CBCS SCHEME

USN 2 V 0 2 5 C I 0 1 6

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
		OR			

Q.4	a.	Explain the Iteration Statements with suitable Examples.	5	L2	
	b.	Explain the role of break and continue statements in C with suitable examples.	5	L2	CO
	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			
.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			
.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	CO
	d.	Develop a C program to access and modify the members of structures, in array of structures in C.	5	L3	CO

VTU Question Paper Solved - Dec-2025/Jan-2026

Programming in C

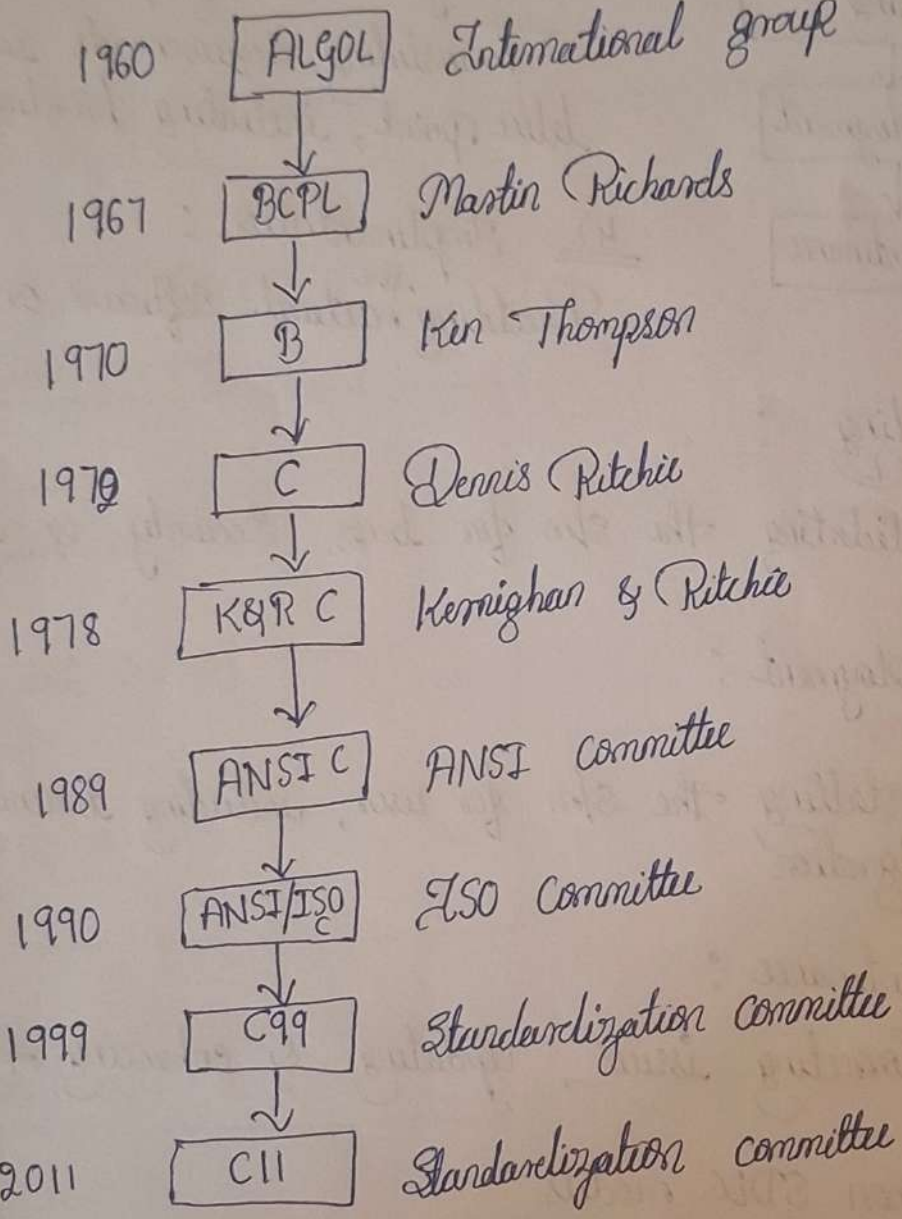
Model - 1

Q.1)

