CBCS SCHEME

USN | | | | | | | | | | |

BPOPS103/203

First/Second Semester B.E./B.Tech. Degree Examination, June/July 2025
Principles of Programming using C

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	Explain the components of a computer with a neat diagram.	06	L2	CO1
	b.	Explain the formatted input and output statements with suitable syntax and example.	06	L2	CO2
	c.	Illustrate an algorithm, flowchart and program to compute area of a circle.	08	L2	CO2
		OR			
Q.2	a.	What is variable? What are the rules to construct a variable?	06	L2	CO2
	b.	Identify the following as valid or invalid identifiers with justification: i) int ii) num2 iii) +add iv) a – 2 v) – sum vi) name _ _123	06	L2	CO2
	c.	Explain the structure of C program. Write a sample program to demonstrate the components in the structure of C program.	08	L2	CO2
		Module – 2			
Q.3	a.	Discuss the functioning of the following operators with example. i) Arithmetic ii) Relational	06	L2	CO2
	b.	Compare between the break and continue statement.	06	L2	CO2
	c.	Explain switch statement with syntax. Write a program to simulate simple calculator.	08	L2	CO2
		OR			
Q.4	a.	Explain for loop with syntax and example.	06	L2	CO2
	b.	Explain with syntax, if and if-else statements in C program.	06	L2	CO2
	c.	Develop a C program that takes three coefficients (a, b and c) of quadratic equation ($ax^2 + bx + c$) as input and compute all possible roots and print them with appropriate messages.	08	L3	CO5
		Module – 3			
Q.5	a.	Explain the syntax of function declaration and function definition with example.	06	L2	CO4
	b.	Write a C program to swap two integers using call by value method of passing arguments to a function.	06	L2	CO3
	c.	Explain different types of storage classes with example.	08	L2	CO3

		OR			
Q.6	a.	Explain the declaration and initialization of 1D and 2D arrays with example.	06	L2	CO3
	b.	Illustrate the concept of recursive function with example.	06	L2	CO4
	c.	Write a C program to implement Bubble sort technique (ascending order).	08	L2	CO3
		Module – 4			
Q.7	a.	Write the operations that can be performed on string using built-in functions. Explain any two functions.	08	L2	CO4
	b.	Develop a C program to concatenate 2 strings without using built-in function.	06	L3	CO5
	c.	Explain array of strings with an example.	06	L3	CO5
		OR			
Q.8	a.	Develop a program using pointer to compute the sum, mean and standard deviation of all elements stored in array of N real numbers.	08	L3	CO5
	b.	Describe the pointer concept such as initialization, declaration with suitable program example.	06	L3	CO3
	c.	Explain gets( ) and puts( ) function with example.	06	L2	CO1
		Module – 5			
Q.9	a.	What is structure? Explain the C syntax of structure declaration with example.	06	L2	CO3
	b.	Compare structures and unions with syntax and example.	06	L2	CO3
	c.	Implement structures to read, write and compute average-marks of the students. List the students scoring above and below the average marks for a class of N students.	08	L3	CO5
		OR			
Q.10	a.	Discuss the different modes of operation on files with suitable example.	08	L2	CO4
	b.	Compare between fgets( ) and gets( ).	06	L2	CO4
	c.	Discuss the enumerated data type. Explain the declaration and access of enumerated data type with a code in C.	06	L2	CO1

* * * * *

1a. Explain the components of a computer with a neat diagram.

Ans: **Components of a Computer System**

A computer is an electronic device that processes data and gives output with the help of various interconnected components. The main components are:

---

1. Input Unit

- The input unit is responsible for **accepting data and instructions** from the user.
- Examples: **Keyboard** (for text input), **Mouse** (for pointing and selection), **Scanner**, **Microphone**, etc.
- It converts user input into **machine-readable form (binary form)** and sends it to the CPU.

---

2. Central Processing Unit (CPU) – *"Brain of the Computer"*

- The CPU controls and executes all instructions.
- It consists of three sub-units:
    1. **Control Unit (CU):**
        - Directs the flow of data between memory, input, output, and ALU.
        - Acts like a **traffic police** of the computer.
    2. **Arithmetic and Logic Unit (ALU):**
        - Performs all **mathematical operations** (addition, subtraction, multiplication, division).
        - Handles **logical operations** (comparisons like >, <, =).
    3. **Registers:**
        - Small, high-speed storage locations inside the CPU.
        - Temporarily store instructions, data, and intermediate results.

