Lesson 1

Programming with C

Programming Techniques:

A set of statements is known as program. In other words, it can be said as a meaning full set of statements which are used to done some work in computer are known as "**program**".

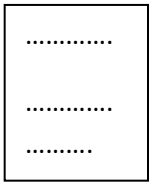
There are three types of programming techniques:

1.Linear programming technique.

2. Structured Programming Technique

3.Object Oriented Programming Technique.

1.Linear programming technique:- In this technique, any program can be written within single block. It means it provides a single unit or block in which we write whole code of program, either the program is small or big.



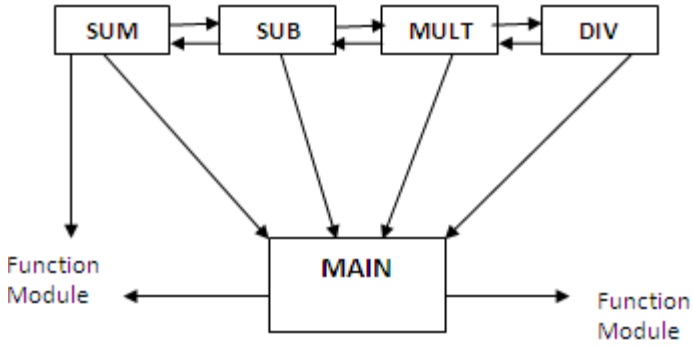
In these techniques, it is very difficult to develop application software, because we can't do large program by these techniques.

Ex:BASIC is the language,which follows the concept of linear programming techniques.

2.Structured Programming Technique:- In this technique, whole program can be divided into different part and each part is known as Module or Function. That's why it is known as Modular programming.

After developing structured programming techniques , programmer can do any program very easily. He can debug the program very easily. And also the program is very easily understandable through this technique.

At last, we connect all the part with each other or with the main program to create a single task.



Program format:-

3.Object Oriented Programming:- This is new technique, which is the extended from the structure programming technique. The programming base of this technique is same as structured programming technique. This is the concept of module or functions is also used in OOPS.

But the difference between STRUCTURE & OOPS is that in any program that is done in OOPS by using the concept of class & object but there is no any concept of class & object in structured programming techninque.

Ex: C++, JAVA, C#.

Introduction of C

C is developed by Dennis Ritchi in 1970 at AT&T Bell Labs of USA.

We can do high level as well as low level programming in "C". That's why it is known as middle level language.

Hence in other words we can also say that C provides the facility to do any program in structured way e.i we can divide the program in different parts and each part is interconnected either from main function or to each other.

Importance of 'C' Language :-

1. One of the most popular computer languages.
2. Programs written in 'C' language are efficient and fast (Fastly Execute).
3. 'C' is highly portable language that means programs written for one computer can be run on another computer with little or no modification.
4. 'C' is well suited for structured programming and modular programming.
5. 'C' is well suited for writing system software and business package. As – Tally.
6. Support low level programming instructions as well as high level instructions.

Features of 'C' Language

1. **Availability** :- 'C' language complier contains the codes of almost of all operating system. It uses ANSI which contains the specification of 'C' language for all operating system, which is supported by almost complier.

2. **Efficiency** :- 'C' language supports low level programming language as well as high level programming language, so that this feature produce highly efficient code for program.

3. **Data Type** :- Basically 'C' provides two data type.

   a. Int. (Integer)
   b. Float

   Another two data type one is character second is double are inherited by 'C' language. By Default 'C' uses int data type. It provides derived data type which can be created with pointers, array, structures and unions.

4. **Function** :- A program is a collection of one or more functions. Every 'C' program contains only one function known as main () function. 'C' supports only 'call by value ' Parameter passing mechanism.

5. **Multiple File Handling** :- A 'C' program may be splited into multiple source file. It is generally used in 'Project Development'.

6. **Recursion :-** In this mechanism the function can be used to call itself. It can reduce the programming instructions.

Structure of C programming

[Documentation System] :- In this section name of program and programmer can be defined as a comment of 'C' language.

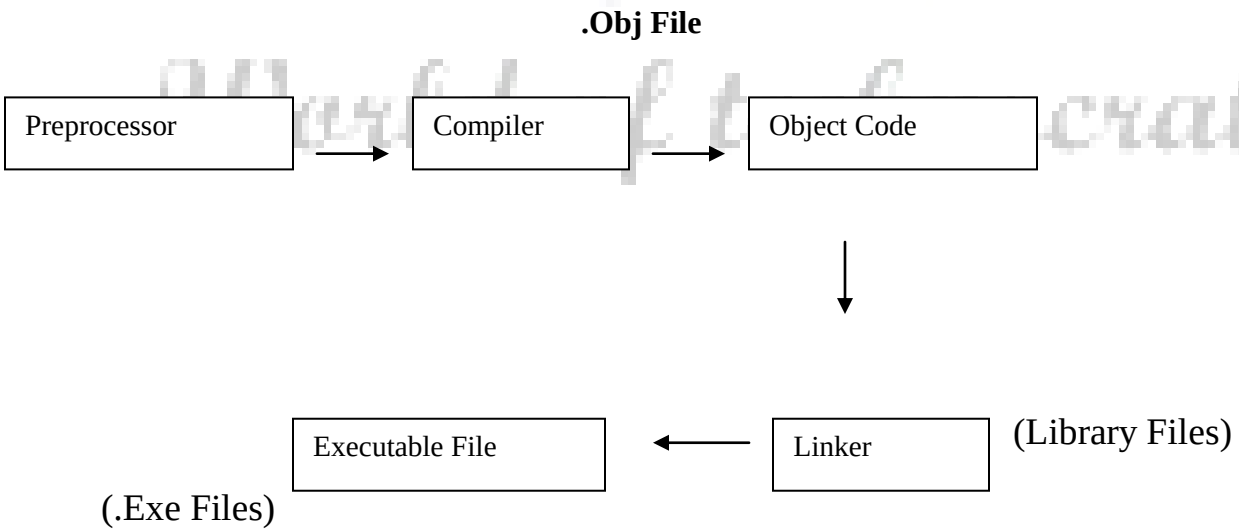
[Declaration Section] :- The prototype of function and files can be defined in this section.