a) Define C. Explain History of C. - 5M

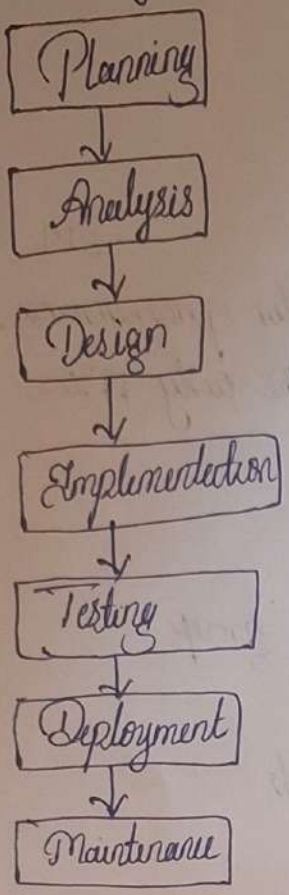
A: C is a general purpose, procedural computer programming language developed by Dennis Ritchie at Bell lab in the early 1970's.

History of C



b) Explain System Development Life cycle

A:



1) Planning :

Defining Project scope, goals, resources & cost.

2) Analysis :

Gathering detailed functional requirements from Stakeholders.

3) System Design :

Translating requirements into a technical blue print, including hardware & s/w architecture

4) Implementation :

Building the actual Software or s/m component

5) Testing :

Validating the s/m for bugs, Security & functionality.

6) Deployment :

Installing the s/m for user, including training & data migration

7) Maintenance :

Correcting issues, updating & enhancing the s/m.

Common SDLC models

* Waterfall
* Agile
* V-Model.

c) Explain General form of a C Program with example.

A:

```
//Short info of this pgm & other info
```
→ Documentation Section

```
#include <stdio.h>
#include <conio.h>
```
→ Link/File Inclusion Section

```
#define PI 3.14
```
→ Definition Section

```
void add();
int x=100;
```
→ Global declaration Section

```
int main()
{
  int a=10;
   :
  a=a+10;
}
```
→ main() function Section

```
void add()
```
→ Sub program Section

Example :-

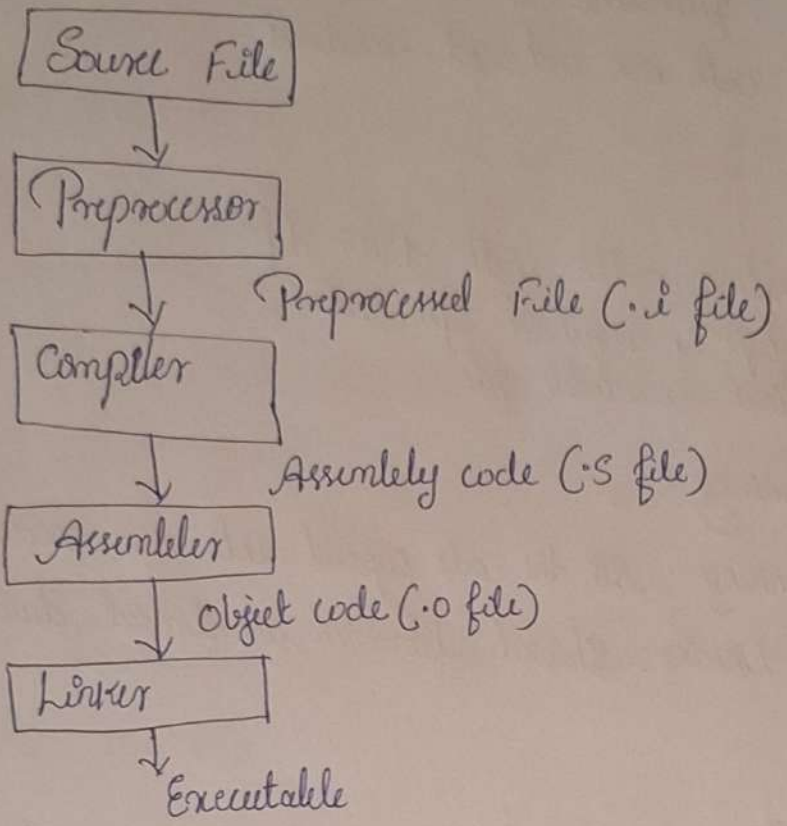
```
// Documentation
  /**
   * file : sum.c
   * Description : Program to find sum  */

#include <stdio.h>
#define  X 20
int  sum(int y)
int  main(void)
{
  int y= 55;
  printf("sum : %d", sum(y));
  return 0;
}

  int sum(int y)
{
  return y+x
}
```

