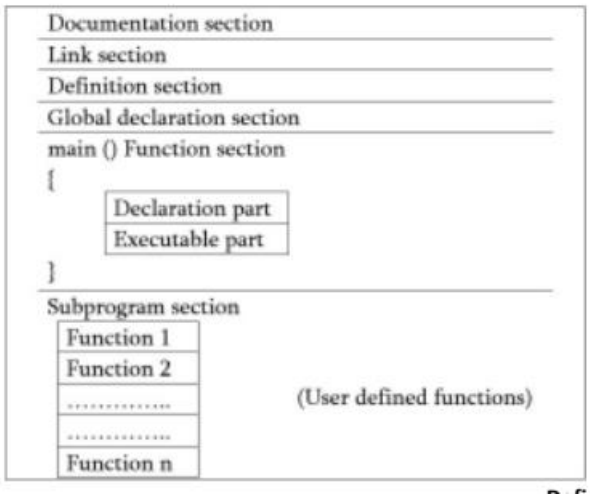
VTU Question Paper Solution & Scheme
Principles of Programming
June/July 2024

Sub:	PRINCIPLES OF PROGRAMMING USING C					Sub Code:	BPOPS203	Branch:CSE, CS-DS
Date:	22-07-2024	Duration:	3 hrs	Max Marks:	100	Sem/Sec:	First Year & I,J,K,L	

Answer any FIVE FULL Questionns

1. a.	**Define Computer. Explain the various types of Computer.** A computer is an electronic device that processes data according to a set of instructions called a program. It performs a wide range of tasks such as calculations, data processing, and automated reasoning. Computers can execute complex algorithms, handle vast amounts of data, and perform repetitive tasks with high speed and accuracy. **Types of Computer:** Computers can be categorized into several types based on their size, power, and intended use. Here are some common types: **1. Supercomputers** Description: Extremely powerful computers used for complex simulations, large-scale calculations, and scientific research. Examples: IBM's Blue Gene, Cray's Titan. **2. Mainframes** Description: Large, powerful computers designed for high-volume data processing and handling multiple users simultaneously. Examples: IBM zSeries, Unisys ClearPath. **3. Minicomputers** Description: Mid-sized computers that are less powerful than mainframes but can still handle multiple tasks and users. They are used in industries and businesses. Examples: Digital Equipment Corporation (DEC) PDP-11, VAX series. **4. Microcomputers (Personal Computers)** Description: Small, general-purpose computers designed for individual use. They come in various forms such as desktops, laptops, and tablets. Examples: Dell Inspiron, Apple MacBook, Lenovo ThinkPad. **5. Smartphones:** Description: Smartphones are general purpose computers that are also capable of making phone calls, have powerful processor usually with multiple cores like quad-cores. It supports gigabytes of main memory Examples: Apple, Samsung, Nokia **6. Embedded Computers:** Description: Specialized computers integrated into other devices to control or manage specific functions. They are designed to perform dedicated tasks within a larger system. Examples: Microcontrollers in appliances, automotive control systems, smart thermostats.	[10]
1.b.	**Explain the basic structures of C program in detail. Write a simple program to demonstrate the components in the structure of C.** Documentation Section: This section is used to write Problem, file name, developer, date etc in	[10]

	comment lines within /*....*/ or separate line comments may start with // . Compiler ignores this section. Documentation enhances the readability of a program. · Link section : To include header and library files whose in-built functions 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 - 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 */ print("Enter radius of the circle : "); /* Execution part*/ scanf("%f", &r); area=PI*r*r; /* using symbolic constant PI */ print("Area of circle = %0.3f square unit\n", area); return (0); } **Output:** Enter radius of the circle: 5.0 Area of circle = 78.540 square unit	
2.a.	**Explain scanf() and printf () functions in C language with syntax and example.** printf() – Library function for formatted output: Output data can be written on to a standard output device using the library function	[8]

printf(). The print statement provides certain features that can be effectively exploited to control the alignment and spacing of printouts on the terminals. The general form of print statement is:
printf ("control string", arg1,arg2, ..., argn);
scanf() – Library function for formatted input Formatted input refers to an input data that has been arranged in a particular format. Input data can be entered into the computer from a standard input device by means of the C library function scanf. In general terms, scanf function is written as
scanf ("control string", &arg1, &arg2, ..., &argn);
The control string specifies the field format in which the data is to be entered and the arguments arg1,arg2.....,argn specify the address of locations where the data is stored.
when scanf() is used to read string input it stops reading when it encounters

Example:

```
#include<stdio.h>
int main(){
int number;
printf("enter a number:");
scanf("%d",&number);
printf("cube of number is:%d ",number*number*number);
return 0;
}
```

Output:

```
enter a number:5
cube of number is:125
```

2. b. **What is a variable? Explain rules for constructing variable in C. Give example for valid and invalid variable.** [6]

A variable in programming, including C, is a named storage location in memory that holds a value which can be modified during program execution. The variable name acts as an identifier to access and manipulate the stored value.