---

3. Memory Unit

- Stores data, instructions, and results temporarily or permanently.
- Types:
    - **Primary Memory:**
        - **RAM (Random Access Memory):** Volatile, stores data temporarily during execution.
        - **ROM (Read Only Memory):** Permanent memory, stores system programs (firmware/BIOS).
    - **Secondary Memory:**
        - Non-volatile storage like **Hard Disk, SSD, Pen drive, CD/DVD** for permanent data storage.

---

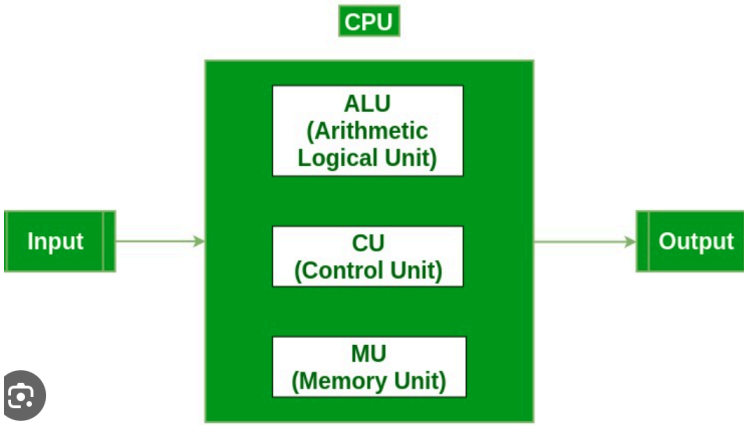
4. Output Unit

- Converts processed data from the computer into a **human-understandable form**.
- Examples: **Monitor** (visual output), **Printer** (hard copy), **Speakers** (audio output).

---

5. Storage Unit

- Used to **store large amounts of data** permanently.
- Works as a **warehouse of data and software**.
- Examples: Hard disks, SSDs, Optical disks, Cloud storage.



1b. Explain the declaration and initialization of 1D and 2D arrays with example.

Ans: **Declaration and Initialization of Arrays**

An **array** is a collection of elements of the **same data type**, stored in **contiguous memory locations**.
There are two types commonly used: **1D array** and **2D array**.

---

1. One-Dimensional (1D) Array

Declaration
data_type array_name[size];

- data_type → type of elements (int, float, char, etc.)
- array_name → name given to array
- size → number of elements

Example:

```
int marks[5];   // declares an integer array with 5 elements
```

Initialization

You can assign values in two ways:

1. At the time of declaration:

```
int marks[5] = {90, 85, 78, 92, 88};
```

2. Partial initialization (remaining elements become 0):

```
int marks[5] = {90, 85};   // means {90, 85, 0, 0, 0}
```

3. Automatic size detection:

```
int marks[] = {90, 85, 78, 92, 88}; // compiler counts size = 5
```

Example Program (1D Array)

```
#include <stdio.h>
int main() {
    int marks[5] = {90, 85, 78, 92, 88};
    for (int i = 0; i < 5; i++) {
        printf("marks[%d] = %d\n", i, marks[i]);
    }
    return 0;
}
```

---

2. Two-Dimensional (2D) Array

A 2D array is like a **matrix (rows × columns)**.

Declaration

```
data_type array_name[rows][columns];
```

Example:

```
int matrix[3][3];   // declares a 3x3 integer matrix
```

Initialization

1. Row-wise initialization:

```
int matrix[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
```

2. Continuous initialization:

```
int matrix[2][3] = {1, 2, 3, 4, 5, 6};
// stored row by row
```

Example Program (2D Array)

```
#include <stdio.h>
int main() {
    int matrix[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };
```

```
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
1 2 3
4 5 6
```

1c. Write a C program to implement Bubble sort technique (ascending order).

Ans: **Bubble Sort in C (Ascending Order)**

Definition

Bubble sort is a simple **comparison-based sorting technique** where each pair of adjacent elements is compared, and they are swapped if they are in the wrong order. This process continues until the array is sorted.

Working Principle

- Repeatedly traverse the array.
- After each pass, the **largest element bubbles up** to the correct position.
- Continue until no swaps are needed.

