

VU Learning HUB

CS201 Quiz File

Chapter-wise Solved MCQs

vulearninghub.xyz

Lecture 1: Introduction to Programming

Q1. How does the CS201 course handout characterize the fundamental nature of computers?

- A) Computers possess human-like reasoning and intuition.
- B) Computers are extraordinarily smart and self-thinking entities.
- C) Computers are incredibly stupid and do exactly what they are explicitly instructed to do.
- D) Computers can interpret ambiguous user requests without explicit code.

Q2. Which of the following is NOT one of the steps in the basic recipe for designing a program properly?

- A) Analyze a problem statement expressed as a word problem.
- B) Formulate statements and comments in a precise language.
- C) Pay attention to detail and evaluate activities through checks and tests.
- D) Write raw executable code immediately without analyzing the problem statement.

Q3. According to Matthias Felleisen's quote in the handout, why is learning programming important for everyone?

- A) Traditional forms of programming are useful for all citizens equally.
- B) It teaches problem-analysis and problem-solving skills without excessive overhead.
- C) It eliminates the need for application software and spreadsheet tools.
- D) It forces non-programmers to master complex low-level machine notations.

Q4. What critical habit is emphasized as essential for good programming practice in Lecture 1?

- A) Avoiding comments to keep code compact.
- B) Paying attention to detail and commenting code.
- C) Writing code directly without testing or checks.
- D) Using vague variable names to speed up typing.

Answer Key

1. C — Computers lack independent thought and follow explicit instructions literally without doing more or less.
2. D — Proper program design requires problem analysis before writing code, rather than starting immediately with raw code.
3. B — Programming as problem-solving teaches analysis skills with minimal overhead, benefiting everyone from secretaries to tech professionals.
4. B — Paying attention to detail and documenting code with comments are highlighted as fundamental principles of software development.

Lecture 2: Software Categories & Development Environment

Q1. Software is broadly categorized into which two primary divisions?

- A) Hardware Software and Firmware Software
- B) System Software and Application Software
- C) Compiled Software and Interpreted Software
- D) Utility Software and Driver Software

Q2. Which component of the C development environment processes directives like `#include` before compilation begins?

- A) Linker
- B) Loader
- C) Preprocessor
- D) Compiler

Q3. What is the role of the Linker in the C execution process?

- A) It places the executable program into primary memory.
- B) It links object code created by the compiler with system libraries.
- C) It translates high-level source code directly into assembly language.
- D) It reads characters typed by the user from the keyboard.

Q4. Which module loads the executable program from disk into primary memory for CPU execution?

- A) Compiler
- B) Preprocessor
- C) Loader
- D) Text Editor

Answer Key

1. B — Software is classified into System Software (operating systems, compilers) and Application Software (word processors, games).
2. C — The preprocessor scans and processes preprocessor directives (beginning with `#`) before passing code to the compiler.
3. B — The linker combines the compiled object file with library routines to form a single executable file.
4. C — The loader transfers the executable program from secondary storage (disk) into RAM for the CPU to execute.

Lecture 3: First C Program & Basic Building Blocks

Q1. In a C/C++ program, why is the function name `main()` special?

- A) It is the only function that can take input arguments.
- B) It is automatically called first when the program is executed.
- C) It is required to return a floating-point value.
- D) It is a preprocessor directive built into `iostream.h`.

Q2. What does the `#include` directive tell the compiler to do?

- A) Declare all user variables automatically as integers.
- B) Include the contents of the standard input/output stream header file.
- C) Clear the screen before executing the `main()` function.
- D) Link the program directly to the operating system loader.

Q3. What is the syntax error called if a programmer forgets the ending semicolon `;` in a C statement?

- A) Logical error
- B) Linker error
- C) Syntax error
- D) Runtime exception

Q4. How are memory locations represented in programs to make them easily accessible to humans?

- A) By absolute hexadecimal addresses directly hardcoded in statements.
- B) By variable names assigned to represent specific memory locations.
- C) By register numbers assigned by the CPU at runtime.
- D) By line numbers assigned by the text editor.

Answer Key

1. B — Execution of every C/C++ program begins at the `main()` function.
2. B — `#include` instructs the compiler to insert the standard stream I/O declarations file.
3. C — Statements in C end with a semicolon; omitting it violates grammar rules and causes a compilation syntax error.
4. B — Variables provide human-readable names for memory locations instead of raw numerical addresses.
-

Lecture 4: Data Types, Variables & Arithmetic Operators**Q1. Which C++ operator is used to obtain the remainder of an integer division?**

- A) `/`
- B) `%`
- C) `*`
- D) `^`

Q2. In the arithmetic expression `TotalAge / 10`, if both `TotalAge` and `10` are integers, what type of division is performed?

- A) Floating-point division preserving exact decimal places.
- B) Integer division where any fractional part is truncated.
- C) Modulus division returning only the remainder.
- D) Bitwise division shifting bits to the right.

Q3. What is the operator precedence evaluation order for arithmetic operators of equal precedence in an expression without parentheses?

- A) Right to left
- B) Left to right
- C) Random order determined by compiler
- D) Outside to inside

Q4. Why is using descriptive variable names strongly recommended in programming?

- A) It reduces the size of the compiled executable file.
- B) It improves code readability, self-documentation, and maintainability.
- C) It speeds up CPU execution time.
- D) It is required by the preprocessor directive.

Answer Key

1. B — The modulus operator `%` yields the remainder when one integer is divided by another.
2. B — Division between two integers truncates any fractional part and yields an integer quotient.
3. B — Operators with equal precedence (like `+` and `-`) evaluate from left to right due to left-associativity.

4. B — Descriptive variable names clarify the intent of the variable, making code easier to understand and maintain.

Lecture 5: Conditional Statements & Flowcharting

Q1. Which C statement is the fundamental structure used for decision making based on conditions?

- A) ``while`` statement
- B) ``if`` statement
- C) ``for`` statement
- D) ``switch`` statement

Q2. In C/C++, which logical operator represents the logical AND operation?

- A) ``||``
- B) ``&&``
- C) ``!``
- D) ``&``

Q3. In an ``if/else`` construct, when is the ``else`` block executed?

- A) Only when the condition in ``if`` evaluates to true.
- B) Only when the condition in ``if`` evaluates to false.
- C) Always, right after the ``if`` block executes.
- D) Whenever a syntax error is detected in the program.

Q4. What shape is conventionally used in flowcharts to represent a decision point?

- A) Rectangle
- B) Oval
- C) Diamond
- D) Parallelogram

Answer Key

1. B — The ``if`` statement is the primary conditional construct for executing code conditionally.
2. B — ``&&`` is the logical AND operator, returning true only if both operands are true.
3. B — The ``else`` clause executes if and only if the ``if`` condition tests false.
4. C — A diamond symbol denotes decision branches in standard program flowcharts.

Lecture 6: Repetition Structures - While Loop

Q1. What type of repetition construct is the ``while`` loop in C?

- A) Post-test loop that executes at least once regardless of condition.
- B) Pre-test loop that tests the logical condition before executing the loop body.
- C) Infinite-only loop requiring a ``break`` statement to exit.
- D) Counter-driven loop requiring an explicit step parameter in headers.

Q2. How many times will the body of a ``while (condition)`` loop execute if the condition is false initially?

- A) Exactly 1 time
- B) Zero times
- C) Indefinitely until an error occurs
- D) Depend on compiler optimization settings

Q3. What happens if a `while` loop condition never becomes false during program execution?

- A) The compiler generates a syntax error.
- B) The loop becomes an infinite loop and runs indefinitely.
- C) The program terminates cleanly with exit code 0.
- D) The variable causes an arithmetic overflow and exits.

Q4. What occurs when a variable attempts to store a value greater than the maximum limit allowed by its data type?

- A) Syntax error
- B) Arithmetic overflow
- C) Preprocessor warning
- D) Memory leak

Answer Key

1. B — A `while` loop evaluates its condition before each iteration, including the first.
2. B — If the condition evaluates to false at the start, the loop body is bypassed entirely (0 iterations).
3. B — If the termination condition is never reached, execution remains trapped in an infinite loop.
4. B — Storing a value exceeding the type's storage capacity leads to an arithmetic overflow condition.