[Global Declaration] :- We can define variables globally in this section. Global variables are used anywhere in 'C' program. Global variables are declared outside the main function.

[Local Declaration] :- Local variables are defined in this section. Local variable is used only in some functions, in which that is defined but not somewhere else in 'C' program.

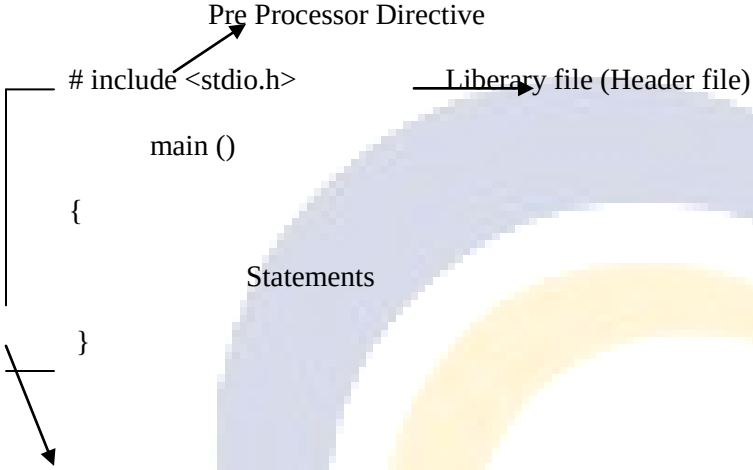
[Sub Program Section] :- Definition of functions are defined after main function.

Flow of execution of C Program



Sample Programs : -

File .C



Source Program

Pre Processor Directive :- It is the instruction of compiler itself before actual compilation process starts. It always starts with has h sign (#). The "#" include directive is used to insert the contents of specific file at the position of "# include" line before compilation. Various directives - # include, # define, # if, # alse, # endif, # ifdef, # ifndef etc.

Instructions :- 'C' program consist of many instruction and keyword. Basically four types of instruction in 'C' –

(1). Type Declaration instruction (as int a;)

(2). Input/Output instructions (

(3). Arithmetic instructions (as - c = a+b)

(4). Contral Instructions (All statements dks Flow esa fy[kuk)

constant :- Value that is fixed daring programming execution file.

(i). Primery constant (a=40, 40 Primery constant gksxk)

(ii). Secondary constant (Const Int a;)

Variables :-

Variables are those quantities whose value can be changed within a program . Variables are used to store constant value in memory because we cannot store constants directly in memory. Memory space is always allocated by the variable name & then this space is used to stpore the constant values.Predefined words cannot be variable. As – if, else, while etc.

Rules of difining variable:

- Variable name may be of 32 characters in length.
- Variable name may be either elphabet or alphanumeric.
- Numeric values cannot be used as variable name.
- Spaces are not allowed, But special characters like underscore(_) can be used in variable name.

Note: - No length restrictions for variables. Length limitation for variable depending on compiler.

Keywords:- Keywords are the words whose meaning has already been explained to the C compiler. The keyword cannot be used as variable names. Ex. Char, const,void ,long etc.

Lesson 2

Data Types and Operators

Data Types

set of some attribute which can be applied on particular value is called data type. Four types of data types:-

1) **Int :-** It allows to store positive or negative value.
2) **Char :-** It is use to store any character value.
3) **Float :-** float can store a single precision floating point (real number)
4) **Double :-** It can store a double floating point.

Additional qualifiers allowed for data type :-

1) Short
2) Long
3) Unsigned/Signed

1. Short :- By default integer value takes two bytes and short allows two byte integer value.

Syntax :- short [int] <variable>;

Example :- short a; /short int a;

2. Long :- It allows four bytes integer value.

Syntax :- Long [int : float : double] <variable>

Example :- Long b; / Long int : float : double b;

3. Unsigned/signed :- By default int takes sign value unsigned allows to assign value without sign but signed allows to assign value with sign.

Syntax :- unsigned / signed [int] <variable>

S.No.	Data types	Sign (Inbit)	Range	Format
1.	Char/signed char	8	-128 to 127	%C
2.	Unsigned ’’	8	-0 to 255	%C
3.	Int/signed int	16	-32768 to 32767	%D
4.	Unsigned Int	16	-0 to 65535	%U
5.	Long Int/ unsigned long int	32	-2147483648 to 2147483647	%Ld
6.	Unsigned long int	32	-0 to 4294967295	%Lu
7.	Short int	16	-32768 to 32767	%Lh
8.	Float	32	Range of long int	%F
9.	Long Float / Double	64	’’	%Lf
10.	Long Double	80	’’	%Lf

Here is first c program named as first.c .

C Language is a case sensitive language.

```
#include <stdio.h>

#include<conio.h>
void main()

{
```

```
Clrscr();
printf("Hello World\n");

Getch();
}
```

Press ALT+F9 to compile your program. If you have any error in your program, you will get the message, remove your errors and then execute your program you will got the output.

Hello World
printf()
The printf() function prints output to stdout, according to format and other arguments passed to printf(). The string format consists of two types of items - characters that will be printed to the screen, and format commands that define how the other arguments to printf() are displayed.

```
printf( "Hello World' );
```

scanf()
The scanf() function reads input from stdin, according to the given format, and stores the data in the other arguments. It works a lot like printf(). The format string consists of control characters, whitespace characters, and non-whitespace characters.

```
void main(void)
{
int i;
scanf("%d",&i);
printf("%d",i);
}
```


Operators

Operators are used to manipulate the operand (data). Or it is an instruction through which the inputted data is processed. And after processing of data, system produces output according to instruction.

