Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.

2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.

3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills).

4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.

5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.

6) In case of some questions credit may be given by judgment on part of examiner of relevant answer based on candidate's understanding.

7) For programming language papers, credit may be given to any other program based on equivalent concept.

1. Attempt any ten of the following: Marks 20

a) Describe generic structure of 'C' program.

(Structure – 1 Mark, Description – 1 Mark)

Ans: Documentation Section

Link Section

Definition Section

Global Declaration Section

main()

{

    Declaration Section

    Executable part

}

Subprogram section

Documentation Section: It consists of a set of comment lines giving the name of the program and other details.

Link Section: The Definition Section defines all symbolic constants.

Global Declaration Section: There are some variables and those variables are declared in this section that is outside of all functions.

main() function: Every C program must have one main function section. This section contains two parts, declaration and executable part.

Subprogram Section: It contains all the user defined functions that are called in the main function.

b) State the logical and relational operator available in C-language.

(Logical Operator – 1 Mark, Relational Operator -1 Mark)

Ans: RELATIONAL OPERATOR

It is comparison operator. These operators check the relation between two variables

OPERATOR	USE	DESCRIPTION
>	Op1 > op 2	Op 1 is greater than op 2
>=	Op 1 >= op2	Op 1 is greater than or equal to op2
<	Op 1 < op 2	Op 1 is less than op 2
<=	Op 1 <= op2	Op 1 is less than or equal to op 2
= =	Op 1 = = op 2	Op 1 and op 2 are equal
! =	Op 1! = op 2	Op 1 and op are not equal

LOGICAL OPERATOR

- It is used to evaluate the conditions and expressions. The logical operators are AND, OR, NOT.
- The AND operators check for all condition to be true then only it returns true else false.
- The OR condition check for anyone be true then it evaluate true otherwise if all condition are false then it returns false.
- The NOT is called as negation operation which checks for negative condition.

OPERATOR	DESCRIPTION	EXAMPLE

& &	AND	X = 6
		Y = 3
		X < 10 & & y > 1 return true

||	OR	X = 6
		Y = 3
		X = = 5 || y == 5 return false

!	NOT	X = 6
		Y = 3
		!(x == y) return true

c) State the use of continue statement.

(Any two Use- 2 Marks)

Ans: Use of continue statement:

- When continue is executed, the statement following the continue are skipped and cause the loop to be terminated with next iteration.
- It can be used only inside for loop, while loop and do-while loop.
- It can be used for labeled loops.

d) Write the syntax of switch case statement.

(Syntax of switch case statement – 2 Marks)

Ans: Syntax:

```
switch (expression)
{
    case constant-expression : statement(s);
                                              break;
    case constant-expression : statement(s);
                                               break;
   default : statement(s);
                  break;
}
```

The following rules apply to a **switch** statement:

- The **expression** used in a **switch** statement must have an integral or enumerated type. You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
- When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a **break**. If no **break** appears, the flow of control will fall through to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

e) Write the output of following program

```
#include<stdio.h>
void main( )
{
  int a[5] = {1,  2, 3, 4, 5};
  printf (“%”, a [4] );
}
```

(For output – 2 Marks)

Ans: Output:

a[4]=5

f) Give the syntax of strcat ( ) string function.

(Syntax – 1 Mark, Explanation -1 Mark)

Ans: Syntax:

strcat(str1,str2);

Concatenates str1 and str2 and resultant string is stored in str1.

g) State the scope of local and global variable.

(Scope of local – 1 Mark, Scope of global – 1 Mark)

Ans: LOCAL VARIABLE

Scope: Local to the block in which variable defined.

Life: Till the control remains within the block in which the variable is defined.

Example:

```
main()
      {
                   int i=4;
                   printf("%d",i);
       }
```

Output: 4

Here i value is 4 only within main block

GLOBAL VARIABLE

Scope: Globally accessed till the termination of the program all sub function can access this variable.

LIFE: As long as program does not come to an end.

Example:

```
void sub();
 int i=4;
       main()
        {
                       sub();
                   printf("%d",i);
         }
void sub()
{
         printf("%d",i);
}
```

Output: 4 4

Here i value is 4 both sub() and main block

h) Distinguish between call by value and call by reference [2 points]

(Any two difference – 2 Marks)

[**Note - Any relevant point to be given Marks]

Ans:

CALL BY VALUE	**CALL BY REFERENCE**
Passing the value of variable to the calling function	Passing the address of variable to the calling function.
Changes will not be reflected back in main function.	Changes will be reflected back in main function.