Lecture 7: Do-While, For Loops & Increment Operators

Q1. What is the fundamental difference between a `while` loop and a `do-while` loop?

- A) A `do-while` loop evaluates its condition at the end, guaranteeing at least one execution of the loop body.
- B) A `while` loop cannot use logical operators in its condition.
- C) A `do-while` loop cannot be nested inside another loop.
- D) A `while` loop requires double curly braces.

Q2. What are the three expressions separated by semicolons in a standard `for` loop header `for (expr1; expr2; expr3)`?

- A) Condition, Increment, Initialization
- B) Initialization, Condition, Increment
- C) Declaration, Allocation, Termination
- D) Input, Process, Output

Q3. What is the effect of the shorthand assignment operator `counter += 5;`?

- A) It sets `counter` equal to 5.
- B) It adds 5 to `counter` and assigns the new result back to `counter`.
- C) It multiplies `counter` by 5.
- D) It compares `counter` with 5 for equality.

Q4. If `x` has an initial value of 5, what is the value of `x` after evaluating `x++`?

- A) 4

- B) 5 in the immediate expression, but `x` becomes 6 afterwards.
- C) 6 immediately before expression evaluation.
- D) 10

Answer Key

1. A — `do-while` tests condition after the body executes, guaranteeing at least 1 iteration.
2. B — The `for` loop header contains initialization, loop continuation condition, and increment/decrement step.
3. B — `counter += 5` is equivalent to the expression `counter = counter + 5`.
4. B — Post-increment `x++` uses current value in expression and increments `x` to 6 after evaluation.

Lecture 8: Switch Statement & Structured Programming

Q1. Which keyword is used in a `switch` construct to exit the switch structure after executing a matching `case`?

- A) `continue`
- B) `exit`
- C) `break`
- D) `return`

Q2. What happens in a `switch` statement if a matching `case` does NOT contain a `break` statement at its end?

- A) The program halts immediately with a runtime error.
- B) Execution falls through into the subsequent `case` statements automatically.
- C) The default case is executed next regardless of value.
- D) The compiler throws a syntax error during compilation.

Q3. Which `switch` clause is executed when the evaluated expression does not match any specified `case` constant?

- A) `else`
- B) `otherwise`
- C) `default`
- D) `catch`

Q4. What is a primary rule of structured programming and flowcharting regarding control constructs?

- A) Each module or construct should have multiple entry points and multiple exit points.
- B) Every loop must contain a `goto` statement.
- C) Each module or construct should have a single entry point and a single exit point.
- D) Functions should avoid taking input parameters.

Answer Key

1. C — `break` causes immediate termination and exit from the `switch` block.
2. B — Omitting `break` leads to fall-through behavior, executing code of subsequent cases.
3. C — `default` block handles any value that matches no explicit `case` constant.
4. C — Structured programming mandates single-entry, single-exit structure for clean execution flow.

Lecture 9: Functions & Top-Down Design

Q1. What design principle breaks a large, complex problem down into smaller, manageable sub-tasks?

- A) Bottom-up synthesis
- B) Top-down design ("Divide and Conquer")
- C) Object inheritance
- D) Linear compilation

Q2. What component informs the compiler about a function's name, return type, and parameter list before its actual definition?

- A) Function body
- B) Function prototype (declaration)
- C) Preprocessor macro
- D) Function invocation

Q3. If a function is declared as `void calculateTotal(int x, float y)`, what does the `void` return type indicate?

- A) The function returns an integer 0.
- B) The function does not return any value to the calling program.
- C) The function parameters are optional.
- D) The function can only be called from inside a `while` loop.

Q4. Why are functions essential in programming?

- A) They eliminate the need for memory allocation.
- B) They promote code reusability, modularity, and easier debugging.
- C) They force all variables to become global.
- D) They execute faster than inline code statements.

Answer Key

1. B — Top-down methodology divides a major task into smaller sub-tasks until each is easily manageable.
2. B — A function prototype declares the function interface (return type, name, parameters) to the compiler.
3. B — `void` indicates that the function performs actions but yields no return value to the caller.
4. B — Functions group logic into reusable modules, improving organization, maintainability, and code reuse.

Lecture 10: Scope Rules, Call Mechanisms & Recursion

Q1. What is the scope of a variable declared inside a block surrounded by curly braces `{}`?

- A) Global scope across all files.
- B) Block-level scope, accessible only within those braces.
- C) Function-level scope accessible everywhere in `main()`.
- D) File-level scope accessible to all header files.

Q2. Under the default parameter passing mechanism "Call by Value", what happens when a function modifies a parameter?

- A) The original variable in the calling program is modified directly.
- B) A local copy on the stack is modified, leaving the caller's original variable unchanged.

- C) A syntax error occurs during compilation.
- D) The variable is automatically converted into a pointer.

Q3. What two essential components must every recursive function contain?

- A) A ``switch`` statement and a ``default`` case.
- B) A base case (termination condition) and a recursive call step.
- C) A global variable and a static variable.
- D) A pointer declaration and a ``malloc`` call.

Q4. What major potential overhead is associated with excessive or deep recursion?

- A) Preprocessor slowdown
- B) Memory stack overhead and risk of stack overflow
- C) Compilation syntax errors
- D) Disk fragmentation

Answer Key

1. B — Block scope limits variable visibility to the enclosing set of braces ``{}``.
2. B — Call by value passes a copy of the argument; modifications inside the function do not affect original caller variables.
3. B — Recursion requires a base case to halt recursion and a recursive step moving towards the base case.
4. B — Each recursive call adds a frame to the system call stack, incurring memory and function-call overhead.

Lecture 11: One-Dimensional Arrays & Linear Search

Q1. What is an array in C/C++?

- A) A collection of variables of different data types stored in random memory.
- B) A data structure containing a fixed-size sequential collection of elements of identical data type stored in contiguous memory.
- C) A dynamic list that automatically expands as elements are added.
- D) A set of functions grouped under a single name.

Q2. If an array is declared as ``int scores[10];``, what are the valid index bounds for accessing its elements?

- A) 1 to 10
- B) 0 to 9
- C) 0 to 10
- D) -1 to 9

Q3. In memory, how much space does an integer array ``int age[10];`` occupy assuming an ``int`` requires 4 bytes?

- A) 10 bytes
- B) 40 bytes
- C) 100 bytes
- D) 4 bytes

Q4. How does a Linear Search algorithm locate a target value in an array?

- A) It calculates the middle element and divides the array in half repeatedly.
- B) It compares the target sequential element-by-element from index 0 until found or end reached.

- C) It sorts the array first using Bubble Sort before searching.
- D) It uses pointer subtraction to leap directly to the index.

Answer Key

1. B — An array is a contiguous memory block holding homogeneous data elements.
2. B — C arrays use zero-based indexing; a 10-element array has indices `0` through `9`.
3. B — Total memory size equals element count times element size: $10 * 4 = 40$ bytes.
4. B — Linear search scans sequentially through elements starting from index 0 until the item is found.

Lecture 12: Character Arrays & Multidimensional Arrays

Q1. How are string literals stored in C character arrays properly terminated?

- A) With a trailing space ` `
- B) With a null character `'\0'`
- C) With a semicolon `';'`
- D) With an exclamation mark `'!'`

Q2. In C/C++, how are elements of a two-dimensional array `int matrix[3][4];` conceptually laid out in memory?

- A) In column-major order
- B) In row-major order as a single contiguous sequence of memory locations
- C) In random memory blocks linked by pointers
- D) In reverse diagonal order

Q3. When passing a multidimensional array to a function, which dimension must be explicitly specified in the parameter list?

- A) None of the dimensions
- B) Only the first (leftmost) dimension
- C) All dimensions except the leftmost (first) dimension
- D) Every dimension must be omitted

Q4. How is an individual element at row 2, column 3 accessed in a 2D array `int A[5][5]`?