Algorithm (Bubble Sort – Ascending)

1. Repeat for i = 0 to n-1
    - For j = 0 to n-i-2
        - If arr[j] > arr[j+1], swap them.
2. End

C Program

```c
#include <stdio.h>

int main() {
    int n, i, j, temp;

    // Input size of array
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter %d elements: ", n);
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Bubble Sort
    for(i = 0; i < n-1; i++) {
        for(j = 0; j < n-i-1; j++) {
            if(arr[j] > arr[j+1]) {
                // Swap
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }

    // Print sorted array
    printf("Sorted array in ascending order:\n");
    for(i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

Sample Input/Output

Input:

Enter number of elements: 5
Enter 5 elements: 64 25 12 22 11

Output:

Sorted array in ascending order:
11 12 22 25 64

2a) What is variable? What are the rules to construct a variable? Answer:

A **variable** in C is a named memory location that stores data which can be changed during program execution. It is used to manipulate and process data in a program.

Rules for constructing variables:

1. Must begin with a **letter or underscore**, not a digit.
2. Can contain **letters, digits, and underscores** only.
3. **No special characters** or spaces allowed.
4. **Keywords** cannot be used as variable names.
5. Variable names are **case-sensitive**.
6. Length may vary, but it should be meaningful.

Example:

int age, marks, total_sum;

2b) Identify the following as valid or invalid identifiers with justification

i) **int** → Invalid → Keyword, cannot be identifier.

ii) **num2** → Valid → Letters + digits allowed, starts with letter.

iii) **+add** → Invalid → Cannot start with special character +.

iv) **a – 2** → Invalid → Contains operator –, spaces not allowed.

v) **–sum** → Invalid → Identifier cannot start with a minus sign.

vi) **name__123** → Valid → Letters, digits, underscores → allowed.

2c) Explain the structure of C program. Write a sample program to demonstrate the components in the structure of C program. (8 Marks)

Answer:

A C program follows a specific structure consisting of different sections:

1. **Documentation Section** – Comments describing the program.
2. **Preprocessor Directives** – Header files like #include <stdio.h>.

3. **Global Declarations** – Global variables, constants, functions.
4. **main() Function** –
    - **Declaration Part:** Variable declaration.
    - **Executable Part:** Program statements.
5. **User-Defined Functions (optional)** – Defined outside main().

Sample C Program

```
#include <stdio.h>   // Preprocessor directive

// Function declaration
int add(int, int);

int main() {
   int a, b, sum;   // Declaration part

   // Input
   printf("Enter two numbers: ");
   scanf("%d %d", &a, &b);

   // Function call
   sum = add(a, b);

   // Output
   printf("Sum = %d\n", sum);

   return 0;   // End of program
}

// Function definition
int add(int x, int y) {
   return x + y;
}
```

3a) Discuss the functioning of the following operators with example (6 Marks)

i) Arithmetic Operators

- Used to perform **basic mathematical operations**.
- Operators: +, -, *, /, %
- Example:

```
#include <stdio.h>
int main() {
    int a = 10, b = 3;
    printf("a+b = %d\n", a+b);  // 13
    printf("a-b = %d\n", a-b);  // 7
    printf("a*b = %d\n", a*b);  // 30
    printf("a/b = %d\n", a/b);  // 3 (integer division)
    printf("a%%b = %d\n", a%b); // 1 (remainder)
    return 0;
}
```

---

ii) Relational Operators

- Used to **compare two values**.
- Operators: ==, !=, >, <, >=, <=
- Result is either **true (1)** or **false (0)**.
- Example:

```
#include <stdio.h>
int main() {
    int a = 5, b = 10;
    printf("a == b : %d\n", a == b);  // 0
    printf("a != b : %d\n", a != b);  // 1
    printf("a > b  : %d\n", a > b);   // 0
    printf("a < b  : %d\n", a < b);   // 1
    return 0;
}
```

3**b) Compare between the break and continue statement (6 Marks)**

Answer:

Both break and continue are **jump statements** in C, but they behave differently inside loops.

Comparison Table

Feature	break	continue
Definition	Terminates the loop immediately and control moves to the statement after the loop.	Skips the remaining code in the current iteration and jumps to the next iteration.
Effect on Loop	Ends the loop completely.	Loop continues but skips only one iteration.

Common Use	When you want to **exit from the loop** early.	When you want to **skip certain iterations** but still continue with the loop.

Example

```
for(int i=1; i<=5; i++) {
    if(i==3) break;
    printf("%d ", i);
}
// Output: 1 2
``` |
```c
for(int i=1; i<=5; i++) {
    if(i==3) continue;
    printf("%d ", i);
}
// Output: 1 2 4 5
```

3c. Write a C program for a simple calculator using switch–case

Ans:

```
#include <stdio.h>

int main() {
    char op;
    double num1, num2, result;

    // Step 1: Input operator
    printf("Enter an operator (+, -, *, /): ");
    scanf(" %c", &op);

    // Step 2: Input two numbers
    printf("Enter two numbers: ");
    scanf("%lf %lf", &num1, &num2);

    // Step 3: Switch-case for operations
    switch(op) {
        case '+':
            result = num1 + num2;
            printf("Result = %.2lf\n", result);
            break;

        case '-':
            result = num1 - num2;
            printf("Result = %.2lf\n", result);
            break;

        case '*':
            result = num1 * num2;
            printf("Result = %.2lf\n", result);
            break;
```

```
        case '/':
            if(num2 != 0)
                result = num1 / num2;
            else {
                printf("Error! Division by zero not allowed.\n");
                return 0;
            }
            printf("Result = %.2lf\n", result);
            break;

        default:
            printf("Invalid operator!\n");
    }

    return 0;
}
```

4**a. Explain for loop with syntax and example. (06 Marks)**

- **Explanation:**
  The for loop in C is an entry-controlled loop that allows repeated execution of a block of statements for a fixed number of times. It is commonly used when the number of iterations is known in advance.

- **Syntax:**

```
for(initialization; condition; increment/decrement) {
    // loop body
}
```

- **Example:**

```
#include <stdio.h>
int main() {
    int i;
    for(i = 1; i <= 5; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```

Output:

```
1
2
3
```

4
5

4**b. Explain with syntax, if and if-else statements in C program. (06 Marks)**

- **if Statement:**
  The if statement is used to execute a block of code only if a given condition is true.

Syntax:

```
if(condition) {
   // statements if condition is true
}
```

Example:

```
if(a > b) {
   printf("a is greater than b");
}
```

- **if-else Statement:**
  The if-else statement executes one block of code if the condition is true, and another block if the condition is false.

Syntax:

```
if(condition) {
   // statements if condition is true
} else {
   // statements if condition is false
}
```

Example:

```
if(a % 2 == 0) {
   printf("Even number");
} else {
   printf("Odd number");
}
```

4**c. C program to compute roots of quadratic equation (08 Marks)**

```
#include <stdio.h>
#include <math.h>

int main() {
   float a, b, c, d, root1, root2, realPart, imagPart;
```

```
    printf("Enter coefficients a, b and c: ");
    scanf("%f %f %f", &a, &b, &c);

    d = b*b - 4*a*c;  // discriminant

    if(d > 0) {
        root1 = (-b + sqrt(d)) / (2*a);
        root2 = (-b - sqrt(d)) / (2*a);
        printf("Roots are real and different.\n");
        printf("Root1 = %.2f and Root2 = %.2f\n", root1, root2);
    }
    else if(d == 0) {
        root1 = root2 = -b / (2*a);
        printf("Roots are real and equal.\n");
        printf("Root1 = Root2 = %.2f\n", root1);
    }
    else {
        realPart = -b / (2*a);
        imagPart = sqrt(-d) / (2*a);
        printf("Roots are complex and different.\n");
        printf("Root1 = %.2f + %.2fi and Root2 = %.2f - %.2fi\n",
               realPart, imagPart, realPart, imagPart);
    }

    return 0;
}
```

Q.5 (a) Explain the syntax of function declaration and function definition with example. (06 marks)

Function Declaration (Prototype):

It tells the compiler about the function's name, return type, and parameters (but no body).

```
return_type function_name(parameter_list);
```

Example:

```
int add(int, int);
```

-

Function Definition:

It contains the actual body of the function (logic).

```
return_type function_name(parameter_list) {
    // function body
}
```

Example:

```
int add(int a, int b) {
    return a + b;
```

}

-

Complete Example:

```
 #include <stdio.h>

// Function declaration
int add(int, int);

int main() {
    int x = 10, y = 20, sum;
    sum = add(x, y);  // function call
    printf("Sum = %d\n", sum);
    return 0;
}

// Function definition
int add(int a, int b) {
    return a + b;
}
```

Q.5 (b) Write a C program to swap two integers using call by value method. (06 marks)

