PROGRAMMING WITH ‘C’

The ‘c’ Programming language was introduced by “Dennis Ritchie” in “At and T” Bell laboratories in early 1970’s.

‘C’-COMPILER: -

A ’c’ program has been converted in to the Machine language before execution. A ‘c’ Compiler translates a ‘c’ program in to a Machine language.

PRE-PROCESSOR: -

Before compailing a C-program a process called Pre-Processor is done on the source board by the program called Pre-Processor. Understanding this process will help us “# include <stdio.h>”. This line is a pre-processor Directory.The pre-pocessor process the C program before the compailer .All Lines in the program beginning with the hash (#) sign are processed by the

Pre-processor.The # include directive causes the Pre-processor to effectively insert the stdio.k by into the C-Program. When the Compailer compiles the program, it seems the contexts of the stdio.h file inserted into the Pre-processor value followed by the rest of the program.

Following are the pre processor directives:

```
#define
#include
#if
#else
#undef
#progma
```

# define:-

#define is used to make macro definition in a ‘c’ program.

Ex:

```
# define MAX 100
```

Here we define a constant max with its value 100. Whenever a program encounted MAX it is replaced to 100.

#include:-

This is used to include the header files containing the function to be used in the function.

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

This will include the stdio.h file from the include directory.

Example program

```
#include<stdio.h>
#include<conio.h>
main()
{
        clrscr();
        printf("Welcome To C");
        getch();
}
```

VARIABLES AND EXPRESSIONS: -

CHARACTER SET: -

The ‘C’ character set consists of upper and lower case Alphabets, Digits, Special Characters and White Spaces. The Alphabets and Digits together called as alphanumeric characters.

Alphabets	:	A……….Z
		a………..z
Digits	:	0…….9
Special characters	:	comma, period, ash.
White space characters	:	blank space, New line.

IDENTIFIERS:-

Identifiers as the name suggests are used to identify or name variables, Symbolical constants, functions…etc.

There are some rules regarding identifiers names as follows:-

(1) Identifier name must be a sequence of digits or letters and must begins with a letter.

(2) The underscore (_) symbol is considered as a letter.

(3) Names shouldn’t be the same as key words.

KEY WORDS:-

Key words have pre-defined meanings and cannot be changed by the user. Key words are pre-defined by the language and cannot be used by the user in any way. The following key words are pre-defined by the language.

Auto	int	for	exterm
Break	long int	while	switch
Case	long	goto	else
Char	if	float	do

………… etc are some of the pre-defined key words in 'c'.

VARIABLES:-

A variable is an antity that has the value and is known to the program by a name. A variable decision associates a memory location with the variable name. A variable can have only one value assigned it at any given time during the execution of the program. Variable names are identifiers used to the name variable. A variable name consists of sequence of letters and digits. The first character must be a letter.

BASIC- DATA TYPES:-

The 'c' language supports the following Data types.

Char	: A single byte that holds a character.
Int	: An Integer.
Float	: Holds a floating point number.
Double	: A double precession floating point number.

RANGE OF DATA TYPES:-

Integer	: - 32768-32767.
Un-signed int	: - 0-65535.
Long int	: - 214783647-2147483647.
Character	: - 128-128.
Un-signed char	: - 0-255.
Integer	: - 2 Bytes.

Float	: -	4 Bytes.
Float range	: -	$-3.4e^{-38}$-$3.4e^{+38}$
Char	: -	1 Byte.
Double	: -	8 Bytes.
Double range	**: -**	**$1.7e^{-308}$-$1.7e^{+308}$.**

ALGORITHM:-

An Algorithm is a finite set of roots for solving a specific type of problems.

(or)

Algorithm is nothing but the step by step process of a program.

STRUCTURE OF ALGORITHM:-

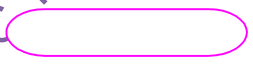
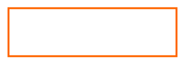
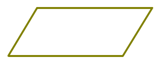
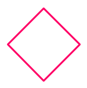
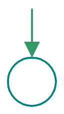
Algorithm consists of following steps:

Step 1	:-	In-put.
Step 2	:-	Assignment step.
Step 3	:-	Decission step.
Step 4	:-	Repitative step.
Step 5	:-	out-put.

FLOW CHART:-

A flow chart is a pictorial diagram to solve a particular program. It is a diagrammatic representation that illustrates the sequence of operations to be performed to arrive the solutions. The operating instructions are placed in boxes, which are connected by arrows.

SYMBOLS:-The symbols are:-

Oval		Start/Stop.
Rectangle		Processing Box.
Parallelogram		Input / Out put.
Rhombus		Decisions box.
Circle		Connector.

OPERATORS AND EXPRESSIONS-

There are Seven types of Operators :-

i. Arthematic Operator.
ii. Relational Operator.
iii. Logical Operator.
iv. Assignment Operator.
v. Conditional Operator.
vi. Bit-Wise Operator.
vii. Comma Operator.

ARTHEMATIC OPERATOR:-

The Arthematic operator performs arthematic operations and can be Classified in to unary and Binary Arthematic Operations.

OPERATORS:-

+	Addition of unary +.
-	Substraction of unary - .
*	Multiplication.
/	Division.
%	Modulo Division.

Example for Arthematic operator (addition):-

```
 #include<stdio.h>
main()
{
  int a,b,c=0;
  printf("Enter Two Numbers a&b:");
  scanf("%d%d",&a,&b);
  c=a+b;
  printf("a+b Is %d",c);
  getch();
}
```

Similarly by changing the symbols of the arthematic operatirs we can see the other operator programs(-,*,/,%)

RELATIONAL OPERATOR:-

These are used to compare arthematic, logical and character expressions.

OPERATORS:-

<	Less than.
>	Greater than.
<=	Less than or Equal to.
>=	Greater than or Equal to.
==	Equal to.
!=	Not equal to.

LOGICAL OPERATOR:-

It is used to compare or equalate logical and relational expressions. There are three logical Operators in 'c'.

i. Logical OR.
ii. Logical AND.
iii. Logical NOT.

LOGICAL AND:-

Consider the following distribution.

a>b && x==0

The expression on the left is 'a' is greater than 'b'.
The expression on the right x==0.The whole expression evaluates to true only if both expressions are true.

LOGICAL OR:-

Consider the following distribution.

a>b || x==0

Whole expression is one of them is true, or if both of them are true, i.e. if the value of 'a' is less than of 'm' or 'n'.

LOGICAL NOT:-

The not operator takes single expression and evaluates to true if the expression is false, evaluates the false if the expression is true.

BIT-WISE OPERATOR:-

It operates each print of Data. These operators are used for testing, Complementing or shifting bits to the right or left.

&&	Bit wise AND.
||	Bit wise OR.
^	Bit wise XOR.
~	Bit wise Complement.
<<	Shift Left.
>>	Shift Right.

EXAMPLE:-

A=13, B=7.

