RGIT FE-SAHYOG

Module 1: Fundamentals of C Programming

Q1) What is the character set in C?

Answer: The C character set is the collection of letters, digits, special symbols, and whitespace characters that can be used to write C programs.

Memory Tip: Think: letters + digits + symbols + spaces = character set.

Q2) What is an identifier in C?

Answer: An identifier is a name given to program elements like variables, functions, arrays, etc., defined by the programmer.

Memory Tip: Identifier = user-given name.

Q3) What are keywords in C?

Answer: Keywords are reserved words that have special meaning in C and cannot be used as identifiers, such as `int`, `float`, `if`, `return`.

Memory Tip: Keywords are "locked" words; you cannot rename them.

Q4) What is a data type in C?

Answer: A data type specifies what kind of data a variable can store, such as `int` for integers, `float` for real numbers, `char` for characters.

Memory Tip: Data type = kind + size of data.

Q5) What is a constant? Give one example.

Answer: A constant is a value that does not change during program execution, for example `10`, `3.14`, or `'A'`.

Memory Tip: "Constant" = fixed value.

Q6) What is a variable?

Answer: A variable is a named memory location whose value can change during program execution, like `int count;`.[1]

Memory Tip: Variable = "value varies".

Q7) Distinguish between variable and constant.

Answer: A variable's value can change while the program runs, whereas a constant's value

remains the same throughout execution.

Memory Tip: Variable changes; constant remains.

Q8) What is the difference between `int` and `float`?

Answer: `int` stores whole numbers without fractional part, while `float` stores real numbers with decimal or fractional part.

Memory Tip: `int` = integer only; `float` = fraction allowed.

Q9) Define `char` data type with an example declaration.

Answer: `char` is used to store a single character; for example, `char grade = 'A';`.

Memory Tip: `char` → "character".

Q10) What is a symbolic (named) constant?

Answer: A symbolic constant is a name that represents a constant value, usually defined using `#define` or `const`, such as `#define PI 3.14`.

Memory Tip: Symbolic constant = "named fixed value".

Q11) What is an operator in C?

Answer: An operator is a symbol that tells the compiler to perform a specific operation on operands, such as `+`, `-`, `*`, `/`.

Memory Tip: Operator = "action symbol".

Q12) List the main types of operators in C.

Answer: Important operator types are arithmetic, relational, logical, assignment, unary, conditional (`?:`), bitwise, comma, and some other special operators.

Memory Tip: Remember "ARLAB C B" → Arithmetic, Relational, Logical, Assignment, Bitwise, Conditional, Comma, etc.

Q13) What is an arithmetic operator? Give two examples.

Answer: Arithmetic operators perform mathematical operations on numeric operands, for example `+` (addition) and `*` (multiplication).

Memory Tip: Same as math symbols.

Q14) What is a relational operator?

Answer: Relational operators compare two values and give a true or false result, such as `>`, `<`,

`==`, `!=`.

Memory Tip: Relational → "relationship between two values".

Q15) What are logical operators in C?

Answer: Logical operators combine or invert conditions and return true or false: `&&` (AND), `||` (OR), `!` (NOT).

Memory Tip: Use them for complex `if` conditions.

Q16) What is the assignment operator?

Answer: The basic assignment operator `=` assigns the value on the right side to the variable on the left side, like `a = 5;`.

Memory Tip: Think "LHS gets RHS".

Q17) What is the conditional (ternary) operator?

Answer: The conditional operator `?:` selects one of two expressions based on a condition, written as `condition ? expr1 : expr2;`.

Memory Tip: Short form of simple `if-else`.

Q18) What is an expression in C?

Answer: An expression is a combination of variables, constants, and operators that is evaluated to produce a value, like `a + b * c`.

Memory Tip: Expression → "computes one value".

Q19) What is a statement in C?

Answer: A statement is a complete instruction given to the compiler, usually ending with a semicolon, such as `x = x + 1;`.

Memory Tip: One statement = one complete command.

Q20) What are library functions? Give examples.

Answer: Library functions are pre-written functions provided by C libraries, like `printf()`, `scanf()`, `sqrt()`, and `strlen()`.

Memory Tip: "Ready-made" functions.

Q21) What is the role of the preprocessor in C?

Answer: The preprocessor handles directives beginning with `#` before compilation, such as

`#include` and `#define`.

Memory Tip: Preprocessor works "pre-compile".

Q22) What does `#include <stdio.h>` do?

