CS Exit Ethiopia

Revision Outlines

Computer Science Exit Examination

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

Computer Programming

Revision Outlines

Problem-to-program workflow | Exam focus

- Problem-to-program workflow
  - Inputs, outputs, legal ranges, edge cases
  - Manual small-case solution
  - Algorithm → pseudocode or flowchart
  - Implement → test → maintain
  - Largest-of-three: best=first input
  - Compare second, then third
  - Zero initialization fails all-negative inputs
- Programming responsibilities
  - Writing, testing, debugging, maintaining
  - Actual versus required result
  - Execution alone insufficient

Algorithms and paradigms | Exam focus

- Algorithm properties
  - Precise executable steps
  - Termination for every legal input
  - Definiteness: unambiguous meaning
  - Effectiveness: performable steps
  - Optional inputs, specified result
  - Correctness against specification
  - Efficiency: desirable, measurable
  - Inefficient terminating algorithms still qualify
  - Module's desirable-efficiency listing
- Programming paradigms
  - Procedural: procedures organize operations
  - Structured: sequence, selection, iteration
  - Object oriented: state and behavior

- Functional: function evaluation
- Conditional: construct, not paradigm

Language generations and portability

- Language generations
  - First: machine instructions
  - Second: symbolic assembly
  - Third: C++, Java, general high-level
  - Fourth: SQL, queries, application generators
  - Fifth: loose historical category
  - Declarative logic, constraints, AI/expert systems
  - Generation independent of interpretation
- Language portability and conventions
  - Machine/assembly: instruction-set dependence
  - Readable high-level source
  - Recompile without platform assumptions
  - Humans write assembly and high-level
  - Python, Perl, PHP: conventional scripting
  - C++: conventionally compiled
  - Usage labels, not technical prohibitions

Compilation, linking and loading | Exam focus

- Conventional C++ toolchain
  - Source → preprocessing → compilation proper
  - Assembly → object files → linking
  - Executable → loading → execution
  - #include: preprocessing directive
  - Compiler may emit assembly
  - Assembler: object-file machine instructions
  - Linker: external-symbol resolution
  - Object files and libraries combined
  - Loader: memory placement and startup
- Translation caveats

- GCC: first four stages
- Compiler output needn't be executable/native
- Java javac → bytecode
- JVM: interpretation or runtime compilation
- Interpreters execute program representations
- Line-by-line: beginner approximation
- Multiple compilation passes
- Whole-program one-step claim overbroad

Identifiers, keywords and comments

• ASCII identifier rules
- Initial letter or underscore
- Subsequent letters, digits, underscores
- Keywords forbidden
- Case sensitivity: exitExam ≠ ExitExam
- delete, float: keywords
- FLOAT, variable_1234: valid
- 4thInput, T-3, spaces: invalid identifiers
- Reserved underscore patterns avoided

• Comments and entry point
- // line comments
- /* ... */ block comments
- Intent, assumptions, maintenance
- Comments don't execute
- No automatic speed/reliability/portability improvement
- main: entry function, not keyword

Declarations, constants and types | Exam focus

• Declaration versus initialization
- type name;
- type name = value;
- int age = 20;
- Declaration: name and type

- Initialization: initial value
- Later assignment replaces value
- Scalar a=b copies value
- Later b changes leave a unchanged
- Pointer assignment copies address
- Java object-reference assignment copies reference

• Types, constants and aliases

- int: whole numbers
- float/double: fractional approximations
- char: character
- bool: logical values
- Implementation-dependent C++ sizes
- Java widths don't transfer
- short n ≡ short int n
- int short n: same signed short
- const int ≡ int const
- Ordinary const int requires initialization
- typedef: alias, not distinct type

Literals, conversions and sizeof

• Literals and storage sizes

- 'A': character literal
- "A": string literal
- Terminator storage beyond visible characters
- sizeof: bytes
- sizeof(char)==1
- Byte width and sizes implementation-dependent
- sizeof(array): total before pointer conversion

• Conversion timing

- int x=5,y=2: x/y=2
- Double assignment after division: 2.0
- static_cast<double>(x)/y=2.5
- Cast before fractional operation