- A) `A[2, 3]`
- B) `A[2][3]`
- C) `A(2)(3)`
- D) `A[2 : 3]`

Answer Key

1. B — C-style strings are null-terminated character arrays ending with `'\0'` (ASCII code 0).
2. B — Multidimensional arrays are stored contiguously in row-major order (row by row).
3. C — The compiler needs column bounds (all dimensions except the 1st) to compute element offset addresses.
4. B — Subscripts are written in separate square brackets: `A[row][col]`, so row 2 col 3 is `A[2][3]`.

Lecture 13: Array Manipulation & Design Recipe Case Study

Q1. What nesting structure is typically required to iterate through and manipulate a 2D matrix?

- A) A single `while` loop
- B) Nested loops (e.g. outer loop for rows, inner loop for columns)
- C) A `switch` statement inside a `do-while`
- D) Sequential `if` statements

Q2. What mathematical operation swaps rows and columns of a square matrix `A` such that `A[i][j]` becomes `A[j][i]`?

- A) Inversion
- B) Transposition
- C) Factorization
- D) Modulus reduction

Q3. When an array is passed as an argument to a function, how is it passed by default?

- A) By value (copying all elements)
- B) By reference (passing starting memory address)
- C) By constant value
- D) By pointer-to-pointer copy

Q4. In the "Unlucky Employees" real-world case study, what caused the anomaly where employees with higher gross salaries had lower take-home pay?

- A) System arithmetic overflow
- B) Disload of matrix bounds
- C) Higher tax deduction rates in higher tax brackets
- D) Miscalculated array indices

Answer Key

1. B — Processing 2D arrays/matrices requires nested loops to traverse rows and columns.
2. B — Matrix transposition interchanges rows and columns (`A[i][j]` becomes `A[j][i]`).
3. B — Array names decay to pointers to their first element, making array passing pass-by-reference.
4. C — Steep tax percentage jumps at bracket boundaries caused higher gross earners to take home less net pay.

Lecture 14: Pointers & Address Arithmetic

Q1. What does a pointer variable store?

- A) A floating-point number
- B) A memory address of another variable
- C) A character string literal
- D) A preprocessor macro definition

Q2. Which operator is used to obtain the memory address of a variable?

- A) Dereference operator `\*`
- B) Address-of operator `&`
- C) Scope resolution operator `::`
- D) Member access operator `->`

Q3. What is the process of accessing or modifying the value stored at the memory location pointed to by a pointer called?

- A) Referencing
- B) Dereferencing (using the `*`` operator)
- C) Allocation
- D) Typecasting

Q4. How does pass-by-reference using pointers benefit functions like `swap(int *x, int *y)`?

- A) It allows the function to modify the caller's actual variables directly.
- B) It prevents any modifications to variables.
- C) It reduces code compilation time.
- D) It eliminates the need for return statements in `main()`.

Answer Key

1. B — A pointer is a variable holding a memory location address as its value.
2. B — The `&`` operator returns the address of a variable in memory.
3. B — Dereferencing (`*ptr``) retrieves or updates the value at the target memory address.
4. A — Passing addresses allows a function to directly manipulate original variables in the calling scope.

Lecture 15: Pointers, Arrays & Pointer Arithmetic

Q1. How is the array name `arr`` related to pointers when `int arr[10];`` is declared?

- A) `arr`` is a dynamic pointer that can be reassigned to any memory location.
- B) `arr`` acts as a constant pointer to the first element (`&arr[0]``).
- C) `arr`` is a pointer to the total size of the array.
- D) `arr`` is unrelated to pointers.

Q2. If `ptr`` points to `arr[0]`` of an `int`` array (where `sizeof(int) = 4``), what memory address offset does `ptr + 1`` point to?

- A) 1 byte ahead
- B) 4 bytes ahead (the next array element `arr[1]``)
- C) 10 bytes ahead
- D) 8 bytes ahead

Q3. What value is returned when subtracting two pointers `ptr2 - ptr1`` that point to elements in the same array?

- A) The difference in raw bytes.
- B) The number of array elements between `ptr1`` and `ptr2``.
- C) The sum of addresses.
- D) Always 0.

Q4. Which of the following pointer arithmetic operations is ILLEGAL?

- A) Adding an integer to a pointer (`ptr + i``)
- B) Subtracting an integer from a pointer (`ptr - i``)
- C) Subtracting two pointers pointing to the same array (`ptr2 - ptr1``)
- D) Adding two pointers together (`ptr1 + ptr2``)

Answer Key

1. B — Array name acts as a constant pointer pointing to element 0; it cannot be reassigned.
2. B — Pointer arithmetic scales additions by the size of the underlying data type ($1 * 4$ bytes).
3. B — Pointer subtraction yields the distance between elements in terms of element counts.
4. D — Adding two memory addresses together ($ptr1 + ptr2$) is meaningless and illegal in C/C++.

Lecture 16: Pointers to Pointers & Command-Line Arguments

Q1. What is a pointer to a pointer?

- A) A variable that stores the address of another pointer variable.
- B) A pointer that points to itself.
- C) A function returning two pointers.
- D) An array of integers.

Q2. How is a pointer to a pointer to an integer declared?

- A) `int *ptr;`
- B) `int ptr;`
- C) `int &&ptr;`
- D) `int &*ptr;`

Q3. What do `argc` and `argv` represent in `main(int argc, char argv)`?

- A) Array count and Array variable
- B) Argument count (number of command-line arguments) and Argument vector (array of option strings)
- C) Allocation size and Address value
- D) Arithmetic condition and Global variable

Q4. In `argv`, what does `argv[0]` typically contain when a program is executed from the command line?

- A) The first argument supplied by the user.
- B) The total count of command-line arguments.
- C) The name or path of the running executable program itself.
- D) A NULL pointer.

Answer Key

1. A — A pointer-to-pointer stores the memory address where another pointer is stored.
2. B — Double asterisks ```` declare a double pointer (pointer to a pointer).
3. B — `argc` holds argument count, and `argv` is an array of C-string argument values.
4. C — `argv[0]` holds the program execution name/path, while user arguments start at `argv[1]`.

Lecture 17: String Handling & Character Functions

Q1. Which ```` function checks whether a character is an uppercase letter?

- A) `islower(c)`
- B) `isupper(c)`
- C) `isalpha(c)`

D) ``isdigit(c)``

Q2. Which function from ``<string.h>` is used to copy string ``s2`` into string ``s1``?

- A) ``strcat(s1, s2)``
- B) ``strcmp(s1, s2)``
- C) ``strcpy(s1, s2)``
- D) ``strlen(s1)``

Q3. What value does ``strcmp(s1, s2)`` return if string ``s1`` is identical to string ``s2``?

- A) 1
- B) 0
- C) -1
- D) Length of string

Q4. What character handling function returns true for spaces, tabs ``\t``, and newlines ``\n``?

- A) ``ispunct(c)``
- B) ``isspace(c)``
- C) ``isctrl(c)``
- D) ``isalnum(c)``

Answer Key

1. B — ``isupper()`` tests if a character falls in the range ``A`` through ``Z``.
2. C — ``strcpy(dest, src)`` copies source string including null terminator into destination buffer.
3. B — ``strcmp`` returns ``0`` when both compared strings contain identical character sequences.
4. B — ``isspace()`` checks for whitespace characters including space, horizontal/vertical tab, newline, feed, and carriage return.

Lecture 18: File Handling & Text Files

Q1. What is the main distinction between computer RAM memory and file storage on disk?

- A) RAM is permanent, while disk storage is volatile.
- B) RAM is volatile (lost on power off), while file storage on disk is non-volatile (persistent).
- C) RAM can only store binary data, while disk only stores plain text.
- D) RAM is slower than disk storage.

Q2. Which C++ stream class is used specifically for reading input from files?

- A) ``ofstream``
- B) ``ifstream``
- C) ``iostream``
- D) ``stdio``

Q3. What does opening a file with mode ``ios::app`` do?

- A) Overwrites existing file content from the beginning.
- B) Opens the file for appending output at the end of the file.
- C) Opens the file for binary input only.
- D) Deletes the file on close.

