PVP 23

Code: 23ES1102

I B.Tech - I Semester – Regular / Supplementary Examinations
DECEMBER 2025

INTRODUCTION TO PROGRAMMING
(Common for ALL BRANCHES)

Duration: 3 hours Max. Marks: 70

Note: 1. This question paper contains two Parts A and B.
2. Part-A contains 10 short answer questions. Each Question carries 2 Marks.
3. Part-B contains 5 essay questions with an internal choice from each unit. Each Question carries 10 marks.
4. All parts of Question paper must be answered in one place.

BL – Blooms Level CO – Course Outcome

PART – A

		BL	CO
1.a)	Differentiate between type conversion and casting.	L2	CO1
1.b)	Explain the process of compilation.	L2	CO1
1.c)	Describe control structures.	L2	CO1
1.d)	Explain the purpose of continue statement.	L2	CO1
1.e)	Describe the syntax for declaring a one-dimensional array in C.	L2	CO1
1.f)	Identify the header file required for string operations.	L1	CO1
1.g)	Explain the use of '&' operator in pointers.	L2	CO1
1.h)	Name any two dynamic memory allocation functions.	L1	CO1
1.i)	What is the use of function parameters?	L1	CO1
1.j)	Write the general syntax of a function in C.	L1	CO1

PART – B

			BL	CO	Max. Marks
		UNIT-I			
2	a)	Draw a flowchart to calculate Simple Interest.	L2	CO1	5 M
	b)	Write a short note on input and output operations in C.	L2	CO1	5 M
		OR			
3	a)	Write an algorithm and pseudo-code to find the largest of three numbers.	L2	CO1	7 M
	b)	Discuss top-down and bottom-up approaches in problem-solving.	L2	CO1	3 M
		UNIT-II			
4	a)	Develop a program to calculate factorial of a number using a loop.	L3	CO2	5 M
	b)	Differentiate between while and do-while loops with an example.	L2	CO1	5 M
		OR			
5	a)	Develop a program to reverse a number using a loop.	L3	CO2	5 M
	b)	Construct a program using switch statement.	L3	CO3	5 M
		UNIT-III			
6	a)	Develop a program for matrix addition using two-dimensional arrays.	L3	CO3	5 M

	b)	Construct a program to check whether a string is a palindrome or not.	L3	CO3	5 M
OR					
7	a)	Develop a program to find the largest element in an array.	L3	CO3	5 M
	b)	Construct a program to compare two strings.	L3	CO3	5 M
UNIT-IV					
8	a)	Demonstrate pointer arithmetic with suitable examples.	L3	CO3	5 M
	b)	Construct a program to demonstrate nested structures.	L3	CO3	5 M
OR					
9	a)	Develop a program to demonstrate the difference between structure and union.	L3	CO3	5 M
	b)	Analyze array of structures with an example.	L4	CO4	5 M
UNIT-V					
10	a)	Illustrate scope and lifetime of variables in C with an example.	L3	CO3	5 M
	b)	Explain file opening modes in C with an example.	L2	CO1	5 M
OR					
11	a)	Develop a program to copy the content from one file to another.	L3	CO3	5 M
	b)	Compare call-by-value and call-by-reference with an example.	L4	CO4	5 M

Scheme of evaluation

PVP 23

Code: 23ES1102

I B.Tech - I Semester – Regular / Supplementary Examinations
DECEMBER 2025

INTRODUCTION TO PROGRAMMING
(Common for ALL BRANCHES)

Duration: 3 hours Max. Marks: 70

Note: 1. This question paper contains two Parts A and B.
2. Part-A contains 10 short answer questions. Each Question carries 2 Marks.
3. Part-B contains 5 essay questions with an internal choice from each unit. Each Question carries 10 marks.
4. All parts of Question paper must be answered in one place.

BL – Blooms Level CO – Course Outcome

PART – A

1.a) Differentiate between type conversion and casting. (2M)

Any two differences - 2M

1.b) Explain the process of compilation. (2M)

The process of compilation - 2M

1.c) Describe control structures. (2M)

Sequential, Selection (Decision-Making) Control Structure, Iteration (Looping) Control Structure – 2M

1.d) Explain the purpose of continue statement (2M)

Any two points - 2M

1.e) Describe the syntax for declaring a one-dimensional array in C. (2M)

Syntax for declaring a one-dimensional array - 2M

1.f) Identify the header file required for string operations. (2M)

Header file - 2M

1.g) Explain the use of '&' operator in pointers. (2M)

Any two uses – 2M

1.h) Name any two dynamic memory allocation functions. (2M)

Any two pre-defined functions – 2M

1.i) What is the use of function parameters? (2M)

Answer with the ease of use – 2M

1.j) Write the general syntax of a function in C. (2M)

Standard function definition syntax – 2M

Part B

Unit -1

2. a) Draw a flowchart to calculate Simple Interest (5M)

Flowchart – 5M

2. b) Write a short note on input and output operations in C. (5M)

Input operations - 3M

Output operations – 2M

OR

3. a) Write an algorithm and pseudo-code to find the largest of three numbers. (7M)

Algorithm – 4M

Pseudo-code - 3M

3. b) Discuss top-down and bottom-up approaches in problem-solving. (3M)

Top-down approach – 2M

Bottom-up approach – 1M

.....