Example:
void main()
{
int x=10,y=20;
printf("%d%d”,x,y);
swap(x,y);
}
void swap(int a,int b)
{
int c;
c=a;
a=b;
b=c;
/*changes here do not affect in values of x and y in main function..*/
}
	Example:
	void main()
	{
	int x=10,y=20;
	printf("%d%d”,x,y);
	swap(&x,&y);
	}
	void swap(int *a,int *b)
	{
	int c;
	c=*a;
	*a=*b;
	*b=c ;
	/*changes here do affect in values of x and y in main function..*/
	}

i) Explain the effect of following statement

int a, *b = & a;

(Effect – 2 Marks)

Ans: int a,*b=&a;

Effects:

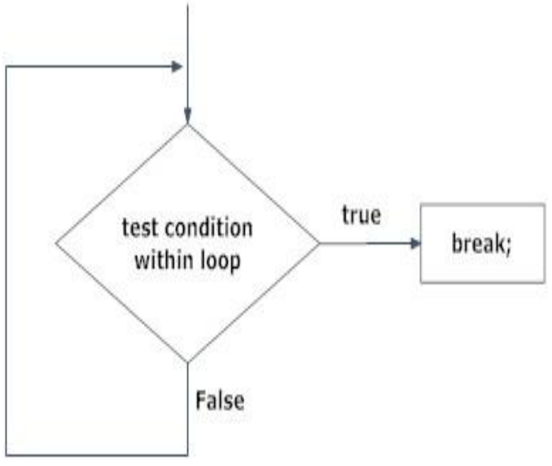
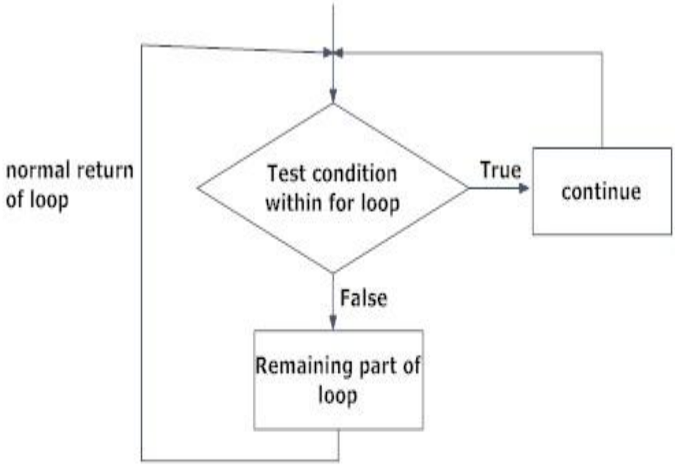
1. Variable "b" will hold the address of "a".

2. Manipulation of "a" will be easier * b will give the value of a.

j) Give the difference between break and continue statement [2 points]

(Any two difference – 2 Marks)

Ans:

BREAK STATEMENT	**CONTINUE STATEMNET**
When break is executed, the statement following break are skipped and cause the loop to be terminated.	When continue is executed, the statement following the continue are skipped and cause the loop to be continued with next iteration.
Syntax of break statement break;	Syntax of continue Statement continue;
 Figure: Flowchart of break statement	 normal return of loop Test condition within for loop True continue False Remaining part of loop Fig: Flowchart of continue statement
It can be used with switch statement to transfer control outside switch	It cannot be used with switch statements it can be used only with loops

Example
for(i=1;i<5;i++)
{
    if(i%2==0) break;
    printf(“%d”,i);
}
output:1
	Example
	for(i=1;i<5;i++)
	{
	    if(i%2==0) continue;
	    printf(“%d”,i);
	}
	output:135

k) State two features of C- language.

(Any two points – 2 Marks)

Ans: 1. High level language .hence user friendly language.

2. C supports efficient use of pointers.

3. Provides bit manipulation.

4. It is procedure oriented language.

5. Highly powerful language with various data types functions.

6. C is compiled language.

l) Give the syntax for loop.

(Syntax – 1 Mark, Explanation -1 Mark)

Ans: Basic syntax of for loop is as follows:

Example

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

In the above syntax:

Initialization: Initializes variables**.**

Condition: Conditional expression, as long as this condition is true, loop will keep executing.

Increment/Decrement: Is the modifier which may be simple increment /decrement of a variable.

2. Attempt any four of the following: Marks 16

a) State the constants and variable with example.

(Constants – 2 Marks, Variable -2 Marks)

Ans: Constants: The difference between variables and constants is that variable can change their value at any time but constants can never change their value.

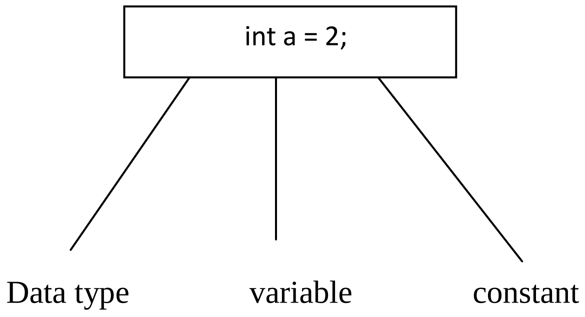
Example: m=56;

here, 56 is constant and m is variable.

