Solution with scheme-VTU 2026

Prof. Rajni Tiwari,									
Final VTU – Feb 2026									
Sub	**Programming in C**					Sub code	**1BEIT105**	Branch	**ISE, AIDS, AIML, CSML**
Date	**16.02.2026**	Duration	**180 mins**	Max Marks	**50**	Sem /Sec	**I Sem P Cycle (A, B, C, D, E, F, G, H)**		**OBE**

CBCS SCHEME

USN

1BEIT105

First Semester B.E./B.Tech. Degree Examination, Dec.2025/Jan.2026

Programming in 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.	Define C. Explain the HISTORY of C.	5	L2	CO1
	b.	Explain the System development life cycle.	5	L2	CO1
	c.	Explain the General form of a C program with example.	5	L2	CO1
	d.	Explain the steps in Compiling a C Program. With suitable Example.	5	L2	CO1
		OR			
Q.2	a.	Define data type. Explain primitive data types supported in C language with example.	5	L2	CO1
	b.	Define Operators. Explain the Increment and Decrement operators with suitable Examples.	5	L2	CO1
	c.	Show the evaluation of the following expressions with intermediate and final values. i) x = a-b/3+ c*2 -1when a = 9, b = 12 , c = 3 10! = 10||5 < 4 && 8.	5	L2	CO1
	d.	Develop a C program to multiply, subtract and division by taking two whole numbers. Choose suitable data types for variables.	5	L3	CO5
		Module – 2			
Q.3	a.	Explain the Reading and Writing characters.	5	L2	CO2
	b.	With a suitable example, Explain formatted input and output statements.	5	L2	CO1
	c.	List the conditional branching statements in'C'. Explain any two with suitable examples.	5	L2	CO2
	d.	Develop a C program to print Floyd's triangle for N rows (N > 0).Choose suitable control statements. [forn=4] 1 23 456 78910	5	L3	CO5

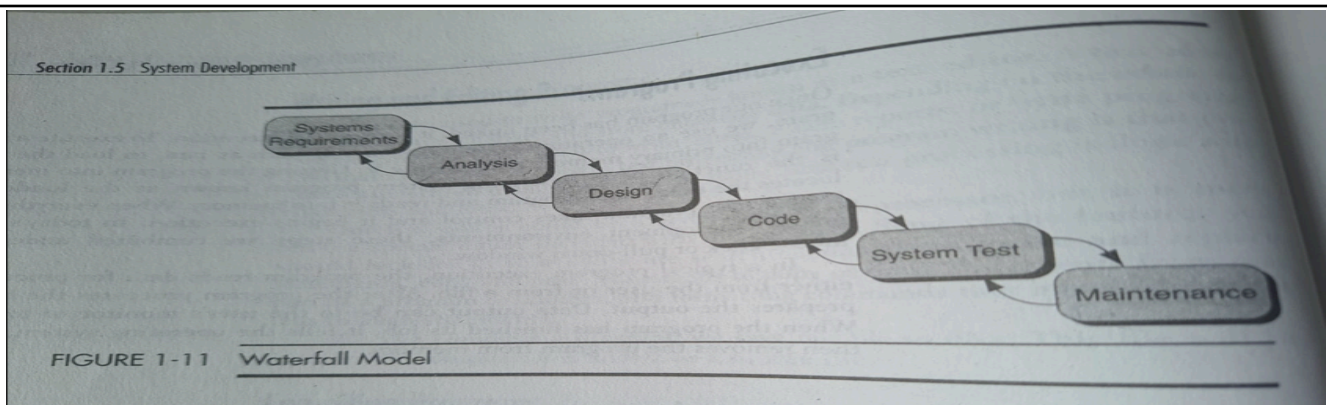
Q.4	a.	Explain the Iteration Statements with suitable Examples.	5	L2	CO
	b.	Explain the role of break and continue statements in C with suitable examples.	5	L2	CO2
	c.	Explain the go to , return and Block Statements in C with suitable examples.	5	L2	CO2
	d.	Develop a program to find the roots of quadratic equations.	5	L3	CO5
		Module – 3			
Q.5	a.	Define an array. How a single dimension and two-dimensional arrays are declared and initialized? Illustrate with suitable examples.	5	L2	CO2
	b.	Explain how arrays are passed to Functions.	5	L2	CO2
	c.	Define variable length array. Illustrate how variable length array is different from static array.	5	L2	CO2
	d.	Develop a C Program to find the Transpose of MATRIX.	5	L3	CO5
		OR			
Q.6	a.	Define a pointer. How do you declare and initialize pointer sin C. Explain accessing array elements using a pointer.	5	L2	CO2
	b.	Explain any two pointers operators and pointer Expressions with suitable Examples.	5	L2	CO2
	c.	Explain the concept of Multiple Indirection in Pointers	5	L2	CO2
	d.	Develop a C program to add two numbers using pointers to the variables.	5	L3	CO5

		Module – 4			
Q.7	a.	Define function. Explain the syntax of function definition and function declaration with a simple example.	5	L2	CO3
	b.	Explain the Function Arguments and Return statements in C.	5	L2	CO3
	c.	Explain the Function Prototypes with suitable Examples.	5	L2	CO3
	d.	Define recursion. Develop a C program and a function to compute factorial of a given number using recursion.	5	L3	CO3
		OR			
Q.8	a.	Define Dynamic memory allocation. List and explain the different functions to handle dynamic memory allocation in C.	5	L2	CO3
	b.	List the advantages of functions in programming.	5	L2	CO3
	c.	Explain TWO techniques of parameter passing to functions with suitable program segments.	5	L2	CO3

	d.	Develop a C-program and a function to check whether the given number is Prime or not.	5	L3	CO3
		Module – 5			
Q.9	a.	Define a structure in C. Explain the different types of structure declarations with examples.	5	L2	CO4
	b.	Explain how Array of Structures is passed to a function.	5	L2	CO4
	c.	Explain the differences between structures and unions.	5	L2	CO4
	d.	Develop a C program to store and display the Employee details using Structures.	5	L3	CO5
		OR			
Q.10	a.	Describe a method to compare two structure variables of the same type, and provide a simple example.	5	L2	CO4
	b.	Define Enumerated data type. Explain the declaration and access of enumerated data types with the help of C program segment.	5	L2	CO4
	c.	What are Bit-fields and typedef in C. Explain with example.	5	L2	CO4
	d.	Develop a C program to access and modify the members of structures, in array of structures in C.	5	L3	CO5

* * * * *

1 a)	**Define C. Explain the HISTORY of C?** **Ans:** **C is a general-purpose, structured, procedural, and mid-level programming language used for developing system software and application software.** **It provides low-level memory access using pointers and supports high-level programming constructs like functions, loops, and conditionals.** **C is widely used for:** ● **Operating systems** ● **Embedded systems** ● **Compilers** ● **Device drivers** ● **Application software** **History of C** The C language was developed by **Dennis Ritchie** in **1972** at **Bell Labs** in the United States. **Evolution of C:** 1. **BCPL (Basic Combined Programming Language)** o Developed by **Martin Richards** in 1967. 2. **B Language** o Developed by **Ken Thompson** in 1970 at Bell Labs. o Used for early versions of the **UNIX** operating system. 3. **C Language (1972)** o Created by Dennis Ritchie as an improved version of B. o Developed to rewrite the UNIX operating system. o Became powerful because it combined low-level and high-level features. **Standardization of C** ● **1989** – ANSI standardized C (ANSI C / C89). ● **1990** – ISO adopted it (C90). ● Later versions include: o C99 o C11 o C17 o C23 (latest standard)
1 b)	Explain the system Development life cycle? Ans:



1. Requirement Analysis
In this phase, developers and analysts work with users to understand and document what the system should do. All functional and non-functional requirements are collected through interviews or discussions and recorded in a Software Requirement Specification (SRS) document.
Example: **Deciding what features a student management system should have, like adding, viewing, or updating student details.**

2. System Design
This phase focuses on how the system will work. Designers create a blueprint showing the system architecture, data flow, database structure, and user interface. It helps developers know what to build.
Example: **Designing tables for students, subjects, and marks, and creating a layout for the input forms.**

3. Implementation (Coding)
Developers now write the actual source code using suitable programming languages based on the design. Each module is coded and then combined to form the complete system.
Example: **Writing the program in C, Java, or Python to store and display student data.**

4. Testing
After coding, the system is tested to find and fix errors. Different testing methods ensure that the software meets requirements and works correctly under all conditions.
Example: **Checking if the system correctly saves and retrieves student details.**

