Model Question Paper- I

CBCS SCHEME

First/ Second Semester B.E Degree Examination, 2025-26

Programming in C (1BEIT105/205)

TIME: 03 Hours **Max.Marks:100**

Notes:

1. *Answer any FIVE full questions, choosing at least ONE question from each MODULE*
2. *M: Marks, L: Bloom's level, C: Course outcomes.*
3.

		Module - 1	M	L	C
Q.1	a	Define data type. Explain primitive data types supported in C language with example.	6	L2	CO1
	b	Explain the general form of a C program with example.	8	L2	CO1
	c	Develop a C program to multiply, subtract and division by taking two whole numbers. Choose suitable datatypes for variables.	6	L3	CO5
		OR			
Q.2	a	What is variable? Explain the rules to construct variables. Classify the following as valid/invalid Identifiers. i) num2 ii) $num1 iii) +add iv) a_2 v) 199_space vi) _apple vii) #12	6	L2	CO1
	b	Show the evaluation of the following expressions with intermediate and final values. i) x = a - b/3 + c * 2 - 1 when a = 9, b = 12, c = 3 ii) 10! = 10 || 5 < 4 && 8.	8	L2	CO1
	c	Develop C program which takes as input p, t, r and calculates the simple interest. Choose suitable data types for variables.	6	L3	CO5
		Module – 2			
Q.3	a	With a suitable example, explain formatted input and output statements.	6	L2	CO1
	b	List the conditional branching statements in 'C'. Explain any two with suitable examples.	6	L2	CO2
	c	Develop a C program to print Floyd's triangle for N rows (N >0). Choose suitable control statements. [for n=4] 1 2 3 4 5 6 7 8 9 10	8	L3	CO5
		OR			
Q.4	a	Explain Jump Statements, Expression Statements, Block Statements with suitable examples.	6	L2	CO1
	b	Explain the role of break and continue statements in C with suitable examples.	6	L2	CO2
	c	Develop a simple calculator program in C language for simple operations like addition, subtraction, multiplication and division. Choose suitable selection statement.	8	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.	8	L2	CO2
	b	Define variable length array. Illustrate how variable length array is different	6	L2	CO2

Model Question Paper- I

		from static array.			
	c	Develop a C program to swap the values of two integer variables using pointers.	6	L3	CO2
		OR			
Q.6	a	Define a pointer. How do you declare and initialize pointers in C. Explain accessing array elements using a pointer.	8	L1	CO2
	b	Show with a suitable program, how a single dimensional array can be passed to a function.	6	L2	CO2
	c	Develop a C program that reads N integers, stores them in an array and calculates the sum of all array elements.	6	L3	CO2
		Module – 4			
Q.7	a	Define function. Explain the syntax of function definition and function declaration with a simple example.	6	L2	CO3
	b	Define dynamic memory allocation. List and explain the different functions to handle dynamic memory allocation in C.	6	L2	CO3
	c	Define recursion. Develop a C program and a function to compute factorial of a given number using recursion.	8	L3	CO3
		OR			
Q.8	a	List the advantages of functions in programming. With suitable program, how pointer is initialized to a function for call/reference?	6	L2	CO3
	b	Explain TWO techniques of parameter passing to functions with suitable program segments.	6	L2	CO3
	c	Develop a C-program and a function to check whether the given number is prime or not.	8	L3	CO3
		Module – 5			
Q.9	a	Define a structure in C. Explain the different types of structure declarations with examples.	6	L2	CO4
	b	Describe a method to compare two structure variables of the same type, and provide a simple example.	6	L2	CO4
	c	Define a structure with a name **student**. Develop a C program that uses a structure named **student.** The program should read and display the details of 'N' students, compute the average marks of the class, and identify the students who have scored marks above and below the class average.	8	L3	CO4
		OR			
Q.10	a	Compare the structure and union in terms of syntax, storage and uses/applicability.	6	L2	CO4
	b	Define Enumerated data type. Explain the declaration and access of enumerated data types with the help of C program segment.	6	L2	CO4
	c	Develop a C program to access and modify the members of structures, in array of structures in C.	8	L3	CO4