- In-range float→int: toward-zero truncation

Arithmetic and compound updates

- Arithmetic grouping and validity
  - *, /, % before +, -
  - Equal-precedence arithmetic: left grouping
  - Integer division: toward-zero truncation
  - a == (a/b)*b + a%b
  - Identity requires valid division
  - %: integral operands only
  - (8.0*3)%5: invalid
  - Division by zero: no ordinary output
  - Signed overflow: no ordinary output
- Compound assignment traces
  - +=, -=, *=, /=, %=
  - Operation then conversion to x's type
  - Simple x op= y: x=x op y
  - p=q=n=8: p+=n → 16
  - p/=q → 2
  - x=10: +=5, /=2 → 7
  - x=x+x doubles, not generally increments

Increment operators and sequencing

- Prefix versus postfix
  - Expression value versus stored value
  - ++x: increment, yield new
  - x++: yield old, increment
  - a=8,b=5: a++ + --b=12
  - Final a=9,b=4
  - x=1,y=10,z=-3: (x++*y--)+ ++z=8
  - Final x=2,y=9,z=-2
- Grouping versus evaluation order
  - Repeated unsequenced modifications unsafe

- C++17 operator-form stream sequencing
- cout<<x++<<" "<<++x: 5 7
- Before C++17: undefined behavior
- Bank options omit correct C++17 output

Boolean, bitwise and precedence rules

• Truth-value operations
- Relational and equality tests
- && conjunction, || disjunction
- ! negation
- C++ zero=false, nonzero=true
- Default bool output: 0/1
- boolalpha changes Boolean formatting
- a=3,b=5: (a==b) prints 0

• Short circuit versus bitwise
- &: bits, no short circuit
- && right only after true left
- || right only after false left
- false && dangerousCall(): skipped
- x=1,y=0,z=4: (x&y)||++z
- Final z=5, Boolean result=1
- Grouping/postfix → unary → multiplicative
- Additive → shifts → relations → equality
- Bitwise & → ^ → |
- && → || → conditional
- Assignment → comma

Conditional and comma expressions

• Conditional and comma operators
- condition ? yes : no
- Condition then exactly one alternative
- Comma: left then right
- Comma result: right operand value

- Argument separators differ from comma operator

• Conditional-comma trace

- a=6,b=8,x=2,y=4
- x>y false: (b--,y)
- Final b=7,c=4,a=6
- Unselected alternative unexecuted
- Internal semicolon: invalid syntax
- Comma restoration requires labeled interpretation

Decisions and flowcharts | Exam focus

• Decision grouping

- Independent consecutive if statements
- else: nearest unmatched if
- else-if: first true branch
- Nested S2: c1 && !c2
- Braces determine grouping

• Decisions and flowchart evidence

- ph=15.3: warning plus Alkaline
- Separate range check and classification
- else/early return creates exclusivity
- Diamond: decision
- Rectangle: processing
- Parallelogram: input/output
- Terminator: start/end
- Pre-test: check before body
- Post-test: check after body
- Missing arrows/condition placement: indeterminate loop

Switch and fall-through

• Traditional C++ switch

- Integral or enumeration selector
- case: constant expressions
- Matching label starts execution

- Fall-through until break/return/end
- default: unmatched value
- default needn't be last

• Switch fall-through trace

- Input 2: Bus.Truck.
- Case 2 enters, case 3 continues
- break: nearest switch or loop
- break doesn't terminate whole program
- case label: entry point
- No automatic branch isolation

Loop mechanics | Exam focus

• Pre-test versus post-test loops

- while: condition before each body
- while: possibly zero executions
- do-while: body before condition
- do-while: at least once
- Repeat while condition true
- Menu/input attempt use case

• for execution order

- Initialization once
- Condition → body → update → condition
- for(;;): absent condition=true
- Known-count convenience, condition-control flexibility
- Indices 0..n-1: i=0,i<n,++i
- i=0,i<5,i+=2: 0 2 4

Control transfers

• Control-transfer scope

- break: nearest loop or switch
- continue: skip current iteration remainder
- for continue → update
- while continue → condition

- return: leave current function
- Optional return result
- main return: normal program end

• goto label tracing

- Transfers within same function
- Follow reached labels
- a → b → end
- Graduating fourth
- Year unreachable from entry a
- Structured alternatives ease reasoning