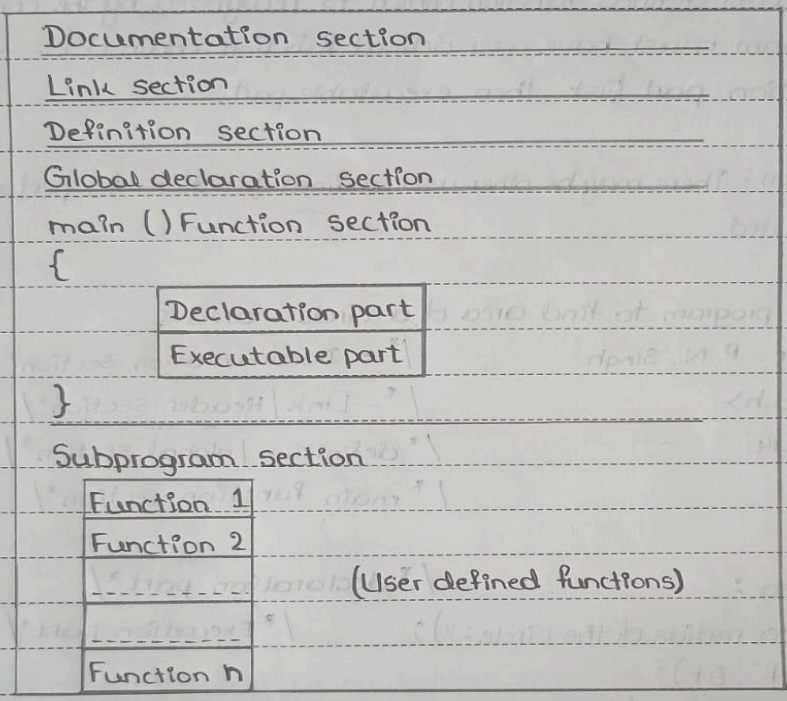
5. Deployment
Once testing is successful, the software is installed and delivered to the user environment. Users start using the system, and minor adjustments may be made based on feedback.
Example: **Installing the student management software in a school computer lab.**

6. Maintenance
After deployment, the system requires regular updates and bug fixes to adapt to new user needs or technologies. This phase ensures the software remains reliable and efficient over time.
Example: **Adding new features like attendance tracking or online access later.**

1 c) Explain the General form of a C program with example.

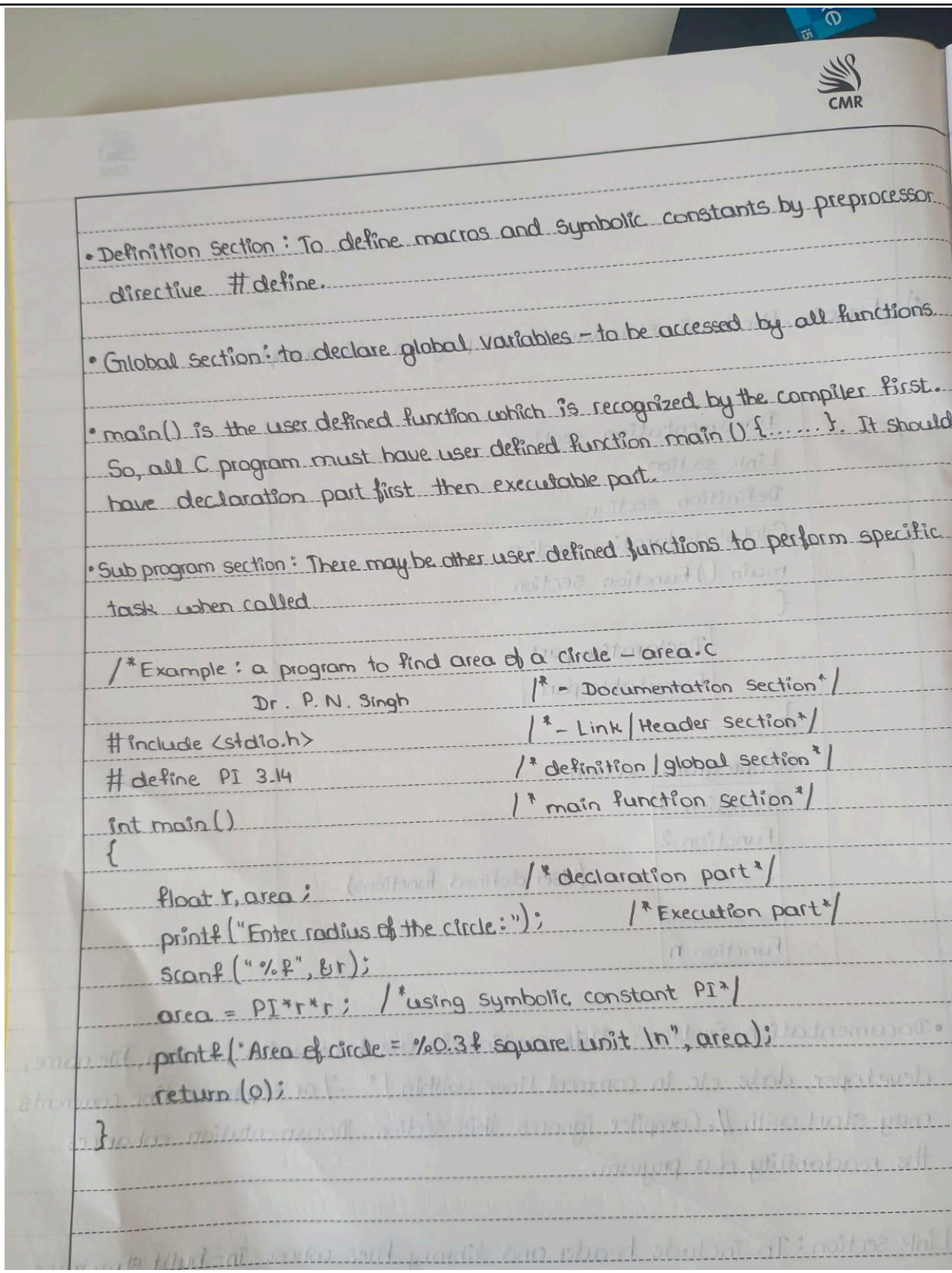
Ans:

1) Explain the basic structure of a C program with example.



- Documentation Section: This section is used to write problem, file n developer, date etc. In comment lines within /*....*/ or separate line com may start with //. Compiler ignores this section. Documentation enhance the readability of a program.

- Link section: To include header and library files whose in-built struct are to be used. Linker also required these files to build a program executable. Files are included with directive, #include

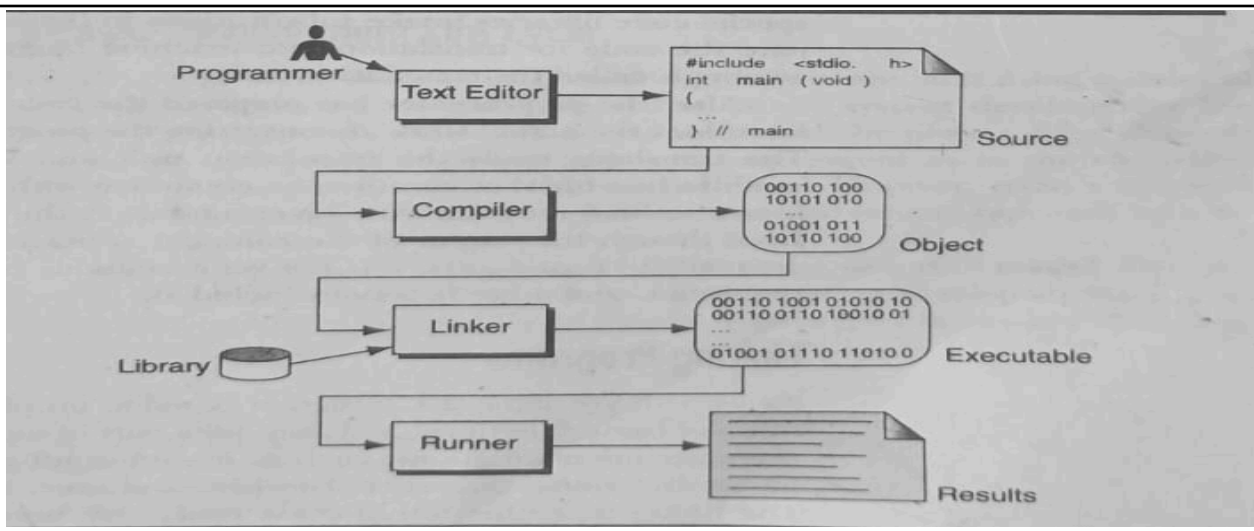


- Definition section : To define macros and symbolic constants by preprocessor directive #define.
- Global section : to declare global variables – to be accessed by all functions.
- main() is the user defined function which is recognized by the compiler first. So, all C program must have user defined function main(){.....}. It should have declaration part first then executable part.
- Sub program section : There may be other user defined functions to perform specific task when called

```
/*Example : a program to find area of a circle – area.c
            Dr. P. N. Singh                    /* - Documentation section*/
#include <stdio.h>                             /* - Link|Header section*/
#define PI 3.14                                /* definition|global section*/
int main()                                     /* main function section*/
{
    float r, area;                             /* declaration part*/
    printf("Enter radius of the circle:");     /* Execution part*/
    scanf("%f", &r);
    area = PI*r*r;  /*using symbolic constant PI*/
    printf("Area of circle = %0.3f square unit \n", area);
    return (0);
}
```