Rules for Constructing Variables in C

1. Naming Conventions:

Start with a Letter or Underscore: Variable names must begin with a letter (a-z, A-Z) or an underscore (_). They cannot start with a digit.
Followed by Letters, Digits, or Underscores: After the initial letter or underscore, variable names can contain letters, digits (0-9), and underscores.
Case Sensitivity: Variable names in C are case-sensitive. For example, myVariable, MyVariable, and MYVARIABLE are considered different variables.

2. Length: While C does not specify a maximum length for variable names, it's a good practice to keep them reasonably short and meaningful.
3. No Reserved Keywords: Variable names cannot be the same as C reserved keywords (e.g., int, float, return, etc.).
4. No Special Characters: Variable names cannot contain special characters like @, #, $, or spaces.

Example:

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

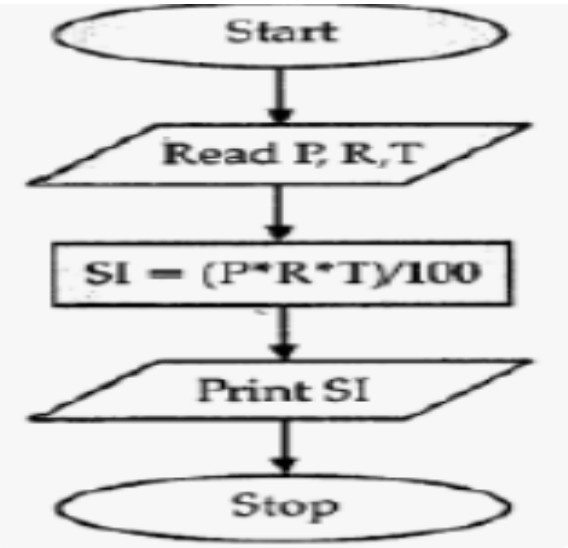
```
int main() {
int myVar = 10;      // Valid
float _var = 5.5;    // Valid
int var1 = 20;       // Valid
int variableName = 30; // Valid
printf("myVar: %d\n", myVar);
printf("_var: %f\n", _var);
printf("var1: %d\n", var1);
printf("variableName: %d\n", variableName);
return 0;
}
```

Output:

```
myVar: 10
_var: 5.500000
var1: 20
variableName: 30
```

2.c. **Illustrate the flowchart and write a C program which takes as input p, t, v and compute the simple interest and display result.** [6]

Flowchart for Computing Simple Interest




```
#include <stdio.h>
int main() {
float principal, time, rate, simpleInterest;
printf("Enter the principal amount (p): ");
scanf("%f", &principal);
printf("Enter the time period in years (t): ");
scanf("%f", &time);
printf("Enter the rate of interest per annum (v): ");
scanf("%f", &rate);
simpleInterest = (principal * time * rate) / 100;
printf("Simple Interest = %.2f\n", simpleInterest);
return 0;
```

	} **Output:** Enter the principal amount (p): 1000.0 Enter the time period in years (t): 5.0 Enter the rate of interest per annum (v): 7.5 Simple Interest = 375.00	
3. a.	**Explain the following operators in 'C.** **i) Relational** Relational operators are used to compare two values. They return a boolean result (either true or false). == (Equal to): Checks if two values are equal. != (Not equal to): Checks if two values are not equal. > (Greater than): Checks if the left value is greater than the right value. < (Less than): Checks if the left value is less than the right value. >= (Greater than or equal to): Checks if the left value is greater than or equal to the right value. <= (Less than or equal to): Checks if the left value is less than or equal to the right value. **ii) Logical** Logical operators are used to perform logical operations, combining multiple conditions. They return a boolean result. **&&** (Logical AND): Returns true if both operands are true. || (Logical OR): Returns true if at least one of the operands is true. **!** (Logical NOT): Returns true if the operand is false, and false if the operand is true. **iii) Conditional** The conditional operator is a ternary operator that allows for a compact form of an if-else statement. It is often used for assigning a value based on a condition. ? : (Conditional or Ternary Operator): result = (condition) ? value_if_true : value_if_false; eg: int max = (a > b) ? a : b; **iv) Bitwise.** Bitwise operators perform operations on the binary representations of integers. They operate on individual bits. & (Bitwise AND): Performs a bitwise AND operation. The result has bits set to 1 where both operands have bits set to 1. | (Bitwise OR): Performs a bitwise OR operation. The result has bits set to 1 where at least one operand has bits set to 1. ^ (Bitwise XOR): Performs a bitwise XOR operation. The result has bits set to 1 where the corresponding bits of the operands are different. ~ (Bitwise NOT): Performs a bitwise NOT operation, inverting all the bits of the operand. << (Left Shift): Shifts the bits of the left operand to the left by the number of positions specified by the right operand. Zeros are shifted in from the right. >> (Right Shift): Shifts the bits of the left operand to the right by the number of positions specified by the right operand. For unsigned integers, zeros are shifted in from the left; for signed integers, the sign bit may be shifted in.	[8]
3. b.	**Explain For Loop statement with syntax and example program.** For loop provides a concise way of writing the loop structure. Unlike other looping statements, a for loop consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping. **Syntax:** for (initialization Statement; test Expression; update Statement) { // statements inside the body of loop	[6]

