The following are the keywords that one should know before we start the concept of any programming language in the world are:

Instruction
Module
Program
Software
System Software
Application Software
Language
Low level language
Middle level language
High level language
System Program
Application Program
Translator
Compiler
Interpreter
Assembler
Storage area or Memory
Main Memory
Secondary Memory
Cache Memory
Virtual Memory
Processor Dependent language
Processor Independent language
Platform Dependent language
Platform Independent language
Linker
Loader
Header Files

Instruction

These notes are dedicated to my father Mr. Mir Farooq Ali, Professor and Ex-Head of the Department of Mathematics (Retired), MJCET and my mom Mrs. Arifa, Vice Principal and Junior Lecturer in Botany (Retired) Govt. Falaknuma Junior College. If you have any doubts in this document, you can call me on 9010149047 or mail me at mir.ahmedali@mjcollege.ac.in, your queries will be solved within 24 hrs.

It is defined as a single statement that instructs the computer to perform some task. For example, let's take an example of C language instruction as

int x;

The above line instructs computer to create a location by name x and the size of location depends on the platform. It may be 2 bytes or 4 bytes.

Module:

It is defined as a collection of some instructions, when executed perform a task. Some languages like Structured languages refer as functions and object oriented languages refer as methods.

Program or Application:

It is defined as a collection of inter related modules, when executed perform a task.

System Program

It is defined as a program that is used by the System. The system can be either referred as operating system or any hardware component of a system. The examples are linker, compiler, assembler.

Application Program

It is defined as a program that is used by the user. The examples are factorial program, Fibonacci program and so on.

Software

It is defined as collection of inter related program, that when executed perform a task.

System Software

It is defined as a software that can be used by the system. The best example of system software is operating system.

Application Software

It is defined as a software that can be used by the user. One of the example is Microsoft Office.

Translator

It is defined as a program whose job is to convert code of one language to code of another language.

Compiler

It is defined as a translator whose job is to convert an intermediate code to assembly code at a time. The other jobs of Compiler are

1. Syntax Checking
2. Optimization

Interpreter

It is defined as a translator whose job is to convert an intermediate code to assembly code line by line.

Assembler

It is defined as a translator whose job is to convert an assembly code to object code.

Low level language

It is defined as a language in the form of 0s and 1s. Low level language is also known as binary language or machine language. It is the language that is understandable by the CPU.

Middle Level language

It is defined as a language that is in the form of Mnemonics. The best example of middle language is assembly language. It is the language that is understandable by the assembler.

High Level Language

It is defined as a language that is written using English language. It is the easiest language for the programmer to write programs. The best examples of high level language are C,C++,Java.

Storage area or Memory

It is defined as an area that is used to store data. The various storage areas that are used by the C Programs are:

1. Buffers
2. Cache Memory
3. RAM or Main Memory
4. Secondary Memory

Main Memory

It is defined as a memory that is used to store current data. In other words, whatever user is doing currently, that data is stored in Main Memory. Main Memory technically is known as RAM.

Secondary Memory

It is defined as a memory that is used to store very large amount of data. The nature of this memory is non-volatile that is if the system gets shutdown or restarted the data present in the secondary memory is not deleted. The various secondary memories available are:

1. Hard disk
2. CD
3. DVD
4. Pen Drive

Cache Memory

It is defined as a memory that is used to store frequently access data. The nature of this memory is volatile and it is the nearest memory to CPU.

Virtual Memory

It is defined as a memory that is virtual and is created with the space present in the drive of the hard disk where the copy of operating system is present. The usage of virtual memory comes with RAM.

Platform

It is defined as a combination of both operating system and the hardware architecture.

Header Files

It is defined as a file that consists of function prototypes. Some examples of header files are

stdio.h	input/output
stdlib.h	general utilities
string.h	string handling
time.h	date and time
math.h	mathematics
ctype.h	character handling

Libraries

It is defined as a file that contains the code of the predefined functions.

Object File

It is defined as a file that contains the object code of a c program which is made with the help of Hardware

architecture and operating system that makes this file as platform dependent.

Processor Dependent Language

It is defined as a language which works with only one processor. In other words, if a language which is compiled by a processor, gets executed with same processor only. The reason is every processor differs in Instruction Set Language. For example, if a language is written using intel processor(say), gets compiled by intel processor (say)only, another processor like spark processor(say) cannot compile it. The examples of processor dependent language are Machine language and Assembly language.

Note:
When there is no word of compilation, then there is no thought of execution.

Processor In-Dependent Language

It is defined as a language which can work with any processor. In other words, if a language which is written on any processor, can be compiled by any processor. For example, if iam writing a program on my machine that is having intel processor(say) can be compiled by any machine that is having another processor like spark(say).The examples of processor in dependent languages can be any High level languages like C,C++,Java.

Platform Dependent Language

It is defined as a language which is compiled on a platform but cannot be executed on other platform is known as platform dependent language. For example, if user writes a language, compiles it on a platform that is having Windows operating system (say)and Intel processor (say)cannot execute the program on Linux operating system(say) and intel processor(say) as the platform is changed. The examples of platform dependent languages are C,C++ etc.

Platform In-Dependent Language

It is defined as a language which is compiled on a platform can be executed on any platform is known as platform independent language. For example, if user writes a language, compiles it on a platform that is having Windows operating system (say)and Intel processor (say) can be executed on any platform that have any operating system and any processor. The examples of platform independent languages are Java etc....

Note:
In case of platform changes means it can be either change of operating system or change of any processor or any Hardware architecture.

Programming Methodology

The steps that has to be followed by a programmer when the programmer wants to solve a problem. The steps are shown below:

1 Analyze the problem

2 Identify the variables involved
3 Design the solution
4 Write the program
5 Enter it into a computer
6 Compile the program and correct errors
7 Correct the logical errors if any
8 Test the program with data
9 Document the program

Block Diagram of Computer:

A computer present in any part of world basically consist of the following components as shown below:

1. Input Unit
2. Output Unit
3. Processing Unit
4. Memory Unit

Input Unit:

It is defined as unit or device whose job is to accept the input from the user and forwards that to Memory unit for storage purpose. The various input devices used are keyboard,Mouse,Joystick etc.

Processing Unit:

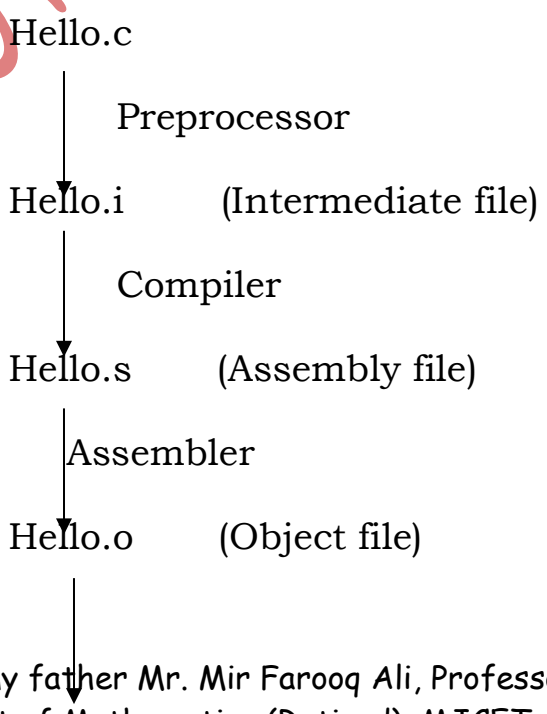
It is defined as a unit whose job is to perform some operations on that data coming from input unit. If the operations has to be performed on integer, then the unit is ALU and if the operations has to be performed on decimal values then the unit is FPU. These both are part of

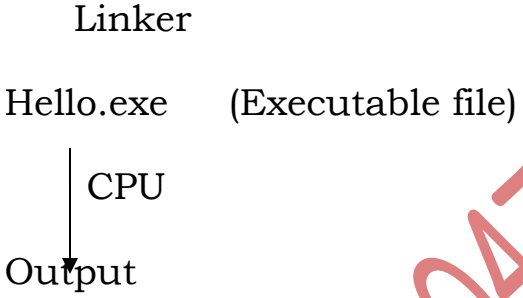
Processing unit which is also known as CPU. A CPU contains some storage area inside it, which is known as registers.