TYPE	EXAMPLE
Integer constants	+226,- 946
Real constants	+ 123.23, - 22.34
Character constants	‘G’ , ‘5’, ‘+’

Variable: Variable in C are memory locations that are given names.

Variables are the entities which can change at different times we use variable to store data in memory. As shown in figure, a is variable of integer type.



Declaring variable as constants

Constant variables are variable whose value cannot be changed throughout the program.

Syntax:

const<data type><variable name> = value;

Example:

const float pi = 3.14 < ------- pi value does not change throughout program.

These types of variables can be declared using 'const' keyword.

b) Enlist different format specifier with its use.

(Each format specifier – 1/2 Marks)

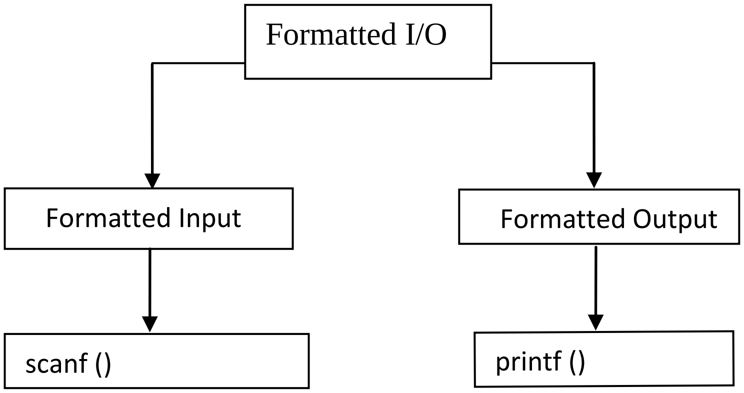
Ans:

FORMAT SPECIFIER	USE
% c	Read /write a single character
% d	Read / write a decimal integer
% e	Read /write a floating point value
% f	Read /write a floating point value
% g	Read /write a floating point value
% h	Read /write a short integer
% i	Read /write a decimal, hexadecimal or octal integer
% o	Read /write an octal integer
% s	Read /write a string
% u	Read /write an unsigned decimal integer
% x	Read /write a l, hexadecimal integer

c) Explain formatted input-output.

(Formatted input- 2 Marks, Formatted output -2 Marks)

Ans: C has standard library functions to accept and display the data



FORMATTED INPUT

Scanf: It is a function used for accepting a value or a data from the keyboard.

Syntax:

Scanf("format string", & variable1, & variable2 …..);

Example:

int a;

scanf ("%", &a); &a → address of variable.

Format String	Meaning	Example	Result
% d	Reads a decimal Interger	Scanf ("%d",&num);	%d accepts the digit number from user and it will get stored in num variable.
%c	Reads a single character	Sanf("%c, &ch);	%c accepts the character form user and it will get stored in 'ch' variable.
%f	Reads a float value	Sacnf ("%f" &num);	% f accepts the float number from the user and it will get stored in num variable.
%s	Reads a string	Scanf ("%s",str); (& is not required for string)	%s accepts the string from user and it will get stored in 'str' variable.

FORMATTED OUTPUT

printf: It is a function used for displaying a value or a data on the screen.

Syntax:

printf("format string", variable1,variable2,……);

Example:

int a =5;

printf ("%d",a); → displays 5 on screen

Format String	Meaning	Example	Result

%d	Prints a decimal Integer	int num = 5;
		Printf ("%d", num);
			5

%c	Prints a single character	Char ch =' r';
		Printf ("%c", ch);
			r

%f	Prints a float value	Float num = 14.44; Printf ("%f",num)	14.44

%s	Prints a string	Char str[] = "goodmorning";
		Printf ("%s", str);
			goodmorning

d) Explain post increment-decrement operator.

(Post increment – 2 Marks, Post decrement – 2 Marks)

Ans: **POST INCREMENT:** In this value of the variable incremented after it is being used

Example:

Consider that x is an int variable having value 8
y = x ++
will first assign the value of x to y (before increment) then increment x by 1 and store the result back into x, the value of x after the statement will be 9 and y will be 8. It is the combined effect of following two statements:

x = 8;

y = x;

x=x+1;

POST DECREMENT: In this value of the variable decremented after it is being used

Example:

Again consider that x is an int variable having value 8

y = x --;

will assign the value of x first (before decrement) to y and then decrement x by 1 and store the result back into x, the value of x after the statement will be 7 and y will be 8. it is the combined effect of following two statements:

y=x;

x=x-1;

e) Write a C-program to accept an integer number and display whether it is odd or even.