}

Example Program:

```
#include<stdio.h>
int main(){
int i=1,number=0;
printf("Enter a number: ");
scanf("%d",&number);
for(i=1;i<=10;i++){
printf("%d \n",(number*i));
}
return 0;
}
```

Output:

```
Enter a number: 2
2
4
6
8
10
12
14
16
18
20
```

3.c. **Write a C program to simulate simple calculator that performs arithmetic operations using switch statement. Error message should be displayed if any attempt is made to divide by zero.** [6]

Program:

```
int main()
{
int num1, num2;
int result;
char op;
printf("Enter the operator \n");
scanf("%c",&op);
printf("Enter two integers :");
scanf("%d%d", &num1,&num2);
if (op == '+')
{
result=num1+num2;
}
else if (op == '-')
{
result=num1-num2;
}
else if (op == '*')
{
result=num1*num2;
}
```

```
else if (op == '/')
{
if (num2 == 0)
{
printf("Divide by zero error \n");
return (1);
}
else
{
 result=num1/num2;
}
}
else if (op == '%')
{
if (num2 == 0)
{
printf("Divide by zero error \n");
return (2);
}
else
{
result=num1%num2;
}
}
else
{
printf("Invalid operator...\n");
return (3);
}
printf("%d %c %d  = %d\n", num1, op, num2, result);
return 0;
}
```

Output:

```
Enter the operator: /
Enter two integers: 2 0
Divide by zero error
Enter the operator: /
Enter two integers: 2 2
2/2=1
```

4. a. **Explain if, if-else, nested if and cascaded if-else statements with syntax and example.** [8]

Simple if:

In C programming, if statement is used to execute a block of code if a specific condition is true.

Syntax:

```
if (condition) {
   // code to be executed if condition is true
}
```

Example:

```
#include <stdio.h>
int main() {
```

```
 int x = 10;
if (x > 5) {
printf("x is greater than 5\n");
}
return 0;
}
```

Output:

x is greater than 5

if-else

if-else statement is used to execute a block of code if a specific condition is true or the else block if condition is false.

Syntax:

```
#include <stdio.h>
int main() {
int x = 10;
if (x > 20) {
printf("x is greater than 20\n");
} else {
printf("x is less than or equal to 20\n");
}
 return 0;
}
```

Output:

x is greater than 20

Nested if statement:

A nested if in C is if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement. Yes, C allow us to nested if statements within if statements, i.e, we can place if statement inside another if statement.

Syntax:

```
if (condition1)
{
  // Executes when condition1 is true
  if (condition_2)
  {
    // statement 1
  }
  else
  {
       // Statement 2
   }
}
else {
    if (condition_3)
  {
    // statement 3
  }
  else
  {
```

```
        // Statement 4
    }
}
```

Program:

```
#include <stdio.h>
int main()
{
int i = 10;
if (i == 10) {
if (i < 15)
printf("i is smaller than 15\n");
if (i < 12)
printf("i is smaller than 12 too\n");
else
printf("i is greater than 15");
}
else {
if (i == 20) {
if (i < 22)
printf("i is smaller than 22 too\n");
else
printf("i is greater than 25");
}
}
return 0;
}
```

Output:

```
i is smaller than 15
i is smaller than 12 too
```

4.b. **Write a C program that takes three coefficient (a, b, c) to calculate roots of quadratic equation, print all possible root with appropriate messages for a set of coefficients.** [6]