Binary representation of a is 0000 0000 1101

Binary representation of a is 0000 0000 0111

BIT-WISE AND:-

Bit wise AND Operator a&&b.

```
      a   =  0000 0000 1101
      b   =  0000 0000 0111
c=a&&b   =  0000 0000 0101
```

BIT-WISE OR:-

The statement a||b.

```
      a   =  0000 0000 1101
      b   =  0000 0000 0111
c=a||b   =  0000 0000 1111
```

BIT-WISE XOR:-

The statement a^b after this statement is exected, a bit in c will be when ever the corresponding bits in a^b differ.

```
      a   =  0000 0000 1101
      b   =  0000 0000 0111
c=a^b    =  0000 0000 1010
```

SHIFT LEFT OPERATOR:-

The left shift operator is a binary operator.
For example consider the statement c=a<<3. The value in integer ‘a’ is shifted to left by 3 bit position. The result is assigned to integer ‘c’.The value of ‘a’ is 0000 0000 1101. The value of ‘c’ after the execution of above statement is 0000 0110 1000 .

SHIFT RIGHT OPERATOR:-

The Right shift operator is also a binary operator.
For example consider the statement c=a>>2. The value in integer ‘a’ is shifted to right by 2 bit position. The result is assigned to integer ‘c’. The value of ‘a’ is 0000 0000 1101. The value of ‘c’ after the execution of above statement is 0000 0000 0011.

BIT-WISE COMPLEMENT :-

It is a unary operator. It gives the value got by complementing each bit of operator.

ARTHEMATIC ASSIGNMENT OPERATOR:-

This Operator evaluates the expression on right, and assigns the value to variable on the left, the assignment operators are:
=, -=, +=, /=, *= .

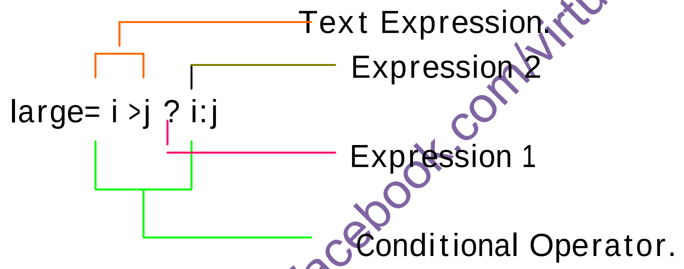
INCREMENT - DECREMENT OPERATOR:-

These are extensively used in “for & while “loops.
The syntax of Operators is givae below.
++(Variable name) INCREMENT --(Variable name)
(Variable name)++ DECREMENT (Variable name)
The operator ++ ads One and -- Substracts One.

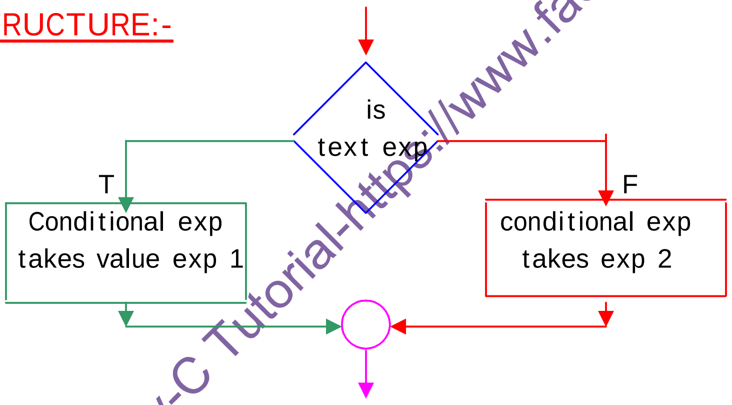
CONDITIONAL OPERATOR:-

The conditional operator consists of two symbols (? And :) Question and coloumn.

SYNTAX:-



STRUCTURE:-



program:-

```
   #include<stdio.h>
  main()
    {
        int i,j,large;
        printf(“enter value for i,j”) ;
        scanf(“%d%d”,&i,&j) ;
        large=i>j?i:j ;
        printf(“large value is %d”,large) ;
    }
```

COMMA OPERATOR:-

A set of expressions separated by comma's is valid construct in the c-language.

Ex:-

i,j are declared by the statement.

int i,j;

program:- Area of a Triangle.

```
#include<stdio.h>
main( )
{
    float b,h,area;
    printf("Enter value for b,h");
    scanf("%d%d",&b,&h);
    area=(b*h)/2;
    printf("Area of the triangle is %d",area);
}
```

INPUT STATEMENTS:-

Input statements are used for reading integer, character, string and mixed type data. For this purpose some standard input functions are used. They are

i. get char( );
ii. scanf( );
iii. gets( );
iv. getche( );

(i). GET CHAR( ); :-

This function is used for reading a single character from the keyboard. The variable in which you want to store a single character should be of character type .

The syntax is

char v;

v= get char( );

(ii). Scanf( ); :-

The scanf input statement is used for reading mixed data types. We can read int, float, char, exha, decimal, octa......etc. by using it. Its control

codes are format codes.

syntax :

Scanf(“control string), &v_1, &v_2,…..);

(iii). Gets( ); :-

The purpose of the gets statement is to read a string. It will read a string until you press enter key from the key board. It will mark null character in the memory at the end of the string when you press the enter key.

syntax :

v= gets( );

here v is the string variable. i.e. character variable.

(iv). get che( ); :-

It will store any value from the key board and also displays it on the screen.

syntax :

getche( );

OUT PUT STATEMENTS:-

Out put statements are used for writing, displaying, printing, character, string, mixed type data. These are :

i. put char ( );
ii. puts ( );
iii. printf ( );

put char ( ); :-

put char is an out put statement. The purpose of put char statement is to display a single character on the screen.

syntax :

Put char (v);

Where v is the variable of character type in which a single character data is stored.

Puts( ); :-

The purpose of the puts statement is the print or displaying a string inputed by the gets statement.

syntax :

Puts (v);

Here v is the string variable.

Printf ( ); :-

The printf statement is used to display a text message or a value stored in the variable.

syntax :

Printf(“control string”, v_1, v_2,……);

Here v_1, v_2,… are variables.

CONTROL STRUCTURES :-

Control Structures are of two types .They are:-

i. Decission making control structures.
   - i. if
   - ii. if-else
   - iii. switch

ii. Loop Constracts.
   - i. for loop.
   - ii. While loop.
   - iii. Do-while loop.

