

Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Question sources

Compiler Design review questions · Question 32

Question 78 | Q01518

Which one of the following is a VALID C++ declaration statement?

A. age int;

B. int 1x, 2x;

C. int FLOAT;

D. float abebe, A-Z;

Hint

Apply the naming, type and syntax rules before deciding whether an option can be used as stated.

Answer C: int FLOAT;

The required declaration form is type followed by identifiers (p.5). int FLOAT fits: uppercase FLOAT is used as an identifier, distinct from lowercase reserved float in the intended C++ reading. A reverses type/name, B starts identifiers with digits, and D uses a hyphen in A-Z. Module support: PDF p. 5, PDF p. 5.

Why each choice is right or wrong

A - Incorrect: "age int;" does not match the rule or result derived here. The required declaration form is type followed by identifiers (p.5). int FLOAT fits: uppercase FLOAT is used as an identifier, distinct from lowercase reserved float in the intended C++ reading. A reverses type/name, B starts identifiers with digits, and D uses a hyphen in A-Z.

B - Incorrect: "int 1x, 2x;" does not match the rule or result derived here. The required declaration form is type followed by identifiers (p.5). int FLOAT fits: uppercase FLOAT is used as an identifier, distinct from lowercase reserved float in the intended C++ reading. A reverses type/name, B starts identifiers with digits, and D uses a hyphen in A-Z.

C - Correct: "int FLOAT;" matches the derived result. The required declaration form is type followed by identifiers (p.5). int FLOAT fits: uppercase FLOAT is used as an identifier, distinct from lowercase reserved float in the intended C++ reading. A reverses type/name, B starts identifiers with digits, and D uses a hyphen in A-Z.

D - Incorrect: "float abebe, A-Z;" does not match the rule or result derived here. The required declaration form is type followed by identifiers (p.5). int FLOAT fits: uppercase FLOAT is used as an identifier, distinct from lowercase reserved float in the intended C++ reading. A reverses type/name, B starts identifiers with digits, and D uses a hyphen in A-Z.

Primary note proof

Study-note section CP05

> In the basic ASCII identifier rules, start with a letter or underscore; continue with letters, digits or underscores. Do not use a keyword. C++ is case sensitive: exitExam and ExitExam differ; delete and float are keywords, but FLOAT is a legal identifier. variable_1234 is valid; 4thInput, T-3 and names containing spaces are not single identifiers. Avoid reserved underscore patterns in application code.

Study-note section CP06

Write type name; or type name = value; for ordinary variables: int age = 20;. Declaration introduces a name and type; initialization gives its initial value; later assignment replaces a value. a=b copies a scalar value, so later changes to b do not change a. A pointer or Java object reference instead copies an address/reference.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 24-33

5. Data Types, Variables, and Constants Variable: - Variable is a symbolic name for a memory location in which data can be stored. - Attribute of Variables: o A type, which is, established when the variable is defined (e.g., integer, float, character) o A value, which can be changed by assigning a new value to the variable Variable Declaration - Declaring a variable means defining (creating) a variable. - Syntax: data type variable name; e.g. int num; - We can create More Than One Variable at a Time e.g., int num1, num2, num3;

Computer Programming module | PDF page 5 | lines 2-10

1. Keywords (reserved words) - Unique meaning within a C++ program and must not be used for any other purposes. E.g., if, namespace for, switch, continue, float, new, default, break Etc. 2. Identifiers - Name associated with a function or data object and used to refer to that function or data object. An identifier must: o Start with a letter or underscore o Consist only of letters, the digits 0-9, or the underscore symbol _ o Not be a reserved word

Question sources

Mock exam 1 · Question 34

Question 79 | Q01519

Which of the following is not true?

A. (8.0 * 3) % 5 is evaluated to 4

B. 8 * 3 / 3 evaluated to 8

C. (5 > 3) || (5 != 5) && (2 <= 2) results in true

D. (7.0 * 3) / 5 is evaluated to 4.2

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: The floating-point remainder expression is invalid in C++.

The expression (8.0 * 3) has floating type, so applying % to it is invalid under CP08; it does not evaluate to 4. The integer multiplication/division in B gives 8. C is true because its left disjunct is true. D uses floating division and gives 21.0/5 = 4.2.

Why each choice is right or wrong

A - Correct: This is the false statement: floating-point operands cannot be used with the C++ remainder operator.

B - Incorrect: Equal-precedence multiplication and division group left to right: 24/3 gives the stated 8.

C - Incorrect: The first comparison is true, so the disjunction is true regardless of its right operand.

D - Incorrect: 7.0 makes the computation floating point, and 21.0 divided by 5 gives 4.2.

Primary note proof

Study-note section CP08

> Multiplication, division and remainder have higher precedence than addition and subtraction; equal-precedence arithmetic groups left to right. Integer division truncates toward zero. For integers, a == (a/b)*b + a%b when the division is valid. % requires integral operands in C++; (8.0*3)%5 is invalid. Division by zero and signed integer overflow cannot be given ordinary numeric outputs.

Study-note section CP10

> Relational and equality operators test values; && and || combine truth values; ! negates. In C++, zero converts to false and a nonzero number to true. Streams display bool as 0 or 1 unless boolalpha is set. With a=3,b=5, cout<<(a==b) prints 0.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Mock exam 1 · Question 35

Question 80 | Q01520

Which one of the following is the result of the below given code fragment?
int a = 7, b;
b = a++;
cout<<++b;

A. 6

B. 9

C. 7

D. 8

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer D: 8

b=a++ assigns the old a value 7 to b and increments a to 8. Then ++b changes b to 8 before output. Thus 8 prints; 7 omits the prefix increment and 6/9 do not follow the trace. Module support: PDF p. 7.

Why each choice is right or wrong

A - Incorrect: "6" does not match the rule or result derived here. b=a++ assigns the old a value 7 to b and increments a to 8. Then ++b changes b to 8 before output. Thus 8 prints; 7 omits the prefix increment and 6/9 do not follow the trace.

B - Incorrect: "9" does not match the rule or result derived here. b=a++ assigns the old a value 7 to b and increments a to 8. Then ++b changes b to 8 before output. Thus 8 prints; 7 omits the prefix increment and 6/9 do not follow the trace.

C - Incorrect: "7" does not match the rule or result derived here. b=a++ assigns the old a value 7 to b and increments a to 8. Then ++b changes b to 8 before output. Thus 8 prints; 7 omits the prefix increment and 6/9 do not follow the trace.

D - Correct: "8" matches the derived result. b=a++ assigns the old a value 7 to b and increments a to 8. Then ++b changes b to 8 before output. Thus 8 prints; 7 omits the prefix increment and 6/9 do not follow the trace.

Primary note proof

Study-note section CP09

> Prefix ++x changes x and yields the new value; postfix x++ yields the previous value and also increments x. Separate the value used in the expression from the stored value after it. Starting a=8,b=5, a++ + --b uses 8+4, gives 12, and leaves a=9,b=4. Starting x=1,y=10,z=-3, (x++*y--)+ ++z gives 8 and leaves x=2,y=9,z=-2.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, | |) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 | | 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Mock exam 1 · Question 36

Question 81 | Q01575

Considering the below given string declaration, which one of the following statements is correct? char msg[] = "This is the final exam for C++";

A. There is an error with the initialization statement

B. strlen(msg) would return 24

C. strcpy(msg, "Hurray") would assign the content of msg to Hurray

D. cout<<msg ; would display: This is the final exam for C++

E. None of the answers are correct

Hint

Apply the naming, type and syntax rules before deciding whether an option can be used as stated.

Answer D: cout<<msg ; would display: This is the final exam for C++

The module supports char-array string initialization and output. Printing msg displays its literal This is the final exam for C++. Its length is 28 characters, not 24. strcpy(destination,source) copies Hurray into msg, not the reverse. Thus A/B/C/E fail and D matches. Module support: PDF p. 22, PDF p. 23, PDF p. 6.

Why each choice is right or wrong

A - Incorrect: "There is an error with the initialization statement" does not match the rule or result derived here. The module supports char-array string initialization and output. Printing msg displays its literal This is the final exam for C++. Its length is 28 characters, not 24. strcpy(destination,source) copies Hurray into msg, not the reverse. Thus A/B/C/E fail and D matches.

B - Incorrect: "strlen(msg) would return 24" does not match the rule or result derived here. The module supports char-array string initialization and output. Printing msg displays its literal This

is the final exam for C++. Its length is 28 characters, not 24. strcpy(destination,source) copies Hurray into msg, not the reverse. Thus A/B/C/E fail and D matches.

C - Incorrect: "strcpy(msg, "Hurray") would assign the content of msg to Hurray" does not match the rule or result derived here. The module supports char-array string initialization and output. Printing msg displays its literal This is the final exam for C++. Its length is 28 characters, not 24. strcpy(destination,source) copies Hurray into msg, not the reverse. Thus A/B/C/E fail and D matches.

D - Correct: "cout<<msg ; would display: This is the final exam for C++" matches the derived result. The module supports char-array string initialization and output. Printing msg displays its literal This is the final exam for C++. Its length is 28 characters, not 24. strcpy(destination,source) copies Hurray into msg, not the reverse. Thus A/B/C/E fail and D matches.

E - Incorrect: "None of the answers are correct" does not match the rule or result derived here. The module supports char-array string initialization and output. Printing msg displays its literal This is the final exam for C++. Its length is 28 characters, not 24. strcpy(destination,source) copies Hurray into msg, not the reverse. Thus A/B/C/E fail and D matches.

Primary note proof

Study-note section CP24

A C string is a character array ending in '\0'. "Hello" needs six char positions; strlen counts five visible characters and excludes the terminator. sizeof a fixed char array reports its allocated bytes, which can exceed strlen+1. cout<<a prints a terminated char sequence, not its numeric address.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 22 | lines 22-38

String and pointer String: - is an ordered sequence of characters - Sequence of character in which the last character is the null character „\0‟ - The null character indicates the end of the string - C++ provides following two types of string representations: o The C-style character string. o The string class type introduced with Standard C++. NB: →If a string has n characters, then it requires an n+1 element array (at least) to store it. →In order to use the string data type, the C++ string header <string> must be included at the top of the program The C-Style Character String - originated within the C language and continues to be supported within C++ - array of characters which is terminated by a null character '\0' Example: char status[5]={„G‟,‟O‟,‟O‟,‟D‟,‟\0‟};//char status[]=”Good”; cout<<status;

Computer Programming module | PDF page 23 | lines 2-8

- C++ supports a wide range of functions that manipulate null-terminated strings: strcpy(s1, s2); Copies string s2 into string s1 strcat(s1, s2); Concatenates string s2 onto the end of string s1 strlen(s1); Returns the length of string s1. strcmp(s1, s2); Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2. strchr(s1, ch); Returns a pointer to the first occurrence of character ch in string s1. strstr(s1, s2); Returns a pointer to the first occurrence of string s2 in string s1.

Computer Programming module | PDF page 6 | lines 4-15

> 6. Communication through console cout (the standard output stream) o used to display information for the user Syntax: cout<<object; • Object can be variable or literals Example: int x=5; cout<<x cout<< “Hello there”; cin(the standard input stream): o used to accept information from the user. Syntax: cin>>variable name; Eg. cin>>x;

Question sources

Mock exam 3 · Question 7

Question 82 | Q01576

Given the structure below, assume ptr1 and ptr2 point to separate live house objects. Which statement copies the rooms value from the first object to the second?
struct house { double price; int rooms; };
house *ptr1, *ptr2;

A. Cout<<ptr1<< ptr2;

B. ptr2->rooms = ptr1->rooms;

C. ptr1 < ptr2;

D. ptr1 = ptr1 + ptr2;

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: B. ptr2->rooms = ptr1->rooms;

Each pointer must refer to a live object before its members are accessed. The -> operator selects a member through a pointer, so assigning ptr1->rooms to ptr2->rooms copies the first object’s integer member into the second. A is an output expression with a misspelled stream name, C compares pointers rather than copying a member, and D tries to add two pointers, which is not permitted pointer arithmetic.

Why each choice is right or wrong

A - Incorrect: This output expression does not copy rooms and uses undeclared Cout.

B - Correct: Both -> expressions select rooms, and assignment copies from ptr1 to ptr2.

C - Incorrect: A pointer comparison does not copy a member value.

D - Incorrect: Adding two pointers is not valid pointer arithmetic.

Primary note proof

Study-note section CP27

> A struct groups named members, possibly of different types. struct House{double price; int rooms;}; defines a type; House h{150000,3}; creates an object. h.rooms uses a dot. House* p=&h;

p->rooms is equivalent to (*p).rooms. The pointer must refer to a live object before member access. A type definition alone does not create an ordinary instance.

Study-note section CP26

Pointer addition p+q is invalid; subtracting pointers within the same array gives an element distance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 38 | lines 10-16

Pointer to structure Pointers to structures may be declared as follow: struct staructuretage *pinternname; eg Employe *emp; When using a pointer to a struct, member access can be achieved with the `.' operator, but can not look graceful in expression Example: (*emp).id Equivalently, the ->' operator can be used; for example: emp->id

Computer Programming module | PDF page 26 | lines 2-22

Pointer - A pointer is a variable that holds a memory address of another variable. - - Its content is a memory address - A pointer is simply the address of a memory location and provides an indirect way of accessing data in memory. - It is defined to „point to“ data of a specific type. - Because a pointer variable holds the address of another piece of data, it "points to" the data Reference operator (&) and Dereference operator (*) &: read as address of - Use address operator & to get address of a variable *: read as value pointed by Declaring variables of pointer types Syntax: type *variable_name; For example: int *ptr1; //pointertoanint double*number; char*ptr2; //pointertoachar The value of a pointer variable is the address to which it points. For example, given the definitions int num; wecanwrite: ptr1=#

Question sources

Mock exam 3 · Question 8

Question 83 | Q01577

Under C++17, which option best summarizes whether initialization forms A-C are valid? Treat each form as a separate declaration; 25.00 is exactly representable as float.

A. struct Time { int h = 06, m = 20; float s = 25.00; };

B. struct Time { int h, m; float s;}t = {06, 20, 25.00};

C. struct Time { int h, m; float s; }; Time t = {06, 20, 25.00};

D. All of them are correct

E. None of the choices are correct

Hint

Apply the naming, type and syntax rules before deciding whether an option can be used as stated.

Answer D: D. All of them are correct

C++17 permits default member initializers, so A's in-class values are allowed. B defines an aggregate object alongside its type, while C defines the type and then aggregate-initializes an object. The specified exactly representable floating constant is permitted here's narrowing exception. All three forms are consequently valid. The question asks for the complete summary, making D the best selection rather than any one individual example.

Why each choice is right or wrong

A - Incorrect: A is valid but alone does not summarize all three forms.

B - Incorrect: B is valid but describes only one of the three valid forms.

C - Incorrect: C is valid but is an incomplete response to the requested summary.

D - Correct: All three forms are valid under the specified C++17 assumptions.

E - Incorrect: The existence of valid forms A-C makes the none-correct summary false.

Primary note proof

Study-note section CP27

> Aggregate initialization assigns members in declaration order. Modern C++ permits default member initializers such as int rooms=3;. C++11 list initialization rejects narrowing; a constant floating value exactly representable in the target floating type can be permitted, so 25.00 to float is not necessarily narrowing. Use 25.0f for clarity and check the language version. Structures can be copied, passed and returned; a structure cannot contain a full instance of itself, but may contain a pointer to its own type.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 35 | lines 37-44

> Initialize value #include<iostream> using namespace std; struct student{ string name; int age; double score; }st={"Abebe",19,24};

Question sources

Mock exam 3 · Question 9

Question 84 | Q01615

If we have object from ofstream class, then default mode of opening the file is _____ .

A. ios::in

B. ios::out

C. ios::out|ios::trunk

D. ios::in|ios::trunc

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: B. ios::out

The ofstream constructor's default mode argument is ios::out. Its usual output-only opening behavior also creates or truncates the file, but that effect does not change the spelling of the default argument. ios::in requests input instead. The printed ios::trunk is not the valid truncation flag, and an input/truncation combination is not the output stream's default. Thus B is the unique default-mode selection.

Why each choice is right or wrong

A - Incorrect: ios::in requests input rather than the default output mode.

B - Correct: ios::out is the default ofstream mode argument.

C - Incorrect: ios::trunk is misspelled; even out|trunc describes an effect rather than the constructor's default argument.

D - Incorrect: Input combined with truncation is not the default output mode argument.

Primary note proof

Study-note section CP29

> Include <fstream>. ofstream writes, ifstream reads, and fstream can do both. The constructor's default ofstream mode argument is ios::out; its ordinary effect is to create/truncate the file. Distinguish the default mode argument from effective out|trunc behavior. ios::app places each write at the end; ios::ate initially seeks to the end; ios::binary requests binary mode.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 41 | lines 2-9

> • fstream: Stream class to both read and write from/to files. File Modes In C++, for every file operation, exists a specific file mode. These file modes allow us to create, read, write, append or modify a file. The file modes are defined in the class ios. Let's see all these different modes in which we could open a file on disk. Create and Write to a File To create a file, use either the ofstream or fstream class, and specify the name of the file. To write to the file, use the insertion operator (<<).

Computer Programming module | PDF page 40 | lines 16-22

> File and File Management Files are used to store data in storage devices permanently. The fstream library allows us to work with files. To use the fstream library, include both the standard <iostream> AND the <fstream> header file: Input/Output with files • ofstream: Stream class to write on files • ifstream: Stream class to read from files

Question sources

Mock exam 2 · Question 4

Question 85 | Q01616

After the prompt and input of vehicleType=2, what text is printed by the switch body?

```
int vehicleType;
cout << "Enter vehicle type: ";
cin >> vehicleType;
switch (vehicleType) {
case 1:
cout << "Passenger car.";
case 2:
cout << "Bus.";
case 3:
cout << "Truck.";
break;
default:
cout << "Unknown vehicle class!";
}
```

A. Passenger Car.

B. Bus.Truck.

C. Truck

D. Bus.

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: B. Bus.Truck.

Input 2 begins execution at case 2, so Bus. prints first. There is no break after that output, so execution continues through case 3 and prints Truck. before reaching the break. The switch-body output is therefore Bus.Truck. The input prompt is excluded explicitly. The other options either start at the wrong case or omit one of the outputs produced by fall-through.

Why each choice is right or wrong

A - Incorrect: Case 1 is not entered when the value is 2.

B - Correct: Case 2 falls through to case 3 before the break.

C - Incorrect: The matching case prints Bus. before Truck.

D - Incorrect: The absent break after case 2 allows Truck. to print too.

Primary note proof

Study-note section CP13

A traditional C++ switch selects using an integral or enumeration value. case labels are constant expressions. Execution starts at the matching label, then continues through subsequent statements until a break, return or the end. default handles an unmatched value but need not be the last physical label.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 10 | lines 15-26

switch Statement The switch statement provides a way of choosing between a set of alternatives, based on the value of an expression. The general form of the switch statement is: switch (expression) { caseconstant1: statements; ... caseconstantn: statements; default: statements; }

Computer Programming module | PDF page 13 | lines 30-35

Using continue and break Statements - The break statement skips the rest of the statements in the loop‟s block, jumping immediately to the next statement outside of the loop. - The continue statement is similar. It also skips the rest of the statements in the loop‟s block, but instead of terminating the loop, it transfers execution to the next iteration of the loop. It continues the loop after skipping the remaining statements in its current iteration.

Question sources

Mock exam 2 · Question 5

Question 86 | Q01617

For the C++ fragment below, assume sizeof(float)==4, byte-addressed memory, a first-element address of 7000, and ordinary array-to-pointer conversion in B. Evaluate each option independently from ptr=num. Which statement is false?
float num[4]={2.0,1.0,3.0,5.5};
float* ptr=num;

A. ptr + 3 is equal to 7012.

B. num is the same as &num[0].

C. The statement cout<<num[3]; will print 5.5.

D. The statement cout<<ptr++; will print 1.0.

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer D: D. The statement cout<<ptr++; will print 1.0.

Pointer arithmetic advances by elements, not single bytes. With four-byte float elements, ptr+3 has address 7000+12=7012. The array expression ordinarily converts to &num[0], and index 3 selects the initialized value 5.5. Inserting ptr++ into cout displays its old pointer address and then advances it. It does not insert an array element’s value, so the claim that it prints 1.0 is false.

Why each choice is right or wrong

A - Incorrect: Three float elements advance the address by 12 bytes.

B - Incorrect: In the stated conversion context num becomes &num[0].

C - Incorrect: Index 3 contains the fourth initializer, 5.5.

D - Correct: ptr++ supplies the old pointer address, not the element value 1.0.

Primary note proof

Study-note section CP26

If p points inside an array, p+k advances k elements, or k*sizeof(*p) bytes. Under the explicit assumption sizeof(float)==4 and first address 7000, p+3 has numeric address 7012. cout<<p++ displays the old address then advances p; cout<<*p++ displays the old element then advances p. ++*p changes the element. Pointer addition p+q is invalid; subtracting pointers within the same array gives an element distance.

Study-note section CP25

An array expression commonly converts to a pointer to its first element, but the array is not itself an assignable pointer variable.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 29 | lines 2-10

Pointer arithmetic - The C++ language allows you to perform integer addition or subtraction operations on pointers. - If ptr points to an integer, ptr + 1 is the address of the next integer in memory after ptr. - ptr - 1 is the address of the previous integer before ptr. Note that ptr + 1 does not return the memory address after ptr, but the memory address of the next object of the type that ptr points to. If ptr points to an integer (assuming 4 bytes), ptr + 3 means 3 integers (12 bytes) after ptr. If ptr points to a char, which is always 1 byte, ptr + 3 means 3 chars (3 bytes) after ptr. When calculating the result of a pointer arithmetic expression, the compiler always multiplies the integer operand by the size of the object being pointed to. This is called scaling.

Computer Programming module | PDF page 31 | lines 2-11

- Array can bound into a pointer that points to the first element (element 0) of the array. - An array access can always be implicitly converted to a pointer access of the appropriate type. For example, consider these two declarations: int myarray[20]; int *mypointer; Mypointer=myarray;// mypointer=&myarray[0]; →Stores the address of the first element of the array in variable mypointer → After that, mypointer and myarray would be equivalent and would have very similar properties. →The main difference being that mypointer can be assigned a different address, whereas myarray can never be assigned anything, and will always represent the same block of 20 elements of type int.

Computer Programming module | PDF page 6 | lines 4-15

6. Communication through console cout (the standard output stream) o used to display information for the user Syntax: cout<<object; • Object can be variable or literals Example: int

x=5; cout<<x cout<< “Hello there”; cin(the standard input stream): o used to accept information from the user. Syntax: cin>>variable name; Eg. cin>>x;

Question sources

Mock exam 2 · Question 6

Question 87 | SM0009

In the conventional native-execution model, which statement about programming-language generations is false?

A. Assembly language is machine independent

B. Machine language can be executed without translation

C. Assembly language should be translated to machine language before execution

D. Fifth-generation languages are historically associated with declarative logic, constraints and AI/expert systems.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: A. Assembly language is machine independent

Assembly is symbolic source tied to an instruction set; it is not machine independent. Before native execution, an assembler translates it into machine instructions. Those machine instructions are already in the target machine’s instruction format. Fifth generation is instead a historical language category associated with declarative logic, constraints and expert systems, rather than a universal hardware-performance requirement. Only A contradicts the specified language and execution model.

Why each choice is right or wrong

A - Correct: Assembly depends on the target instruction set, so this claim is false.

B - Incorrect: Machine instructions do not require assembly-to-machine translation.

C - Incorrect: Symbolic assembly is translated before native execution.

D - Incorrect: This historical association is the fifth-generation description in the note.

Primary note proof

Study-note section CP03

Machine instructions are conventionally first generation; symbolic assembly is second; general high-level languages such as C++ and Java are third. Database query and application generators such as SQL are conventionally fourth generation. Fifth generation is a loose historical category associated with declarative logic, constraints and AI/expert systems. Generations do not determine whether an implementation uses an interpreter.

