

C Programming

Computer Fundamentals

Unit-1

Computer:

- The word “Computer” is derived from the Latin word “Computare” which means to calculate.
- Hence, computer is normally considered as a calculating device, which performs the entire task at very fast rate.
- A computer is an electronic machine that accepts data and instructions, stores it until the data is needed, processes large amount of data according to the instructions provided by the user, and finally returns the results to the user

Advantages of computer:

- Computer is a very fast machine.
- Computers operate with almost 100% accuracy.
- Computer is capable of doing required task again and again without any difficulty.
- Computers have capability of storing huge amount of data.
- Computers can perform variety of jobs.

Disadvantages of computer:

- Computer is an automatic device, even it requires operator to operate the computer.
- Computer is very expensive in comparison to other electronic devices, so every people cannot afford it.
- Computer cannot think itself to give decisions.
- Computer requires technical person for maintenance if it does not work properly.

Types of computers:

Analog computer

Digital computer

Analog computer

- The computer which can process analog quantities (continuous data) is called an analog computer.
- They are designed to accept physical quantities such as temperature, pressure, speedometer, etc and record them as readings along a continuous scale.
- Examples of analog devices are: thermometer, speedometer, etc.

Features of analog computer:

- a) These computers work on the derived quantities as temperature, speed, voltage, etc.
- b) They take only one instruction at a time, execute it, give us the result and take next instruction.
- c) To use these computer knowledge of programming is not required.
- d) They do not perform any calculation.

Digital computer

- The computer which accepts discrete data (discontinuous data) is known as digital computer.
- The digital computers operate by counting numbers, or in other words, it represents information in discrete form.
- It operates directly on numbers expressed in binary number system.
- Digital computers are more accurate than analog computer, since there is no analogous representation.

Features of digital computers

- These computers only operate on digit and they are 0 & 1.
- Digital computers do calculation to give us results.
- A digital computer takes all the instruction together and then execute one by one.
- To use these computers knowledge of programming is essential.

Software

- A set of instructions given to the computer in machine code that tells the computer what to do and how to perform the given task of the user is known as computer software.
- The main objective of software is to enhance the performance capability of hardware. Examples: Windows XP, MS- Word, MS-Excel, etc.

Type of Software

- **System software**
- **Application software**

System Software

- It is a collection of program, language, etc which allows the user to communicate with computer equipments, the software makes the machine easier to use and make efficient use of resources of hardware.
- It is like a layer which acts as interface between computer and application software.
- It is used in almost all kinds of computer like mainframe, mini or micro computer.

Types of system software

- Operating system
- Translating Program
- System support software/ utility software

Application software

- application software is prepared and supplied by software company and computer manufacturer.
- Application software is a subclass of computer software that employs the capabilities of computer directly to a task that user wishes to perform.
- Application software are designed to process data and support users in an organization such as solving equations or producing bills, result processing of Pokhara university, etc.

Types of application software

- **Tailored package**
- **Packaged software**

Tailored package

- Tailored software is the software, which are specially designed and developed to solve a specific or particular task.
- The nature of processing of data seems similar but actually they are different in many ways.
- High-level language such as COBOL, BASIC, FORTRAN, PASCAL, C, etc. is used to prepare tailored software.
- Examples: salary sheet, sales ledger, payroll system, library management, etc.

Packaged software

- Packaged software is the software which is generalized a set of programs and developed for general purpose.
- These programs are user friendly and designed for use in more than one environment.
- Examples: Word processing packages: MS-Word, MS-Star etc.
- Spreadsheet: Excel, Lotus, etc.
- Graphics: Photoshop, 3D studio max, etc.
- DBMS Package: Dbase III, Oracle, etc.

WHAT IS Programming LANGUAGE

?

- Language is a collection of words and symbols which can be used to perform certain task or activities and to establish a communication between person to person.
- In the same manner computer languages are the collection of predefined key words which can be used to perform certain task and to communicate between two entities like between two machines or between human and computers or computers and other peripherals.