Iteration counts and nontermination

• Loop state tables

- Condition inputs, body output, updated values
- i=0,j=8: i<j, i++,j--
- Entries (0,8),(1,7),(2,6),(3,5)
- Odd-i continue: two prints
- Outer i=1,i<5,i+=5: once
- Inner j=10,j>i,--j: nine prints

• Progress and early exit

- n=1,n<=18,n+=2
- Break before multiples of seven
- Output: 1 3 5
- Fixed a=6>b=5: infinite while
- data[0]=5: %10 guard false
- Conditional-only increment: no progress
- No output, infinite loop
- <=10→<10 alone insufficient

Functions and reusable modules | Exam focus

• Functions and call syntax

- Named task grouping
- int add(int a,int b){return a+b;}

- add(x,y): no call-site parameter types
- Parameters: definition names
- Arguments: supplied expressions/values
- void: no return value
- Side effects still possible
- return; permits early void exit

• Reusable organization

- Method: class/object-associated function
- Methods can return values
- Modules: responsibilities behind interfaces
- Reuse reduces duplication
- Independent functions: understandable, testable
- Libraries: reusable components
- Frameworks: organizing structure
- Frameworks often invoke callbacks

Value, reference and pointer parameters | Exam focus

• Pass-by-value isolation

- Separate parameter object from argument
- func: a*=5,b+=5,return a+b
- func(9,3)=53
- Caller values remain 9,3

• Aliasing and copied addresses

- int& aliases caller variable
- Reference swap changes originals
- int*: copied address
- *p=value mutates pointed object
- p=anotherAddress changes local pointer
- int a[] parameter adjusts to int*
- Element mutation possible
- Array length passed separately
- Java always passes by value
- Object reference copied, caller variable unaliased

Overloads, defaults and inline

- Overload resolution and defaults
  - Same name, different parameter count/types/order
  - Return type alone insufficient
  - Argument types then required conversions
  - No unique best match: ambiguity
  - sum(x,y,z=0): two/three arguments
  - Ordinary defaults occupy trailing positions
- inline meaning
  - No compulsory call-site expansion
  - Small frequent functions: optimization candidates
  - Optimizer decides expansion
  - Linkage/definition rules also matter
  - Not simply a speed command

Scope, lifetime and static locals

- Scope versus lifetime
  - Scope: name usability
  - Lifetime: object existence
  - Local name: block restriction
  - Inner names may shadow outer
  - Global availability requires visible declaration
  - ::name selects hidden global
- Automatic versus static local
  - Automatic: fresh on entry
  - Static: initialized once
  - Static value persists, name stays local
  - Even +1, odd +2, static x=0
  - Five calls: 1,3,5,7,9
  - Sum=25
  - Namespace/static integers: zero-initialized
  - Uninitialized local int: no assumed zero

Recursion and call tracing | Exam focus

- Recursive termination
  - Direct or indirect self-call
  - Base case plus progress
  - Every supported input must reach base
  - factorial(0)=1
  - factorial(n)=n*factorial(n-1), n>0
  - factorial(3)=3*2*1*1=6
  - Negative inputs: reject/separately define
  - n-1 doesn't reach zero from negative
- Recursive call tracing
  - Conditions before expansion
  - if(n<m) return 0;
  - f(1,4): immediate zero
  - Unreachable recursion contributes nothing
  - Record local parameters and pending operations
  - Unwind return values
  - Smaller instances commonly used
  - Base branch alone doesn't prove reachability

Arrays and bounds

- Built-in array model
  - Fixed count, same type
  - Contiguous storage
  - int a[4]={10,20,30,40};
  - Valid indices: 0..3
  - a[4]: undefined C++ access
  - No predicted nearby-memory output
  - Java: ArrayIndexOutOfBoundsException
- Array declarations and initialization
  - Standard bound: compile-time constant
  - Variable-length locals: compiler extension
  - int a[5]={1,2,3}: remaining zeros

- Uninitialized automatic int array differs
- a[i] ≡ *(a+i)
- Brackets not required for element access