AY: 2025-26

Department: Computer Science & Engineering **VTU Model Question Paper**
Subject with Sub. Code:Programming using C (1BEIT205)
Semester / Division: IIFaculty Name: Prof. Vijet Swadi

Q. No	Solution and Scheme	Marks
	Module - 1	
Q1.a	Defination of data type — 1 mark	06
	A data type specifies the type of data that a variable can store	
	Primitive data types — 1×5 = 5 marks	
	(1) int	
	·Used to store whole numbers	
	·Typically occupies 2 to 4 bytes (depending on system)	
	·Can store +ve or -ve values	
	eg: 20, -10	
	(2) float	
	·Used to store decimal numbers	
	·Single precision (4 bytes)	
	·Stores up to 6-7 decimal digits	
	eg:- 99.5, 12.2	
	(3) char	
	·Used to store single character	
	·It occupies 1 byte	
	·Stores character in ASCII form	
	eg: 'A', 'b', '8', '@'	
	(4) double	
	·Used to store large decimal number	
	·Occupies 8 bytes	
	·stores up to 15 to 16 decimal digits	
	eg:- 45678.8805	

Q. No	Solution and Scheme	Marks
	(5) void	
	·Represents absence of value	
	·Used in functions that do not return any value	
	eg:- void display();	
1.6.	General Form of C program	08
	It includes 6 sections — 6 x 1 = 6 marks	
	(1) Documentation section	
	·This section contains information about the program or comments	
	· Comments are ignored by compiler	
	· Comments are written using // or /* */	
	(2) Link section	
	· Includes header files	
	· Uses #include directives to insert header file	
	· Provides access to standard library functions	
	(3) Defination section	
	· Global variables are defined in this section using #define	
	(4) Global declaration section	
	·Declares global variables and function prototypes	
	·Variables declared here can be used throughout the program	
	(5) main() function section	
	·It is the starting point of program execution	
	·Every C program must have one main() function	

	structure:- main()
	{
	//declaration part
	// executable statements
	}	

Q. No	Solution and Scheme	Marks

(c) Subprogram section

- Contains user defined functions
- Written after main() or before main()

Example program —— 2 marks

```
/* Program to add two numbers */   //documentation section
#include<stdio.h>   //link section
# define VALUE 10   //defination section
int num1, num2;   // global declaration
int main()
{
   int sum;          //local declaration
   num1 = 5;
   num2 = VALUE;
   sum= num1 + num2;  //executable statement
   printf("Sum= %d", sum);
}
```

1.c C program to multiply, subtract and division by taking two whole numbers 06

```
#include<stdio.h>
int main()
{
  int num1, num2;
  int multiply, subtract;
  float division;
  printf("Enter two numbers \n");
  scanf("%d%d", &num1, &num2);
  multiply = num1 * num2;
  subtract = num1 - num2;
  if (num2!=0)
      division = (float) num1 /num2;
```

Q. No	Solution and Scheme	Marks

	else
	{
	printf("Division by zero \n");
	return 0;
	}
	printf(".Multiplication = %d \n", multiply);
	printf(" Subtraction = %d \n", subtract);
	printf(" Division = %d. ", division);
	}	

2. a.	Variable defination — 1 mark A variable is a named memory location. used to store data in a program Rules to construct variables 1 × 4 = 4 marks · It must begin with letter or underscore · made up of letter, digit and underscore only · Case sensitive · keywords cannot be used · First 32 characters are significant Valid/invalid identifiers — 0.5 × 2 = 1 mark (i) num2 — valid (ii) $num1 → invalid (iii) +add — invalid (iv) a_2 → valid (v) 199_space — invalid (vi) _apple → valid (vii) #12 — invalid	06
2. b.	Evaluation of expression — 2 × 4 = 8 marks (i) x = a-b/3+c*2-1 when a=9, b=12, C=3 = 9-12/3+3*2-1 = 9-4+6-1 = 5+6-1 = 11-1 = 10	08

