



KAMALDNP

kamaldnp.com • IT Training, Varanasi

PDF created by Raj Kamal using AI

Practice Question Paper (Theory Part) — Computer Programming Using C

BTEUP Diploma in Computer Science & Engineering • 3rd Semester • Session 2026-27
Session 2026-27

Practice Questions / Model Paper — NOT a BTEUP board paper

Time: 2.30 Hours (suggested practice time) Practice Score: 50 (self-check only)

General Instructions:

- All questions are compulsory. Each question has three parts (a), (b) and (c) — **answer any two** of the three parts in every question.
- Each part carries 5 marks; each question carries 10 marks.
- Draw neat, labelled diagrams wherever they are asked or wherever they support your answer.
- Write your answers in your own words, in the same order as the questions.

About this paper: In the verified BTEUP 2026-27 scheme this subject is a PRACTICUM (100 marks — lecture and practical work together); it has no separate board theory paper. This practice paper is KAMALDNP Academy's own question set for the theory (lecture) part of the practicum, written from the official syllabus topics, so you can test your concepts for the practicum assessment and viva. The marks shown (10 per question, 50 in all) are a practice weighting for self-checking only — they are NOT board marks. This is NOT a BTEUP board paper, and no question in it is promised to appear anywhere.

Q1. (From: Unit 1 — Introduction to Problem Solving) — Answer any two of the following:

- (a) What is meant by the computational way of thinking? Write an algorithm and draw a flowchart to check whether a given number is even or odd. (5)
- (b) Differentiate between a variable and a constant. Define an identifier and a keyword, and state the rules for naming an identifier. (5)
- (c) Explain the primary data types used in C — int, float, double, char and void — with one declaration example of each. (5)

Q2. (From: Unit 2 — Operators & Expression) — Answer any two of the following:

- (a) Explain arithmetic, relational, logical and bitwise operators in C, with one example of each. Also write the values of $5 \& 3$, $5 | 3$ and $5 \wedge 3$. (5)
- (b) Predict the output of this program and explain each value. Also state what type casting means. Program: `#include <stdio.h> / int main() { int a = 7, b = 3; float c; c = (float)a / b; printf("%d %d %.2f", a / b, a % b, c); return 0; }` (5)
- (c) Explain formatted input and output in C using printf and scanf. Then evaluate the expression $5 + 2 * 3 - 4 / 2$ step by step, stating the rule you used. (5)

Q3. (From: Unit 3 — Conditional & Looping Statements) — Answer any two of the following:

- (a) Write a C program to find the largest of three numbers entered by the user, using an if-else ladder. (5)
- (b) Write a C program to check whether a number entered by the user is prime or not. Use a loop and a break statement in your program. (5)
- (c) Differentiate between a while loop and a do-while loop. Then predict the output of each snippet. Snippet 1: `int i = 5; while (i < 3) { printf("%d ", i); i++; } printf("End");` Snippet 2: `int i = 5; do { printf("%d ", i); i++; } while (i < 3); printf("End");` (5)

Q4. (From: Unit 4 — Arrays & String) — Answer any two of the following:

- (a) What is an array? Write a C program to read 5 numbers into an array and print their sum and average. (5)
- (b) Explain a two-dimensional array with an example declaration. Write the logic, using nested loops, to store values in a 2 x 3 integer matrix and display it row by row. (5)
- (c) What is a string in C and how is it declared? Explain the string functions strlen, strcpy and strcmp. Also write the logic to print a given string in reverse order. (5)

Q5. (From: Unit 5 — Function & Pointer) — Answer any two of the following:

- (a) What is a function in C? Explain function declaration, function call and return, and the types of functions. Also differentiate between local and global variables. (5)
- (b) Write a C program to swap the values of two variables using a function and pointers (call by reference). Differentiate between call by value and call by reference. (5)
- (c) What is a recursive function? Write a recursive C function to find the factorial of a number and trace its working for the input 4. (5)

Model Answer Outline — Computer Programming Using C

The key points a full-marks answer contains, question by question.

Practice Questions / Model Paper — NOT a BTEUP board paper

Q1(a)

- Computational thinking means breaking a problem into clear, ordered steps that a computer can follow: understand the input, the processing and the required output before writing code.
- Algorithm (in words or pseudocode): Step 1 Start; Step 2 Read a number n ; Step 3 If $n \% 2 == 0$ print 'Even', otherwise print 'Odd'; Step 4 Stop.
- Flowchart symbols to use in the drawing: oval for Start/Stop, parallelogram for reading n and printing the result, and a diamond for the decision $n \% 2 == 0$ with Yes/No paths.
- Flow of the chart: Start, then input, then the decision diamond; the Yes path prints Even and the No path prints Odd, and both paths join and go to Stop.
- An algorithm is the step list in plain language, while a flowchart is the same logic drawn with standard symbols — state this difference in one line.