- A programming language is the series of commands written by programmer in a systematic order which can perform work easily in higher speed.
- A programming language is the platform which provides the environment to write program.
- Actually programming language have their own set of grammatical rules used to write program. Without programming language program is useless. Examples: C, C++, COBOL, Pascal, etc.

There are three different levels of programming languages. They are

(1) Machine languages (low level)

(2) Assembly languages

(3) High-level language

(1) Machine Level Languages (MLL) :-

- Computers are made of No. of electronic components and they all are two – state electronic components means they understand only the 0 – (pulse) and 1 (non – pulse).
- Therefore the instruction given to the computer must be written using binary numbers. 1 and 0.
- Computers do not understand English or any other language but they only understand or respond to binary numbers(0 and 1).
- So each computer has its own Machine languages.

Advantages:

- 1. Time taken to execute program is less.
- 2. The program written in this language need not to be translated.

Disadvantages:

1. Difficult to learn.
2. The knowledge of internal architecture is essential for program coding.

Assembly Level Languages (ALL) :-

- In assembly language instruction are given in English like words ,such as MOV,ADD,SOB etc.
- Hence it is easy to write and understood assembly language.
- As we know computer can understand only machine level language .
- Hence assembly language program must be translated into machine level language.
- The translator which is used for translating is called “assembler”.

Advantages:

- 1. Assembly language is easier to understand as compared to machine language.
- 2. Easier for program debugging.

Disadvantages:

1. It is machine dependent.
2. Program written in assembly language is comparatively larger.

High level language:-

- It is designed keeping in mind the features of portability means these language are machine independent.
- These are the English like language, so it is easy to write and understand the programs of high level language.
- For translating high level language program into machine language we used “compiler” and “Interpreter” are used.
- The language in this category is:-
FORTHAN, COBOL, C, C++, etc.

Advantages:

1. They are easier to learn as compared to assembly and machine language.
2. Lower program preparation cost.
3. Machine independent.

Disadvantage:

1. Require more time and memory to execute.

WHAT IS LANGUAGE TRANSLATORS?

- As we know computer can understand only machine level language ,which is in binary 0 or 1 form .
- Which is difficult to write and maintain the program of machine level language.
- So the need arises for converting the code of high level and low level languages into machine level language, so the translator are used to achieve this process.

Type of Translator

- Assembler
- Compiler
- Interpreter

Assembler

- It is used for converting the code of assembly language into machine level language.
- This translation is done with the help of a translator program called assembler.
- Assembler is a system software supplied by computer manufacturers.

Compiler

- It is used for converting the code of high level language into machine level.
- The compiler searches all the errors of program and lists them.
- A Compiler checks the entire program written by user and if it is free from error and mistakes then produces a complete program in Machine language known as object program.

Interpreter

- The work of interpreter is same as compiler.
- It is also used for converting high level language program into machine level language but it checks the errors of program statement by statement .

Differences between Compiler & Interpreter

- Error finding: Error finding is much easier in Interpreter because it checks and executes each statement at a time. So wherever it finds some error it will stop the execution. Where Compiler first checks all the statements for error and provides lists of all the errors in the program.
- Time: Interpreter takes more time for the execution of a program compared to Compilers because it translates and executes each statement one by one.
- Examples: C, C++, FORTRAN, etc are compiler whereas BASIC, LISP, etc are interpreter.

Problem Solving Method

Unit-2

Problem Analysis

- It arises during the required analysis phase of software development problem analysis is done for obtaining the user requirement.
- This is the process of becoming familiar with the problem that will be solved with a computer program.
- From analysis we can obtain, what is the input and output that is the first step for designing the program.

Algorithm

- It is a simple and informed set of instruction that tell about the logical modules of operations of solving of problem.
- This is the most fundamental concept in computer science.
- It can be defined as a ordered the sequence of well define and effective operation which produce a result and terminate in a finite amount of time when executed.

Features of Algorithm

- Finiteness: - An algorithm terminates after a fixed number of steps.
- Definiteness: - Each steps of a algorithm is precisely defined.
- Effectiveness: - All operation use in algorithm are basic (division, multiplication, compression, etc) and can be performed exactly infixed duration.
- Input: - Algorithm has certain precise input.
- Output: - Algorithm has one or more output.