Q. No	Solution and Scheme	Marks
	(ii) 10! = 10 || 5 < 4 && 8 ⇒ 0 || 0 && 8 ⇒ 0 || 0 ⇒ 0	

2. c.	Program to calculate simple interest
	#include <stdio.h>
	int main()
	{
	Float p, t, r;
	Float si;
	printF("Entere Principal, time and rate of interest \n");
	scanf("%F%F%F", &p, &t, &r);
	si=(p*t*r)/100;
	printF("Simple interest = %f \n", si);
	}
		06

	<u>Module - 2</u>	
3. a.	Formatted input, output statements 2×3 = 6 marks (i) Formatted output - printF() · Used to display output on screen · Allows Formatting using format specifiers syntax: printF("Format string", variable1, variable2); format specifiers %d — integers %c — character %F — Float %s — string %lF — double	06

Q. No	Solution and Scheme	Marks

eg:-
```
int age=18;
Float marks= 85.5
printf("age = %d", age);
printf("marks= %f", marks);
```

(2) Formatted input - scanf()

·Used to read input from user

·Uses format specifiers

Syntax:-
```
scanf("format_string", &variables);
```

eg:-
```
int num;
Float price;
printf("Enter a number and price \n");
scanf("%d %f", &num, &price);
```

3. b. Conditional branching statements in C — 2 marks

(1) if

(2) if - else

(3) else - if ladder

(4) Switch

(5) conditional operator

if statement — 2 marks

· It is used to check the condition

· If condition is true, a block of statements will be executed

· It is called one way branching statement

Syntax:-
```
if (condition)
{
  // statement block
}
```

Q. No	Solution and Scheme	Marks

	eg: int age = 18;
	if (age >= 18)
	printf("Eligible to vote");
	(2) <u>if - else statement</u>
	· Used to execute one block of code, if the condition is true and another block, if the condition is false
	Syntax: if (condition)
	{
	// true block statements
	}
	else
	{
	// false block statements
	}
	eg:- int a=10;
	if (a%2==0)
	printf("%d is even", a);
	else
	printf("%d is odd", a);	

3. c.	C program to print Floyd's triangle
	#include<stdio.h>
	int main()
	{
	int n,i,j;
	int num=1;
	printf("Enter no. of rows (N>0): ");
	scanf("%d", &n);
	if(n>0)
	{
		08

Q. No	Solution and Scheme	Marks

	for(i=1; i<=n; i++)
	{
	for(j=1; j<=i; j++)
	{
	printf("%d", num);
	num++;
	}
	printf("\n");
	}
	}
	else
	{
	printf("Invalid input");
	}
	}	

Q4.a.	Jump statements — 2 marks
	Jump statements are used to transfer control from one part of the program to another
	eg:- break
	continue
	go to
	return
	Expression statement — 2 marks
	• An expression statement is an expression followed by semicolon(;)
	• It performs some computation or assignment
	eg:- a=5;
	x=y+10;
	block statement — 2 marks
	• A block statement is a group of statements enclosed within curly braces {}
	• It is also called compound statement
		06

Q. No	Solution and Scheme	Marks

```
eg:- main()
      {
        //statements are enclosed here
      }
      if(a>b)
       {
         //statements are enclosed her
       }
```

4.c. <u>Simple calculator program</u> 08

```
#include <stdio.h>
int main()
{
  float num1, num2, result;
  int choice;
  printf("Simple calculator");
  printf(" 1. Addition \n 2. subtraction \n
            3. Multiplication 4. Division \n);
  printf("Enter your choice(1-4): \n");
  scanf("%d", &choice);
  printf("Enter two numbers \n");
  scanf("%f %f", &num1, &num2);
  switch(choice)
  {
    case 1: result = num1 + num2;
            printf("Result = %f", result);
    case 2: result = num1 - num2;
            printf("Result = %f", result);
    case 3: result = num1 * num2;
            printf("Result = %f", result);
```