Q1(b)

- Variable: a named memory location whose value can change during the program, for example `int age = 17`; Constants: values that do not change, written as literals or declared with `const`, for example `const float PI = 3.14`;
- Identifier: the name given by the programmer to a variable, constant or function. Rules: it may contain letters, digits and underscore, must not start with a digit, and must not be a keyword.
- Keywords are reserved words with fixed meaning in C, such as `int`, `float`, `if`, `for` and `return`; they cannot be used as identifiers.
- Naming examples to give: `total_marks` and `roll2` are valid identifiers; `2roll` is invalid (starts with a digit) and `int` cannot be used as a name (keyword).
- Case matters in C: `Marks` and `marks` are two different identifiers.

Q1(c)

- A data type tells the compiler what kind of value a variable holds, how much memory it needs and which operations are allowed on it.
- Primary types with one example each: `int` for whole numbers (`int count = 10`), `float` for decimal values (`float price = 99.5`), `double` for decimal values needing more precision, `char` for one character (`char grade = 'A'`).
- `void` means no value and is used, for example, for a function that returns nothing.

- Representation point: an int is stored as a binary whole number, a float/double in floating-point form, and a char as the numeric code of the character.
- Derived types such as arrays are built from these primary types and are covered in Unit 4 — one line is enough here.

Q2(a)

- Arithmetic operators (+, -, *, /, %) do calculations; % gives the remainder of integer division, for example $7 \% 3 = 1$.
- Relational operators (>, <, >=, <=, ==, !=) compare two values and give a true (1) or false (0) result, and are used to build conditions.
- Logical operators (&& AND, || OR, ! NOT) combine or reverse conditions; for example (marks >= 33 && marks <= 100) is true only when both parts are true.
- Bitwise operators work bit by bit on integer values: & (AND), | (OR), ^ (XOR), ~ (complement), << (shift left) and >> (shift right).
- Worked values to state: for 5 (binary 101) and 3 (binary 011): $5 \& 3 = 1$, $5 | 3 = 7$ and $5 \wedge 3 = 6$.

Q2(b)

- a / b with both int does integer division: $7 / 3 = 2$, the decimal part is discarded. This is why casting is needed for a decimal result.
- $a \% b$ gives the remainder of 7 divided by 3, which is 1.
- (float)a / b first converts 7 to 7.0, so the division is $7.0 / 3 = 2.3333\dots$, printed with %.2f as 2.33.
- Type casting means changing a value from one data type to another, written as (type) value; it does not change the original variable, only the value used in that expression.
- Final output line exactly: 2 1 2.33

Q2(c)

- printf displays formatted output: format specifiers such as %d for int, %f for float and %c for char are replaced by the values listed after the string.
- scanf takes formatted input from the keyboard: scanf('%d', &age); — the & gives the address of the variable where the value is stored, and forgetting it is a common error.
- Expression evaluation follows precedence: multiplication, division and remainder are done before addition and subtraction, and brackets are done first.
- Worked value: $5 + 2 * 3 - 4 / 2 = 5 + 6 - 2 = 9$.
- Assignment check to state: int x = 10; x += 3 makes x = 13, then x *= 2 makes x = 26.

Q3(a)

- Logic: read a, b and c; use an if-else ladder comparing them in turn, and print the one that is not smaller than the other two.
- Essential code: if (a >= b && a >= c) largest = a; else if (b >= a && b >= c) largest = b; else largest = c; then print largest. Using >= keeps the answer correct when two numbers are equal.

- Complete program frame: `#include <stdio.h>`, `int main()`, `scanf` for the three values, the ladder above, `printf` for the result, `return 0`.
- State the && condition idea: a is largest only when it is greater than or equal to both other numbers.
- Sample run to quote: for inputs 12, 25 and 9, the output is that 25 is the largest number.

Q3(b)

- Logic: a number n greater than 1 is prime if no number from 2 to $n/2$ (or $n-1$ in the simplest version) divides it exactly; use a flag variable to record whether a divisor was found.
- Essential code: `for (i = 2; i <= n / 2; i++) { if (n % i == 0) { flag = 1; break; } }` and then print Prime if flag is still 0, otherwise Not Prime. $n = 0$ and $n = 1$ are handled as not prime.
- `break` ends the loop as soon as one divisor is found, because the answer is already decided; without it the loop wastefully checks all remaining values.
- Sample runs to quote: 7 gives Prime (no divisor found), 9 gives Not Prime (divisor 3 found, loop breaks).
- Edge care: start the flag at 0 for every new number tested.