(Correct logic -2 Marks, Syntax-2 Marks)

Ans:

```
#include<stdio.h>
#include<conio.h>
main()
{
        int a;
        printf("enter a number");
        scanf("%d",&a);
        if(a%2==0)
        printf("NUMBER IS EVEN");
         else
        printf("NUMBER IS ODD");
        getch();
    }
```

f) State do while loop with example.

(Syntax – 1 Mark, explanation – 1 Mark and example with explanation -2 Marks)

Ans: DO..WHILE loops are useful for things that want to loop at least once.

Syntax:

```
do

{

/* body of do while */

} (condition);
```

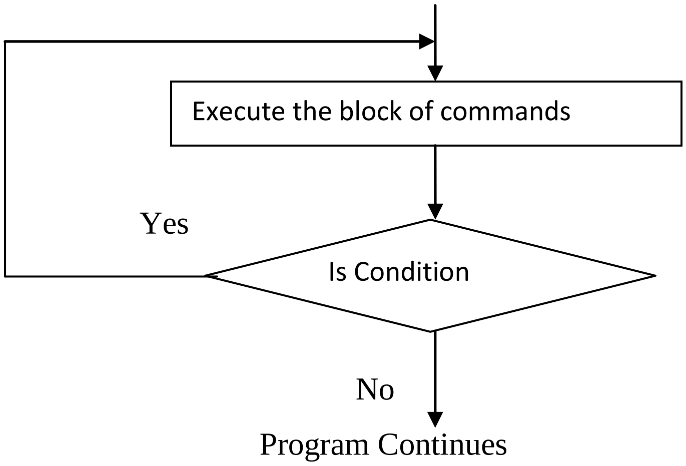
- The condition is tested at the end of the block instead of the beginning, so the block will be executed at least once.
- If the condition is true, we jump back to the beginning of the block and execute it again.
- A do..while loop is almost the same as a while loop except that the loop body is guaranteed to execute at least once.
- A while loop says "Loop while the condition is true, and execute this block of code", a do..while loop says "Execute this block of code, and then continue to loop while the condition is true

Example:

```
main( )
{
        int i = 0;
        do
        {
                printf(“HELLO”);
        }While (i!=0);
}
```

OUTPUT: HELLO

Even though the condition is false the Hello is printed once



3. Attempt any four of the following: Marks 16

a) Write a program to accept an integer number and print whether it is palindrome or not.

(Logic - 2 Marks, Syntax – 2 Marks)

Ans:

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

void main()
{
   int num, temp, remainder, reverse = 0;

   printf("Enter an integer \n");
   scanf("%d", &num);

   /*  original number is stored at temp */
   temp = num;
   while (num > 0)
   {
      remainder = num % 10;
      reverse = reverse * 10 + remainder;
      num /= 10;
   }
```

```
    printf("Given number is = %d\n", temp);
    printf("Its reverse is  = %d\n", reverse);

    if (temp == reverse)
        printf("Number is a palindrome \n");
    else
        printf("Number is not a palindrome \n");
}
```

```
Enter an integer
6789
Given number is = 6789
Its reverse is  = 9876
Number is not a palindrome
```

b) **Find out the errors from following program, justify the same and write correct program:**

```
void main( )
{
        int j = 1
        for (i=0; i <20; i ++)
        printf("%d", i, j);
        printf("%d", i);
}
```

(Finding error - 2 Marks, Correct program – 2 Marks)

Ans: **Errors:**

1) Semicolon is missing after declaration of int j.

2) %d is missing in printf statement.

Correct Program:

```
void main()
{
int j=1;
for(i=0;i<20;i++)
printf("%d %d",i,j);
printf("%d",i);
}
```

c) Explain nested if-else with example.

(Syntax - 2 Marks, Example - 2 Marks)

Ans: Nested if…else statements

Syntax:

```
if (test condition1)
{
  if(test condition2)
  {
    Statement-1;
   }
 else
  {
    statement-2;
   }
 }
 else
{
  Statement-3;
 }
Statement-x;
```

If the condition-1 is false statement-3 will be executed otherwise it continue to perform second test. If condition-2 is true, the statement-1 will be executed. Otherwise statement-2 will gets executed and the control transferred to the statement-x.

For Example:

```
 void main()
{
```

```
int a,b,c;
clrscr();
printf("Enter the number");
 scanf("%d%d%d",&a,&b,&c);
 if(a>b)
  {
    if(a>c)
      printf("%d\n",a);
   else
     printf("%d\n",c);
  }
else
 {
   if(c>b)
      printf("%d\n",c);
  else
     printf("%d\n",b);
  }
 }
```

d) Write a program to print the following pattern