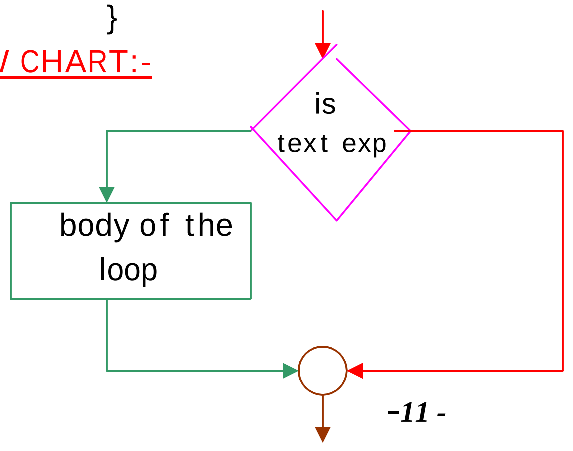
DECISSION MAKING CONTROL STRUCTURES :-

IF STATEMENT:-

SYNTAX:-

```
                if(text exp)
               Statement 1;          Single Statement
        if(text exp)
        {
            Statement 1;                 Compound Statement
            Statement 2;
        }
```

FLOW CHART:-



program:- Largest of three numbers

```
# include<stdio.h>
main( )
    {
      int a,b,c,big;
      printf(“Enter any three integer values “);
      scanf(“%d%d%d”,&a,&b,&c);
      if(big=0)
     printf(“a is big”);
     if(b=big)
          printf(“B is big”);
    if(c=big)
         printf(“c is big”);
   printf(“The largest value is %d”, big);
 }
```

IF-ELSE STATEMENT :-

SYNTAX:-

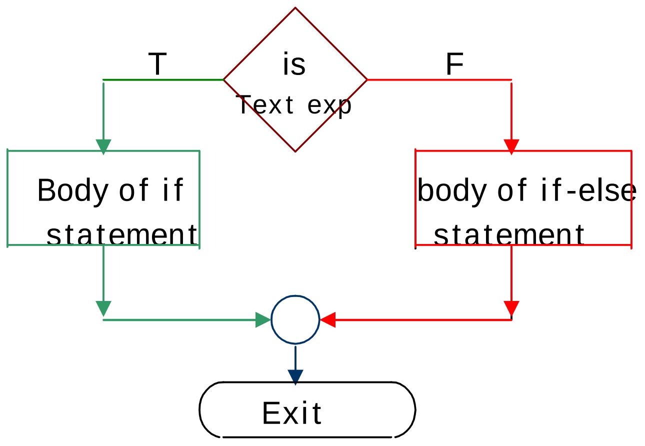
SINGLE STATEMENT :-

```
if(text expression)
          statement 1;
else
          statement 2;
```

COMPOUND STATEMENTS:-

```
if(text expression)
 {
    statement 1;
    statement 2;
    ……………….
 }
else
  {
     statement 1;
     statement 2;
     ………………….
  }
```

FLOW CHART :-



Program:-

```
 #include<stdio.h>
main( );
int I;
printf(“Enter value for I”);
scanf(“%d”,&i) ;
   if(I%3==0)
       printf(“%d is divisible by 3”, I) ;
  else
       printf(“%d is not divisible by 3 “, I) ;
}
```

SWITCH STATEMENT:-

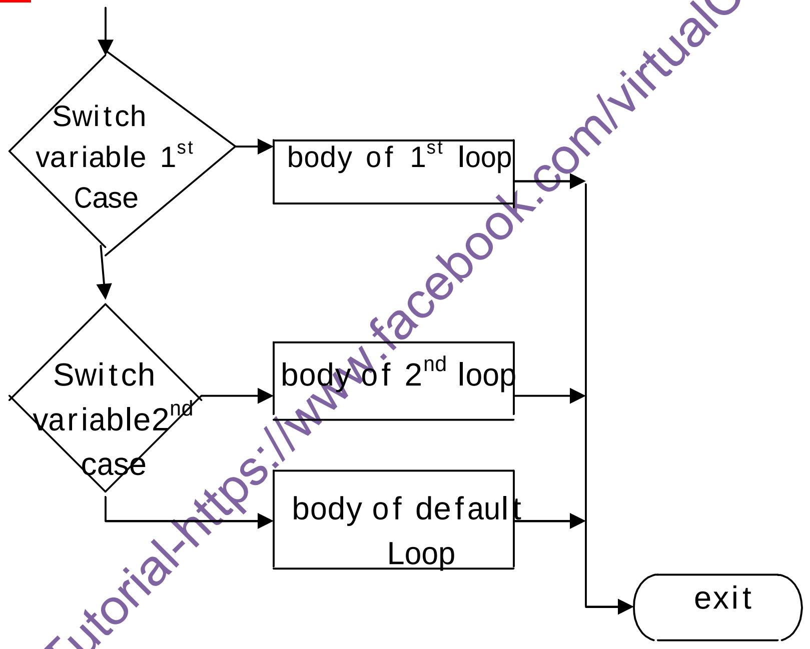
A switch statement allows the user to choose a statement or group of statements among several alternatives. The switch statement is useful when a variable is to be compared with different constants and in case it is equal to a constant a set of statements are to be executed. Also the constants in the case statement can be of char, int data types only .

SYNTAX:-

```
Switch(n)
 {
    case 1:
       Statement 1;
       Statement 2;
       Break;
   Case 2:
       Statement 1;
       Statement 2;
       Break;
```

```
            ……….;
        default;
            statement 1;
     }
```

FLOW CHART:-



Example:-

Roots of Quadratic Expressions:

```
  #include<stdio.h>
  #include<math.h>
    main()
    {
        float a,b,c,d,real,img, r1,r2;
        int k;
        printf(“Enter the three values for a,b,c”);
        scanf(“%f%f%f”,&a,&b,&c);
        d=b*b-(4*a*c);
             if(a!=0)
               {
                    if(d< 0)
                    k=1;
                else
                    if(d==0)
                        k=2;
                    if(d>0)
                        k=3;
            switch(k)
             {
             case 1:
                    printf("The roots are imaginary");
                    d=-d;
                    real=-b/(2*a);
                    img=sqrt(d/(2*a));
                    printf("The roots are");
                    printf("r1=%f+%f",real,img);
                    printf("r2=%f-%f",real,img");
                    break;
             case 2:
                    printf("The roots are real and equal");
                    r1=-b/(2*a);
                    printf("r1=r2=%f",r1);
                    break;
             case 3:
                     printf("The roots are real & un equal");
                     real=-b/(2*a);
                     img=sqrt(d/(2*a));
                     printf("The roots are");
                     printf("r1=%f+%f",real,img);
                     printf("r2=%f-%f",real,img");
                     break;
             }
        }
        else
             printf("The equation is linear");
  }
```

LOOP CONSTRUCTS :-

Loops in 'c' called as a section of a program to be executed repeatedly while an expression is true . When the expression becomes false, the loop terminates & the control passes on to the statement following the loop. A loop consists of two statements.

i. Control Statement.
ii. Body of the loop.

FOR LOOP:-

It is used while executing the loop a number of times.

SYNTAX:-

For(initialization; text exp; update exp)
Statement /compound statement;

EXAMPLE :-

The program that displays the first 10 multiples of 5 on a single line is given below.

```
#include<stdio.h>
main( )
 {
     int i;
     for(i=0;i<+5;i++)
     printf("%d",i);
   }
```

The for key word is followed by three components enclosed within round braces"( )".These three components are separated by semicoloun(;).

In the above example the three components are i=1,i<=5,i++ The first component i=1 is executed only once prior to the statements with in the for loop. This is called initialization expression.

The second component I<+5 is evaluated once before every execution of the statement with in the loop. This is called the text expression of the loop. If this expression is true the statement with in the loop executes . If it is false, the loop terminates & the control of the execution is transfered to the statement following the for loop.