Unit -2

4.a) Develop a program to calculate factorial of a number using a loop. (5M)

Program – 5M

4.b) Differentiate between while and do-while loops with an example. (5M)

Any 3 differences – 3M

Example – 2M

OR

5.a) Develop a program to reverse a number using a loop. (5M)

Program – 5M

5. b) Construct a program using switch statement. (5M)

Program – 5M

.....

Unit -3

6.a) Develop a program for matrix addition using two-dimensional arrays. (5M)

Program – 5M

6.b) Construct a program to check whether a string is a palindrome or not. (5M)

Program – 5M

OR

7.a) Develop a program to find the largest element in an array. (5M)

Program – 5M

7.b) Construct a program to compare two strings. (5M)

Program – 5M

.....

Unit -4

8.a) Demonstrate pointer arithmetic with suitable examples. (5M)

Pointer arithmetic explanation - 3M

Examples - 2M

8.b) Construct a program to demonstrate nested structures. (5M)

Program – 5M

OR

9.a) Develop a program to demonstrate the difference between structure and union. (5M)

Program – 5M

9.b) Analyze array of structures with an example. (5M)

Example with explanation – 5M

.....

Unit -5

10.a) Illustrate scope and lifetime of variables in C with an example. (5M)

Scope and lifetime explanation – 3M

Examples - 2M

10.b) Explain file opening modes in C with an example. (5M)

Any 5 modes – 5 M

OR

11.a) Develop a program to copy the content from one file to another. (5M)

Program – 5M

11.b) Compare call-by-value and call-by-reference with an example. (5M)

Any 3 differences - 3M

Examples - 2M

.....

Code: 23ES1102

PVP 23

I B.Tech - I Semester – Regular / Supplementary Examinations
DECEMBER 2025

INTRODUCTION TO PROGRAMMING
(Common for ALL BRANCHES)

Duration: 3 hours Max. Marks: 70

Note: 1. This question paper contains two Parts A and B.
2. Part-A contains 10 short answer questions. Each Question carries 2 Marks.
3. Part-B contains 5 essay questions with an internal choice from each unit. Each Question carries 10 marks.
4. All parts of Question paper must be answered in one place.

BL – Blooms Level CO – Course Outcome

PART – A

1.a) Differentiate between type conversion and casting. (2M)

Any two differences - 2M

Ans) Type Conversion is an automatic process performed by the compiler to convert one data type into another during expression evaluation. It happens only when the source and destination data types are compatible.

Example: int a = 5; float b = a;

Type Casting is a manual conversion explicitly performed by the programmer using a cast operator.

Example: float x = (float)5 / 2;

Type casting can be performed between both compatible and incompatible data types, whereas type conversion occurs only between compatible data types.

(OR)

Feature	Type Conversion	Type Casting
Definition	Automatic conversion done by the compiler	Manual conversion done by the programmer
Control	Implicit (no programmer control)	Explicit (programmer controlled)
Syntax	No special syntax required	Uses cast operator (data_type)
Example	float b = a;	float x = (float)5 / 2;

1.b) Explain the process of compilation. (2M)

Process of compilation - 2M

Ans) The compilation process is the systematic procedure by which a C source program is transformed into a machine-executable file. This transformation occurs in multiple stages, where each stage performs specific operations such as code translation, error detection, and program preparation for execution. Through these stages, the high-level instructions written by the programmer are gradually converted into low-level machine instructions that the computer can understand and execute efficiently.

1.c) Describe control structures. (2M)

Sequential, Selection (Decision-Making) Control Structure, Iteration (Looping) Control Structure – 2M

Ans) Control structures in C are programming constructs that control the flow of execution of statements in a program. They allow a program to make decisions, repeat tasks, and execute statements in a specific order, thereby improving logical structure and readability.

i. Sequential Control Structure

In this structure, statements are executed **one after another** in the order in which they appear in the program. It is the **default execution flow** in C.

ii. Selection (Decision-Making) Control Structure

Selection control structures allow the program to choose different execution paths based on **conditions**.