Some conventions used in developing algorithms

- Each algorithm will be enclosed by two statements START and STOP
- To accept data from user, the INPUT or READ statement is used.
- To display any user message, the PRINT or DISPLAY statement is used.
- The arithmetic operators(+,-,* and /) are used in conditions.
- The relational operators(>,<,>=,<=,==,!=) are used in conditions.
- The logical operators (AND, OR, NOT) are used for logical expressions.

Example of Algorithm

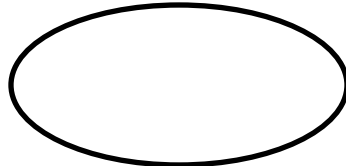
- Step 1: START
- Step 2: Read principle amount, rate of interest and number of years.
- Step 3: Calculate simple interest using the following formula
- $\text{Simple interest} = (\text{Principle amount} * \text{rate of interest} * \text{number of years}) / 100$
- Step 4: Write simple interest result.
- Step 5: STOP

Flow chart

- It is a graphical representation of an algorithm using standard symbols.
- In other words, flowchart is a pictorial representation of an algorithm that uses boxes of different shapes to denote different type of instructions.
- This is a problem solving technique that is widely used.

Symbols that are used in flow chart:

- START/STOP:



The oval shape represents start/stop of the program. Generally, contains words like START, END or STOP.

- Input/output:



The parallelogram represents the I/O function. This step is used to obtaining a number from an input device and showing the results.

- Connection:



Used to join different flowlines and to connect remote parts of the flowchart on the same page.

- Process:



A rectangular represents processing operation.

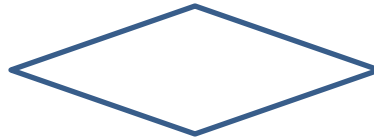
A process changes one or more data.

- Line or arrow:



Line or arrows, represent processing operation direction. The flow of control normally be up to down, right to left.

- Decision making:-



The diamond represent a decision of switching type of option that determine which one of alternative paths to be followed.

Advantages of Flowcharts

Communication:

- Flowcharts are a better way of communications.
- They quickly and clearly provide logic, ideas and descriptions of algorithms to other programmers, students, teacher, computer operators and users.

Effective Analysis:

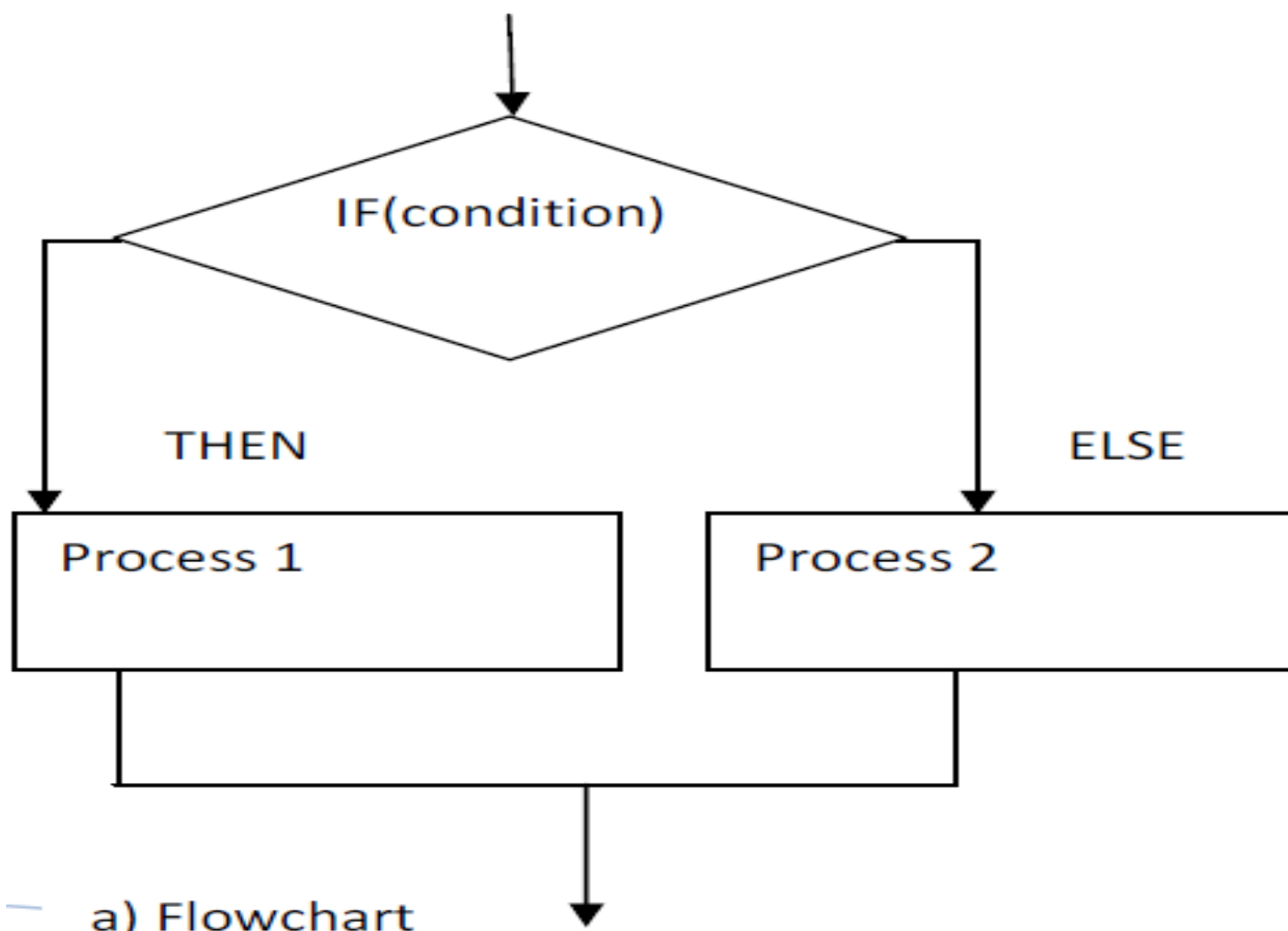
- Flowcharts provide a clear overview of the entire problem and its algorithm for solution.
- They show all major elements and their relationships.
- Thus, with the help of flowcharts, problems can be analyzed more effectively.

Proper Documentation

- The flowchart provides a permanent recording of program logic.
- It documents the steps followed in an algorithms.

Efficient Coding:

- Flowcharts show all major parts of a program.
- A programmer can code the programming instructions in a computer language with more ease with a comprehensive flowchart as a guide.



Stages of Software Development (or Software Development Life Cycle):

- During each phase of the life cycle, a set of well-defined activities are carried out.