Memory Unit:
It is defined as a unit that is used to store data that is either coming from input device, or coming from processing unit or that have to be display on any Output unit. Basically, this unit is categorized into two , the first is primary and the second is secondary. Primary unit contains that data which is presently used by the system and whatever data that is to be saved for future purpose is stored in secondary unit. The basic example of Primary Unit is Main Memory, which is technically known as RAM. The basic example of secondary unit is Hard disk.

Stages of a C Program:

Let us consider an example of a C program by name Hello.c





The structure of a C program is as shown below:

```
header files
preprocessing statements
main()
{
Deceleration statements;
Initialization statements;
Output statements;
Input statements;
Processing statements;
Conditional statements;
Looping statements;
.
.
.
.
}
```


Lets us understand each and every statement in detail as discussed below:

Header files:

It is defined as a collection of function prototypes or function declerations. It is the responsibility of the Linker to connect the header file to the respective Libraries that have function code. This happens during the preprocessing stage. The information to the linker is given by the preprocessor.

Header files are of two categories the first one is default header file and the other is mandatory header file. The examples of the default header files are stdio.h, stdlib.h and the examples of mandatory header file is string.h, math.h and so on.

In case of default header file if the programmer does not mention the header file in the program and when a program gets compiled, a warning is given to the programmer, if the programmer ignores the warning the program gets compiled and gets executed. But in case of mandatory header file, if the programmer doesnot mention the header file in the program, and when the program gets compiled it gives error.

Preprocessing Statements:

It is defined as those statements that starts with #(hash) symbol. These lines are executed by a system program known as preprocessor. If programmer wants to use macro, then we have to go preprocessing statements. We will be studying preprocessing statements in UNIT II in detail.

main() is said to be the entry point for execution of the program.

Decleration statements:

It is defined as a statement that is used to create a variable in Main Memory. The size of the variable depends on the data type, but the creation of that location is done by the Compiler. Once a variable is created, then it is the compiler who executed an algorithm that generate a value to the newly created variable and that value is known as garbage value.

The syntax of decleration statement is as shown below:

datatype variablename; → for one variable

datatype variablename(s); → for many variables

for example:

If programmer is interested to create two variables to store integer values then the decleration statement is

int x,y;

where x and y are variables that are destined to store only integer values.

In general, if the programmer want to create variable, then they must know two things, the first is what is the type of value they want to store and how many variables they require by keeping in mind in one variable only one value can be stored.

Initialization statement:

If the programmer wants to assign some value to a variable, then one of the way is by using assignment operator. The

statement which is used to assign the value using assignment operator is known as initialization statement. The syntax of initialization is

variable = value

The variable in which the programmer is interested to store the value should be present on the LHS of the assignment operator and the value to be send is to be on the RHS of assignment operator.

It must be noted that the assigning of values that happen using assignment operator is done by the Compiler during compilation time.

Output Statement:

It is defined as a statement whose job is to display a message or value or both on the Monitor. The value can be either normal value or resultant value.

Below are the syntaxes for the message, value and both

Syntax for message:

printf("message");

For example, if programmer wants to display a message as Good Morning then by using printf() it can be done as follows

printf("Good Morning");

It must be noted that it is compulsory to write the message with in the double quotes as shown above.

Syntax for value:

printf("formatspecifier",variablename);

If programmer is interested to display the value of the variable on to the Monitor, then the programmer should specify the formatspecifier of the value and the name of the variable. For example, if the programmer want to display 5 (integer)value present in varaible x on the Monitor, then the format specifier is %d and the variable name is x as shown below:

printf("%d",x);

The above syntax work for all variables belonging to any data type.

Syntax for both message and value

printf("message formatspecifier",variablename);

If programmer wants to display both message and value, then the programmer should be aware of formatspecifier of the value and the variablename and it must be remembered that the position of message and formatspecifier with in printf() function depends on the requirement.

For example, if the programmer want to display as

Your balance is 30000, then the printf() is

printf("Your balance is %d",bal);

where bal is name of the integer variable, where 30000 is stored.

The other output statements are puts(), putch(), putchar() etc.

Input Statement:

It is defined as a statement whose job is to accept value during execution of the program. The commonly used input statement is scanf() and the syntax of scanf() is

Syntax of scanf():

scanf("format specifier",&variablename);

If the programmer want to capture the value during execution of the program, then the programmer should have an idea about what type of input value to be stored and the name of the variable in which it has to be stored.

& is known as ampersand , but good to be said as addressof operator. The job of addressof operator is to get the address of the variable, in which value have to be stored.

For example, if the programmer is interested to store integer value in a variable known as balance, then it is written as follows:

scanf("%d", &balance);

The number of formatspecifier and variablename depends on programmer requirement.

Processing statement:

It contains those statements that represent the logic of the program which is considered to be the most important part of the program. Logic in computer science is known as formula in Mathematics.

Conditional statements:

It contains those statements that are used to create conditions that make a code to be executed based on the result of the condition.

Looping statements:

It contains those statements that have to be executed as many number of times based on the condition.

Escape Sequence:

An escape sequence is used to express non printing character like a new line, tab etc. it begin with the backslash (\ ) followed by letter
like a, n, b, t, v, r, etc. the commonly used escape sequence are

\a : for alert
\n : new line
\0 : null
\b : backspace

\f : form feed
\? : question mark
\f : horizontal tab
\r : carriage return
\' : single quote
\v : vertical tab
\" : quotation mark

Conditional Statements:

It is defined as those statements that works based on the condition. In other words, they work on the output given by the condition whenever it is evaluated. As we know, that a condition gives two values either true or false. If the condition is true, a group of statements are executed and if the condition is false another group of statements are executed. The following conditional statements are:

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

if statement:

The syntax of if statement is

```
if(condition)
{
Statement1;
Statement2;
Statement3;
}
```

The above conditional statement is used if the requirement of the programmer is a single statement or some group of statements has to be executed whenever condition is true and nothing has to be executed whenever the condition is false.

It must be remembered by the programmer that the statements to be executed , whenever the condition is true must be kept after if.

And it must also be remembered that the control will go and execute those statement that are present after if whenever the condition is true and will just come out of the program whenever the condition is false.

For example, the below program explain the concept of if conditional statement as shown below:

```
void main()
{
char ch;
printf("Enter any character\n");
scanf("%d",&ch);
if(ch == 'a')
printf("The entered character is alphabet a\n");
}
```

In the above program we are concerned with statement to be executed when the condition is true but not concerned with when the condition is false.

If-else statement:

The syntax of if statement is

```
if(condition)
{
Statement1;
Statement2;
Statement3;
}
else
{
Statement4;
Statement5;
Statement6;
}
```

The above conditional statement is used if the requirement of the programmer is a single statement or some group of statements has to be executed whenever condition is true and a single statement or some group of statements has to be executed whenever condition is false.