Study-note section CP04

In the conventional C++ toolchain: source → preprocessing → compilation proper → assembly → object files → linking → executable → loading → execution. Preprocessing handles directives such as #include; compilation may emit assembly; the assembler emits machine instructions in an object file. The linker resolves external symbols and combines object files and libraries. The loader places an executable and required components in memory and starts it.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Question sources

MOE Computer Science Exit Exam · Question 3-1

National Computer Science Exit Exam · Question 43

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 71

Question 88 | SM0030

For standard built-in C++ arrays declared with an explicit bound, which statement is false?

A. We use [] square brackets at the time of array declaration

B. Size of an array should be constant at the time of array declaration

C. An array is a collection of similar data objects

D. Elements can be accessed only by writing a bracketed index such as a[i].

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: D. Elements can be accessed only by writing a bracketed index such as a[i].

A standard built-in array declaration uses square brackets and a compile-time constant bound. Its elements share one type. Element access is not restricted to bracket syntax: a[i] is equivalent to *(a+i), and a pointer to an element can also be dereferenced. Option D asserts that bracketed indexing is the only access method, which the pointer equivalence directly disproves.

Why each choice is right or wrong

A - Incorrect: An explicit built-in array bound appears inside square brackets.

B - Incorrect: Standard built-in array bounds must be compile-time constants.

C - Incorrect: Built-in arrays contain same-type elements.

D - Correct: Pointer notation can access elements, so bracketed indexing is not the only method.

Primary note proof

Study-note section CP22

> A built-in C++ array has a fixed number of same-type elements in contiguous storage: int a[4]={10,20,30,40};. Valid indices are 0 through 3. a[4] is outside the array; accessing it has undefined behavior. Java checks bounds and throws ArrayIndexOutOfBoundsException. Do not predict a C++ out-of-range value from nearby memory.

Study-note section CP22

> A standard built-in array bound must be a compile-time constant; variable-length local arrays are compiler extensions. int a[5]={1,2,3}; fills the remaining elements with zeros. An uninitialized automatic array of int is different. a[i] equals *(a+i), so elements can be accessed using pointer notation without explicitly writing brackets.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 19 | lines 4-29

> Arrays - an array is a collection of fixed-size number of elements of the same type stored sequentially in memory. - An individual element of an array is identified by its own unique index (or subscript). - The number of elements must be an integer. - The size of the array is referred to as its dimension - Array in C++ is Zero indexed. Declaring of one-dimensional Array Syntax: elementstype arrayname [size]; → You can use constants such as macro definitions or const ints as the sizes of arrays. Example: int num[5]; string fruit[3]; #define numstudents 100 or const int numstudents=100; double Score[numstudents] ; double Score[numstudents] ; Initialization of an Array - Arrays can be initialized at declaration time in two different ways: Syntax: elementstype arrayname[size] = {list each value separated by comma}; E.g int num[5]={1,2,3,4,5}; elementstype arrayname[ ] = { list with any number of values }; E.g int num[]={1,2,3,4,5}; string fruit[]={“Apple”,“orange”, “Mango”}; Accessing Array Elements - Each element could be accessed using an index into the array - When accessing an array, the index given must be a positive integer from 0 to n-1, where n is the dimension of the array.

Computer Programming module | PDF page 31 | lines 31-36

> NB: →Generally, array[n] is the same as *(array + n), where n is an integer. →Considering that int *ptr; int arr[5]; ptr = arr; ptr + 1 is equivalent to &arr[1]; ptr + 2 is equivalent to &arr[2];

Question sources

MOE Computer Science Exit Exam · Question 9-2

National Computer Science Exit Exam · Question 35

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 72

Question 89 | SM0032

What values are printed by this C++ fragment? Ignore the trailing space; assume cout is available.

```
for(int n=1;n<=18;n=n+2){
if(n%7!=0) cout<<n<<" ";
else break;
}
```

A. 3 5 7

B. 1 3 5 7 9 11 13 15 17

C. 1 3 5

D. 1 2 3 4 5 6

Hint

Fix the input case and cost model, then count dominant work or simultaneously live storage.

Answer C: C. 1 3 5

The loop begins at 1 and adds two after each completed iteration. The values 1, 3 and 5 are not divisible by seven, so each is printed with a following space. At n=7 the condition is false and the else branch executes break, ending the loop before 7 is printed. Later odd values are never reached. The options show value sequences rather than exact trailing whitespace.

Why each choice is right or wrong

A - Incorrect: The loop prints 1 first and stops before printing 7.

B - Incorrect: break prevents 7 and all later odd values from printing.

C - Correct: The three printed values are 1, 3 and 5.

D - Incorrect: Adding two does not visit consecutive integers.

Primary note proof

Study-note section CP16

> For n=1; n<=18; n+=2, print n unless n%7==0 then break: output is 1 3 5. A while(a>b) with fixed a=6,b=5 never changes its true condition. In data[10]={5,10,...,50}, i=0, advancing i only if data[i]%10==0 causes no progress because data[0]=5. It prints nothing and loops indefinitely; changing <=10 to <10 alone does not repair progress.

Study-note section CP15

break exits the nearest enclosing loop or switch. continue skips the rest of the current loop iteration: a for loop then runs its update; a while loop goes directly to its condition. return leaves the current function and may supply a result. In main it ends the program normally.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 12 | lines 19-28

The for loop. Its format is: for (initialization; condition; increase) statement; 1, initialization is executed. Generally, it is a initial value setting for a counter variable. This is executed only once 2, condition is checked, if it is true the loop continues, otherwise the loop finishes and statement is skipped. 3, statement is executed. As usual, it can be either a single instruction or a block of instructions enclosed within curly brackets { }. 4, finally, whatever is specified in the increase field is executed and the loop gets back to step 2.

Computer Programming module | PDF page 13 | lines 30-35

Using continue and break Statements - The break statement skips the rest of the statements in the loop‟s block, jumping immediately to the next statement outside of the loop. - The continue statement is similar. It also skips the rest of the statements in the loop‟s block, but instead of terminating the loop, it transfers execution to the next iteration of the loop. It continues the loop after skipping the remaining statements in its current iteration.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

MOE Computer Science Exit Exam · Question 10-1

Question 90 | SM0055

In a C++ translation unit, int requests=0; is declared at namespace scope before the function definitions. Functions can use it wherever that declaration is visible. Which listed term describes its scope category rather than its storage duration?

A. Static

B. Automatic

C. Local

D. Global

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: D. Global

The declaration is at namespace scope, outside every function, so the variable has global scope in the terminology used here. Its declaration must still be visible where it is used, and a local name can shadow it. Static and automatic concern storage properties rather than identifying this outside-function scope category. The question therefore selects Global without claiming unrestricted visibility across every file.

Why each choice is right or wrong

A - Incorrect: Static storage duration does not uniquely identify scope; a static local can still have block scope.

B - Incorrect: Automatic describes storage duration and does not name the namespace-scope category.

C - Incorrect: A local name is restricted to its function or block.

D - Correct: Global is the category for this namespace-scope declaration, subject to visibility and shadowing.

Primary note proof

Study-note section CP20

> Scope is where a name can be used; lifetime is when its object exists. A local name is restricted to its block; an outer name may be shadowed. A namespace/global variable is available where its declaration is visible, not magically in every file. C++ ::name selects a global name hidden by a local one.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 17 | lines 14-29

> Scope of variables Scope of variables defined as the extent of the program code within which the variable can be accessed or declared or worked with mainly two types of variable scopes: local

variable: - Variables defined within a function or block are said to be local to those functions - Declared inside a block{} Global Variable - variable that can be accessed from any part of the program. - They are available throughout the life time of a program. - They are declared at the top of the program outside all of the functions or blocks. NB: - Usually when two variables with same name are defined then the compiler produces a compile time error. But if the variables are defined in different scopes, then the compiler allows it. - Whenever there is a local variable defined with same name as that of a global variable then the compiler will give precedence to the local variable

Question sources

MOE Computer Science Exit Exam · Question 16-2

National Computer Science Exit Exam · Question 60

National Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 74

Question 91 | SM0088

Which one of the following is NOT an example of scripting languages?

A. Python

B. Perl

C. PHP

D. C++

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: C++.

CP03 identifies Python, Perl and PHP as commonly called scripting languages and C++ as conventionally compiled. This supports C++ as the intended exception. The classification describes common use rather than an absolute prohibition on interpreting C++ or compiling languages normally used for scripting.

Why each choice is right or wrong

A - Incorrect: Python is explicitly included among the note’s commonly called scripting languages.

B - Incorrect: Perl is explicitly included among the note’s commonly called scripting languages.

C - Incorrect: PHP is explicitly included among the note’s commonly called scripting languages.

D - Correct: C++ is the conventionally compiled language contrasted with the three scripting-language examples.

Primary note proof

Study-note section CP03

Machine and assembly code depend on an instruction set. High-level source is easier to read and often portable after recompilation, provided it avoids platform assumptions. People can write both assembly and high-level programs. Python, Perl and PHP are commonly called scripting languages; C++ is conventionally compiled. These labels describe common use, not an absolute technical prohibition.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Computer Programming module | PDF page 4 | lines 4-19

programming paradigm: provides the programmer's view of code execution. The most influential paradigms are: Procedural Programming Languages Procedural programming specifies a list of operations that the program must complete to reach the desired state. Each program has a starting state, a list of operations to complete, and an ending point. This approach is also known as imperative programming. Structured Programming Languages Structured programming is a special type of procedural programming. It provides additional tools to manage the problems that larger programs were creating. Structured programming requires that programmers break program structure into small pieces of code that are easily understood. One of the well-known features of structural programming is that it does not allow the use of the GOTO statement. It is often associated with a "top-down" approach to design. Object-Oriented Programming Languages Object-oriented programming is one the newest and most powerful paradigms. In object- oriented programs, the designer specifies both the data structures and the types of operations that can be applied to those data structures.

Question sources

MOE Computer Science Exit Exam · Question 25-1

National Computer Science Exit Exam · Question 12

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 28

Question 92 | SM0092

What is the output of the following C++ fragment code? int a, b; a = 13; b = 9; while((20 && 0) && (a > b)) { cout << "Plants are our life"; }

A. Displays 'Plants are our life' infinitely

B. No message will be displayed

C. Displays 'Plants are our life' 20 times

D. Displays 'Plants are Our Life' 4 times

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: No message will be displayed

20 is logically true but 0 is false, so 20&&0 is false. Its conjunction with a>b remains false even though 13>9. The while body is never entered, hence no message; none of the positive repetition counts or infinite-output option fits. Module support: PDF p. 11, PDF p. 7.

Why each choice is right or wrong

A - Incorrect: "Displays 'Plants are our life' infinitely" does not match the rule or result derived here. 20 is logically true but 0 is false, so 20&&0 is false. Its conjunction with a>b remains false even though 13>9. The while body is never entered, hence no message; none of the positive repetition counts or infinite-output option fits.

B - Correct: "No message will be displayed" matches the derived result. 20 is logically true but 0 is false, so 20&&0 is false. Its conjunction with a>b remains false even though 13>9. The while body is never entered, hence no message; none of the positive repetition counts or infinite-output option fits.

C - Incorrect: "Displays 'Plants are our life' 20 times" does not match the rule or result derived here. 20 is logically true but 0 is false, so 20&&0 is false. Its conjunction with a>b remains false even though 13>9. The while body is never entered, hence no message; none of the positive repetition counts or infinite-output option fits.

D - Incorrect: "Displays 'Plants are Our Life' 4 times" does not match the rule or result derived here. 20 is logically true but 0 is false, so 20&&0 is false. Its conjunction with a>b remains false even though 13>9. The while body is never entered, hence no message; none of the positive repetition counts or infinite-output option fits.

Primary note proof

Study-note section CP10

> Relational and equality operators test values; && and || combine truth values; ! negates. In C++, zero converts to false and a nonzero number to true. Streams display bool as 0 or 1 unless boolalpha is set. With a=3,b=5, cout<<(a==b) prints 0.

Study-note section CP14

while(condition) checks before each body execution and may run zero times. do {body;} while(condition); runs the body once before checking; it repeats while the condition remains true. Use it when a menu or input attempt must occur at least once.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 11 | lines 5-13

Iteration/Looping Statement: Iteration is the repetition of a statement or block of statements in a program. - Loops have as objective to repeat a statement a certain number of times or while a condition is fulfilled. - C++ has three iteration statements: the while statement, the do. While statement, and the for statement. The while loop: Its format is: while (expression) {statement;} First expression (called the loop condition) is evaluated. If the outcome is nonzero then statement (called the loop body) is executed and the whole process is repeated. Otherwise, the loop is terminated.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

MOE Computer Science Exit Exam · Question 26-2

National Computer Science Exit Exam · Question 85

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 78

Question 93 | SM0204

In C++, what does the cout object typically represent?

A. Standard input stream

B. Keyboard input buffer

C. Error output stream

D. Standard output stream

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: Standard output stream

The module explicitly calls cout the standard output stream (p.6). Standard input is cin, and keyboard buffer/error stream do not name the cout role. Module support: PDF p. 6.

Why each choice is right or wrong

A - Incorrect: "Standard input stream" does not match the rule or result derived here. The module explicitly calls cout the standard output stream (p.6). Standard input is cin, and keyboard buffer/error stream do not name the cout role.

B - Incorrect: "Keyboard input buffer" does not match the rule or result derived here. The module explicitly calls cout the standard output stream (p.6). Standard input is cin, and keyboard buffer/error stream do not name the cout role.

C - Incorrect: "Error output stream" does not match the rule or result derived here. The module explicitly calls cout the standard output stream (p.6). Standard input is cin, and keyboard buffer/error stream do not name the cout role.

D - Correct: "Standard output stream" matches the derived result. The module explicitly calls cout the standard output stream (p.6). Standard input is cin, and keyboard buffer/error stream do not name the cout role.

Primary note proof

Study-note section CP28

> Include <iostream> and qualify std::cin/std::cout or introduce the names explicitly. cout is the standard output stream; << inserts values. cin is standard input; >> extracts formatted values. std::endl inserts a newline and flushes. C++ names are case sensitive: Cout, Int and std: :cout are not equivalent valid spellings of cout, int and std::cout.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 6 | lines 4-15

> 6. Communication through console cout (the standard output stream) o used to display information for the user Syntax: cout<<object; • Object can be variable or literals Example: int x=5; cout<<x cout<< “Hello there”; cin(the standard input stream): o used to accept information from the user. Syntax: cin>>variable name; Eg. cin>>x;

Question sources

Computer Science Exit Exam · Sene 2016 · Question 1-4

Computer Science Exit Exam · Sene 2016 · Question 19-2

Computer Science Exit Exam practice collection · Question 93

Question 94 | SM0209

What is the term for a named block of code that performs a specific task?

A. Condition

B. Variable

C. Function

D. Loop

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: Function

A function is defined as a block of code performing a specific task and called from a program point (p.14). A condition supplies a test, a variable stores data, and a loop provides repetition rather than the named task unit. Module support: PDF p. 14.

Why each choice is right or wrong

A - Incorrect: "Condition" does not match the rule or result derived here. A function is defined as a block of code performing a specific task and called from a program point (p.14). A condition supplies a test, a variable stores data, and a loop provides repetition rather than the named task unit.

B - Incorrect: "Variable" does not match the rule or result derived here. A function is defined as a block of code performing a specific task and called from a program point (p.14). A condition supplies a test, a variable stores data, and a loop provides repetition rather than the named task unit.

C - Correct: "Function" matches the derived result. A function is defined as a block of code performing a specific task and called from a program point (p.14). A condition supplies a test, a variable stores data, and a loop provides repetition rather than the named task unit.

D - Incorrect: "Loop" does not match the rule or result derived here. A function is defined as a block of code performing a specific task and called from a program point (p.14). A condition supplies a test, a variable stores data, and a loop provides repetition rather than the named task unit.

Primary note proof

Study-note section CP17

> A function groups a named task. C++ definition: int add(int a,int b){return a+b;}. Call it with add(x,y);, without parameter type declarations at the call site. Parameters are names in the definition; arguments are supplied expressions or values. A void function produces no return value but can have side effects and return; to exit early.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 14 | lines 21-33

> Function A function is a group of statements that structure our programs in a more modular way and executed when it is called from some point of the program. It is a block of code that performs a specific task. It is important to Dividing a complex problem into smaller chunks makes our program easy to understand and reusable. When we need function? - When you need to repeat the same process over and over in a program. - The function can be called many times but appears in the code once Function - Categories C++ functions can be divided into 2 categories: • pre-defined (standard function) / library function - Which is the definitions have been written and it is ready to be used. - User needs to include pre-defined header file (i.e., math.h, time.h,cmath) To use it: FunctionName (value) e.g., sqrt (4), floor (4.3)

Question sources

Computer Science Exit Exam · Sene 2016 · Question 3-2

Computer Science Exit Exam practice collection · Question 89

Question 95 | SM0217

What does the if-else statement allow you to do in C++?

A. Declare and define functions

B. Allocate memory dynamically

C. Execute a block of code only if a condition is true, otherwise execute another block

D. Repeat a block of code multiple times

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: Execute a block of code only if a condition is true, otherwise execute another block

The p.9 if-else rule evaluates a condition and executes statement1 when true, otherwise statement2. This matches C. It is not a declaration, allocation mechanism or repetition construct. Module support: PDF p. 9.

Why each choice is right or wrong

A - Incorrect: "Declare and define functions" does not match the rule or result derived here. The p.9 if-else rule evaluates a condition and executes statement1 when true, otherwise statement2. This matches C. It is not a declaration, allocation mechanism or repetition construct.

B - Incorrect: "Allocate memory dynamically" does not match the rule or result derived here. The p.9 if-else rule evaluates a condition and executes statement1 when true, otherwise statement2. This matches C. It is not a declaration, allocation mechanism or repetition construct.

C - Correct: "Execute a block of code only if a condition is true, otherwise execute another block" matches the derived result. The p.9 if-else rule evaluates a condition and executes statement1

when true, otherwise statement2. This matches C. It is not a declaration, allocation mechanism or repetition construct.

D - Incorrect: "Repeat a block of code multiple times" does not match the rule or result derived here. The p.9 if-else rule evaluates a condition and executes statement1 when true, otherwise statement2. This matches C. It is not a declaration, allocation mechanism or repetition construct.

Primary note proof

Study-note section CP12

> A standalone if may execute independently of the following if. An else binds to its nearest unmatched if; an else-if ladder executes the first true branch only. In nested if(c1){if(c2) S1; else S2;}, S2 requires c1 true and c2 false. Braces determine grouping.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 9 | lines 9-21

> Conditional Statements A. The if Statement Syntax: if (expression) statement; First expression is evaluated. If the outcome is nonzero (true) then statement is executed. Otherwise, nothing happens. if-else statement Syntax: if (expression) statement1; else statement2; First expression is evaluated. If the outcome is nonzero (true) then statement1 is executed. Otherwise, statement2 is executed.

Question sources

Computer Science Exit Exam · Sene 2016 · Question 6-1

Computer Science Exit Exam practice collection · Question 90

Question 96 | SM0231

In computer programming, what is the purpose of control structures such as loops and conditional statements?

A. To control the flow of program execution

B. To perform mathematical operations

C. To define the data types used in a program

D. To allocate memory for variables

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: To control the flow of program execution

The module says flow control determines what executes during a run; branches choose alternative paths and loops repeat computations (p.9). Thus these structures control execution flow, rather than merely defining types, allocating memory or computing arithmetic. Module support: PDF p. 9.

Why each choice is right or wrong

A - Correct: "To control the flow of program execution" matches the derived result. The module says flow control determines what executes during a run; branches choose alternative paths and loops repeat computations (p.9). Thus these structures control execution flow, rather than merely defining types, allocating memory or computing arithmetic.

B - Incorrect: "To perform mathematical operations" does not match the rule or result derived here. The module says flow control determines what executes during a run; branches choose alternative paths and loops repeat computations (p.9). Thus these structures control execution flow, rather than merely defining types, allocating memory or computing arithmetic.

C - Incorrect: "To define the data types used in a program" does not match the rule or result derived here. The module says flow control determines what executes during a run; branches choose alternative paths and loops repeat computations (p.9). Thus these structures control execution flow, rather than merely defining types, allocating memory or computing arithmetic.

D - Incorrect: "To allocate memory for variables" does not match the rule or result derived here. The module says flow control determines what executes during a run; branches choose alternative paths and loops repeat computations (p.9). Thus these structures control execution flow, rather than merely defining types, allocating memory or computing arithmetic.

Primary note proof

Study-note section CP12

> A standalone if may execute independently of the following if. An else binds to its nearest unmatched if; an else-if ladder executes the first true branch only. In nested if(c1){if(c2) S1; else S2;}, S2 requires c1 true and c2 false. Braces determine grouping.

Study-note section CP14

> while(condition) checks before each body execution and may run zero times. do {body;} while(condition); runs the body once before checking; it repeats while the condition remains true. Use it when a menu or input attempt must occur at least once.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 9 | lines 2-8

> Control Statements Control statements or flow control determines what is executed during a run and what is not, therefore affecting the overall outcome of the program. C++ provides different forms of statements for different purposes. Declaration statements are used for defining variables. Assignment-like statements are used for simple, algebraic computations. Branching statements are used for specifying alternate paths of execution, depending on the outcome of a

logical condition. Loop statements are used for specifying computations, which need to be repeated until a certain logical condition is satisfied

Question sources

Computer Science Exit Exam · Sene 2016 · Question 10-2

Computer Science Exit Exam practice collection · Question 91

Question 97 | SM0236

When encountering a syntax error in code, what is the first step you should take to resolve it?

A. Rewrite the entire

B. Ignore the error and continue execution

C. Usea debugger tool

D. Check the code for typos or incorrect syntax

Hint

Apply the naming, type and syntax rules before deciding whether an option can be used as stated.

Answer D: Check the nearby code for typos and incorrect syntax.

CP32 recommends reading the first useful compiler diagnostic and inspecting nearby tokens and declarations before repairing and rebuilding. Among the offered actions, checking for typos or incorrect syntax matches that early repair step. Ignoring the error cannot fix translation, and a debugger is most useful once a runnable program exists.

Why each choice is right or wrong

A - Incorrect: Rewriting everything is unnecessary; the note recommends diagnosing and repairing the relevant local code.

B - Incorrect: Ignoring a syntax error neither corrects grammar nor establishes that an executable can be built.

C - Incorrect: A debugger is useful after a runnable program exists; compiler diagnostics guide syntax repair first.

D - Correct: Inspecting tokens and declarations for syntax mistakes matches the first useful action among these options.

Primary note proof

Study-note section CP32

A syntax error violates grammar; a type error can be syntactically well formed but rejected during semantic checking. A runtime error occurs during execution; a logic error gives the wrong behavior. A bug is a defect; debugging locates and corrects it. Read the first useful compiler diagnostic, inspect nearby tokens and declarations, repair and rebuild; a debugger is useful after a runnable program exists.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

> Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Question sources

Computer Science Exit Exam · Sene 2016 · Question 12-1

Question 98 | SM0276

Which C++ statement is used to terminate a loop prematurely and transfer control to the next iteration of the loop?

A. break

B. exit

C. return

D. continue

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: continue

The phrase transfer control to the next iteration describes continue, which skips remaining statements of the current iteration and preserves the loop (p.13). break exits the loop entirely; return leaves a function; exit is not the loop-iteration statement. Module support: PDF p. 13.

Why each choice is right or wrong

A - Incorrect: "break" does not match the rule or result derived here. The phrase transfer control to the next iteration describes continue, which skips remaining statements of the current iteration and preserves the loop (p.13). break exits the loop entirely; return leaves a function; exit is not the loop-iteration statement.

B - Incorrect: "exit" does not match the rule or result derived here. The phrase transfer control to the next iteration describes continue, which skips remaining statements of the current iteration and preserves the loop (p.13). break exits the loop entirely; return leaves a function; exit is not the loop-iteration statement.

C - Incorrect: "return" does not match the rule or result derived here. The phrase transfer control to the next iteration describes continue, which skips remaining statements of the current

iteration and preserves the loop (p.13). break exits the loop entirely; return leaves a function; exit is not the loop-iteration statement.