Output :

Sum: 75

d) Explain Steps in compiling C program with suitable example.

A:



1) We need to create file with extension .c

Filename.c

2) Pre-processing
   - → Removes comments
   - → Expands macros
   - → Expands header file
   - → Handles conditional compilation
   - → O/P file : filename.i

3) Compiling
   - → Converts filename.i into assembly code
   - → O/P file : filename.s

4) Assembling
→ Converts assembly code into m/c code
→ O/p file : filename.O
→ Functions call are not yet resolved.

5) Linking
→ Links function calls with their defn.
→ Adds startup & required s/m code
→ Produces final executable file.

Types of Linking :
→ Static Linking : All the code copied into one executable code.
→ Dynamic Linking : Shared libraries are linked during execution.

Q.2>

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

A: Data type :

It is classification that specifies which type of value a variable can hold & what operations can be performed on it.
It determines the memory size allocated to the variable.

Primitive Data types in C

a) Integer (int) : Used to store whole numbers without decimals.
Ex: int age = 20;

b) Character (char) : Used to store a single character. It is enclosed in single quotes.
char grade = 'A';

c) Floating Point (float): Used to store real numbers with single precision decimal points.

Ex: float price = 99.9;

d) Double (double): Used for storing large floating point numbers with double precision

Ex: double distance = 12345.6789;

e) void: Represents the absence of a value or type, usually used with functions to indicate they do not return anything.

Q.2)

b) Define Operators. Explain the Increment & Decrement operators with Suitable Examples.

A: Operators are symbols or special characters used in programming languages to perform operations on variable & values.

Increment & Decrement Operators

These are unary operators that increase/decrease the value of a var by 1.

- Increment operator (++): Increases the value of a variable by 1
- Decrement operator (--): Decreases the value of a variable by 1

They can be used in two forms:

- Prefix form (++x or --x): The variable is incremented or decremented first, then the value is used
- Postfix form (x++ or x--): The current value of the variable is used first, then it is incremented / decremented

Ex:

```
int x=5;

int a = ++x;
int b = x++;

int y=5;
int c = --y;
int d = y--;
```

C) Show the evaluation of the following expressions with intermediate & final values.

i) x = a-b/3+c*2-1 when a=9, b=12, c=3

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

i) x = a-b/3+c*2-1

Soln Step: 1)

Substitute a=9, b=12, c=3

x = a-b/3+c*2-1

= 9-(12/3)+(3*2)-1

Step: 2)

Compute division & multiplication

*12/3 = 4

* 3*2 = 6

Now expression

x = 9-4+6-1

Step: 3)

Compute addition & substraction (left to right)

* 9-4 = 5

~~6-1=5~~

* 5+6 = 11

* 11-1 = 10

ii)

1) $10! = 10 || 5 < 4 \&\& 8$

2) Evaluate Relational operators:

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

$5 < 4 \rightarrow 0$ (false)

$\therefore$ Expression is now,

$0 \&\& 8$

d) Develop a C program to multiply, substract and division by taking to whole numbers. Choose suitable data types for variable.