1 d) **Explain the steps in Compiling a C Program. With suitable Example.**

Ans:



The given diagram explains the steps involved in executing a C program.
First, the **programmer** writes the C program using a **text editor** and saves it as a source file with the extension .c. This file contains the program code written in C language.
Next, the **compiler** translates the source code into machine language (binary form). During compilation, the compiler checks for syntax errors. If there are no errors, it generates an **object file** (.obj or .o).
After that, the **linker** links the object file with the required **library files** (such as standard input/output functions like printf() and scanf()). It then produces an **executable file**.
Finally, the **runner (loader)** loads the executable file into memory and executes it. The program runs and displays the **output (result)** on the screen.
Thus, the main steps are:
Editing → Compilation → Linking → Execution.

2 a) Define data type. Explain primitive data types supported in C language with example.

Ans:
Ans:A data type in C defines the type of data a variable can store, the amount of memory it occupies, and the operations that can be performed on that data.
The primitive data types supported in C are
● int: It is used to store whole numbers. Memory size: 2 or 4 bytes (depends on system)
Example: int age = 20;
● float: is a primitive data type in C that is used to store real numbers (numbers with decimal or fractional values) with single precision.
Memory size: 4 bytes
Example: float salary = 25000.75;
● double: It is used to store decimal numbers with double precision.
Memory size: 8 bytes
Example: double pi = 3.14159;
● char: It is used to store a single character. Memory size: 1 byte
Example: char grade = 'A';

2 b) Define Operators. Explain the Increment and Decrement operators with suitable Examples.

Ans:

An **operator** is a symbol that performs a specific operation on one or more operands (variables or values) to produce a result.

Example:
`+`, `-`, `*`, `/`, `++`, `--` are operators in C.

Increment and Decrement Operators

In C, the **increment (++)** and **decrement (--)** operators are used to increase or decrease the value of a variable by 1.

1 Increment Operator (`++`)

The increment operator increases the value of a variable by 1.

There are two types:

(a) Pre-increment (`++i`)

- First increases the value.
- Then uses the updated value.

Example:
int i = 5;
int a = ++i;
Here,
i becomes 6 first,
a = 6

(b) Post-increment (`i++`)

- First uses the value.
- Then increases it.

Example:

int i = 5;
int a = i++;

2 Decrement Operator (`--`)

The decrement operator decreases the value of a variable by 1.

(a) Pre-decrement (`--i`)

- First decreases the value.
- Then uses it.

Example:

int i = 5;
int a = --i;

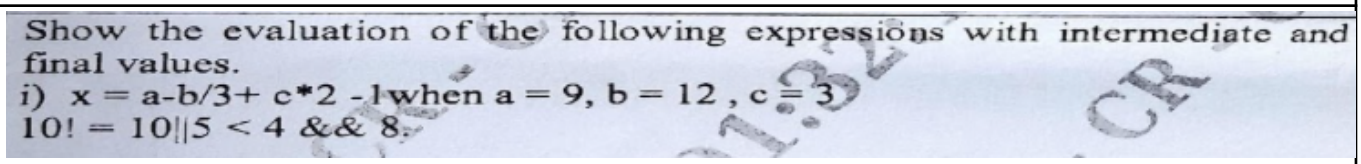
(b) Post-decrement (`i--`)

- First uses the value.
- Then decreases it.

Example:

int i = 5;
int a = i--;

2 c)


Show the evaluation of the following expressions with intermediate and final values.
i) x = a-b/3+ c*2 -1when a = 9, b = 12 , c = 3
10! = 10||5 < 4 && 8;

Ans:

(i) x=a−b/3+c∗2−1

Substitute the values:

x=9−12/3+3∗2−1

Step 1: Follow Operator Precedence

Division and multiplication first:

$$12/3 = 4$$

$$3 * 2 = 6$$

Expression becomes:

x=9−4+6−1

Step 2: Perform addition and subtraction left to right:

$$9 - 4 = 5$$

$$5 + 6 = 11$$

$$11 - 1 = 10$$

Final Answer:

x=10

(ii) 10 != 10 || 5 < 4 && 8

Step 1: Evaluate relational operators first

$$10! = 10 \rightarrow 0\ (False)$$

$$5 < 4 \rightarrow 0\ (False)$$

Expression becomes:

0 || 0 && 8

Step 2: Logical AND (&&) has higher precedence than OR (||)

0 && 8 \rightarrow 0

Expression becomes:

0 || 0 && 8

Step 2: Logical AND (&&) has higher precedence than OR (||)

0 && 8 \rightarrow 0

Expression becomes:

$$0||0$$

Step 3: Logical OR

$$0||0 \rightarrow 0$$

Final Answer:

$$Result = 0\ (False)$$

2 d) Develop a C program to multiply, subtract and division by taking two whole numbers. Choose suitable data types for variables.

Ans:

```
#include <stdio.h>

int main()
{
   int num1, num2;
   int subtraction, multiplication;
   float division;
```

```
    printf("Enter two whole numbers: ");
    scanf("%d %d", &num1, &num2);

    subtraction = num1 - num2;
    multiplication = num1 * num2;

    if(num2 != 0)
        division = (float)num1 / num2;
    else
    {
        printf("Division by zero is not possible.\n");
        return 0;
    }

    printf("Subtraction = %d\n", subtraction);
    printf("Multiplication = %d\n", multiplication);
    printf("Division = %.2f\n", division);

    return 0;
}
```

3 a) Explain reading and writing characters?

Ans

1 Reading a Character – `getchar()`

✅ **Definition:**

`getchar()` is used to read a single character from the keyboard.

✅ **Syntax:**

```
char ch;
ch = getchar();
```

Example:

```
#include <stdio.h>

int main()
{
    char ch;

    printf("Enter a character: ");
    ch = getchar();   // Reads one character

    printf("You entered: ");
    putchar(ch);

    return 0;
}
```

2 Writing a Character – `putchar()`

✅ **Definition:**

	`putchar() is used to display a single character on the screen.`
	
	✅ **Syntax:**
	
	putchar(character);
	Example:
	#include <stdio.h>
	
	int main()
	{
	char ch = 'A';
	putchar(ch); // Displays A
	return 0;
	}

3 b)	With suitable example explain formatted input output statements.
	In C, input involves providing data to a program, and output means displaying or sending data from the program. The stdio.h header file contains essential input and output functions, such as scanf() for reading data and printf() for displaying it.These all functions are collectively known as Standard I/O Library function.
	Formatted I/O functions:
	
	Formatted I/O functions which refers to an Input or Ouput data that has been arranged in a particular format. There are mainly two formatted I/O functions discussed as follows:
	● printf()
	● scanf()
	
	printf():
	printf() function is used in a C program to display any value like float, integer, character, string, etc on the console screen. It is a pre-defined function that is already declared in the stdio.h(header file).
	Syntax 1:
	To display any variable value.
	printf("Format Specifier", var1, var2, ..., varn);
	Syntax 2:
	To display any string or a message
	printf("Enter the text which you want to display");
	Example of printf()
	printf("%d %c", info_a, info_b);
	scanf():
	scanf() function is used in the C program for reading or taking any value from the keyboard by the user, these values can be of any data type like integer, float, character, string, and many more. This function is declared in stdio.h(header file), that's why it is also a pre-defined function. In scanf() function we use &(address-of operator) which is used to store the variable value on the memory location of that variable.
	