There are eight types of operators supported by 'c':-

- **Arithmetical operator**
- **Relational Operators**
- **Logical operators**
- **Assignment operators**
- **Increment/Decrement operators(unary operators)**
- **Conditional operators**
- **Bit wise operators**
- **Special Operators**

Arithmetical Operators

The arithmetic operator is a binary operator, which requires two operands to perform its operation of arithmetic. Following are the arithmetic operators that are available. Ex. C=a+b;

Operator	Description
+	Addition
-	Subtraction
/	Division
*	Multiplication
%	Modulo or remainder

Relational or Comparison operator

Relational operators compare between two operands and return in terms of true or false i.e. 1 or 0. In C and many other languages a true value is denoted by the integer 1 and a false value is denoted by the integer 0. Relational operators are used in conjunction with logical operators and conditional & looping statements.

Following are the various relational operators

<	Less than
<=	Less than or equal to

>	Greater than
>=	Greater than or equal to

Closely related to the relational operators is the equality operator as follows:

==	Equal to
!=	Not equal to

Logical Operators

A logical operator is used to compare or evaluate logical and relational expressions. There are three logical operators available in the C language.

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Assignment Operator

An assignment operator (=) is used to assign a constant or a value of one variable to another.

Example: a = 5; b = a;

Interest rate = 10.5

Result = (a/b) * 100;

Remember that there is a remarkable difference between the equality operator (==) and the assignment operator (=). The equality operator is used to compare the two operands for equality (same value), whereas the assignment operator is used for the purposes of assignment.

Multiple assignments:

You can use the assignment for multiple assignments as follows:

a = b= c = 10;

At the end of this expression all variables a, b and c will have the value 10. Here the order of evaluation is from right to left. First 10 is assigned to c, hence the value of c now becomes 10. After that, the value of c (which is now 10) is assigned to b, hence the value of b becomes 10. Finally, the value of b (which is now 10) is assigned to a, hence the value of a becomes 10.

Arithmetic Assignment Operators

Arithmetic Assignment operators are a combination of arithmetic and the assignment operator. With this operator, the arithmetic operation happens first and then the result of the operation is assigned.

Example	Expands as
Sum+=3	Sum = sum + 3
Count-=4	Count -= 4
Factorial*=num	Factorial = factorial * num
Num/=10	Num = num /10
A%=3	A = a % 3

Unary Operators

Increment and Decrement Operators

These operators also fall under the broad category or unary operators but are quite distinct than unary minus. The increment and decrement operators are very useful in C language. They are extensively used in for and while loops. The syntax of these operators is given below.

++,--

The ++ operator increments the value of the variable by one, whereas the -- operator decrements the value of the variable by one.

These operators can be used in either the postfix or prefix notation as follows:

Postfix: a++ or a--
Prefix: ++a or –a

Postfix notation

In the postfix notation, the value of the operand is used first and then the operation of increment or decrement takes place, e.g. consider the statements below:

```
int a,b;
a = 10;
b = a++;
printf("%d",a);
printf("%d",b);
```

Program Output

```
10
11
```

Prefix notation

In the prefix notation, the operation of increment or decrement takes place first after which the new value of the variable is used.

```
int a=10,b;
```

```
b = ++a;
printf("%d",a);
printf("%d",b);
```

Conditional Operator

A conditional operator checks for an expression, which returns either a true or a false value. If the condition evaluated is true, it returns the value of the true section of the operator, otherwise it returns the value of the false section of the operator.

Its general structure is as follows:

Expression1 ? expression 2 (True Section): expression3 (False Section)

The variable c will have the value 2, because when the expression (a>b) is checked, it is evaluated as false. Now because the evaluation is false, the expression b-a is executed and the result is returned to c using the assignment operator.

Precedence of Operators

Operator	Associavity
Unary	Right to Left
Arithmetic Operators	Left to Right
Relational Operator	Left to Right
Equality Operator	Left to Right
Logical Operator	Left to Right
Conditional Operator	Right to Left
Assignment Operator	Right to Left
Comma operator	Right to Left

Lesson 3

Conditional Statements

Conditional statements are used to execute a statement or a group of statement based on certain conditions.

We will look into following conditional statements.

1. if
2. if else
3. else if
4. switch

1) if conditional statement:

Syntax for if statement in C :

```
if(condition)
{
Valid C Statements;
}
```

If the condition is true the statements inside the parenthesis { }, will be executed, else the control will be transferred to the next statement after if.

if.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
a=10;
b=5;
if(a>b)
{
printf("a is greater");
}
```

```
getch();
}
```

2) if else :

Syntax for if :

```
if(condition)
{
Valid C Statements;
}
else
{
Valid C Statements;
}
```

In if else if the condition is true the statements between if and else is executed. If it is false the statement after else is executed.

Sample Program :

if_else.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
printf("Enter a value for a:");
scanf("%d",&a);
printf("\nEnter a value for b:");
scanf("%d",&b);
if(a>b)
{
printf("\n a got greater value");
}
else
{
printf("\n b got greater value");
}
printf("\n Press any key to close the Application");
getch();
}
```

3) else if :

Syntax :

```
if(condition)
{
Valid C Statements;
}
else if(condition 1)
{
Valid C Statements;
}
else if(condition n)
{
Valid C Statements;
}
else
{
Valid C Statements;
}
```

In else if, if the condition is true the statements between if and else if is executed. If it is false the condition in else if is checked and if it is true it will be executed. If none of the condition is true the statement under else is executed.

else_if.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
printf("Enter a value for a:");
scanf("%d",&a);
printf("\nEnter a value for b:");
scanf("%d",&b);
if(a>b)
{
printf("\n a is greater than b");
}
else if(b>a)
{
printf("\n b is greater than a");
}
else
{
```

```
printf("\n Both a and b are equal");
}
printf("\n Press any key to close the application");
getch();
}
```

4) Switch statement:

The switch and case statements help control complex conditional and branching operations. The switch statement transfers control to a statement within its body.

Control passes to the statement whose case constant-expression matches the value of switch (expression). The switch statement can include any number of case instances, but no two case constants within the same switch statement can have the same value. Execution of the statement body begins at the selected statement and proceeds until the end of the body or until a break statement transfers control out of the body.

Use of the switch statement usually looks something like this:

```
switch ( expression )

{    Case expression:

     statements executed if the expression equals the

     value of this constant-expression ;

     break;

  default :

     statements executed if expression does not equal

     any case constant-expression;

}
```

Switch statements can also be called matching case statements. If matches the value in variable (switch(variable)) with any of the case inside, the statements under the case that matches will be executed. If none of the case is matched the statement under default will be executed.

switch.c

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

```
void main(){
int a;
printf("Enter a no between 1 and 5 : ");
scanf("%d",&a);

switch(a){
case 1:
printf("You choosed One");
break;
case 2:
printf("You choosed Two");
break;
case 3:
printf("You choosed Three");
break;
case 4:
printf("You choosed Four");
break;
case 5:
printf("You choosed Five");
break;
default :
printf("Wrong Choice. Enter a no between 1 and 5");
break;
}
getch();
}
```

Lesson 4

Control Structure (Looping)

"Loop means repetition of particular task for number of times. "

In every programming language, thus also in the C programming language, there are circumstances were you want to do the same thing many times. For instance you want to print the same words 10 times. You could type ten printf function, but it is easier to use a loop. The only thing you have to do is to setup a loop that execute the same printf function ten times. There are three basic types of loops which are:

- "for loop"
- "while loop"
- "do while loop"

The for loop

The "for loop" loops from one number to another number and increases by a specified value each time.

The "for loop" uses the following structure:

```
 for (Start value; end condition; increase value)
{
        statement;
}
```

Example:

```
        #include<stdio.h>
        int main()
        {
        int i;
        for (i = 0; i < 10; i++)
        {
        printf ("Hello\n");
```

```
printf ("World\n");
}
return 0;
}
```

Note: A single instruction can be placed behind the "for loop" without the curly brackets.

Let's look at the "for loop" from the example: We first start by setting the variable i to 0. This is where we start to count. Then we say that the for loop must run if the counter i is smaller then ten. Last we say that every cycle i must be increased by one (i++).

In the example we used i++ which is the same as using i = i + 1. This is called incrementing. The instruction i++ adds 1 to i. If you want to subtract 1 from i you can use i--. It is also possible to use ++i or --i. The difference is is that with ++i (prefix incrementing) the one is added before the "for loop" tests if i < 10. With i++ (postfix incrementing) the one is added after the test i < 10. In case of a for loop this make no difference, but in while loop test it makes a difference. But before we look at a postfix and prefix increment while loop example, we first look at the while loop.

The while loop

The while loop can be used if you don't know how many times a loop must run. Here is an example:

Syntax:

```
While(Condition)
{
Statement;
Increment/decrement;
}
```

Example:

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

```
i = 0;
while(i++ < 5)
{
printf("%d\n", i);
}
printf("\n");
i = 0;
while(++i < 5)
 {
printf("%d\n", i);
}
return 0;
}
```

The output of the postfix and prefix increment example will look like this:

```
1
2
3
4
5
1
2
3
4
```

i++ will increment the value of i, but is using the pre-incremented value to test against < 5. That's why we get 5 numbers.
++i will increment the value of i, but is using the incremented value to test against < 5. That's why we get 4 numbers.

The do while loop

The "do while loop" is almost the same as the while loop. The "do while loop" has the following form:

```
do
    {
        do something;
    }
    while (expression);
```

Do something first and then test if we have to continue. The result is that the loop always runs once. (Because the expression test comes afterward). Take a look at an example:

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


 counter = 5;
    do
    {
      printf("hello");
            counter--;
    }
    while ( counter !=0);
    return 0;
```

```
}
```

Note: There is a semi-colon behind the while line.

Break and continue

To exit a loop you can use the break statement at any time. This can be very useful if you want to stop running a loop because a condition has been met other than the loop end condition. Take a look at the following example:

```
#include<stdio.h>


int main()
{
        int i;


        i = 0;
        while ( i < 20 )
        {
        i++;
        if ( i == 10)
                break;
        }
        return 0;
}
```


In the example above, the while loop will run, as long i is smaller then twenty. In the while loop there is an if statement that states that if i equals ten the while loop must stop (break).

With “continue;” it is possible to skip the rest of the commands in the current loop and start from the top again. (the loop variable must still be incremented). Take a look at the example below:

```
#include<stdio.h>
int main()
{
int i;
i = 0;
while ( i < 20 )
{
i++;
continue;
printf("Nothing to see\n");
}
return 0;
}
```

In the example above, the printf function is never called because of the “continue”

Lesson 5

Array

"Array by definition is a variable that holds multiple elements which have the same data type."

Declaring Arrays

In order to declare an array, you need to specify:

- Data type of the array's elements. It could be int, float, double, char...etc.
- The name of the array.
- Fixed number of elements that array contains. The number of elements is put inside square brackets followed the array name.

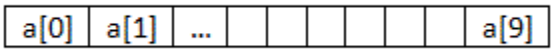
The following illustrates the typical syntax of declaring an array:

```
1 data_type array_name[size];
```

For example, to declare an array of integers with size of 10, you can do as follows:

```
1 int a[100];
```

In the memory, you have 10 consecutive objects like following:



Initializing Arrays

Similar to a variable, an array can be initialized. To initialize an array, you provide default values enclosed within curly braces in the declaration and assign that expression to the array.

Here is an example of initializing an array of integers.

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

Accessing Array's Elements

You can access array elements via indexes like *array_name[index]*.