Program:

```
#include<stdio.h>
#include<math.h>
int main()
{
float a,b,c,desc;
float r1,r2;
float realpart,imgpart;
printf("Enter the coefficients - a, b and c :");
scanf("%f%f%f",&a,&b,&c);
if(a==0)          {
printf("Coefficient of 'a' cannot be zero....\n");
printf("It's a linear equation\n");
return 1;
}
Desc = (b * b) - (4.0 * a * c);
if(desc==0)
{
```

```
printf("The roots are real and equal\n");
r1 = r2 = (-b) / (2.0*a);
printf("The two roots are r1=r2=%f\n",r1);
}
else if(desc>0)
{
printf("The roots are real and distinct\n");
r1 = (-b+sqrt(desc)) / (2.0*a);
r2 = (-b-sqrt(desc)) / (2.0*a);
printf("The roots are r1=%f and r2=%f\n",r1,r2);
}
else
{
printf("The roots are imaginary\n");
realpart = (-b) / (2.0*a);
imgpart = sqrt(-desc) / (2.0*a);
printf("The roorts are \n");
printf("r1 = %f + i %f \n",realpart,imgpart);
printf("r2 = %f - i %f \n",realpart,imgpart);
}
return 0;
}
```

Output:

```
Enter the coefficients of a, b and c :1 1 1
The roots are imaginary
The roots are
r1=-0.500000 + i 0.866025
r2=-0.500000 - i 0.866025
```

4.c. [6]

Explain Break and Continue statements are used to control the flow of loops in C programming.

Break Statement:

- The break statement is used to terminate the loop immediately.
- When a break statement is encountered, the loop is exited, and the control jumps to the statement immediately following the loop.

Continue Statement:

- The continue statement is used to skip the current iteration of the loop and move to the next iteration.
- When a continue statement is encountered, the current iteration is terminated, and the control jumps to the beginning of the loop for the next iteration.

While Loop:

- Break: Terminates the while loop when a certain condition is met.

```
while (condition) {
   // code
   if (condition) {
      break; // terminates the loop
   }
   // code
}
```

- Continue: Skips the current iteration and moves to the next iteration when a certain condition is met.

```
while (condition) {
```

```
 // code
    if (condition) {
        continue; // skips the current iteration
    }
    // code
}
```

Do-While Loop:

- Break: Terminates the do-while loop when a certain condition is met.

```
do {
    // code
    if (condition) {
        break; // terminates the loop
    }
    // code
} while (condition);
```

- Continue: Skips the current iteration and moves to the next iteration when a certain condition is met.

```
do {
    // code
    if (condition) {
        continue; // skips the current iteration
    }
    // code
} while (condition);
```

For Loop:

- Break: Terminates the for loop when a certain condition is met.

```
for (initialization; condition; increment/decrement) {
    // code
    if (condition) {
        break; // terminates the loop
    }
    // code
}
```

- Continue: Skips the current iteration and moves to the next iteration when a certain condition is met.

```
for (initialization; condition; increment/decrement) {
    // code
    if (condition) {
        continue; // skips the current iteration
    }
    // code
}
```

Example:

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break; // terminates the loop when i reaches 5
    }
    printf("%d ", i);
}
```

Output:

0 1 2 3 4

Example:

```
for (int i = 0; i < 10; i++) {
if (i == 5) {
continue; // skips the iteration when i reaches 5
}
printf("%d ", i);
}
```

Output:

0 1 2 3 4 6 7 8 9

5. a. **Define function. Explain categories of user defined functions.** [10]

A function is a block of code that performs a specific task and can be called multiple times from different parts of a program. It takes input parameters, processes them, and returns output.
There are four types of user-defined functions divided on the basis of arguments they accept and the value they return:

- Function with no arguments and no return value
- Function with no arguments and a return value
- Function with arguments and no return value
- Function with arguments and with return value

1. Function with No Arguments and No Return Value

Functions that have no arguments and no return values. Such functions can either be used to display information or to perform any task on global variables.

Example:

```
#include <stdio.h>
 void sum()
{
int x, y;
printf("Enter x and y\n");
scanf("%d %d", &x, &y);
printf("Sum of %d and %d is: %d", x, y, x + y);
}
int main()
{
sum();
return 0;
}
```

Output

Enter x and y
Sum of 4195522 and 0 is: 4195522

2. Function with No Arguments and With Return Value

Functions that have no arguments but have some return values. Such functions are used to perform specific operations and return their value.

Example:

```
#include <stdio.h>
int sum()
{
int x, y, s = 0;
printf("Enter x and y\n");
scanf("%d %d", &x, &y);
s = x + y;
return s;
}
int main()
```

{
printf("Sum of x and y is %d", sum());
return 0;
}

Output

Enter x and y
Sum of x and y is 4195536

3. Function With Arguments and No Return Value

Functions that have arguments but no return values. Such functions are used to display or perform some operations on given arguments.

```
#include <stdio.h>
void sum(int x, int y)
{
printf("Sum of %d and %d is: %d", x, y, x + y);
}
int main()
{
int x, y;
printf("Enter x and y\n");
scanf("%d %d", &x, &y);
sum(x, y);
return 0;
}
```

Output:

Enter x and y
Sum of 0 and 0 is: 0

4. Function With Arguments and With Return Value

Functions that have arguments and some return value. These functions are used to perform specific operations on the given arguments and return their values to the user.