Q4. Why is error checking mandatory immediately after attempting to open a file stream?

- A) To prevent compiler syntax errors.
- B) To verify if the file exists and opened successfully before performing I/O operations.
- C) To automatically format the text inside the file.
- D) To clear the CPU cache.

Answer Key

1. B — RAM loses data when powered down (volatile); file storage on disk provides persistent data storage.
2. B — ``ifstream`` (input file stream) handles reading data from files.
3. B — ``ios::app`` positions the output file pointer at the end of file before every write operation.
4. B — Checking stream status (e.g. ``if (!file)``) prevents crashes and invalid operations if file opening fails.

Lecture 19: Random Access Files & File Pointers

Q1. Which C++ stream functions are used to obtain the current position of input and output file pointers respectively?

- A) ``seekg()`` and ``seekp()``
- B) ``tellg()`` and ``tellp()``
- C) ``get()`` and ``put()``
- D) ``read()`` and ``write()``

Q2. What is the purpose of ``file.seekg(10L, ios::beg)``?

- A) Move input file pointer 10 bytes backward from the end.
- B) Move input file pointer 10 bytes forward from the beginning of the file.
- C) Move output pointer to position 10.
- D) Read 10 bytes from current position.

Q3. Why is direct data overwriting in the middle of a sequential text file problematic if the new data length differs?

- A) The file system rejects middle writes.
- B) Existing subsequent data will be overwritten or corrupted due to fixed byte alignment on disk.
- C) Pointers cannot move backwards in text files.
- D) Text files do not support search functions.

Q4. Which binary stream functions read and write unformatted raw byte blocks between memory and files?

- A) ``cin`` and ``cout``
- B) ``read()`` and ``write()``
- C) ``getline()`` and ``putchar()``
- D) ``format()`` and ``parse()``

Answer Key

1. B — ``tellg()`` returns input get pointer location; ``tellp()`` returns output put pointer location.
2. B — ``seekg(offset, dir)`` with ``ios::beg`` offsets pointer from the start of the file.
3. B — Disk files cannot expand middle sections without overwriting adjacent bytes or copying to a new file.
4. B — ``read()`` and ``write()`` perform unformatted raw byte transfers to/from file streams.

Lecture 20: Structures (`struct`) & Unions (`union`)

Q1. What is a structure (`struct`) in C/C++?

- A) An array that only holds integer values.
- B) A user-defined data type that groups related variables of different data types under a single name.
- C) A preprocessor function for math operations.
- D) A built-in system file handle.

Q2. Which operator is used to access structure data members when using a structure pointer variable `sptr`?

- A) Dot operator `.`
- B) Arrow operator `->`
- C) Scope operator `::`
- D) Asterisk operator `\*` alone

Q3. What is the fundamental memory difference between a `struct` and a `union`?

- A) `struct` members share the same memory location, while `union` members have separate memory.
- B) `union` members share the same memory location (sized to its largest member), while `struct` members each have separate memory.
- C) `struct` cannot hold pointers, while `union` can.
- D) `union` cannot be used in functions.

Q4. If `struct Student { char name[30]; int age; float gpa; };` is declared, how is a member `age` accessed on an object `s1`?

- A) `s1->age`
- B) `s1.age`
- C) `Student.age`
- D) `s1::age`

Answer Key

1. B — A `struct` aggregates multiple heterogeneous data items into a single composite user-defined data type.
2. B — Member access through a pointer uses the arrow operator `sptr->member` (or `(\*sptr).member`).
3. B — Union members overlap in a single shared memory area large enough for the largest member; struct allocates distinct memory per member.
4. B — Direct structure objects access members using the dot operator `object.member`.

Lecture 21: Bit Manipulation & Bitwise Operators

Q1. Which bitwise operator performs a logical AND on corresponding individual bits of two integer operands?

- A) `&&`
- B) `&`
- C) `|`
- D) `^`

Q2. What is the result of applying the Bitwise XOR operator `^` on two identical bits (e.g. `1 ^ 1` or `0 ^ 0`)?

- A) ``1``
- B) ``0``
- C) ``-1``
- D) Depends on sign bit

Q3. Which bitwise operation is commonly used to "set" (force to 1) a specific bit in a bit flag variable?

- A) Bitwise AND ``&``
- B) Bitwise OR ``|``
- C) Bitwise NOT ``~``
- D) Left Shift ``<<``

Q4. What does the bitwise left shift expression ``x << 1`` do to an unsigned integer value arithmetic-wise?

- A) Divides ``x`` by 2
- B) Multiplies ``x`` by 2
- C) Sets ``x`` to 0
- D) Negates ``x``

Answer Key

1. B — Single ampersand ``&`` is the bitwise AND operator operating on corresponding bit pairs.
2. B — Bitwise XOR (``^``) yields ``0`` when bits match, and ``1`` when bits differ.
3. B — Bitwise OR (``|``) with a mask containing ``1`` at target bit sets that bit to ``1`` without altering others.
4. B — Shifting bit patterns left by 1 position effectively doubles the integer value (``x * 2``).

Lecture 22: Bitwise Assignment Operators & Mid-Term Review

Q1. What is the equivalent compound assignment expression for ``a = a & b;``?

- A) ``a &= b;``
- B) ``a &&= b;``
- C) ``a &= &b;``
- D) ``a =| b;``

Q2. Why is there NO compound assignment operator available for the Bitwise NOT operator ``~``?

- A) Because ``~`` is a binary operator.
- B) Because ``~`` is a unary operator operating on a single operand (``~a``).
- C) Because ``~`` is not supported in C++.
- D) Because ``~`` only operates on floating point numbers.

Q3. Which core control constructs are sufficient to express any computable programming logic according to structured programming theory?

- A) Pointers, Dynamic Allocation, Recursion
- B) Sequences, Decisions (``if/else``), and Loops (``while/for``)
- C) Preprocessor Macros, Headers, Linkers
- D) Structures, Unions, Classes

Q4. What is the default function parameter passing mechanism in C for ordinary variables?

- A) Call by Reference
- B) Call by Value
- C) Call by Name
- D) Call by Pointer

Answer Key

1. A — `a &= b` is short-hand compound bitwise AND assignment.
2. B — Bitwise NOT (`~`) takes only one operand (unary), so a compound binary assignment operator does not exist.
3. B — Sequence, selection (decision), and iteration (repetition) form the three foundational control structures.
4. B — C passes scalar arguments by value by default unless explicit pointers/references are used.

Lecture 23: Preprocessor Directives & Macros

Q1. What character must precede all preprocessor directives in C/C++?

- A) Semicolon `;`
- B) Hash / Sharp sign `#`
- C) Asterisk `*`
- D) Double slash `//`

Q2. What does the directive `#define PI 3.14159` instruct the preprocessor to do?

- A) Declare a global floating point variable `PI`.
- B) Replace every occurrence of the token `PI` in source code with `3.14159` before compilation.
- C) Include a library named `PI`.
- D) Check if `PI` is defined in system memory.

Q3. How can a previously defined symbolic constant `PI` be removed so it can be redefined or disabled?

- A) `#delete PI`
- B) `#undef PI`
- C) `#remove PI`
- D) `#clear PI`

Q4. Why should preprocessor macros taking arguments always enclose parameters in parentheses inside definitions (e.g. `#define SQUARE(x) ((x)*(x))`)?

- A) To allow compiler optimization.
- B) To avoid unexpected precedence errors during macro text expansion.
- C) Required by C syntax parser.
- D) To convert arguments to pointers.

Answer Key

1. B — All preprocessor directives begin with the `#` symbol.
2. B — `#define` performs literal macro token text replacement during preprocessor expansion.
3. B — `#undef` undefines a macro symbol, removing it from the preprocessor symbol table.
4. B — Parentheses enforce correct operator evaluation order when complex expressions are passed into macros.

Lecture 24: Dynamic Memory Allocation in C

Q1. Where is dynamic memory allocated from at runtime?

- A) Call Stack
- B) Free Store / Heap
- C) CPU Registers
- D) Code Segment

Q2. Which standard C library function allocates `n` blocks of memory and initializes all bytes to zero?