- In **call by value**, a copy of the actual arguments is passed to the function. Changes inside the function do **not** affect the original values.

```
#include <stdio.h>



// Function to swap (call by value)

void swap(int a, int b) {

    int temp = a;

    a = b;

    b = temp;

    printf("Inside function: a = %d, b = %d\n", a, b);

}
```

```
int main() {
    int x = 10, y = 20;
    printf("Before swap: x = %d, y = %d\n", x, y);

    swap(x, y);  // call by value

    printf("After swap: x = %d, y = %d\n", x, y); // No change
    return 0;
}
```

Output:

Before swap: x = 10, y = 20

Inside function: a = 20, b = 10

- After swap: x = 10, y = 20

Q.5 (c) Explain different types of storage classes with example. (08 marks)

Storage classes define **scope, visibility, lifetime, and default value** of variables.

1. **auto (default for local variables):**
    - Scope: Local to the block
    - Lifetime: Till the block ends
    - Default value: Garbage
      Example:

```
void main() {
    auto int a = 10;
    printf("%d", a);
}
```

2.
3. **register:**
    - Stored in CPU register (faster access)
    - Scope: Local
    - Cannot take address using &
      Example:

```
void main() {
   register int i;
   for(i=0; i<5; i++)
      printf("%d ", i);
}
```

4.
5. **static:**
    - Retains value between function calls
    - Scope: Local to function / file
    - Default value: 0
      Example:

```
void fun() {
   static int x = 0;
   x++;
   printf("%d ", x);
}

void main() {
   fun(); fun(); fun(); // Output: 1 2 3
}
```

6.
7. **extern:**

- Declares a global variable defined in another file or before main()
- Used for sharing variables between files
  Example:

```
int a = 10;   // global variable


void fun() {
   extern int a;
   printf("%d", a);
}
```

Q.6 (a) Explain the declaration and initialization of 1D and 2D arrays with example.

Answer:

- **1D Array**: A one-dimensional array is a list of elements stored in contiguous memory locations.

Declaration:

```
int arr[5];   // declares an integer array of size 5
```

-

Initialization:

```
int arr[5] = {10, 20, 30, 40, 50};
```

-
- Here, arr[0] = 10, arr[1] = 20, etc.

- **2D Array**: A two-dimensional array is like a table with rows and columns.

Declaration:

```
int mat[2][3];   // declares a 2D array with 2 rows and 3 columns
```

-

Initialization:

```
int mat[2][3] = { {1, 2, 3}, {4, 5, 6} };
```

-
- This stores numbers in a 2 × 3 matrix.

Q.6 (b) Illustrate the concept of recursive function with example.

Answer:

- **Recursive Function:** A function that calls itself until a base condition is met.

Example: Factorial using Recursion

```
#include <stdio.h>


int factorial(int n) {
    if (n == 0 || n == 1)   // base condition
        return 1;
    else
        return n * factorial(n - 1); // recursive call
}


int main() {
    int num = 5;
    printf("Factorial of %d = %d", num, factorial(num));
    return 0;
}
```

-
- **Explanation:**
  factorial(5) → 5 * factorial(4) → 5 * 4 * factorial(3) … until base case factorial(1) is reached.

Q.6 (c) Write a C program to implement Bubble sort technique (ascending order).

Answer:

```
#include <stdio.h>


int main() {
    int arr[5] = {64, 25, 12, 22, 11};
    int n = 5, i, j, temp;


    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // swap
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }


    printf("Sorted array: ");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);


    return 0;
}
```

Output:

Sorted array: 11 12 22 25 64

-

7**a. Write the operations that can be performed on string using built-in functions. Explain any two functions. (08 Marks)**

String operations with built-in functions in C:

1. **strlen()** – Finds the length of a string.
2. **strcpy()** – Copies one string into another.
3. **strcat()** – Concatenates (joins) two strings.
4. **strcmp()** – Compares two strings.
5. **strupr() / strlwr()** – Converts string into uppercase / lowercase.
6. **strrev()** – Reverses the string.