	Syntax:
	scanf("Format Specifier", &var1, &var2, ..., &varn);
	Example of scanf()

scanf(“%d %c”, &info_a,&info_b);

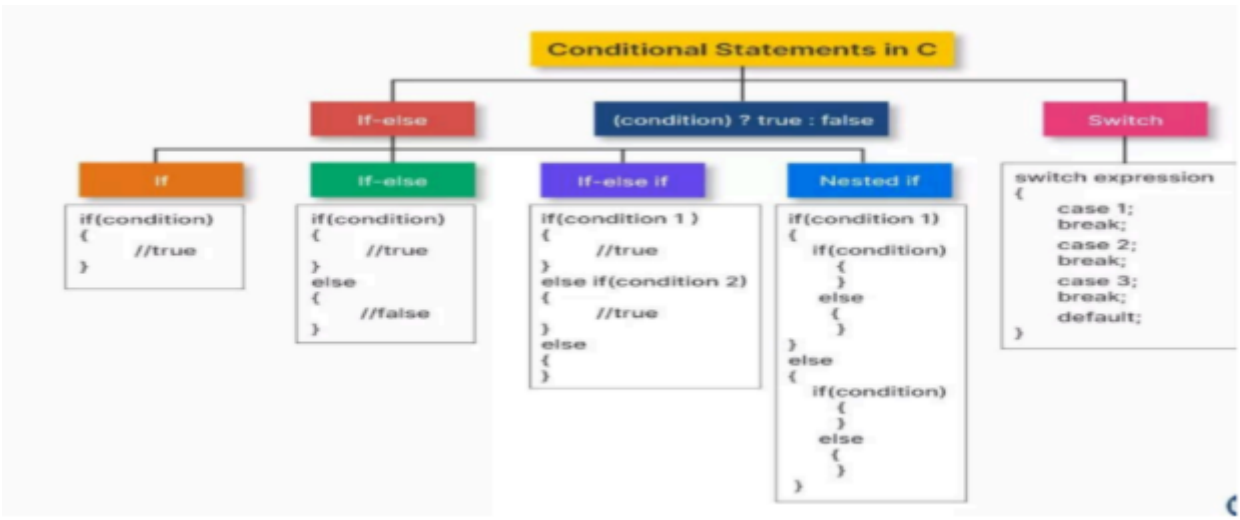
Example:

```
// C program to implement
// scanf() function
#include <stdio.h>


// Driver code
int main()
{
int num1;
// Printing a message on
// the output screen
printf("Enter a integer number: ");
// Taking an integer value
// from keyboard
scanf("%d", &num1);
// Displaying the entered value
printf("You have entered %d", num1);
return 0;
}
```

Output:
Enter a integer number: 56
You have entered 56

3 c) List the conditional branching statements in‘C’. Explain any two with suitable examples.

Ans:
conditional branching statements are used to control the flow of execution based on certain conditions. Here are two commonly used conditional branching Statements:



1. if Statement

The if statement is used to execute a block of code only if a specified condition is true. If the condition is false, the code block is skipped.

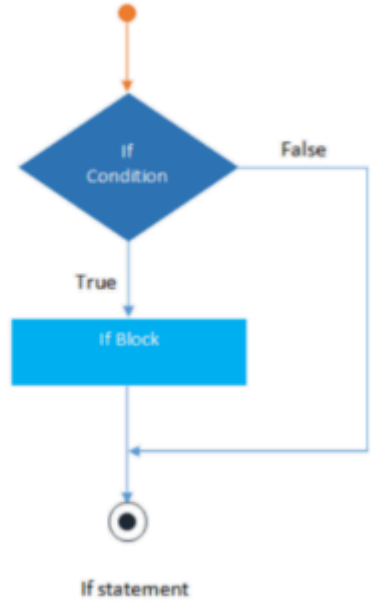
Syntax:

1. Simple if statement:

The general form of single if statement is :

```
if ( test expression)
{
statement-Block;
}
statement-Blocks;
```

Flowchart:



If statement

Working:
Condition: This is a Boolean expression that evaluates to either true (non-zero value) or false
(zero). If the condition is true, the statements inside the block are executed.
Statement block: This is the code that runs if the condition is true. If the condition is false, this
block is skipped.
Example:

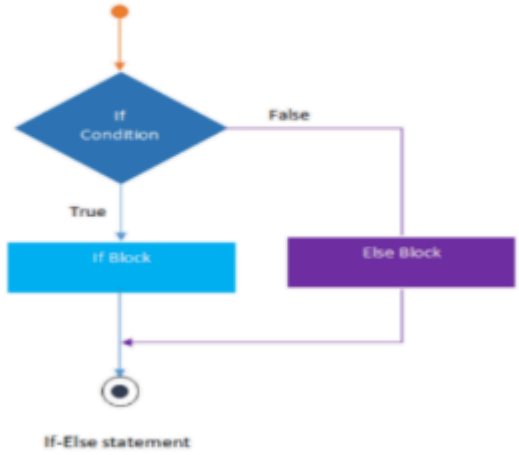
```
#include<stdio.h>
int main()
{
int num;
num = 4;
if(num%2 == 0)
{
printf("\n%d is even",num);
}
printf("\nOut of if body...");
return 0;
}
```

Output:
4 is even
Out of if body...
2. if else statement
The if-else statement is a two way decision making statement in C. The if-else statement allows two different blocks of code to execute based on whether a condition is true or false. If the condition is true, the if block is executed; otherwise, the else block is executed.

Syntax: **Flowchart:**

The general form of if-else statement is

```
if ( test expression)
{
 statement-block1;
}
else
{
statement-block2;

}
```



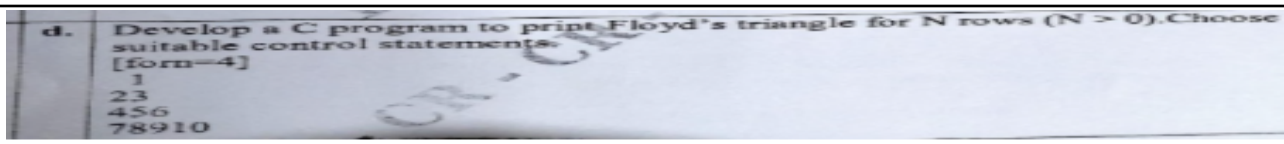
If-Else statement

Working:
Condition: This is a Boolean expression that evaluates to either true (non-zero value) or false
(zero). If the condition is true, the statements inside the if block are executed.If the condition is
false else block will execute.
Example:

```
#include<stdio.h>
int main()
{
int num;
num = 4;
if(num%2 == 0)
{
printf("\n%d is even",num);

}
else
{
printf("\n%d is odd",num);
}
printf("\nOut of if-else...");
return 0;
}
```

Output 1: for num=4
4 is even
Out of if-else...
Output 2: for num=5
5 is odd
Out of if-else...

3 d)


d. Develop a C program to print Floyd's triangle for N rows (N > 0).Choose suitable control statements.
[form=4]
1
23
456
78910

```
#include <stdio.h>
int main() {
int n, i, j;
int num = 1; // starting number
printf("Enter number of rows (N > 0): ");
scanf("%d", &n);
// Outer loop for rows
for (i = 1; i <= n; i++) {
// Inner loop for printing numbers in each row
for (j = 1; j <= i; j++) {
printf("%d ", num);
num++; // increment the number
}
printf("\n");
}
return 0;
}
```

Output:

```
Enter number of rows (N > 0): 4
1
2 3
4 5 6
7 8 9 10
PS C:\Users\91819\OneDrive\Desktop\C_Programming>
```

4 a)

Explain iteration statements with suitable examples.

Ans:

Iteration statements are used to **repeat a block of code multiple times** until a given condition becomes false. They are also called **looping statements**.

In C, there are **three types of iteration statements**:

1. `for` loop
2. `while` loop
3. `do-while` loop

1️⃣ for Loop

✅ **Used when the number of iterations is known.**

🔹 **Syntax:**

```
for(initialization; condition; increment/decrement)
{
   // statements
```

```
}
Example:
#include <stdio.h>

int main()
{
   int i;
   for(i = 1; i <= 5; i++)
   {
      printf("%d ", i);
   }
   return 0;
}
```

2 while Loop

✅ **Used when the number of iterations is not fixed.**

✅ **Condition is checked before execution.**

🔹 **Syntax:**

```
while(condition)
{
   // statements
}
Example:
#include <stdio.h>

int main()
{
   int i = 1;
   while(i <= 5)
   {
      printf("%d ", i);
      i++;
   }
   return 0;
}
```

3 do-while Loop

✅ **Executes the loop body at least once.**

✅ **Condition is checked after execution.**

🔹 **Syntax:**

```
do
{
   // statements
} while(condition);
 Example:
#include <stdio.h>
```

```
int main()
{
   int i = 1;
   do
   {
      printf("%d ", i);
      i++;
   } while(i <= 5);

   return 0;
}
```

4 b) Explain the role of break and continue statements in c with suitable examples.