These (if, if–else, else–if ladder, switch) statements evaluate a condition and execute a particular block of code accordingly.

iii. Iteration (Looping) Control Structure

Iteration control structures enable **repeated execution** of a block of statements as long as a condition is satisfied. For, while, do–while statements evaluate a condition and execute a particular block of code repeatedly.

1.d) Explain the purpose of continue statement. (2M)

Any two points - 2M

Ans) The continue statement is used within looping constructs for, while, and do–while to skip the remaining statements of the current loop iteration and immediately transfer control to the next iteration of the loop.

- In a for loop, control moves to the increment/decrement expression.
- In a while / do–while loop, control jumps to the condition check.

Key Purpose

- To avoid executing certain statements for specific conditions
- To improve program control and readability
- To selectively skip iterations without terminating the loop

---

1.e) Describe the syntax for declaring a one-dimensional array in C. (2M)

Syntax for declaring a one-dimensional array - 2M

Ans) A one-dimensional array in C is used to store multiple values of the same data type under a single variable name, with each element accessed using an index.

Syntax: data_type array_name[size];

1.f) Identify the header file required for string operations. (2M)

Header file - 2M

Ans) The header file required for string operations in C is:

#include <string.h>

It provides functions like strlen(), strcpy(), strcmp(), strcat(), etc.

1.g) Explain the use of '&' operator in pointers. (2M)

Any two uses – 2M

In C, the & operator is called the address-of operator or referencing operator. It is used to obtain the memory address of a variable, which is essential for pointer operations.

Purpose

- Assigns the address of a variable to a pointer
- Enables indirect access to variables through pointers

1.h) Name any two dynamic memory allocation functions. (2M)

Any two functions – 2M

Ans) Dynamic memory allocation functions in C are:

- malloc() - void* malloc(size_t size);
- calloc() - void* calloc(size_t num, size_t size);

(Other valid answers include realloc() and free().)

1.i) What is the use of function parameters? (2M)

Answer with the ease of use – 2M

Ans) Function parameters are used to pass data (values or variables) from the calling function to the called function. They enable a function to receive input, perform operations on that input, and produce meaningful results, making programs modular and reusable.

return_type function_name(data_type parameter1, data_type parameter2, ...);

1.j) Write the general syntax of a function in C. (2M)

Standard function definition syntax – 2M

Ans) A function in C is a self-contained block of code that performs a specific task and may return a value.

```
return_type function_name(parameter_list)
{
  // statements
}
```

- return_type → type of value returned by the function
- function_name → name of the function
- parameter_list → inputs to the function (if any)

PART - B

UNIT -1

2. a) Draw a flowchart to calculate Simple Interest. (5M)

Flowchart – 5M

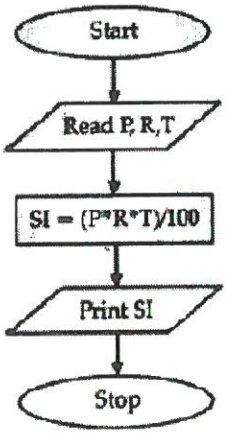
Ans) A flowchart is a graphical representation of an algorithm or process. It uses standard symbols connected by arrows to show the step-by-step flow of control in solving a problem. The **Simple Interest (SI)** is calculated using the formula:

$$SI = \frac{P \times R \times T}{100}$$

Where:

- **P** = Principal amount (initial amount of money)
- **R** = Rate of interest (per year, in %)
- **T** = Time period (in years)

Flowchart:



2. b) Write a short note on input and output operations in C. (5M)

Input operations - 3M

Output operations – 2M

Ans) In C, input and output operations are classified as **formatted** and **unformatted** based on whether **format specifiers** are used.

Formatted Input / Output

Formatted I/O uses **format specifiers** to control the type and format of data.

- **Formatted Input:** scanf()
- **Formatted Output:** printf()

Example:

```
scanf("%d", &a);
printf("Value = %d", a);
```

Unformatted Input / Output

Unformatted I/O does **not use format specifiers** and deals directly with characters or strings.

- **Unformatted Input:** getchar(), gets()
- **Unformatted Output:** putchar(), puts()

Example:

```
ch = getchar();
putchar(ch);
```

OR

3. a) Write an algorithm and pseudo-code to find the largest of three numbers. (7M)

Algorithm – 4M
Pseudo-code - 3M

Ans)

Algorithm: Largest of Three Numbers

Step 1: Start
Step 2: Read three numbers A, B, and C
Step 3: If A ≥ B and A ≥ C, then
  Set Largest ← A
Step 4: Else if B ≥ A and B ≥ C, then
  Set Largest ← B
Step 5: Else
  Set Largest ← C
Step 6: Display Largest
Step 7: Stop

Pseudo-code: Largest of Three Numbers

```
BEGIN
  READ A, B, C

  IF (A >= B) AND (A >= C) THEN
    Largest ← A
  ELSE IF (B >= A) AND (B >= C) THEN
    Largest ← B
  ELSE
    Largest ← C
  END IF

  PRINT Largest
END
```

Note: Any other equivalent logic award full marks.