```
*
* *
* * *
* * * *
* * * * *
```

(Logic - 2 Marks, Syntax – 2 Marks)

Ans:

```
#include <stdio.h>
#include<conio.h>
 void main()
 {
  int n, c, k;
```

```
              for ( c = 1 ; c <=5 ; c++ )
         {
           for( k = 1 ; k <= c ; k++ )
                  {
                   printf("*");
                   }
                   printf("\n");
         }
         }
```

Output

```
*
* *
* * *
* * * *
* * * * *
```

e) Write a program which will count number of digit in entered number.

(Logic - 2 Marks, Syntax – 2 Marks)

Ans:

```
 #include <stdio.h>
#include<conio.h>
void  main()
{
  int n,count=0;
  printf("Enter an integer: ");
  scanf("%d", &n);
  while(n!=0)
  {
       n=n/10 ;
      count++;
  }
  printf("Number of digits: %d",count);
}
```

Output

```
Enter an integer: 34523
Number of digits: 5
```

f) Write a program to find out sum of numbers from 1 to 100.

(Logic - 2 Marks, Syntaxes – 2 Marks)

Ans:

```
#include <stdio.h>
#include<conio.h>
void  main()
{
  int count = 0;
  int total, num;
              while (count <= 99)
              {
                  count++;
                  num= count;
                  total = num + count;
               }
                  Printf("sum=%d",total));
 }
```

OR

```
#include <stdio.h>
#include<conio.h>
void  main()
{
int sum = 0;
            for (int i= 1; i < =100; i++)
            {
            sum=sum+i;
            }
            printf("sum=%d",sum);
    }
```

4. Attempt any four of the following: Marks 16

a) Write a program to copy one string into another and count the number of character copied.

(Logic - 2 Marks, Syntax– 2 Marks)

Ans:

```
#include <stdio.h>
#include <string.h>
//#include<conio.h>
void main()
{
  char src[25],dest[25];
  int count_char=0,int i;
  clrscr();
  printf("\nEnter the String which is to be copied to another String:");
  gets(src);
  strcpy(dest,src);
  printf("\nCopied String is: %s",dest);

for(i=0; dest[i]!='\0'; i++)
 {
  count_char++;
  }
 printf("\nNumber of characters in string : %d",count_char);

getch();

}
```

```
Output
Enter the String which is to be copied to another String: abc
Copied String is: abc
Number of characters in string : 3
```

b) Explain the use of the following function with syntax:

i) strcmp ( )

ii) strlen ( )

(strcmp syntax – 1 Mark ,use – 1 Mark and strlen Syntax -1 Mark ,use – 1 Mark)

Ans: i) strcmp():-

This function compares two strings identified by arguments and returns zero if both strings are equal, otherwise it returns the difference between ASCII values of first non matching character pair from the strings.

Syntax:

strcmp (string1, string2);

strcmp ("there", "their");

a (difference between 'r' & 'e')

ii) strlen():-

It can be used to get the length of the string. It uses string argument which is name of character array. It returns an integer showing no. of characters from the string excluding last'\0' (null character).

Syntax:

strlen(string1);

Example:

int n; n=strlen("abc");

then n=3

c) Explain how to initialize two dimensional array with example.

(Explanation of initialization-2 Marks, Example -2 Marks)

Ans: Following is an array with 3 rows and each row has 4 columns. The one pair of brackets represent value for a rows. Now in below example there are three rows initialized.

```
int a[3][4] = {
 {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
 {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
 {8, 9, 10, 11}   /* initializers for row indexed by 2 */
};
```

Example:

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

```
void main ()
{
  /* an array with 5 rows and 2 columns*/
  int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
  int i, j;
   /* output each array element's value */
  for ( i = 0; i < 5; i++ )
  {
    for ( j = 0; j < 2; j++ )
    {
      printf("a[%d][%d] = %d\n", i,j, a[i][j] );
    }
  }
}
```

When the above code is compiled and executed, it produces the following result:

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

d) State various categories of function with one example of each.

(Functions - 2 Marks, Examples - 2 Marks)

Ans: 1. Functions with no arguments and no return values.

2. Function with arguments and no return values.

3. Function with no arguments and return values.

4. Function with arguments and return values.

1. Example for Function with no arguments and no return values.

```
#include<stdio.h>
 void main()
{
 void evenodd(void)
clrscr();
evenodd();
getch();
}
void evenodd()
{
int num=25;
 if(num%2==0)
printf(“%d is even”,num);
 else
printf(“%d is odd”,num);
}
```

2. Example for Function with arguments and no return values.