D - Correct: "continue" matches the derived result. The phrase transfer control to the next iteration describes continue, which skips remaining statements of the current iteration and preserves the loop (p.13). break exits the loop entirely; return leaves a function; exit is not the loop-iteration statement.

Primary note proof

Study-note section CP15

> break exits the nearest enclosing loop or switch. continue skips the rest of the current loop iteration: a for loop then runs its update; a while loop goes directly to its condition. return leaves the current function and may supply a result. In main it ends the program normally.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 13 | lines 30-35

> Using continue and break Statements - The break statement skips the rest of the statements in the loop‟s block, jumping immediately to the next statement outside of the loop. - The continue statement is similar. It also skips the rest of the statements in the loop‟s block, but instead of terminating the loop, it transfers execution to the next iteration of the loop. It continues the loop after skipping the remaining statements in its current iteration.

Question sources

Computer Science Exit Exam · Sene 2016 · Question 25-1

Computer Science Exit Exam practice collection · Question 94

Question 99 | SM0288

Which is a desirable performance quality rather than a necessary defining requirement for a useful algorithm with a specified result on every legal input?

A. Definiteness

B. Output

C. Finiteness

D. Efficiency

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: D. Efficiency

The note distinguishes the existence of an algorithm from how efficiently it runs. Its instructions must be unambiguous, it must specify a result, and it must terminate for legal inputs. An inefficient procedure can still satisfy all these defining requirements and correctly solve its problem. Efficiency is therefore desirable and measurable but is the exception when the question asks specifically about necessary defining requirements.

Why each choice is right or wrong

A - Incorrect: Definiteness requires unambiguous instructions.

B - Incorrect: A useful algorithm specifies a result.

C - Incorrect: Finiteness requires termination for legal inputs.

D - Correct: Efficiency is desirable but an inefficient terminating algorithm still qualifies.

Primary note proof

Study-note section CP02

> An algorithm gives precise, executable steps and terminates for every legal input. Definiteness means each step has an unambiguous meaning; effectiveness means a step can actually be performed. Inputs may be absent, but a useful algorithm specifies a result. Correctness means the result meets the specification. Efficiency is desirable and measurable, but an inefficient terminating algorithm is still an algorithm. Efficiency is also a desirable property of an algorithm.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 2 | lines 19-26

> Algorithm is an ordered set of unambiguous, executable steps, defining a terminating process. Important Properties of Algorithms - Correct: always returns the desired output for all legal instances of the problem. - Unambiguous: Each step must be clearly defined, having only one interpretation. - Precise: simple but efficient - Efficient: can be measured in terms of Time and Space - Finiteness: the execution of the algorithm must terminate in a finite amount of time for each possible set of input data/must have an end.

Question sources

Computer Science Exit Exam · Sene 2016 · Question 29-1

Computer Science Exit Exam practice collection · Question 119

Question 100 | SS049

What is the output of the following C++ fragment code? for(int n = 1; n <= 18; n = n + 2) { if(n % 7 != 0) cout << n << " "; else break; }

A. 3 5 7

B. 1 3 5 7 9 11 13 15 17

C. 1 3 5

D. 1 2 3 4 5 6

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer C: 1 3 5

The loop prints n=1,3,5 because each has nonzero remainder modulo 7. At n=7 the remainder is zero, so break exits before printing 7. A includes a forbidden 7 and omits 1; B ignores break; D ignores the +2 update. Module support: PDF p. 12, PDF p. 13, PDF p. 7.

Why each choice is right or wrong

A - Incorrect: "3 5 7" does not match the rule or result derived here. The loop prints n=1,3,5 because each has nonzero remainder modulo 7. At n=7 the remainder is zero, so break exits before printing 7. A includes a forbidden 7 and omits 1; B ignores break; D ignores the +2 update.

B - Incorrect: "1 3 5 7 9 11 13 15 17" does not match the rule or result derived here. The loop prints n=1,3,5 because each has nonzero remainder modulo 7. At n=7 the remainder is zero, so break exits before printing 7. A includes a forbidden 7 and omits 1; B ignores break; D ignores the +2 update.

C - Correct: "1 3 5" matches the derived result. The loop prints n=1,3,5 because each has nonzero remainder modulo 7. At n=7 the remainder is zero, so break exits before printing 7. A includes a forbidden 7 and omits 1; B ignores break; D ignores the +2 update.

D - Incorrect: "1 2 3 4 5 6" does not match the rule or result derived here. The loop prints n=1,3,5 because each has nonzero remainder modulo 7. At n=7 the remainder is zero, so break exits before printing 7. A includes a forbidden 7 and omits 1; B ignores break; D ignores the +2 update.

Primary note proof

Study-note section CP16

> For n=1; n<=18; n+=2, print n unless n%7==0 then break: output is 1 3 5. A while(a>b) with fixed a=6,b=5 never changes its true condition. In data[10]={5,10,...,50}, i=0, advancing i only if data[i]%10==0 causes no progress because data[0]=5. It prints nothing and loops indefinitely; changing <=10 to <10 alone does not repair progress.

Study-note section CP15

break exits the nearest enclosing loop or switch. continue skips the rest of the current loop iteration: a for loop then runs its update; a while loop goes directly to its condition. return leaves the current function and may supply a result. In main it ends the program normally.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 12 | lines 19-28

The for loop. Its format is: for (initialization; condition; increase) statement; 1, initialization is executed. Generally, it is a initial value setting for a counter variable. This is executed only once 2, condition is checked, if it is true the loop continues, otherwise the loop finishes and statement is skipped. 3, statement is executed. As usual, it can be either a single instruction or a block of instructions enclosed within curly brackets { }. 4, finally, whatever is specified in the increase field is executed and the loop gets back to step 2.

Computer Programming module | PDF page 13 | lines 30-35

Using continue and break Statements - The break statement skips the rest of the statements in the loop‟s block, jumping immediately to the next statement outside of the loop. - The continue statement is similar. It also skips the rest of the statements in the loop‟s block, but instead of terminating the loop, it transfers execution to the next iteration of the loop. It continues the loop after skipping the remaining statements in its current iteration.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

National Computer Science Exit Exam · Question 48

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 73

Question 101 | SS160

For input b=4 and e=3, what numeric result is printed after the prompt by this repaired C++ program?

```
#include <iostream>
using namespace std;
int main(){
int b,e,r=1;
cout<<"Enter b and e";
cin>>b>>e;
for(int i=1;i<=e;i++){r=r*b;}
cout<<r;
return 0;
}
```

A. 81

B. 256

C. 12

D. 64

Hint

Fix the input case and cost model, then count dominant work or simultaneously live storage.

Answer D: D. 64

The accumulator r starts at one. With exponent e=3, the for loop executes for i=1, 2 and 3, multiplying by the base b=4 on each pass. Its successive values are 4, 16 and 64. The final insertion therefore prints 64 after the prompt. This is iterative exponentiation, rather than multiplying the base by the exponent or reversing their roles.

Why each choice is right or wrong

A - Incorrect: 81 is 3 raised to 4, reversing base and exponent.

B - Incorrect: 256 would require four multiplications by 4.

C - Incorrect: 12 multiplies base by exponent rather than computing the power.

D - Correct: Three multiplications by 4 starting at 1 give 64.

Primary note proof

Study-note section CP30

> C++ has no built-in exponentiation operator: ^ is bitwise XOR and ** is not a power operator. To compute b^e for a nonnegative integer e, start r=1 and repeat r*=b exactly e times; b=4,e=3 gives 64. Large powers and factorials can overflow fixed-width types; specify a valid input range.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 12 | lines 19-28

The for loop. Its format is: for (initialization; condition; increase) statement; 1, initialization is executed. Generally, it is a initial value setting for a counter variable. This is executed only once 2, condition is checked, if it is true the loop continues, otherwise the loop finishes and statement is skipped. 3, statement is executed. As usual, it can be either a single instruction or a block of instructions enclosed within curly brackets { }. 4, finally, whatever is specified in the increase field is executed and the loop gets back to step 2.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Computer Science model exam collection · Question 67

Question 102 | SS167

A program reads exactly 50 numbers and adds only positive values to a total. Which listed construct best expresses the known repetition count, while a separate if statement handles the positivity test?

A. for

B. break

C. jump

D. continue

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: A. for

The task has a counting responsibility and a filtering responsibility. A for loop naturally expresses the fixed total of 50 input attempts through its initialization, condition and update. The question assigns the positivity decision to a separate if, so it asks specifically about counting. continue can skip one contribution but cannot supply the repetition count; break would stop early, and jump is not a specific count-controlled construct.

Why each choice is right or wrong

A - Correct: for naturally expresses exactly 50 iterations.

B - Incorrect: break could stop processing before all 50 inputs.

C - Incorrect: Jump does not specify a count-controlled loop.

D - Incorrect: continue skips the rest of an iteration but does not establish 50 repetitions.

Primary note proof

Study-note section CP23

> For a sum, start total=0 and add each element once. For a maximum/minimum, initialize from a[0] only after confirming the array is nonempty. For 50 inputs where nonpositive values are skipped, use a count-controlled loop and either if(x>0) total+=x; or continue for x<=0. These constructs solve different parts of the task.

Study-note section CP14

> for(initialization; condition; update) executes initialization once, then condition → body → update → condition. A missing condition in for(;;) acts as true. A for loop is convenient for a known count but can also be condition controlled. To print indices 0 through n-1, use i=0; i<n; ++i. Starting i=0 with i<5 and i+=2 prints 0 2 4.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 12 | lines 19-28

> The for loop. Its format is: for (initialization; condition; increase) statement; 1, initialization is executed. Generally, it is a initial value setting for a counter variable. This is executed only once 2, condition is checked, if it is true the loop continues, otherwise the loop finishes and statement is skipped. 3, statement is executed. As usual, it can be either a single instruction or a block of instructions enclosed within curly brackets { }. 4, finally, whatever is specified in the increase field is executed and the loop gets back to step 2.

Computer Programming module | PDF page 13 | lines 30-35

> Using continue and break Statements - The break statement skips the rest of the statements in the loop‟s block, jumping immediately to the next statement outside of the loop. - The continue statement is similar. It also skips the rest of the statements in the loop‟s block, but instead of terminating the loop, it transfers execution to the next iteration of the loop. It continues the loop after skipping the remaining statements in its current iteration.

Question sources

Computer Science model exam collection · Question 74

Question 103 | SS173

Which sequence of array values is printed by this Java code? Ignore each println call's trailing space and line separator.

```
class PrintResult {
public static void main(String[] args){
int[] arr={3,4,5,6,7};
for(int i=0;i<arr.length-2;++i){
System.out.println(arr[i]+" ");
}
}
}
```

A. 3 4 5 6 7

B. 3 4 5 6

C. 3 4

D. 3 4 5

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: D. 3 4 5

The initializer contains five values, so arr.length-2 is 3. The loop starts at index zero and continues while i<3, visiting indices 0, 1 and 2. Those positions contain 3, 4 and 5. Each println also ends its line, but the question explicitly asks only for the value sequence.

Why each choice is right or wrong

A - Incorrect: The loop never reaches indices 3 and 4.

B - Incorrect: Four values would require the excluded index 3.

C - Incorrect: Index 2 also satisfies i<3, so a third value prints.

D - Correct: Indices 0,1,2 contain 3,4,5 in that order.

Primary note proof

Study-note section CP23

> In Java, length belongs to each array; a traversal ending at arr.length-2 visits fewer elements than a full traversal.

Study-note section CP22

> Valid indices are 0 through 3.

Study-note section CP14

> for(initialization; condition; update) executes initialization once, then condition → body → update → condition.

Study-note section CP-S04

Java System.out.print writes the supplied value without automatically ending the line. System.out.println writes its value and then a line separator. Successive println calls therefore place their values on separate lines, even when each printed expression also contains a trailing space.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 19 | lines 4-29

Arrays - an array is a collection of fixed-size number of elements of the same type stored sequentially in memory. - An individual element of an array is identified by its own unique index (or subscript). - The number of elements must be an integer. - The size of the array is referred to as its dimension - Array in C++ is Zero indexed. Declaring of one-dimensional Array Syntax: elementstype arrayname [size]; → You can use constants such as macro definitions or const ints as the sizes of arrays. Example: int num[5]; string fruit[3]; #define numstudents 100 or const int numstudents=100; double Score[numstudents] ; double Score[numstudents] ; Initialization of an Array - Arrays can be initialized at declaration time in two different ways: Syntax: elementstype arrayname[size] = {list each value separated by comma}; E.g int num[5]={1,2,3,4,5}; elementstype arrayname[ ] = { list with any number of values }; E.g int num[]={1,2,3,4,5}; string fruit[]={“Apple”,“orange”, “Mango”}; Accessing Array Elements - Each element could be accessed using an index into the array - When accessing an array, the index given must be a positive integer from 0 to n-1, where n is the dimension of the array.

Computer Programming module | PDF page 12 | lines 19-28

The for loop. Its format is: for (initialization; condition; increase) statement; 1, initialization is executed. Generally, it is a initial value setting for a counter variable. This is executed only once 2, condition is checked, if it is true the loop continues, otherwise the loop finishes and statement is skipped. 3, statement is executed. As usual, it can be either a single instruction or a block of instructions enclosed within curly brackets { }. 4, finally, whatever is specified in the increase field is executed and the loop gets back to step 2.

Question sources

Computer Science model exam collection · Question 80

Question 104 | SS177

Which activity comes last in fundamental program development in c++ ?

A. Compiling program

B. Writing program

C. Memory allocation for variables

D. Linking

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: Linking

Among the listed activities, linking comes after writing/editing and compilation in the module sequence edit source -> compile -> link -> execute (p.4). Memory allocation for variables is not the later named development phase. Execution/loading occurs afterward, but is not an offered option. Module support: PDF p. 4.

Why each choice is right or wrong

A - Incorrect: "Compiling program" does not match the rule or result derived here. Among the listed activities, linking comes after writing/editing and compilation in the module sequence edit source -> compile -> link -> execute (p.4). Memory allocation for variables is not the later named development phase. Execution/loading occurs afterward, but is not an offered option.

B - Incorrect: "Writing program" does not match the rule or result derived here. Among the listed activities, linking comes after writing/editing and compilation in the module sequence edit source -> compile -> link -> execute (p.4). Memory allocation for variables is not the later named development phase. Execution/loading occurs afterward, but is not an offered option.

C - Incorrect: "Memory allocation for variables" does not match the rule or result derived here. Among the listed activities, linking comes after writing/editing and compilation in the module sequence edit source -> compile -> link -> execute (p.4). Memory allocation for variables is not the later named development phase. Execution/loading occurs afterward, but is not an offered option.

D - Correct: "Linking" matches the derived result. Among the listed activities, linking comes after writing/editing and compilation in the module sequence edit source -> compile -> link -> execute (p.4). Memory allocation for variables is not the later named development phase. Execution/loading occurs afterward, but is not an offered option.

Primary note proof

Study-note section CP04

> In the conventional C++ toolchain: source → preprocessing → compilation proper → assembly → object files → linking → executable → loading → execution. Preprocessing handles directives such as #include; compilation may emit assembly; the assembler emits machine instructions in an object file. The linker resolves external symbols and combines object files and libraries. The loader places an executable and required components in memory and starts it.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 4 | lines 21-30

> The compilation process of C++ program consists of the following steps. • edit source code → compile → link → execute (Editor) (Compiler) (Linker) (Loader) I. Program source code is entered into a file using a text editor. II. After the code has been entered, a compiler program is started

that translates the source into an object code file. III. The object code file is linked with other object code files that come with the compiler and an executable file (or program) is created. IV. In order to execute the program, a program called the loader copies the executable file into the memory of the computer and sends an execute command to the CPU.

Question sources

Computer Science model exam collection · Question 84

Question 105 | SS179

The correct structure of the for loop statement is:
eooo
ao
5

A. for(condition, initialization, increment/decrement)

B. for(initialization; condition)

C. for(increment/decrement; initialization; condition)

D. for(initialization; condition; increment/decrement)

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: for(initialization; condition; increment/decrement)

The module gives for(initialization; condition; increase) statement (p.12). D has this order and two semicolons. A uses commas and wrong order, B omits the update field and its separator, and C reverses the roles. Module support: PDF p. 12.

Why each choice is right or wrong

A - Incorrect: "for(condition, initialization, increment/decrement)" does not match the rule or result derived here. The module gives for(initialization; condition; increase) statement (p.12). D has this order and two semicolons. A uses commas and wrong order, B omits the update field and its separator, and C reverses the roles.

B - Incorrect: "for(initialization; condition)" does not match the rule or result derived here. The module gives for(initialization; condition; increase) statement (p.12). D has this order and two semicolons. A uses commas and wrong order, B omits the update field and its separator, and C reverses the roles.

C - Incorrect: "for(increment/decrement; initialization; condition)" does not match the rule or result derived here. The module gives for(initialization; condition; increase) statement (p.12). D has this order and two semicolons. A uses commas and wrong order, B omits the update field and its separator, and C reverses the roles.

D - Correct: "for(initialization; condition; increment/decrement)" matches the derived result. The module gives for(initialization; condition; increase) statement (p.12). D has this order and two semicolons. A uses commas and wrong order, B omits the update field and its separator, and C reverses the roles.

Primary note proof

Study-note section CP14

> for(initialization; condition; update) executes initialization once, then condition → body → update → condition. A missing condition in for(;;) acts as true. A for loop is convenient for a known count but can also be condition controlled. To print indices 0 through n-1, use i=0; i<n; ++i. Starting i=0 with i<5 and i+=2 prints 0 2 4.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 12 | lines 19-28

> The for loop. Its format is: for (initialization; condition; increase) statement; 1, initialization is executed. Generally, it is a initial value setting for a counter variable. This is executed only once 2, condition is checked, if it is true the loop continues, otherwise the loop finishes and statement is skipped. 3, statement is executed. As usual, it can be either a single instruction or a block of instructions enclosed within curly brackets { }. 4, finally, whatever is specified in the increase field is executed and the loop gets back to step 2.

Question sources

Computer Science model exam collection · Question 86

Question 106 | SS180

Which one of the following is the correct identifier in c++?

A. 7variable

B. 7VARIABLE

C. $variable

D. variable_1234

Hint

Apply the naming, type and syntax rules before deciding whether an option can be used as stated.

Answer D: variable_1234

variable_1234 begins with a letter and contains only letters, underscore and digits. A/B begin with 7, violating the initial-character rule, while C uses $, outside the permitted character set. Module support: PDF p. 5.

Why each choice is right or wrong

A - Incorrect: "7variable" does not match the rule or result derived here. variable_1234 begins with a letter and contains only letters, underscore and digits. A/B begin with 7, violating the initial-character rule, while C uses $, outside the permitted character set.

B - Incorrect: "7VARIABLE" does not match the rule or result derived here. variable_1234 begins with a letter and contains only letters, underscore and digits. A/B begin with 7, violating the initial-character rule, while C uses $, outside the permitted character set.

C - Incorrect: "$variable" does not match the rule or result derived here. variable_1234 begins with a letter and contains only letters, underscore and digits. A/B begin with 7, violating the initial-character rule, while C uses $, outside the permitted character set.

D - Correct: "variable_1234" matches the derived result. variable_1234 begins with a letter and contains only letters, underscore and digits. A/B begin with 7, violating the initial-character rule, while C uses $, outside the permitted character set.

Primary note proof

Study-note section CP05

> In the basic ASCII identifier rules, start with a letter or underscore; continue with letters, digits or underscores. Do not use a keyword. C++ is case sensitive: exitExam and ExitExam differ; delete and float are keywords, but FLOAT is a legal identifier. variable_1234 is valid; 4thInput, T-3 and names containing spaces are not single identifiers. Avoid reserved underscore patterns in application code.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 2-10

> 1. Keywords (reserved words) - Unique meaning within a C++ program and must not be used for any other purposes. E.g., if, namespace for, switch, continue, float, new, default, break Etc. 2. Identifiers - Name associated with a function or data object and used to refer to that function or data object. An identifier must: o Start with a letter or underscore o Consist only of letters, the digits 0-9, or the underscore symbol _ o Not be a reserved word

Question sources

Computer Science model exam collection · Question 87

Question 107 | AM0245

Which of the following is NOT a programming paradigm?

A. Procedural

B. Object-oriented

C. Functional

D. Conditional

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: Conditional is a construct rather than a programming paradigm.

CP02 distinguishes procedural, object oriented and functional programming as ways of organizing computation. A conditional is a selection construct used inside programs rather than a separate paradigm in this classification. Thus the first three options name paradigms, while Conditional is the requested exception.

Why each choice is right or wrong

A - Incorrect: Procedural programming is a paradigm organizing operations into procedures, as described in CP02.

B - Incorrect: Object oriented programming is a paradigm organizing state and behavior in objects.

C - Incorrect: Functional programming is a paradigm emphasizing function evaluation, as described in CP02.

D - Correct: Conditional denotes a selection construct used within paradigms rather than a separate paradigm here.

Primary note proof

Study-note section CP02

> Procedural programming organizes operations in procedures; structured programming combines sequence, selection and iteration; object oriented programming organizes state and behavior in objects; functional programming emphasizes function evaluation. A conditional statement is a construct, not a separate programming paradigm.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 4 | lines 4-19

> programming paradigm: provides the programmer's view of code execution. The most influential paradigms are: Procedural Programming Languages Procedural programming specifies a list of operations that the program must complete to reach the desired state. Each program has a starting state, a list of operations to complete, and an ending point. This approach is also known as imperative programming. Structured Programming Languages Structured programming is a special type of procedural programming. It provides additional tools to manage the problems that larger programs were creating. Structured programming requires that programmers break program structure into small pieces of code that are easily understood. One of the well-known features of structural programming is that it does not allow the use of the GOTO statement. It is often associated with a "top-down" approach to design. Object-Oriented Programming Languages Object-oriented programming is one the newest and most powerful paradigms. In object- oriented programs, the designer specifies both the data structures and the types of operations that can be applied to those data structures.

Question sources

Computer Science MCQ practice collection · Question 245

Question 108 | AM0246

What is the purpose of a compiler?

A. To execute programs

B. To debug programs

C. To translate source code into machine code

D. To optimize program performance

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: To translate source code into machine code

The module describes translating program source before execution, producing object code for linking and later loading. This supports source-to-machine translation as the compiler purpose. Execution belongs later in the pipeline; debugging and optimizing are not its defining role here. Module support: PDF p. 4, PDF p. 3.

Why each choice is right or wrong

A - Incorrect: "To execute programs" does not match the rule or result derived here. The module describes translating program source before execution, producing object code for linking and later loading. This supports source-to-machine translation as the compiler purpose. Execution belongs later in the pipeline; debugging and optimizing are not its defining role here.

B - Incorrect: "To debug programs" does not match the rule or result derived here. The module describes translating program source before execution, producing object code for linking and later loading. This supports source-to-machine translation as the compiler purpose. Execution belongs later in the pipeline; debugging and optimizing are not its defining role here.

C - Correct: "To translate source code into machine code" matches the derived result. The module describes translating program source before execution, producing object code for linking and later loading. This supports source-to-machine translation as the compiler purpose. Execution belongs later in the pipeline; debugging and optimizing are not its defining role here.

D - Incorrect: "To optimize program performance" does not match the rule or result derived here. The module describes translating program source before execution, producing object code for linking and later loading. This supports source-to-machine translation as the compiler purpose. Execution belongs later in the pipeline; debugging and optimizing are not its defining role here.

Primary note proof

Study-note section CP04

In the conventional C++ toolchain: source → preprocessing → compilation proper → assembly → object files → linking → executable → loading → execution. Preprocessing handles directives such as #include; compilation may emit assembly; the assembler emits machine instructions in an object file. The linker resolves external symbols and combines object files and libraries. The loader places an executable and required components in memory and starts it.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 4 | lines 21-30

The compilation process of C++ program consists of the following steps. • edit source code → compile → link → execute (Editor) (Compiler) (Linker) (Loader) I. Program source code is entered into a file using a text editor. II. After the code has been entered, a compiler program is started that translates the source into an object code file. III. The object code file is linked with other object code files that come with the compiler and an executable file (or program) is created. IV. In order to execute the program, a program called the loader copies the executable file into the memory of the computer and sends an execute command to the CPU.