3. b) Discuss top-down and bottom-up approaches in problem-solving. (3M)

Top-down approach – 2M
Bottom-up approach – 1M

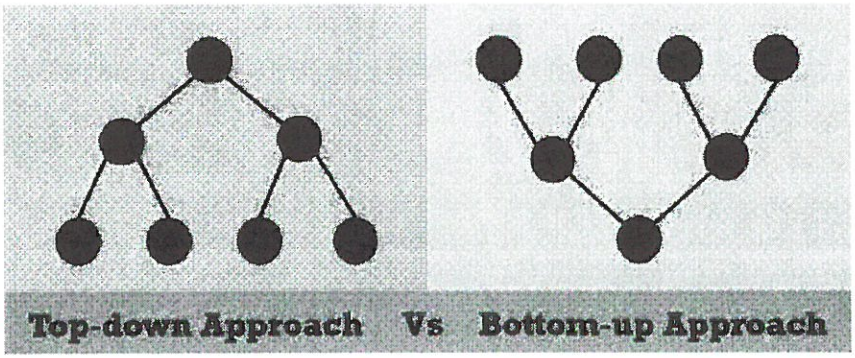
Ans)

i. Top-Down Approach (Deductive)

The top-down approach starts with the main problem and gradually breaks it down into smaller, manageable sub-problems until each part is simple enough to solve directly.
In this approach we focus on breaking up the big program into smaller program.
If the sub program is difficult, further we will break into smaller program.
Mainly this is used by structured programming languages like C, Fortan, COBOL etc.

ii. Bottom-Up Approach (Inductive)

The **bottom-up approach** starts by solving **small, basic components first**, and then combines them to build the complete system.
In bottom-up approach we will create small problems first, then we will solve the smaller problems.
Then we will integrate all the small problems into a whole and complete the solution.
Mainly used by object-oriented programming like C++, Java, Python etc.



.....

Unit -2

4.a) Develop a program to calculate factorial of a number using a loop. (5M)

Program – 5M

Ans)

The **factorial** of a non-negative integer **n** is the **product of all positive integers from 1 to n**. It is denoted by **n!**.

Example

- $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$
- $1! = 1$
- $0! = 1$

```
#include <stdio.h>

int main() {
   int n, i;
   long long fact = 1;

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

   for(i = 1; i <= n; i++) {
      fact = fact * i;
   }

   printf("Factorial of %d = %lld", n, fact);
   return 0;
}
```

4.b) Differentiate between while and do-while loops with an example. (5M)

Any 3 differences – 3M

Example – 2M

Ans) A while loop is an **entry-controlled** loop that checks the condition *before* executing the loop body, while a do-while loop is an **exit-controlled** loop that executes the body *at least once* before checking the condition.

Aspect	**while Loop**	**do-while Loop**
Condition Check	Checked at the beginning of the loop.	Checked at the end of the loop.
Minimum Executions	0 times (if the condition is initially false).	1 time (guaranteed execution of the body).

Control Type	Entry-controlled loop.	Exit-controlled loop.
Semicolon Usage	No semicolon is used after the while condition.	A semicolon is required after the while condition in the syntax (e.g., } while(condition);).
Use Case	When you want the loop body to run only if the condition is true.	When the loop body must run at least once (e.g., for user input validation).

while loop Example:

```
#include <stdio.h>

int main() {
   int i = 1; // Initialization

   while (i <= 5) { // Condition checked before each iteration
      printf("%d ", i); // Loop body statement
      i++;          // Update expression
   }

   printf("\\n"); // Newline for cleaner output
   return 0;
}
```

do-while loop Example:

```
#include <stdio.h>

int main() {
   int i = 1; // Initialization

   do {
      printf("%d ", i); // Loop body statements
      i++;          // Update expression
   } while (i <= 5);    // Condition checked after each iteration

   printf("\\n"); // Newline for cleaner output
   return 0;
}
```

OR

5.a) Develop a program to reverse a number using a loop. (5M)

Program – 5M

Ans)

The concept of reversing a number is based on extracting digits one by one from the original number and constructing a new number in reverse order.