It must be remembered by the programmer that the statements to be executed , whenever the condition is true should be kept after if and the statements should be executed whenever the condition is false must be kept after else.

And it must also be remembered that the control will go and execute those statement that are present after if whenever the condition is true and whenever the condition is false, the control will go and executed the statements that are present after else.

For example, the below program explain the concept of if-else conditional statement as shown below:

```
void main()
{
char ch;
printf("Enter any character\n");
scanf("%d",&ch);
if(ch == 'a')
printf("The entered character is alphabet a\n");
else
printf("The entered character is not alphabet a\n");

}
```

In the above program we are concerned with statement to be executed when the condition is true and a statement to be executed when the condition is false.

If-else-if statement:

The syntax of if statement is

```
if(condition)
{
Statement1;
Statement2;
Statement3;
}
else
if(condition)
{
Statement4;
Statement5;
Statement6;
}
```

```
else
if(condition)
{
Statement7;
Statement8;
Statement9;
}
else
{
Statement10;
Statement11;
Statement12;
}
```

The above conditional statement is used if the requirement of the programmer is to execute a single statement or some group of statements by checking many conditions. In other words, for the first time if the condition is true, some statements are executed, but if the condition is false, again it checks the condition and if it is true some statements are executed and if it is false again we check the condition. This process goes on, till the requirement of the programmer is met.

For example, the below program explain the concept of if-else-if conditional statement as shown below:

```
void main()
{
char ch;
printf("Enter any character\n");
scanf("%d",&ch);
if(ch == 'a')
printf("The entered character is alphabet a\n");
```

```
else
if(ch == 'b')
printf("The entered character is not alphabet b\n");
else
if(ch == 'c')
printf("The entered character is not alphabet c\n");
else
printf("The entered character is neither a,b,c\n");
}
```

The difference between if-else and if-else-if is in case of the if-else when the condition is false, some code gets executed. But in the case of latter, again we check the condition.

Switch statement:

This concept is used when the input of values are of specific range or limited values. This statement works with the help of the choice variable, that is used to compare the label of each case present in the switch block. The syntax of switch block is shown below:

```
switch(variable)
{

case label:          statement1;
                      statement2;
                      statement n;
                      break;


case label:          statement5;
                      statement6;
```

```
                    statement n;
                    break;

default :           statement7;
                    statement8;
                    statement n;

}
```

The variable present inside the switch block is known as choice variable. The label with in the switch can be either integer or character but can never be float. If anyone of the label doesn't match or if anyone of the case doesn't gets matched then the statements in default case gets executed. The reason we don't write break statement in the default case is it is the last one with in the switch block.

For example, let us understand the concept of switch using the below example:

A program to display what operation to be performed based on the choice given by the programmer using switch as shown below:

```
int main()
{
int ch,a,b;
printf("Enter any two values to perform some operation\n");
scanf("%d %d",&a,&b);
printf("Press 1 for addition\n");
```

```
printf("Press 2 for substraction\n");
printf("Press 3 for multiplication\n");
printf("Enter your choice\n");
scanf("%d",&ch);
switch(ch)
{
int c;
case 1:   c=a+b;
          printf("Addition of two numbers is %d",c);
          break;

case 2:   c=a-b;
          printf("Substraction of two numbers is %d",c);
          break;


case 3:   c=a*b;
          printf("Multiplication of two numbers is %d",c);
          break;

default:  printf("please enter option only 1 to 3\n");
}
return 98;
}
```

In the above program, the variable, programmer is using to compare the labels of the cases here is ch which comes under the category of choice variable.

It must be remembered that when the programmer want to use switch , the code to be present in any cases should always be on the right hand side of the case only.

Loops

It is defined as a concept that is used if the programmer wants to execute a statement or group of statements(which is also known as code) many number of times by just writing single time.

A loop consists of four parts

1. initialization
2. condition
3. body
4. increment/decrement

The explanation is given below:

Initialization contains those statements that are used to start a loop , that is initializing a loop. Just now, we have seen the definition of loop as a concept that makes a code to be executed many number of times. We can say many , when we initialize it with some number. For example, if i want the loop to be executed 20 times then the initialization part and condition part may be as shown below:

Initialization	Condition	Increment/Decrement
x=0	x<20	x= x + 1
a=2	a<=40	a = a + 2
d=5	d<=100	d = d + 5

here , in the above we can see that the above when used can make the loop to be executed 20 times.

Conditions are made using relational operators that make the loop to be executed , as expected by the

programmer till we get the output. The statement that will make the loop to execute for finite number of times is increment/decrement. If this statement is missing, then the loop will go into infinite mode.

The group of statements a programmer want to execute many number of times comes under the body of the loop.

The below program is used to display hello 20 times can be written in the following ways as shown below:

```
int main()
{
int i;
for(i=0;i<20;i=i+1)
printf("hello\n");
}
```

(OR)

```
int main()
{
int i;
for(i=2;i<=40;i=i+2)
printf("hello\n");
}
```

(OR)

```
int main()
{
int i;
for(i=5;i<=100;i=i+5)
printf("hello\n");
}
```

Write a program to find the reverse of a number

```
void main()
{
int rev=0,num,a;
printf("Enter the value to be reversed\n");
scanf("%d",&num);
while(num >0)
{
a = num %10;
rev = rev *10 +a;
num = num /10;
}
printf("The reverse of the number is %d\n",rev);
}
```

Write a program to find the whether a number is palindrome or not

```
void main()
{
int rev=0,num,a,p;
printf("Enter the value to be reversed\n");
scanf("%d",&num);
p=num;
while(num >0)
{
a = num %10;
rev = rev *10 +a;
num = num /10;
}
if(p == rev)
printf("Number is palindrome\n");
```

```
else
printf("Number is not palindrome\n");
}
```

Write a program to find whether the number is Armstrong or not

```
void main()
{
int arms=0,num,a,p;
printf("Enter any value \n");
scanf("%d",&num);
p=num;
while (num >0)
{
a = num %10;
arms = arms *10 +(a*a*a);
num = num /10;
}
if(p == arms)
printf("Number is armstrong\n");
else
printf("Number is not armstrong \n");
}
```

In the above examples , the statement

```
num = num /10
```

is acting as increment/decrement statement.

Write a program to display the following pattern as shown below:

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

void main()
{
printf("Enter number of lines\n");
scanf("%d",&n);
for(i=0;i<=n;i++)
{
printf("\n");
for(j=0;j<=i;j++)
printf("%d",j);
}
}
```

Datatypes:

It is a concept that helps a programmer to use different types of data in the program. In real time, we can notice that the different types of data used in a program are integers, decimal, charachters, strings and so on. If suppose if user want to make use of any value or numbers then it is mandatory to specify to compiler what is the datatype of that value.

If the value is integer, then datatype can be either int or signed integer or unsigned integer or long int. If the value is decimal , then the datatype can be either float or double. But if the value is an alphabet or a character then the datatype is char. If the value is a word or string, then the data type is character array. In general, it is mandatory for

the programmer to specify a datatype, whenever a value is used or stored for a program.

If a decimal value is small then we go for float datatype and if the decimal value is very big then we go for double datatype.

If the requirement of the program is to ask a single alphabet, then we go for char datatype. For example, if we are asking user to enter **m** if the gender is male and **f** if the gender is female, then the datatype used in char.

But, if the requirement is to ask the name, branch, city, address, fathername, mothername then the datatype char will not work as it is used to store single alphabet where as in the above requirement we require to store many alphabets and in case of address we have to store combination of digits and alphabets then we require character array. It must be remembered that in computer science, collection of alphabets is known as string.

Whenever compiler sees a statement that contains data type it allocates the memory for that datatype. The size of location depends on the platform. Some platform, gives the size of int as 2 bytes or 4 bytes. It must be remembered that size of any datatype varies from platform to platform.

It can be noted further that the size of float datatype is sometimes equal to int or double of int datatype and the size of double datatype is always double of float datatype. For example if the size of int is 2 bytes then the size of float on some platform can be 2 bytes or 4 bytes. But, the size of double is 8 bytes.

The below table gives us an idea of the size of the location (variable) in terms of bytes and the range it has.

SNo	Datatype	Format Specifier	Size in bytes	Range of the datatype
1	char	%c	1	0 to 127
2	int	%d	2	-32767 to 32768
3	Unsigned int	%u	2	0 to 65,535
4	float	%f	4	3.4e-38 to 3.4e+38 @ 6 decimal places
5	double	%lu	8	1.7e-308 to 1.7e+308 @ 15 decimal places
6	long double		10	3.4E-4932 to 1.1E+4932 19 decimal places
7	long int	%ld	4	-2,147,483,648 to 2,147,483,647

You can write a small program that explains you the various datatypes of C language.

This document contains the following topics:

Introduction to Function
Advantages

Types of Functions
Syntax of C Function
Function Prototype
Return statement
Types of function with and without parameters and return statement
Parameter Passing techniques
Recursion
Storage Classes
Arrays
Definition
Declerations
Initialization
Memory allocation
Types of arrays
One dimensional arrays
Two dimensional arrays
Linear Search
Binary Search
Selection sort
Bubble Sort
Pointers and addresses
Pointers and function arguments
Pointers and arrays
Address arithmetic
Command line arguments

Introduction to Function:

The following concepts should be known to anyone , who want to know the functions as follows:

Function

It is defined as group of statements when get executed perform a single task.

Types of Functions

Functions are categorized into two:

1. Predefined function
2. User defined function

Predefined function:

It is defined as a function whose body is written by the developers. These function are present in the libraries, and their function prototypes are present in the header files. It is the linker who connect the function prototypes to the libraries.

User defined function:

It is defined as a function whose body is written by the programmers. The best example of the user defined function is main function.

Function call:

It is defined as a transfer of control from one location to another location.

Calling function:

It is defined as a function that initiates a function call.

Called function:

It is defined as a function that gets initiated as a result of the function call.

Parameter

It is defined as a variable whose job is to send value from one function to other function. In other words, if calling function want to send value to called function, then it is possible by using parameters.

Actual Parameter

It is defined as a parameter that are present inside the calling function.

Formal Parameter

It is defined as a parameter that is present inside called function.

Return value

It is defined as a scenario, where a called function wants to return value to calling function, then it is possible with the help of return statement. By using return statement, we can either returned a value or a variable that contains a value.

Function prototype

It is defined as declaring a function with return datatype. It is also known as function decleration.

Return datatype

The value of the return datatype depends on the value present in the return statement or the datatype of the variable present in the return statement. But, if the function doesn't want to return a value that is if a function is not interested to return a value, then the return datatype becomes void.

Syntax of function:

```
returndatatype  functionname(parameter(s))
{
statements;
return variable/value;
}
```

Syntax of function call statement:

```
functioncall(parameter(s));
```

Or

```
functioncall();
```

Syntax of function prototype statement:

```
returndatatype functioncall(parameter(s));
```

Or

returndatatype functioncall();

Types of function with and without parameters and return statement

A function is categorized into 4 types depending on parameters and return statement. The types are:

1. Function with no parameter and no return value
2. Function with no parameter and return value
3. Function with parameter and no return value
4. Function with parameter and return value.

Function with no parameter and no return value:

```
#include<stdio.h>
int main()
{
void add();
add();
return 23;
}
void add()
{
printf("Enter any two values\n");
scanf("%d %d",&a,&b);
c=a+b;
printf("Addition is %d\n",c);
}
```

As we know, that the execution of the program always start from main() function. When controls enters main function, it finds a function call with no parameters, immediately it jumps to the function body, executes the statements present inside the function body and after executing all the statements of that called function, returns back to the calling function.

In the called function, the memory allocation is already done by the Compiler. The values entered by the user will be stored in the variables, added and the result is then stored in the variable c. Then the answer is displayed using printf function. Since, the called function doesn't want to return value to the calling function, that is why the return datatype is void.

Function with parameter and no return value:

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

```
int main()
{
void add(int x,int y);
int a,b;
printf("Enter any two values\n");
scanf("%d %d",&a,&b);
add(a,b);
return 23;
}
void add(int x,int y)
{
int z;
z = x + y;
printf("Addition is %d\n",z);
}
```

As we know, that the execution of the program always start from main() function. When controls enters main function, it will ask the user to enter the value and then pass that value via a function call. During a function call, the control gets transferred from calling function to called function and the value present in actual parameter is copied to formal parameter and once the value comes to called function, its added and the value will be displayed using printf function. Since, the called function doesn't want to return value to the calling function, that is why the return datatype is void.

Function with no parameter and return value:

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

```
int ans;
ans=add();
printf("The addition is %d\n",ans);
return 23;
}
int add()
{
int a,b,c;
printf("Enter any two values\n");
scanf("%d %d",&a,&b);
c = a + b;
return c;
}
```

As we know, that the execution of the program always start from main() function. When controls enters main function, it finds a function call with no parameters, immediately it jumps to the function body, executes the statements present inside the function body and after executing all the statements of that called function, returns back to the calling function by using return statement.

In the called function, the memory allocation is already done by the Compiler. The values entered by the user will be stored in the variables, added and the result is then stored in the variable c. later, this value present is returned to calling function by using return statement. The return datatype now is int as the return statement contains variable of type int.

It must be noted that the value present in the return statement will be returned from called function to exactly from where the control left the calling function. Now, we want to store that value, it is possible with the help of assignment operator and we store that value in the variable

which is present on the left hand side of assignment operator and later that value is displayed using printf() function.

Function with parameter and return value

```
#include<stdio.h>
int main()
{
int add(int x,int y);
int a,b,c;
printf("Enter any two values\n");
scanf("%d %d",&a,&b);
c=add(a,b);
printf("Addition is %d\n",c);
return 23;
}
int add(int x,int y)
{
int z;
z = x + y;
return z;
}
```

As we know, that the execution of the program always start from main() function. When controls enters main function, it will ask the user to enter the value and then pass that value via a function call. During a function call, the control gets transferred from calling function to called function and the value present in actual parameter is copied to formal parameter and once the value comes to called function, its added and stored in the variable z. later, this value is returned to calling function by using return statement. The

return datatype now is int as the return statement contains variable of type int.

It must be noted that the value present in the return statement will be returned from called function to exactly from where the control left the calling function. Now, we want to store that value, it is possible with the help of assignment operator and we store that value in the variable which is present on the left hand side of assignment operator and later that value is displayed using printf() function.

Recursion:

It is defined as a process where a function call itself. It must be remembered that all the programs cannot be converted to recursion, if and only if it contains a loop.

Program to explain fibonacci series using recursion

```
#include<stdio.h>
int main()
{
int fib(int a,int b,int i,int n);
int a=0,b=1,n,i=0;
printf("Enter the limit\n");
scanf("%d",&n);
fib(a,b,i,n);
return 33;
}
int fib(int a,int b,int i,int n)
{
        int c;
        c=a+b;
```