Ans:

break statement would only exit from the loop containing it.

```
//Example program to checka number whether it is prime
#include<stdio.h>
int main()
{
int n,d,prime=1;
printf("Enter a number:");
scanf("%d",&n);
for(d=2;d<n/2;d++)
{
if(n%d==0)
prime=0;
break ;
}
if(prime)
printf("Yes%disaprimenumber.\n",n);
else
printf("No,%disnotaprimenumber.\n",n);
return(0);
}
```

The continue statement is used in loops to skip the following statements in the loop and to continue with the next iteration (current iteration in for loop).

```
//Example program to entermarks(0-100)in 6 subjects for sum
 #include<stdio.h>
intmain()
{
intmarks,sum=0,x;
for(x=1;x<=6;x++)
{
printf("Entermarks%d:",x)
scanf("%d",&marks);

if(marks<0||marks>100)
{
printf("Invalidmarks!!!\n");
continue;/*again read marks for same paper*/
}
sum+=marks;
```

	} printf(“sumofmarks=%d.\n”,sum); return (0); }

4 c)	Explain goto, return block statements in c with suitable example.
	
	goto statement of C unconditional control statement and it transfers the control from one place to other followedby alabel name. Itmay corruptthe structure ofC program so,programmers avoid to use it.
	Examplevof working of goto:
	#include <stdio.h>
	intmain(){
	intmarks;
	again:
	printf("Entermarks(0-100):");
	scanf("%d",&marks);
	if(marks<0||marks>100)
	{
	printf("InvalidMarks!!!\n");
	gotoagain;/*Controlwillbetransferredtolabelagain:whereitiswritteninprogram*/
	}
	if (marks >= 40)
	printf("Pass\n");
	else
	printf("Fail\n");
	return (0);
	}
	Expectedoutput:
	Entermarks(0-100):120
	Invalid marks
	Entermarks(0-100):67 Pass
	
	## Return Statement in Block (Compound) Statements in C
	
	In C, a **block statement** (also called a compound statement) is a group of statements enclosed in curly braces:
	
	{
	// statements
	}
	
	The `return` statement can be used **inside a block**, such as inside:`if,else,for,while` any function block.When `return` executes inside a block:
	
	● The function **immediately stops**
	
	● Control goes back to the calling function

	• Any remaining statements in the function are skipped
	
	🔹 **Example 1: `return` inside an `if` block**
	#include <stdio.h>
	
	int check(int num) {
	  if (num < 0) {
	  return -1; // return inside block
	  }
	  return 1;
	}
	
	int main() {
	  int result = check(-5);
	  printf("%d\n", result);
	  return 0;
	}

4 d)	Develop c program to find roots of quadratic equation .
	
	/* C program to find roots of a quadratic equation */
	#include <stdio.h>
	#include <math.h>
	int main()
	{
	double a, b, c, d, r1, r2, real, imag;
	printf("\nEnter coefficients a, b and c: ");
	scanf("%lf %lf %lf",&a, &b, &c);
	d = b*b-4*a*c; /* discriminant */
	if (d == 0)
	{
	r1 = r2 = -b/(2*a);
	printf("Roots are real : root1 = root2 = %.2lf;", r1);
	}
	else
	if (d > 0)
	{
	r1 = (-b+sqrt(d))/(2*a);
	r2 = (-b-sqrt(d))/(2*a);
	printf("Roots are distinct: root1 = .2lf root2 = %.2lf\n",r1, r2);
	}
	else
	{
	real = -b/(2*a);
	imag = sqrt(-d)/(2*a);

```
printf("Roots are imaginary:\n");
printf("root1 = %.2lf+%.2lfi and root2 = %.2lf-%.2lfi", real, imag, real,
imag);
}
return (0);
}
```

Expected Output:

Enter coefficients a, b and c:

Enter coefficients a, b and c: 2 4 2

Roots are real : root1 = root2 = -1.00;

5 a) **Define an array. How a single dimension and two-dimensional arrays are declared and initialized? Illustrate with suitable examples.**

Ans:

An array is a collection of similar data items stored at contiguous memory locations and elements can be accessed randomly using indices of an array.

(1) 1-D Array declaration by specifying size

Syntax:

```
datatype array[size];
// Array declaration by specifying size
int arr1[10];
```

(2) 1-D Array declaration by initializing elements

```
// Array declaration by initializing elements
int arr[ ] = { 10, 20, 30, 40 };
```

Compiler creates an array of size 4.

above is same as int arr[4] = {10, 20, 30, 40}

```
// Program to take 5 values from the user and store them in an
// print the elements stored in the array
#include <stdio.h>
int main()
{
int values[10] ;
int i;
printf("Enter 5 integers: "); // 2 4 3 5 7
// taking input and storing it in an array
for( i = 0; i < 5; i++)
{
scanf("%d", &values[i]);
// Reading 2 4 3 5 7
}
printf("Displaying integers: ");
// printing elements of an array
for(i = 0; i < 5; i++)
{
printf("%d\n", values[i]);
}
return 0;
}
```

2D array
Syntax to declare 2D array
data_type array_name[rows][columns];

example:

int matrix[3][4]; // 3 rows, 4 columns

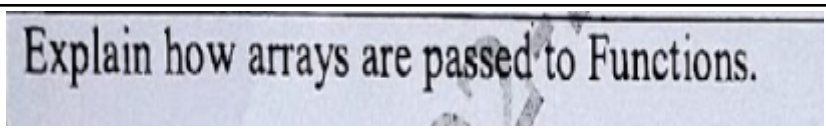
Initialization of Two-Dimensional Arrays

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

Example Program Using 2D Array

```
#include <stdio.h>
int main() {
int matrix[2][3] = {
{10, 20, 30},
{40, 50, 60}
};
printf("Matrix elements:\n");
for (int i = 0; i < 2; i++) {
for (int j = 0; j < 3; j++) {
printf("%d ", matrix[i][j]);
}
printf("\n");
}
return 0;}
```

5b)



Ans:

In C, **arrays are passed to a function by address (reference)**.
When we pass an array to a function, we actually pass the **base address of the first element**.

So, the function works on the **original array**, not a copy.

Syntax:

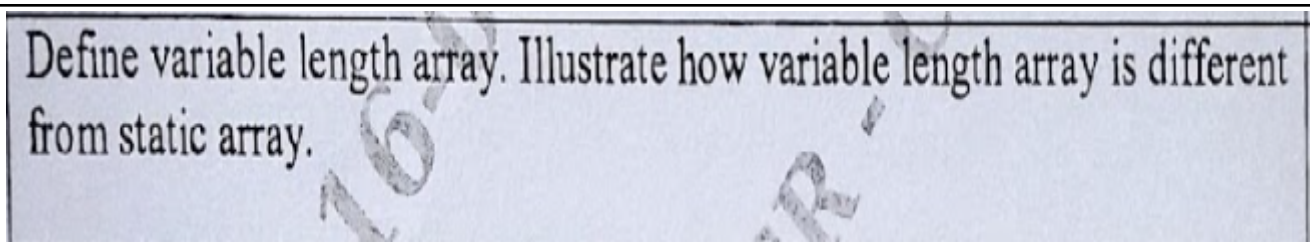
```
void functionName(int arr[], int size);
```

Example Program:

```
#include <stdio.h>

// Function to display array elements
void display(int arr[], int n) {
    int i;
    for(i = 0; i < n; i++) {
        printf("%d ", arr[i]);
```

```
    }
}

int main() {
    int a[5] = {10, 20, 30, 40, 50};

    display(a, 5);   // Passing array to function

    return 0;
}
```

5 c)


Define variable length array. Illustrate how variable length array is different from static array.

Ans:

A Variable Length Array is a data structure introduced in C99 that allows the length of an array to be specified using a variable whose value is known at runtime, enabling more flexible memory use on the stack.

```
#include <stdio.h>
int main() {
int n;
printf("Enter the size of the array: ");
scanf("%d", &n); // User decides the size at runtime
int arr[n]; // Variable Length Array (VLA)
// Take input into the array
printf("Enter %d numbers:\n", n);
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
// Print the array
printf("You entered: ");
for (int i = 0; i < n; i++) {
printf("%d ", arr[i]);
}
printf("\n");
return 0;
}
```

OUTPUT:

```
Enter the size of the array: 4
Enter 4 numbers:
10 20 30 40
You entered: 10 20 30 40
```

A static array is an array with a size that is known before the program starts executing, and memory for the array is allocated at compile time.

```
#include <stdio.h>
int main() {
int numbers[5] = {10, 20, 30, 40, 50}; // Static array with fixed size 5
for(int i = 0; i < 5; i++) {
printf("%d ", numbers[i]);
}
return 0;
```

}

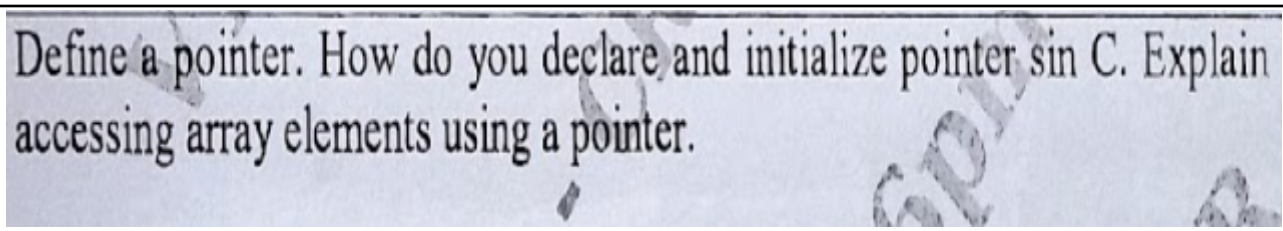
· A static array has a fixed size known at compile time.
· A variable length array (VLA) has a size determined at runtime.