Index of array starts from zero (0) not one (1) so the last element of an array is *array_name[size-1]* where size is the number of elements of the array.

```
int a[4];
```

```
a[0] = 1;
a[1] = 2;
a[2] = 3;
a[3] = 4;
```

Arrays and loops

One of the nice things about arrays is that you can use a loop to manipulate each element. When an array is declared, the values of each element are not set to zero automatically. In some cases you want to "initialize" the array (which means, setting every element to zero). This can be done like in the example above, but it is easier to use a loop. Here is an example:

```
#include<stdio.h>

int main()

{       int a[4];

        int i;

        for ( i = 0; i < 4; i++ )

                a[i] = 0;

        for ( i = 0; i < 4; i++ )

                printf("a[%d] = %d\n", i , a[i]);

        return 0;

}
```

Two Dimensional Array:

Two-dimensional arrays have rows and columns. Like

```
int a[2][3];
```

In two dimensional array first index represents rows and second index represents column.

Example:

```
#include <stdio.h>

int main()

{

    int m, n, c, d, first[10][10], second[10][10], sum[10][10];
```

```
    printf("Enter the number of rows and columns of matrix ");
    scanf("%d%d", &m, &n);
    printf("Enter the elements of first matrix\n");
    for ( c = 0 ; c < m ; c++ )
            for ( d = 0 ; d < n ; d++ )
                    scanf("%d", &first[c][d]);
  printf("Enter the elements of second matrix\n");
    for ( c = 0 ; c < m ; c++ )
            for ( d = 0 ; d < n ; d++ )
            scanf("%d", &second[c][d]);
    for ( c = 0 ; c < m ; c++ )
            for ( d = 0 ; d < n ; d++ )
               sum[c][d] = first[c][d] + second[c][d];
         printf("Sum of entered matrices:-\n");
    for ( c = 0 ; c < m ; c++ )          {
            for ( d = 0 ; d < n ; d++ )
            printf("%d\t", sum[c][d]);
            printf("\n");
    }
    return 0;
}
```

Lesson 6

Function

"Function is a sub program or module which is used to do a specific job. It is very important and useful concept of structured programming technique."

Most languages allow you to create functions of some sort. Functions are used to break up large programs into named sections. Functions are often used when the same piece of code has to run multiple times.

Features of Function:

- One function can do only one work at a time because each function is assigned individual work.
- Every function must have a unique name for its identification.
- Function can return a single value at a time or cannot.
- It can receive any number of arguments/values within it.
- Function is activated only when it is called from anywhere.

Syntax:

Return type function_name (list of arguments)

{

….

…

..

}

In this case you can put this piece of code in a function and give that function a name. When the piece of code is required you just have to call the function by its name. (So you only have to type the piece of code once).

In the example below we declare a function with the name MyPrint. The only thing that this function does is to print the sentence: Printing from a function. If we want to use the function we just have to call MyPrint() and the printf statement will be executed. (Don't forget to put the round brackets behind the function name when you call it or declare it).

```
#include<stdio.h>
void MyPrint()     //Declaration
{
        printf("Printing from a function.\n");
}
int main( )
{
        MyPrint();  //calling
        return 0;
}
```

Parameters and return

Functions can accept parameters and can return a result. (C functions can accept an unlimited number of parameters).

Where the functions are declared in your program does not matter, as long as a functions name is known to the compiler before it is called. In other words: when there are two functions, i.e. functions A and B, and B must call A, than A has to be declared in front of B.

Types of Functions

- No return type and no argument.
- Return type but no argument.
- No return type with argument.
- Return type with argument.

1. **No return type and no argument:**

```
#include<stdio.h>
#include<conio.h>
void sum()
{
```

```
        int a,b,c;
        printf("Please enter first number:");
        scanf("%d",&a);
        printf("Please enter second number:");
        scanf("%d",&b);
        c=a+b;
        printf("The sum is %d",c);
}
void main()
{
        sum();
        getch();
        clrscr();
        sum();
        getch();
}
```

2. Return type with no argument:

```
#include<stdio.h>
#include<conio.h>
int sum()
{
        int a,b;
```

```
        printf("Please enter first number:");
        scanf("%d",&a);
        printf("Please enter second number:");
        scanf("%d",&b);
        return a+b;
}
void main()
{
        int c;
      c=sum();
     printf("%d",c);
        getch();
}
```

3. No return type with argument:

```
 #include<stdio.h>
#include<conio.h>
void sum(int a, int b)
{
int c;
c=a+b;
printf("The sum is %d",c);
}
void main()
```

```
{
        sum(10,20);
        getch();
        clrscr();
        sum(30,40);
        getch();
}
```

4. With return type with argument

```
 #include<stdio.h>
#include<conio.h>
int sum(int a, int b)
{
return a+b;
}
void main()
{       sum(10,20);
        getch();
        clrscr();
        sum(30,40);
        getch();
}
```

Void

If you don't want to return a result from a function, you can use void return type instead of the int. So let's take a look at an example of a function that will not return an integer:

```
void our_site()
{       printf("www");
        printf(".NextDawn");
        printf(".nl");
}
```

Note: As you can see there is not an int before our_site() and there is not a return 0; in the function.

The function can be called by the following statement: our_site();

Global and local variables

A local variable is a variable that is declared inside a function. A global variable is a variable that is declared outside **all** functions. A local variable can only be used in the function where it is declared. A global variable can be used in all functions.

See the following example:

```
#include<stdio.h>
// Global variables
int A;
int B;
int Add()
{
        Return  A + B;
}
int main()
{
        int answer; // Local variable
        A = 5;
        B = 7;
        answer = Add();
        printf("%d\n",answer);
```

```
        return 0;
    }
```

As you can see two global variables are declared, A and B. These variables can be used in main() and Add().
The local variable answer can only be used in main().

Types of calling a function:

1) Call by value 2)Call by reference

Call by value: In call by value method, the called function creates a new set of variables and copies the values of arguments into them.
Example: Program showing Call by value method

```
void swap(int x, int y)
{
  int temp;
  temp = x;
  x = y;
  y = temp;
  printf("Swapped values are a = %d and b = %d", x, y);
}

void main()
{
  int a = 7, b = 4;
  swap(a, b);
  printf("Original values are a = %d and b = %d", a, b);
  printf("The values after swap are a = %d and b = %d", a, b);
}
```

Output:

Original Values are a = 7 and b = 4

Swapped values are a = 4 and b = 7

The values after swap are a = 7 and b = 4

This happens because when function swap() is invoked, the values of a and b gets copied on to x and y. The function actually swaps x and y while the original variables a and b remains intact.