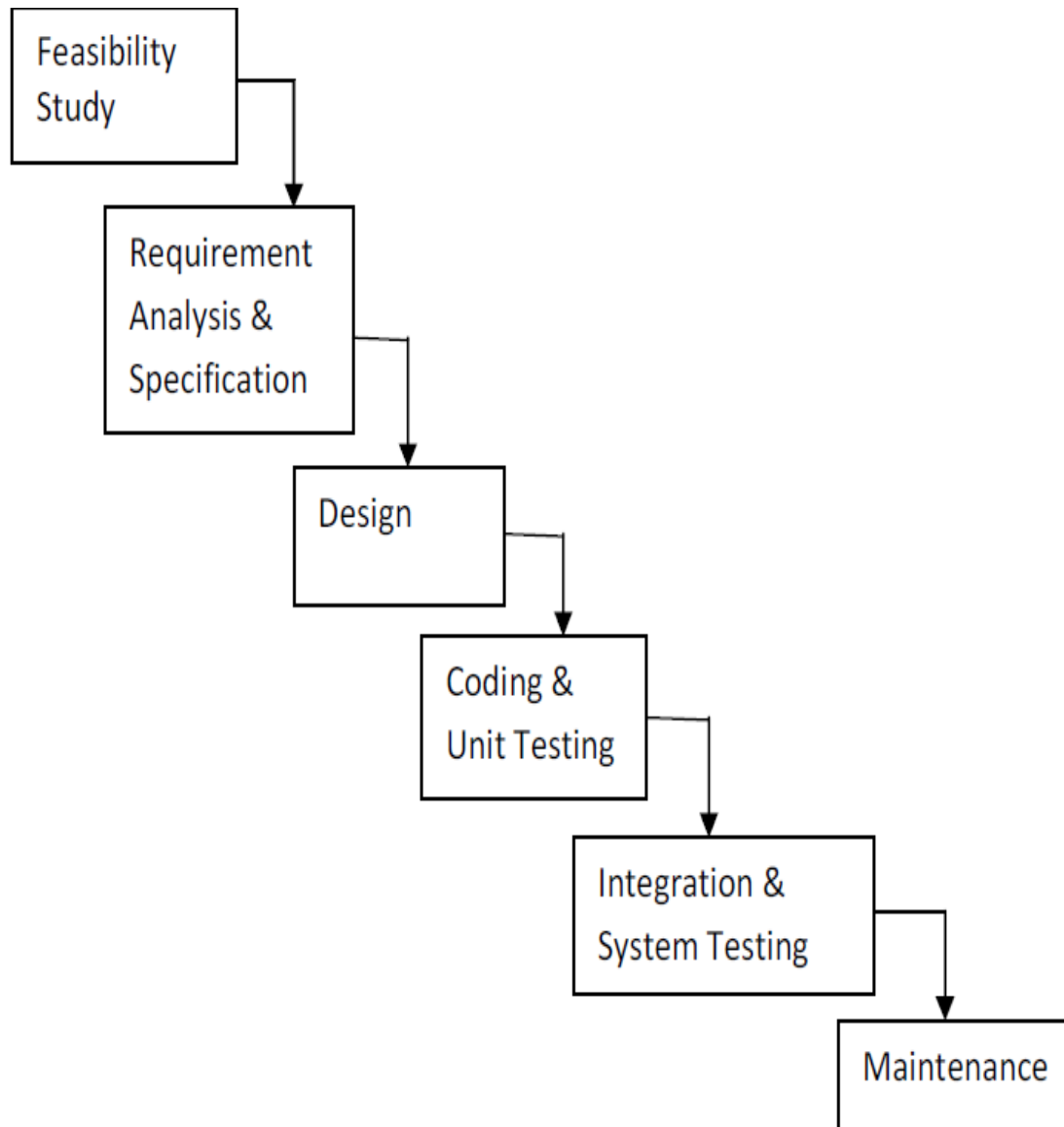


Fig: Software Development Life Cycle (SDLC)

1) Feasibility Study: The main objective of the feasibility study is to determine whether developing the product is financially and technically feasible.

2) Requirement Analysis & Specification: Its main aim is to understand the exact requirements of the customer & to document them properly.

3) Design: The goal of the design phase is to transform the requirements specification into a structure that is suitable for implementation in some programming language.

4) Coding & Unit Testing: It is also called implementation phase. Its purpose is to translate the software design into source code. During the implementation phase, each component of the design is implemented as program module, & each of these program module is unit tested, debugged & document. The purpose of unit testing is to determine the correct working of the individual modules.

5) Integration & System Testing: During this phase the different modules are integrated in a planned manner.

The system testing usually consists of three different kind of testing activities:

- i. α - testing
- ii. β - testing
- iii. Acceptance testing.

6) Maintenance: Maintenance involves performing any one or more of the following three kinds of activities:

- i. Correcting errors that were not discovered during the product development phase. This is called corrective maintenance.
- ii. Improving the implementation of the system & enhancing the functionalities of the system according to the customer's requirements. This is called perfective maintenance.
- iii. Posting the software to a new environment, e.g., to a new computer or to a new operating system. This is called adaptive maintenance.

Step:7> Documentation:-

Documentation refers to a collection of information about the program. There are two type of documentation:

- Programmer's Documentation
- User Documentation.

a). Programmer's Documentation:-It include detailed requirement of program layout of all output reports and input data, the top down documentation.

b). User's documentation: - It provides procedural instruction for installing the program.