Step	Number (n)	Extracted Digit	Reversed (rev)
1	1234	4	4
2	123	3	43
3	12	2	432
4	1	1	4321

```
#include <stdio.h>

int main()
{
   int n, rev = 0, digit;

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

   while (n > 0)
   {
      digit = n % 10;
      rev = rev * 10 + digit;
      n = n / 10;
   }

   printf("Reversed number = %d", rev);

   return 0;
}
```

Note: Any other equivalent logic can award full marks.

5. b) Construct a program using switch statement. (5M)

Program – 5M

Ans) The switch statement is a selection control structure used in programming to execute one block of code out of many choices, based on the value of a single expression or variable. It is an alternative to using multiple if–else statements when the decision depends on fixed, discrete values.

```
#include <stdio.h>

int main()
{
   int choice;
   float a, b;

   printf("Enter two numbers: ");
   scanf("%f %f", &a, &b);

   printf("\nMenu:\n1. Addition\n2. Subtraction\n3. Multiplication\n4. Division");
   printf("\nEnter your choice: ");
   scanf("%d", &choice);

   switch (choice)
   {
      case 1:
         printf("Result = %.2f", a + b);
         break;

      case 2:
         printf("Result = %.2f", a - b);
         break;
```

```
        case 3:
            printf("Result = %.2f", a * b);
            break;

        case 4:
            if (b != 0)
                printf("Result = %.2f", a / b);
            else
                printf("Division by zero not allowed");
            break;

        default:
            printf("Invalid choice");
    }

    return 0;
}
```

Note: Any other equivalent program can award full marks.

.....

Unit -3

6.a) Develop a program for matrix addition using two-dimensional arrays. (5M)

Program – 5M

Ans) Matrix addition is a process in which **two matrices of the same order (same number of rows and columns)** are added together to form a new matrix. In this process, the **corresponding elements** of the two matrices are added and stored in the same position of the resultant matrix.

C[i][j]=A[i][j]+B[i][j].

```
#include <stdio.h>

int main()
{
    int r, c, i, j;
    int A[10][10], B[10][10], C[10][10];

    printf("Enter number of rows and columns: ");
    scanf("%d %d", &r, &c);

    printf("Enter elements of Matrix A:\n");
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c; j++)
        {
            scanf("%d", &A[i][j]);
        }
    }

    printf("Enter elements of Matrix B:\n");
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c; j++)
```

```
        {
            scanf("%d", &B[i][j]);
        }
    }

    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c; j++)
        {
            C[i][j] = A[i][j] + B[i][j];
        }
    }

    printf("Resultant Matrix (A + B):\n");
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c; j++)
        {
            printf("%d ", C[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

6.b) Construct a program to check whether a string is a palindrome or not. (5M)

Program – 5M

Ans) A palindrome is a word, phrase, number, or other sequence of characters that reads the same forward and backward. The simplest way to check for a palindrome is to compare the original string with its reversed version.

Example: MADAM, MALAYALAM

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

int main()
{
    char str[100];
    int i, len, flag = 1;

    printf("Enter a string: ");
    scanf("%s", str);

    len = strlen(str);

    for (i = 0; i < len / 2; i++)
    {
        if (str[i] != str[len - i - 1])
        {
            flag = 0;
            break;
        }
```

```
    }

    if (flag)
      printf("The string is a Palindrome");
    else
      printf("The string is NOT a Palindrome");

    return 0;
}
```

OR

7.a) Develop a program to find the largest element in an array. (5M)

Program – 5M

Ans) Given an array **arr.** The task is to find the largest element in the given array.

Examples:
Input: *arr[] = [10, 20, 4]*
Output: *20*
Explanation: *Among 10, 20 and 4, 20 is the largest.*
Input: *arr[] = [20, 10, 20, 4, 100]*
Output: *100*

```
#include <stdio.h>

int main()
{
   int n, i;
   int arr[100];
   int largest;

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

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

   largest = arr[0];

   for (i = 1; i < n; i++)
   {
      if (arr[i] > largest)
      {
         largest = arr[i];
      }
   }

   printf("Largest element = %d", largest);

   return 0;
}
```

7.b) Construct a program to compare two strings. (5M)

Program – 5M

Note: Full marks can be awarded to the answer with using built-in function or without using built-in function.

Ans) The string comparison program checks two strings character by character to determine whether they are equal or not.