Computer Programming module | PDF page 3 | lines 8-17

Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Question sources

Computer Science MCQ practice collection · Question 246

Question 109 | AM0247

What is the main advantage of high-level programming languages?

A. They are easier to read and write than low-level languages.

B. They are faster and more efficient than low-level languages.

C. They provide direct control over hardware resources.

D. They are platform-specific and provide better compatibility.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: High-level languages are easier to read and write.

CP03 says high-level source is easier to read and often portable after recompilation when platform assumptions are avoided. Readability is the advantage stated in A. The note does not promise universally faster execution or greater hardware control, and platform specificity conflicts with the portability advantage rather than explaining it.

Why each choice is right or wrong

A - Correct: Easier source reading and writing is the abstraction benefit directly supported by CP03.

B - Incorrect: The note gives no universal speed or efficiency guarantee for high-level languages over low-level ones.

C - Incorrect: Direct hardware control is not the high-level readability advantage described here.

D - Incorrect: Platform specificity contradicts the potential source portability discussed in CP03.

Primary note proof

Study-note section CP03

> Machine and assembly code depend on an instruction set. High-level source is easier to read and often portable after recompilation, provided it avoids platform assumptions. People can write both assembly and high-level programs. Python, Perl and PHP are commonly called scripting languages; C++ is conventionally compiled. These labels describe common use, not an absolute technical prohibition.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

> Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Question sources

Computer Science MCQ practice collection · Question 247

Question 110 | AM0248

Which option most precisely states the information a C++ variable declaration introduces, independently of whether that declaration also defines storage or supplies an initializer?

A. To assign a value to a variable

B. To create a new variable

C. To specify the data type and name of a variable

D. To check the validity of a variable

Hint

Apply the naming, type and syntax rules before deciding whether an option can be used as stated.

Answer C: C. To specify the data type and name of a variable

Declaration introduces a variable's name and type into the program. Initialization is the separate act of supplying its initial value, and later assignment replaces a value. Describing a declaration merely as creating a variable does not precisely identify its information or distinguish it from an object definition. The question asks for that precise name-and-type purpose, which C directly states.

Why each choice is right or wrong

A - Incorrect: Assigning a value is a different operation from introducing a declaration.

B - Incorrect: Creating an object is less precise and can depend on whether the declaration is also a definition.

C - Correct: A declaration introduces the variable's name and type.

D - Incorrect: A declaration is not primarily a validity check on an existing variable.

Primary note proof

Study-note section CP06

> Write type name; or type name = value; for ordinary variables: int age = 20;. Declaration introduces a name and type; initialization gives its initial value; later assignment replaces a value. a=b copies a scalar value, so later changes to b do not change a. A pointer or Java object reference instead copies an address/reference.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 24-33

> 5. Data Types, Variables, and Constants Variable: - Variable is a symbolic name for a memory location in which data can be stored. - Attribute of Variables: o A type, which is, established when the variable is defined (e.g., integer, float, character) o A value, which can be changed by assigning a new value to the variable Variable Declaration - Declaring a variable means defining (creating) a variable. - Syntax: data type variable name; e.g. int num; - We can create More Than One Variable at a Time e.g., int num1, num2, num3;

Question sources

Computer Science MCQ practice collection · Question 248

Question 111 | AM0249

What is the purpose of a loop in programming?

A. To perform mathematical calculations

B. To make decisions based on conditions

C. To store and retrieve data

D. To repeat a set of instructions

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: To repeat a set of instructions

Iteration repeats a statement or block, and loops specify computations repeated while/until a logical condition controls continuation (pp.9,11). Arithmetic, decision-making and storage describe different roles, so D is the specific loop purpose. Module support: PDF p. 9, PDF p. 11.

Why each choice is right or wrong

A - Incorrect: "To perform mathematical calculations" does not match the rule or result derived here. Iteration repeats a statement or block, and loops specify computations repeated while/until a logical condition controls continuation (pp.9,11). Arithmetic, decision-making and storage describe different roles, so D is the specific loop purpose.

B - Incorrect: "To make decisions based on conditions" does not match the rule or result derived here. Iteration repeats a statement or block, and loops specify computations repeated while/until a logical condition controls continuation (pp.9,11). Arithmetic, decision-making and storage describe different roles, so D is the specific loop purpose.

C - Incorrect: "To store and retrieve data" does not match the rule or result derived here. Iteration repeats a statement or block, and loops specify computations repeated while/until a logical condition controls continuation (pp.9,11). Arithmetic, decision-making and storage describe different roles, so D is the specific loop purpose.

D - Correct: "To repeat a set of instructions" matches the derived result. Iteration repeats a statement or block, and loops specify computations repeated while/until a logical condition controls continuation (pp.9,11). Arithmetic, decision-making and storage describe different roles, so D is the specific loop purpose.

Primary note proof

Study-note section CP14

while(condition) checks before each body execution and may run zero times. do {body;} while(condition); runs the body once before checking; it repeats while the condition remains true. Use it when a menu or input attempt must occur at least once.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 9 | lines 2-8

Control Statements Control statements or flow control determines what is executed during a run and what is not, therefore affecting the overall outcome of the program. C++ provides different forms of statements for different purposes. Declaration statements are used for defining variables. Assignment-like statements are used for simple, algebraic computations. Branching statements are used for specifying alternate paths of execution, depending on the outcome of a logical condition. Loop statements are used for specifying computations, which need to be repeated until a certain logical condition is satisfied

Computer Programming module | PDF page 11 | lines 5-13

Iteration/Looping Statement: Iteration is the repetition of a statement or block of statements in a program. - Loops have as objective to repeat a statement a certain number of times or while a condition is fulfilled. - C++ has three iteration statements: the while statement, the do. While statement, and the for statement. The while loop: Its format is: while (expression) {statement;} First expression (called the loop condition) is evaluated. If the outcome is nonzero then statement (called the loop body) is executed and the whole process is repeated. Otherwise, the loop is terminated.

Question sources

Computer Science MCQ practice collection · Question 249

Question 112 | AM0250

Which statement gives the defining relationship between a method and a function?

A. A method is a function associated with a class or object.

B. Functions return a value, while methods do not.

C. Functions can take parameters, while methods cannot.

D. There is no difference; the terms are used interchangeably.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: A. A method is a function associated with a class or object.

The defining feature of a method is its association with a class or object. It is therefore a function with that association, rather than a fundamentally different operation with no return value or no parameters. Both functions and methods can return values or accept arguments according to their definitions. The two terms are related, but method adds the class/object qualification that ordinary function alone does not assert.

Why each choice is right or wrong

A - Correct: A method is a function associated with a class or object.

B - Incorrect: Methods can return values, and functions may return void.

C - Incorrect: Both ordinary functions and methods can take parameters.

D - Incorrect: Method adds a specific class/object association.

Primary note proof

Study-note section CP17

> A method is a function associated with a class or object; methods can return values. Modules group related responsibilities behind interfaces. Reuse reduces duplication; independent functions are easier to understand and test. A library provides reusable components; a framework also supplies an organizing structure and often calls application callbacks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 14 | lines 21-33

> Function A function is a group of statements that structure our programs in a more modular way and executed when it is called from some point of the program. It is a block of code that performs a specific task. It is important to Dividing a complex problem into smaller chunks makes our program easy to understand and reusable. When we need function? - When you need to repeat the same process over and over in a program. - The function can be called many times but appears in the code once Function - Categories C++ functions can be divided into 2 categories: • pre-defined (standard function) / library function - Which is the definitions have been written and it is ready to be used. - User needs to include pre-defined header file (i.e., math.h, time.h,cmath) To use it: FunctionName (value) e.g., sqrt (4), floor (4.3)

Computer Programming module | PDF page 4 | lines 4-19

> programming paradigm: provides the programmer's view of code execution. The most influential paradigms are: Procedural Programming Languages Procedural programming specifies a list of operations that the program must complete to reach the desired state. Each program has a starting state, a list of operations to complete, and an ending point. This approach is also known as imperative programming. Structured Programming Languages Structured programming is a special type of procedural programming. It provides additional tools to manage the problems that larger programs were creating. Structured programming requires that programmers break program structure into small pieces of code that are easily understood. One of the well-known features of structural programming is that it does not allow the use of the

GOTO statement. It is often associated with a "top-down" approach to design. Object-Oriented Programming Languages Object-oriented programming is one the newest and most powerful paradigms. In object- oriented programs, the designer specifies both the data structures and the types of operations that can be applied to those data structures.

Question sources

Computer Science MCQ practice collection · Question 250

Question 113 | AM0251

Which type of error occurs when the program violates the rules of the programming language?

A. Syntax error

B. Runtime error

C. Logic error

D. Semantic error

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: Syntax error

Syntax is defined as rules governing language symbols, whereas semantics deals with meaning of syntactically correct constructs (p.3). Therefore violating the stated language rules is a syntax error. Runtime and logic problems are not the rule category described. Module support: PDF p. 3.

Why each choice is right or wrong

A - Correct: "Syntax error" matches the derived result. Syntax is defined as rules governing language symbols, whereas semantics deals with meaning of syntactically correct constructs (p.3). Therefore violating the stated language rules is a syntax error. Runtime and logic problems are not the rule category described.

B - Incorrect: "Runtime error" does not match the rule or result derived here. Syntax is defined as rules governing language symbols, whereas semantics deals with meaning of syntactically correct constructs (p.3). Therefore violating the stated language rules is a syntax error. Runtime and logic problems are not the rule category described.

C - Incorrect: "Logic error" does not match the rule or result derived here. Syntax is defined as rules governing language symbols, whereas semantics deals with meaning of syntactically correct constructs (p.3). Therefore violating the stated language rules is a syntax error. Runtime and logic problems are not the rule category described.

D - Incorrect: "Semantic error" does not match the rule or result derived here. Syntax is defined as rules governing language symbols, whereas semantics deals with meaning of syntactically correct

constructs (p.3). Therefore violating the stated language rules is a syntax error. Runtime and logic problems are not the rule category described.

Primary note proof

Study-note section CP32

> A syntax error violates grammar; a type error can be syntactically well formed but rejected during semantic checking. A runtime error occurs during execution; a logic error gives the wrong behavior. A bug is a defect; debugging locates and corrects it. Read the first useful compiler diagnostic, inspect nearby tokens and declarations, repair and rebuild; a debugger is useful after a runnable program exists.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 3 | lines 8-17

> Categories of programming language: - Low-level languages: - - are machine-dependent - They are designed to be run on a particular computer (not portable). Eg. Machine language and Assembly Language High-level languages - Are machine-independent - Can be run on a variety of computers. Programming language require translation source code to machine language before execution. This translation is accomplished by either a compiler or an interpreter. Compilers translate the entire source

Question sources

Computer Science MCQ practice collection · Question 251

Question 114 | AM0252

What is the purpose of debugging in programming?

A. To execute programs

B. To write programs

C. To optimize program performance

D. To find and fix errors in programs

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: Find and fix errors in programs.

A bug is a defect, and CP32 says debugging locates and corrects it. Executing or writing programs describes broader activities; optimizing performance has a different primary aim. Debugging can involve running code and editing it, but those steps serve its specific purpose of finding and fixing errors.

Why each choice is right or wrong

A - Incorrect: Execution can reveal defects, but simply running a program does not define debugging's purpose.

B - Incorrect: Writing programs is implementation; debugging specifically investigates and corrects defects in them.

C - Incorrect: Performance optimization aims at efficiency, whereas debugging aims at correcting erroneous behavior.

D - Correct: Finding and fixing program errors directly matches CP32's stated debugging purpose.

Primary note proof

Study-note section CP32

> A syntax error violates grammar; a type error can be syntactically well formed but rejected during semantic checking. A runtime error occurs during execution; a logic error gives the wrong behavior. A bug is a defect; debugging locates and corrects it. Read the first useful compiler diagnostic, inspect nearby tokens and declarations, repair and rebuild; a debugger is useful after a runnable program exists.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 2 | lines 2-18

> 1. Introduction to programming Computer programming is the process of writing, testing, debugging/troubleshooting, and maintaining the source code of computer programs. Programs are the instructions that tells the computer what to do .so, writing computer programs means writing instructions, that will make the computer follow and run a program based on those instructions. We are going to write computer programs to solve problems. Problem Solving Problem: gap between the existing and the desired or proposed situation where problem solving will try to fill this gap. Problem solving: The process of transforming the description of a problem into the solution by using our knowledge of the problem-solving strategies, techniques and tools. Problem solving techniques • Analyze the problem: Carefully understand the problem • Outline the problem requirements • Does program require user interaction? • Does program manipulate data? • What is the output? • Design steps (algorithm) to solve the problem

Question sources

Computer Science MCQ practice collection · Question 252

Question 115 | AM0253

What is the role of comments in programming?

A. To execute specific sections of code

B. To define variables and constants

C. To provide documentation and explanations within the code

D. To display output to the user

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: To provide documentation and explanations within the code

Comments are descriptive text explaining aspects of a program, ignored by the compiler and intended for human readers (p.5). Thus they document/explain code; they do not execute, declare variables by themselves or display text at runtime. Module support: PDF p. 5.

Why each choice is right or wrong

A - Incorrect: "To execute specific sections of code" does not match the rule or result derived here. Comments are descriptive text explaining aspects of a program, ignored by the compiler and intended for human readers (p.5). Thus they document/explain code; they do not execute, declare variables by themselves or display text at runtime.

B - Incorrect: "To define variables and constants" does not match the rule or result derived here. Comments are descriptive text explaining aspects of a program, ignored by the compiler and intended for human readers (p.5). Thus they document/explain code; they do not execute, declare variables by themselves or display text at runtime.

C - Correct: "To provide documentation and explanations within the code" matches the derived result. Comments are descriptive text explaining aspects of a program, ignored by the compiler and intended for human readers (p.5). Thus they document/explain code; they do not execute, declare variables by themselves or display text at runtime.

D - Incorrect: "To display output to the user" does not match the rule or result derived here. Comments are descriptive text explaining aspects of a program, ignored by the compiler and intended for human readers (p.5). Thus they document/explain code; they do not execute, declare variables by themselves or display text at runtime.

Primary note proof

Study-note section CP05

> Comments use // to the line end or /* ... */ for a block. They explain intent and assumptions for maintenance; ordinary comments do not execute or automatically improve speed, reliability or portability. main is the conventional entry function name, not a C++ keyword.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 18-20

> 4. Comments A comment is a piece of descriptive text which explains some aspect of a program. Program comments are totally ignored by the compiler and are only intended for human readers.

Question sources

Computer Science MCQ practice collection · Question 253

Question 116 | AM0254

What is the purpose of version control in programming?

A. To write programs

B. To execute programs

C. To manage and track changes in code over time

D. To optimize program performance

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: Manage and track code changes over time.

CP32 says version control records changes and supports comparison and rollback. That is the management of code history described by C. A version-control system may support a development workflow, but it does not itself define the process of writing code, executing a program or optimizing its runtime performance.

Why each choice is right or wrong

A - Incorrect: Developers write programs; version control records and manages the resulting revisions rather than creating program logic.

B - Incorrect: Execution is performed by the program's runtime or machine, not by revision-history management.

C - Correct: Tracking revisions, comparing changes and rolling back are the version-control purposes explicitly stated in CP32.

D - Incorrect: Runtime performance optimization changes program efficiency; version control manages the history of such changes.

Primary note proof

Study-note section CP32

> Test ordinary, boundary and invalid inputs and trace the first point where actual state differs from expected state. Exceptions transfer control to a matching handler for exceptional conditions; they do not make arbitrary undefined behavior recoverable. Refactoring improves internal structure while preserving observable behavior. Version control records changes, supports comparison and rollback; documentation explains use, design and maintenance assumptions.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 2 | lines 2-18

1. Introduction to programming Computer programming is the process of writing, testing, debugging/troubleshooting, and maintaining the source code of computer programs. Programs are the instructions that tells the computer what to do .so, writing computer programs means writing instructions, that will make the computer follow and run a program based on those instructions. We are going to write computer programs to solve problems. Problem Solving Problem: gap between the existing and the desired or proposed situation where problem solving will try to fill this gap. Problem solving: The process of transforming the description of a problem into the solution by using our knowledge of the problem-solving strategies, techniques and tools. Problem solving techniques • Analyze the problem: Carefully understand the problem • Outline the problem requirements • Does program require user interaction? • Does program manipulate data? • What is the output? • Design steps (algorithm) to solve the problem

Question sources

Computer Science MCQ practice collection · Question 254

Question 117 | AM0255

What is the difference between a parameter and an argument in programming?

A. Parameters are used in functions, while arguments are used in loops.

B. Parameters are passed to functions, while arguments are used to define functions.

C. Parameters are variables used to define a function, while arguments are the values passed to the function when it is called.

D. There is no difference; the terms are used interchangeably.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: Parameters are variables used to define a function, while arguments are the values passed to the function when it is called.

The definition syntax has a Parameter List (p.15), whereas the call Addition(x,y) supplies values; the passing-by-value discussion calls the passed actual value an argument (p.16). This supports C. A assigns arguments to loops, B reverses definition/call roles, and D ignores the distinction illustrated. Module support: PDF p. 15, PDF p. 16.

Why each choice is right or wrong

A - Incorrect: "Parameters are used in functions, while arguments are used in loops." does not match the rule or result derived here. The definition syntax has a Parameter List (p.15), whereas the call Addition(x,y) supplies values; the passing-by-value discussion calls the passed actual value an argument (p.16). This supports C. A assigns arguments to loops, B reverses definition/call roles, and D ignores the distinction illustrated.

B - Incorrect: "Parameters are passed to functions, while arguments are used to define functions." does not match the rule or result derived here. The definition syntax has a Parameter List (p.15), whereas the call Addition(x,y) supplies values; the passing-by-value discussion calls the passed actual value an argument (p.16). This supports C. A assigns arguments to loops, B reverses definition/call roles, and D ignores the distinction illustrated.

C - Correct: "Parameters are variables used to define a function, while arguments are the values passed to
the function when it is called." matches the derived result. The definition syntax has a Parameter List (p.15), whereas the call Addition(x,y) supplies values; the passing-by-value discussion calls the passed actual value an argument (p.16). This supports C. A assigns arguments to loops, B reverses definition/call roles, and D ignores the distinction illustrated.

D - Incorrect: "There is no difference; the terms are used interchangeably." does not match the rule or result derived here. The definition syntax has a Parameter List (p.15), whereas the call Addition(x,y) supplies values; the passing-by-value discussion calls the passed actual value an argument (p.16). This supports C. A assigns arguments to loops, B reverses definition/call roles, and D ignores the distinction illustrated.

Primary note proof

Study-note section CP17

> A function groups a named task. C++ definition: int add(int a,int b){return a+b;}. Call it with add(x,y);, without parameter type declarations at the call site. Parameters are names in the definition; arguments are supplied expressions or values. A void function produces no return value but can have side effects and return; to exit early.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 15 | lines 29-46

> • User-defined - Function that been created by the user. - These functions need to be declared and defined by the user. Declaration of function Function defines using the following syntax: Function_type function_name (Parameter List) { ...function body } Example: void Addition(int num1,int num2) #include<iostream> using namespace std; void Addition(int num1,int num2) { int sum=num1+num2; cout<<"The sum is:"<<sum; } int main() { int x=10,y=20; Addition(x,y);

Computer Programming module | PDF page 16 | lines 13-20

> Passing by Value In call by value, the actual value that is passed as argument is not changed after performing some operation on it. it creates a copy of that variable into the stack section in memory. When the value is changed, it changes the value of that copy, the actual value remains the same Example: Passing by Reference: It allows a function to modify a variable without having to create a copy of it. We have to declare reference variables. The memory location of the passed variable and parameter is the same and therefore, any change to the parameter reflects in the variable as well.

Question sources

Computer Science MCQ practice collection · Question 255

Question 118 | AM0256

What is the purpose of logical operators in programming?

A. To perform arithmetic calculations

B. To control the flow of program execution

C. To combine or manipulate boolean values

D. To create user interfaces

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: To combine or manipulate boolean values

The module lists !, && and || as logical operators producing 1/0 truth outcomes (p.7). They manipulate and combine truth values. Arithmetic operators perform numerical calculations; UI construction and flow control are broader/different roles. Module support: PDF p. 7.

Why each choice is right or wrong

A - Incorrect: "To perform arithmetic calculations" does not match the rule or result derived here. The module lists !, && and || as logical operators producing 1/0 truth outcomes (p.7). They manipulate and combine truth values. Arithmetic operators perform numerical calculations; UI construction and flow control are broader/different roles.

B - Incorrect: "To control the flow of program execution" does not match the rule or result derived here. The module lists !, && and || as logical operators producing 1/0 truth outcomes (p.7). They manipulate and combine truth values. Arithmetic operators perform numerical calculations; UI construction and flow control are broader/different roles.

C - Correct: "To combine or manipulate boolean values" matches the derived result. The module lists !, && and || as logical operators producing 1/0 truth outcomes (p.7). They manipulate and combine truth values. Arithmetic operators perform numerical calculations; UI construction and flow control are broader/different roles.

D - Incorrect: "To create user interfaces" does not match the rule or result derived here. The module lists !, && and || as logical operators producing 1/0 truth outcomes (p.7). They manipulate and combine truth values. Arithmetic operators perform numerical calculations; UI construction and flow control are broader/different roles.

Primary note proof

Study-note section CP10

> Relational and equality operators test values; && and || combine truth values; ! negates. In C++, zero converts to false and a nonzero number to true. Streams display bool as 0 or 1 unless boolalpha is set. With a=3,b=5, cout<<(a==b) prints 0.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Computer Science MCQ practice collection · Question 256

Question 119 | AM0257

What is the purpose of modular programming?

A. To create user interfaces

B. To write efficient code

C. To divide a program into separate, reusable modules

D. To optimize program performance

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: To divide a program into separate, reusable modules

The module says functions structure programs modularly, and dividing a complex problem into smaller chunks improves understanding and reuse (p.14). That directly supports separate reusable modules. Efficiency may benefit indirectly but is not the stated modular decomposition purpose. Module support: PDF p. 14.

Why each choice is right or wrong

A - Incorrect: "To create user interfaces" does not match the rule or result derived here. The module says functions structure programs modularly, and dividing a complex problem into smaller chunks improves understanding and reuse (p.14). That directly supports separate reusable modules. Efficiency may benefit indirectly but is not the stated modular decomposition purpose.

B - Incorrect: "To write efficient code" does not match the rule or result derived here. The module says functions structure programs modularly, and dividing a complex problem into smaller chunks improves understanding and reuse (p.14). That directly supports separate reusable modules. Efficiency may benefit indirectly but is not the stated modular decomposition purpose.

C - Correct: "To divide a program into separate, reusable modules" matches the derived result. The module says functions structure programs modularly, and dividing a complex problem into smaller chunks improves understanding and reuse (p.14). That directly supports separate reusable modules. Efficiency may benefit indirectly but is not the stated modular decomposition purpose.

D - Incorrect: "To optimize program performance" does not match the rule or result derived here. The module says functions structure programs modularly, and dividing a complex problem into smaller chunks improves understanding and reuse (p.14). That directly supports separate reusable modules. Efficiency may benefit indirectly but is not the stated modular decomposition purpose.

Primary note proof

Study-note section CP17

> A method is a function associated with a class or object; methods can return values. Modules group related responsibilities behind interfaces. Reuse reduces duplication; independent functions are easier to understand and test. A library provides reusable components; a framework also supplies an organizing structure and often calls application callbacks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 14 | lines 21-33

> Function A function is a group of statements that structure our programs in a more modular way and executed when it is called from some point of the program. It is a block of code that performs a specific task. It is important to Dividing a complex problem into smaller chunks makes our program easy to understand and reusable. When we need function? - When you need to repeat the same process over and over in a program. - The function can be called many times but appears in the code once Function - Categories C++ functions can be divided into 2 categories: • pre-defined (standard function) / library function - Which is the definitions have been written and it is ready to be used. - User needs to include pre-defined header file (i.e., math.h, time.h,cmath) To use it: FunctionName (value) e.g., sqrt (4), floor (4.3)

Question sources

Computer Science MCQ practice collection · Question 257

Question 120 | AM0258

What is the role of data structures in programming?