```
#include <stdio.h>
 int sum(int x, int y) { return x + y; }
 int main()
{
int x, y;
printf("Enter x and y\n");
scanf("%d %d", &x, &y);
printf("Sum of %d and %d is: %d", x, y, sum(x, y));
return 0;
}
```

Output:

Enter x and y
Sum of 0 and 0 is: 0

5.b. **Define two-dimension array. Write a C program to multiply 2 matrixes by ensuring their multiplication compatibility.** [10]

A Two-Dimensional array or 2D array in C is an array that has exactly two dimensions. They can be visualized in the form of rows and columns organized in a two-dimensional plane.

Syntax:

array_name[size1] [size2];

Program:

```
#include<stdio.h>
int main()
{
int a[10][10],b[10][10],c[10][10];
int m,n,p,q;
```

```
int i,j,k;
printf("Enter the order of matrix A :");
scanf("%d%d",&m,&n);
printf("Enter the order of matrix B:");
scanf("%d%d",&p,&q);
if(n!=p)
{
printf("Number of columns of Matrix A is not equal to number of rows of matrix B\n");
printf("Matrix Multiplication not possible....\n");
return (1);
}
printf("\nEnter %d elements into matrix A : ", m*n);
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("\nThe  matrix A is ---\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
Input the elements into Matrix B
printf("\nEnter %d elements into matrix B : ", p*q);
for (i=0;i<p;i++)
{
for (j=0;j<q;j++)
{
scanf("%d",&b[i][j]);
}
}
printf("\nThe matrix B is ---\n");
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
{
printf("%d\t",b[i][j]);
}
printf("\n");
}
for(i=0;i<m;i++)
{
for(j=0;j<q;j++)
{
c[i][j] = 0;
for(k=0;k<n;k++)
{
c[i][j] = c[i][j] + (a[i][k] * b[k][j]);
}
}
}
printf("\nThe product matrix is ---\n\n");
```

	for(i=0;i<m;i++) { for(j=0;j<q;j++) { printf("%d\t",c[i][j]); } printf("\n"); } return 0; } **Output:** Enter the order of matrix A :2 2 Enter the order of matrix B:2 2 Enter 4 elements into matrix A : 1 2 3 4 The matrix A is --- 1 2 3 4 Enter 4 elements into matrix B : 1 2 3 4 The matrix B is --- 1 2 3 4 The product matrix is --- 7 10 15 22	
6.a.	**Explain function call, function definition and function prototype with syntax and example for each.** A function is a block of code that performs a specific task. It has a unique name and it is reusable i.e. it can be called from any part of a program. Parameter or argument passing to function is optional. It is optional to return a value to the calling program. Function which is not returning any value from function, their return type is void. While using function, three things are important. **Function Declaration** Like variables, all the functions must be declared before they are used. The function declaration is also known as function prototype or function signature. It consists of four parts, Function type (return type) Function name Parameter list Terminating semicolon **Syntax:** FunctionName (Argument1, Argument2,...); **Example:** int sum(int , int); In this example, function return type is int, name of function is sum, 2 parameters are passed to function and both are integer. **Function Definition** Function Definition is also called function implementation. It has mainly two parts. Function header: It is the same as function declaration but with argument name. Function body: It is actual logic or coding of the function. **Function Call**	[10]

Function is invoked from main function or other function that is known as function call.
Function can be called by simply using a function name followed by a list of actual argument enclosed in parentheses.

Syntax:

```
FunctionName (Argument1, Argument2,...) {
        Statement1;
        Statement2;
        Statement3;
}
```

Program:

```
#include <stdio.h>
int sum(int, int);
void main() {
int a, b, ans;
scanf("%d%d", &a, &b);
ans = sum(a, b);
printf("Answer = %d", ans);
}
int sum (int x, int y) {
int result;
result = x + y;
return (result);
}
```

Output:

```
2 3
Answer=5
```

6.b. **Write a C Program to implement Binary Search for integers.** [5]

```
#include <stdio.h>
int main()
{
 int a[20];
 int n,i,j,temp,key;
 int first,mid,last;
 printf("Enter the size of the array :");
 scanf("%d",&n);
printf("Enter %d elements :",n);
 for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
printf("The elements of the array before sorting is ----\n");
for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}
for(i=0;i<n-1;i++)
{
for(j=0;j<n-1-i;j++)
{
```