Answer: It tells the preprocessor to include the standard input/output header file so that functions like `printf()` and `scanf()` can be used.

Memory Tip: Think: "include standard I/O definitions".

Q23) What does `getchar()` do?

Answer: `getchar()` reads a single character from standard input (usually the keyboard).

Memory Tip: "Get one char."

Q24) What does `putchar()` do?

Answer: `putchar()` writes a single character to standard output (usually the screen).

Memory Tip: "Put one char."

Q25) What is the purpose of `scanf()`?

Answer: `scanf()` is used to read formatted input from standard input into variables.

Memory Tip: `scanf` → "scan from keyboard".

Q26) What is the purpose of `printf()`?

Answer: `printf()` is used to display formatted output on the screen.

Memory Tip: `printf` → "print to screen".

Q27) Why are `gets()` and `puts()` used?

Answer: `gets()` reads an entire line of text including spaces into a string, and `puts()` prints a string followed by a newline. (Note: in modern C, `gets()` is considered unsafe.)

Memory Tip: `gets` = get string; `puts` = put string.

Q28) What are the basic parts of a C program?

Answer: A typical C program has preprocessor directives, global declarations (optional), the `main()` function, and possibly user-defined functions.

Memory Tip: "Headers + main + functions."

Q29) Why is `main()` important in C?

Answer: `main()` is the starting point of execution; program execution always begins from the

`main()` function.

Memory Tip: `main` = “entry point”.

Module 2: Control Structures

Q1) What is a control structure?

Answer: A control structure determines the flow of control in a program, deciding the order in which statements execute.

Memory Tip: Control structure = "controls flow".

Q2) What is the `if` statement used for?

Answer: The `if` statement allows execution of a statement or block only when a given condition is true.

Memory Tip: "If condition true, then do."

Q3) What is the difference between `if` and `if-else`?

Answer: `if` executes a block only when the condition is true, while `if-else` executes one block when the condition is true and another when it is false.

Memory Tip: `if-else` covers both true and false.

Q4) What is a nested `if`?

Answer: A nested `if` is an `if` statement used inside another `if` or `else` block to check multiple levels of conditions.

Memory Tip: Think "if inside if".

Q5) What is an else-if ladder?

Answer: The else-if ladder is a series of `if` and `else if` statements used to test multiple conditions one after another.

Memory Tip: Conditions checked "step by step" like a ladder.

Q6) When is a multiway decision needed?

Answer: Multiway decisions are needed when more than two alternatives must be selected, typically implemented using else-if ladder or `switch`.

Memory Tip: More than 2 choices → multiway.

Q7) Why are loops used in C?

Answer: Loops are used to execute a block of code repeatedly as long as a specified condition

remains true.

Memory Tip: Loop = “repeat until done”.

Q8) What is a `while` loop?

Answer: A `while` loop checks the condition first and then executes the loop body repeatedly while the condition is true.

Memory Tip: `while` = “check before execute”.

Q9) What is a `do-while` loop?

Answer: A `do-while` loop executes the body first and then checks the condition, ensuring the body runs at least once.

Memory Tip: `do-while` = “execute then check”.

Q10) What is a `for` loop?

Answer: The `for` loop combines initialization, condition, and increment/decrement in one line, and repeats the body while the condition is true.

Memory Tip: Good when number of iterations is known.

Q11) What is the `switch` statement?

Answer: `switch` is a multiway selection statement that chooses one case block to execute based on the value of an expression.

Memory Tip: `switch` = menu-style selection.

Q12) Why is `break` used inside a `switch`?

Answer: `break` is used to exit from the current `case` and prevent fall-through into the next case.

Memory Tip: `break` = “stop and get out”.

Q13) What does the `continue` statement do in a loop?

Answer: `continue` skips the remaining statements in the current iteration and jumps to the next iteration of the loop.

Memory Tip: “Skip this turn, go next.”

Q14) What is `goto`? Why is it discouraged?

Answer: `goto` transfers control unconditionally to a labeled statement, but it is discouraged because it makes programs hard to read and maintain.

Memory Tip: Avoid `goto` for cleaner code.

Q15) What is meant by nested control structures?

Answer: Nested control structures occur when one control statement (like `if`, loop, or `switch`) is placed inside another control statement.

Memory Tip: “Control inside control.”

Q1) What is a function in C?

Answer: A function is a self-contained block of code that performs a specific task and can be called from other parts of the program.

Memory Tip: Function = "mini-program".