A. To store and organize data efficiently

B. To control the flow of program execution

C. To define user interfaces

D. To perform input/output operations

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: To store and organize data efficiently

The module organizes homogeneous indexed values as arrays (p.19) and grouped fields as structures (pp.34-35). These are concrete ways of storing and organizing data, matching A. Flow control, interface definition and input/output are separate program operations, not the storage organization role. Module support: PDF p. 19, PDF p. 35.

Why each choice is right or wrong

A - Correct: "To store and organize data efficiently" matches the derived result. The module organizes homogeneous indexed values as arrays (p.19) and grouped fields as structures (pp.34-35). These are concrete ways of storing and organizing data, matching A. Flow control, interface definition and input/output are separate program operations, not the storage organization role.

B - Incorrect: "To control the flow of program execution" does not match the rule or result derived here. The module organizes homogeneous indexed values as arrays (p.19) and grouped fields as structures (pp.34-35). These are concrete ways of storing and organizing data, matching A. Flow control, interface definition and input/output are separate program operations, not the storage organization role.

C - Incorrect: "To define user interfaces" does not match the rule or result derived here. The module organizes homogeneous indexed values as arrays (p.19) and grouped fields as structures (pp.34-35). These are concrete ways of storing and organizing data, matching A. Flow control, interface definition and input/output are separate program operations, not the storage organization role.

D - Incorrect: "To perform input/output operations" does not match the rule or result derived here. The module organizes homogeneous indexed values as arrays (p.19) and grouped fields as structures (pp.34-35). These are concrete ways of storing and organizing data, matching A. Flow control, interface definition and input/output are separate program operations, not the storage organization role.

Primary note proof

Study-note section CP22

A built-in C++ array has a fixed number of same-type elements in contiguous storage: int a[4]={10,20,30,40};. Valid indices are 0 through 3. a[4] is outside the array; accessing it has undefined behavior. Java checks bounds and throws ArrayIndexOutOfBoundsException. Do not predict a C++ out-of-range value from nearby memory.

Study-note section CP27

A struct groups named members, possibly of different types. struct House{double price; int rooms;}; defines a type; House h{150000,3}; creates an object. h.rooms uses a dot. House* p=&h; p->rooms is equivalent to (*p).rooms. The pointer must refer to a live object before member access. A type definition alone does not create an ordinary instance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 19 | lines 4-29

Arrays - an array is a collection of fixed-size number of elements of the same type stored sequentially in memory. - An individual element of an array is identified by its own unique index (or subscript). - The number of elements must be an integer. - The size of the array is referred to as its dimension - Array in C++ is Zero indexed. Declaring of one-dimensional Array Syntax: elementstype arrayname [size]; → You can use constants such as macro definitions or const ints as the sizes of arrays. Example: int num[5]; string fruit[3]; #define numstudents 100 or const int numstudents=100; double Score[numstudents] ; double Score[numstudents] ; Initialization of an Array - Arrays can be initialized at declaration time in two different ways: Syntax: elementstype arrayname[size] = {list each value separated by comma}; E.g int num[5]={1,2,3,4,5}; elementstype arrayname[ ] = { list with any number of values }; E.g int num[]={1,2,3,4,5}; string fruit[]={“Apple”,“orange”, “Mango”}; Accessing Array Elements - Each element could be accessed using an index into the array - When accessing an array, the index given must be a positive integer from 0 to n-1, where n is the dimension of the array.

Computer Programming module | PDF page 35 | lines 13-17

Accessing members of a structure variable A structure member can be accessed using dot operator’.’ notation syntax: structvariable.member; eg. st.id=4;

Question sources

Computer Science MCQ practice collection · Question 258

Question 121 | AM0259

What is the purpose of exception handling in programming?

A. To perform input/output operations

B. To optimize program performance

C. To handle and recover from runtime errors

D. To define variables and constants

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: Handle exceptional runtime conditions and recover where appropriate.

CP32 describes exceptions as transferring control to a matching handler for exceptional conditions. This supports handling and possible recovery from runtime errors. It does not promise that every runtime fault is recoverable: arbitrary undefined behavior is specifically excluded. Input/output, optimization and declarations are not exception handling's defining purpose.

Why each choice is right or wrong

A - Incorrect: Input/output may raise exceptional conditions, but performing I/O is not the defining purpose of exception handling.

B - Incorrect: Exception handling routes exceptional control flow rather than being a mechanism for performance optimization.

C - Correct: Matching handlers support handling and possible recovery from runtime exceptions, subject to the note's stated limits.

D - Incorrect: Variables and constants are introduced by declarations, not by exception-handling control flow.

Primary note proof

Study-note section CP32

> Test ordinary, boundary and invalid inputs and trace the first point where actual state differs from expected state. Exceptions transfer control to a matching handler for exceptional conditions; they do not make arbitrary undefined behavior recoverable. Refactoring improves internal structure while preserving observable behavior. Version control records changes, supports comparison and rollback; documentation explains use, design and maintenance assumptions.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 9 | lines 2-8

> Control Statements Control statements or flow control determines what is executed during a run and what is not, therefore affecting the overall outcome of the program. C++ provides different forms of statements for different purposes. Declaration statements are used for defining variables. Assignment-like statements are used for simple, algebraic computations. Branching statements are used for specifying alternate paths of execution, depending on the outcome of a logical condition. Loop statements are used for specifying computations, which need to be repeated until a certain logical condition is satisfied

Question sources

Computer Science MCQ practice collection · Question 259

Question 122 | AM0260

Which statement accurately distinguishes static typing from dynamic typing? Consider type-checking time rather than mandatory annotations or programming paradigms.

A. Static typing is more efficient than dynamic typing.

B. Static typing checks types before execution, possibly using inferred types; dynamic typing checks operations using runtime values.

C. Static typing is used in object-oriented programming, while dynamic typing is used in procedural programming.

D. There is no difference; the terms are used interchangeably.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: B. Static typing checks types before execution, possibly using inferred types; dynamic typing checks operations using runtime values.

Static typing checks types before execution, and the static type may be inferred rather than explicitly written. Dynamic typing checks operations using values at runtime. The difference is therefore about the typing/checking model, not a guarantee of execution speed or a choice between object-oriented and procedural programming. B states this distinction; static typing does not require explicitly written type declarations.

Why each choice is right or wrong

A - Incorrect: Typing discipline alone does not guarantee a universal efficiency ranking.

B - Correct: This correctly distinguishes before-execution checking from runtime checking and allows inference.

C - Incorrect: Object-oriented versus procedural organization does not determine typing discipline.

D - Incorrect: The two checking models have distinct meanings.

Primary note proof

Study-note section CP33

> Static typing checks types before execution; type inference can still infer a static type, as with C++ auto. Dynamic typing checks operations using runtime values; it does not mean variables must have explicit static declarations. Static typing does not always require explicit type annotations.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 24-33

5. Data Types, Variables, and Constants Variable: - Variable is a symbolic name for a memory location in which data can be stored. - Attribute of Variables: o A type, which is, established when the variable is defined (e.g., integer, float, character) o A value, which can be changed by assigning a new value to the variable Variable Declaration - Declaring a variable means defining (creating) a variable. - Syntax: data type variable name; e.g. int num; - We can create More Than One Variable at a Time e.g., int num1, num2, num3;

Question sources

Computer Science MCQ practice collection · Question 260

Question 123 | AM0261

What is the purpose of libraries and frameworks in programming?

A. To perform input/output operations

B. To optimize program performance

C. To provide reusable code and functionality

D. To create user interfaces

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: To provide reusable code and functionality

Library functions are described as predefined definitions ready to use, with headers required before calling them (p.14). This supplies reusable functionality rather than forcing users to implement it again. Input/output and UI are possible particular facilities, not the broad purpose; optimization is not the definition. Module support: PDF p. 14, PDF p. 15.

Why each choice is right or wrong

A - Incorrect: "To perform input/output operations" does not match the rule or result derived here. Library functions are described as predefined definitions ready to use, with headers required before calling them (p.14). This supplies reusable functionality rather than forcing users to implement it again. Input/output and UI are possible particular facilities, not the broad purpose; optimization is not the definition.

B - Incorrect: "To optimize program performance" does not match the rule or result derived here. Library functions are described as predefined definitions ready to use, with headers required before calling them (p.14). This supplies reusable functionality rather than forcing users to implement it again. Input/output and UI are possible particular facilities, not the broad purpose; optimization is not the definition.

C - Correct: "To provide reusable code and functionality" matches the derived result. Library functions are described as predefined definitions ready to use, with headers required before calling them (p.14). This supplies reusable functionality rather than forcing users to implement it

again. Input/output and UI are possible particular facilities, not the broad purpose; optimization is not the definition.

D - Incorrect: "To create user interfaces" does not match the rule or result derived here. Library functions are described as predefined definitions ready to use, with headers required before calling them (p.14). This supplies reusable functionality rather than forcing users to implement it again. Input/output and UI are possible particular facilities, not the broad purpose; optimization is not the definition.

Primary note proof

Study-note section CP17

A method is a function associated with a class or object; methods can return values. Modules group related responsibilities behind interfaces. Reuse reduces duplication; independent functions are easier to understand and test. A library provides reusable components; a framework also supplies an organizing structure and often calls application callbacks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 14 | lines 21-33

Function A function is a group of statements that structure our programs in a more modular way and executed when it is called from some point of the program. It is a block of code that performs a specific task. It is important to Dividing a complex problem into smaller chunks makes our program easy to understand and reusable. When we need function? - When you need to repeat the same process over and over in a program. - The function can be called many times but appears in the code once Function - Categories C++ functions can be divided into 2 categories: • pre-defined (standard function) / library function - Which is the definitions have been written and it is ready to be used. - User needs to include pre-defined header file (i.e., math.h, time.h,cmath) To use it: FunctionName (value) e.g., sqrt (4), floor (4.3)

Computer Programming module | PDF page 15 | lines 29-46

• User-defined - Function that been created by the user. - These functions need to be declared and defined by the user. Declaration of function Function defines using the following syntax: Function_type function_name (Parameter List) { ...function body } Example: void Addition(int num1,int num2) #include<iostream> using namespace std; void Addition(int num1,int num2) { int sum=num1+num2; cout<<"The sum is:"<<sum; } int main() { int x=10,y=20; Addition(x,y);

Question sources

Computer Science MCQ practice collection · Question 261

Question 124 | AM0262

What is the purpose of code refactoring in programming?

A. To write efficient code

B. To optimize program performance

C. To improve the readability and maintainability of code

D. To handle and recover from runtime errors

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: Improve code readability and maintainability while preserving behavior.

CP32 defines refactoring as improving internal structure while preserving observable behavior. Better readability and maintainability are the offered purpose that matches this definition. Performance tuning can accompany other changes, but is not the defining aim; runtime-error recovery belongs to exception handling rather than refactoring.

Why each choice is right or wrong

A - Incorrect: Writing efficient code is broader than refactoring and does not specify preserving existing observable behavior.

B - Incorrect: Performance optimization has a different primary aim; refactoring concerns structure and maintainability.

C - Correct: Readability and maintainability improve through internal restructuring while observable program behavior is preserved.

D - Incorrect: Handling exceptional runtime conditions belongs to exception handling, not to refactoring's structural purpose.

Primary note proof

Study-note section CP32

> Test ordinary, boundary and invalid inputs and trace the first point where actual state differs from expected state. Exceptions transfer control to a matching handler for exceptional conditions; they do not make arbitrary undefined behavior recoverable. Refactoring improves internal structure while preserving observable behavior. Version control records changes, supports comparison and rollback; documentation explains use, design and maintenance assumptions.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 2 | lines 2-18

> 1. Introduction to programming Computer programming is the process of writing, testing, debugging/troubleshooting, and maintaining the source code of computer programs. Programs

are the instructions that tells the computer what to do .so, writing computer programs means writing instructions, that will make the computer follow and run a program based on those instructions. We are going to write computer programs to solve problems. Problem Solving Problem: gap between the existing and the desired or proposed situation where problem solving will try to fill this gap. Problem solving: The process of transforming the description of a problem into the solution by using our knowledge of the problem-solving strategies, techniques and tools. Problem solving techniques • Analyze the problem: Carefully understand the problem • Outline the problem requirements • Does program require user interaction? • Does program manipulate data? • What is the output? • Design steps (algorithm) to solve the problem

Computer Programming module | PDF page 5 | lines 18-20

4. Comments A comment is a piece of descriptive text which explains some aspect of a program. Program comments are totally ignored by the compiler and are only intended for human readers.

Question sources

Computer Science MCQ practice collection · Question 262

Question 125 | AM0263

What is the role of algorithms in programming?

A. To store and organize data efficiently

B. To perform input/output operations

C. To define user interfaces

D. To provide step-by-step instructions for solving problems

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: To provide step-by-step instructions for solving problems

The module defines an algorithm as ordered unambiguous executable steps defining a terminating process (p.2). Therefore it supplies step-by-step problem-solving instructions. Storage organization, input/output and UI design are possible concerns of a program rather than this general definition. Module support: PDF p. 2.

Why each choice is right or wrong

A - Incorrect: "To store and organize data efficiently" does not match the rule or result derived here. The module defines an algorithm as ordered unambiguous executable steps defining a terminating process (p.2). Therefore it supplies step-by-step problem-solving instructions. Storage organization, input/output and UI design are possible concerns of a program rather than this general definition.

B - Incorrect: "To perform input/output operations" does not match the rule or result derived here. The module defines an algorithm as ordered unambiguous executable steps defining a terminating process (p.2). Therefore it supplies step-by-step problem-solving instructions. Storage organization, input/output and UI design are possible concerns of a program rather than this general definition.

C - Incorrect: "To define user interfaces" does not match the rule or result derived here. The module defines an algorithm as ordered unambiguous executable steps defining a terminating process (p.2). Therefore it supplies step-by-step problem-solving instructions. Storage organization, input/output and UI design are possible concerns of a program rather than this general definition.

D - Correct: "To provide step-by-step instructions for solving problems" matches the derived result. The module defines an algorithm as ordered unambiguous executable steps defining a terminating process (p.2). Therefore it supplies step-by-step problem-solving instructions. Storage organization, input/output and UI design are possible concerns of a program rather than this general definition.

Primary note proof

Study-note section CP02

> An algorithm gives precise, executable steps and terminates for every legal input. Definiteness means each step has an unambiguous meaning; effectiveness means a step can actually be performed. Inputs may be absent, but a useful algorithm specifies a result. Correctness means the result meets the specification. Efficiency is desirable and measurable, but an inefficient terminating algorithm is still an algorithm. Efficiency is also a desirable property of an algorithm.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 2 | lines 19-26

> Algorithm is an ordered set of unambiguous, executable steps, defining a terminating process. Important Properties of Algorithms - Correct: always returns the desired output for all legal instances of the problem. - Unambiguous: Each step must be clearly defined, having only one interpretation. - Precise: simple but efficient - Efficient: can be measured in terms of Time and Space - Finiteness: the execution of the algorithm must terminate in a finite amount of time for each possible set of input data/must have an end.

Question sources

Computer Science MCQ practice collection · Question 263

Question 126 | AM0264

What is the purpose of documentation in programming?

A. To execute programs

B. To optimize program performance

C. To provide explanations and instructions for using and maintaining the code

D. To create user interfaces

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: To provide explanations and instructions for using and maintaining the code

The module defines comments as descriptive text explaining a program for human readers (p.5). Documentation serves this explanatory role; instructions for use and maintenance are a reasonable application of that role. Module support: PDF p. 5.

Why each choice is right or wrong

A - Incorrect: Execution follows program instructions, whereas explanatory comments are ignored by the compiler.

B - Incorrect: The cited purpose is human understanding, not performance optimization.

C - Correct: Explanations for readers support using and maintaining code.

D - Incorrect: Creating interfaces is not the descriptive role defined on p.5.

Primary note proof

Study-note section CP05

> Comments use // to the line end or /* ... */ for a block. They explain intent and assumptions for maintenance; ordinary comments do not execute or automatically improve speed, reliability or portability. main is the conventional entry function name, not a C++ keyword.

Study-note section CP32

> Test ordinary, boundary and invalid inputs and trace the first point where actual state differs from expected state. Exceptions transfer control to a matching handler for exceptional conditions; they do not make arbitrary undefined behavior recoverable. Refactoring improves internal structure while preserving observable behavior. Version control records changes, supports comparison and rollback; documentation explains use, design and maintenance assumptions.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 18-20

4. Comments A comment is a piece of descriptive text which explains some aspect of a program. Program comments are totally ignored by the compiler and are only intended for human readers.

Question sources

Computer Science MCQ practice collection · Question 264

Question 127 | AM0265

What does the cout object do in C++?

A. It reads input from the user.

B. It writes output to the console.

C. It performs mathematical calculations.

D. It declares variables.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: It writes output to the console.

Page 6 explicitly calls cout the standard output stream and says it displays information for the user. Module support: PDF p. 6.

Why each choice is right or wrong

A - Incorrect: Input is assigned to cin on p.6.

B - Correct: This is the stated output role.

C - Incorrect: cout displays expressions; it is not the calculation operator.

D - Incorrect: Declarations use a data type and name (p.5), not cout.

Primary note proof

Study-note section CP28

> Include <iostream> and qualify std::cin/std::cout or introduce the names explicitly. cout is the standard output stream; << inserts values. cin is standard input; >> extracts formatted values. std::endl inserts a newline and flushes. C++ names are case sensitive: Cout, Int and std: :cout are not equivalent valid spellings of cout, int and std::cout.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 6 | lines 4-15

> 6. Communication through console cout (the standard output stream) o used to display information for the user Syntax: cout<<object; • Object can be variable or literals Example: int x=5; cout<<x cout<< “Hello there”; cin(the standard input stream): o used to accept information from the user. Syntax: cin>>variable name; Eg. cin>>x;

Question sources

Computer Science MCQ practice collection · Question 265

Question 128 | AM0266

Without any user-defined type aliases, which option declares an ordinary C++ int variable named variableName?

A. variableName = value;

B. value = variableName;

C. int variableName;

D. variableName var;

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: C. int variableName;

An ordinary C++ variable declaration places a type specifier before the variable name. Option C uses int followed by variableName and a semicolon, so it introduces the requested integer variable. The assignment expressions in A and B do not introduce a type. D would require variableName to be a type, contrary to the stated requested variable name and absence of aliases.

Why each choice is right or wrong

A - Incorrect: An assignment does not declare the variable's type.

B - Incorrect: Reversing an assignment still does not introduce a declaration.

C - Correct: int variableName; has the required type-then-name form.

D - Incorrect: variableName is not a type under the stated assumptions.

Primary note proof

Study-note section CP06

> Write type name; or type name = value; for ordinary variables: int age = 20;. Declaration introduces a name and type; initialization gives its initial value; later assignment replaces a value. a=b copies a scalar value, so later changes to b do not change a. A pointer or Java object reference instead copies an address/reference.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 24-33

> 5. Data Types, Variables, and Constants Variable: - Variable is a symbolic name for a memory location in which data can be stored. - Attribute of Variables: o A type, which is, established when the variable is defined (e.g., integer, float, character) o A value, which can be changed by assigning a new value to the variable Variable Declaration - Declaring a variable means defining (creating) a variable. - Syntax: data type variable name; e.g. int num; - We can create More Than One Variable at a Time e.g., int num1, num2, num3;

Question sources

Computer Science MCQ practice collection · Question 266

Question 129 | AM0267

What is the purpose of the for loop in C++?

A. To perform conditional branching.

B. To read input from the user.

C. To execute a block of code repeatedly.

D. To declare variables.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: To execute a block of code repeatedly.

The for syntax on p.12 checks a condition, executes the body, performs the increase, then returns to the condition. This gives repeated execution. Module support: PDF p. 12.

Why each choice is right or wrong

A - Incorrect: Branch selection is not the loop purpose.

B - Incorrect: Input uses cin.

C - Correct: The body repeats while the condition holds.

D - Incorrect: An initializer can declare a counter, but declaration is not the loop purpose.

Primary note proof

Study-note section CP14

> for(initialization; condition; update) executes initialization once, then condition → body → update → condition. A missing condition in for(;;) acts as true. A for loop is convenient for a known count but can also be condition controlled. To print indices 0 through n-1, use i=0; i<n; ++i. Starting i=0 with i<5 and i+=2 prints 0 2 4.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 12 | lines 19-28

> The for loop. Its format is: for (initialization; condition; increase) statement; 1, initialization is executed. Generally, it is a initial value setting for a counter variable. This is executed only once 2, condition is checked, if it is true the loop continues, otherwise the loop finishes and statement is skipped. 3, statement is executed. As usual, it can be either a single instruction or a block of instructions enclosed within curly brackets { }. 4, finally, whatever is specified in the increase field is executed and the loop gets back to step 2.

Question sources

Computer Science MCQ practice collection · Question 267

Question 130 | AM0268

What is the output of the following code snippet?

```
int x = 5;
cout << x++ << endl;
cout << x << endl;
```

A. 5, 6

B. 5, 5

C. 6, 6

D. 6, 5

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer A: 5, 6

With x initially 5, postfix x++ yields its old value and then increases x to 6; the next statement displays 6. The module operator example on p.8 contrasts postfix b++ and prefix ++a; its previously inspected output confirms this distinction. The two cout statements thus print 5 then 6 on separate lines. Module support: PDF p. 7, PDF p. 8, PDF p. 6.

Why each choice is right or wrong

A - Correct: Old value followed by updated value.

B - Incorrect: Fails to include the increment in the second output.

C - Incorrect: Treats postfix as prefix in the first output.

D - Incorrect: Reverses both the initial output and final state.

Primary note proof

Study-note section CP09

> Prefix ++x changes x and yields the new value; postfix x++ yields the previous value and also increments x. Separate the value used in the expression from the stored value after it. Starting a=8,b=5, a++ + --b uses 8+4, gives 12, and leaves a=9,b=4. Starting x=1,y=10,z=-3, (x++*y--)+ ++z gives 8 and leaves x=2,y=9,z=-2.

Study-note section CP28

> std::endl inserts a newline and flushes.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y,

Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Computer Programming module | PDF page 8 | lines 12-23

{int a=10,b=3,c,d,e,f; c=a%b++; cout<<"c="<<c<<endl; b+=2; cout<<"b="<<b<<endl; d=++a+b; cout<<"d="<<d<<endl; e=a>b?a:b; e++; cout<<"e="<<e++<<endl; f=a+b+c+d+e; cout<<"f="<<f;

Computer Programming module | PDF page 6 | lines 4-15

6. Communication through console cout (the standard output stream) o used to display information for the user Syntax: cout<<object; • Object can be variable or literals Example: int x=5; cout<<x cout<< “Hello there”; cin(the standard input stream): o used to accept information from the user. Syntax: cin>>variable name; Eg. cin>>x;

Question sources

Computer Science MCQ practice collection · Question 268

Question 131 | AM0269

What is the purpose of the switch statement in C++?

A. To perform mathematical calculations.

B. To read input from the user.

C. To execute a block of code repeatedly.

D. To perform different actions based on different conditions.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: To perform different actions based on different conditions.

Page 10 defines switch as choosing among alternatives based on an expression value; p.11 describes matching that value to case labels. D is the matching description, with conditions understood as case matches. Module support: PDF p. 10.

Why each choice is right or wrong

A - Incorrect: Arithmetic is not switch selection.

B - Incorrect: Input is performed by cin.

C - Incorrect: Repetition belongs to loops.

D - Correct: Selecting actions by expression values matches the definition.