- A) ``malloc()`
- B) ``calloc()`
- C) ``realloc()`
- D) ``free()`

Q3. What severe problem occurs when dynamically allocated memory is no longer referenced but is never freed using ``free()`?

- A) Syntax Error
- B) Memory Leak
- C) Dangling Pointer
- D) Stack Overflow

Q4. What is a "Dangling Pointer"?

- A) A pointer initialized to ``NULL``.
- B) A pointer that continues to point to a memory address that has already been deallocated/freed.
- C) A pointer pointing to a global variable.
- D) A double pointer that points to another pointer.

Answer Key

1. B — Dynamic memory is requested and allocated at runtime from the system Heap / Free Store.
2. B — ``calloc(num, size)`` allocates contiguous memory and clears all bits to zero; ``malloc`` leaves memory uninitialized.
3. B — Memory leaks occur when allocated heap memory loses all pointers without being deallocated.
4. B — A dangling pointer references memory that has been deallocated, posing severe runtime risks if accessed.

Lecture 25: C++ Features, Inline Functions & Overloading

Q1. How does an ``inline`` function differ from a standard function call?

- A) It executes in a separate thread.
- B) The compiler attempts to insert the function's code inline at the call site, eliminating function call overhead.
- C) It can only be called once per program.
- D) It cannot take any parameters.

Q2. What is Function Overloading in C++?

- A) Defining multiple functions with the exact same name and parameter list.
- B) Defining multiple functions with the same name but different parameter lists (type or number of arguments).
- C) Calling a function recursively until memory overflows.
- D) Replacing function calls with preprocessor macros.

Q3. Can two overloaded functions differ ONLY by their return types?

- A) Yes, C++ differentiates them by return type.
- B) No, functions must differ in their parameter signatures; return type alone is insufficient.
- C) Yes, if both return types are pointers.
- D) Yes, if one return type is `void`.

Q4. What does C++ allow regarding variable declarations compared to C?

- A) Variables must all be declared at the top of a function block in C++.
- B) Variables can be declared anywhere in the code right before they are used.
- C) Variables can never be declared inside `for` loop headers.
- D) Variables must be global.

Answer Key

1. B — `inline` suggests the compiler replace call sites with full function body code to save context-switch overhead.
2. B — Function overloading allows same function name with distinct parameter lists.
3. B — Overload resolution depends exclusively on parameter count and types, not on the return type.
4. B — C++ permits flexible placement of variable declarations close to their first usage point.

Lecture 26: Classes, Objects & Encapsulation

Q1. What is a Class in C++?

- A) A built-in array of integers.
- B) A user-defined data type that combines data members and member functions that operate on that data.
- C) A preprocessor file directive.
- D) A standard file stream.

Q2. What are instances of a class called?

- A) Methods
- B) Objects
- C) Structures
- D) Pointers

Q3. What is the DEFAULT access visibility specifier for members of a `class` in C++ if unspecified?

- A) `public`
- B) `private`
- C) `protected`
- D) `global`

Q4. What C++ OOP concept hides internal data implementation details and restricts direct external access?

- A) Inheritance
- B) Encapsulation (Data Hiding)
- C) Polymorphism
- D) Compilation

Answer Key

1. B — A class is a blueprint combining data fields (members) and code methods into a unified user-defined type.
2. B — An object is a concrete runtime instance of a class type.
3. B — Members of a C++ `class` default to `private` access (unlike `struct` which defaults to `public`).
4. B — Encapsulation wraps data and code together while restricting raw external access to private data.
-

Lecture 27: Constructors, Utility Functions & Destructors**Q1. What is a Constructor in a C++ class?**

- A) A special member function with the same name as the class, called automatically when an object is created.
- B) A function that deletes an object from memory.
- C) A preprocessor statement that links libraries.
- D) A function that overloads math operators.

Q2. Which of the following is true regarding Destructors in C++?

- A) A class can have multiple overloaded destructors.
- B) A destructor takes the same name as the class preceded by a tilde `~` and accepts no arguments.
- C) Destructors return an integer exit code.
- D) Destructors must be called manually using `cout`.

Q3. What are "Utility Functions" in a class?

- A) Public functions called directly by `main()`.
- B) Private member functions that support inner operations of other class methods and are hidden from outside users.
- C) Functions that handle file input/output.
- D) Global functions defined outside the class.

Q4. When is a class's Destructor executed automatically?

- A) When an object is declared.
- B) When an object goes out of scope or is explicitly deleted.
- C) When a syntax error occurs.
- D) Before the constructor runs.

Answer Key

1. A — Constructors share the class name, have no return type, and initialize objects upon instantiation.
2. B — Destructors are named `~ClassName()`, take no arguments, return no value, and cannot be overloaded.

- 3. B — Utility functions are private helper methods designed for internal class operations.
- 4. B — Destructors automatically execute when an object's scope ends or when deleted from the heap.
-

Lecture 28: Dynamic Memory Allocation in C++ (`new` & `delete`)

Q1. What operator replaces C's `malloc()` in C++ to dynamically allocate memory and invoke constructors?

- A) `alloc`
- B) `new`
- C) `create`
- D) `build`

Q2. How is a dynamically allocated array of 10 `Date` objects created using `new`?

- A) `Date *dptr = malloc(10 * sizeof(Date));`
- B) `Date *dptr = new Date[10];`
- C) `Date *dptr = new Date(10);`
- D) `Date *dptr = create Date[10];`

Q3. What is the correct syntax to deallocate a dynamically allocated array `dptr` in C++?

- A) `delete dptr;`
- B) `delete [] dptr;`
- C) `free(dptr);`
- D) `remove dptr[];`

Q4. What happens when `new` is used to allocate an object of a class?

- A) Memory is allocated from heap, and then the class constructor is executed.
- B) Only memory is allocated without calling constructors.
- C) The object is declared as global.
- D) Destructor is invoked immediately.

Answer Key

- 1. B — `new` allocates memory and calls the appropriate constructor.
- 2. B — `new ClassName[count]` allocates an array of objects dynamically.
- 3. B — Deallocating dynamic arrays requires `delete [] pointer;` to ensure destructors run for all array elements.
- 4. A — `new` performs memory allocation followed by constructor execution for initialization.
-

Lecture 29: Friend Functions & Friend Classes

Q1. What privilege does declaring a function as a `friend` inside a class grant that function?

- A) It allows the function to inherit the class.
- B) It grants the function direct access to all `private` and `protected` members of the class.
- C) It prevents the function from accessing public members.
- D) It makes the function a static member.

Q2. Is friendship between classes in C++ mutual (two-way) by default?

- A) Yes, if Class A is a friend of Class B, Class B is automatically a friend of Class A.
- B) No, friendship is one-way; Class A declaring Class B as a friend does NOT automatically make Class A a friend of B.
- C) Yes, friendship is always transitive across all classes.
- D) No, friend classes are not permitted in C++.

Q3. How is a friend function `displaySecret` declared inside a class `Vault`?

- A) `public friend void displaySecret(Vault &v);`
- B) `friend void displaySecret(Vault &v);`
- C) `void friend :: displaySecret(Vault &v);`
- D) `class friend displaySecret(Vault &v);`

Q4. Where is the body of a friend function typically defined?

- A) Inside the private section of the class.
- B) Outside the class scope as an ordinary standalone function (without `ClassName::`).
- C) Inside `main()` only.
- D) In a preprocessor macro file.

Answer Key

1. B — `friend` status permits an external non-member function to access private class members.
2. B — Friendship is non-reciprocal and non-transitive; it must be explicitly granted by each class.
3. B — The `friend` keyword precedes the function declaration inside the granting class.
4. B — Friend functions are standalone non-member functions and are defined without scope resolution operators.

Lecture 30: Reference Data Types

Q1. What is a Reference in C++?

- A) A pointer that can be set to NULL.
- B) An alias or synonym for an existing variable.
- C) A dynamic memory address on heap.
- D) A copy of a variable created on stack.