```
        printf("%d  ",c);
        a=b;
        b=c;
        i=i+1;
        if(i<n)
        fib(a,b,i,n);
}
```

Program to find factorial using recursion

```
#include<stdio.h>
int main()
{
        int fact(int n);
        int n,ans;
        printf("Enter the value to find factorial\n");
        scanf("%d",&n);
        ans =fact(n);
        printf("The factorial of the number %d is %d",n,ans);
        return 45;
}

int fact(int n)
{
        if((n == 0 ) ||(n == 1))
        return 1;
        else
        return n*fact(n-1);
}
```

Program to find gcd using recursion

```
#include<stdio.h>
int main()
{
int a,b,ans;
printf(Enter any two values to find gcd\n");
scanf("%d %d",&a,&b);
ans = gcd(a,b);
printf("The gcd of two numbers %d and %d is %d",a,b,ans);
return 33;
}
int gcd(int m,int n)
{
if(m = = n)
return m;
else
If(m>n)
return gcd(m-n,n);
else
if(m<n)
return gcd(m,n-m);
}
```

Scope and Life time of variable:

Life time of the variable means when the time duration at which a variable is created till it gets destroyed. Accessing the variable during its lifetime, is known as scope of the variable.

A variable can be accessed by either input function or output function or by using some operator that indirectly access the variable.

Storage Classes

Storage classes explain the different types of variables with its scope and life time. The different types of variables are

1. Local variable
2. Register variable
3. Static variable
4. Global variable

Local Variable

It is a variable that is used if the programmer requirement is it requires a variable which is local to either function or a block or a loop or a program. Once, the control comes out of a block the variable gets destroyed and the value is deleted.

The default value of local variable is garbage value. The life time of local variable is when the control enters the block and the life time of the variable is over once the control leaves the block. It is to be remembered that during its life time only, the variable can be accessed. If you access the variable, outside its lifetime then an error gets generated because the variable does not exist.

It must be noted that during any arithmetic operation, the values present in the memory is copied to registers, then given to ALU which will store the result into another register, and later transferred to a memory location when it has to be displayed on the Monitor.

Register Variable:

It is of type local variable but it leads to faster execution of the program. The syntax of using a register variable is just to use register keyword preceding the local variable syntax.

By using register variable, the value is directly stored in the register, if any arithmetic operation is performed, values are accessed from the register, after operation is performed it is directly displayed on the monitor.

Static Variable:

This variable is used if the programmer requirement is that the life time of the variable should be throught the program and scope of the variable should be with in the function itself. It means that the variable should be dedicated to a single function but it should not be destroyed once control comes out of the function. In other words, once control comes out of the function, the value will be again used when if it enters the function once again whenever it is required.

The default value of static variable is 0. The scope of static variable is within the function and the life time of the static variable is within the program.

Global Variable:

This variable is used if the programmer requirement is that the variable should be accessed from any function or a variable whose value to be shared by all the functions that is any function can access and can change the value.

The life time of global variable is throught the program and the scope of the global variable is throught the program. The default value is zero. Generally, we declare a global variable before the main function.

The summary of all the variables is tabulated as shown below:

SNo	Type of Variable	Scope variable	Lifetime of variable	Default Value
1	Local Variable	Within the function	Within the function	Garbage Value
2	Register Variable	Within the function	Within the function	Garbage Value
3	Static Variable	Within the function	Within the program	Zero
4	Global Variable	Within the program	Within the program	Zero

Array

It comes under the category of the advanced datatype or derived datatype because it is made from the existing datatype. For example, if I just say array it has no meaning but if I say that integer array means that there is meaning.

Syntax of 1-D array:

datatype arrayname[size];

the above syntax contains a datatype that explains what type of data to be stored, what is the name of the array and the number of locations of that array. Number of locations we generally represents with help of size.

Eg: int arr[50];

Here the memory is allocated during compilation time in a continuous way and the name of that memory is arr and this memory can store only integer values.

Note:
An array can store any type of data but it must of same datatype and the size of the array can be anything, generally depends on the programmer requirement.

If programmer requirement is to store the name, college name, branch name then it is mandatory to go for character array by giving some size.

Write a program to accept and display the value of an array

```
#include<stdio.h>
int main()
{
int arr[30],i,n;
printf("Enter the size of an array\n");
scanf("%d",&n);
printf("Enter values of an array\n");
for(i=0;i<n;i++)
scanf("%d",&arr[i]);
printf("The value of array are\n");
for(i=0;i<n;i++)
printf("%d\n",arr[i]);
return 33;
}
```

Write a program to accept values and find out sum of all the values of an array

```
#include<stdio.h>
int main()
{
int arr[30],i,n,sum=0;
printf("Enter the size of an array\n");
scanf("%d",&n);
printf("Enter values of an array\n");
for(i=0;i<n;i++)
scanf("%d",&arr[i]);
printf("The sum of all values of an array are\n");
for(i=0;i<n;i++)
sum = sum + arr[i];
printf("%d\n",sum);
return 33;
}
```

Write a program to find the maximum and minimum value of the array

```
#include<stdio.h>
int main()
{
int arr[30],i,n,sum=0,max,min;
printf("Enter the size of an array\n");
scanf("%d",&n);
printf("Enter values of an array\n");
for(i=0;i<n;i++)
scanf("%d",&arr[i]);
max=arr[0];
min=arr[0];
for(i=1;i<n;i++)
{
if(arr[i]>max)
max = arr[i];
if(arr[i]<min)
min = arr[i];
}
printf("The maximum value is %d\n",max);
printf("The maximum value is %d\n",min);
return 33;
}
```

Searching Techniques

It is defined as a technique whose job is to search for a required value among group of values in an array. The value to be searched is accepted in a variable, whose name we are giving as key. After accepting the value, we search the value

present in the key with all the values of the array. Once the value is found we instantly stop searching by just displaying a message as value found and come out of the program by executing exit() function which is a predefined function present in a header file known as stdlib.h . But, after searching all the values of the array, if we did not find any value, we display a message value not found, which is a printf() statement that is present at the end of the loop.

The two basic techniques for searching are:

1. Linear Search
2. Binary Search

It must be noted that Linear search is a very basic and time consuming technique where the value present in the key is compared with all the values of an array. If there is a match at initial level it is said to be best case and if there is a match at the last then it is said to be worst case. Due to this the performance of the application gets degraded.

To improve the performance a new technique by name binary search is introduced. In binary search, the value of key is compared with the middle value of the array. If it matches, we display a message value found and make the control to come out of the program by using exit() function. But if the value is not matched then we see that the value present in the key is either less than the middle value or greater than the middle value. If it is less than the middle value then the searching goes on the left hand side and the right hand side is ignored. But if the value is more than the middle value, then searching goes on the right hand side leaving the left hand side and this procedure repeats till there is a match.Once the loop completes and we didn't find the value we just display a message that value is not found.

Now let us understand the concept of linear search and binary search with the help of programs as shown below:

Program to find linear search

```
#include<stdio.h>
int main()
{
int arr[50],n,i,key;
printf("Enter number of values to be stored\n");
scanf("%d",&n);
printf("Enter the values\n");
for(i=0;i<n;i++)
scanf("%d",&arr[i]);
printf("Enter the value to be searched\n");
scanf("%d",&key);
for(i=0;i<n;i++)
{
if(key == arr[i])
{
printf("Element found at %d index on %d location",i,i+1);
exit(0);
}
}
printf("Element not found\n");
return 44;
}
```

Program to compute binary search

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