Primary note proof

Study-note section CP13

> A traditional C++ switch selects using an integral or enumeration value. case labels are constant expressions. Execution starts at the matching label, then continues through subsequent statements until a break, return or the end. default handles an unmatched value but need not be the last physical label.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 10 | lines 15-26

> switch Statement The switch statement provides a way of choosing between a set of alternatives, based on the value of an expression. The general form of the switch statement is: switch (expression) { caseconstant1: statements; ... caseconstantn: statements; default: statements; }

Question sources

Computer Science MCQ practice collection · Question 269

Question 132 | AM0270

With no user-defined return-type aliases, which C++ function definition can be called and then reach its closing brace normally without omitting a required return value? Assume this is an ordinary function other than main.

A. function myFunction() { }

B. void myFunction() { }

C. int myFunction() { }

D. myFunction() { }

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: B. void myFunction() { }

The void definition needs no result value and can safely finish at the end of its empty body. The int definition can be accepted as a definition, but reaching its end without returning a value has undefined behavior. The other forms omit a known return type. Specifying normal completion of an ordinary call distinguishes a safe complete definition from merely syntactically accepted but unsafe fall-through.

Why each choice is right or wrong

A - Incorrect: function is not a declared return type or C++ definition keyword here.

B - Correct: A void function can finish its empty body without returning a value.

C - Incorrect: Reaching the end of this int function omits a required return value.

D - Incorrect: An ordinary C++ function requires an explicit return type.

Primary note proof

Study-note section CP17

> A function groups a named task. C++ definition: int add(int a,int b){return a+b;}. Call it with add(x,y);, without parameter type declarations at the call site. Parameters are names in the definition; arguments are supplied expressions or values. A void function produces no return value but can have side effects and return; to exit early.

Study-note section CP35

> A void function definition can be complete without returning a value; int myFunction(){} can compile, but reaching its end is undefined behavior.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 15 | lines 29-46

• User-defined - Function that been created by the user. - These functions need to be declared and defined by the user. Declaration of function Function defines using the following syntax: Function_type function_name (Parameter List) { ...function body } Example: void Addition(int num1,int num2) #include<iostream> using namespace std; void Addition(int num1,int num2) { int sum=num1+num2; cout<<"The sum is:"<<sum; } int main() { int x=10,y=20; Addition(x,y);

Question sources

Computer Science MCQ practice collection · Question 270

Question 133 | AM0271

What is the purpose of the sizeof operator in C++?

A. To determine the memory size of a variable or data type.

B. To perform mathematical calculations.

C. To compare two variables.

D. To check the equality of two values.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: To determine the memory size of a variable or data type.

Page 29 ties pointer scaling to the size in bytes of the pointed object, and its example labels sizeof(x) as size of int. This supports sizeof as memory-size measurement, rather than arithmetic or comparison. Module support: PDF p. 29.

Why each choice is right or wrong

A - Correct: Memory size is the role shown by size of int and sizeof(x).

B - Incorrect: The example measures storage rather than calculating an arithmetic expression.

C - Incorrect: Comparisons use relational operators, not sizeof.

D - Incorrect: Equality uses == (p.7), not sizeof.

Primary note proof

Study-note section CP07

Single quotes denote a character literal, such as 'A'; double quotes denote a string literal, such as "A". A string includes more storage than just its visible characters when a terminator is required. sizeof reports size in bytes, with sizeof(char)==1; the byte width and other sizes are implementation dependent. sizeof(array) gives total array storage before conversion to a pointer.

Study-note section CP10

Relational and equality operators test values; && and || combine truth values; ! negates.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 29 | lines 6-17

> Note that ptr + 1 does not return the memory address after ptr, but the memory address of the next object of the type that ptr points to. If ptr points to an integer (assuming 4 bytes), ptr + 3 means 3 integers (12 bytes) after ptr. If ptr points to a char, which is always 1 byte, ptr + 3 means 3 chars (3 bytes) after ptr. When calculating the result of a pointer arithmetic expression, the compiler always multiplies the integer operand by the size of the object being pointed to. This is called scaling. Example: #include <iostream> using namespace std; int main() { int x=7; int* ptr=&x; cout<<"size of int:"<<sizeof(x)<<endl;

Variant 1 | Alternative version 1

What is the purpose of the sizeof operator in C++?

A. To determine the size of a variable in bytes.

B. To perform mathematical calculations.

C. To compare two variables.

D. To check the equality of two values.

Answer A: To determine the size of a variable in bytes.

Apply the same study-note rule to this alternative source set; the choice contrasts below show the derivation.

Why each choice is right or wrong

A - Correct: The pointer-scaling discussion uses bytes and the example labels sizeof(x) as size of int (p.29), supporting size of a variable in bytes.

B - Incorrect: sizeof measures storage size in the example, not arithmetic.

C - Incorrect: Comparing values uses relational operators (p.7).

D - Incorrect: Equality uses ==, not sizeof (p.7).

Primary note proof

Study-note section CP07

> Single quotes denote a character literal, such as 'A'; double quotes denote a string literal, such as "A". A string includes more storage than just its visible characters when a terminator is required. sizeof reports size in bytes, with sizeof(char)==1; the byte width and other sizes are implementation dependent. sizeof(array) gives total array storage before conversion to a pointer.

Study-note section CP10

> Relational and equality operators test values; && and || combine truth values; ! negates.

Question sources

Computer Science MCQ practice collection · Question 271

Computer Science MCQ practice collection · Question 278

Question 134 | AM0272

What does break do when executed inside a C++ loop or switch?

A. To terminate the program.

B. To exit the nearest enclosing loop or switch.

C. To repeat the loop from the beginning.

D. To execute a specific case in a switch statement.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: B. To exit the nearest enclosing loop or switch.

break exits the nearest enclosing loop or switch and resumes control after that construct. It does not terminate the entire program merely because it leaves the construct. Skipping only the remainder of the current iteration is the role of continue, while selecting a case is the switch expression's role. Option B states the exact enclosing-construct exit behavior rather than confusing it with iteration skipping.

Why each choice is right or wrong

A - Incorrect: Leaving a loop or switch does not itself terminate the program.

B - Correct: break exits the nearest enclosing loop or switch.

C - Incorrect: break exits rather than restarting the loop.

D - Incorrect: The switch expression selects a case; break exits the switch.

Primary note proof

Study-note section CP15

> break exits the nearest enclosing loop or switch. continue skips the rest of the current loop iteration: a for loop then runs its update; a while loop goes directly to its condition. return leaves the current function and may supply a result. In main it ends the program normally.

Study-note section CP13

> A traditional C++ switch selects using an integral or enumeration value. case labels are constant expressions. Execution starts at the matching label, then continues through subsequent statements until a break, return or the end. default handles an unmatched value but need not be the last physical label.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 13 | lines 30-35

Using continue and break Statements - The break statement skips the rest of the statements in the loop‟s block, jumping immediately to the next statement outside of the loop. - The continue statement is similar. It also skips the rest of the statements in the loop‟s block, but instead of terminating the loop, it transfers execution to the next iteration of the loop. It continues the loop after skipping the remaining statements in its current iteration.

Question sources

Computer Science MCQ practice collection · Question 272

Question 135 | AM0273

What is the output of the following code snippet?

```
int i = 0;
while (i < 5) {
cout << i << " ";
i++;
}
```

A. 0 1 2 3 4

B. 1 2 3 4 5

C. 5 4 3 2 1

D. 0 0 0 0 0

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer A: 0 1 2 3 4

The loop prints i before incrementing. Values 0,1,2,3,4 pass i<5; after the last increment i=5 fails the test (p.11). Module support: PDF p. 11, PDF p. 7, PDF p. 6.

Why each choice is right or wrong

A - Correct: Lists the five values printed before increments.

B - Incorrect: Starts after the increment and includes the excluded endpoint.

C - Incorrect: The counter increases from zero, not decreases.

D - Incorrect: i++ prevents i remaining zero.

Primary note proof

Study-note section CP14

while(condition) checks before each body execution and may run zero times. do {body;} while(condition); runs the body once before checking; it repeats while the condition remains true. Use it when a menu or input attempt must occur at least once.

Study-note section CP09

Prefix ++x changes x and yields the new value; postfix x++ yields the previous value and also increments x. Separate the value used in the expression from the stored value after it. Starting a=8,b=5, a++ + --b uses 8+4, gives 12, and leaves a=9,b=4. Starting x=1,y=10,z=-3, (x++*y--)+ ++z gives 8 and leaves x=2,y=9,z=-2.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 11 | lines 5-13

Iteration/Looping Statement: Iteration is the repetition of a statement or block of statements in a program. - Loops have as objective to repeat a statement a certain number of times or while a condition is fulfilled. - C++ has three iteration statements: the while statement, the do. While statement, and the for statement. The while loop: Its format is: while (expression) {statement;} First expression (called the loop condition) is evaluated. If the outcome is nonzero then statement (called the loop body) is executed and the whole process is repeated. Otherwise, the loop is terminated.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, | |) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 | | 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Computer Programming module | PDF page 6 | lines 4-15

6. Communication through console cout (the standard output stream) o used to display information for the user Syntax: cout<<object; • Object can be variable or literals Example: int x=5; cout<<x cout<< “Hello there”; cin(the standard input stream): o used to accept information from the user. Syntax: cin>>variable name; Eg. cin>>x;

Question sources

Computer Science MCQ practice collection · Question 273

Question 136 | AM0274

What is the purpose of the cin object in C++?

A. To write output to the console.

B. To declare variables.

C. To perform mathematical calculations.

D. To read input from the user.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: To read input from the user.

Page 6 identifies cin as standard input and says it accepts information from the user. Module support: PDF p. 6.

Why each choice is right or wrong

A - Incorrect: This is cout.

B - Incorrect: A declaration requires a type and identifier.

C - Incorrect: Arithmetic operators calculate.

D - Correct: Matches the stated input role.

Primary note proof

Study-note section CP28

> Include <iostream> and qualify std::cin/std::cout or introduce the names explicitly. cout is the standard output stream; << inserts values. cin is standard input; >> extracts formatted values. std::endl inserts a newline and flushes. C++ names are case sensitive: Cout, Int and std: :cout are not equivalent valid spellings of cout, int and std::cout.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 6 | lines 4-15

> 6. Communication through console cout (the standard output stream) o used to display information for the user Syntax: cout<<object; • Object can be variable or literals Example: int x=5; cout<<x cout<< “Hello there”; cin(the standard input stream): o used to accept information from the user. Syntax: cin>>variable name; Eg. cin>>x;

Question sources

Computer Science MCQ practice collection · Question 274

Computer Science MCQ practice collection · Question 275

Question 137 | AM0276

Under C++17, what does this fragment print? Assume cout is available.
int x=5;
cout<<x++<<" "<<++x;

A. 5 7

B. 5 5

C. 6 5

D. 6 6

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer A: A. 5 7

In C++17 the chained operator-form stream insertions are sequenced left to right. The postfix increment supplies the original value 5 to the first insertion and leaves x equal to 6. The later prefix increment changes x to 7 before supplying its value to the second insertion. The output is therefore 5 followed by a space and 7. Specifying the language version makes this a defined trace.

Why each choice is right or wrong

A - Correct: The postfix supplies 5 and the later prefix supplies 7.

B - Incorrect: The two increments prevent the second value from remaining 5.

C - Incorrect: The first postfix yields 5 rather than 6.

D - Incorrect: The first value is 5 and the second is 7, not two sixes.

Primary note proof

Study-note section CP09

> Precedence determines grouping, not a universal evaluation order. Avoid expressions that modify a variable multiple times without sequencing. In C++17 and later, chained operator-form stream insertions are sequenced left to right: with x=5, cout<<x++<<" "<<++x prints 5 7. Before C++17 this expression has undefined behavior. For the C++17 expression above, 5 6, 5 5, 6 5 and 6 6 are all incorrect outputs.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, | |) Like the relational operators,

logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Computer Programming module | PDF page 8 | lines 12-23

```
{int a=10,b=3,c,d,e,f; c=a%b++; cout<<"c="<<c<<endl; b+=2; cout<<"b="<<b<<endl; d=++a+b; cout<<"d="<<d<<endl; e=a>b?a:b; e++; cout<<"e="<<e++<<endl; f=a+b+c+d+e; cout<<"f="<<f;
```

Question sources

Computer Science MCQ practice collection · Question 276

Question 138 | AM0277

What does this C++ fragment print? Assume cout is available and execution reaches the fragment normally.

```
int total=0;
for(int i=1;i<=4;++i){
if(i==3) break;
total+=i;
}
cout<<total<<" done";
```

A. No output; the program terminates when i reaches 3.

B. 3 done

C. 7 done

D. 10 done

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: B. 3 done

At i=1 and i=2, the condition is false and total becomes 1 and then 3. At i=3, break exits the nearest enclosing loop before the addition, so neither 3 nor 4 is added. Control continues after the loop and the output statement prints 3 done. A total of 7 would result from skipping only the i=3 iteration with continue; 10 would require completing every addition.

Why each choice is right or wrong

A - Incorrect: break exits the loop, so execution still reaches the following cout statement.

B - Correct: Only 1 and 2 are added before loop exit, and cout then appends the literal space and done.

C - Incorrect: 7 would include 4 after skipping 3, which describes continue rather than break.

D - Incorrect: 10 adds all four loop values and ignores the exit at i=3.

Primary note proof

Study-note section CP15

> break exits the nearest enclosing loop or switch. continue skips the rest of the current loop iteration: a for loop then runs its update; a while loop goes directly to its condition. return leaves the current function and may supply a result. In main it ends the program normally.

Study-note section CP14

> for(initialization; condition; update) executes initialization once, then condition → body → update → condition. A missing condition in for(;;) acts as true. A for loop is convenient for a known count but can also be condition controlled. To print indices 0 through n-1, use i=0; i<n; ++i. Starting i=0 with i<5 and i+=2 prints 0 2 4.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 13 | lines 30-35

> Using continue and break Statements - The break statement skips the rest of the statements in the loop‟s block, jumping immediately to the next statement outside of the loop. - The continue statement is similar. It also skips the rest of the statements in the loop‟s block, but instead of terminating the loop, it transfers execution to the next iteration of the loop. It continues the loop after skipping the remaining statements in its current iteration.

Question sources

Computer Science MCQ practice collection · Question 277

Question 139 | AM0279

What is the purpose of the return statement in a function?

A. To terminate the program.

B. To declare variables.

C. To pass values to another function.

D. To return a value from the function.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: To return a value from the function.

The sum function on p.17 returns x+y+z+w and its callers display the resulting values. This shows return delivering a value to the caller. Module support: PDF p. 17, PDF p. 14.

Why each choice is right or wrong

A - Incorrect: An ordinary function return need not terminate the program.

B - Incorrect: No variable is declared by return.

C - Incorrect: Arguments pass values into a function; the example return supplies its result back.

D - Correct: Matches the sum example and caller usage.

Primary note proof

Study-note section CP15

> break exits the nearest enclosing loop or switch. continue skips the rest of the current loop iteration: a for loop then runs its update; a while loop goes directly to its condition. return leaves the current function and may supply a result. In main it ends the program normally.

Study-note section CP17

> A function groups a named task. C++ definition: int add(int a,int b){return a+b;}. Call it with add(x,y);, without parameter type declarations at the call site. Parameters are names in the definition; arguments are supplied expressions or values. A void function produces no return value but can have side effects and return; to exit early.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 17 | lines 2-10

> int sum(int x, int y, int z = 0, int w = 0) { return (x + y + z + w); } int main() { cout << sum(10, 15) << endl; cout << sum(10, 15, 25) << endl; cout << sum(10, 15, 25, 30) << endl;

Computer Programming module | PDF page 14 | lines 21-33

> Function A function is a group of statements that structure our programs in a more modular way and executed when it is called from some point of the program. It is a block of code that performs a specific task. It is important to Dividing a complex problem into smaller chunks makes our program easy to understand and reusable. When we need function? - When you need to repeat the same process over and over in a program. - The function can be called many times but appears in the code once Function - Categories C++ functions can be divided into 2 categories: • pre-defined (standard function) / library function - Which is the definitions have been written and it is ready to be used. - User needs to include pre-defined header file (i.e., math.h, time.h,cmath) To use it: FunctionName (value) e.g., sqrt (4), floor (4.3)

Question sources

Computer Science MCQ practice collection · Question 279

Question 140 | AM0280

Which options are valid definitions of an ordinary initialized constant integer x in C++? Treat each option independently and assume no custom macros or aliases. Select all that apply.

A. const int x=5;

B. final int x;

C. int const x=5;

D. int x;

Hint

Apply the naming, type and syntax rules before deciding whether an option can be used as stated.

Answer A, C: A. const int x=5;; C. int const x=5;

C++ permits const before or after the integer type specifier. Both const int x=5 and int const x=5 therefore define initialized constant integers. The initializer supplies the value required for the ordinary const object. final does not replace const in this declaration, and an unqualified int remains mutable. Treating the options independently avoids redeclaration conflicts while recognizing both equivalent const placements.

Why each choice is right or wrong

A - Correct: const int with an initializer defines the requested constant.

B - Incorrect: final is not the const qualifier in an ordinary integer declaration.

C - Correct: int const is equivalent to const int and has an initializer here.

D - Incorrect: An unqualified int variable remains mutable.

Primary note proof

Study-note section CP06

> Use int for whole numbers, float/double for fractional approximations, char for a character and bool for logical values in C++. Type sizes depend on the implementation; do not import Java widths into C++. short n, short int n and int short n denote the same signed short type. const int n=5 and int const n=5 are equivalent; an ordinary const int needs initialization. typedef creates an alias, not a distinct new type.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 8 | lines 27-33

> Constant: A constant is any expression that has a fixed value like integer, float, character and string Eg. 120, 15.2, „A‟, “Hello There” Generally, we have two ways to define constant: 1. Using Const Key word Syntax: const type constant name=value; e.g. const float PI=3.14; 2. Using #define Keyword Syntax: #define constant name value e.g. #define PI 3.14

Question sources

Computer Science MCQ practice collection · Question 280

Question 141 | AM0281

Which option describes the defining execution and test behavior of a C++ do-while loop?

A. To execute a block exactly once regardless of its condition.

B. To perform mathematical calculations.

C. To iterate over a collection of elements.

D. To execute the body before its first test and repeat while the continuation condition is true.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: D. To execute the body before its first test and repeat while the continuation condition is true.

A do-while loop executes its body before the first condition test. It then repeats while its continuation condition is true and stops when that condition becomes false. Thus at least one body execution is guaranteed when the loop is reached, but exactly one is not guaranteed. Collection traversal and arithmetic can occur inside many constructs; neither describes this loop's distinctive control rule.

Why each choice is right or wrong

A - Incorrect: A do-while loop can repeat, so it is not limited to exactly one execution.

B - Incorrect: Arithmetic is possible in the body but is not the defining control behavior.

C - Incorrect: Collection traversal is an application rather than the defining test-and-repeat rule.

D - Correct: The body runs first and further iterations require a true continuation condition.

Primary note proof

Study-note section CP14

> while(condition) checks before each body execution and may run zero times. do {body;} while(condition); runs the body once before checking; it repeats while the condition remains true. Use it when a menu or input attempt must occur at least once.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 12 | lines 2-7

> The do-while loop The condition in the do-while is evaluated after the execution of statement instead of before, granting at least one execution of statement even if condition is never fulfilled. do {statement; }while (condition); First statement is executed and then condition is evaluated. If

the outcome of the latter is nonzero then the whole process is repeated. Otherwise, the loop is terminated.

Question sources

Computer Science MCQ practice collection · Question 281

Question 142 | AM0282

What is the output of the following code snippet?

```
int i = 0;
while (i < 5) {
cout << i << " ";
i += 2;
}
```

A. 0 1 2 3 4

B. 0 2 4

C. 0 1 2 3 4 5

D. 1 3 5

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: 0 2 4

Starting at zero, printing precedes i+=2. The successive printed values are 0,2,4, followed by i=6 and a failed i<5 test. Page 7 explicitly illustrates compound addition assignment. Module support: PDF p. 11, PDF p. 7.

Why each choice is right or wrong

A - Incorrect: Would require unit increments.

B - Correct: Matches the traced states before each increment.

C - Incorrect: Includes intermediate values and 5 although the test excludes 5.

D - Incorrect: Starts at 1 instead of the specified 0.

Primary note proof

Study-note section CP14

while(condition) checks before each body execution and may run zero times. do {body;} while(condition); runs the body once before checking; it repeats while the condition remains true. Use it when a menu or input attempt must occur at least once.

Study-note section CP08

x+=y, x-=y, x*=y, x/=y and x%=y update x using the operation and conversion to its type. For simple variables, trace them as x=x op y. Starting p=q=n=8, p+=n gives 16 and p/=q gives 2. Starting x=10, x+=5 then x/=2 gives 7. x=x+x doubles; it does not generally add one.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 11 | lines 5-13

Iteration/Looping Statement: Iteration is the repetition of a statement or block of statements in a program. - Loops have as objective to repeat a statement a certain number of times or while a condition is fulfilled. - C++ has three iteration statements: the while statement, the do. While statement, and the for statement. The while loop: Its format is: while (expression) {statement;} First expression (called the loop condition) is evaluated. If the outcome is nonzero then statement (called the loop body) is executed and the whole process is repeated. Otherwise, the loop is terminated.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Computer Science MCQ practice collection · Question 282

Question 143 | AM0283

What is the purpose of the continue statement in a loop?

A. To terminate the program.

B. To skip the remaining code within the loop and move to the next iteration.

C. To repeat the loop from the beginning.

D. To execute a specific case in a switch statement.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: To skip the remaining code within the loop and move to the next iteration.

Page 13 defines continue as skipping the remaining statements of the current iteration and transferring to the next iteration. Module support: PDF p. 13.

Why each choice is right or wrong

A - Incorrect: Does not terminate the program.

B - Correct: Exactly matches the cited definition.

C - Incorrect: Too broad: it proceeds through the loop continuation rules, rather than necessarily restarting all loop setup.

D - Incorrect: Does not select a switch case.

Primary note proof

Study-note section CP15

> break exits the nearest enclosing loop or switch. continue skips the rest of the current loop iteration: a for loop then runs its update; a while loop goes directly to its condition. return leaves the current function and may supply a result. In main it ends the program normally.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 13 | lines 30-35

> Using continue and break Statements - The break statement skips the rest of the statements in the loop‟s block, jumping immediately to the next statement outside of the loop. - The continue statement is similar. It also skips the rest of the statements in the loop‟s block, but instead of terminating the loop, it transfers execution to the next iteration of the loop. It continues the loop after skipping the remaining statements in its current iteration.

Question sources

Computer Science MCQ practice collection · Question 283

Question 144 | AM0284

What is the purpose of the modulo operator (%) in C++?

A. To perform addition.

B. To perform multiplication.

C. To perform division.

D. To calculate the remainder of integer division.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer D: To calculate the remainder of integer division.

The arithmetic list on p.7 includes %, and p.8 evaluates c=a%b++ for a=10,b=3; the previously inspected output c=1 demonstrates the remainder interpretation. Module support: PDF p. 7, PDF p. 8.

Why each choice is right or wrong

A - Incorrect: Addition is +.

B - Incorrect: Multiplication is *.

C - Incorrect: The quotient operation is /; the example produces a remainder.

D - Correct: 10 divided by 3 leaves the illustrated remainder 1.

Primary note proof

Study-note section CP08

> Multiplication, division and remainder have higher precedence than addition and subtraction; equal-precedence arithmetic groups left to right. Integer division truncates toward zero. For integers, a == (a/b)*b + a%b when the division is valid. % requires integral operands in C++; (8.0*3)%5 is invalid. Division by zero and signed integer overflow cannot be given ordinary numeric outputs.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, | |) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 | | 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto

decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Computer Programming module | PDF page 8 | lines 12-23

{int a=10,b=3,c,d,e,f; c=a%b++; cout<<"c="<<c<<endl; b+=2; cout<<"b="<<b<<endl; d=++a+b; cout<<"d="<<d<<endl; e=a>b?a:b; e++; cout<<"e="<<e++<<endl; f=a+b+c+d+e; cout<<"f="<<f;

Question sources

Computer Science MCQ practice collection · Question 284

Question 145 | AM0285

What is the output of the following code snippet?

```
int x = 10;
int y = 3;
cout << x / y;
```

A. 3.33333

B. 3

C. 3.0

D. 3.33333...

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: 3.