Q. No	Solution and Scheme	Marks

```
        case 4: if(num2!=0)
                {
                  result=num1/num2;
                  printf(" Result = %f", result);
                }
                else{
                    ~~divisso~~ printf("Division by zero");
                    }
      }
    }
```

Module - 3

5.a. Array defination — 2 marks   08

An array is a collection of elements of the same data type, stored in contiguous memory locations. Elements are accessed using common name with an index

Single dimensional array:

Initialization: data_type array_name[size];
This declares an array of size 5
eg: int marks[5];

Initialization:
int marks[5] = {10, 20, 30, 40, 50};
int marks[] = {10, 20, 30};
int marks [5] = {10, 20};

Two dimensional array:

declaration: datatype array_name[rows] [columns];
eg: int matrix[2] [3];
This declares a 2 x 3 size matrix

Q. No	Solution and Scheme	Marks

Initialization: (1) Row-wise

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

(2) Single list

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

5.b. Variable length array Def^n — 06

A variable length array is an array, whose size is determined at run time instead of compile time

Syntax:

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

Here the size of array depends on the value entered by the user.

Eg:-

```
#include <stdio.h>
int main()
{
   int n, i;
   printf("Enter the size of array");
   scanf("%d", &n);
   int arr[n];
   for(i=0; i<n; i++)
   {
     arr[i]=i+1;
   }
   for(i=0; i<n; i++)
   {
     printf("%d", arr[i]);
   }
}
```

- In VLA size is determined at run time, but in static array size is given at compile time
- VLA can use variable, but static array must use constant to represent size

Q. No	Solution and Scheme	Marks
5. c.	Program to swap the values of two integer variables using pointers	06

```
#include<stdio.h>
//Function to swap values using pointers
void swap (int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}

int main()
{
  int num1, num2;
  printf("Enter two integers:");
  scanf("%d %d", &num1, &num2);
  printf("Before swapping: num1 = %d, num2 = %d
          \n", num1, num2);
  // passing address of variables
  swap(&num1, &num2);
  printf("After swapping: num1 = %d, num2 = %d
          \n", num1, num2);
}
```

Q. No	Solution and Scheme	Marks
6.a.	Defination of pointer	08

Pointer is a variable, that stores the address of another variable.

Declaration of pointer

The general systax to declare pointer is

```
data_type *pointer_name;
```

```
eg: int *p
    Float *fp;
```

Here p can store the address of an integer variable

Initialization of pointer

A pointer is initialized by assigning it the address of a variable using the address-of operator (&)

```
eg: int a=10;
    int *p;
    p = &a;  // p is initialized with the
                address of a
```

Accessing array elements using a pointer

The name of an array represents the base address of the array. So pointer can be used to access array elements

```
Eg:- #include<stdio.h>
     int main()
     {
       int arr[5]={10, 20, 30, 40, 50};
       int *p=arr;
       For(i=0; i<5; i++)
       {
        printf("%d", *(p+i));
       }
     }
```

Q. No	Solution and Scheme	Marks
6.b	Passing a single dimensional array to a function in C	06

In C, array can be passed to a function by passing its base address. When we pass an array to a function, only the address of the first element is passed, not the entire array. Therefore any changes made inside the function affect the original array.

eg:
```
#include<stdio.h>
    void display (int arr[], int n);
    int main()
    {
      int a[5] = {10, 20, 30, 40, 50};
      // passing array to function
      display (a, 5);
      return 0;
    }
    void display (int arr[], int n)
    {
      int i;
      for (int i=0; i<n; i++)
       {
       {printf ("%d", arr[i]);
    }
```