Array algorithms and matrix indexing | Exam focus

- Array traversal algorithms
  - Sum: total=0, add each once
  - Maximum/minimum: initialize from a[0]
  - First verify nonempty array
  - Fifty inputs: count-controlled loop
  - Positive-only sum: if(x>0) total+=x
  - Skip nonpositive: continue for x<=0
  - Counting versus filtering responsibilities
- Two-dimensional indexing
  - int a[R][C]: R rows, C elements
  - Zero-based row then column
  - Row-major offset: r*C+c
  - Bank 4×4: a[1][1]=6
  - a[2][3]=6, a[3][3]=2
  - Java length belongs to each array
  - arr.length-2 endpoint: incomplete traversal

C strings and std::string

- C-string termination and storage
  - Character array ending '\0'
  - "Hello": six char positions
  - strlen: five visible characters
  - Terminator excluded from strlen
  - sizeof fixed array: allocated bytes
  - Capacity may exceed strlen+1
  - cout<<a: terminated sequence, not address
- Text APIs and capacity
  - <cstring>: strcpy(destination,source)

- strcat(destination,source), strlen, strcmp
- Sufficient destination capacity
- strcmp equality: zero
- Otherwise sign, not necessarily ±1
- <string>: owning resizable std::string
- +: concatenation, ==: content
- size()/length(): stored-character count
- substr(start,count): segment
- No std::string toUpperCase()/trim() members

Pointers and dereferencing

• Pointer identity versus pointed value
  - int x=10, int* p=&x
  - *p: read/write x
  - &p: pointer variable's address
  - p=&a,q=&b: p=q redirects
  - No change to a or b
  - *p=*q: pointed-value copy
  - Uninitialized/null/dangling dereference forbidden
• Arrays and pointer levels
  - Common array-to-first-element pointer conversion
  - Array isn't assignable pointer variable
  - int** pp=&p
  - **pp reaches x
  - void*: holds object address
  - Convert appropriately before C++ dereference

Pointer arithmetic and allocation

• Scaled pointer arithmetic
  - p+k: k elements
  - k*sizeof(*p) byte displacement
  - Assume sizeof(float)==4
  - Address 7000: p+3 → 7012

- cout<<p++: old address, advance
- cout<<*p++: old element, advance
- ++*p: change element
- p+q: invalid
- Same-array subtraction: element distance

• Allocation and pointer boundaries
- Within-array or one-past arithmetic
- One-past dereference forbidden
- new T ↔ delete
- new T[n] ↔ delete[]
- Prefer resource-owning standard containers
- Primitive initialization must be requested

Structures and member access

• Structures and member access
- Named members, possibly differing types
- House definition: price and rooms
- House h{150000,3}: object creation
- h.rooms: dot access
- p->rooms ≡ (*p).rooms
- Pointer must reference live object
- Type definition alone creates no instance

• Aggregate and recursive structure rules
- Initialization in declaration order
- Default member initializer: int rooms=3
- C++11 list initialization rejects narrowing
- Exactly representable floating constant exception
- 25.00→float not necessarily narrowing
- 25.0f clarity, language-version check
- Structures copied, passed, returned
- No full self-instance member
- Self-type pointer allowed

Console streams and input state

- Console streams and spelling
  - <iostream>, std::cin/std::cout
  - Explicit qualification or introduced names
  - cout <<: output insertion
  - cin >>: formatted input extraction
  - std::endl: newline plus flush
  - Cout, Int: incorrect case
  - std: :cout: invalid printed spelling
- Input state and output formatting
  - getline(cin,text): full line
  - Numeric extraction leaves pending newline
  - Consume newline before line read
  - Check input failure before value use
  - Exact double 4 may print 4
  - Text, spaces, newlines affect traces

File streams and safe reading

- File stream types and modes
  - <fstream>: file-stream header
  - ofstream writes, ifstream reads
  - fstream: both directions
  - Default ofstream argument: ios::out
  - Ordinary effect: create/truncate
  - Signature versus effective out|trunc
  - ios::app: every write at end
  - ios::ate: initial seek to end
  - ios::binary: binary mode
- Safe file-reading lifecycle
  - if(!file): opening failure
  - while(file>>record)
  - while(getline(file,line))
  - EOF discovered by attempted read

- while(!file.eof()): possible invalid extra record
- Close or rely on destruction
- ofstream class versus <fstream> header