Both x and y have type int, so CP08's integer-division rule applies: 10/3 truncates toward zero to 3. The stream prints that integer value. No floating conversion occurs before division, and no decimal formatting is requested, so neither a fractional quotient nor the literal output 3.0 is correct.

Why each choice is right or wrong

A - Incorrect: 3.33333 is a fractional approximation, but division of these int operands discards the fractional part.

B - Correct: Integer division gives 3, and cout prints the resulting integer without a decimal suffix.

C - Incorrect: 3.0 is not the printed representation of the int quotient in this code.

D - Incorrect: An indefinitely repeating decimal is not produced by integer division or this stream insertion.

Primary note proof

Study-note section CP08

Multiplication, division and remainder have higher precedence than addition and subtraction; equal-precedence arithmetic groups left to right. Integer division truncates toward zero. For integers, a == (a/b)*b + a%b when the division is valid. % requires integral operands in C++; (8.0*3)%5 is invalid. Division by zero and signed integer overflow cannot be given ordinary numeric outputs.

Study-note section CP28

Read a full line with getline(cin,text). After formatted numeric extraction, consume the pending newline before a line read when required. Input failure must be tested before trusting a value. Console output formatting matters: by default a double storing an exact 4 may print 4 rather than 4.0. Text, spaces and newlines are part of an output trace.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, | |) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 | | 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Computer Science MCQ practice collection · Question 285

Question 146 | AM0286

Which listed C++ construct computes a number raised to a power, using the standard mathematical library?

A. ^

B. **

C. pow()

D. ^=

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: C. pow()

C++ supplies pow as a mathematical library function for exponentiation. It is a function call rather than a built-in exponentiation operator. The caret denotes bitwise XOR, and ** is not a C++ power operator; ^= consequently does not provide exponentiation assignment. With the required mathematical declarations available, pow is the construct matching the requested operation, so the intended C key remains valid.

Why each choice is right or wrong

A - Incorrect: ^ is bitwise XOR rather than exponentiation.

B - Incorrect: ** is not a C++ exponentiation operator.

C - Correct: The standard-library pow function computes powers.

D - Incorrect: ^= uses the bitwise operation, not exponentiation.

Primary note proof

Study-note section CP30

> C++ has no built-in exponentiation operator: ^ is bitwise XOR and ** is not a power operator. To compute b^e for a nonnegative integer e, start r=1 and repeat r*=b exactly e times; b=4,e=3 gives 64. Large powers and factorials can overflow fixed-width types; specify a valid input range.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 15 | lines 2-7

> - ceil : smallest integral value not less than x - fmod : floating-point remainder function - log : logarithm - pow : Raise a number by a power. - sin : The sine of an integer. - sqrt : Square root of a number.

Question sources

Computer Science MCQ practice collection · Question 286

Question 147 | AM0287

What is the output of the following code snippet?

```
double result = sqrt(16);
cout << result;
```

A. 2

B. 4

C. 8

D. 16

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: 4

Page 15 defines sqrt as square root. Since 4 multiplied by 4 is 16, sqrt(16) is 4. Its worked example also displays sqrt(25) as 5, supporting the plain integral-looking output format. Module support: PDF p. 15, PDF p. 15.

Why each choice is right or wrong

A - Incorrect: 2 squared is 4, not 16.

B - Correct: 4 squared equals 16.

C - Incorrect: 8 squared is 64.

D - Incorrect: 16 is the argument, not its square root.

Primary note proof

Study-note section CP30

> Use <cmath> for sqrt, pow, floor and ceil; integer abs is also provided by <cstdlib>. sqrt(16)=4; pow(3,4)=81; abs(-7)=7. ceil(x) is the least integer-valued result >=x; floor(x) the greatest <=x. Thus ceil(-2.7)=-2 and floor(-2.7)=-3; truncation toward zero differs from floor for negatives.

Study-note section CP28

> Read a full line with getline(cin,text). After formatted numeric extraction, consume the pending newline before a line read when required. Input failure must be tested before trusting a value. Console output formatting matters: by default a double storing an exact 4 may print 4 rather than 4.0. Text, spaces and newlines are part of an output trace.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 15 | lines 2-7

- ceil : smallest integral value not less than x - fmod : floating-point remainder function - log : logarithm - pow : Raise a number by a power. - sin : The sine of an integer. - sqrt : Square root of a number.

Computer Programming module | PDF page 15 | lines 10-21

Example #include<iostream> #include <cmath> using namespace std; int main() {double x=-25; float y=36.5; int z=25; cout<<"absolute value:"<<fabs(x)<<endl; cout<<floor(y)<<endl; cout<<ceil(y)<<endl; cout<<sqrt(z);

Question sources

Computer Science MCQ practice collection · Question 287

Question 148 | AM0288

Which header file should be included to use mathematical functions in C++?

A. <iostream>

B. <cmath>

C. <string>

D. <iomanip>

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: <cmath>

The module math-function example on p.15 includes <cmath> before using fabs, floor, ceil and sqrt. This directly supports B among these headers. Module support: PDF p. 15.

Why each choice is right or wrong

A - Incorrect: The example also includes iostream for streams; the math header is cmath.

B - Correct: This is the demonstrated math header.

C - Incorrect: string is not the math header in the cited example.

D - Incorrect: iomanip is not the math header demonstrated here.

Primary note proof

Study-note section CP30

> Use <cmath> for sqrt, pow, floor and ceil; integer abs is also provided by <cstdlib>. sqrt(16)=4; pow(3,4)=81; abs(-7)=7. ceil(x) is the least integer-valued result >=x; floor(x) the greatest <=x. Thus ceil(-2.7)=-2 and floor(-2.7)=-3; truncation toward zero differs from floor for negatives.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 15 | lines 10-21

> Example #include<iostream> #include <cmath> using namespace std; int main() {double x=-25; float y=36.5; int z=25; cout<<"absolute value:"<<fabs(x)<<endl; cout<<floor(y)<<endl; cout<<ceil(y)<<endl; cout<<sqrt(z);

Question sources

Computer Science MCQ practice collection · Question 288

Question 149 | AM0289

What is the output of the following code snippet?

```
int x = 5;
int y = 2;
cout << x % y;
```

A. 0

B. 1

C. 2

D. 3

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: 1

Using the remainder operation illustrated by c=10%3 on p.8, 5=2*2+1 leaves remainder 1. Module support: PDF p. 8.

Why each choice is right or wrong

A - Incorrect: Zero would require exact divisibility.

B - Correct: The remainder after two whole groups of 2 is 1.

C - Incorrect: 2 is the whole-number quotient, not the remainder.

D - Incorrect: A remainder of 3 would exceed the positive divisor 2.

Primary note proof

Study-note section CP08

> Multiplication, division and remainder have higher precedence than addition and subtraction; equal-precedence arithmetic groups left to right. Integer division truncates toward zero. For integers, a == (a/b)*b + a%b when the division is valid. % requires integral operands in C++; (8.0*3)%5 is invalid. Division by zero and signed integer overflow cannot be given ordinary numeric outputs.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 8 | lines 12-23

> {int a=10,b=3,c,d,e,f; c=a%b++; cout<<"c="<<c<<endl; b+=2; cout<<"b="<<b<<endl; d=++a+b; cout<<"d="<<d<<endl; e=a>b?a:b; e++; cout<<"e="<<e++<<endl; f=a+b+c+d+e; cout<<"f="<<f;

Question sources

Computer Science MCQ practice collection · Question 289

Question 150 | AM0290

What is the output of the following code snippet?

```
double result = pow(3, 4);
cout << result;
```

A. 7

B. 12

C. 27

D. 81

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer D: 81

The p.15 definition of pow gives 3 raised to 4: 3*3*3*3=81. Module support: PDF p. 15.

Why each choice is right or wrong

A - Incorrect: 7 is addition of the arguments.

B - Incorrect: 12 is multiplication of the arguments.

C - Incorrect: 27 uses exponent 3 instead of 4.

D - Correct: 81 is the fourth power.

Primary note proof

Study-note section CP30

> Use <cmath> for sqrt, pow, floor and ceil; integer abs is also provided by <cstdlib>. sqrt(16)=4; pow(3,4)=81; abs(-7)=7. ceil(x) is the least integer-valued result >=x; floor(x) the greatest <=x. Thus ceil(-2.7)=-2 and floor(-2.7)=-3; truncation toward zero differs from floor for negatives.

Study-note section CP28

> Read a full line with getline(cin,text). After formatted numeric extraction, consume the pending newline before a line read when required. Input failure must be tested before trusting a value. Console output formatting matters: by default a double storing an exact 4 may print 4 rather than 4.0. Text, spaces and newlines are part of an output trace.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 15 | lines 2-7

- ceil : smallest integral value not less than x - fmod : floating-point remainder function - log : logarithm - pow : Raise a number by a power. - sin : The sine of an integer. - sqrt : Square root of a number.

Question sources

Computer Science MCQ practice collection · Question 290

Question 151 | AM0291

What is the purpose of the abs() function in C++?

A. To calculate the absolute value of a number.

B. To calculate the average of two numbers.

C. To round a floating-point number to the nearest integer.

D. To compare two numbers for equality.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: To calculate the absolute value of a number.

Page 14 distinguishes fabs for a floating-point absolute value and abs for an integer. Thus abs supplies the integer absolute value. Module support: PDF p. 14.

Why each choice is right or wrong

A - Correct: Matches the stated integer absolute-value role.

B - Incorrect: No averaging role is given.

C - Incorrect: Rounding is not the abs role; floor and ceil are separate.

D - Incorrect: Equality is a relational operation, not abs.

Primary note proof

Study-note section CP30

> Use <cmath> for sqrt, pow, floor and ceil; integer abs is also provided by <cstdlib>. sqrt(16)=4; pow(3,4)=81; abs(-7)=7. ceil(x) is the least integer-valued result >=x; floor(x) the greatest <=x. Thus ceil(-2.7)=-2 and floor(-2.7)=-3; truncation toward zero differs from floor for negatives.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 14 | lines 34-36

> The Standard Math Library - fabs : absolute value of floating point number, #include<cstdlib> to call abs() for integer number -floor: largest integral value not greater than x

Question sources

Computer Science MCQ practice collection · Question 291

Question 152 | AM0292

What is the output of the following code snippet?

```
int x = -7;
cout << abs(x);
```

A. -7

B. 7

C. 0

D. Undefined

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: 7

Page 14 identifies abs for integers. The absolute value is distance from zero, so -7 gives 7. Module support: PDF p. 14.

Why each choice is right or wrong

A - Incorrect: Keeps the negative sign instead of taking magnitude.

B - Correct: Magnitude of -7 is 7.

C - Incorrect: Only zero has zero magnitude.

D - Incorrect: No undefined input is present for this small integer under the stated function assumption.

Primary note proof

Study-note section CP30

> Use <cmath> for sqrt, pow, floor and ceil; integer abs is also provided by <cstdlib>. sqrt(16)=4; pow(3,4)=81; abs(-7)=7. ceil(x) is the least integer-valued result >=x; floor(x) the greatest <=x. Thus ceil(-2.7)=-2 and floor(-2.7)=-3; truncation toward zero differs from floor for negatives.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 14 | lines 34-36

> The Standard Math Library - fabs : absolute value of floating point number, #include<cstdlib> to call abs() for integer number -floor: largest integral value not greater than x

Question sources

Computer Science MCQ practice collection · Question 292

Question 153 | AM0293

What is the purpose of the ceil() function in C++?

A. To calculate the ceiling value of a floating-point number.

B. To calculate the factorial of an integer.

C. To round a floating-point number down to the nearest integer.

D. To perform bitwise complement on an integer.

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: To calculate the ceiling value of a floating-point number.

Page 15 defines ceil as the smallest integral value not less than x. This is the ceiling operation. Module support: PDF p. 15.

Why each choice is right or wrong

A - Correct: Matches the exact ceiling definition.

B - Incorrect: Factorial is unrelated to the cited definition.

C - Incorrect: Rounding downward corresponds to floor, not ceil.

D - Incorrect: No bitwise operation appears in the definition.

Primary note proof

Study-note section CP30

> Use <cmath> for sqrt, pow, floor and ceil; integer abs is also provided by <cstdlib>. sqrt(16)=4; pow(3,4)=81; abs(-7)=7. ceil(x) is the least integer-valued result >=x; floor(x) the greatest <=x. Thus ceil(-2.7)=-2 and floor(-2.7)=-3; truncation toward zero differs from floor for negatives.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 15 | lines 2-7

> - ceil : smallest integral value not less than x - fmod : floating-point remainder function - log : logarithm - pow : Raise a number by a power. - sin : The sine of an integer. - sqrt : Square root of a number.

Question sources

Computer Science MCQ practice collection · Question 293

Question 154 | AM0294

Which of the following data types in C++ can store fractional values?

A. int

B. char

C. float

D. bool

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: C. float

The float type represents fractional numeric approximations, so it can retain a fractional component. int represents whole numbers; converting a fractional value to it does not preserve that component. char represents a character, and bool represents the logical values true and false. These different value domains make float the unique listed type intended for fractional quantities, without implying that every real number is represented exactly.

Why each choice is right or wrong

A - Incorrect: An int stores a whole-number result rather than a retained fractional component.

B - Incorrect: A char represents a character, not the listed fractional numeric type.

C - Correct: A float represents fractional numeric approximations.

D - Incorrect: A bool represents logical true/false values rather than fractional magnitudes.

Primary note proof

Study-note section CP06

> Use int for whole numbers, float/double for fractional approximations, char for a character and bool for logical values in C++. Type sizes depend on the implementation; do not import Java widths into C++. short n, short int n and int short n denote the same signed short type. const int n=5 and int const n=5 are equivalent; an ordinary const int needs initialization. typedef creates an alias, not a distinct new type.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 24-33

> 5. Data Types, Variables, and Constants Variable: - Variable is a symbolic name for a memory location in which data can be stored. - Attribute of Variables: o A type, which is, established when the variable is defined (e.g., integer, float, character) o A value, which can be changed by assigning a new value to the variable Variable Declaration - Declaring a variable means defining (creating) a variable. - Syntax: data type variable name; e.g. int num; - We can create More Than One Variable at a Time e.g., int num1, num2, num3;

Computer Programming module | PDF page 15 | lines 15-21

```
{double x=-25; float y=36.5; int z=25; cout<<"absolute value:"<<fabs(x)<<endl;
cout<<floor(y)<<endl; cout<<ceil(y)<<endl; cout<<sqrt(z);
```

Question sources

Computer Science MCQ practice collection · Question 294

Question 155 | AM0295

What is the result of the expression 5 / 2 in C++?

A. 2

B. 2.5

C. 2.0

D. 2.25

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: The integer value 2.

Both literals in 5/2 are integers. CP07 states that the integer quotient is 2, and even assigning that already-computed quotient to double would not restore the lost fraction. The question asks for the expression's result, so the integer 2 is the precise choice rather than a floating representation or fractional quotient.

Why each choice is right or wrong

A - Correct: Two integer operands produce the integer quotient 2 after truncating the fractional part.

B - Incorrect: 2.5 requires floating division, which these uncast integer literals do not perform.

C - Incorrect: 2.0 is a floating representation, while this expression itself has an integer result.

D - Incorrect: 2.25 follows neither integer truncation nor the ordinary mathematical quotient of five divided by two.

Primary note proof

Study-note section CP07

With int x=5,y=2, x/y is 2. Assigning that quotient to double gives 2.0, not 2.5. static_cast<double>(x)/y performs floating division and yields 2.5. Converting an in-range fractional floating value to int truncates toward zero. Cast before the operation if a fractional result is needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Computer Science MCQ practice collection · Question 295

Question 156 | AM0296

What is the output of the following code snippet?

```
int x = 5;
int y = 2;
double result = static_cast<double>(x) / y;
cout << result;
```

A. 2

B. 2.5

C. 2.0

D. 2.25

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: 2.5.

static_cast<double>(x) converts 5 to a floating value before the division takes place. CP07 explicitly gives this form as yielding 2.5 when x=5 and y=2. The result therefore retains the fractional part; storing it in double does not first perform an integer quotient.

Why each choice is right or wrong

A - Incorrect: Casting x before division prevents the integer truncation that would have produced 2.

B - Correct: The converted floating operand makes the quotient 2.5, which is stored and printed.

C - Incorrect: 2.0 would result from first computing the integer quotient and then converting it, unlike this code.

D - Incorrect: The floating quotient of 5 by 2 is 2.5, not 2.25.

Primary note proof

Study-note section CP07

With int x=5,y=2, x/y is 2. Assigning that quotient to double gives 2.0, not 2.5. static_cast<double>(x)/y performs floating division and yields 2.5. Converting an in-range fractional floating value to int truncates toward zero. Cast before the operation if a fractional result is needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 28 | lines 25-41

void pointers - Void pointer point to any data type Example: void *ptr; int x=5; Ptr=&x; #include <iostream> using namespace std; int main() {int a=10; float b=20; void*ptr; ptr=&a; cout<<*(static_cast<int*>(ptr))<<endl; //cout<<*((int*)ptr) ptr=&b; cout<<*((float*)ptr)<<endl;

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Computer Science MCQ practice collection · Question 296

Question 157 | AM0297

What is the value of x after executing the following code snippet?

```
int x = 10;
x += 5;
x /= 2;
```

A. 5

B. 7

C. 8

D. 10

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: x becomes 7.

The initial x is 10. x+=5 updates it to 15; x/=2 then performs integer division because x is int, truncating 15/2 toward zero to 7. CP08 provides this exact sequence and result. The updates happen in order, so the second operation acts on 15 rather than the original 10.

Why each choice is right or wrong

A - Incorrect: Five divides the original value by two and ignores the preceding addition of five.

B - Correct: The sequence gives 10+5=15, then integer 15/2=7.

C - Incorrect: Integer division truncates 7.5 to 7 rather than rounding it to 8.

D - Incorrect: Both compound assignments update x; it does not retain its initial value of 10.

Primary note proof

Study-note section CP08

> x+=y, x-=y, x*=y, x/=y and x%=y update x using the operation and conversion to its type. For simple variables, trace them as x=x op y. Starting p=q=n=8, p+=n gives 16 and p/=q gives 2. Starting x=10, x+=5 then x/=2 gives 7. x=x+x doubles; it does not generally add one.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Computer Science MCQ practice collection · Question 297

Question 158 | AM0298

What is the result of the following expression?

bool result = (5 > 3) && (10 < 8);

A. true

B. false

C. compilation error

D. runtime error

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: false

Page 7 lists relational and logical operators and represents true and false as 1 and 0. Here 5>3 is true and 10<8 is false; logical AND requires both, so result is false. Module support: PDF p. 7.

Why each choice is right or wrong

A - Incorrect: Would require both comparisons to be true.

B - Correct: The second comparison is false, making the conjunction false.

C - Incorrect: The operators are listed module constructs, and the declaration follows the taught type-name form.

D - Incorrect: No runtime failure is implied by either simple comparison.

Primary note proof

Study-note section CP10

> Relational and equality operators test values; && and || combine truth values; ! negates. In C++, zero converts to false and a nonzero number to true. Streams display bool as 0 or 1 unless boolalpha is set. With a=3,b=5, cout<<(a==b) prints 0.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

> Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto

decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Question sources

Computer Science MCQ practice collection · Question 298

Question 159 | AM0299

What is the output of the following code snippet?

```
int x = 3;
int y = 5;
cout << (x == y);
```

A. 1

B. 0

C. true

D. false

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: 0

Page 7 explicitly says relational operators evaluate to 1 for true or 0 for false. Since 3 and 5 differ, x==y evaluates to 0; cout displays that result (p.6). Module support: PDF p. 7, PDF p. 6.

Why each choice is right or wrong

A - Incorrect: 1 would mean equality is true.

B - Correct: 0 represents the false equality.

C - Incorrect: The equality is not true.

D - Incorrect: This names the logical result but differs from the numeric representation taught by the module.

Primary note proof

Study-note section CP10

Relational and equality operators test values; && and || combine truth values; ! negates. In C++, zero converts to false and a nonzero number to true. Streams display bool as 0 or 1 unless boolalpha is set. With a=3,b=5, cout<<(a==b) prints 0.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Computer Programming module | PDF page 6 | lines 4-15

6. Communication through console cout (the standard output stream) o used to display information for the user Syntax: cout<<object; • Object can be variable or literals Example: int x=5; cout<<x cout<< “Hello there”; cin(the standard input stream): o used to accept information from the user. Syntax: cin>>variable name; Eg. cin>>x;

Question sources

Computer Science MCQ practice collection · Question 299

Question 160 | AM0300

What is the value of result after executing the following code snippet?

```
int a = 3;
int b = 2;
int result = pow(a, b);
```

A. 6

B. 9

C. 8

D. 5

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: 9

The module defines pow as raising a number to a power (p.15). With a=3 and b=2, it computes 3*3=9; this whole-number value is the intended integer result. Module support: PDF p. 15.

Why each choice is right or wrong

A - Incorrect: 6 multiplies base and exponent.

B - Correct: 9 is 3 squared.

C - Incorrect: 8 is 2 cubed, reversing arguments.

D - Incorrect: 5 adds the arguments.

Primary note proof

Study-note section CP30

> Use <cmath> for sqrt, pow, floor and ceil; integer abs is also provided by <cstdlib>. sqrt(16)=4; pow(3,4)=81; abs(-7)=7. ceil(x) is the least integer-valued result >=x; floor(x) the greatest <=x. Thus ceil(-2.7)=-2 and floor(-2.7)=-3; truncation toward zero differs from floor for negatives.

Study-note section CP07

> With int x=5,y=2, x/y is 2. Assigning that quotient to double gives 2.0, not 2.5. static_cast<double>(x)/y performs floating division and yields 2.5. Converting an in-range fractional floating value to int truncates toward zero. Cast before the operation if a fractional result is needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 15 | lines 2-7

> - ceil : smallest integral value not less than x - fmod : floating-point remainder function - log : logarithm - pow : Raise a number by a power. - sin : The sine of an integer. - sqrt : Square root of a number.

Question sources

Computer Science MCQ practice collection · Question 300

Question 161 | AM0301

What is the output of the following code snippet?

```
int x = -5;
cout << abs(x);
```

A. -5

B. 5

C. 0

D. compilation error

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: 5

Page 14 identifies integer abs. The magnitude of -5 is 5, so cout displays 5. Module support: PDF p. 14.

Why each choice is right or wrong

A - Incorrect: Retains the negative sign.

B - Correct: Gives the magnitude of -5.

C - Incorrect: -5 is not zero.

D - Incorrect: With the required declarations, this is the integer abs usage described in the module.

Primary note proof

Study-note section CP30

> Use <cmath> for sqrt, pow, floor and ceil; integer abs is also provided by <cstdlib>. sqrt(16)=4; pow(3,4)=81; abs(-7)=7. ceil(x) is the least integer-valued result >=x; floor(x) the greatest <=x. Thus ceil(-2.7)=-2 and floor(-2.7)=-3; truncation toward zero differs from floor for negatives.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 14 | lines 34-36

> The Standard Math Library - fabs : absolute value of floating point number, #include<cstdlib> to call abs() for integer number -floor: largest integral value not greater than x

Question sources

Computer Science MCQ practice collection · Question 301

Question 162 | AM0302

What is the value of result after executing the following code snippet?

```
double pi = 3.14159;
int result = floor(pi);
```

A. 3

B. 4

C. 3.14

D. 3.14159

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer A: 3

Page 14 defines floor as the largest integral value not greater than x. For 3.14159 that value is 3, because 3<=3.14159<4. Module support: PDF p. 14.

Why each choice is right or wrong

A - Correct: 3 is the greatest integer not exceeding pi.

B - Incorrect: 4 is greater than pi and would be a ceiling.