```
if(a[j]>a[j+1])
{
temp=a[j];
a[j]=a[j+1];
a[j+1]=temp;
}
}
}
printf("\n\nThe sorted array is ---\n");
for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}
printf("\n\nEnter the element to be searched :");
scanf("%d",&key);
first=0;
last=n-1;
while(first <= last)
{
mid=(first+last)/2;
if(key==a[mid])
{
printf("\nThe element %d is found at location %d\n",key,mid+1);
return (0);
}
else if (key < a[mid])
{
last = mid-1;
}
else
{
first = mid+1;
}
}
printf("\nThe element %d is not found in the array\n",key);
return (1);
}
```

6.c. **What is Recursion? Write a C program to compute factorial of number Using recursion.** [5]

Recursion is the process of a function calling itself repeatedly till the given condition is satisfied. A function that calls itself directly or indirectly is called a recursive function and such kind of function calls are called recursive calls.

Program:

```
#include<stdio.h>
int multiplyNumbers(int n);
int main() {
int n;
printf("Enter a positive integer: ");
scanf("%d",&n);
printf("Factorial of %d = %ld", n, multiplyNumbers(n));
```

	return 0; } int multiplyNumbers(int n) { if (n>=1) return n*multiplyNumbers(n-1); else return 1; } **Output:** Enter a positive integer: 6 Factorial of 6 = 720	
7.a.	**Define string. Explain any four string manipulating functions with example.** A String in C programming is a sequence of characters terminated with a null character '\0'. The C String is stored as an array of characters. The difference between a character array and a C string is that the string in C is terminated with a unique character '\0'. **1. strcat() Function** The strcat() function in C is used for string concatenation. It will append a copy of the source string to the end of the destination string. Syntax: char* strcat(char* dest, const char* src); Program: #include <stdio.h> int main() { char dest[50] = "This is an"; char src[50] = " example"; printf("dest Before: %s\n", dest); strcat(dest, src); printf("dest After: %s", dest); return 0; } Output: dest Before: This is an dest After: This is an example **2. strlen() Function** The strlen() function calculates the length of a given string. It doesn't count the null character '\0'. **Syntax:** int strlen(const char *str); **Program:** #include <stdio.h> #include <string.h> int main() { char str[] = "GodisGreat"; size_t length = strlen(str); printf("String: %s\n", str); printf("Length: %zu\n", length); return 0; } Output:	[10]

String: GodisGreat
Length: 10

3. strcmp() Function

The strcmp() is a built-in library function in C. This function takes two strings as arguments and compares these two strings lexicographically.

Syntax:

```
int strcmp(const char *str1, const char *str2);
```

Program:

```
#include <stdio.h>
#include <string.h>
int main()
{
char str1[] = "God";
char str2[] = "Is";
char str3[] = "God";
int result1 = strcmp(str1, str2);
int result2 = strcmp(str2, str3);
int result3 = strcmp(str1, str1);
printf("Comparison of str1 and str2: %d\n", result1);
printf("Comparison of str2 and str3: %d\n", result2);
printf("Comparison of str1 and str1: %d\n", result3);
return 0;
}
```

Output:

Comparison of str1 and str2: 1
Comparison of str2 and str3: -1
Comparison of str1 and str1: 0

4. strcpy

The strcpy() is a standard library function in C and is used to copy one string to another.

Syntax:

```
char* strcpy(char* dest, const char* src);
```

Program:

```
#include <stdio.h>
#include <string.h>
int main()
{
char source[] = "GodIsGood";
char dest[20];
strcpy(dest, source);
printf("Source: %s\n", source);
printf("Destination: %s\n", dest);
return 0;
}
```

Output:

Source: GodIsGood
Destination: GodIsGood

7.b. Write a C program to concatenate two strings without using built-in function strcat( ). [5]

Program:

```
#include <stdio.h>
void concatenateStrings(char str1[], char str2[]) {
int i, j;
```

```
for (i = 0; str1[i] != '\0'; i++);
for (j = 0; str2[j] != '\0'; j++) {
str1[i + j] = str2[j];
}
str1[i + j] = '\0';
}
int main() {
char str1[100], str2[100];
printf("Enter first string: ");
scanf("%s", str1);
printf("Enter second string: ");
scanf("%s", str2);
concatenateStrings(str1, str2);
printf("Concatenated string: %s\n", str1);
return 0;
}
```

Output:

```
Enter first string: Hello
Enter second string: World
Concatenated string: HelloWorld
```

7.c. **Explain string unformatted input/output functions with example.** [5]

Unformatted I/O functions are used only for character data type or character array/string and cannot be used for any other datatype. These functions are used to read single input from the user at the console and it allows displaying the value at the console.

The following unformatted I/O functions will be discussed in this section-

getch():getch() function reads a single character from the keyboard by the user but doesn't display that character on the console screen and immediately returned without pressing enter key. This function is declared in conio.h(header file). getch() is also used for hold the screen

getche():getche() function reads a single character from the keyboard by the user and displays it on the console screen and immediately returns without pressing the enter key.

getchar():The getchar() function is used to read only a first single character from the keyboard whether multiple characters is typed by the user and this function reads one character at one time until and unless the enter key is pressed.

putchar():The putchar() function is used to display a single character at a time by passing that character directly to it or by passing a variable that has already stored a character.

gets():gets() function reads a group of characters or strings from the keyboard by the user and these characters get stored in a character array.

puts():puts() function is used to display a group of characters or strings which is already stored in a character array.

putch():putch() function is used to display a single character which is given by the user and that character prints at the current cursor location.

Program:

```
#include <conio.h>
#include <stdio.h>
int main()
{
 char ch;
 printf("Enter any character:\n ");
ch = getch();
```

	printf("\nEntered character is: "); putch(ch); return 0; } **Output:** Enter any character: Entered character is: d	
8.a.	**Define pointer. Explain pointer variable declaration and initialization with suitable example.** A pointer is defined as a derived data type that can store the address of other C variables or a memory location. We can access and manipulate the data stored in that memory location using pointers. **1. Pointer Declaration** In pointer declaration, we only declare the pointer but do not initialize it. To declare a pointer, we use the (*) dereference operator before its name. **Example** int *ptr; The pointer declared here will point to some random memory address as it is not initialized. Such pointers are called wild pointers. **2. Pointer Initialization** Pointer initialization is the process where we assign some initial value to the pointer variable. We generally use the (&: ampersand) addressof operator to get the memory address of a variable and then store it in the pointer variable. Example int var = 10; int * ptr; ptr = &var; We can also declare and initialize the pointer in a single step. This method is called pointer definition as the pointer is declared and initialized at the same time. **Example** int *ptr = &var; **Program:** #include <stdio.h> void point() { int var = 10; int* ptr; ptr = &var; printf("Value at ptr = %p \n", ptr); printf("Value at var = %d \n", var); printf("Value at *ptr = %d \n", *ptr); } int main() { point(); return 0; } **Output:** Value at ptr = 0x7ffca84068dc	[8]

8.b.	Value at var = 10 Value at *ptr = 10 **Explain pass by value and pass by address with example.** **Pass by value:** The values of actual parameters are copied to the function's formal parameters. There are two copies of parameters stored in different memory locations. One is the original copy and the other is the function copy. Any changes made inside functions are not reflected in the actual parameters of the caller. **Program:** #include <stdio.h> void swapx(int x, int y); int main() { int a = 10, b = 20; swapx(a, b); printf("In the Caller:\na = %d b = %d\n", a, b); return 0; } void swapx(int x, int y) { int t; t = x; x = y; y = t; printf("Inside Function:\nx = %d y = %d\n", x, y); } **Output:** Inside Function: x = 20 y = 10 In the Caller: a = 10 b = 20 **Pass by Address:** The address of the actual parameters is passed to the function as the formal parameters. Both the actual and formal parameters refer to the same locations. Any changes made inside the function are actually reflected in the actual parameters of the caller. **Program**: #include <stdio.h> void swapx(int*, int*); int main() { int a = 10, b = 20; swapx(&a, &b); printf("Inside the Caller:\na = %d b = %d\n", a, b); return 0; } void swapx(int* x, int* y) { int t; t = *x; *x = *y;	[4]

```
*y = t;
printf("Inside the Function:\nx = %d y = %d\n", *x, *y);}
```

Output:

```
Inside the Function:
x = 20 y = 10
Inside the Caller:
a = 20 b = 10
```

8.c. **Write a C program using pointers to compute sum, mean, standard deviation of all elements stored in an array of n real numbers.** [8]

Program:

```
#include <stdio.h>
#include <math.h>
 int main()
{ int i, n;
float a[10],sum,mean,var,sd;
float *p;
printf("Enter Number of elements:");
scanf("%d", &n);
printf("Enter %d numbers :",n);
 p = a;
for (i=0;i<n;i++)
{
scanf("%f", p);
p++;
}
sum = mean = var = sd = 0.0;
 p = a;
 for(i=0;i<n;i++)
 {
 sum = sum + (*p);
 p++;
 }
 mean = sum / (float) n;
 p = a;
 for(i=0;i<n;i++)
 {
 var = var + pow( (*p - mean),2);
 p++;
 }
 var = var / (float) n;
 sd = sqrt(var);
 printf("Sum = %f\n", sum);
 printf("Mean = %f\n", mean);
 printf("Standard Deviation = %f\n", sd);
 return 0;
 }
```

Output :