Example 1: Equal Strings

Strings:

str1 = "CAT"

str2 = "CAT"

Position	Character (str1)	ASCII	Character (str2)	ASCII	Result
0	C	67	C	67	Equal
1	A	65	A	65	Equal
2	T	84	T	84	Equal
3	\0	0	\0	0	End

Example 2: Unequal Strings

Strings:

str1 = "CAT"

str2 = "CAR"

Position	Character (str1)	ASCII	Character (str2)	ASCII	Result
0	**C**	**67**	**C**	**67**	**Equal**
1	**A**	**65**	**A**	**65**	**Equal**
2	**T**	**84**	**R**	**82**	**Not Equal**

Conclusion:

Mismatch found at position 2 → Strings are not equal

Program (Without using strcmp)

```
#include <stdio.h>

int main()
{
   char str1[100], str2[100];
   int i = 0, flag = 1;

   printf("Enter first string: ");
   scanf("%s", str1);

   printf("Enter second string: ");
   scanf("%s", str2);

   while (str1[i] != '\0' || str2[i] != '\0')
   {
      if (str1[i] != str2[i])
      {
         flag = 0;
         break;
      }
      i++;
   }
```

```
    if (flag)
      printf("Strings are equal");
    else
      printf("Strings are not equal");

    return 0;
}
```

Alternative (Using Library Function)

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

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

    printf("Enter first string: ");
    scanf("%s", str1);

    printf("Enter second string: ");
    scanf("%s", str2);

    if (strcmp(str1, str2) == 0)
      printf("Strings are equal");
    else
      printf("Strings are not equal");

    return 0;
}
```

.....

Unit -4

8.a) Demonstrate pointer arithmetic with suitable examples. (5M)

Pointer arithmetic explanation - 3M
Examples - 2M

Ans) **Pointer arithmetic** refers to performing arithmetic operations on **pointers** to access different memory locations. In C, pointers do not move by 1 byte when incremented or decremented; instead, they move by the **size of the data type** they point to.

Basic Rules of Pointer Arithmetic

Only the following operations are allowed:

- Increment (ptr++)
- Decrement (ptr--)
- Addition (ptr + n)
- Subtraction (ptr - n)
- Difference between two pointers (ptr1 - ptr2)

Note: Multiplication and division are **not allowed** on pointers.

Example 1: Pointer Increment

```
#include <stdio.h>

int main()
{
  int a = 10;
```

```
    int *p = &a;

    printf("Address of a = %p\n", p);
    p++;
    printf("Address after increment = %p\n", p);

    return 0;
}
```

Explanation

- If p points to an int, then p++ increases the address by **4 bytes** (on most systems).
- Pointer moves to the **next integer location**, not the next byte.

Example 2: Pointer Decrement

```
#include <stdio.h>

int main()
{
    int arr[3] = {5, 10, 15};
    int *p = &arr[2];

    printf("Value = %d\n", *p);
    p--;
    printf("After decrement, Value = %d\n", *p);

    return 0;
}
```

Explanation

- p-- moves the pointer to the **previous element** in memory

Example 3: Difference Between Two Pointers

```
#include <stdio.h>

int main()
{
    int arr[5];
    int *p1 = &arr[0];
    int *p2 = &arr[4];

    printf("Difference = %ld\n", p2 - p1);

    return 0;
}
```

Explanation

- The result gives the **number of elements** between the two pointers
- Here, output will be 4

8.b) Construct a program to demonstrate nested structures. (5M)

Program – 5M

Ans) A nested structure in C is a structure that contains another structure as one of its members. This concept is used to represent hierarchical or composite data, where a complex entity is naturally composed of smaller sub-entities. Nested structures enhance data organization, modularity, and readability in large programs.

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

```
/* Inner structure */
struct Date {
    int day;
    int month;
    int year;
};

/* Outer structure */
struct Student {
    int roll;
    char name[30];
    struct Date dob;   // Nested structure
};

int main() {
    struct Student s1;

    /* Assign values */
    s1.roll = 101;
    strcpy(s1.name, "Vinay Kumar");

    s1.dob.day = 29;
    s1.dob.month = 1;
    s1.dob.year = 2000;

    /* Display values */
    printf("Student Details\n");
    printf("Roll No   : %d\n", s1.roll);
    printf("Name      : %s\n", s1.name);
    printf("DOB       : %02d-%02d-%04d\n",
        s1.dob.day, s1.dob.month, s1.dob.year);

    return 0;
}
```

OR

9.a) Develop a program to demonstrate the difference between structure and union. (5M)

Program – 5M

Ans) In C, structures and unions are user-defined data types used to store different data types under a single name.
The main difference lies in memory allocation and data accessibility.