5 d) Develop a C Program to find the Transpose of MATRIX.

Ans:

```
#include <stdio.h>

int main() {
    int a[10][10], transpose[10][10];
    int rows, cols, i, j;

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

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

    // Finding transpose
    for(i = 0; i < rows; i++) {
        for(j = 0; j < cols; j++) {
            transpose[j][i] = a[i][j];
        }
    }

    // Display original matrix
    printf("\nOriginal Matrix:\n");
    for(i = 0; i < rows; i++) {
        for(j = 0; j < cols; j++) {
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }

    // Display transpose matrix
    printf("\nTranspose Matrix:\n");
    for(i = 0; i < cols; i++) {
        for(j = 0; j < rows; j++) {
            printf("%d ", transpose[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

6 a)



Ans:A pointer is a variable that holds a memory address. This address is the location of another object (typically another variable) in memory.
For example, if one variable contains the address of another variable, the first variable is said to point to the second.

1. Pointer Declaration

A pointer is a variable that stores the **address of another variable**.

👉 **Syntax:**

data_type *pointer_name;

Example:

int *p;

float *f;

char *c;

2. Pointer Initialization

Initialization means assigning the **address of a variable** to the pointer.

We use **address operator (&)**.

👉 **Syntax:**

pointer_name = & var_name;

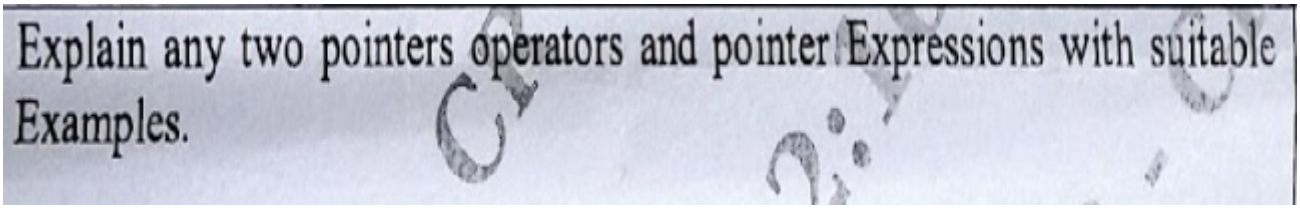
Example:

p = &a;

```
#include <stdio.h>

int main()

{

   int arr[100];

   int n, i;

   int sum = 0;

   float mean;

   int *ptr;
```

```
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }
    ptr = arr;   // pointer points to first element of array
    /* Calculate sum using pointer */
    for (i = 0; i < n; i++)
    {
        sum = sum + *(ptr + i);
    }
    mean = (float)sum / n;
    printf("Sum of elements = %d\n", sum);
    printf("Mean of elements = %.2f\n", mean);
    return 0;
}
```

6 b)



Ans:**Pointer Operators in C**

In C, pointers mainly use **two special operators**:

1. & (Address-of operator)
2. * (Indirection / Dereference operator)

1 Address-of Operator (&)

- It gives the **address of a variable**.
- Used to assign the address to a pointer.

📌 **Example:**

```c
#include <stdio.h>

int main()
{
   int a = 10;
   int *p;

   p = &a;   // & gives address of a

   printf("Value of a = %d\n", a);
   printf("Address of a = %p\n", &a);
   printf("Value stored in pointer p = %p\n", p);

   return 0;
}
```

2️⃣ Indirection / Dereference Operator (*)

- Used to **access the value stored at the address**.
- It is also used to declare a pointer.

📌 Example:

```c
#include <stdio.h>

int main()
{
   int a = 20;
   int *p = &a;

   printf("Value of a = %d\n", *p);   // *p gives value of a

   return 0;
}
```

Pointer Expression in C

Pointer expressions are operations performed using pointers.

✅ Common Pointer Expressions:

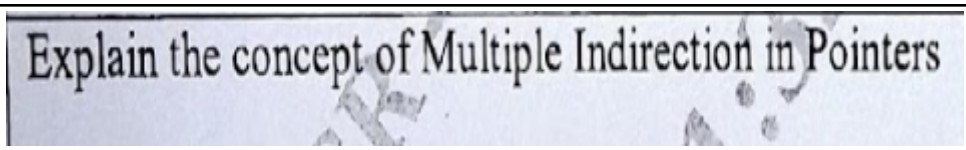
1. `p++` → Move pointer to next memory location
2. `(*p)++` → Increment value stored at pointer
3. `p + n` → Move pointer n positions forward

📌 Example of Pointer Expression:

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

```
int main()
{
   int arr[] = {10, 20, 30};
   int *p = arr;

   printf("First value = %d\n", *p);

   p++;   // Pointer expression (move to next element)

   printf("Second value = %d\n", *p);

   return 0;
}
```

6 c)



Ans:

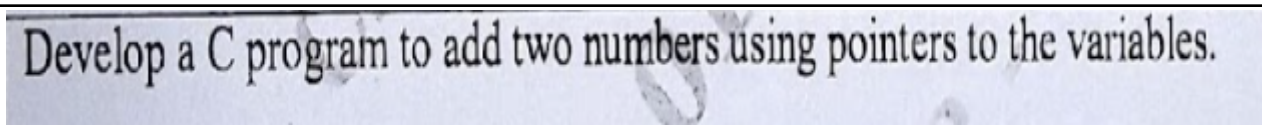
Multiple Indirections in Pointer (C)

Multiple indirection means a **pointer pointing to another pointer**, which finally points to a normal variable. It is also called a **pointer to pointer**. In this concept, each extra * represents one more level of indirection. It is useful when we want to modify a pointer inside a function or when working with dynamic memory and 2D arrays. To access the final value, we use the same number of * as the pointer level.

Example:

```
#include <stdio.h>

int main()
{
   int a = 5;
   int *p = &a;      // Single pointer
   int **q = &p;     // Double pointer

   printf("Value of a = %d\n", a);
   printf("Using single pointer = %d\n", *p);
   printf("Using double pointer = %d\n", **q);

   return 0;
}
```

6 d)



Ans:

```
#include <stdio.h>

int main()
{
```

```
    int num1, num2, sum;
    int *p1, *p2;

    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    // Assign addresses to pointers
    p1 = &num1;
    p2 = &num2;

    // Add using pointers
    sum = *p1 + *p2;

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

    return 0;
}
```

7 a) Define function. Explain the syntax of function definition and function declaration with a simple example.

Ans:
Functions are the building blocks of C and the place where all program activity occurs. The General Form of a Function:
ret-type function-name(parameter list) {
body of the function }
A function declaration tells the compiler:
- the name of the function
- the return type
- the parameter types

It does not contain the body (the actual code). It ends with a semicolon.
Syntax: return_type function_name(parameter_list);
Example: int add(int a, int b); // function declaration

```
int add(int a, int b) // function definition
{
return a + b;
}
#include <stdio.h>
int add(int a, int b); // function declaration
int main() {
int result = add(3, 5);
printf("Sum = %d", result);
return 0;
}
int add(int a, int b) { // function definition
return a + b;
}
```

OutPut: Sum = 8

7 b) Explain the Function Arguments and Return statements in C.

Ans:

Function arguments are the values passed to a function when it is called. They allow the function to receive input data.

Why we use arguments?

- To send data from one function to another
- To make functions reusable
- To perform operations on different values

✅ **Example:**

```
#include <stdio.h>

int add(int a, int b)   // a and b are arguments (parameters)
{
   int sum = a + b;
   return sum;
}

int main()
{
   int result = add(5, 3);   // 5 and 3 are actual arguments
   printf("Sum = %d", result);
   return 0;
}
```

Types of Arguments:

1. **Actual Arguments** – Values passed in function call → `add(5, 3)`
2. **Formal Arguments** – Variables in function definition → `int a, int b`

Return Statement in C

The `return` statement is used to send a value back from a function to the calling function.

✅ **Why we use return?**

- To give output from a function
- To stop execution of the function
- To send result back to main()

✅ **Syntax:**

```
return value;
```

7 c) Explain the Function Prototypes with suitable Examples.

Ans:

A **function prototype** is a declaration of a function that tells the compiler:

- Function name
- Return type
- Number of parameters
- Data type of parameters

It is written **before the main() function**.

👉 It helps the compiler to check whether the function is called correctly or not.

Syntax:

return_type function_name(data_type parameter1, data_type parameter2);

Example Program

```
#include <stdio.h>

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

int main()
{
   int result;
   result = add(10, 5);   // Function call
   printf("Sum = %d", result);
   return 0;
}

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

7 d) Define recursion. Develop a C program and a function to compute factorial of a given number using recursion.