Q3(c)

- `while` is an entry-controlled loop: the condition is tested first, so the body may not run even once. `do-while` is exit-controlled: the body runs first and the condition is tested after, so the body runs at least once.
- First snippet trace: i is 5 and the condition $i < 3$ is false at the very first test, so nothing inside the loop prints; only the final `printf` runs.
- First snippet output exactly: End
- Second snippet trace: the body runs once before any test, printing 5 and making $i = 6$; the condition $i < 3$ is then false, so the loop ends and the final `printf` runs.
- Second snippet output exactly: 5 End

Q4(a)

- An array stores many values of the same data type under one name, in consecutive memory locations, reached by an index starting from 0.
- Declaration and reading: `int a[5]`; then a `for` loop with `scanf` to fill `a[0]` to `a[4]` — the loop is the standard way to read or print all elements.
- Logic: keep a running total in a loop, `sum = sum + a[i]`; print sum, and average as `(float)sum / 5` so the division is not integer division.
- Sample check to quote: for values 10, 20, 30, 40, 50 the sum is 150 and the average is 30.00 (printed with `%.2f`) or 30.0.
- Size point: the number in the declaration is the length of the array; valid indexes here are 0 to 4, and using index 5 goes outside the array.

Q4(b)

- A two-dimensional array is an array of arrays, understood as a table of rows and columns; declaration `int m[2][3]`; means 2 rows and 3 columns, 6 elements in total.

- Element access uses two indexes, row first then column: `m[0][0]` is the first element and `m[1][2]` is the last element of this array.
- Display logic: nested loops — the outer loop takes one row at a time and the inner loop prints each element of that row, followed by a new line after each row.
- The same nested-loop pattern is used for reading the matrix and for row-wise jobs such as finding each row's sum.
- State storage briefly: elements are stored in memory row by row, even though we picture them as a table.

Q4(c)

- A string in C is a one-dimensional array of characters ending with the null character `'\0'`, which marks the end; declaration example: `char name[20] = "Student";` needs room for 7 letters plus the `'\0'`.
- `strlen(s)` gives the length of the string without counting `'\0'` — for 'Student' it gives 7. `strcpy(dest, src)` copies `src` into `dest`. `strcmp(a, b)` compares two strings and returns 0 when they are equal, a negative value when `a` comes before `b`, and a positive value when `a` comes after `b`.
- Reverse logic without a library function: find the length, then run a loop from the last index down to 0 and print one character at a time; the stored string itself is not changed by printing it in reverse.
- Include `<string.h>` before using the library string functions.
- Common error to warn against: comparing strings with `==` does not compare their contents — `strcmp` must be used.

Q5(a)

- A function is a self-contained block that performs one task; it is written once and can be called many times, which keeps `main()` short and avoids repeating code.
- Parts to name: function declaration (prototype, which states the name, return type and parameters), function call (in the calling code, with actual values), and function definition with a return statement sending the result back.
- Types by return value and arguments: no return and no arguments; no return with arguments; return value with no arguments; return value with arguments. One short example category statement for each type earns the marks.
- Local variables are declared inside a function, exist only while it runs and cannot be used by other functions; global variables are declared outside all functions and can be used and changed by every function in the file.
- Standard (library) functions such as `printf`, `scanf` and `strlen` are ready-made functions supplied with C.

Q5(b)

- Call by value copies the value into the function, so changes inside do not affect the original variables; an ordinary swap written this way fails to swap the caller's variables.
- Call by reference passes the addresses of the variables, so the function works on the originals through pointers and the swap lasts after the function returns.

- Essential code: `void swap(int *a, int *b) { int temp = *a; *a = *b; *b = temp; }` called as `swap(&x, &y);`
- In the function, `*a` means the value stored at the address held in `a` — this line is the heart of the answer.
- Sample run to quote: before the call `x = 4` and `y = 9`; after the call `x = 9` and `y = 4`.

Q5(c)

- A recursive function calls itself with a smaller part of the same problem, and must have a base (stopping) case, otherwise it calls itself for ever.
- Essential code: `int fact(int n) { if (n == 0 || n == 1) return 1; else return n * fact(n - 1); }` and `main()` reads `n`, calls `fact(n)` and prints the returned value.
- Trace for `n = 4`: `fact(4) = 4 * fact(3) = 4 * 3 * fact(2) = 4 * 3 * 2 * fact(1) = 4 * 3 * 2 * 1 = 24`.
- Output to state: the factorial of 4 printed by the program is 24.
- Each call keeps its own copy of `n` on the stack; the results combine as the calls return, from `fact(1)` back up to `fact(4)`.

KAMALDNP • kamaldnp.com • support@kamaldnp.com • +91 8318013442 • Varanasi, UP
Prepared by KAMALDNP Academy from the official BTEUP 2026-27 syllabus. This is a practice/model paper only.