```
#include <stdio.h>

/* Structure declaration */
struct StructExample {
    int i;
    float f;
    char c;
};
```

```
/* Union declaration */
union UnionExample {
    int i;
    float f;
    char c;
};

int main() {
    struct StructExample s;
    union UnionExample u;

    printf("Size of structure = %lu bytes\n", sizeof(s));
    printf("Size of union     = %lu bytes\n\n", sizeof(u));

    /* Assign values to structure members */
    s.i = 10;
    s.f = 20.5;
    s.c = 'A';

    printf("Structure values:\n");
    printf("i = %d\n", s.i);
    printf("f = %.1f\n", s.f);
    printf("c = %c\n\n", s.c);

    /* Assign values to union members */
    u.i = 10;
    printf("After assigning i:\n");
    printf("i = %d\n", u.i);

    u.f = 20.5;
    printf("\nAfter assigning f:\n");
    printf("f = %.1f\n", u.f);

    u.c = 'A';
    printf("\nAfter assigning c:\n");
    printf("c = %c\n", u.c);
    return 0;
}
```

Aspect	**Structure**	**Union**
Memory Allocation	Allocates separate memory for each member.	All members share the same memory location.
Size	Size equals the sum of sizes of all members (plus padding).	Size equals the size of the largest member.
Data Storage	All members can store values simultaneously.	Only one member can store a value at a time.
Access to Members	Accessing one member does not affect others.	Accessing one member overwrites previous data.
Memory Efficiency	Less memory efficient when many members exist.	More memory efficient due to shared storage.

Data Safety	Safer, as each field retains its value.	Risky, since updating one field destroys others.
Use Case	Used when all data fields are required together.	Used when only one of the data fields is required at a time.
Initialization	Multiple members can be initialized.	Only one member can be initialized at a time.
Keyword Used	struct	union

9.b) Analyze array of structures with an example. (5M)

Example with explanation – 5M

Ans) An **array of structures** in C is a collection of structure variables stored in contiguous memory locations, where each element of the array is an individual structure (or "record") of the same type. This approach is used to manage multiple records that share the same data format, such as a list of employees, students, or products, in an organized and efficient manner.

Conceptual Analysis

A single structure represents **one entity** (one record). However, real-world applications usually involve **multiple entities with identical attributes**. Declaring an array of structures allows the programmer to store, access, and manipulate a large number of such entities using indexing, similar to arrays of primitive data types.

From a design perspective, an array of structures provides:

- Related data fields are logically grouped.
- Multiple records can be handled dynamically using loops.
- Reduces complexity compared to using multiple separate arrays for each field.

```
#include <stdio.h>

struct Student {
   int roll;
   char name[20];
   float marks;
};

int main() {
   struct Student s[3];   // Array of structures
   int i;

   /* Input data */
   for (i = 0; i < 3; i++) {
      printf("\nEnter details of student %d\n", i + 1);

      printf("Roll No: ");
      scanf("%d", &s[i].roll);

      printf("Name: ");
      scanf("%s", s[i].name);

      printf("Marks: ");
      scanf("%f", &s[i].marks);
   }
```

```
    /* Display data */
    printf("\n--- Student Details ---\n");
    for (i = 0; i < 3; i++) {
        printf("\nStudent %d\n", i + 1);
        printf("Roll No : %d\n", s[i].roll);
        printf("Name    : %s\n", s[i].name);
        printf("Marks   : %.2f\n", s[i].marks);
    }

    return 0;
}
```

An **array of structures** combines the advantages of both arrays and structures. It is an essential concept in C programming for handling complex data efficiently and is widely used in practical and real-time applications.

.....

Unit -5

10.a) Illustrate scope and lifetime of variables in C with an example. (5M)

Scope and lifetime explanation – 3M
Examples - 2M

Ans) Scope and Lifetime of Variables in C — Illustration with Example

In **C language**, every variable has **two important properties**:

- **Scope** → *Where the variable can be accessed in the program*
- **Lifetime** → *How long the variable exists in memory during program execution*

i. Scope of Variables

Local Scope

- Declared **inside a function or block**
- Accessible **only within that block**
- Created when the block is entered

Global Scope

- Declared **outside all functions**
- Accessible throughout the program (from declaration to end)

ii. Lifetime of Variables

Variable Type	Lifetime
Local variable	Exists while the block/function is executing
Global variable	Exists for the entire program execution
Static variable	Exists for entire program but scope may be local

iii. Example Program Demonstrating Scope and Lifetime

```
#include <stdio.h>

/* Global variable */
int g = 10;

void display() {
    /* Static local variable */
    static int s = 0;