👉 **Explanation of any two**:

strlen():
Prototype: int strlen(char str[]);
Example:

char str[] = "Hello";

int len = strlen(str); // len = 5

-

strcat():
Prototype: char *strcat(char dest[], char src[]);
Example:

char str1[20] = "Hello ";

char str2[] = "World";

- strcat(str1, str2); // str1 becomes "Hello World"

7**b. Develop a C program to concatenate 2 strings without using built-in function. (06 Marks)**

```
#include <stdio.h>

int main() {
    char str1[100], str2[100];
    int i, j;

    printf("Enter first string: ");
    gets(str1);
    printf("Enter second string: ");
    gets(str2);

    // Find end of str1
    for(i = 0; str1[i] != '\0'; i++);

    // Copy str2 at the end of str1
    for(j = 0; str2[j] != '\0'; j++, i++) {
        str1[i] = str2[j];
    }
    str1[i] = '\0';

    printf("Concatenated String: %s", str1);
    return 0;
}
```

7**c. Explain array of strings with an example. (06 Marks)**

- An **array of strings** is a 2D character array, where each row represents a string and each column stores characters.

Syntax:

```
char names[5][20];  // 5 strings, each can store up to 19 characters + null
```

-

Example:

```
#include <stdio.h>
int main() {
    char names[3][20] = {"Alice", "Bob", "Charlie"};

    for(int i = 0; i < 3; i++) {
        printf("Name %d: %s\n", i+1, names[i]);
    }
    return 0;
}
```

Output:

Name 1: Alice

Name 2: Bob

Name 3: Charlie

Q.8 (a) Program using pointer to compute sum, mean, and standard deviation

```
#include <stdio.h>
#include <math.h>

int main() {
    int n, i;
    float arr[100], *p, sum = 0.0, mean, sd = 0.0;
```

```
printf("Enter number of elements: ");
scanf("%d", &n);

printf("Enter %d real numbers:\n", n);
for(i = 0; i < n; i++) {
    scanf("%f", &arr[i]);
}

p = arr;  // pointer pointing to array

// Calculate sum
for(i = 0; i < n; i++) {
    sum += *(p + i);
}

mean = sum / n;

// Calculate standard deviation
for(i = 0; i < n; i++) {
    sd += pow(*(p + i) - mean, 2);
}
sd = sqrt(sd / n);

printf("Sum = %.2f\n", sum);
printf("Mean = %.2f\n", mean);
```

```
    printf("Standard Deviation = %.2f\n", sd);


    return 0;
}
```

✅ **Explanation**:

- p = arr; → pointer points to first element of array.
- *(p+i) → accesses elements using pointer arithmetic.
- Standard deviation formula used:
  SD=∑(xi−mean)2nSD = \sqrt{\frac{\sum (x_i - \text{mean})^2}{n}}SD=n∑(xi−mean)2

Q.8 (b) Pointer concept explanation

Declaration:

```
 int *ptr;   // ptr is a pointer to int
float *fptr; // fptr is a pointer to float
```

-

Initialization:

```
 int a = 10;
int *p = &a;  // p stores address of a
```

-

Accessing value:

```
 printf("%d", *p);  // prints value of a (dereferencing)
```

-

👉 Example Program:

```
#include <stdio.h>
int main() {
    int x = 25;
```

```
    int *p = &x;  // pointer points to x
    printf("Address of x = %p\n", p);
    printf("Value of x = %d\n", *p);
    return 0;
}
```

Q.8 (c) gets() and puts() functions

- **gets()** → Reads a string from standard input until newline (\n).
- **puts()** → Prints the string to standard output.

⚠️ Note: gets() is unsafe (buffer overflow issue), but used in basic C programs.

👉 Example:

```
#include <stdio.h>
int main() {
    char str[50];
    printf("Enter a string: ");
    gets(str);   // read string
    printf("You entered: ");
    puts(str);   // print string
    return 0;
}
```

✅ **Output Example**:

```
Enter a string: Hello World
You entered: Hello World
```

9**(a) What is structure? Explain the C syntax of structure declaration with example. (6**

Marks)

Definition:
A structure in C is a user-defined data type that groups different types of variables under a single name.
It is useful when you want to store data of different data types together, like student info (name, roll, marks).

Syntax:

```
struct structure_name {
    data_type member1;
    data_type member2;
    ...
};
```

Example:

```
#include <stdio.h>
struct Student {
    int roll;
    char name[30];
    float marks;
};

int main() {
    struct Student s1 = {1, "Ravi", 85.5};
    printf("Roll: %d, Name: %s, Marks: %.2f\n", s1.roll, s1.name, s1.marks);
    return 0;
}
```

9**(b) Compare structures and unions with syntax and example. (6 Marks)**

Feature	Structure	Union
Memory Allocation	Separate memory for each member.	Single shared memory, equal to the largest member.
Size	Sum of sizes of all members.	Size of the largest member.
Value Storage	All members can hold values at the same time.	Only one member can hold a value at a time.
Use Case	Store multiple attributes (student details).	Memory saving when variables are used one at a time.