C - Incorrect: 3.14 remains fractional and is not floor.

D - Incorrect: 3.14159 is the original argument, not its floor.

Primary note proof

Study-note section CP30

> Use <cmath> for sqrt, pow, floor and ceil; integer abs is also provided by <cstdlib>. sqrt(16)=4; pow(3,4)=81; abs(-7)=7. ceil(x) is the least integer-valued result >=x; floor(x) the greatest <=x. Thus ceil(-2.7)=-2 and floor(-2.7)=-3; truncation toward zero differs from floor for negatives.

Study-note section CP07

> With int x=5,y=2, x/y is 2. Assigning that quotient to double gives 2.0, not 2.5. static_cast<double>(x)/y performs floating division and yields 2.5. Converting an in-range fractional floating value to int truncates toward zero. Cast before the operation if a fractional result is needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 14 | lines 34-36

> The Standard Math Library - fabs : absolute value of floating point number, #include<cstdlib> to call abs() for integer number -floor: largest integral value not greater than x

Question sources

Computer Science MCQ practice collection · Question 302

Question 163 | AM0303

What is the output of the following code snippet?

```
int x = 4;
int y = 2;
cout << (x % y);
```

A. 0

B. 1

C. 2

D. 4

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer A: 0

The remainder operation illustrated on p.8 gives 4%2=0 because 4=2*2+0. Module support: PDF p. 8.

Why each choice is right or wrong

A - Correct: Exact divisibility leaves zero remainder.

B - Incorrect: A remainder of 1 would represent an odd dividend.

C - Incorrect: 2 is the quotient and cannot be the remainder for positive divisor 2.

D - Incorrect: 4 is the original dividend, not its remainder.

Primary note proof

Study-note section CP08

> Multiplication, division and remainder have higher precedence than addition and subtraction; equal-precedence arithmetic groups left to right. Integer division truncates toward zero. For integers, a == (a/b)*b + a%b when the division is valid. % requires integral operands in C++; (8.0*3)%5 is invalid. Division by zero and signed integer overflow cannot be given ordinary numeric outputs.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 8 | lines 12-23

{int a=10,b=3,c,d,e,f; c=a%b++; cout<<"c="<<c<<endl; b+=2; cout<<"b="<<b<<endl; d=++a+b; cout<<"d="<<d<<endl; e=a>b?a:b; e++; cout<<"e="<<e+++<<endl; f=a+b+c+d+e; cout<<"f="<<f;

Question sources

Computer Science MCQ practice collection · Question 303

Question 164 | CS0086

What is the output of the following C++ fragment code? int x = 15, y = 2; int z = x / y; cout << z;

A. 7.5

B. 7

C. 3

D. 12.5

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer B: 7.

x and y are int variables with values 15 and 2. Their division therefore produces integer 7 before assignment to z, as CP08 specifies truncation toward zero. z also has type int and cout prints 7. None of the fractional or unrelated numerical options follows from this expression.

Why each choice is right or wrong

A - Incorrect: 7.5 is the mathematical fractional quotient, but these int operands produce an integer quotient.

B - Correct: The integer quotient 15/2 truncates toward zero to 7, which z stores and cout prints.

C - Incorrect: Three is not the quotient obtained by dividing the stated values 15 and 2.

D - Incorrect: 12.5 is neither the integer quotient nor the fractional quotient of 15 divided by 2.

Primary note proof

Study-note section CP08

Multiplication, division and remainder have higher precedence than addition and subtraction; equal-precedence arithmetic groups left to right. Integer division truncates toward zero. For integers, a == (a/b)*b + a%b when the division is valid. % requires integral operands in C++; (8.0*3)%5 is invalid. Division by zero and signed integer overflow cannot be given ordinary numeric outputs.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 7 | lines 5-23

Assignment Operators (=) Example: int x=5; or used with a combination of other operators‟ example, x+=2;//x=x+2, x=y=10; Arithmetic Operators (+, -, *, /, %) E.g. Assume int X=9, Y=2 X+Y, Y-X, X%Y Relational operators (==,!=, <, <=,>,>=) Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome). Assume int X=9, Y=2 X==Y, X!=Y, X>Y 'A' < 'F' // gives 1 (is like 65 < 70) Logical operators (! Negation, &&, ||) Like the relational operators, logical operators evaluate to 1 or 0. Example: 5 < 6 || 6 < 5 // gives 1 In general, any nonzero value can be used to represent the logical true, whereas only zero represents the logical false !12 // gives 0 10 && 5 // gives 1 Increment/decrement Operators The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. Both operators can be used in prefix and postfix form.

Computer Programming module | PDF page 5 | lines 24-33

5. Data Types, Variables, and Constants Variable: - Variable is a symbolic name for a memory location in which data can be stored. - Attribute of Variables: o A type, which is, established when the variable is defined (e.g., integer, float, character) o A value, which can be changed by assigning a new value to the variable Variable Declaration - Declaring a variable means defining (creating) a variable. - Syntax: data type variable name; e.g. int num; - We can create More Than One Variable at a Time e.g., int num1, num2, num3;

Question sources

Computer Science Exit Exam practice collection · Question 86

Question 165 | CS0087

Writing comments in between our C++ program makes our program _____.

A. Portable

B. Maintainable

C. Reliable

D. Efficient

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer B: Maintainable

Page 5 says comments explain aspects of a program for human readers and are ignored by the compiler. Explanations help a maintainer understand and change code, so maintainable is the best fit. Module support: PDF p. 5.

Why each choice is right or wrong

A - Incorrect: Human explanatory text does not establish cross-platform compatibility.

B - Correct: Explanatory text helps people understand code during maintenance.

C - Incorrect: Comments alone do not establish correctness or reliable execution.

D - Incorrect: Ignored comments do not establish runtime efficiency.

Primary note proof

Study-note section CP05

> Comments use // to the line end or /* ... */ for a block. They explain intent and assumptions for maintenance; ordinary comments do not execute or automatically improve speed, reliability or portability. main is the conventional entry function name, not a C++ keyword.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 5 | lines 18-20

> 4. Comments A comment is a piece of descriptive text which explains some aspect of a program. Program comments are totally ignored by the compiler and are only intended for human readers.

Question sources

Computer Science Exit Exam practice collection · Question 87

Question 166 | CS0088

What is the output of the following C++ fragment code? int data[10] = {5,10,15,20,25,30,35,40,45,50}; int i = 0; while (i <= 10) { if(data[i] % 10 == 0) { cout << data[i] << " "; i = i + 2; } }

A. 5 15 25 35 45 55

B. 10 20 30 40 50

C. 5 10 15 20 25 30 35 40 45 50

D. No output

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer D: No output

Arrays are zero indexed (p.19), so initially data[0]=5. Its remainder on division by 10 is 5, not zero, so the if body is skipped. The only increment of i lies inside that skipped body. Therefore i stays zero, i<=10 remains true, and no cout is ever reached. The loop runs indefinitely with no output; it never reaches the potentially invalid index 10. Module support: PDF p. 19, PDF p. 11, PDF p. 8, PDF p. 9.

Why each choice is right or wrong

A - Incorrect: No print occurs at the first iteration or later, so this odd-value sequence cannot appear.

B - Incorrect: The loop never advances to data[1]=10.

C - Incorrect: No array traversal occurs because i never changes.

D - Correct: Correct: nothing is printed, and the loop does not terminate.

Primary note proof

Study-note section CP16

> For n=1; n<=18; n+=2, print n unless n%7==0 then break: output is 1 3 5. A while(a>b) with fixed a=6,b=5 never changes its true condition. In data[10]={5,10,...,50}, i=0, advancing i only if data[i]%10==0 causes no progress because data[0]=5. It prints nothing and loops indefinitely; changing <=10 to <10 alone does not repair progress.

Study-note section CP22

> A built-in C++ array has a fixed number of same-type elements in contiguous storage: int a[4]={10,20,30,40};. Valid indices are 0 through 3. a[4] is outside the array; accessing it has undefined behavior. Java checks bounds and throws ArrayIndexOutOfBoundsException. Do not predict a C++ out-of-range value from nearby memory.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 19 | lines 4-29

Arrays - an array is a collection of fixed-size number of elements of the same type stored sequentially in memory. - An individual element of an array is identified by its own unique index (or subscript). - The number of elements must be an integer. - The size of the array is referred to as its dimension - Array in C++ is Zero indexed. Declaring of one-dimensional Array Syntax: elementstype arrayname [size]; → You can use constants such as macro definitions or const ints as the sizes of arrays. Example: int num[5]; string fruit[3]; #define numstudents 100 or const int numstudents=100; double Score[numstudents] ; double Score[numstudents] ; Initialization of an Array - Arrays can be initialized at declaration time in two different ways: Syntax: elementstype arrayname[size] = {list each value separated by comma}; E.g int num[5]={1,2,3,4,5}; elementstype arrayname[ ] = { list with any number of values }; E.g int num[]={1,2,3,4,5}; string fruit[]={“Apple”,“orange”, “Mango”}; Accessing Array Elements - Each element could be accessed using an index into the array - When accessing an array, the index given must be a positive integer from 0 to n-1, where n is the dimension of the array.

Computer Programming module | PDF page 11 | lines 5-13

Iteration/Looping Statement: Iteration is the repetition of a statement or block of statements in a program. - Loops have as objective to repeat a statement a certain number of times or while a condition is fulfilled. - C++ has three iteration statements: the while statement, the do. While statement, and the for statement. The while loop: Its format is: while (expression) {statement;} First expression (called the loop condition) is evaluated. If the outcome is nonzero then statement (called the loop body) is executed and the whole process is repeated. Otherwise, the loop is terminated.

Computer Programming module | PDF page 8 | lines 12-23

{int a=10,b=3,c,d,e,f; c=a%b++; cout<<"c="<<c<<endl; b+=2; cout<<"b="<<b<<endl; d=++a+b; cout<<"d="<<d<<endl; e=a>b?a:b; e++; cout<<"e="<<e++<<endl; f=a+b+c+d+e; cout<<"f="<<f;

Computer Programming module | PDF page 9 | lines 9-21

Conditional Statements A. The if Statement Syntax: if (expression) statement; First expression is evaluated. If the outcome is nonzero (true) then statement is executed. Otherwise, nothing happens. if-else statement Syntax: if (expression) statement1; else statement2; First expression is evaluated. If the outcome is nonzero (true) then statement1 is executed. Otherwise, statement2 is executed.

Question sources

Computer Science Exit Exam practice collection · Question 88

Question 167 | CS0092

After a compiler reports a syntax error and before rebuilding, which action directly helps locate and repair its cause?

A. Rewrite the entire program

B. Ignore the error and continue execution

C. Use a debugger tool

D. Check the code for typos or incorrect syntax

Hint

Apply the naming, type and syntax rules before deciding whether an option can be used as stated.

Answer D: D. Check the code for typos or incorrect syntax

A syntax error violates the language grammar, so inspecting the relevant tokens and nearby declarations is the direct diagnostic step. The note recommends starting with the first useful compiler diagnostic, repairing the code and rebuilding afterward. Rewriting everything is unnecessary, ignoring the diagnostic leaves the defect in place, and a debugger is useful after a runnable program exists. The revised timing asks for inspection rather than the final verification step.

Why each choice is right or wrong

A - Incorrect: A complete rewrite is unnecessary for locating a local grammar mistake.

B - Incorrect: Ignoring a compiler error leaves the cause unresolved.

C - Incorrect: A debugger is useful after a runnable program exists.

D - Correct: Inspecting nearby tokens and declarations helps locate and correct grammar mistakes.

Primary note proof

Study-note section CP32

> A syntax error violates grammar; a type error can be syntactically well formed but rejected during semantic checking. A runtime error occurs during execution; a logic error gives the wrong behavior. A bug is a defect; debugging locates and corrects it. Read the first useful compiler diagnostic, inspect nearby tokens and declarations, repair and rebuild; a debugger is useful after a runnable program exists.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Computer Programming module | PDF page 4 | lines 21-30

> The compilation process of C++ program consists of the following steps. • edit source code → compile → link → execute (Editor) (Compiler) (Linker) (Loader) I. Program source code is entered into a file using a text editor. II. After the code has been entered, a compiler program is started that translates the source into an object code file. III. The object code file is linked with other

object code files that come with the compiler and an executable file (or program) is created. IV. In order to execute the program, a program called the loader copies the executable file into the memory of the computer and sends an execute command to the CPU.

Computer Programming module | PDF page 2 | lines 2-18

1. Introduction to programming Computer programming is the process of writing, testing, debugging/troubleshooting, and maintaining the source code of computer programs. Programs are the instructions that tells the computer what to do .so, writing computer programs means writing instructions, that will make the computer follow and run a program based on those instructions. We are going to write computer programs to solve problems. Problem Solving Problem: gap between the existing and the desired or proposed situation where problem solving will try to fill this gap. Problem solving: The process of transforming the description of a problem into the solution by using our knowledge of the problem-solving strategies, techniques and tools. Problem solving techniques • Analyze the problem: Carefully understand the problem • Outline the problem requirements • Does program require user interaction? • Does program manipulate data? • What is the output? • Design steps (algorithm) to solve the problem

Question sources

Computer Science Exit Exam practice collection · Question 92

Question 168 | NS0278

What is the difference between a compiler and an interpreter? Which is Java?

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer: Compilers translate representations; interpreters execute them. Java uses bytecode compilation and JVM interpretation/JIT.

CP04 says javac produces bytecode, then a JVM interprets or compiles it at runtime. A compiler translates rather than necessarily producing a complete native executable, while an interpreter executes a program representation. Java therefore cannot be reduced to exclusively compiled or interpreted.

Primary note proof

Study-note section CP04

> In the conventional C++ toolchain: source → preprocessing → compilation proper → assembly → object files → linking → executable → loading → execution. Preprocessing handles directives such as #include; compilation may emit assembly; the assembler emits machine instructions in an object file. The linker resolves external symbols and combines object files and libraries. The loader places an executable and required components in memory and starts it.

Study-note section CP04

> GCC documents the first four stages explicitly. A compiler need not emit a complete executable or even native machine code. Java javac produces bytecode, which a JVM interprets or compiles at runtime. Interpreters execute a program representation; “one source line at a time” is a beginner approximation. Compilation uses multiple passes, so “whole program in one step” is not literally universal.

Question sources

Computer Programming review questions · Question 1

Question 169 | NS0279

What is the difference between = (assignment) and == (comparison)? Why is this a common mistake?

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer: = assigns a value; == compares values for equality.

CP06 describes assignment as replacing a variable's value; CP10 describes equality as a value test. Confusing them can change state instead of testing it. In C++ a numeric assigned result can also convert to a condition, allowing an unintended assignment to alter control flow.

Primary note proof

Study-note section CP-S05

> In C++, simple assignment changes its left-hand object, and the assignment expression refers to that updated object. When an ordinary numeric assignment is used as a condition, the newly assigned value is converted to truth: zero is false and nonzero is true. Equality testing with == instead produces a Boolean result without assigning either operand. Confusing the two can both change program state and choose an unintended branch.

Study-note section CP10

> Relational and equality operators test values; && and || combine truth values; ! negates. In C++, zero converts to false and a nonzero number to true. Streams display bool as 0 or 1 unless boolalpha is set. With a=3,b=5, cout<<(a==b) prints 0.

Question sources

Computer Programming review questions · Question 2

Question 170 | NS0280

What is the difference between while and do-while loops? When would you use do-while?

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer: while tests first; do-while runs once before testing. Use do-while for a mandatory first menu/input attempt.

CP14 allows a while body to execute zero times because its condition is checked first. A do-while body executes before its first check, making it suitable for a menu or input attempt that must happen at least once.

Primary note proof

Study-note section CP14

> while(condition) checks before each body execution and may run zero times. do {body;} while(condition); runs the body once before checking; it repeats while the condition remains true. Use it when a menu or input attempt must occur at least once.

Question sources

Computer Programming review questions · Question 3

Question 171 | NS0281

What happens if you try to access array index array[5] when the array has 5 elements (indices 0-4)?

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer: Index 5 is out of bounds: Java throws; C++ access has undefined behavior.

Five elements occupy indices 0 through 4, so array[5] is outside the array. CP22 distinguishes Java's bounds exception from C++ undefined behavior; an adjacent-memory value must not be guessed. The question does not specify a language, so both consequences are stated.

Primary note proof

Study-note section CP22

> A built-in C++ array has a fixed number of same-type elements in contiguous storage: int a[4]={10,20,30,40};. Valid indices are 0 through 3. a[4] is outside the array; accessing it has undefined behavior. Java checks bounds and throws ArrayIndexOutOfBoundsException. Do not predict a C++ out-of-range value from nearby memory.

Question sources

Computer Programming review questions · Question 4

Question 172 | NS0282

Why do we use functions? What are the benefits?

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer: Functions decompose named tasks, enable reuse, reduce duplication and simplify understanding/testing.

CP17 defines a function as grouping a named task and explains that reuse reduces duplication while independent functions are easier to understand and test. Decomposition localizes responsibilities and supports maintenance through those clearer units.

Primary note proof

Study-note section CP17

> A function groups a named task. C++ definition: int add(int a,int b){return a+b;}. Call it with add(x,y);, without parameter type declarations at the call site. Parameters are names in the definition; arguments are supplied expressions or values. A void function produces no return value but can have side effects and return; to exit early.

Question sources

Computer Programming review questions · Question 5

Question 173 | NS0283

What is the difference between pass-by-value and pass-by-reference?

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer: C++ value parameters copy; reference parameters alias caller variables. Java passes copied values, including references.

CP18 makes a value parameter a separate initialized object, while int& aliases the caller variable and can change it directly. Passing a pointer copies an address and can change its target without rebinding the caller's pointer. Java copies object references by value.

Primary note proof

Study-note section CP18

> A C++ value parameter is a separate object initialized from the argument. In func(int a,int b){a*=5;b+=5;return a+b;}, calling func(9,3) returns 53 and leaves the caller values 9 and 3 unchanged.

Study-note section CP18

> An int& parameter aliases the caller variable, so swap(int& a,int& b) changes both originals. An int* parameter copies an address; *p=value changes the pointed-to object, but p=anotherAddress only changes the local pointer. An array parameter int a[] is adjusted to int*: elements can change, and the length must be passed separately. Java always passes by value, including a copy of an object reference; it does not pass caller variables by reference.

Question sources

Computer Programming review questions · Question 6

Question 174 | NS0301

What does this C++ fragment print? Assume cout is available.

```
int a=6,b=8;
int x=2,y=4;
int c(x>y ? (a--,x) : (b--,y));
cout<<"a="<<a;
cout<<" b="<<b;
cout<<" c="<<c;
```

A. a=6 b=5 c=4

B. a=6 b=7 c=5

C. a=8 b=6 c=5

D. a=6 b=7 c=4

Hint

Trace the initial state and each executed operation in order, preserving the requested output layout.

Answer D: D. a=6 b=7 c=4

The condition x>y compares 2 with 4 and is false. Only the false branch is evaluated. Its comma expression first decrements b from 8 to 7 and then yields y, which is 4, for initializing c. The true branch does not execute, so a remains 6. With valid stream spelling, comma operators and ordinary string delimiters, the output is a=6 b=7 c=4.

Why each choice is right or wrong

A - Incorrect: b decreases once from 8 to 7, not to 5.

B - Incorrect: The comma expression yields y=4, not 5.

C - Incorrect: The false branch leaves a=6 and produces b=7,c=4.

D - Correct: Only b decrements; the branch yields y=4 and a remains 6.

Primary note proof

Study-note section CP11

> condition ? yes : no evaluates the condition and exactly one alternative. The comma operator evaluates its left expression, then its right, and yields the right value. Do not confuse it with commas separating function arguments.

Study-note section CP09

> Prefix ++x changes x and yields the new value; postfix x++ yields the previous value and also increments x. Separate the value used in the expression from the stored value after it. Starting a=8,b=5, a++ + --b uses 8+4, gives 12, and leaves a=9,b=4. Starting x=1,y=10,z=-3, (x++*y--)+ ++z gives 8 and leaves x=2,y=9,z=-2.

Question sources

Computer Science Exit Exam · 2018 · Question 19

Question 175 | NS0325

Assume that you are computer science expert and you need to develop large programs within short period of time. Finally, you have planned to provide portable program to the concerned body. Which type of programming language is preferable for the above scenario?

A. Low level

B. Machine level

C. High level

D. Assembly

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer C: A high-level language, with portability constraints checked.

CP03 says high-level source is easier to read and often portable after recompilation if platform assumptions are avoided. That fits rapid development of a large program and a portable result better than instruction-set-dependent machine or assembly code. Portability remains conditional on the target environment; the answer is a preference, not an unconditional guarantee.

Why each choice is right or wrong

A - Incorrect: Low-level code retains instruction-set details and does not best match the stated readability/development and portability goals.

B - Incorrect: Machine-level code depends directly on the instruction set, opposing the requested portable-source preference.

C - Correct: High-level source offers the note's readability and conditional portability advantages for the stated scenario.

D - Incorrect: Assembly depends on the instruction set and is not the preferred abstraction for these development goals.

Primary note proof

Study-note section CP03

> Machine and assembly code depend on an instruction set. High-level source is easier to read and often portable after recompilation, provided it avoids platform assumptions. People can write both assembly and high-level programs. Python, Perl and PHP are commonly called scripting languages; C++ is conventionally compiled. These labels describe common use, not an absolute technical prohibition.

Question sources

Computer Science Exit Exam · 2018 · Question 51

Question 176 | NS0346

Which one of the following identifier name is INVALID in C++ programming language?

A. delete

B. for_cpp

C. jan2025

D. Hello

Hint

Apply the naming, type and syntax rules before deciding whether an option can be used as stated.

Answer A: delete is a reserved keyword.

delete is a C++ keyword and cannot be used as an ordinary identifier. for_cpp is not the keyword for, and jan2025 and Hello use letters followed only by permitted letters or digits.

Why each choice is right or wrong

A - Correct: delete is a reserved C++ keyword and therefore invalid as an ordinary identifier.

B - Incorrect: for_cpp is a distinct permitted name containing letters and underscore, rather than the keyword for.

C - Incorrect: jan2025 begins with a letter and contains only permitted letters and trailing digits.

D - Incorrect: Hello contains permitted letters and is not a reserved keyword identified by the question.

Primary note proof

Study-note section CP05

> In the basic ASCII identifier rules, start with a letter or underscore; continue with letters, digits or underscores. Do not use a keyword. C++ is case sensitive: exitExam and ExitExam differ; delete and float are keywords, but FLOAT is a legal identifier. variable_1234 is valid; 4thInput, T-3 and names containing spaces are not single identifiers. Avoid reserved underscore patterns in application code.

Question sources

Computer Science Exit Exam · 2018 · Question 79

Question 177 | NS0358

Which of the following statement about function overloading is TRUE?

A. Overload function have same name but different parameters/type

B. Overload function is not supported

C. Overload function must have different return type

D. Overload function must have the same number of parameters

Hint

Use the cited language rules to separate the purpose of each construct from its implementation details.

Answer A: Overloads share a name and differ in their parameter lists.

CP19 requires a shared name with differences in parameter count or types/order. A states that relationship. Return type alone does not distinguish overloads, and neither different return types nor equal parameter counts are mandatory. Thus the other statements either deny supported overloading or impose restrictions contradicted here.

Why each choice is right or wrong

A - Correct: This states the shared-name and differing-parameter-list relationship required by CP19.

B - Incorrect: CP19 explicitly describes supported C++ function overloading, contradicting this blanket denial.

C - Incorrect: Different return types are not required and return type alone cannot distinguish overloads.

D - Incorrect: Parameter count may differ between overloads, so equal counts are not mandatory.

Primary note proof

Study-note section CP19

> Overloads share a name but differ in parameter count or types/order. Return type alone cannot distinguish them. Choose a candidate using argument types, then required conversions; report ambiguity if there is no unique best match. int sum(int x,int y,int z=0) allows two or three arguments; defaults occupy trailing positions in ordinary declarations.

Question sources

Computer Science Exit Exam · 2018 · Question 96