```
#include<stdio.h>
#include<conio.h>
void sqr(int);
void main()
{
int pos=0;
clrscr();
printf("\nEnter last position");
scanf("%d",&pos);
sqr(pos);
getch();
}
```

```
 void sqr(int x)
{
 int i=1;
for(i=1;i<=x;i++)
{
 printf("%d\n",i*i);
 }
 }
```

3. Example for Function with no arguments and return values.

```
#include<stdio.h>
 #include<conio.h>
void main()
 {
 int fact(void);
clrscr();
printf("\nfactorial of number:%d",fact());
getch();
 }
 int fact(void)
 {
int no=1;
int fact=1,i=0;
printf("\n Enter number:");
 scanf("%d",&no);
 for(i=1;i<=no;i++)
 {
fact=fact*i;
}
 return fact;
}
```

4. Example for Function with arguments and return values.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int fact(int);
int no;
clrscr();
printf("\n Enter number:");
scanf("%d",&no);
 printf("\nfactorial of number:%d",fact(no));
getch();
}
int fact(int n)
{
int fact=1,i=0;
for(i=1;i<=n;i++)
{
fact=fact*i;
}
 return fact;
 }
```

e) Explain any two storage classes.

(For each – 2 Marks)

Ans: storage classes are as follows:

Auto (local): This is default storage class (i.e. if storage class of variable is not specified then storage class will be auto) .Auto variables are available within block in which they are declared. auto is the default storage class for all local variables.

```
{
int Count;
auto int Month;
}
```

The example above defines two variables with the same storage class. Auto can only be used within functions, i.e. local variables.

Auto variable inside function will available within that function only &will have life within that function only.

Register: Is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location).

register is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and cant have the unary '&' operator applied to it (as it does not have a memory location).

```
{
register int  Miles;
}
```

Register should only be used for variables that require quick access - such as counters. It should also be noted that defining 'register' goes not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register - depending on hardware and implementation restrictions.

Register should only be used for variables that require quick access

Static: - static can also be defined within a function. If this is done the variable is initialized at run time but is not reinitialized when the function is called.

static is the default storage class for global variables. The two variables below (count and road) both have a static storage class.

```
static int Count;
    int Road;

    {
      printf("%d\n", Road);
    }
```

static variables can be 'seen' within all functions in this source file. At link time, the static variables defined here will not be seen by the object modules that are brought in.

static can also be defined within a function. If this is done the variable is initialised at run time but is not reinitialized when the function is called.

External: - extern is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern' the variable cannot be initialized as all it does is point the variable name at a storage location that has been previously defined.

extern is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern' the variable cannot be initialized as all it does is point the variable name at a storage location that has been previously defined.

When you have multiple files and you define a global variable or function which will be used in other files also, then extern will be used in another file to give reference of defined variable or function. Just for understanding extern is used to declare a global variable or function in another files.

File 1:

```
main.c
 int count=5;
 main()
  {
 write_extern();
 }
```

File 2:

```
 write.c
void write_extern(void);
extern int count;
void write_extern(void)
{
  printf("count is %i\n", count);
 }
```

f) Define:

i) Function definition

ii) Function body

iii) Function call

iv) Function prototype.

(For each – 1 Mark)

Ans: i) Function definition:

A function definition contains a function declaration and the body of a function.

```
return-type function-name(parameters)
{
declarations
statements
return value;
}
```

- Return-type: type of value returned by function or void if none.
- Function-name: unique name identifying function.
- Parameters: comma-separated list of types and names of parameters.
- Value: value returned upon termination.

(not needed if return-type void)

The list of parameters is a declaration in the form type 1 par 1, ..., type n par n and represents external values needed by the function. The list of parameters can be empty.

ii) Function body:

The function body is the bit between the opening and closing braces that follow the line where the function name appears. The function body contains all the statements that define what the function does. The example's function main() has a very simple function body consisting of just one statement:

For example:

```
Void main()
{
  Printf("hello");
}
```

Every function must have a body, although the body can be empty and just consist of the two braces without any statements between them. In this case, the function will do nothing.

iii) Function call:

- Function call statement is used in main() or any other function wherever we requires to execute code of function.
- Function statement involves specifying values in function call statement in bracket these values specified in function call are called as actual parameters.
- Function call must end with semicolon
- Control transfer in function call:- wherever function call is program then current execution is kept on hold & code of function definition is executed .after executing code of function program come back to next instruction of function call statement & it is executed .

```
void  hello(void);
        void main()
        {
         Printf("you are in main");
          Hello();
           Printf("you are in main again");
          }
  Function call
        Void hello(void)
      {
        Printf("you are in function hello");
       }
```

Output:-

```
You are in main
You are in function hello
You are in main again
```