Q. No	Solution and Scheme	Marks
6.c.	Program to read n integers and to find sum	06

```
#include<stdio.h>
int main()
{
  int n, i, sum=0; arr[100];
  printf ("Enter the numbers of elements");
  scanf(" %d", &n);
  printf (" Enter
```

Q. No	Solution and Scheme	Marks

```
if (n>100 || n<=0)
{
  printf("Invalid size ");
  return 1;
}
printf("Enter %d integers: \n",n);
for (i=0; i<n; i++)
 {
   scanf("%d", &arr[i]);
 }

 for(i=0; i<n; i++)
 {
   sum = sum + arr[i];
 }
 printf("Sum of array elements = %d\n",
                                   sum);
}
```

Module - 4

7. a. Function defination

A Function is a self contained block of code that performs a specific task.

Function defination :

Syntax:.

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

where return type is type of value returned by called function to calling function.

function name is the valid identifier name

parameter list specifies zero or more arguments sent by calling function.

Q. No	Solution and Scheme	Marks

```
eg:: int add (int a, int b)
     {
       int sum;
       sum = a+b;
       return sum;
     }
```

Function declaration

A function declaration tells the compiler

- Function nare
- Return type
- Numbes and type of parameters

It is written before main() or at the begining of the program.

Syntax:

```
return_type function_nare (parameter_list)
eg: int add (int, int);
```

Example program:

```
#include <stdio.h>
int add (int, int);
int main ()
{
  int result;
  result = add (10, 20);
  printf ("Sum = %d \n", result);
}
int add (int a, int b)
{
  return a+b;
}
```

Q. No	Solution and Scheme	Marks

7. b.	Dynamic memory allocation (DMA)
	It is the process of allocating the memory at run time from heap area using standard library functions. It allows programs to request memory as needed instead of fixing the size at compile time
	Functions used for DMA
	malloc(), calloc(), realloc(), free()
	(i) malloc()
	It allocates specified number of bytes and return a pointer to the allocated memory.
	Syntax:. ptr = (type *) malloc (size_in. bytes);
	eg:. int *ptr;
	ptr = (int *) malloc (5 * sizeof(int));
	(ii) calloc()
	Allocates memory for multiple elements and initialise them to zero.
	Syntax:.
	ptr = (type *) calloc(number_of_elements, size_of_each.elements);
	eg:. int *ptr = (int *) calloc(5, sizeof(int));
	(iii) realloc()
	It is used to resize previously allocated memory.
	Syntax:. ptr = (type *) realloc(ptr, new_size);
	eg:. ptr = (int *) realloc(ptr, 10 * sizeof(int));
		06

Q. No	Solution and Scheme	Marks

(iv) free()

Frees the dynamically allocated memory

Syntax :- free(ptr);

It releases memory back to heap.

7.c. Recursion defination 08

Recursion is a process in which a function calls itself repeatedly until a specified condition is satisfied. A recursive function has two cases

(1) Base case - stops the recursion

(2) Recursive case - Function calls itself

Program to find factorial using recursion

```
#include<stdio.h>
long int Factorial (int n);
int main()
{
  int num;
  long int result;
  printf("Enter a number");
  scanf("%d", &num);
  if(num<0)
   {
    printf("Factorial of -ve no. is not defined");
   }
  else
   {
    result = Factorial(num);
    printf("Factorial of %d = %d \n", num, result);
   }
 }
```

Q. No	Solution and Scheme	Marks
Q.8.a	Advantages of Functions in programming	06

1. Modularity
2. Code reusabality
3. Easy debugging
4. Improves redabality
5. Reduces code duplication
6. Supports structured programming.