Call by reference: In call by reference method, instead of passing a value to the function being called a reference/pointer to the original variable is passed.

Example: Program showing Call by reference method

```
void swap(int &x, int &y)
{
  int temp;
  temp = x;
  x =y;
  y = temp
  printf("Swapped values are a = %d and b = %d", x, y);
}
void main()
{
  int a = 7, b = 4;
  swap(&a, &b);
  printf("Original values are a = %d and b = %d",a,b);
  printf("The values after swap are a = %d and b = %d",a,b);
}
```

Output:

Original Values are a = 7 and b = 4

Swapped values are a = 4 and b = 7

The values after swap are a = 4 and b = 7
This happens because when function swap() is invoked, it creates a reference for the first incoming integer a in x and the second incoming integer b in y.

Lesson 7

Pointers

Pointers

"Pointer is a special type of variable, which is used to store an address or memory location of simple variable of the same type".

Pointers are used (in the C language) in three different ways:

- To create dynamic data structures.
- To pass and handle variable parameters passed to functions.
- To access information stored in arrays. (Especially if you work with links).

Pointer basics

To better understand pointers, it sometimes helps to compare a "normal variable" with a pointer.

When a "normal variable" is declared, memory is claimed for that variable. Let's say you declare an integer variable MYVAR. Four bytes of memory is set aside for that variable. The location in memory is known by the name MYVAR. At the machine level that location has a memory address.

A pointer differs in the way that a pointer is a variable that points to another variable. A pointer holds the memory address of that variable. That variable contains a value. Pointers are also called **address variables** because they contain the addresses of other variables.

Example:

```
#include<stdio.h>

int main()
{       int x;

        int *ptr_p;

        x = 5;

        ptr_p = &x;

         printf("%d",ptr_p); //displays address of x.

         printf("%d",x); // displays value of x.

         printf("%d",*ptr_p); // displays value of x.
```

```
        return 0;
}
```

Note: The integer x contains the value 5 on a specific address. The address of the integer x is copied in the pointer ptr_p. So ptr_p points to the address of x. In short: ptr_p = &x; means, "Assign to ptr_p the address of x."

To access the value of the integer that is being pointed to, you have to dereference the pointer. The * is used to dereference a pointer. Take a look at the following example:

```
#include<stdio.h>
int main()
{       int x,y;
        int *ptr_p;
        x = 5;
        ptr_p = &x;
        y = *ptr_p;
        printf("%d\n", y);
        return 0;       }
```

Note: The integer x has a value of five. The pointer ptr_p gets the address of integer x. The value pointed to is *ptr_p. (in this case five). So the integer y now contains the value of five.

Pointer and Array

When we declare an array variable and assigned it to pointer variable then, pointer variable will point to the base address of an array variable by default. The address of first block of an array is known as base address.

Ex. int x[5],*p;

p=x;

X

10	20	30	40	50
0	1	2	3	4
1000	1002	1004	1006	1008

Example:

```
#include <stdio.h>

int my_array[] = {1,23,17,4,-5,100};

int *ptr;

int main(void)

{

        int i;

        ptr = &my_array[0];      /* point our pointer to the first


    element of the array */

        printf("\n\n");

        for (i = 0; i < 6; i++)

        {

            printf("my_array[%d] = %d",i,my_array[i]);/*<-- A */

            printf("ptr + %d = %d\n",i, *(ptr + i)); /*<-- B */

        }

        return 0;

}
```

Pointer and Strings

A string is an array of character that is terminated by a NULL('\0'). For this we declare a character pointer which points first index of array.

```
#include<stdio.h>

Void main()

{

Char str[20], *ptr;
```

```
Printf(“Enter a string”);
Scanf(“%s”,&str);
For(ptr=str;*ptr!=’\0’;ptr++)
Printf(“%d”,*ptr);
}
```

Array of Pointer:

Ex.

```
#include<stdio.h>
void main()
{
char *ptr[3];
ptr[0]="ABC";
ptr[1]="welcome";
ptr[2]="programming";

printf("the content of ptr0=%s",ptr[0]);
printf("\n the content of ptr1=%s",ptr[1]);
printf("\n the content of ptr2=%s",ptr[2]);
}
```

Lesson 8

Structure & Union

"Structure is a collection of dissimilar data types."

Structure is user defined data type which is used to store heterogeneous data under unique name. Keyword 'struct' is used to declare structure.

The variables which are declared inside the structure are called as 'members of structure'.

Syntax:

```
struct structure_nm
{
        <data-type> element 1;
        <data-type> element 2;
        - - - - - - - - - - -
        - - - - - - - - - - -
        <data-type> element n;
}struct_var;
```


Example :

```
struct emp_info
{
        char emp_id[10];
        char nm[100];
        float sal;
```

```
}emp;
```

Note :

1. Structure is always terminated with semicolon (;).
2. Structure name as emp_info can be later used to declare structure variables of its type in a program.