A:

```
#include<stdio.h>
int main()
{
    int n1, n2;
    int sub, mult;
    float div;
    printf("Enter two whole numbers: ");
    scanf("%d %d", &n1, &n2);

    sub = n1 - n2;
    mult = n1 * n2;
    if (n2 != 0)
        div = (float) n1 / n2;
    else
        printf("Div by zero not possible\n");
    printf("Substraction : %d\n", sub);
    printf("Multiplication : %d\n", mult);
    if(num2 != 0)
    printf("Div : %2f\n", div);
    return 0;
}
```

Module - 2

Q.3)

a) Explain the Reading and Writing characters.

A: In C, characters are read & written using Std I/O functions.

Reading character

-> getchar() Reads the single character from Keyboard

-> Scanf("%c", &ch) Reads a character.

Writing character

-> Putchar() - Displays a single character.

-> printf("%c", ch) - prints a character

Ex:
```
#include<stdio.h>
int main()
{
  char ch;
  Printf("Enter a character:");
  ch = getchar();
  printf("You entered:");
  Putchar(ch);
  return 0;
}
```

b) With a suitable example, Explain formatted i/p & o/p Statements

A: Formatted I/O allows data to be read or printed in a specific format

Scanf() - Formatted input

Printf() - Formatted output

Common formed Specifier

%d - Integer

%f - float

%c - character

%s - String

Ex:

```
#include<stdio.h>
int main()
{
  int roll;
  float marks;
  printf("Enter roll number and marks:");
  scanf("%d %f", &roll, &marks);
  printf("Roll NO: %d\n", roll);
  printf("Marks: %f\n", marks);
  return 0;
}
```

C). List the conditional branching statement in C. Explain any two

A: Conditional Branching Statements

1) if
2) if-else
3) else-if ladder
4) switch
5) Conditional operator ?:

2) if-else

Syntax:

```
if (condition)
{
   = body of if
}
else
{
  = body of else
}
```

The if-else statement is used to execute one block of code if the condition is true & another block when it is false. It helps in making decision based on a given condition.

Ex:

```
#include <stdio.h>
int main()
{
  int num;
  printf("Enter a number: ");
  scanf("%d", &num);
  if(num%2 == 0)
    printf("Even number");
  else
    printf("Odd number");
  return 0;
}
```

4). Switch :-

Syntax:

```
switch(expression)
{
  case value1:
      //Statement
    break;
  case value2:
      //Statement
    break;
  default:
      //default Statements
}
```

The switch statement is used to select one block of code from multiple options based on the value of an expression. It compares the expression with different case values & executes the matching case.

code when

Ex:

```
#include <Stdio.h>
int main()
{
  int day;
  Printf("Enter a day (1-3):");
  Scanf("%d", &day)
  Switch(day)
  {
      Case 1:
          printf("monday");
          break;
      Case 2:
          printf("Tuesday");
          break;
      Case 3:
          printf("Wednesday");
          break;
      default:
          printf("Invalid day");
  }
    return 0;
}
```

d) Develop a C pgm to print floyd Δle for N rows (N>0) choose suitable control statement [for n = 4]

```
1
2 3
4 5 6
7 8 9 10
```

```
#include <Stdio.h>
int main()
{
  int n, i, j, num = 1;
  Printf("Enter a number of rows:");
```

```
    scanf("%d", &n);
    for(i=1; i<=n; i++)
    {
      for(j=1; j<=i; j++)
      {
        printf("%d", num);
        num++;
      }
    printf("\n");
  }
  return 0;
}
```

Output :-

```
Enter number of rows : 4
1
2  3
4  5  6
7  8  9  10
```

4>

a) Explain the iteration statements with Example.

Iteration statements are used to execute a block of code repeatedly until a condition becomes false

Types:

1. for
2. while
3. do-while

Ex:

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

Ex:

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

b)

i) break:

Terminates the loop or switch statement immediately.