Syntax:

```
// Structure
struct Student {
    int roll;
    float marks;
};

// Union
union Data {
    int i;
    float f;
};
```

Example:

```
#include <stdio.h>
```

```
struct Student {
    int roll;
    float marks;
};

union Data {
    int i;
    float f;
};

int main() {
    struct Student s = {1, 85.5};
    union Data d;
    d.i = 10;
    d.f = 20.5; // overwrites i

    printf("Structure: Roll=%d, Marks=%.2f\n", s.roll, s.marks);
    printf("Union: i=%d, f=%.2f\n", d.i, d.f);
    return 0;
}
```

9 **(c) Implement structures to read, write and compute average marks of the students. List the students scoring above and below the average marks for a class of N students. (8 Marks)**

Program:

```
#include <stdio.h>
```

```
struct Student {
    int roll;
    char name[30];
    float marks;
};

int main() {
    int n, i;
    float sum = 0, avg;

    printf("Enter number of students: ");
    scanf("%d", &n);

    struct Student s[n];

    for(i = 0; i < n; i++) {
        printf("Enter Roll, Name, Marks: ");
        scanf("%d %s %f", &s[i].roll, s[i].name, &s[i].marks);
        sum += s[i].marks;
    }

    avg = sum / n;
    printf("\nClass Average = %.2f\n", avg);

    printf("\nStudents scoring ABOVE average:\n");
```

```
    for(i = 0; i < n; i++) {
        if(s[i].marks > avg)
            printf("%d %s %.2f\n", s[i].roll, s[i].name, s[i].marks);
    }

    printf("\nStudents scoring BELOW average:\n");
    for(i = 0; i < n; i++) {
        if(s[i].marks < avg)
            printf("%d %s %.2f\n", s[i].roll, s[i].name, s[i].marks);
    }

    return 0;
}
```

Q.10 (a) Discuss the different modes of operation on files with suitable example. (08 Marks)

In C, files are opened using fopen() with different modes:

1. **"r" (read mode):** Opens an existing file for reading.
    - File must exist, otherwise NULL is returned.

```
FILE *fp = fopen("data.txt", "r");
```

2. **"w" (write mode):** Opens a file for writing.
    - If the file exists, old content is erased.
    - If not, a new file is created.

```
FILE *fp = fopen("data.txt", "w");
```

3. **"a" (append mode):** Opens a file for appending.
    - Data is written at the end of the file.

```
FILE *fp = fopen("data.txt", "a");
```

4. **"r+" (read + write mode):** Opens a file for both reading and writing.
    - File must exist.

```
FILE *fp = fopen("data.txt", "r+");
```

5. **"w+" (write + read mode):** Creates a file for reading and writing.
    - Overwrites if the file exists.

```
FILE *fp = fopen("data.txt", "w+");
```

6. **"a+" (append + read mode):** Opens a file for both reading and appending.
    - File pointer is at the end of file for writing.

```
FILE *fp = fopen("data.txt", "a+");
```

Q.10 (b) Compare between fgets() and gets(). (06 Marks)

Aspect	**gets()**	**fgets()**
Safety	Unsafe (no limit on input size).	Safe (you specify maximum characters to read).
Buffer Overflow	Possible.	Prevented.
Input Source	Reads from standard input only.	Can read from any file stream (stdin, file).
Newline Handling	Stops at newline, doesn't keep it.	Stores newline character if present.
Example	gets(str);	fgets(str, sizeof(str), stdin);

Hence, gets() is **deprecated**, and fgets() is preferred.

Q.10 (c) Discuss the enumerated data type. Explain the declaration and access of enumerated data type with a code in C. (06 Marks)

- **Definition:**
  An enum (enumeration) is a user-defined data type in C that consists of a set of named integer constants. It increases code readability.

Declaration Syntax:

enum enum_name {value1, value2, value3, ...};

-

Example Code:

```
#include <stdio.h>


enum Week {MON=1, TUE, WED, THU, FRI, SAT, SUN};


int main() {
    enum Week today;
    today = WED;
    printf("Day number: %d\n", today); // Output: 3
    return 0;
}
```

-
- **Explanation:**
    - enum Week defines constants MON=1, TUE=2, ... , SUN=7.
    - today = WED; assigns value 3 to variable today.
    - Provides meaningful names instead of plain integers.