A function pointer is a pointer, that stores the address of a function. It allows a function to be called using pointer,
following program illustrates how pointer is initialized to a function for call/reference

```
#include<stdio.h>
int add(int a, int b)
 {
  return a+b;
 }
int main()
{
   int (*fptr)(int, int);
   Fptr = add;
   int result = Fptr(10, 20);
   printf("Sum = %d\n", result);
 }
```

Q. No	Solution and Scheme	Marks
8.b.	Parameter passing techniques	06

In C, there are two techniques, of passing parameters to functions

(1) call by value

(2) call by reference

Q. No	Solution and Scheme	Marks

(1) Call by value

In call by value, a copy of actual argcement is passed to the function. Changes made inside the Function do not affect the original variables.

eg:
```
#include <stdio.h>
void swap (int a, int b)
{
  int temp;
  temp = a;
  a=b;
  b=temp;
  printf("Inside function: a = %d , b=%d \n",
              a,b);
}
int main()
{
  int x =10 . y=20;
  swap (x,y);
   printf ("In main : x= %d, y= %d \n",
              x,y);
}
```

(2) Call by reference

In call by reference, the address of variables is passed to the function. Changes made inside the function affect the original variables

eg:
```
#include <stdio.h>
Void swap (int *a, int *b);
{
  int temp;
  temp = *a;
```

Q. No	Solution and Scheme	Marks

	*a = *b;
	*b = temp;
	}
	main()
	{
	int x=10, y=20;
	swap(&x, &y);
	printf("After swapping x=%d, y=%d", x, y);
	}	

8.c.	Program to check whether a numbers is prime or not
	#include<stdio.h>
	int isprime(int n);
	int main()
	{
	int num;
	printf("Enter a number \n");
	scanf("%d", &num);
	if (isPrime(num));
	printf("%d is a prime number \n", num);
	else
	printf("%d is not a prime number", num);
	}
	int isPrime(int n)
	{
	int i;
	if (n<=1)
	return 0;
		08

Q. No	Solution and Scheme	Marks

```
    for(i=2 ; i<=n/2 : i++)
    {
       if (n%i==0)
          return 0;
    }
 }
```

Module -5

9.a. Define a structure in C. Explain the different types of structure declaration with example. 06

A structure in C is a user defined data type that allows grouping of variables of different data types under a single name. It is declared using the struct keyword.

```
Syntax:- struct structure_name
          {
            data_type member1;
            data_type member2;
               :
          };
```

Types of structure declaration

(i) With structure variable

Structure is defined and variables are declared immediately

```
eg: struct Student
     {
       int roll;
       char name[20];
       float marks;
     } s1, s2;
```

Q. No	Solution and Scheme	Marks

	(2) Structure is defined first, then variables declared.
	eg:- struct student
	{
	int roll;
	char name[20];
	float marks;
	};
	struct Student s1 , s2 ;	

	(3) Anonymous structure. i.e structure without name. eg:- ~~structure~~ struct { int roll; float marks; } s1 ;	

	(4) Using typedef with structure
	eg:- typdef struct
	{
	int roll;
	char name[20];
	float marks;
	} Student;
	Student s1, s2 ;	

9.b.	Comparing two structure variables in C Structure variables cannot be compared directly using == operator. So to compare two structures of the same type, we must compare each member individually.	

Q. No	Solution and Scheme	Marks

method to compare two structure

1. Ensure both structures are of the same type.
2. Compare each member one by one
3. If all members are equal → structure are not equal
4. If any members differs ⇒ structures are not equal.

eg:-
```
#include <stdio.h>
#include <string.h>
struct Student
{
  int roll;
  char name[20];
  float marks;
};
int main()
{
  struct Student s1 = {1, "Rahul", 85.5};
  struct Student s2 = {1, "Rahul", 85.5};
  if (s1.roll == s2.roll &&
      strcmp(s1.name, s2.name)==0 &&
      s1.marks == s2.marks)
    {
     printf("Structures are equal \n");
    }
  else {
     printf("structures are not equal \n");
    }
}
```