Ex:

```
for (int i=1; i<=5; i++)
{
  if (i==3)
    break;
  printf("%d", i);
}
```

ii) Continue :

Skips the current iteration & continues with the next iteration

```
for (int i=1; i<=5; i++)
{
    if (i==3)
        continue;
    printf("%d", i);
}
```

C)

goto :

Transfers control to a labeled Statement

```
goto lable;
label:
printf("Hello");
```

return :

Exits from a function & returns value.

```
return 0;
```

Block Statement :

Group of Statements enclosed in { }

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

Quadratic Equation

$D = b^2 - 4ac$

```
#include<Stdio.h>
#include<math.h>
int main()
{
   float a, b, C, D, root1, root2;
   printf("Enter a,b,C:");
   Scanf("%f %f %f", &a, &b, &C);

   D = b*b - 4*a*C;

   if (D>0)
   {
      root1 = (-b + Sqrt(D))/(2*a);
      root2 = (-b - Sqrt(D))/(2*a);
      Printf("Roots are real and different \n");

    else if (D>0)
   {
      root1 = root2 = -b/(2*a);
      Printf("Roots are real & equal \n");

   }
     else
   {
      printf("Roots are imaginary \n");
   }
     return 0;
}
```

Module - 3

Q.5>

a)

Define Array & Declaration

Array : Collection of same data type elements stored in contigous memory location.

1D Array

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

2D Array

```
int b[2][2] = {{1,2}, {3,4}};
```

b)

Passing Array to functions

Arrays are passed by reference (address is passed)

```
void display (int arr[], int n)
{
    for (int i=0; i<n; i++)
    Printf ("%d", arr[i]);
}
```

call:

```
int a[3] = {1, 2, 3};
display (a, 3);
```

c)

Variable Length Array [VLA]

Array whose size is decided to runtime

```
int n;
scanf ("%d", &n);
int arr[n];
```

Difference

Static Array → size is fixed at compile time.

VLA → size decided at runtime.

d)

Transpose of Matrix

```
#include <stdio.h>
int main()
{
    int a[2][2], i, j;
    printf ("Enter a matrix:");
    for (i=0; i<2; i++)
        for (j=0; j<2; j++)
```

```
        scanf("%d", &a[i][j]);
    printf("Transpose :\n");
     for(i=0; i<2; i++)
     {
      for(j=0; j<2; j++)
       printf("%d", a[j][i]);
      printf("\n");
     }
     return 0;
}
```

Q.6)

a)

Pointer

Variable that stores address of another variable

Declaration :

```
int *p;
```

Initialization :

```
int a = 10;
p = &a;
```

Access array using pointer :

```
int arr[3] = {1, 2, 3}
int *p = arr;
printf("%d", *(p+1));
```

Two Pointer Operators

1) & → Address operator.
2) * → Dereference operator.

Ex:

```
int a=10;
int *p=&a;
printf("%d", *p);
```

C) Multiple Indirection

Pointer to pointer

```
int a=10;
int *p=&a;
int **q=&p;
printf("%d", **q);
```

d)

Add two numbers using pointers

```
#include <stdio.h>
int main()
{
    int a=5, b=10, sum;
    int *p=&a, *q=&b;

    sum = *p + *q;
    printf("sum = %d", sum);
    return 0;
}
```

Module - 4

Q.7)

a) Function Definition & Declaration

Function : Block of code that performs a specific task

Declaration

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

Definition :

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

Function Arguments & Return

Arguments :

Value passed to function

Return :

Sends value back to calling function

```
int sum(int a, int b)
{
   return a+b;
}
```

c) Function Prototype

Function prototype declares function before main().