In given program after executing hello(); the execution control is transferred to definition part of function and all code of function definition is executed &after completion of execution of function program execution control come back to main & next line of program i.e. printf("you are in main again"); will be executed.

iv) Function Prototype:

This statement specifies to compiler details of user defined functions

The details specified by declaration statement are:-

a) Return type:-data type of value returned by current function to caller .if function is not returning any value to caller then return type void is specified

b) Name of function (identifier):-it is name given to function. it must follow the identifier naming rules. These parameters are called as formal parameters. (refer rules of variable naming in chapter -1)

c) Name of formal parameter can be different than parameter provided in function definition

d) Name of parameter can be omitted but its data type must be given in parameter list of formal parameter

e) Formal Parameters (arguments):- these are the list of input taken by function along with their data type. If function is taking more than one argument then arguments are separated by comma.

f) **Syntax :-**

Return type name_of_function(data type para-1, data type para-2……., data type para-N);

Example:-Void area(int radius);

g) Every function declaration statement must be ended with semicolon

h) Function declaration dose not allocate space in memory for variables

5. Attempt any four of the following: Marks 16

a) Define array with its need and how elements are allocated with space in memory for one dimensional array.

(Definition of array – 1 Mark, Need – 1 Mark, Element allocation – 2 Marks)

Ans: An array is a fixed-size sequenced collection of elements of the same data type.

Need of an array:

A variable can store only one value at a given time. Therefore they can be used only to handle limited amount of data. To process such large amount of data, we need a powerful data type that facilitates efficient storing, accessing and manipulation of data items. Array can be used for such kinds of applications.

For one dimensional array elements are allocated by following way:

int number[5]={35,40,56,32,12};

The computer reserves five storage locations as below:

	number[0]
	number[1]
	number[2]
	number[3]
	number[4]

The values to the array elements can be assigned as follows:

number[0]=35;
number[1]=40;
number[2]=56;
number[3]=32;
number[4]=12;

This would cause the array number to store the values as shown below:

number[0]	35
number[1]	40
number[2]	56
number[3]	32
number[4]	12

b) Write a program to sort elements of an array in ascending order. Read elements of array from user [using scanf function].

(Logic - 2 Marks, Syntax – 2 Marks)

Ans:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int arr[10],i,j,temp;
        clrscr();
         printf("Enter array elements:");
         for(i=0;i<10;i++)
        {
                scanf("%d",&arr[i]);
        }
                printf("\n\nArray elements are:");
         for(i=0;i<10;i++)
        {
                printf("%d ",arr[i]);
         }
         for(j=0;j<10;j++)
         {
         for(i=0;i<10;i++)
          {
           if(arr[i+1]<arr[i])
           {
                temp=arr[i];
                arr[i]=arr[i+1];
                arr[i+1]=temp;
           }
           }
           }
```

```
                printf("\n\nArray elements in ascending order are:");
            for(i=0;i<10;i++)
            {
                    printf("%d  ",arr[i]);
            }
           getch();
    }
```

c) Write a program to declare structure student having member variables are roll-no, name and marks. Accept data for one student and display it.

(Logic - 2 Marks, Syntax – 2 Marks)

Ans:

```
#include<stdio.h>
#include<conio.h>
struct student
{
 int roll_no;
 char name[20];
 float marks;
}s;
void main()
{
 clrscr();
 printf("Enter student's roll number:");
 scanf("%d",&s.roll_no);
 printf("\nEnter student's name:");
 scanf("%s",s.name);
 printf("\nEnter student's marks:");
 scanf("%f",&s.marks);
 printf("\n\nStudent's details are:");
 printf("\nRoll number=%d",s.roll_no);
 printf("\nName=%s",s.name);
 printf("\nMarks=%f",s.marks);
 getch();
}
```

d) Explain recursion with suitable example.

(Explanation – 2 Marks, Example – 2 Marks)

(Note: Any relevant example can be given marks.)

Ans: When a function is called by itself within its own body then that process is called as Recursion. I process chaining occurs.

Example:

```
#include<stdio.h>
#include<conio.h>
int fact(int);
void main()
{
 int a,b;
 clrscr();
 printf("Enter a number:");
 scanf("%d",&a);
 b=fact(a);
 printf("\nFactorial of a given number is:%d",b);
 getch();
}
int fact(int n)
{
 int f;
 if (n==1)
 return(1);
 else
 f=n*fact(n-1);  /*recursion occurs here*/
 return(f);
}
```

In above example recursion is used where function named fact is called by itself to get the factorial of an entered number.

e) Explain structure with example.

(Explanation- 2 Marks, Example -2 Marks)

(Note: Any relevant example can be given marks.)

Ans: Structure is user defined data type which is used to store elements of different data types. Structure is a convenient tool for handling a group of logically related data items.

Example:

```
#include<stdio.h>
#include<conio.h>
struct student
{
 int roll_no;
 char name[20];
 float marks;
}s;
void main()
{
 clrscr();
 printf("Enter student's roll number:");
 scanf("%d",&s.roll_no);
 printf("\nEnter student's name:");
 scanf("%s",s.name);
 printf("\nEnter student's marks:");
 scanf("%f",&s.marks);
 printf("\n\nStudent's details are:");
 printf("\nRoll number=%d",s.roll_no);
 printf("\nName=%s",s.name);
 printf("\nMarks=%f",s.marks);
 getch();
}
```

f) Write a function to swap the value of variables say a and b. Use function name "swap".

(Logic - 2 Marks, Syntax – 2 Marks)

Ans:

```
#include<stdio.h>
#include<conio.h>
void swap(int,int);
void main()
{
 int a,b;
 clrscr();
 printf("\nEnter the values of a b:");
 scanf("%d %d",&a,&b);
 printf("\nBefore swapping:");
 printf("\na=%d b=%d",a,b);
 swap(a,b);
 getch();
}
void swap(int x,int y)
{
 int temp;
 temp=x;
 x=y;
 y=temp;
 printf("\nAfter swapping:");
 printf("\na=%d b=%d",x,y);
}
```

6. Attempt any four of the following: Marks 16

a) Write a function to display fibonnacci series up to given number using recursion .Use function name "Fibbo".

(Logic - 2 Marks, Syntax – 2 Marks)

Ans:

```
#include<stdio.h>
#include<conio.h>
int Fibbo(int n);
void main()
{
 int c,i=0,n;
 clrscr();
 printf("Enter fibonacci number:");
 scanf("%d",&n);
 printf("\nFibonacci number series:");
 for(c=1;c<=n;c++)
 {
  printf("%d ",Fibbo(i));
  i++;
 }
 getch();
}
int Fibbo(int n)
{
if(n==0)
return(1);
else if(n==1)
return(1);
else
return(Fibbo(n-1)+Fibbo(n-2));
}
```

b) Define pointer. State the syntax to declare pointer variable with example.

(Definition -1 Mark, Syntax- 1 Mark, Example -2 Marks)

(Note: Any relevant example can be given marks.)

Ans: Pointer is a variable which stores the memory address of another variable.

Syntax to declare pointer variable:
data_type *pt_name;

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int x=10;
 int *ptr;   //Pointer declaration
 clrscr();
 ptr=&x;
 printf("Value of x=%d",x);
 printf("\nAddress of x=%u",ptr);
 getch();
}
```

c) Give the output of the following code:

```
# include<stdio.h>
void main( )
{
        int * a [4];
        int I;
        int m = 10 , n = 20, p = 30, q =40;
        a[0] = & m;
        a[1] = & n;
        a[2] = & p;
```

```
    a[3] = & q;
    for( I = 0; i < 4; i ++)
    printf(“%d \n”, *a [1]);
}
```

(for each line of Correct Output –1 Mark)

Ans: **Output**

```
20
20
20
20
```

d) Write a program in C using pointers to determine length of a string.

(Logic - 2 Marks, Syntax – 2 Marks)

Ans:

```
 #include<stdio.h>
#include<conio.h>
void main()
{
        char *ptr,str[10];
        int cnt=0;
        clrscr();
        ptr=str;
        printf("Enter string:");
        scanf("%s",str);
        while(*ptr!='\0')
    {
        cnt=cnt+1;
        tr=ptr+1;
    }
    printf("\nLength of a string is:%d",cnt);
    getch();
}
```

e) Write a program using pointers to compute the sum of all elements stored in an array.

(Logic - 2 Marks, Syntax – 2 Marks)

Ans:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int *ptr,a[10],sum=0,i;
        clrscr();
        ptr=a;
        printf("Enter array elements:");
        for(i=0;i<10;i++)
        {
                scanf("%d",&a[i]);
        }
        for(i=0;i<10;i++)
        {
        sum=sum+*ptr;
        ptr++;
        }
        printf("\nSum of all the elelments stored in an array is:%d",sum);
        getch();
}
```

f) The following is segment of a program:

```
void main( )
{
        int a, b, *p1, *p2, x, y;
        a = 10; b=5;
        p1 = & a;
        p2= & b;
        x = *p1 * * p2 – 6;
        y = *p1 * * p2 + 10;
        printf (“a = %d, b = %d”, a,b);
        printf(“x = %d, y = %d”, x, y);
}
```

What will be the output of program?

(Correct output -4 Marks (Each Value- 1 Mark))

Ans: Output

```
a=10, b=5 x=44, y=60
```