    /* Local variable */
    int l = 5;
```

```
    s++;
    g++;

    printf("Inside display(): g = %d, s = %d, l = %d\n", g, s, l);
}

int main() {
    int i;

    for (i = 0; i < 3; i++) {
        display();
    }

    printf("Inside main(): g = %d\n", g);

    return 0;
}
```

Note: Any other equivalent answer, award full marks.

10.b) Explain file opening modes in C with an example. (5M)

Any 5 modes – 5 M

Ans) In C, files are opened using the function fopen().

The **file opening mode** determines **how a file is accessed** (read, write, append) and **how the file is treated** (created, overwritten, or preserved).

Syntax

```
FILE *fp;
fp = fopen("filename", "mode");
```

File Opening Modes

1. Read Mode ("r")

- Opens an **existing file** for reading
- File must exist, otherwise NULL is returned
- File pointer starts at the **beginning**

2. Write Mode ("w")

- Opens a file for writing
- Creates a new file if it does not exist
- **Erases existing contents** if the file already exists

3. Append Mode ("a")

- Opens a file for appending data
- Creates a file if it does not exist
- Data is added at the **end of the file**

4. Read & Write Modes

Mode	Description
"r+"	Read and write, file must exist
"w+"	Read and write, file is created or truncated
"a+"	Read and write, data is appended

5. Binary File Modes

Used for non-text files (images, audio, executables):

Mode	Description
"rb"	Read binary
"wb"	Write binary

Mode Description

"ab" Append binary

"rb+" Read/write binary

Example Program Demonstrating File Modes

```
#include <stdio.h>

int main() {
   FILE *fp;

   /* Write mode */
   fp = fopen("sample.txt", "w");
   fprintf(fp, "C File Handling Example\n");
   fclose(fp);

   /* Append mode */
   fp = fopen("sample.txt", "a");
   fprintf(fp, "Appending a new line\n");
   fclose(fp);

   /* Read mode */
   fp = fopen("sample.txt", "r");
   char ch;
   while ((ch = fgetc(fp)) != EOF) {
      putchar(ch);
   }
   fclose(fp);

   return 0;
}
```

Note: Any other equivalent answer, award full marks.

Mode	File Exists?	Operation	Pointer Position
"r"	Must exist	Read	Beginning
"w"	Optional	Write	Beginning
"a"	Optional	Write	End
"r+"	Must exist	Read/Write	Beginning
"w+"	Optional	Read/Write	Beginning
"a+"	Optional	Read/Write	End

OR

11.a) Develop a program to copy the content from one file to another. (5M)

Program – 5M

Ans) To copy the contents from one file to another in C, you need to open two files: the source file in read mode and the destination file in write mode. The process involves reading data from the source file character by character (or in blocks) and writing it to the destination file until the end of the source file is reached.

```
#include <stdio.h>

int main() {
```

```
    FILE *source, *target;
    char ch;

    /* Open source file in read mode */
    source = fopen("source.txt", "r");
    if (source == NULL) {
        printf("Cannot open source file.\n");
        return 1;
    }

    /* Open target file in write mode */
    target = fopen("target.txt", "w");
    if (target == NULL) {
        printf("Cannot open target file.\n");
        fclose(source);
        return 1;
    }

    /* Copy contents character by character */
    while ((ch = fgetc(source)) != EOF) {
        fputc(ch, target);
    }

    printf("File copied successfully.\n");

    /* Close both files */
    fclose(source);
    fclose(target);

    return 0;
}
```

11.b) Compare call-by-value and call-by-reference with an example. (5M)

Any 3 differences - 3M
Examples - 2M

Ans) Comparison of Call-by-Value and Call-by-Reference (with Example)

In programming, the way arguments are passed to a function affects whether the function can modify the original variables. The two common methods are **call-by-value** and **call-by-reference**.

i. Call-by-Value

- A **copy of the actual value** is passed to the function.
- Changes made inside the function **do not affect** the original variable.

Example (Call-by-Value in C)

```
#include <stdio.h>

void change(int x) {
    x = 50;
}

int main() {
```

```
    int a = 10;
    change(a);
    printf("a = %d", a);
    return 0;
}
```

Output

a = 10

ii. Call-by-Reference

- The **address of the variable** is passed to the function.
- Changes made inside the function **directly affect** the original variable.
- In C, this is achieved using **pointers**.

Example (Call-by-Reference in C)

```
#include <stdio.h>

void change(int *x) {
    *x = 50;
}

int main() {
    int a = 10;
    change(&a);
    printf("a = %d", a);
    return 0;
}
```

Output

a = 50

Note: Any other equivalent program, award full marks.

Feature	Call-by-Value	Call-by-Reference
Data passed	Copy of value	Address of variable
Effect on original variable	No change	Original value changes
Use of pointers	Not required	Required
Memory usage	More (copy created)	Less
Safety	Safer	Less safe

.....