Q2. How is an integer reference `ref` initialized to refer to variable `x`?

- A) `int &ref; = x;`
- B) `int ref = &x;`
- C) `int *ref = x;`
- D) `int ref & x;`

Q3. Which of the following is a key difference between Pointers and References?

- A) Pointers can be reassigned to point to different variables, whereas references CANNOT be reassigned after initialization.
- B) References can perform arithmetic (`ref++`), but pointers cannot.
- C) Pointers must be initialized upon declaration, but references do not need initialization.
- D) References require dereference asterisks `*` to access values.

Q4. What is a "Dangling Reference"?

- A) A reference pointing to a global constant.

- B) A reference referring to an object or memory location that has already been destroyed or gone out of scope.
- C) A reference initialized to 0.
- D) An unassigned pointer.

Answer Key

1. B — A reference acts as an alternative alias name for an existing variable location.
2. A — `int &ref; = x;` binds reference `ref` directly to variable `x`.
3. A — References cannot be rebound to new variables after initialization and do not support arithmetic like pointers.
4. B — Returning or using references to out-of-scope local stack variables produces invalid dangling references.

Lecture 31: Operator Overloading Concepts

Q1. What is Operator Overloading in C++?

- A) Creating new custom symbols like `@` or `#` for operations.
- B) Giving existing C++ operators custom implementations when applied to user-defined class objects.
- C) Changing operator precedence rules.
- D) Overloading function names with numbers.

Q2. Which of the following operators CANNOT be overloaded in C++?

- A) `+`
- B) `==`
- C) Scope Resolution Operator `::`
- D) `[]`

Q3. Can operator overloading change the number of operands an operator takes (e.g. making binary `+` take 3 operands)?

- A) Yes, operands can be set to any count.
- B) No, the arity (number of operands) and precedence of existing operators remain fixed and unchanged.
- C) Yes, if defined as a friend function.
- D) Yes, if defined as inline.

Q4. When an operator is overloaded as a CLASS MEMBER function, how is the left-hand operand passed?

- A) As the first argument in the parameter list.
- B) Implicitly passed as the calling object pointed to by `this`.
- C) As a global variable.
- D) It must be passed by value.

Answer Key

1. B — Operator overloading redefines how standard built-in operators behave with class objects.
2. C — Operators like `::`, `&`, `*`, and `sizeof` cannot be overloaded in C++.
3. B — Overloading preserves the intrinsic syntax, arity, precedence, and associativity of operators.

4. B — For member operator functions, the left operand is the implicit caller object (`\*this`).

Lecture 32: Overloading Unary & Binary Arithmetic Operators

Q1. How does C++ distinguish between Prefix increment `++x` and Postfix increment `x++` operator overloading signatures?

- A) Postfix increment includes a dummy `int` parameter in its signature `operator++(int)`.
- B) Prefix increment uses double plus signs `++++`.
- C) Postfix returns `void`, while prefix returns `int`.
- D) They use different operator keywords.

Q2. If binary `+` is overloaded as a member function `Complex operator+(const Complex &rhs); const`, what represents the left-hand operand in expression `c1 + c2`?

- A) `rhs`
- B) The caller object `c1` (accessed via `this`)
- C) `c2`
- D) A global temporary

Q3. Why is overloading operators with non-intuitive meanings (e.g. making `+` perform subtraction) considered bad programming practice?

- A) The compiler will reject the code.
- B) It makes code unreadable, confusing, and prone to severe logical errors.
- C) It slows down program execution speed.
- D) It violates access rules.

Q4. In the `Date` class case study, what complex logic must `operator++()` handle when incrementing a date?

- A) Only incrementing the `day` integer.
- B) Rollover of days across different month lengths (28, 30, 31 days), leap years, and year changes.
- C) Converting string dates to ASCII.
- D) Clearing system stream buffers.

Answer Key

- 1. A — `operator++()` declares prefix increment; `operator++(int)` uses a dummy int argument to signal postfix increment.
- 2. B — In member binary operators, `c1 + c2` invokes `c1.operator+(c2)`, making `c1` the implicit `this` object.
- 3. B — Operators should preserve conventional expected semantics to maintain clear, readable code.
- 4. B — Date incrementation must manage month boundaries, varying days per month, leap years, and year increments.

Lecture 33: Assignment Operator & Type Conversions

Q1. Why must classes with dynamic memory allocation explicitly overload the Assignment Operator (`operator=`)?

- A) To prevent syntax errors.

- B) To perform a Deep Copy (allocating new memory) instead of a default memberwise Shallow Copy (which leads to double deletion).
- C) To allow assignment to integer variables.
- D) To disable object assignment completely.

Q2. What critical check should be performed at the beginning of an overloaded assignment operator `operator=`?

- A) Check for division by zero.
- B) Self-assignment check (`if (this == &rhs;) return *this;`).
- C) Check if object is NULL.
- D) Clear all member variables to 0.

Q3. What keyword/mechanism allows an object of a class to be implicitly converted from a single primitive argument type (e.g. `Complex c = 5.0;`)?

- A) Conversion Constructor (a single-parameter constructor)
- B) Preprocessor macro
- C) Scope resolution
- D) Friend function

Q4. What does the `this` pointer represent inside a class member function?

- A) A pointer to the next object in memory.
- B) A pointer to the current instance (calling object) executing the member function.
- C) A pointer to the global class table.
- D) A reference to the parent class.

Answer Key

1. B — Default assignment performs shallow copies; custom `operator=` ensures deep memory duplication.
2. B — Self-assignment check (`this == &rhs;`) prevents deleting memory of an object assigned to itself (`x = x`).
3. A — Constructors callable with a single argument act as implicit type conversion constructors.
4. B — `this` holds the memory address of the specific object instance executing the member method.

Lecture 34: Arrays of Objects & Operator `[]`

Q1. When an array of objects `Date calendar[10];` is declared statically, which constructor is called for each element?

- A) The parameterized constructor with integer arguments.
- B) The Default (parameterless) Constructor.
- C) The Copy Constructor.
- D) No constructor is called.

Q2. How can `operator new` and `operator delete` be overloaded for a specific class?

- A) By defining custom `void * operator new(size_t size)` and `void operator delete(void *ptr)` member or global functions.
- B) By redefining preprocessor `#include` directives.
- C) By modifying compiler settings.
- D) They cannot be overloaded.

Q3. Why is overloading the Subscript Operator `operator[]` useful for custom Array classes?

- A) It allows objects to be printed using `cout`.
- B) It enables array-like bracket syntax `arr[i]` with custom bounds checking and safety.
- C) It converts arrays into linked lists.
- D) It speeds up array sorting algorithms.

Q4. What must an overloaded subscript operator `int & operator[](int index)` return to allow assignment to array elements (e.g. `arr[3] = 25;`)?

- A) A copy of the value (`int`)
- B) A reference to the element (`int &`)
- C) A void pointer
- D) A boolean flag

Answer Key

1. B — Instantiating static arrays of objects automatically invokes the default parameterless constructor for each element.
 2. A — Overloading `operator new` takes `size_t` and returns `void*`; `operator delete` takes `void*` pointer to free.
 3. B — Overloading `[]` allows custom array classes to enforce index safety and bounds checking.
 4. B — Returning an lvalue reference (`int&`) permits elements to receive assigned values on the left side of `=`.
-

Lecture 35: Stream I/O & Stream Classes

Q1. In C++, what is a Stream conceptually?

- A) A random collection of memory addresses.
- B) An ordered sequence of bytes flowing from a source to a destination.
- C) A file compression algorithm.
- D) A thread execution pool.

Q2. Which predefined stream object represents the standard input (keyboard) in C++?

- A) `cout`
- B) `cin`
- C) `cerr`
- D) `clog`

Q3. What is the advantage of Buffered I/O in stream operations?

- A) It encrypts input data.
- B) It collects data in a temporary memory buffer to minimize expensive physical I/O disk/device operations.
- C) It eliminates syntax errors.
- D) It automatically formats numbers as hexadecimal.

Q4. Which `cin` member function reads a full line of text including spaces into a character array buffer?