The third component i++ is executed once every execution of the statement with in the loop. In this case this increment the value of 'i' by 1. It is called the update expression.

INITIALIZATION EXPRESSION:-

The control variable is initialized. This part is executed once before executing the body of the loop.

TEST EXPRESSION:-

The value of the control variable is tested and if found true ,the body of the loop is executed other wise the loop is terminated. This expression is evaluated before every execution of the loop body.

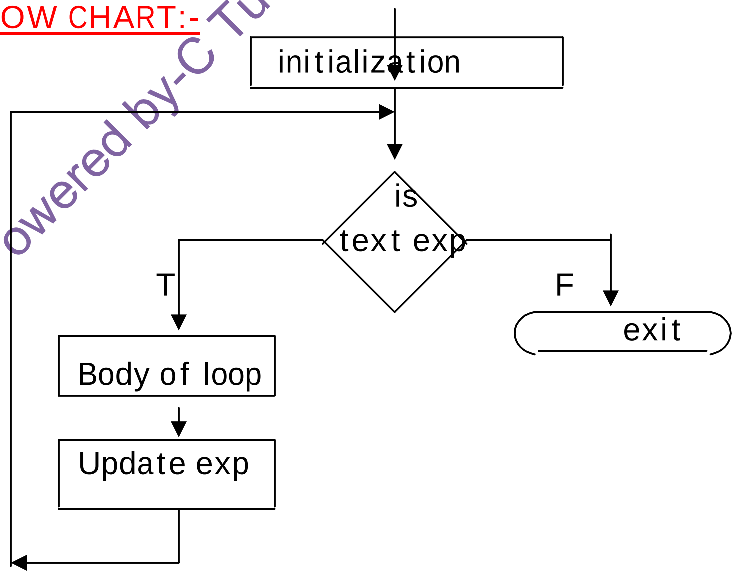
UP DATE EXPRESSION:-

After executed the last exp in the for loop the control is transferred back to the for loop statement. The control variable is updated using an exp such as I++ and the text exp is evaluated to see if it is true before executing the body of the loop. For multiple initialization and up date exp and up date loop Variables we use the comma operator.

EXAMPLE:-

```
#include<stdio.h>
main( )
  {
     int i,k;
     printf("It is 5th table ");
      for(i=0;i<=20;i++)
          {
            k=i*5;
            printf("5*%d=%d\n",i,k);
          }
    }
```

FLOW CHART:-



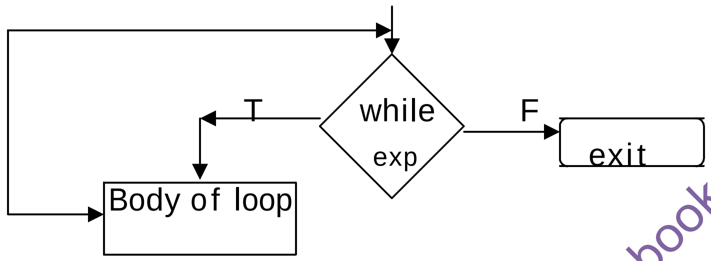
WHILE LOOP:-

The while loop is used when the no of times the loop is to be executed is not known in advance. While loop is top tested loop.

SYNTAX:-

```
            While(text exp)
            {
                    statement 1;
                    statement 2;
                    …………..;
             }
```

FLOW CHART:-



EXAMPLE :-

```
/* Average of n numbers */
          #include <stdio.h>
          main( )
            {
               int i,n=0;
               float sum=0,avg;
               printf(“Enter values –1 to terminate “);
               scanf(“%d”,&i);
            while (i!=-1)
                  {
                      sum=sum+i;
                      n=n++;
                      scanf(“%d”,&i);
                    }
               avg=sum/2;
               printf(“Average of %f is ”,avg);
                }
```

The first scanf statement just before the while loop inputs the first marks and stores it in ‘i’. The scanf statement inside the loop reads

the rest of the numbers one by one when –1 is read, the condition in the while statement.
i!=-1 evaluates to false. So, the while loop terminates and the program execution proceedes with the statements immediately after the while loop. Which in the above program.

DO-WHILE LOOP:-

The Do-while loop evaluates the condition after the execution of statement in its construct. The statement with in the Do- loop are executed at least o0nce. So, the Do-while loop is called a bottom tested loop.

If a single statement has to be executed repeatedly, then the syntax is

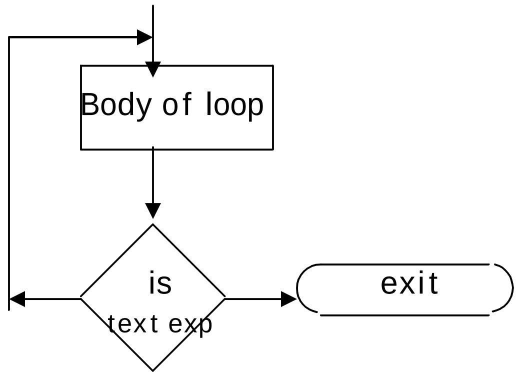
SINGLE EXPRESSION:

```
do
      statement 1;
       Whole (text exp);
```

MULTI STATEMENT :

```
do
      {
         statement 1;
           stement 2;
       }
 while (text exp);
```

FLOW CHART:-



example program:-

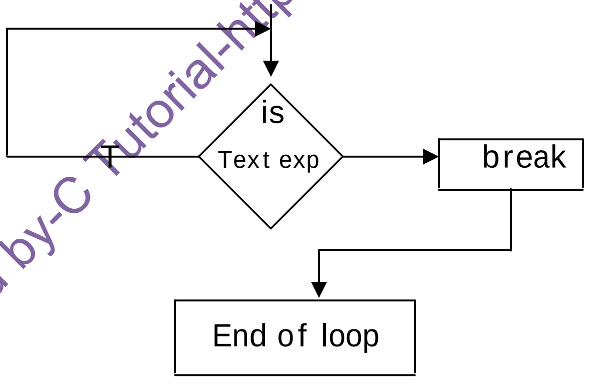
```
/* Number Palindrome */
#include<stdio.h>
main( )
```

```
{
   int num,digit,rev,n;
   printf (”Enter value for n”);
   scanf (“%d”,&n);
   num=n;
     do
         {
             digit=n/10;
             rev=rev*10*digit;
             n=n%10;
          }
           while n!=0);
               if(num==rev)
            printf(“%d is Palindrome”, num);
               else
             printf(“%d is not palindrome “,num);
    }
```

BREAK:-

A break statement terminates the loop & control is transferred to the statement immediately following the loop.