ACEESSING STRUCTURE MEMBERS :

Structure members can be accessed using member operator '**.**' . It is also called as '**dot operator**' or '**period operator**'.

```
structure_var.member;
```

Program :

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

struct comp_info
{
        char nm[100];
        char addr[100];
}info;

void main()
{
        clrscr();
        printf("\n Enter Company Name : ");
        gets(info.nm);
        printf("\n Enter Address : ");
        gets(info.addr);
        printf("\n\n Company Name : %s",info.nm);
        printf("\n\n Address : %s",info.addr);
        getch();
}
```

Output :

Enter Company Name : TechnoExam, Technowell Web Solutions

Enter Address : Sangli, Maharashtra, INDIA

Company Name : TechnoExam, Technowell Web Solutions

Address : Sangli, Maharashtra, INDIA

ARRAY IN STRUCTURES :

Sometimes, it is necessary to use structure members with array.

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

struct result
{
int rno, mrks[5];
char nm;
}res;

void main()
{
        int i,total;
        clrscr();
        total = 0;
        printf("\n\t Enter Roll Number : ");
        scanf("%d",&res.rno);
        printf("\n\t Enter Marks of 3 Subjects : ");
        for(i=0;i<3;i++)
        {
                scanf("%d",&res.mrks[i]);
                total = total + res.mrks[i];
        }
        printf("\n\n\t Roll Number : %d",res.rno);
        printf("\n\n\t Marks are :");
        for(i=0;i<3;i++)
        {
                printf(" %d",res.mrks[i]);
```

```
        }
        printf("\n\n\t Total is : %d",total);
        getch();
}
```

Output :

Enter Roll Number : 1

Enter Marks of 3 Subjects : 63 66 68

Roll Number : 1

Marks are : 63 66 68

Total is : 197

STRUCTURE WITH ARRAY :

We can create structures with array for ease of operations in case of getting multiple same fields.

Program :

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

struct emp_info
{
        int emp_id;
        char nm[50];
}emp[2];

void main()
{
        int i;
        clrscr();
        for(i=0;i<2;i++)
        {
                printf("\n\n\t Enter Employee ID : ");
                scanf("%d",&emp[i].emp_id);
                printf("\n\n\t Employee Name : ");
                scanf("%s",emp[i].nm);
        }
        for(i=0;i<2;i++)
        {
```

```
        printf("\n\t Employee ID : %d",emp[i].emp_id);
        printf("\n\t Employee Name : %s",emp[i].nm);
    }
    getch();
}
```

Output :

```
    Enter Employee ID : 1

    Employee Name : ABC

    Enter Employee ID : 2

    Employee Name : XYZ

    Employee ID : 1
    Employee Name : ABC
    Employee ID : 2
    Employee Name : XYZ
```

STRUCTURES WITHIN STRUCTURES (NESTED STRUCTURES):

Structures can be used as structures within structures. It is also called as 'nesting of structures'.

Syntax:

```
struct structure_nm
{
        <data-type> element 1;
        <data-type> element 2;
        - - - - - - - - - - -
        - - - - - - - - - - -
        <data-type> element n;

        struct structure_nm
        {
                <data-type> element 1;
                <data-type> element 2;
                - - - - - - - - - - -
                - - - - - - - - - - -
                <data-type> element n;
        }inner_struct_var;
```

```
}outer_struct_var;
```

Example :

```
struct stud_Res
{
        int rno;
        char nm[50];
        char std[10];

        struct stud_subj
        {
        char subjnm[30];
        int marks;
        }subj;
}result;
```

In above example, the structure stud_Res consists of stud_subj which itself is a structure with two members. Structure stud_Res is called as 'outer structure' while stud_subj is called as 'inner structure.' The members which are inside the inner structure can be accessed as follow :

```
result.subj.subjnm
result.subj.marks
```

Program :

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

struct stud_Res
{
        int rno;
        char std[10];
        struct stud_Marks
        {
                char subj_nm[30];
                int subj_mark;
        }marks;
}result;

void main()
{
        clrscr();
        printf("\n\t Enter Roll Number : ");
```

```
        scanf("%d",&result.rno);
        printf("\n\t Enter Standard : ");
        scanf("%s",result.std);
        printf("\n\t Enter Subject Code : ");
        scanf("%s",result.marks.subj_nm);
        printf("\n\t Enter Marks : ");
        scanf("%d",&result.marks.subj_mark);
        printf("\n\n\t Roll Number : %d",result.rno);
        printf("\n\n\t Standard : %s",result.std);
        printf("\nSubject Code : %s",result.marks.subj_nm);
        printf("\n\n\t Marks : %d",result.marks.subj_mark);
        getch();
}
```

Output :

Enter Roll Number : 1

Enter Standard : MCA(Sci)-I

Enter Subject Code : SUB001

Enter Marks : 63

Roll Number : 1

Standard : MCA(Sci)-I

Subject Code : SUB001

Marks : 63

Union:

Union is user defined data type used to stored data under unique variable name at single memory location. Union is similar to that of stucture. Syntax of union is similar to stucture. But the major **difference between structure and union is 'storage.'** In structures, each member has its own storage location, whereas all the members of union use the same location. Union contains many members of different types, it can handle only one member at a time.

To declare union data type, **'union'** keyword is used.

Union holds value for one data type which requires larger storage among their members.

Syntax:

```
union union_name
{
        <data-type> element 1;
        <data-type> element 2;
        <data-type> element 3;
}union_variable;
```

Example:

```
union techno
{
        int comp_id;
        char nm;
        float sal;
}tch;
```

In above example, it declares tch variable of type union. The union contains three members as data type of int, char, float. We can use only one of them at a time.

MEMORY ALLOCATION :

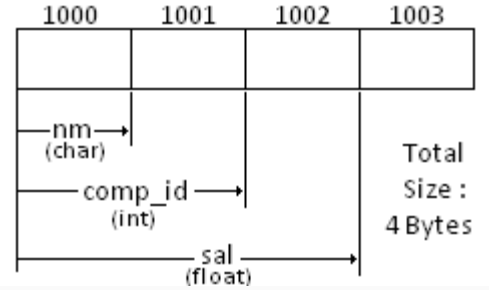


Fig : Memory allocation for union

To access union members, we can use the following syntax.

```
tch.comp_id
tch.nm
tch.sal
```

Program :