Unit 3: HISTORY OF 'C'

The c language was developed in 1970's at Bell laboratories, by Dennis Ritchie . It is designed for the operating system called UNIX.

C is derived from the B language, which is written by ken Thompson at AT and T Bell laboratories. It is excellent, efficient and general purpose language for most of the applications, such as scientific business , system software & application software.

Structure of C program

C program is a collection of one or more function. Every function is a collection of statements that performs some specific task. We can define the structure of c program as:

Comments

Preprocessor directives

Global variables

void main()

{

local variables

statements

.....

.....

}

function 1()

{

local variables

statement

.....

.....

}

Elements of c

Every language has some grammatical rules and basic elements. Before understanding programming it is must to know the basic of c language. These basic elements are:-

- 1). C character set
- 2).Variables
- 3).Data types
- 4).constants
- 5).Keywords
- 6).Variable declaration
- 7).Expression
- 8).statements,etc.

C character set

The character that are used in programs are given below:-

a). Alphabets:- A,B,C,.....Z

a,b,c.....z

b). Digits:-

0,1,2,3.....9

c). Special character:-

Besides some special character are also used which are given below:-

/,+,_*,\$(,),[,],{,},@,.....etc.

Execution character/Escape sequence

Characters are printed on the screen through the keyboard but some character such as newline , tab, backspace, can not be printed like other normal characters. C supports the combination of (\) with the c character set to print these characters. These combinations are known as escape sequences.

Some escape sequence are given below:

<u>Escape sequence</u>	<u>Meaning</u>
\b	backspace
\a	bell
\r	carriage return
\n	new line
\f	form feed
\o	null
\t	tab

Reserved words/ keywords

There are certain words which are reserved for doing specific tasks. These words are known as keywords. These keywords have standard, predefined meaning in C. There are 32 keywords available in C which are given below:-

auto	break	case
continue	else	default
enum	for	struct
goto	double	long
Float	signed	const
int	switch	unsigned
short	void	register
size of	do	type of
volatile	extreme	while
union	char	static
if return		

Variables/Identifiers

Variable is a name that can be used to stored values. These variables can take different values but one at a time. These values can be changed during execution.

Variable names may be consists of uppercase character, lowercase character and under core(_). Rule to giving the name to the variable are as-

- 1). First character should be a letter or underscore (_)
- 2). The variable name cannot be a keyword.
- 3). Uppercase and lower case letter are significant, for example code, Code, CODE is three different variables.

Data types

C support different types of data storage representation of these data type are also different in memory . They are four fundamental data types in C, which are given below:

int,char,float,double

Int:- It is used to store the integer value.

Char:-char is used to store any single character.

Float:- float is used for storing floating point number.

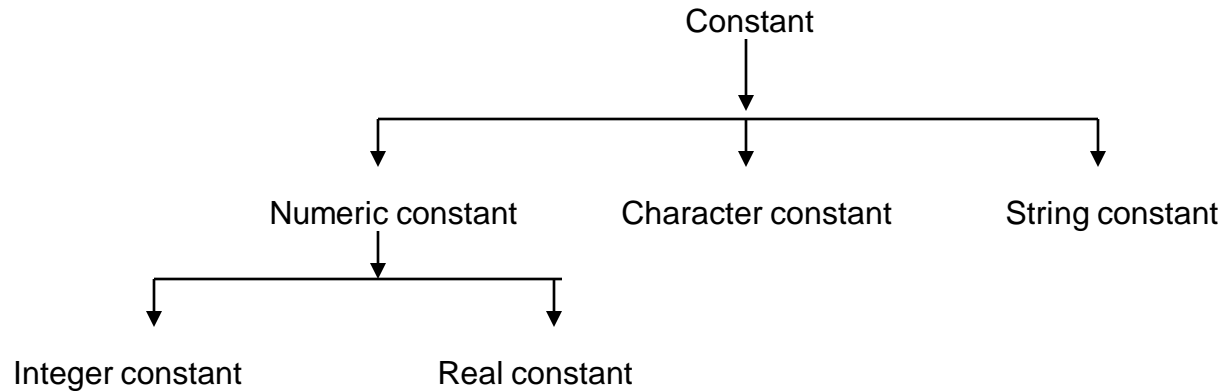
Double:-It is used for storing long range of floating point number.