```
int arr[50],n,i,key,low,high;
printf("Enter number of values to be stored\n");
scanf("%d",&n);
printf("Enter the values\n");
for(i=0;i<n;i++)
scanf("%d",&arr[i]);
printf("Enter the value to be searched\n");
scanf("%d",&key);
low=0;
high=n-1;
while(low<=high)
{
mid = (low+high)/2;
if(key == arr[mid])
{
printf("Element found at %d index on %d location",i,i+1);
exit(0);
}
if(key < arr[mid])
high = mid – 1;
else
low = mid + 1;
}
printf("Element not found\n");
return 33;
}
```

Sorting techniques:

If want to arrange the values either in ascending order or descending order, then we can use any one of the sorting technique. It must be remembered that the default sorting is ascending order. The various sorting techniques are

1. Selection Sort
2. Bubble Sort
3. Quick Sort
4. Merge Sort
5. Heap Sort

Selection Sort

It is a technique, where we select the first element which we say as pivot element. Then we compare the value of pivot with the remaining elements of the array, if the value present in the array is less than the value at pivot element, then we interchange the values. This procedure is followed till all the values of the array get sorted. The below program explains the concept of selection sort as follows:


```
#include<stdio.h>
int main()
{
int arr[50],n,i;
printf("Enter the size of the array\n");
scanf("%d",&n);
printf("Enter the values\n");
for(i=0;i<n;i++)
scanf("%d",&arr[i]);
for(i=0;i<=n-2;i++)
{
   for(j=i+1; j<=n-1;j++)
       {
       if(a[i] > a[j])
```

```
        {
        int temp;
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
        }
        }
}
printf("The sorted array is follows\n");
for(i=0;i<n;i++)
printf("%d\n",arr[i]);
return 33;
}
```

Bubble Sort:

In this sorting technique, the first value of the array is compared with the second value of the array and that goes on. In other words, comparision in bubble sort is with neighboring elements of the array. This is done till all the values comes in sorted order. The below program explains the concept of bubble sort as shown below:

```
#include<stdio.h>
int main()
{
int arr[50],n,i;
printf("Enter the size of the array\n");
scanf("%d",&n);
printf("Enter the values\n");
```

```
for(i=0;i<n;i++)
scanf("%d",&arr[i]);
for(i=0;i<=n-2;i++)
{
    for(j=0; j<=n-1-i;j++)
        {
        if(a[j] > a[j+1])
        {
        int temp;
        temp = a[j];
        a[j] = a[j+1];
        a[j+1] = temp;
        }
        }
}
printf("The sorted array is follows\n");
for(i=0;i<n;i++)
printf("%d\n",arr[i]);
return 33;
}
```

Two dimensional array:

If the programmer requirement is dividing an array further, then it is said to be two dimensional array.

Syntax:

datatype arrayname[horizontal][vertical];

for example,

int a[2][2]: means it has two rows and two columns, and totally it can store around 4 values.

Write a program to add two matrices

```
#include<stdio.h>
int main()
{
  int a[2][2],b[2][2],c[2][2],i,j;
```

```
 printf("Enter the First matrix\n");
 for(i=0;i<2;i++)
 for(j=0;j<2;j++)
 scanf("%d",&a[i][j]);
 printf("\nEnter the Second matrix\n");
 for(i=0;i<2;i++)
 for(j=0;j<2;j++)
 scanf("%d",&b[i][j]);
 for(i=0;i<2;i++)
 for(j=0;j<2;j++)
 c[i][j]=a[i][j]+b[i][j];
  printf("\nThe Addition of two matrix is\n");
  for(i=0;i<2;i++){
  printf("\n");
  for(j=0;j<2;j++)
  printf("%d\t",c[i][j]);
  }
  return 0;
}
```

Write a program to substract two matrices

```
#include<stdio.h>
int main()
{
  int a[2][2],b[2][2],c[2][2],i,j;
  printf("Enter the First matrix\n");
  for(i=0;i<2;i++)
  for(j=0;j<2;j++)
  scanf("%d",&a[i][j]);
  printf("\nEnter the Second matrix\n");
  for(i=0;i<2;i++)
  for(j=0;j<2;j++)
```

```
 scanf("%d",&b[i][j]);
 for(i=0;i<2;i++)
 for(j=0;j<2;j++)
 c[i][j]=a[i][j]-b[i][j];
  printf("\nThe Substraction of two matrix is\n");
  for(i=0;i<2;i++){
  printf("\n");
  for(j=0;j<2;j++)
  printf("%d\t",c[i][j]);
  }
  return 0;
}
```

Write a program to multiply two matrices

```
#include<stdio.h>
int main(){
  int a[5][5],b[5][5],c[5][5],i,j,k,sum=0,m,n,o,p;
  printf("\nEnter the row and column of first matrix");
  scanf("%d %d",&m,&n);
  printf("\nEnter the row and column of second matrix");
  scanf("%d %d",&o,&p);
  if(n!=o){
      printf("Matrix mutiplication is not possible");
      printf("\nColumn of first matrix must be same as row of
second matrix");
  }
  else{
      printf("\nEnter the First matrix\n");
      for(i=0;i<m;i++)
      for(j=0;j<n;j++)
           scanf("%d",&a[i][j]);
      printf("\nEnter the Second matrix\n");
```

```
    for(i=0;i<o;i++)
    for(j=0;j<p;j++)
        scanf("%d",&b[i][j]);

    for(i=0;i<m;i++){ //row of first matrix
    for(j=0;j<p;j++){  //column of second matrix
        c[i][j] =0;
        for(k=0;k<n;k++)
            c[i][j] = c[i][j ]+a[i][k]*b[k][j];

    }
    }

  printf("\nThe multiplication of two matrix is\n");
  for(i=0;i<m;i++){
      printf("\n");
      for(j=0;j<p;j++){
          printf("%d\t",c[i][j]);
      }
  }
  return 0;
}
```

Write a program to find transpose of a matrix

```
#include<stdio.h>
int main()
{
  int a[2][2],b[2][2],c[2][2],i,j;
  printf("Enter the First matrix\n");
  for(i=0;i<2;i++)
  for(j=0;j<2;j++)
  scanf("%d",&a[i][j]);
```

```
 printf("\nEnter the Second matrix\n");
 for(i=0;i<2;i++)
 for(j=0;j<2;j++)
 b[i][j] = a[j][i];
 printf("\nThe Transpose of the matrix is\n");
 for(i=0;i<2;i++){
 printf("\n");
 for(j=0;j<2;j++)
 printf("%d\t",b[i][j]);
 }
 return 0;
}
```

Write a program to find whether the matrix is symmetric or not

Write a program to find transpose of a matrix

```
#include<stdio.h>
int main()
{
  int a[2][2],b[2][2],c[2][2],i,j;
  printf("Enter the First matrix\n");
  for(i=0;i<2;i++)
  for(j=0;j<2;j++)
  scanf("%d",&a[i][j]);
  printf("\nEnter the Second matrix\n");
  for(i=0;i<2;i++)
  for(j=0;j<2;j++)
  b[i][j] = a[j][i];
   for(i=0;i<2;i++)
   for(j=0;j<2;j++){
   if(a[i][j] !=b[i][j])
```

```
  {
   printf("Matrix is not symmetric\n");
   exit(0);
  }
  }
 printf("Matrix is symmetric\n");
 return 0;
}
```

Command line arguments:

It is defined as a concept where arguments are passed to the main program during execution of the program. This values are passed directly to main function.

The syntax of main function that accepts argument is:

int main(int argc, char *argv[]);

argc: count number of arguments are there in the buffer

argv[]: array of type pointer that is pointing string

Example:

Write a program to add two numbers through command line arguments