Q2) Differentiate between library function and user-defined function.

Answer: Library functions are pre-compiled and provided by C libraries (like `printf()`), whereas user-defined functions are written by the programmer.

Memory Tip: Library = given; user-defined = created by you.

Q3) What is the role of `main()` function?

Answer: `main()` is a special function where execution starts and where the program usually ends.

Memory Tip: `main` is always the entry point.

Q4) What is a function prototype?

Answer: A function prototype is a declaration that tells the compiler the function name, return type, and parameter types before its use.

Memory Tip: Prototype = "function's declaration line".

Q6) What is the return type of a function?

Answer: The return type specifies the type of value a function will send back to the caller, such as `int`, `float`, or `void`.

Memory Tip: Return type = "type of answer".

Q7) What is recursion in C?

Answer: Recursion is when a function calls itself to solve a problem in smaller sub-problems.

Memory Tip: "Function calling itself."

Q8) When should recursion be avoided?

Answer: Recursion should be avoided when an iterative solution is simpler or when deep recursion may cause high memory use or stack overflow.

Memory Tip: Too many self-calls → risk of overflow.

Q9) What is a storage class?

Answer: A storage class defines the scope, lifetime, and visibility of a variable in a program.

Memory Tip: Storage class = "where, how long, and who can see the variable".

Q10) What is the default storage class for local variables?

Answer: The default storage class for local (block) variables is `auto`.

Memory Tip: Local variables are auto by default.

Q11) What is an `extern` variable?

Answer: An `extern` variable is a global variable defined in another file or part of the program and declared with `extern` to indicate external linkage.

Memory Tip: `extern` = "defined elsewhere".

Q12) What is a `static` variable inside a function?

Answer: A `static` local variable retains its value between function calls but is accessible only within that function.

Memory Tip: Static = "stays" for entire program.

Q13) What is a `register` variable?

Answer: A `register` variable requests the compiler to store it in a CPU register for faster access, usually for frequently used variables.

Memory Tip: Register = "try to keep in CPU".

Q14) How does the scope of global and local variables differ?

Answer: Local variables are accessible only within the block or function where they are declared, while global variables are accessible throughout all functions after their declaration.

Memory Tip: Local = limited; global = global view.

Module 4: Arrays, Strings, Structures

Q1) What is an array in C?

Answer: An array is a collection of elements of the same data type stored in contiguous memory locations.

Memory Tip: Array = "same type, side by side."

Q2) How do you declare a one-dimensional integer array of size 5?

Answer: It is declared as `int a[^5];`.

Memory Tip: Name[Size].

Q3) How is array indexing done in C?

Answer: Array indexing starts from 0, so the first element is at index 0 and the last is at index `size - 1`.

Memory Tip: First element = index 0.

Q4) What is a two-dimensional array?

Answer: A two-dimensional array is like a table with rows and columns, declared as an array of arrays.

Memory Tip: Think matrix (row, column).

Q5) How are arrays passed to functions?

Answer: Arrays are passed to functions by passing the address of their first element (effectively by reference).

Memory Tip: Array name acts as pointer to first element.

Q6) What is a string in C?

Answer: A string is a one-dimensional array of characters terminated by the null character `'\0'`.

Memory Tip: String = "char array + `\0` at end".

Q7) How is a string constant written in C?

Answer: A string constant is written in double quotes, for example `"Hello"`.

Memory Tip: Single quotes for char, double for string.

Q8) What is an array of strings?

Answer: An array of strings is a two-dimensional array of characters, where each row represents one string.

Memory Tip: "String table": each row is one word.

Q9) Name any three functions from `<string.h>`.

Answer: Common string functions include `strlen()`, `strcpy()`, `strcat()`, `strcmp()`.[1]

Memory Tip: Remember "len, copy, cat, cmp".

Q10) What does `strlen()` return?

Answer: `strlen()` returns the length of a string (number of characters before the null terminator).

Memory Tip: It does not count `'\0'`.

Q11) What is strcmp()?

Answer: `strcmp()` compares two strings lexicographically. It returns 0 if equal, positive or negative otherwise. Used in sorting and searching string operations.

Q12) What is a structure in C?

Answer: A structure is a user-defined data type that groups variables of different types under one name.

Memory Tip: Structure = "mixed-type bundle".

Q13) How do you declare a simple structure?

Answer: A structure is declared using the `struct` keyword, listing member variables inside braces and giving the structure a tag name.

Memory Tip: `struct` + members + `};`.