FLOW CHART:-

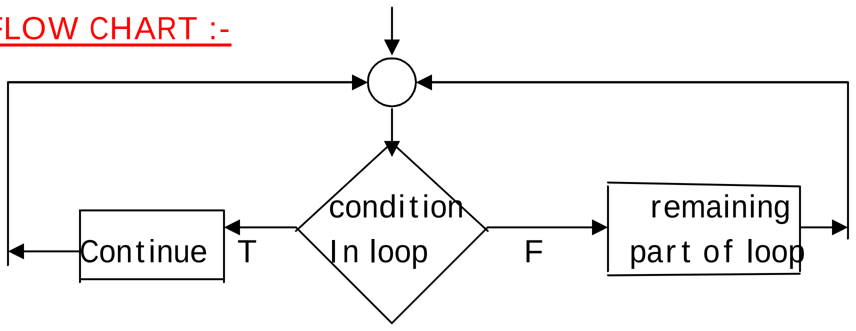


Example for break is similar to the switch which are used combindly in many of our requirements.

CONTINUE :-

It is used to by pass the remainder of the current pass through a loop. The loop doesn’t terminate when a continue statement is encounted. The remaining loop statements are skipped and computation proceeds directly to the next pass through the loop. This statement is given as :

FLOW CHART :-



Example:-

Sum of two positive integers.

```
#include<stdio.h>
  main()
    {
        int i=0,num,sum=0;
        for(i=0; i<5; i++)
            {
                printf("Enter any values");
                scanf("%d",&num);
                    if(num<0)
                      {
                        printf("It is negative number");
                        continue;
                      }
                sum+=num;
            }
        printf("Sum of +ve number is %d",sum);
   }
```

CASTING A VALUE :-

The process of converting one data type to another data type is called casting a value. To cast a value we have to follow the below syntax:

```
[var=]data type (exp);
x=float( );
```

ex:

```
x=10/3                      x =float(10)/3
value in x is 3.000         x=3.3333
```

OPERATOR PRECIDENCE:-

The precidenci of operators will always be considered by the compiler when evaluating an expression. The precidenci of operators is as shown below.

```
( )
.
*,/
+,-,%
<, >, >=, <=, ==, !=
&&
||
!
```

FUNCTIONS:-

A number of functions looped in to a single logical unit is referred to as a function.

STRUCTURE OF A FUNCTION:-

```
Return type function name(asrgs list)
{
        body of the function
}
```

Example:-

```
        #include<stdio.h>
        int cube (int  i)          Function declaration.
                                   Parameter type.
            {                      Parameter.
                int ret val;       Return type.
                                   Local variable.
                ret Val=i*i*i;
                  return ret val;
            }
         main( )
           {
            int a;   ────────▶ Local variable.
            printf("Enter value ");
            scanf("%d",&a);
           printf("The cube of value of %d is %d",.a, cube(int a));
        }                                                 │
                                                          ▼
                                                    Function call.
```

PARTS OF A FUNCTION:-

i. Function declaration.
ii. Function definition.
iii. Return Statement.
iv. Function Call.

FUNCTION DECLARATION:-

A function declaration provides the following information to the compailer.

i. The name of the function.
ii. The type of the value returned.
iii. The number and the type of arguments that must be supplied in a call to the function.

When a function call is encounted the compailer checks the function call with its declaration. A function declaration is has the following syntax.

SYNTAX:

Return type function name (type);

Return type specifies the data type of the value in the return statement. The function can written in any data type. If there is no return value the key void is placed before the function name. Function declarations are also called prototypes.

FUNCTION DEFINATION:-

It is similar to the function declaration but doesn't have the semicoloun the first line in the function def is called function declaration. This is followed by the function body. It is compared of the statement that makes up the function, delimited by braces. The declaration & declaration must use the same function name, no of arguments, argument type & the return type. No function definition is allowed with in a function definition.

Syntax:-

Return type Function name (arguments);

FUNCTION RETURN VALUE: -

Functions in ‘c’ may or may not have return values. If a function doesn’t return a value, the return type in function definition and declaration is specified as valid.

FUNCTION CALL: -

A function call is specified by the function name followed by the value of the parameter enclosed with in parenthesis, terminated by the semicoloun (;).

Example:-

Maximum of three numbers

```
#include<stdio.h>
          int max func(int I,int j, int k)
             {
                int max;
                    if(i>=j && i>=k)
                            max=I;
                    else
                        if(j>=k)
                            max=j;
                          else
                             max=k;
                     return max;
             }
main()
       {
           int a,b,c,m;
           printf(“Enter the values for a,b,c”);
           scanf(“%d%d%d”,&a,&b,&c);
      m=maxfunc(a,b,c);
           printf(“Maximum of %d%d%d is %d”,a,b,c,m);
      }
```

FUNCTION PARAMETER: -

These are the means of communication between the calling & called functions. They can be classified in to former parameters & actual parameters. The former parameters are the parameters given in the function declaration & function definition. The actual parameters, often known as arguments are specified in the function call.

CATEGORY OF FUNCTIONS:-

The function categories are of three types:

I. Function with no arguments and no return values.
II. Function with arguments and no return values.
III. Function with arguments and with return statements.

I.FUNCTION WITH NO ARGUMENTS AND NO RETURN VALUES: -

When a function has no arguments it does not receive data from the calling function. Similarly when it doesn't return a value the calling function does not receive any value. It effect there is no transfer of data between calling and called functions.

II.FUNCTION WITH ARGUMENTS AND NO RETURN VALUES:

When a function has arguments it must receive data from the calling function. It this category data will be transfer in a one-way direction. From calling function to called function.

III.FUNCTION WITH ARGUMENTS AND WITH RETURN STATEMENTS:

It this category data will be transfer both the ways, from calling to called function and called to calling function.

ADVANCED FEATURES OF FUNCTIONS:-

I. Function declaration and proto type.
II. Calling function by value or by reference.
III. Recursive functions.

I. FUNCTION DECLARATION AND PROTO TYPE.

Any 'C' function by default returns an integer value. To return a value other than an integer be have to specify the proto type declaration.

Syntax:-

Return type function type (argument);

In the above syntax return type specify the type of value return by the function.

Function name is the name of the function and arguments are the data types of the values to be passed to the function.

The prototype declaration must be specified the declaration part of the calling function. This declaration specifies the following.

Name of the function.

Return type of the function.

The number and data type of the arguments.

Ex:

```
#include<stdio.h>
main( )
 {
    int a,b,c;
    int power (int, int );
    c=power(a,b);
 }
 power int x, int y)
       {
          float abc (int, int);
           abc (x,y);
       }
  float abc(int k, int l)
    {
 }
```

II. CALLING FUNCTIONS BY VALUE OR REFFERENCE:-

Passing the values of variables from one function to another function is called call by value.

CALL BY REFERENCE:-

In a function call statement if we pass the address of a variable then such a function call is called as call by reference.

III.RECURSION METHOD:-

Expressing any intity interms of itself is called RECURSION. In ‘c’ a function can call any function that has been defined including itself.

Recursion:-

Expressing an intity in terms of itself is called RECURSION. In ‘c’ a function can call any function that has been defined including itself.