```
Enter Number of elements:3
Enter 3 numbers :55 88 22
Sum = 165.000000
```

	Mean = 55.000000 Standard Deviation = 26.944387	
9.a.	**Explain structure declaration and how structure member are accessed with Example.** The structure in C is a user-defined data type that can be used to group items of possibly different types into a single type. The struct keyword is used to define the structure in the C programming language. **Structure Declaration Syntax:** struct structure_name { data_type member_name1; data_type member_name1; .... }; We can access structure members by using the (.) dot operator. **Syntax:** structure_name.member1; strcuture_name.member2; **Program:** #include <stdio.h> struct str1 { int i; char c; float f; char s[30]; }; struct str2 { int ii; char cc; float ff; } var; int main() { struct str1 var1 = { 1, 'A', 1.00, "GodIsGreat" }, var2; struct str2 var3 = { .ff = 5.00, .ii = 5, .cc = 'a' }; var2 = var1; printf("Struct 1:\n\ti = %d, c = %c, f = %f, s = %s\n",var1.i, var1.c, var1.f, var1.s); printf("Struct 2:\n\ti = %d, c = %c, f = %f, s = %s\n",var2.i, var2.c, var2.f, var2.s); printf("Struct 3\n\ti = %d, c = %c, f = %f\n", var3.ii, var3.cc, var3.ff); return 0; } **Output:** Struct 1: i = 1, c = A, f = 1.000000, s = GodIsGreat Struct 2: i = 1, c = A, f = 1.000000, s = GodIsGreat Struct 3 i = 5, c = a, f = 5.000000	[10]
9.b.	**Implement a structure to read, write and compute average marks and the students scoring above and below average of class N students.**	[10]

Program:

```
#include<stdio.h>
struct student
{
int id;
char name[20];
float sub[6];
float avg;
};
int main()
{
struct student s[20];
float sum=0;
int i,j,n;
printf("Enter the number of records :");
scanf("%d",&n);
printf("Enter %d student details...\n",n);
for(i=0;i<n;i++)
{
printf("\n\nEnter student ID, name :");    // Student ID
scanf("%d%s",&s[i].id, s[i].name);
printf("Enter 6 subject marks :");
for (j=0;j<6;j++)
{
scanf("%f", &s[i].sub[j]);
}}
for(i=0;i<n;i++)
{
sum=0;
for (j=0;j<6;j++)
{
sum = sum + s[i].sub[j];
}
s[i].avg = sum / 6;
}
printf("Students scoring above the average marks....\n");
printf("\n\nID\tName\tAverage\n\n");
for(i=0;i<n;i++)
{
if(s[i].avg>=35.0)
printf("%d\t%s\t%f\n",s[i].id,s[i].name,s[i].avg);
}
printf("\n\nStudents scoring below the average marks....\n");
printf("\n\nID\tName\tAverage\n\n");
for(i=0;i<n;i++)
{
if(s[i].avg<35.0)
printf("%d\t%s\t%f\n",s[i].id,s[i].name,s[i].avg);
}
return 0;
```

}

Output:

Enter the number of records: 3
Enter 3 student details
Enter student ID, name:1 Lyn
Enter 6 subject marks:
91 92 93 94 95 96

10.a. **Compare between structure and union with syntax and example.** [6]

	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.

10.b. **Explain fopen( ), fclose( ), fscanf( ) and fprintf() with syntax and example program considering all above functions.** [10]

fopen() function is used to open a file to perform operations such as reading, writing etc. In a C program, we declare a file pointer and use fopen() as below. fopen() function creates a new file if the mentioned file name does not exist.

FILE *fopen (const char *filename, const char *mode);

fclose() function closes the file that is being pointed by file pointer fp. In a C program, we close a file as fclose (fp);

int fclose(FILE *fp);

fscanf () function reads formatted data from a file.

fprintf () function writes formatted data to a file.

Program:

```
# include <stdio.h>
# include <string.h>
int main( )
{
FILE *fp ;
char data[50];
printf( "Opening the file test.c in write mode" ) ;
fp = fopen("test.c", "w") ;
if ( fp == NULL )
{
printf( "Could not open file test.c" ) ;
return 1;
}
printf( "\n Enter some text from keyboard" \  " to write in the file test.c" ) ;
while ( strlen ( gets( data ) ) > 0 )
{
fputs(data, fp) ;
fputs("\n", fp) ;
```

10.c.	} printf("Closing the file test.c") ; fclose(fp) ; return 0; } **Output:** Opening the file test.c in write mode Enter some text from keyboard to write in the file test.c Hai, How are you? Closing the file test.c **What are enumeration variable? How are they declared?** Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain . **Program:** #include<stdio.h> enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun}; int main() { enum week day; day = Wed; printf("%d",day); return 0; } **Output:** 2	[4]