Math functions and iterative powers | Exam focus

- Mathematical functions
  - <cmath>: sqrt, pow, floor, ceil
  - <cstdlib>: integer abs also
  - sqrt(16)=4, pow(3,4)=81, abs(-7)=7
  - ceil(x): least integer-valued result ≥x
  - floor(x): greatest integer-valued result ≤x
  - ceil(-2.7)=-2, floor(-2.7)=-3
  - Negative truncation differs from floor
- Iterative exponentiation
  - No built-in C++ exponentiation operator
  - ^: bitwise XOR
  - **: not power operator
  - Nonnegative integer e: r=1
  - Repeat r*=b exactly e times
  - 4^3 → 64
  - Powers/factorials: fixed-width overflow risk
  - Specify valid input range

Proofs and loop invariants | Exam focus

- Proof methods
  - Direct P⇒Q: assume P, derive Q
  - Contrapositive: ¬Q⇒¬P
  - Contradiction: false claim assumption → impossibility
  - Counterexample refutes universal claim
  - Induction: base case
  - P(k) assumption → P(k+1)
- Loop-invariant summation proof
  - Initial sum=0,i=1

- Guard: i<=n
- Invariant: sum=(i-1)*i/2
- Initialization establishes invariant
- Add i → i*(i+1)/2
- Increment i restores invariant
- Exit: i=n+1
- Result: sum=n*(n+1)/2
- Termination measure: n-i+1
- Nonnegative, decreasing while guard holds
- Assumptions: n>=0, no overflow

Debugging, exceptions and maintenance | Exam focus

- Error categories and debugging
  - Syntax: grammar violation
  - Type: semantic rejection despite valid syntax
  - Runtime: during execution
  - Logic: wrong behavior
  - Bug: defect
  - Debugging: locate and correct
  - First useful diagnostic, nearby tokens/declarations
  - Repair and rebuild
  - Debugger requires runnable program
- Testing and maintenance tools
  - Ordinary, boundary, invalid inputs
  - First actual/expected state divergence
  - Exceptions: matching-handler control transfer
  - No arbitrary undefined-behavior recovery
  - Refactoring: structure improved, behavior preserved
  - Version control: records, comparisons, rollback
  - Documentation: use, design, maintenance assumptions

Static typing and Java distinctions

- Static versus dynamic typing

- Static: types checked before execution
- C++ auto: inferred static type
- Dynamic: operations checked using runtime values
- No mandatory explicit static declarations
- Static typing allows type inference

• Java versus C++ APIs
- Java boolean, fixed-width numeric primitives
- Object references
- Java String: immutable
- substring(begin,end): exclusive end
- equals: content, ==: references
- Java arrays: length field
- Scanner nextInt() → pending nextLine() newline
- C++ std::string: mutable
- substr(start,count): different API

Malformed-code interpretation

• Evidence before output prediction
- Complete code required
- Diagram edges, conditions, labels required
- Post-test loop
- Entry body before decision
- Branch returns to body
- Post-test do-while: answer B
- Literal invalid versus labeled intended trace
- Documented OCR correction only

• Malformed-code reference cases
- Pointer members: p->m, (*p).m
- Cout/Cin ≠ cout/cin
- C1 ≠ c1, car_mode ≠ car_model
- goto skips unreached output statements
- Assignment trace: z=5, a=1
- Intended z=5 then a=1

- Case matters: Z ≠ z
- Negative-value test: x < 0
- C++ entry point: int main()
- C++ spelling: int and cout
- Quotes and comma punctuation repair
- No invented unique literal output

Interpret programming questions precisely

• Ambiguous question options
- Linking versus loading
- Machine instructions versus executable files
- Data members and initialization
- Equivalent short declarations
- Language version and narrowing conversions
- Equivalent pointer-member expressions

• Incorrect-key and wording traps
- Static typing versus explicit annotations
- Scalar declaration: int x;
- Void function definitions
- int myFunction(){}: may compile
- Reaching int function end: undefined behavior
- break exits the innermost loop
- Const spelling and required initialization
- SS167: counting versus skipping
- Loop exit versus iteration skip
- Inspect, rebuild, and retest
- Rebuild and retest complete repair