Q14) What is meant by initializing a structure?

Answer: Initializing a structure means giving initial values to its members at the time of declaration.

Memory Tip: Similar to array initialization but with different types.

Q15) What is a nested structure?

Answer: A nested structure is a structure that has another structure as a member.

Memory Tip: Structure inside structure.

Q16) What is an array of structures?

Answer: An array of structures is a collection of structure variables of the same type stored in contiguous memory.

Memory Tip: Like “table of records”.

Q17) How are structures passed to functions?

Answer: Structures can be passed to functions by value (copy of the structure) or by reference using pointers to structures.

Memory Tip: Use pointer to avoid copying large structures.

Module 5: Pointers

Q1) What is a pointer in C?

Answer: A pointer is a variable that stores the memory address of another variable.

Memory Tip: Pointer = "address holder".

Q2) What is the address operator?

Answer: The address operator `&` gives the address of a variable in memory.

Memory Tip: `&x` → "where x lives".

Q3) What is a pointer variable?

Answer: A pointer variable is declared to store an address, for example `int *p;` stores the address of an integer variable.

Memory Tip: `*` in declaration → "pointer to type".

Q4) What is dereferencing (indirection) in pointers?

Answer: Dereferencing a pointer using `*` gives access to the value stored at the address held by the pointer.

Memory Tip: `*p` → "value at p".

Q5) What is a `void` pointer?

Answer: A `void *` pointer is a generic pointer that can hold the address of any data type but must be cast to a specific type before dereferencing.

Memory Tip: Void pointer = "type-less address".

Q6) What is pointer arithmetic?

Answer: Pointer arithmetic means adding or subtracting integers from pointers, which actually moves the pointer by multiples of the size of the pointed data type.

Memory Tip: Incrementing pointer moves to next element, not by 1 byte always.

Q7) What is a pointer to a pointer?

Answer: A pointer to a pointer is a variable that stores the address of another pointer, declared like `int **pp;`

Memory Tip: Two stars = pointer to pointer.

Q8) How are pointers and arrays related?

Answer: The name of an array acts as a pointer to its first element, so pointer notation and array indexing can often be used interchangeably.

Memory Tip: `a[i]` is same as `*(a + i)`.

Q9) What is a NULL pointer?

Answer: A NULL pointer points to nothing. It is used for safety in programs. Helps avoid accessing invalid memory.

Module 6: Files

Q1) What is a file in C programming?

Answer: A file is a named area on storage (like disk) used to store data permanently beyond program execution.

Memory Tip: File = "permanent data store".

Q2) What is a `FILE *` pointer?

Answer: `FILE *` is a pointer to a structure that holds information about an open file, used with standard file I/O functions.

Memory Tip: `FILE *` = "handle to file".

Q3) What does `fopen()` do?

Answer: `fopen()` opens a file and returns a `FILE *` pointer if successful or `NULL` if the open fails.

Memory Tip: Always check if `fopen` returned `NULL`.

Q4) Name any three common file opening modes.

Answer: Examples are `"r"` (read), `"w"` (write, truncate or create), and `"a"` (append).

Memory Tip: r = read, w = write new, a = add at end.

Q5) Why is `fclose()` used?

Answer: `fclose()` closes an open file, releases associated resources, and ensures data is properly written to disk.

Memory Tip: Always close what you open.

Q6) Which functions are used for character-wise file I/O?

Answer: `fgetc()` reads a character from a file and `fputc()` writes a character to a file.

Memory Tip: `get` for read, `put` for write.

Q7) Which functions are used for string-wise file I/O?

Answer: `fgets()` reads a string (line) from a file, and `fputs()` writes a string to a file.

Memory Tip: `gets`/`puts` for strings in files.

Q8) What is the difference between text file and binary file?

Answer: A text file stores data as human-readable characters with possible conversions, while

a binary file stores data in raw binary form without translation.

Memory Tip: Text → readable; binary → raw bytes.

Q9) What is meant by processing a file?

Answer: Processing a file means performing operations such as reading, writing, updating, searching, or counting data stored in the file.

Memory Tip: Think of file as a "data source" for tasks like reports, logs, etc.

Q10) What is fprintf()?

Answer: fprintf() writes formatted output to a file. Works similar to printf but writes to file instead of screen. Helps store structured data.

Q11) What is fscanf()?

Answer: fscanf() reads formatted input from a file. Similar to scanf. Used for reading numbers, strings, and characters.