Q. No	Solution and Scheme	Marks
9.c.	Program to compute average marks and marks above and below average	

```
#include<stdio.h>
#define MAX 100
struct student
{
   int roll;
   char name[50];
   float marks;
 };
int main()
{
   struct student s[MAX];
   int n, i;
   Float sum=0, average;
   printf("Enter number of students");
   scanf("%d", &n);
   for(i=0; i<n; i++)
   {
      printf("Enter the details of %d student",
                                     i+1);
      printf("Roll No: ");
      scanf("%d", &s[i].roll);
      printf("Name ");
      scanf("%s", s[i].name);
      printf(" Marks ");
      scanf("%f", &s[i].marks);
      sum += s[i].marks;
   }
   average = sum/n;
```

Q. No	Solution and Scheme	Marks

```
printf("\n -- student details --\n");
for(i=0 ; i<n ; i++)
{
    printf("Roll no:%d", s[i].roll);
    printf("Name: %s", s[i].name);
    printf("Marks: %f", s[i].marks);
}
printf("Class average = %f", average);

}
```

10.a. Comparison between structure & union — 06

(1) Difference in syntax

Structure:	Union:

structure student
{
int roll;
Float marks;
char grade;
};
	Union Data
	{
	int i;
	Float f;
	char ch;
	};

(2) Difference in storage

Structure	Union
(i) Separate memory for each member	(i) Shared memory for each member
(ii) Size is sum of sizes of all members	(ii) Size of largest member
(iii) All members can store values simultaneously	(iii) only one member can store value at a time

(3) Difference in uses / accessibality

Structure	Union
(i) Used when all members are required	(i) Used when only one member is needed at a time

Q. No	Solution and Scheme	Marks

structure	Union
·Used when all members are require	·Used when only one member is needed at a time
·Consumes memory	·Saves memory

10.b. Enumarated data type — 6.

It is a user defined data type, that consists of a set of named integers constants.

declaration: syntax:

```
enum enum_name
{
  constant1,
  constant2,
  constant3.
    :
};
```

- By default, the first constant has value 0
- The next constant increase automatically by 1.

eg:-

```
enum day { mon, tue, wed, thu, fri, sat};
```

Here mon → 0, tue → 1, wed → 2 and so on.

declaring enum variables

```
enum day today;
```

example program

```
#include <stdio.h>
enum week{mon=1, tue, wed, thu, fri, sun};
int main()
{
 enum week day;
 day = wed;
```

Q. No	Solution and Scheme	Marks

```
    printf("Enum value of WED = %d", day);
    if (day = WED)
       printf(" Today is wednesday \n");
  }
```

Q. No	Solution and Scheme	Marks
10.c.	Program to access & modify members in an array of structure	08

```
#include <stdio.h>
# define MAX 100
 struct student
 {
   int roll;
   char name [50];
   Float marks;
 };
int main()
 {
   struct student s[MAX];
   int n, i, choice, r;
   printf ("Enter number of students ");
   scanf ("%d", &n);
   For (i=0; i<n ; i++)
   {
     printf(" Enter the details of student %d",
                i+1);
     printf(" Roll no ");
     scanf(" %d", & s[i].roll);
     printf(" Name ");
     scanf("%s", &s [i]. name);
     printf(" Marks ");
     scanf ("%f", &s[i].marks
   }
```

Q. No	Solution and Scheme	Marks

```
printf("\n - student details - \n);
for(i=0; i<n ; i++)
{
  printf("\n Roll no: %d", s[i].roll);
  printf("\n Name : %s", s[i].name);
  printf("\n Marks : %f", s[i].marks);
}
printf("\n Enter a roll_no to modify
          marks");
scanf("%d", &r);
 for(i=0; i<n; i++)
 {
   if(s[i].roll == r)
   {
     printf("Enter new marks");
     scanf("%f", &s[i].marks);
     printf("Marks updated successfully");
     break;
   }
 }
}
```

(Vijet Suadi)

26/2/26

27/2/26.

Q. No	Solution and Scheme	Marks