Example:-

Factorial of a Numberusing Recursion.

```
#include<stdio.h>
  main()
   {
        int n;
        long int x;
        printf(“Enter the value for n”);
        scanf(“%d”, &n);
        n=fact(n);
        printf(“Factorial of %d is %d”, n,x);
  }
         long int fact(int num)
             {
                if (num==0)
                        return 1;
                else
                        return num*fact(num-1);
             }
```

STORAGE CLAUSES:-

The storage clause of a variable indicates the allocation of storage place to the variable. Storage clauses are of four types:

i. Auto.
ii. Register.
iii. Static.
iv. Exterm.

AUTO Variable:-

All variables declared with in a function are AUTO by default. Variables declared AUTO can only be accessed only with in the function or the nested block with in which they are declared. They are created in to and destroyed when it is exited.

Note:- All the programs we are written are taken or treated as AUTO variables .

REGISTER VARIABLE: -

There are stored in the register of the microprocessors. The number of variables which can be declared register are limited. If more variables are declared they are treated as AUTO variables. A program that uses REGISTER variables executes faster as compared to a similar program without register variables.

STATIC VARIABLES: -

Static variables are of two types:

i. Static variables that are declared with in a function.

ii. File static Variable.

STATIC VARIABLES THAT ARE DECLARED WITH IN A FUNCTION:-

These variables retain their values from the previous call.
i.e. the values which they had before returning from the function.

FILE STATIC VARIABLE :-

These are declared outside any function using the keyword STATIC. File static variables are accessable only in the file in which they are declared.

Example:-

```
  #include<stdio.h>
     print count()
       {
         static int count=1;
         printf("count=%d", count);
         count=count+1;
       }
    main()
      {
         print count();
         print count();
         print count();
      }
```

EXTERM :-

When a program scans across different files we want to have a global variable accessible to all functions in these files, the key word EXTERM should be used before the data type name in the declarations in all the files where it is accessed except one. The linker requires that only one of these files have the definition of the identifiers. Global variables definitions can occur in one file only .

ARRAYS:-

An array is a sequence of data in memory where in all data are of the same type, and are placed in physically adjacent location. A string can be considered as sequence of characters.

Arrays are classified as follows

One-dimensional array.
Two- dimensional array.
Multi dimensional array.

ONE-DIMENSIONAL ARRAY.

This is an array which as only a single row.

To declare one-dimensional array we have to use the following syntax.

Data type argument [size];

Syntax:

return type variable name[size of the array];

Example:-

Average of N Numbers.

```
#include<stdio.h>
main()
{
  int a[15],I,n;
  float sum=0,avg;
  printf("Enter the range of the array");
  scanf("%d",&n);
  printf("Enter the values");
  for(i=0;i<n;i++)
     {
          scanf("%d";       &a[i]);
          sum=sum+a[i];
      }
```

```
            for(i=0;i<n;i++)
            printf(“%4d”,a[i]);
            avg=sum/n;
            printf(“avg=%f”,avg)
}
```

TWO- DIMENSIONAL ARRAY.

An array is used to store data in more than one row and one coloumn is called two-dimensional arrays.

Syntax:

Data type array name [rows][columns];

Example:-

matrix addition using arrays

```
#include<stdio.h>
main()
   {
        int i, a[10][10], b[10][10], c[10][10], j,m,n,p,q;
        printf(“Enter the size of the array A”);
        scanf(“%d%d”,&m,&n);
        printf(" Enter the values of the matrices A");
        for(i=0; i< m; i++)
                for(j=0; j< n; j++)
                   {
                       scanf("%d",&a[i][j]);
                   }
        printf(“Enter the size of the array B”);
        scanf(“%d%d”,&p,&q);
        printf(" Enter the values of the matrices B");
        for(i=0;i< p;i++)
                for(j=0;j< q ;j++)
                   {
                       scanf("%d",&b[i][j]);
                   }
        for(i=0;i< m;i++)
                for(j=0;j< q;j++)
                  {
```

```
                c[i][j]=a[i][j]+b[i][j];
              }
         printf("The Matrix sum is ");
           for(i=0;i< m;i++)
              {
                for(j=0;j< n;j++)
                   {
                        printf("%d\t",c[i][j]);
                    }
                  printf("\n");
               }
     }
```

Multi – Dimensional Array:-

C allows us of three or more dimensions. The exact limit is determined by the compailer. The general form of this is as follows

Syntax:-

Type name array name [s1][s2][s3]......[si];

Where si is the size of the ith dimension.

Example:-

Int [3][4][7];

STRINGS :-

Strings are arrays of characters i.e. they are characters arranged one after another in memory. To make the end of the string, 'c' uses the null character. Strings in'c' are enclosed with in double quotes.

STRING FUNCTIONS:-

There are four types of string functions :-

I Str cat.
II Str cmp.
III Str cpy.
IV Str len.

To use these functions the header file string.h must be include in the program with the statement.

STR CAT:- (string concatenation)

This function concotinates two strings i.e. in appends one string and that of another. The function accepts two strings as parameters and stores the contents of the second string at the end of the first.

Example:-

```
#include<stdio.h>
#include<string.h>
main()
  {
        char str1[20]="phanindra";
        char str2[20]="chowdary";
         printf("String1=%s",str1);
         printf("String2=%s",str2);
         strcat(str1,str2);
         printf("After concatenating the string is %s→", str1 ");
 }
```

STR CMP: - (string comparision)

The function ctr cmp compares two strings. This functionc is useful while writing programs for constructing and searching string as arranged in a directory. The function accepts two strings as parameters and returns an integer whose value is less than zero if the first string is less than the second.

=0 If both are identical.

>0 If the first string is greater than second.

The function str cmp compares the two strings, character by character, to decide the greater one. When ever the two characters in the string differ , the string which has the character, with a higher ASCII value is greater.

Example:-

```
#include<stdio.h>
#include<string.h>
main()
  {
        char str1[20]="phanindra";
        char str2[20]="phanindra chowdary";
        int result;
        result=strcmp(str1,str2);
        if(result>0)
       printf("String1 is greater than string2 =%s,%s",str1,str2);
    else
```

```
        if(result==0)
              printf(“String1is equals to string2=%s,%s”,str1,str2);
        else
               printf(String1 is leess than string2 =%s,%s”,str1,str2);
  }
```

STR CPY:-   (string copy)

The str cpy function copies one string to another. The function accepts two strings as parameters and copies the second string character by character in to the first one, up to and including the null character of the second string.

Example:-

```
#include<stdio.h>
#include<string.h>
main()
  {
         char str1[20]=”phanindra”;
         char str2[20]=”chowdary Maganti”;
          printf(“Before copyingString1,string2=%s,%s”,str1,str2);
          strcpy(str1,str2);
          printf(“After copying the string is  %s→”, str1 ”);

  }
```

STR LEN :-   (String length)

This function returns the integer which denotes the length of the string passed. The length of the string is the number of characters percent in it. Excluding the terminating the null character.

```
#include<stdio.h>
#include<string.h>
main()
  {
         char str1[20]=”phanindra”;
         int len;
         len=str len(str1);
         printf(“The length of the String1=%s”,len);
  }
```

POINTERS:-

Memory is organized as a sequence of byte-sized notation. These bytes are numbered beginning with a zero. The number associated with a byte is known as its address or memory location. A pointer is an intity which contains a memory address. A pointer is a number, which specifies a location in memory.