Ex:

```
int add(int, int);
int main()
{
    int s = add(2,3);
}
```

d)

Recursion - Factorial

Recursion : Function calling itself

```
int fact (int n)
{
  if (n==0)
    return 1;
    return n * fact(n-1);
}
```

Q.8)

a) Dynamic Memory Allocation

Memory allocated at runtime using

1) malloc ()
2) calloc ()
3) realloc ()
4) free ()

Ex:

```
int *p = (int *) malloc (5 * sizeof(int));
free (p);
```

Advantages of functions

1) Code of Reusability
2) Reduce duplication
3) Easy debugging
4) Improves readability
5) Modular Programming.

c)

Two Parameter Passing Techniques

1). Call by value

Copy of value passed

```
void fun(int a)
{
   a=20;
}
```

2) Call by Reference

Address passed using Pointer

```
void fun(int *a)
{
   *a=20;
}
```

d) Prime or not

```
#include <stdio.h>
int n, i, flag=0;
Printf ("Enter a positive integer : ");
Scanf ("%d", &n);
if (n==0 || n==1)
  flag = 1;
for (i=2; i<=n/2; i++)
{
   if (n%. i ==0)
  {
     flag = 1;
     break;
  }
}
if (flag ==0)
Printf (%d is a prime number.", n);
else
 Printf ("%d is a not a prime number." n);
return 0;
}
```

Module - 5

Structure :

A structure in a C is a user-defined data type used to group variables of different data types under a single name.

Types of Structure declaration :

1) Structure declaration only

```
Struct Student
{
    int roll;
    char name[20];
};
```

2) Structure with variables

```
Struct Student
{
    int roll;
    char name[20];
} S1, S2;
```

3) Using typedef

```
typedef Struct
{
    int roll;
    char name[20];
} Student;
Student S1;
```

b) Passing Array of Structures to a function

An array of Structure is passed to a function by passing its name

Ex:

```
Struct Student
{
   int roll;
   char name[20];
};

void display (Struct Student S[], int n)
{
   for (int i=0; i<n ; i++)
  printf ("%d %s \n", S[i].roll, S[i].name);
}
```

Call:

```
Struct Student S[2];
displays (S, 2);
```

c)

Structure V/S Union

Structure	Union
1) Memory allocated for all members	Memory shared by all members
2) Size = Sum of all members	Size = Size of largest member
3) All members can be used Simultaneously	Only one member used at a time
4) Uses more memory	Saves memory

Program to Store and display Employee details

```
#include <Stdio.h>
 Struct Employee
{
    int id;
    char name[20];
     float Salary;
};

    int main()
   {
     Struct Employee e;
    printf("Enter ID, Name, Salary:");
    scanf("%d %s %f", &e.id, &e.name, &e.salary);
    printf("\n Employee Details :\n");
    printf("ID : %d \name : %s\n Salary : %2f ", e.id, e.name,
          e.salary);
    return 0;
    }
```

Q.10

a) Compare two Structure Variables

Structures cannot be compared directly using ==.
Each member must be compared individually

Ex:

```
Struct Student
{
   int roll;
   int marks;
};

Struct Student S1 = {1, 90};
Struct Student S2 = {1, 90};

if (S1.roll == S2.roll && S1.marks == S2.marks)
   printf ("Structures are equal");
else
   printf ("Not equal");
```

b)

Enumerated Data types

Enum is a user-defined data type that all assigns names to integer constants

Enum Day {MON, TUE, WED, THU, FRI} ~~SAT~~

```
#include <stdio.h>

enum Day{MON, TUE, WED};
int main()
{

  enum Day d;
    d = TUE;
  Printf("value : %d", d);
  return 0;
}
```

c)

Bit - field & typedef

Bit - field :

Used to allocate specific number of bits to structure members

```
Struct Data
 {
    unsigned int a : 3;
    unsigned int b : 5;
 };
```

typedef :

Creates a new name an existing data type.

```
typedef unsigned int unit;
 unit x = 10;
```

d)

Access & Modify Members in Array of Structures

```
#include<stdio.h>
  Struct Student
 {
   int roll;
   int marks;
};

 int main()
 {
    Struct Student S[2];
    for (int i=0; i<2; i++)
    {
       printf ("Enter the roll & marks :");
       scanf ("%d %d", &S[i].roll, &S[i].marks);
    }
     S[0].marks = 95;
     for (int i=0; i<2; i++)
   {
     printf ("%d %d \n", S[i].roll, S[i].marks);
   }
   return 0;
}
```