Ans:

Recursion is a programming technique where a function calls itself to solve a problem.

A recursive function typically contains:

1. Base Case – the condition under which the function stops calling itself.
2. Recursive Case – the part where the function calls itself with a smaller/simpler input.

Recursion is useful for solving problems that can be broken into smaller subproblems, such as factorial, Fibonacci series, tree traversal, etc.

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

```
// Recursive function to compute factorial
long long factorial(int n) {
if (n == 0 || n == 1) { // Base case
return 1;
} else {
return n * factorial(n - 1); // Recursive call
}
}

int main() {
int num;
printf("Enter a number: ");
scanf("%d", &num);

if (num < 0) {
printf("Factorial of a negative number doesn't exist.\n");
} else {
printf("Factorial of %d is %lld\n", num, factorial(num));
}

return 0;
}
```

Output: Enter a number: 5
Factorial of 5 is 120

8 a) Define Dynamic memory allocation. List and explain the different functions to handle dynamic memory allocation in C.

Ans:

Dynamic memory allocation refers to the process of allocating memory at runtime (when the program is running) as opposed to static memory allocation, where memory is allocated before execution starts (usually at compile time).

Dynamic memory allocation allows programs to be more flexible by using memory as needed and releasing it when it's no longer required, thus avoiding memory waste and ensuring efficient memory use.

In C, dynamic memory allocation is handled through a set of library functions provided by the stdlib.h library.

C provides several functions to allocate, deallocate, and manipulate dynamic memory. The functions are:

1.malloc() (Memory Allocation)

● Purpose: Allocates a block of memory of a specified size and returns a pointer to it.

Syntax:

```
void* malloc(size_t size);
```

● Description: The malloc() function allocates a specified number of bytes of memory and returns a pointer to the first byte of the allocated memory. The initial content of the memory block is not initialized (may contain garbage values).

○ If the memory allocation fails, malloc() returns NULL.

Example:

```
int *arr = (int*) malloc(5 * sizeof(int)); // Allocates
```

memory for 5 integers
if (arr == NULL) {

printf("Memory allocation failed.\n");
}
2. calloc() (Contiguous Allocation)
● Purpose: Allocates memory for an array of elements and initializes all the elements to zero.

Syntax:
void* calloc(size_t num_elements, size_t size_of_element);
● Description: calloc() allocates memory for an array of num_elements, each of size size_of_element. Unlike malloc(), calloc() also initializes the allocated memory to zero.
○ If the memory allocation fails, calloc() returns NULL.

Example:
int *arr = (int*) calloc(5, sizeof(int)); // Allocates memory for 5 integers and initializes to 0
if (arr == NULL) {
printf("Memory allocation failed.\n");
}
3. realloc() (Reallocate Memory)
● Purpose: Changes the size of a previously allocated memory block.

Syntax:
void* realloc(void* ptr, size_t new_size);
● Description: realloc() is used to resize a previously allocated memory block (pointed to by ptr). It can increase or decrease the size of the block.
○ If realloc() successfully resizes the memory block, it returns a pointer to the newly allocated memory. If it fails, it returns NULL, and the original memory is not freed.
○ If the new size is larger, the new memory is not initialized; the extra memory may contain garbage values.

Example:
int *arr = (int*) malloc(5 * sizeof(int)); // Initially

allocate memory for 5 integers
// Resize the allocated memory to hold 10 integers
arr = (int*) realloc(arr, 10 * sizeof(int));
if (arr == NULL) {
printf("Memory reallocation failed.\n");
}
4. free() (Deallocate Memory)
● Purpose: Frees previously allocated memory.

Syntax:
void free(void* ptr);
● Description: free() deallocates memory that was previously allocated using malloc(), calloc(), or realloc(). After calling free(), the pointer ptr is no longer valid, and using it can lead to undefined behavior (i.e., dangling pointer).

	○ After freeing memory, the pointer is not automatically set to NULL. It's a good practice to set the pointer to NULL after freeing it to avoid accidental use.
	
	Example:
	int *arr = (int*) malloc(5 * sizeof(int));
	// Use the allocated memory...
	free(arr); // Deallocates memory
	arr = NULL; // Avoid dangling pointer
	#include <stdio.h>
	#include <stdlib.h>
	int main() {
	// Allocate memory using malloc
	int *arr = (int*) malloc(5 * sizeof(int));
	if (arr == NULL) {
	printf("Memory allocation failed.\n");
	return 1; // Exit if memory allocation fails
	}
	
	// Initialize memory with values
	for (int i = 0; i < 5; i++) {
	arr[i] = i + 1;
	}
	// Print the values
	printf("Array values using malloc:\n");
	for (int i = 0; i < 5; i++) {
	printf("%d ", arr[i]);
	}
	printf("\n");
	
	// Reallocate memory using realloc to hold 10 integers
	arr = (int*) realloc(arr, 10 * sizeof(int));
	if (arr == NULL) {
	printf("Memory reallocation failed.\n");
	return 1;
	}
	// Initialize new memory elements
	for (int i = 5; i < 10; i++) {
	arr[i] = i + 1;
	}
	// Print the values after realloc
	printf("Array values after realloc:\n");
	for (int i = 0; i < 10; i++) {
	printf("%d ", arr[i]);
	}
	printf("\n");
	// Free the allocated memory
	free(arr);
	arr = NULL;
	return 0;}

8 b)	List the advantages of functions in programming.

Ans:
Advantages of Functions in Programming
1. Modularity
○ Functions break a large program into smaller, manageable parts.
2. Reusability
○ A function can be used multiple times in the program without rewriting code.
3. Improved Readability
○ Code becomes easier to read, understand, and maintain.
4. Easy Debugging
○ Errors can be isolated inside specific functions, making debugging simpler.
5. Abstraction
○ The user only needs to know what a function does, not how it works internally.
6. Avoids Code Duplication
○ Instead of repeating a block of code many times, the function is written once and reused.

7. Helps in Teamwork
○ Different people can work on different functions of the same program.

8 c) Explain TWO techniques of parameter passing to functions with suitable program segments.

Ans:
Two Techniques of Parameter Passing in C
1. Call by Value
Definition
In call by value, the function receives a copy of the actual argument.
Changes made inside the function do NOT affect the original variables.
✔ Program Segment (Call by Value)

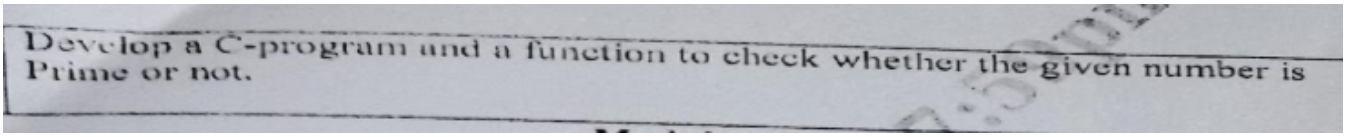
```
#include <stdio.h>
void change(int x) {
x = x + 10; // changes local copy only
printf("Inside function (x): %d\n", x);
}
int main() {
int a = 5;
change(a); // pass by value
printf("In main (a): %d\n", a); // original remains same
return 0;
}
```

2. Call by Reference
Definition
In call by reference, the address of the variable is passed to the function.
Thus, changes made inside the function directly modify the original variable.
✔ Program Segment (Call by Reference)

```
#include <stdio.h>
void change(int *x) {
*x = *x + 10; // changes actual variable using pointer
}
int main() {
```

```
int a = 5;
change(&a); // pass address of variable
printf("After function call (a): %d\n", a); // value modified
return 0;
}
```

8 d)


Develop a C-program and a function to check whether the given number is Prime or not.