Each point in memory is associated with a unique address.

An address is an integer having fixed number of bits, labeling a byte in the memory.

Addresses after positive integer values that range from zero to some positive integers constant corresponding to the last location in the memory.

Every object i.e. loaded in memory is associated with a valid range of address.

The size of the data type is referenced by the pointer is the no of bytes that may be accessed directly by using that address.

USES OF POINTERS:-

i. Accessing Array elements.
ii. Passing arguments through functions.
iii. Passing arrays and strings through functions.
iv. Creating data structures such as linked lists, trees etc.
v. Obtaining memory from the system dynamic memory.

Example:-

Working of a pointer using addtresses.

```
#include<stdio.h>
  main()
    {
      int a=5,b=10,c=20;
      printf(“Address of %d is %d”,a,&a);
      printf(“Address of %d is %d”,b,&b);
      printf(“Address of %d is %d”,c,&c);
```

```
        }
```

Example:-

Working an variable using pointer.

```
 #include<stdio.h>
main()
  {
     int *ptr;
     int var1,var2;
     var=100;
   var=200;
   ptr=&var1;
   printf(“Address and contents of var1 is %d”,ptr,var1);
   ptr=&var2;
   printf(“Address and contents of var2 is %d”,ptr,var2);
   ptr=125;
   printf(“Address and contents of var2 is %d”,ptr,var2);
   var1=*ptr+1;
   printf(“Address and contents of var1 is %d”,ptr,var1);
}
```

Address operator:-

All the variables defined in a program resisted specific address. These possible to obtain the address of a program variable by using the address operator (&) ampersand.

DE-REFERENCING POINTER:-

It is an operator performs to access and manipulate data contained in memory location pointed to by a pointer. The operator (*) is used to de-referencing pointers. A pointer variable is de-referenced when the unary operator (*) in this case as a prefix to the pointer variable or pointer expression.

Example:-

```
 #include<stdio.h>
  main()
    {
        int *ptr,var1,var2;
        *ptr=25;
        *ptr+=10;
```

```
            printf("variable var1 contains %d",*ptr);
            vae2=*ptr;
            ptr=&var2;
            *iptr+=20;
            printf("variable var2 contains %d",*ptr);
     }
```

Example:-

```
  #include<stdio.h>
  main()
    {
        float data1,data2;
        float swap(float,float);
        printf("Enter any two floating type values");
        scanf("%f%f",&data1,data2);
      printf("Before swaping the numbers are %f,%f", data1,data2);
      swap(&data1, &data2);
      printf("After swaping the numbers are");
      printf("data1=%f,data2+%f",data1,data2);
 }
   float swap(float *d1, float *d2);
    {
       float temp;
       temp=*d1;
       *d1=*d2;
      *d2=temp;
    }
```

POINTERS WITH ARRAY:-

The elements of an array can be efficiently accessed by using pointer.

(1). Array elements are always stored in continuos memory location irrespective of the size of Array.

(2). The size of the data type which the pointer variable prefers to, is dependent on the data type pointed to by pointer.

For example if the pointer variable points to an integer data type, then it access two bytes in memory.

(3). A pointer when incremented, always points to a location after skipping the no of bytes required from the data type pointed to by it.
For example a pointer variable points to an integer type at a certain address 1000,then on incrementing the pointer variable if points to the location 1002.

Eample:-

Searching the smallest element of an ARRAY using POINTER

```
 #include<stdio.h>
 main()
  {
       int *ptr, a [20],i,n, small;
      printf(“Enter the size of the array”);
      scanf(“%d”,&n);
      printf(“Enter the Elements of the array”);
      for(I=0;I< n;I++)
         scanf(“%d”,&a[I]);
       ptr=a;
       small=*ptr;
       ptr++;
    for(I=1;I< n;I++)
        {
           if(small>*ptr)
           small=*ptr;
           ptr++;
       }
     printf(“Smallest element is %d”, small);
  }
```

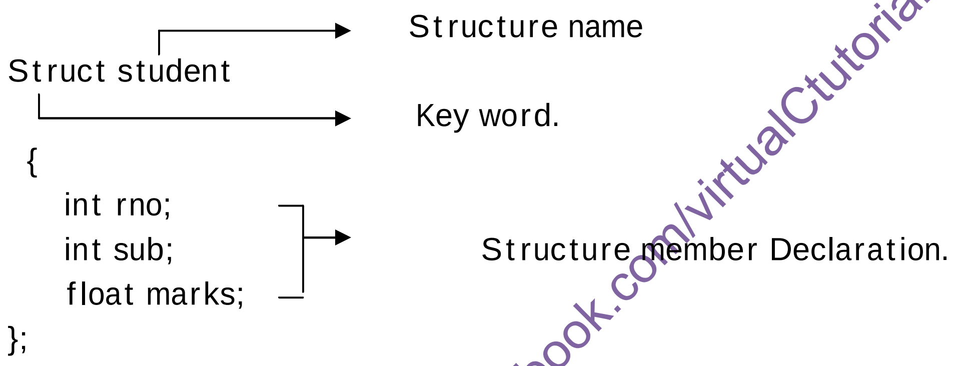
POINTER TO POINTER:-

Every pointer created in the memory has an address. This address can be stored in another pointer variable . the pointer which stores the address of another pointer is called pointer to pointer. To declare a pointer to pointer variable we have to use the following syntax.
Data type ** pointer ;
Ex:

```
int **k;
```

STRUCTURES :-

A structure in 'c' is a hetrogenous user defined data. A structure may contain different data types. It groups variables in to a single antity.



```
Struct student
 {
    int rno;
    int sub;
     float marks;
};
```

RULES FOR DECLARATION OF INDIVIDUAL MUMBERS:-

The individual members of a structure may be any of common data types such as int, float,..... Etc, pointers, arrays or even other structures. All member names with in a particular structures must be different. Individual members cannot be initialized inside the structure declaration.

DECLARING STRUCTURE VARIABLE:-

(1). IN THE STRUCTURE DECLARATION:-

The structure variable can be specified after the closing braces.

```
struct student
{
    int rno;
  int sub;
    float marks;
  } student 1, student 2;
```

Student is the structure tag. Structure 1 and structure 2 are variables of type student. If other variables of the structure are not required the tag name student can be omitted.

(2). USING THE STRUCTURE TAG:-

The variables of the structure may also be declared separately by using the structure tag.

Struct student student 1, student 2;

student 1 and student 2 are structure variables of the type indicated by the structure tag.

STRUCTURE INITIALISATION:-

```
Struct student
  {
     char name(25);
    int rno;
    int sub;
     float marks;
 };
```

A variable of this structure can initialized during its declaration as shown below.

Struct student student 1={“AAA”;3205,2,90.5};

STRUCTURE WITH IN STRUCTURE:-

```
struct date
       {
     int day;
     int month;
     int year;
  };
struct person
    {
     char name[25];
     struct date birthday;
      float salary;
    };
```

The embaded structure type (date) must be declared before its use with in the containing structure.