```
#include<stdio.h>
int main(int argc,char *argv[])
{
int i=0,sum,x,y;
```

```
if((arg<3) || (argc>3))
{
printf("Invalid usage\n");
exit(1);
}
x = atoi(argv[1]);
y = atoi(argv[2]);

sum = x + y;

printf("The sum is %d\n",sum);
}
```

Note:

atoi is a function that converts a string to integer

This document contains the following topics:

Introduction to Structures
Syntax of defining the structure
Array of Structures
Array with in a structure
Structures with in a structure
Structures and functions
Pointers to structures
Self referential structures
Unions
File basics
File Handling functions
File Random access functions

Introduction to Structures:

It is defined as a advanced datatype or a derived datatype whose job is to store different types of data as a single entity.

Syntax of a structure

```
struct StructureName
{
datatype membervariable1;
datatype membervariable2;
datatype membervariable3;
};
```

Every structure decleration starts with keyword struct followed by structurename . The body of the structure is

ended with semicolon. The variables present inside a structure are known as member variables. For the above structure decleration no memory is allocated.

If we want some memory to be allocated to the variables, then it is possible by creating structure object. The size of each structure object is sum of all member variables present in that object. It must be noted that the memory allocated to member variables of a structure object is continuous that to depend on the order in which members are present in a structure. The syntax to create a structure object is can be any one of the following:

```
struct StructureName
{
datatype membervariable1;
datatype membervariable2;
datatype membervariable3;
} structureobject;
```

Or

```
struct structurename structureobject;
```

or

```
structurename structureobject;
```

For example, lets create a structure for Rectangle as shown below:

```
main()
{
structure Rectangle
{
int l;
int b;
}r;
printf("Enter the length value\n");
scanf("%d",&r.l);
printf("Enter the breadth value\n");
scanf("%d",&r.b);
area = r.l * r.b;
printf("The area is %d\n", area);
}
```

In the above example, Rectangle is the name of the structure whose object name is r. l and b are known as members variables of the structure.

Array with in a structure:

Write a program to enter details of a student using structure.

```
struct student
{
int rollno;
char name[50];
char gender;
};

main()
{
student s1;
```

```
printf("Enter the rollno of a student\n");
scanf("%d",&s1.rollno);
printf("Enter the name of a student\n");
fflush(stdin);
gets(s1.name);
printf("Enter the gender of a student \n");
fflush(stdin);
scanf("%c",&s1.gender);
printf("The details of student are\n");
printf("The name is");
puts(s1.name);
printf("\n The rollno is %d\n",s1.rollno);
printf("The gender is %c\n",s1.gender);
}
```

Note:

It must be remembered that whenever you ask user to enter a character or word, it is preferable to use fflush(stdin) as it solves the problem of Buffer Management.

Array of Structures:

If we have to create many structure objects or many objects belonging to same structure, then instead of creating many objects we go for array of structure objects.

```
struct student
{
int rollno;
char name[50];
char gender;
};
```

```
main()
{
student s[50];
printf("Enter number of students info\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter the rollno of a student\n");
scanf("%d",&s[i].rollno);
printf("Enter the name of a student\n");
fflush(stdin);
gets(s[i].name);
printf("Enter the gender of a student \n");
fflush(stdin);
scanf("%c",&s[i].gender);
}
printf("The details of %d students are\n",n);
for(i=0;i<n;i++)
{
printf("The rollno of student %d is %d",i,s[i].rollno);
printf("The name of student %d is %s",i,s[i].name);
printf("The gender of student %d is %c",i,s[i].gender);
}
}
```

In the above, we are asking the user to enter number of students he is interested for and that value will be stored in a variable by name n.

Structures with in a structure:

If we write structure with in a structure, then it is known as Nested structure.

If in a requirement, if we have to break a member of a structure into parts then it is possible by using nested structure. For example , if address is a member of a structure, and if we want to break into parts like HNo, area, streetno then we go for a nested structure as shown below:

```
struct student
{
int rollno;
char name[80];
struct address
{
int sno;
char hno[50];
char locality[30];
};
};

void main()
{
student s[50];
address a[50];
printf("Enter number of students info\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter the rollno of a student\n");
scanf("%d",&s[i].rollno);
printf("Enter the name of a student\n");
fflush(stdin);
gets(s[i].name);
printf("Enter the housenumber of a student \n");
```

```
fflush(stdin);
scanf("%s",&s[i].a[i].hno);
printf("Enter the locality of a student \n");
fflush(stdin);
gets(s[i].a[i].locality);
printf("Enter the streetno of a student \n");
fflush(stdin);
scanf("%d",&s[i].a[i].sno);
}
printf("The details of %d students are\n",n);
for(i=0;i<n;i++)
{
printf("The rollno of student %d is %d",i,s[i].rollno);
printf("The name of student %d is %s",i,s[i].name);
printf("The address of student %d is %s",i,s[i].a[i].hno);
printf("The address of student %d is %s",i,s[i].a[i].hno);
printf("The address of student %d is %s",i,s[i].a[i].hno);
}
}
```

Pointer to Structures

This is indeed pointer to structure object. If we want to access a member variable then it is possible with the help of a structure object. But, if we want to access a member variable, without any structure object then it is possible with the help of a pointer. It must be noted that the pointer variable should be of same datatype to which object belongs whose address has to be stored.

```
/* Program for pointer to one structure object*/
struct student
{
```

```
int rollno;
char name[50];
char gender;
};

main()
{
student s1,*p;
p = &s1;
printf("Enter the rollno of a student\n");
scanf("%d",&s1.rollno);
printf("Enter the name of a student\n");
fflush(stdin);
gets(s1.name);
printf("Enter the gender of a student \n");
fflush(stdin);
scanf("%c",&s1.gender);
printf("The details of student are\n");
printf("The name is");
puts(p->name);
printf("\n The rollno is %d\n",p->rollno);
printf("The gender is %c\n",p->gender);
}
```

Self referential structure:

It is defined as a structure whose one of its member, is a pointer variable of its type. This pointer will store the address of another object. In other words the pointer variable which will store the address of another structure object will be a member of a preceding structure object, in this way a chain is formed between the structure object. If the programmer want to stop this chain, then the programmer can assign the

value NULL in the pointer variable of the last structure object.

```
struct student
{
int rollno;
char name[50];
char gender;
struct student *next;
};

main()
{
student s[50],*head;
head = &s[0];
printf("Enter number of students info\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter the rollno of a student\n");
scanf("%d",&s[i].rollno);
printf("Enter the name of a student\n");
fflush(stdin);
gets(s[i].name);
printf("Enter the gender of a student \n");
fflush(stdin);
scanf("%c",&s[i].gender);
}
printf("The details of %d students are\n",n);
for(i=0;i<n;i++)
{
s[i].next = &s[i+1];
printf("The rollno of student %d is %d",i,s[i].next->rollno);
```

```
printf("The name of student %d is %s",i,s[i] .next->.name);
printf("The gender of student %d is %c",i,s[i].next->gender);
}
s[i+1].next = NULL;
}
```

Union:

It is defined as an advanced datatype or a derived datatype whose job is to store different types of data as a single entity but only one data at a time because the memory allocated to union object depends on the member variable who occupy the maximum size.

The below program illustrates the concept of union as shown below:

```
union Check
{
char ch;
int i;
};

void main()
{
union Check c1;
printf("\n %d %d",c1.ch,c1.i);
c1.i = 511;
printf("\n %d %d",c1.ch,c1.i);
c1.ch=10;
printf("\n %d %d",c1.ch,c1.i);
}
```

In union, memory is allocated to one member and all these members will share the same memory.

File Handling in C

We frequently use files for storing information which can be processed by our programs. In order to store information permanently and retrieve it, we use files.

In order to use files we have to learn about ***File I/O*** i.e. how to write information to a file and how to read information from a file. If you wish to use a file in your programs, then you must specify which file or files you wish to use, which can be kept as a parameter in fopen() function.

When you open a file you must also specify what you wish to do with it i.e. **Read** from the file, **Write** to the file, or both.

As a programmer if you want your program to interact with the file, the mediator used is the file pointer that makes our program either to read data from the file or write data into the file. It must be remembered that for each file to be accessed we require a file pointer. For example, if my program wants to access two files, then I require two file pointers.

The syntax for creating a file pointer variable is:

```
FILE  *fp1, *fp2, *fp3;
```

The variables fp1, fp2, fp3 are file pointers, and the name of the variable can be of choice.

The header file, that contain all the information related to files that is predefined functions, macros is <stdio.h> . For example, EOF and NULL are defined in <stdio.h>.

Modes in a file:

1. r mode:
   If the programmer wants to create a file in read mode by using fopen function, then that file should be existing in the system. If the file is not existing , then operating system will not create the file.

2. w mode:
   If the programmer wants to create a file in write mode by using fopen function, then that file should be existing in the system, then the operating system will delete that file and creates a new empty file with same name which is mentioned in fopen function. If the file is existing or not existing, then operating system will create a new file with same name which is mentioned in fopen function.

3. a mode
   If the programmer wants to create a file in append mode by using fopen function, then that file should be existing in the system, then operating system will open that file and writes the data at the end of the existing file. But, if the file is not existing , then operating system will create a new file by same name which is mentioned in the fopen function..

Functions that are used in File I/O

The function **fopen** is one of the Standard Library functions whose job is to return the starting address of the file. For example

```
fp = fopen( “prog.c”,  “r”) ;
```

The above statement returns the address of the file and assigns to file pointer variable fp.When we wish to access this file for I/O, we use the file pointer variable fp to refer to it.

The function **fgetc** is one of the Standard Library functions whose job is to return character from the file, to which file pointer is pointing. Since, it returns a character, the return datatype of this function is char

```
ch = fgetc(fp);
```

The function **fputc** is one of the Standard Library function whose job is to transfer the character from a character variable to a location to which a file pointer is pointing

```
fputc(ch,fp);
```

The function fclose() is one of the Standard Library function whose job is to disconnect the connection between the file pointer variable and the file. If we want to close a single connection we just use fclose(filepointervariable) and if we want to close many connections then we use fcloseall().

Example 1: Write a program to display contents of a file on screen

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

void main()
{
```

```
        FILE *fp;
        char ch ;
        fp = fopen( “hello.c”, “r” );
        if(fp == NULL)
        {
        printf(“File not created\n”);
        exit(1);
        }
        while(12)
        {
        ch = fgetc( fp ) ;
        if(ch == EOF)
        break;
        printf(“%c”,c);
        }
        fclose( fp );
}
```

In the above program, we are creating a file pointer variable , whose default value is NULL. The input file is hello.c. Here the user want to read the contents from the file and to display on the Monitor. It must be remembered that whenever, you want to copy the data from a file, then that file has to be in read mode, which is represented by “r”.

If the file is not existing, then fopen function returns nothing, and the default value of file pointer variable, is not changed i.e. it will remain as NULL. When the file is not existing in the system, then there is no need to make the control to be in the program, we can just make the control to forcefully come out of the program by using exit() function, which present in the header file known as stdlib.h

We then read a character from the file by using fgetc() function. If the file is empty, we are at the end, so fgetc()

returns EOF a special value to indicate that the end of file has been reached. (Normally -1 is used for EOF).

This reading should happen for infinite times as we are not aware of how many characters are there in the file, that is why iam using an infinite loop.

But, if the file is non-empty, then the while loop simply keeps reading characters from the file and displaying them immediately on the Monitor, until the end of the file is reached. Once the end of file is reached, we make the control come out of the loop by using break statement.

Once the entire data is copied to the Monitor, we break the connection between the file and filepointer by using the function **fclose(fp) , where fp is file pointer variable.**

Example 2: Write a program to prompt user for filename and display file on screen

OR

Write a program to display the contents of file on screen

```
#include <stdio.h>
#include<stdlib.h>
void main()
{
        FILE *fp;
        char c ;
        char filename[40] ;

        printf("Enter file to be displayed: ");
        gets( filename ) ;

        fp = fopen( filename, "r" );
        if(fp == NULL)
        {
        printf("File not created\n");
        exit(1);
        }
        while(12)
        {
        c = fgetc( fp ) ;
        if(c == EOF)
        break;
        printf("%c",c);
        }
        fclose( fp );
}
```

Example 3: Write a program to count the number of lines, words and characters in a file.

```
#include <stdio.h>
#include<stdlib.h>
void main()
{
        FILE, *fp;
        char c ;
        char filename[40] ;
        int nl=1,ns=0,nc=0;

        printf("Enter file to be displayed: ");
        gets( filename ) ;

        fp = fopen( filename, "r" );
        if(fp == NULL)
        {
        printf("File not created\n");
        exit(1);
        }
        while(12)
        {
        c = fgetc( fp ) ;
        if(c == EOF)
        break;
        nc = nc + 1;
        if(c == ' ')
        ns = ns +1;
        if(c =='\n')
        nl = nl +1;
        }
```

```
printf("The number of lines are %d\n",nl);
printf("The number of lines are %d\n",nc);
printf("The number of words are %d\n", nl + ns);
fclose(fp);
}
```

Example 4: Write a program to merge the contents of two files

```
#include <stdio.h>
#include<stdlib.h>
void main()
{
        FILE *fp,*f1;
        char ch ;
        char filename[40] ;
        char filename1[40] ;
        char filename2[40] ;

        printf("Enter file 1: ");
        gets( filename ) ;

        printf("Enter file 2 : ");
        gets( filename1 ) ;

        printf("Enter file3 : ");
        gets( filename2 ) ;


        fp = fopen( filename, "r" );
        f1 = fopen( filename1, "r" );
        f2 = fopen( filename2, "w" );
        if((fp == NULL)|| (f1 == NULL) || (f2 == NULL))
        {
```

```
        printf("File not created\n");
        exit(1);
        }
        while(12)
        {
        ch = fgetc( fp ) ;
        if(ch == EOF)
        break;
        fputc(ch,f2);
        }
        while(14)
        {
        ch = fgetc( f1 ) ;
        if(ch == EOF)
        break;
        fputc(ch,f2);
        }
        fcloseall( );
}
```


Example 5: Write a program to compare the contents of two files and display wherever they differ

```
#include <stdio.h>
#include<stdlib.h>
void main()
{
        FILE *fp,*f1;
```

```
        char ch,ch1 ;
        char filename[40] ;
        char filename1[40] ;

        printf("Enter file 1: ");
        gets( filename ) ;

        printf("Enter file 2 : ");
        gets( filename1 ) ;

        fp = fopen( filename, "r" );
        f1 = fopen( filename1, "r" );

        if((fp == NULL)|| (f1 == NULL))
        {
        printf("File not created\n");
        exit(1);
        }
        while(12)
        {
        ch = fgetc( fp );
        ch1 = fgetc (f1);
        if((ch == EOF) && (ch1 == EOF))
        break;
        if(ch != ch1)
        printf("%c %c",ch,ch1);
        }
        fcloseall( );
}
```