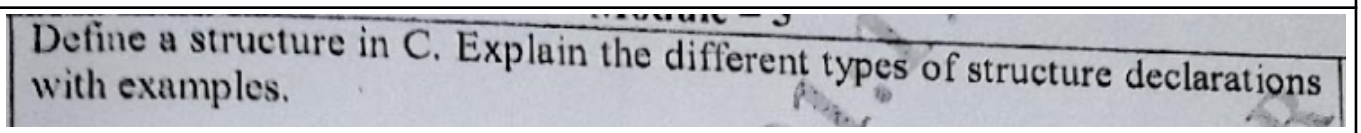
Ans:

```
#include <stdio.h>
// Function to check prime
int isPrime(int n) {
int i;
if (n <= 1)
return 0; // 0 and 1 are not prime
for (i = 2; i <= n/2; i++) {
if (n % i == 0)
return 0; // not prime
}
return 1; // prime
}
int main() {
int num;
printf("Enter a number: ");
scanf("%d", &num);
if (isPrime(num))
printf("%d is a Prime Number\n", num);
else
printf("%d is Not a Prime Number\n", num);
return 0;
}
```

O/P:

```
PS C:\Users\91819\OneDrive\Desktop\C_Programming>
Enter a number: 13
13 is a Prime Number
PS C:\Users\91819\OneDrive\Desktop\C_Programming>
Enter a number: 90
90 is Not a Prime Number
PS C:\Users\91819\OneDrive\Desktop\C_Programming>
```

9 a)


Define a structure in C. Explain the different types of structure declarations with examples.

Ans:

A structure in C is a user-defined data type that allows storing a collection of variables of different data types under a single name.

It is used to group related data items like student details, employee records, etc.
Types of Structure Declarations in C
1. Normal Structure Declaration (Tag + Members)
This is the most common method.
Syntax

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

2. Structure Declaration With Variable Creation
You can create structure variables at the time of declaring the structure.
Example

```
struct employee {
int id;
float salary;
} e1, e2; // structure variables created
```

3. Anonymous Structure (No Tag Name)
Structure has no name, only variables.
Example:

```
struct {
char name[20];
int age;
} person1, person2;
```

Here, we cannot create more variables later because the structure has no tag name.
4. Structure With typedef
Useful to create a short alias name for the structure.
Example

```
typedef struct person {
char name[20];
int age;
} P;
P p1, p2; // using typedef alias
```

5. Nested Structure (Structure inside Structure)
A structure contains another structure as a member.
Example

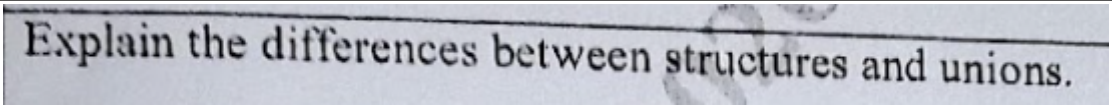
```
struct address {
char city[20];
int pincode;
};
struct student {
int roll;
struct address addr; // nested structure
};
```

6. Array of Structures
Used to store multiple records of the same type.
Example

```
struct student {
int roll;
float marks;
```

	}; struct student s[50]; // array of 50 students

9 b)	Explain how array of Structures is passed to a function
	Ans:
	
	In C, when we pass an **array of structures** to a function, we actually pass the **base address of the array** (just like normal arrays).
	
	That means the function receives a **pointer to structure**.
	
	Syntax
	void functionName(struct StructureName arr[], int size);
	OR
	void functionName(struct StructureName *arr, int size);
	Example Program
	#include <stdio.h>
	
	struct Student {
	int roll;
	float marks;
	};
	
	// Function to display student details
	void display(struct Student s[], int n) {
	int i;
	for(i = 0; i < n; i++) {
	printf("\nStudent %d\n", i+1);
	printf("Roll: %d\n", s[i].roll);
	printf("Marks: %.2f\n", s[i].marks);
	}
	}
	
	int main() {
	struct Student s[2] = {
	{1, 85.5},
	{2, 90.0}
	};
	
	display(s, 2); // Passing array of structures
	
	return 0;
	}

9 c)	 Ans:

	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

9 d) Develop a C program to store and display the Employee details using Structures.

Ans:

```
#include <stdio.h>

// Define structure
struct Employee {
    int emp_id;
    char name[50];
    float salary;
};

int main() {
    struct Employee e[3];   // Array of structures
    int i;

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

        printf("Employee ID: ");
        scanf("%d", &e[i].emp_id);

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

        printf("Salary: ");
        scanf("%f", &e[i].salary);
    }

    // Display employee details
    printf("\n---- Employee Details ----\n");
```

```
    for(i = 0; i < 3; i++) {
        printf("\nEmployee %d\n", i+1);
        printf("ID: %d\n", e[i].emp_id);
        printf("Name: %s\n", e[i].name);
        printf("Salary: %.2f\n", e[i].salary);
    }

    return 0;
}
```

10 a) Describe a method to compare two structure variables of the same type, and provide a simple example.

Ans:

Comparing Two Structure Variables in C
C does not allow direct comparison of structures using ==.
Therefore, to compare two structure variables of the same type, we must compare each member individually.

Method
1. Take two structure variables of the same structure type.
2. Compare each corresponding member using ==.
3. If all members are equal → structures are equal.
4. If any member is not equal → structures are different.

Example Program

```
#include <stdio.h>
#include <string.h>
struct student {
int roll;
char name[20];
float marks;
};
int main() {
struct student s1 = {1, "Raj", 85.5};
struct student s2 = {1, "Raj", 85.5};
// Compare each member
if (s1.roll == s2.roll &&
strcmp(s1.name, s2.name) == 0 &&
s1.marks == s2.marks)
{
printf("Both structures are equal.\n");
}
else {
printf("Structures are not equal.\n");
}
return 0;
}
```

10 b) Define Enumerated data type. Explain the declaration and access of enumerated data types with the help of C program segment.

Ans:
Enumerated Data Type (enum)

Definition
An enumerated data type, known as enum, is a user-defined data type in C that consists of a set of named integer constants. It is used to assign symbolic names to integral values, which improves code readability and maintainability.

Declaration of enum
Syntax
enum enum_name {
value1, value2, value3, ...
};
- By default, the first name has value 0, the next 1, and so on.
- You can also assign custom integer values.

Example
enum week { MON, TUE, WED, THU, FRI, SAT, SUN };

Accessing enum Variables
Steps
1. Declare an enum type
2. Create variables of that type
3. Assign or use the enum values

C Program:

```
#include <stdio.h>
enum week { MON = 1, TUE, WED, THU, FRI, SAT, SUN };
int main() {
enum week today;
today = WED; // accessing enum value
printf("Numeric value of today is: %d\n", today);
if (today == WED) {
printf("It is Wednesday!\n");
}
return 0;
}
```

10 c) What are Bit-fields and typedef in C. Explain with example.

Ans:

1 Bit-fields in C

A **bit-field** is a feature in C that allows us to allocate a specific number of bits to a structure member. It is mainly used to save memory when variables require only a few bits.

Syntax:

```
struct structure_name {
   data_type variable_name : number_of_bits;
};
```

Example:

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

```
struct Student {
    unsigned int age : 4;   // uses only 4 bits
    unsigned int marks : 7; // uses only 7 bits
};

int main() {
    struct Student s;
    s.age = 10;
    s.marks = 85;

    printf("Age = %d\n", s.age);
    printf("Marks = %d\n", s.marks);

    return 0;
}
```

2 typedef in C

`typedef` is a keyword used to create a new name (alias) for an existing data type.
It improves code readability.

Syntax:

```
typedef existing_data_type new_name;
```

Example:

```
#include <stdio.h>

typedef unsigned int uint;

int main() {
    uint num = 10;
    printf("Number = %u", num);
    return 0;
}
```

Example with Structure:

```
typedef struct {
    int id;
    char name[20];
} Student;
```

10 d) Develop a C program to access and modify the members of structures, in array of structures in C.

Ans:

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

// Define structure
struct Student {
```

```
    int roll;
    char name[50];
    float marks;
};

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

    // Accessing and storing values
    for(i = 0; i < 3; i++) {
        printf("\nEnter details of student %d\n", i+1);

        printf("Roll Number: ");
        scanf("%d", &s[i].roll);

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

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

    // Modifying structure member
    printf("\nModifying marks of student 1...\n");
    s[0].marks = 95.5;   // Modify marks of first student

    // Displaying data (Accessing members)
    printf("\nStudent Details:\n");
    for(i = 0; i < 3; i++) {
        printf("\nStudent %d\n", i+1);
        printf("Roll: %d\n", s[i].roll);
        printf("Name: %s\n", s[i].name);
        printf("Marks: %.2f\n", s[i].marks);
    }

    return 0;
}
```