- A) `cin.get()`
- B) `cin.getline(buffer, size)`

- C) ``cin.read()```
- D) ``cin >> buffer```

Answer Key

1. B — A stream represents a sequential flow of bytes between programs and I/O devices.
2. B — ``cin`` (instance of ``istream``) handles standard keyboard input.
3. B — Buffering aggregates small transfers into larger blocks, dramatically speeding up I/O throughput.
4. B — ``cin.getline()``` reads characters up to newline or buffer limit, preserving whitespace inside strings.

Lecture 36: Stream Manipulators & Formatting

Q1. Which non-parameterized stream manipulator changes integer output display to hexadecimal format?

- A) ``oct```
- B) ``dec```
- C) ``hex```
- D) ``setbase(16)```

Q2. What does the parameterized manipulator ``setw(10)``` do when outputting a value?

- A) Sets floating-point precision to 10 decimal places.
- B) Sets the minimum field width for the next output item to 10 spaces.
- C) Repeats the output item 10 times.
- D) Inserts 10 newline characters.

Q3. Which header file must be included to use parameterized manipulators like ``setw()```, ``setfill()```, and ``setprecision()```?

- A) ````
- B) ````
- C) ````
- D) ````

Q4. What is the purpose of ``cout << endl;``` compared to ``cout << "```

- A) ``endl``` outputs a newline AND flushes the output stream buffer immediately.
- B) ``endl``` only inserts a tab space.
- C) ``endl``` clears the monitor screen.
- D) There is no difference whatsoever.

Answer Key

1. C — ``hex`` sets the stream conversion base to 16 for subsequent integer outputs.
2. B — ``setw(n)``` specifies the field width in columns for the next formatted item.
3. B — Parameterized manipulators require ```` (or ````).
4. A — ``endl`` inserts a newline character and forces a buffer flush via ``flush()```.

Lecture 37: Overloading Stream Operators (``<<`` and ``>>``)

Q1. Why MUST Stream Insertion (`<<`) and Extraction (`>>`) operators be overloaded as FRIEND functions rather than member functions?

- A) Because stream operators cannot access private data.
- B) Because the left-hand operand is a Stream object (`ostream` or `istream`), NOT an object of our custom class.
- C) Because friend functions execute faster.
- D) Required by the preprocessor compiler.

Q2. What return type must an overloaded stream insertion operator `operator<<` have to support cascaded operations like `cout << obj1 << obj2;`?

- A) `void`
- B) `ostream &` (reference to stream object)
- C) `int`
- D) `bool`

Q3. What parameter signature is expected for `operator>>` overloading Stream Extraction for class `Matrix`?

- A) `istream & operator>>(istream &is, Matrix &m;)`
- B) `void operator>>(Matrix &m, istream &is;)`
- C) `Matrix operator>>(istream &is;)`
- D) `ostream & operator>>(ostream &os, Matrix m)`

Q4. Why is the class object parameter passed by REFERENCE (`Matrix &m;`) in `operator>>`?

- A) To prevent copying streams.
- B) To allow the function to modify the actual object's data members with input values.
- C) To make it a constant argument.
- D) Required for inline expansion.

Answer Key

1. B — Left operand is `ostream`/`istream`; since we cannot modify stream class definitions, non-member friends are used.
2. B — Returning `ostream &` passes the stream reference along to evaluate chained `<<` operators sequentially.
3. A — `operator>>` receives `istream&` and non-const reference `Matrix&` to populate input fields.
4. B — Extraction mutates the object; passing by non-const reference allows direct writing to original fields.

Lecture 38: User-Defined Manipulators & Static Keywords

Q1. What is the lifetime of a `static` variable declared inside a function?

- A) It is destroyed as soon as the function returns.
- B) It persists throughout the entire execution lifetime of the program, retaining its value between function calls.
- C) It lasts only during compiler execution.
- D) It is lost when CPU changes threads.

Q2. How are Static Data Members of a class shared among objects of that class?

- A) Every object receives its own independent copy.

- B) Exactly ONE single copy exists in memory, shared by ALL instances/objects of that class.
- C) Static members are stored on the call stack.
- D) They are destroyed after constructor finishes.

Q3. How must a Static Data Member `static int count;` of class `Item` be defined outside the class?

- A) `int Item::count = 0;`
- B) `static int Item::count = 0;`
- C) `void Item::count = 0;`
- D) `Item.count = 0;`

Q4. How is a user-defined non-parameterized stream manipulator (e.g. `newLine`) implemented?

- A) As a function taking and returning `ostream &` reference (e.g. `ostream& newLine(ostream &os;)`).
- B) As a preprocessor macro.
- C) As a virtual destructor.
- D) As a class constructor.

Answer Key

1. B — Static local variables are initialized once and persist in memory for the program's entire duration.
2. B — Static class members belong to the class itself; a single shared memory location is accessed by all objects.
3. A — Static data members require explicit global definition/storage allocation outside class header without repeating `static`.
4. A — Stream manipulators take `ostream&` as parameter and return `ostream&` reference to maintain stream chaining.

Lecture 39: Comprehensive Review of Pointers & Memory Rules

Q1. When an object containing dynamic memory pointers is passed by value to a function, what major issue occurs if no Copy Constructor is defined?

- A) Preprocessor compilation halts.
- B) Default shallow copy duplicates raw pointer addresses, causing both objects to point to identical memory and resulting in double deletion runtime crashes.
- C) Memory size expands automatically.
- D) The original object is converted into a reference.

Q2. What three critical components form the "Rule of Three" for classes managing dynamic memory?

- A) Constructor, Inline function, Friend function
- B) Custom Copy Constructor, Overloaded Assignment Operator, and Destructor
- C) Static Member, Utility Function, Virtual Method
- D) Header file, Main function, Preprocessor macro

Q3. What is the purpose of a Copy Constructor signature `ClassName(const ClassName &source;)`?

- A) To initialize a newly created object as a deep copy of an existing object.
- B) To delete an existing object.

- C) To convert an object to an integer.
- D) To overload stream operators.

Q4. Why is pass-by-reference preferred over pass-by-value for large object parameters?

- A) It enforces global variable access.
- B) It eliminates the performance overhead of copying large memory structures onto the stack.
- C) It prevents functions from returning values.
- D) It guarantees thread safety.

Answer Key

1. B — Shallow copying duplicates pointer addresses; when destructors run for both objects, same heap memory is freed twice.
2. B — Managing dynamic resources requires custom Copy Constructor, Assignment Operator, and Destructor (Rule of Three).
3. A — Copy constructor creates an independent object initialized with deep-copied data from source.
4. B — Passing references avoids expensive copy operations and stack allocations for large objects.

Lecture 40: Composition (Containment) & Nested Classes

Q1. What is Object Composition (Containment)?

- A) Deriving a class from another class.
- B) Including an object of one class as a data member inside another class.
- C) Overloading functions inside a struct.
- D) Defining friend functions.

Q2. What is the order of Constructor execution when Class `B` contains an object of Class `A` as a data member?

- A) Class `B`'s constructor executes first, then Class `A`'s constructor.
- B) Contained object `A`'s constructor executes FIRST, followed by containing object `B`'s constructor.
- C) Both execute simultaneously.
- D) Neither constructor is executed.

Q3. What is the order of Destructor execution for composite objects?

- A) Contained object `A` is destroyed first.
- B) Containing object `B`'s destructor executes FIRST, followed by contained object `A`'s destructor (reverse of construction).
- C) Order is random.
- D) Only `B`'s destructor runs.

Q4. What C++ construct inside a constructor header allows passing initialization arguments to contained member objects?

- A) Scope resolution
- B)_INITIALIZER List (e.g. `B() : a(args) { ... }`)
- C) Preprocessor directive
- D) Friend declaration

Answer Key

1. B — Composition ("has-a" relationship) embeds object instances of other classes as data fields.

2. B — Member objects are constructed first before the outer enclosing class constructor executes.
3. B — Destruction order strictly reverses construction order: outer enclosing object destructs before inner member objects.
4. B — Member Initializer Lists pass arguments directly to contained member object constructors before body runs.