This is because, only legal data types are allowed in the structure declaration.

OPERATIONS ON STRUCTURE:-

'C' provides the period or dot(.) operator to access the members of a structure independently. The dot operator connects the member with the structure variable. This can be represented as

struct var. member var;

Her "struct var" is a structure variable and "member var" is on is one of its member. The dot operator must have a structure variable on its left and a legal member name on its right.

```
Struct person
    {
       char name [25];
        int age;
        float salary;
    } emprec;
```

emprec is a structure variable of type person and name, age, salary are the members of the structure. Emprec. Name will access emprec's name. Emprec. Age will access emprec's age.

ARRAY OF STRUCTURE:-

The array will have individual structures as its elements. Similar to declaring structure variable.

```
Struct person
            {
              char name[25];
              struct date;

                  {
                      int day, month, year;
                  } birthday ;
              float salary ;
          } emprec[10];
```

emprec is an array of 10 person structures. Each element of the array emprec will contain the structure of type person. The person structure consists of three individual members. An array name, salary and another structure data.

"emprec[4].name" access the name of 5th structure.

TYPE DEF STATEMENT:-

Type def statement is used for defining new data types involving structures. A new data type representing the structure is declared using the type def key word.

```
Type def struct student struct
    {
      int roll no ;
      int subject ;
      int marks ;
    } student ;
```

Here student struct is the tag. Student is the name of the new type.

Student student 1, student 2;

Student 1, student 2 are structure variables.

STRUCTURE OF FUNCTIONS :-

A structure variable can be passed to a function in the same manner as that of an ordinary variable. in the function call statement we specify the structure variable as an argument. When we pass a structure variable to a function the entire contents of the structure variable will be passed to the function. The structure variable declare in the formal parameter must belong to the same structure as that of the actual variable.

When ever we pass variable to a function the structure must be declared global, this is because the structure definition must be available to all the functions in the program.

UNIONS:-

A union is a type in ‘c’ which allows the over lay of more than one variable in the same memory area.

Union is similar to a structure. The only change in the declaration is the substitution of the key word “UNION” for the key word “STRUCT”.

```
Union [<union type name>]
    {
       < type > <variable name>;
           ..........
           ..........
     }[< union variable>];
```

All variables inside a union share storage space. The compailer will allocate sufficient storage for the union variable to accomidate the largest element in the union. Other elements of the union use the same space individual variables in a union occupy the same location in the memory. Thus writing into one will overwrite the other. Elements of a union or accessed in the same manner as the elements of a structure.

DIFFERENCE BETWEEN STRUCTURE AND UNION:-

The amount of memory required to store a structure variable is the sum of sizes of all the members in addition to the padding bytes that may be provided by the compailer.
In case of a union, the amount of memory requiring is the same as that required by its largest member.

SIZE OF ( ) OPERATOR:-

It is also called compailer operator or compile type operator. It displays no of bytes covered by a variable or an expression. The syntax of size of operator is

n=size of (v);

Here n should be of integer type.
Here v is variable.

TYPE OPERATOR:-

Type operator is used for conversion purpose. So it is called as convert operator. This operator converts float type data in integer from converts float type data in integer from and vice versa. So it is also used for casting a value and process to convert one from to another is called casting. she syntax of type operator is

(type)v or e

Where v is variable.
e is an expression.

FILES:-

A file is a group of data items or information stored some way on the secondary storage device.
To store the data given to a particular program on a secondary storage device we have to use the concept of FILES. Once we store the data permanently that data can be used from the files when ever necessary.

To do file operations we must declare a pointer of type FILE. This is because a file can be accessed only through its pointer.

Syntax:

File * pointer;

Ex:

File * fp;

The operations that can be perform on a file or reading and writing. To perform any operations on a file we have to follow the following steps.

i. select a sutiable data structure for the file .
ii. select a sutiable for the file and open it.
iii. Perform the operation on the file.
iv. Close the file.

OPENING A FILE :-

To open a file we have the function fopen( ); whose syntax is as shown below. This function opens the specified file and returns a pointer if the file has been successfully opened. Otherwise it returns a null value.

Syntax:

File pointer= fopen ("file name","mode");

In the above syntax file name is the name of the file which is to be opened. Mode is the purpose for which we want to open the file. The different file opening modes are

W, r, a, w+, r+, a+.

W :- Opens a file for writing only. If the file exists the contents in the FILE will be deleted and the new contents will be return to the FILE. If the FILE doesn't exist a new FILE will be created.

R :- Opens a file only for reading purpose.

A :- Opens a file in append mode. This means that we can add contents to a file from the end of that file. When a file is opened in this mode the existing data will remain the same and the new data will be added from the end. If the FILE doesn't exists a new file will be created.

W+:- Identical to w. but allows both reading and writing oper&ations.

R+ :- Opens a file for reading and writing.

A+ :- Opens a file in append mode and allows reading also.

READING FROM A FILE :-

To read values from a file. We have the function f scanf( );.

Syntax:

fscanf(file pointer,"control string",& variable list);

in the above syntax file pointer specifies the FILE from which data is to be read. Control string and variable list are same as in scanf( );

ex:

fscanf(fp, "%d%s%d",&emp.id,emp.name,&emp.sal);

WRITING A FILE: -

To write a file ,we have the function fprintf ( );

SYNTAX :-

Fprintf (file pointer,"control string",& variable list);

In the above syntax file pointer specifies the file where the contents or to be written. Control string and variable list are same as that in printf( );

Ex: printf(fp,"n%d\n%d\n%s", emp.id,emp.sal,emp.name);

CLOSING A FILE:-

To close a file we have a function called fclose( ); this function takes one argument which is a file pointer. This function close
the file which is associated with the specified file pointer.

Syntax:

Fclose(file pointer);

Ex:

Fclose (fp);

DETECTING END OF FILE:-

To detect the end of the file we have the feof ( ). This function returns n true values if it has reached the end of the file otherwise it will return a false value.

Syntax: Feof (file pointer);

File pointer specifies the file for which we want to detect.

RANDOM ACCESS OF FILE:-

This means locating the file pointer at a desired position in the file. To achieve this task we have the fseek( );

SYNTAX:

Fseek (fp, no, ofby, position);

FILE POINTER:-

This parameter specifies the file pointer which we want to move.

This parameter specifies the no of bytes by which we want to move the file pointer. If we specify a positive value file pointer will be move in the forward direction. If we specify the negative value it will move in the back ward direction.

Ftell( ); :-

This function returns the current position of the FILE pointer in the FILE, i.e. no of bytes it is away from the starting point of the FILE.

Syntax:-

Var=ftell (file pointer);

In the above syntax FILE pointer is the pointer whose current position is to be known. Since this function returns a long int type of value the variable should belong to long int data type.

Ex:-

X=ftell (fp);

ENUMERATED DATA TYPE:-

Enumerated data type is the data type defined by the user it allows the user to define this own data type. To declare the enumerated the enumerated data type we use the key word enum.

Syntax:-

Enum name {constant};

Ex:-

Enum colour {red, green, blue}