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

union techno
{
        int id;
```

```
        char nm[50];
}tch;

void main()
{
        clrscr();
        printf("\n\t Enter developer id : ");
        scanf("%d", &tch.id);
        printf("\n\n\t Enter developer name : ");
        scanf("%s", tch.nm);
        printf("\n\n Developer ID : %d", tch.id);//Garbage
        printf("\n\n Developed By : %s", tch.nm);
        getch();
}
```

Output :

Enter developer id : 101

Enter developer name : technowell

Developer ID : 25972

Developed By : technowell

Lesson 9

String

String :

A string is a collection of characters. Strings are always enlosed in double quotes as "string_constant".
Strings are used in string handling operations such as,

- Counting the length of a string.
- Comparing two strings.
- Copying one string to another.
- Converting lower case string to upper case.
- Converting upper case string to lower case.
- Joining two strings.
- Reversing string.

Declaration :
The string can be declared as follow :
Syntax:

```
char string_nm[size];
```

Example:

```
char name[50];
```


String Structure :

When compiler assigns string to character array then it automatically supplies**null character ('\0')** at the end of string. Thus, size of string = original length of string + 1.

```
char name[7];
name = "TECHNO"
```

'T'	'E'	'C'	'H'	'N'	'O'	'\o'
1	2	3	4	5	6	7

Read Strings :

To read a string, we can use scanf() function with format specifier %s.

```
char name[50];
scanf("%s",name);
```

The above format allows to accept only string which does not have any blank space, tab, new line, form feed, carriage return.

Write Strings :

To write a string, we can use printf() function with format specifier %s.

```
char name[50];
scanf("%s",name);
printf("%s",name);
```


string.h header file :

'string.h' is a header file which includes the declarations, functions, constants of string handling utilities. These string functions are widely used today by many programmers to deal with string operations.
Some of the standard member functions of string.h header files are,

Function Name	Description

strlen -	Returns the length of a string.
strlwr -	Returns upper case letter to lower case.
strupr -	Returns lower case letter to upper case.
strcat -	Concatenates two string.
strcmp -	Compares two strings.
strrev -	Returns length of a string.
strcpy -	Copies a string from source to destination.

Program :

```
/*  Program to demonstrate string.h header file working.

Creation Date : 06 Nov 2010 08:10:13 PM

Author : www.technoexam.com [Technowell, Sangli] */

#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
        char str[50];
        clrscr();
        printf("\n\t Enter your name : ");
        gets(str);
        printf("\nLower case of string: %s",strlwr(str));
        printf("\nUpper case of string: %s",strupr(str));
        printf("\nReverse of string: %s",strrev(str));
        printf("\nLength of String: %d",strlen(str));
        getch();
}
```

Output :

Enter your name : Technoexam

Lower case of string: technoexam

Upper case of string: TECHNOEXAM

Reverse of string: MAXEONHCET

Length of String: 10

Lesson 10

File Handling

File handling

The file I/O functions and types in the C language are straightforward and easy to understand. To make use of these functions and types you have to include the stdio library. (Like we already did in most of the tutorials).

The file I/O functions in the stdio library are:

- **fopen** – opens a text file.
- **fclose** – closes a text file.
- **feof** – detects end-of-file marker in a file.
- **fscanf** – reads formatted input from a file.
- **fprintf** – prints formatted output to a file.
- **fgets** – reads a string from a file.
- **fputs** – prints a string to a file.
- **fgetc** – reads a character from a file.
- **fputc** – prints a character to a file.

File I/O: opening a text file

The fopen library function is used to open a text file. You also have to use a specified mode when you open a file. The three most common modes used are read (r), write (w), and append (a). Take a look at an example:


```
#include<stdio.h>

int main()

{

        FILE *ptr_file;

        int x;

        ptr_file =fopen("output.txt", "w");
```

```
        if (!ptr_file)
                return 1;
        for (x=1; x<=10; x++)
                fprintf(ptr_file,"%d\n", x);
                 fclose(ptr_file);
                 return  0;
}
```

So let's take a look at the example:

ptr_file =fopen("output", "w");

The fopen statement opens a file "output.txt" in the write (w) mode. If the file does not exist it will be created. But you must be careful! If the file exists, it will be destroyed and a new file is created instead. The fopen command returns a pointer to the file, which is stored in the variable ptr_file. If the file cannot be opened (for some reason) the variable ptr_file will contain NULL.

if (!ptr_file)

The if statement after de fopen, will check if the fopen was successful. If the fopen was not successful, the program will return a one. (Indicating that something has gone wrong).

for (x=1; x<=10; x++)

This for loop will count to ten, starting from one.

fprintf(ptr_file,"%d\n", x);

The fprintf statement should look very familiar to you. It can be almost used in the same way as printf. The only new thing is that it uses the file pointer as its first parameter.

fclose(ptr_file);

The fclose statement will close the file. This command must be given, especially when you are writing files. So don't forget it. You have to be careful that you don't type "close" instead of "fclose", because the close function exists. But the close function does not close the files correctly. (If there are a lot of files open but not closed properly, the program will eventually run out of file handles and/or memory space and crash.)

File I/O: reading a text file

If you want to read a file you have to open it for reading in the read (r) mode. Then the fgets library functions can be used to read the contents of the file. (It is also possible to make use

of the library function fscanf. But you have to be sure that the file is perfectly formatted or fscanf will not handle it correctly). Let's take a look at an example:

```
#include<stdio.h>


int main()

{

        FILE *ptr_file;

        char buf[1000];


        ptr_file =fopen("input.txt","r");

        if (!ptr_file)

        return 1;


        while (fgets(buf,1000, ptr_file)!=NULL)

        printf("%s",buf);


        fclose(ptr_file);

        return 0;

}
```


Note: The printf statement does not have the new-line (\n) in the format string. This is not necessary because the library function fgets adds the \n to the end of each line it reads.

A file "input.txt" is opened for reading using the function fopen en the mode read (r). The library function fgets will read each line (with a maximum of 1000 characters per line.) If the end-of-file (EOF) is reached the fgets function will return a NULL value. Each line will be printed on stdout (normally your screen) until the EOF is reached. The file is then closed and the program will end.