Lecture 41: Function Templates

Q1. What is the main purpose of Function Templates in C++?

- A) To speed up disk reading operations.
- B) To write generic functions that work with arbitrary data types without duplicating code.
- C) To replace classes with structures.
- D) To define preprocessor macros.

Q2. How is a Function Template declared in C++?

- A) ``template`` or ``template``
- B) ``generic``
- C) ``macro``
- D) ``include``

Q3. If a template function ``template T getMax(T a, T b)`` is called with ``getMax(5, 10)``, what type does the compiler deduce for ``T``?

- A) ``float``
- B) ``int``
- C) ``double``
- D) ``char``

Q4. What must a custom class provide if it is passed as a type argument to a Function Template that uses operators like ``<`` or ``+``?

- A) Friend class definitions.
- B) Overloaded implementations of those specific operators (``<`` or ``+``).
- C) Virtual destructors.
- D) Private constructors.

Answer Key

1. B — Templates support generic programming, enabling logic to work seamlessly across multiple data types.
2. A — ``template`` or ``template`` introduces template type parameter ``T``.
3. B — Compiler template argument deduction infers ``T = int`` from integer arguments ``5`` and ``10``.
4. B — Custom types must support all operators used within the generic template function logic.

Lecture 42: Class Templates & STL Introduction

Q1. How is an object of a Class Template ``Array`` instantiated for storing ``float`` values?

- A) ``Array float obj;``
- B) ``Array obj;``

- C) ``Array(float) obj;``
- D) ``template Array[float] obj;``

Q2. Can Class Templates accept non-type parameters (such as integer constants) in their template header ``template``?

- A) No, templates only accept data types.
- B) Yes, non-type parameters pass compile-time constant values (like fixed array sizes) into templates.
- C) No, non-type parameters cause syntax errors.
- D) Only if ``size`` is a float.

Q3. How are static data members handled in Class Templates?

- A) All instantiations share one global static member.
- B) Each distinct template type instantiation (e.g. ``Array`` vs ``Array``) gets its OWN separate static data member.
- C) Class templates cannot have static members.
- D) Static members are stored in header files.

Q4. What is the C++ Standard Template Library (STL)?

- A) A collection of hardware driver utilities.
- B) A standard C++ library providing powerful, reusable class templates for container data structures, algorithms, and iterators.
- C) A set of preprocessor macros for graphic interfaces.
- D) A database compiler.

Answer Key

1. B — Class template instantiation requires explicit type arguments in angle brackets ``ClassName``.
2. B — Non-type parameters allow compile-time constants (e.g. buffer sizes) to configure template instances.
3. B — Each type-specific instantiation of a class template maintains its own independent set of static variables.
4. B — STL provides standardized, highly optimized generic containers (vectors, lists), algorithms, and iterators.

Lecture 43: Matrix Class Case Study - Analysis & Design

Q1. In the Matrix class design case study, what condition MUST be met to add or subtract two matrices ``A`` and ``B``?

- A) Both matrices must be square (rows = columns).
- B) Both matrices must have identical dimensions (same number of rows and same number of columns).
- C) Column count of ``A`` must equal row count of ``B``.
- D) Neither matrix can contain zeros.

Q2. What condition MUST be satisfied to multiply Matrix ``A`` (size $m \times n$) by Matrix ``B`` (size $p \times q$)?

- A) $m = q$
- B) The number of columns in A (n) must equal the number of rows in B (p).
- C) Both matrices must be 3×3 .
- D) $m = p$ and $n = q$

Q3. What will be the resulting dimension of Matrix `C` when multiplying Matrix `A` (3×4) by Matrix `B` (4×5)?

- A) 4×4
- B) 3×5
- C) 5×3
- D) 12×20

Q4. Why is overloading operators like `+`, `-`, `\*`, `+=`, `-=` appropriate for a custom `Matrix` class?

- A) To bypass C++ access visibility rules.
- B) To allow matrix expressions to be written using natural mathematical notation (e.g. `C = A + B * 2.0;`).
- C) Required by compiler loaders.
- D) To prevent dynamic memory leaks.

Answer Key

1. B — Matrix addition/subtraction requires conformable matrices with matching row and column counts.
2. B — Matrix multiplication requires inner dimensions to match: columns of left matrix = rows of right matrix.
3. B — Multiplying ($m \times n$) by ($n \times p$) yields a matrix of outer dimensions ($m \times p$), i.e., 3×5 .
4. B — Operator overloading allows custom objects to be manipulated with natural, intuitive algebraic syntax.

Lecture 44: Matrix Class Implementation - Memory & Allocation

Q1. How is a 2D dynamic array representation allocated in memory for a `Matrix` class with `numRows` and `numCols`?

- A) As a single 1D array `new double[numRows * numCols]`.
- B) As an array of pointers to `double` (`new double*[numRows]`), where each pointer allocates a 1D array of `numCols` doubles.
- C) Using `malloc(sizeof(Matrix))`.
- D) As a static local array.

Q2. How does the `Matrix` Destructor properly free memory allocated for its 2D dynamic array `elements`?

- A) `delete elements;`
- B) Loop through each row pointer freeing `delete [] elements[i];`, then free the main pointer array `delete [] elements;`.
- C) `free(elements);`
- D) `delete [][] elements;`

Q3. What does the Copy Constructor `Matrix::Matrix(const Matrix &m);` do?

- A) Copies only the raw `elements` pointer address.
- B) Allocates new row and column arrays and performs a element-by-element deep copy of `m`'s data values.
- C) Sets all elements to zero.
- D) Deletes matrix `m`.

Q4. How are elements accessed in a 2D dynamically allocated array ``elements`` inside ``Matrix`` methods?

- A) ``elements[row, col]``
- B) ``elements[row][col]``
- C) ``elements(row)(col)``
- D) ``elements->row->col``

Answer Key

1. B — 2D dynamic arrays allocate a primary array of row pointers, then allocate column arrays for each row pointer.
2. B — Proper 2D deallocation requires deleting each individual row array before deleting the top-level pointer array.
3. B — Deep copy allocates fresh 2D memory arrays and duplicates every scalar entry from the source matrix.
4. B — Double subscript syntax ``elements[row][col]`` dereferences row pointer and accesses column index.
-

Lecture 45: Matrix Operators, SQL & Course Synthesis

Q1. In the ``Matrix`` class, why is scalar addition ``m + d`` implemented as a member function, whereas ``d + m`` (double + ``Matrix``) requires a FRIEND function?

- A) Member functions cannot return matrices.
- B) In ``d + m``, the left operand is primitive ``double``, so the operator cannot be a caller member method of ``Matrix``.
- C) ``d + m`` is an illegal operation in C++.
- D) Friend functions run faster.

Q2. How does ``operator/`` handle division of a ``Matrix`` by a double constant ``d = 0.0``?

- A) Throws an immediate hardware interrupt.
- B) Checks for division by zero and avoids division to prevent runtime crashes.
- C) Sets all matrix elements to infinity.
- D) Converts matrix to string.

Q3. What three core constructs form the foundation required to write any computer program?

- A) Pointers, References, Memory Allocation
- B) Sequence, Decision (``if/else``), and Repetition (``loops``)
- C) Classes, Objects, Inheritance
- D) Compilers, Linkers, Loaders

Q4. What mechanism does Java use to handle unreferenced dynamic memory automatically, contrasting with manual ``delete`` in C++?

- A) Destructor chains
- B) Automatic Garbage Collection
- C) Preprocessor deletion
- D) Pointer counting

Answer Key

1. B — Member operators require left-operand to be a class instance; when left-operand is ``double``, a non-member friend function is required.

2. B — Checking `d != 0.0` guards against fatal division-by-zero runtime exceptions.
3. B — Sequence, selection (decision), and repetition (loops) are the three fundamental control structures of algorithm design.
4. B — Java features automatic garbage collection to reclaim unreferenced memory, eliminating manual deallocation required in C++.

End of Quiz

CS201 — 180 solved MCQs
Prepared by VU Learning HUB
vulearninghub.xyz