Size and Range of some data types are given below:

Data types	Size(bytes)	Range
Char or signed char	1	-128 to 127
Unsigned char	1	0 to 255
int or signed int	2	-32768 to 32768
Unsigned int	2	0 to 65535
Float	4	3.4E-38 to 3.4E + 38
Double	8	1.7E-308 to 1.7E+ 308

Constants

It is a value that can be stored in memory and cannot be changed during execution of the program. There are three types of constants:-



1. Numeric Constant

Numeric constants are numeric digit which may or may not decimal point(.).There are the rules for defining numeric constant.

Decimal number	0,1,2,3.....9(Base 10)
Octal number	0,1,2,3.....7(base 8)
Hexadecimal number	0,1,2,3.....A,B,C.....F(base 16)

b). Real constant: Floating point constants are the numbers which hold the decimal point. Some valid floating point constants are:

0.5

5.3

4000.0

0.00073

39.087600

2). Character constant:

A character constant is a single character that is enclosed with single quotes (' '), for example:-

'9'

'D'

'\$'

''

'#'

Every character constant has a unique integer value associated with it.

3).String Constant

A string constant has a zero, one or more than one character. A string constant is enclosed in double quotes(""). At the end of string \0 is automatically placed.

Some example of string constants.

"Bijay"

"593"

//sample program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    const int i=10;
    const float f=5.5;
    const char c='c';
    const char s[15]="hellow";

    printf("\n Integer Constant : %d",i);
    printf("\n Real Constant : %f",f);
    printf("\n Single Character Constant : %c",c);
    printf("\n String Constant : %s",s);
    getch();
}
```

Symbolic constant/define constant

If we want to use certain unique several times then we can use symbolic constant as a name that substitutes for a sequence of characters. The character may represent a numeric constant, a character constant or string constant. This is generally written at the beginning of the program. This is written as:

```
#define name value
```

For e.g.:

```
#define MAX 100
```

```
#define PI 3.14
```

```
#define NAME "Bijay"
```

//Sample Program

#include <stdio.h>

#define LENGTH 10

#define WIDTH 5

#define NEWLINE '\n'

int main()

{

int area;

area = LENGTH * WIDTH;

printf("value of area : %d", area);

printf("%c", NEWLINE);

return 0;

}

These are **ASCII(American standard code for information Interchange)** value for each character of the character set. As example:

A-Z	ASCII	value (65-90)
a-z	ASCII	value (97-122)
0-9	ASCII	value (48-57)
;	ASCII	value (59)

Declaration of variables

It is must to declare the variable before it is used in program. We declare the variable with data type means the variable name can store the value of this data type. Declaration variable can be done as:

Some examples

Data types variable name:

```
int a;
```

```
float b;
```

```
int a=6;
```

```
int a,b,c;
```


Statement

It represent a set of declaration or steps in a sequence of actions. A statement causes the computer to carry out some action. All the statement end with semicolon and execution in sequence. These statement are given below:

1). Assignment statements: In this types of statement we use the assignment operator (=) for assigning the value to the variable. This can be written as:

Variable name=value;

Some example as

Basic=1200;

x=y=z=1;

2). Null statement: A statement that has only semicolon is called null statement. For e.g.; null statement.

3). Expression statement; An expression statement consists of expression. These expression can be arithmetic, logic or relation .As example:

x=5;

x=y-z;

a<=b;

4).Block of statement: A block of statements consists of several statements enclosed with a pair of curly bracket ({}). The statement can be expressions, assignment or control statement. As example:

{

l=4;

b=5;

Area=l*b;

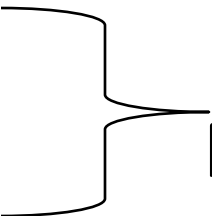
}

Comments

Comments are used for understanding the program later. The comments are used for documentation. Comments are given by starting with `/*` and ending with `*/`. It can be of one or many number of lines.

For e.g.:

`// This is a C program //` Single line comment.

`/* a=13;
B=a-c;
C=a/b; */`  Multiple line comments.

Operators and expressions

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C language is rich in built-in operators and provides following type of operators:

- ✓ Arithmetic Operators
- ✓ Relational Operators
- ✓ Logical Operators
- ✓ Bitwise Operators
- ✓ Assignment Operators
- ✓ Ternary operator
- ✓ Increment and decrement operator

1). Arithmetic operator

It are used for numeric calculation. They are four types:-

a) Binary Arithmetic operator:-It required two operands. There are five operators for binary arithmetic operator.

Operator	meaning	Purpose
+	plus	Addition
-	minus	Subtraction
*	asterisk	Multiplication
%	percent(modulus)	Reminder
/	slash	Division

Example:

a+b

a-b

a/b

a*b

a%b

b) Integer Arithmetic operator

When both operands are integer than the arithmetic operation with these operands is called integer arithmetic.

Example: Let us two variables a and b.

a=17

b=4

Expression	result
a+b	21
a*b	68
a/b	4
a%b	1

```
/*program operate the arithmetic operations */  
#include<stdio.h>  
void main( )  
{  
int a,b;  
printf("Enter the first number a:");  
scanf("%d",&a);  
printf("Enter the second number b:");  
scanf("%d",&b);  
printf("Result of addition:%d\n",a+b);  
printf("Result of subtraction:%d\n",a-b);  
printf("Result of multiplication:%d\n",a*b);  
printf("Result of division:%d\n",a\b);  
printf("Result of modulus:%d\n",a%b);  
getch();  
clrscr();  
}
```

c) Floating point Arithmetic

When both the operands are of float type then arithmetic operation with these operands is called floating point arithmetic.

For example: Let us take two variables a and b. Then the value of $a=124.4$ and $b=3.1$. Then result of these operation are as:

<u>Expression</u>	<u>Result</u>
$a+b$	127.5
$a-b$	121.3
$a*b$	385.64
a/b	40.129


```
/*program to understand floating point arithmetic
operation*/
#include<stdio.h>
void main()
{
float a=9.6,b=1.6;
printf("After addition:%f\n",a+b);
printf("After subtraction:%f\n",a-b);
printf("After multiplication:%f\n",a*b);
printf("After division:%f\n",a/b);
getch();
clrscr();
}
```

2) Relation operator

Relational operation is used to compare two values depending on their relations. It requires values depending on their relations. It required two operands.

An expression that contains the relation operator is called relational expression. If the relation is true then it returns the value 1, if the relation as false then it returns value 0. Then operators are:

Operators	Meaning
<	less then
<=	less then or equal to
= =	equal
!=	not equal
>	greater than
>=	greater than or equal to.

Examples:

Let $a=9$ and $b=5$, then the operation with relational operator are

Expression	Result
$a < b$	false(0)
$a \leq b$	false(0)
$a = b$	false(0)
$a \neq b$	True(1)
$a > b$	True(1)
$a \geq b$	True(1)

3) Logical Operator

An expression that combines two or more expression is terms as a logical expression, for combining these expression we use the logical operator. After testing the value of the condition(which is true or false), it gives the logical status(true or false). C has three logical operators:

- And(&&) operators: This operator gives the net result if both the condition have the value true, otherwise gives the result false.

Example:- let a=10 and b=5 the logical expression is `a==10&&b<a`
this gives the result true because both the condition is true.

- Or(||) operator: This operator gives the net result false. If both the conditions have the value false, otherwise it gives the result true.

Example:- Let us take

a=7 ,b=12

The logical expression $a \leq b \mid \mid b > 15$ this gives the result true because one condition is true.

- Not(!) operators: This is unary operator. This relates the value of the condition. If the value of condition is true then it gives the result false.

Example: Let us take a=10

Then logical expression $!(a == 10)$ Hence, the result is false.

4).Assignment operator

A value can be stored in variables with the use of assignment operator. This assignment statement “=” is used for assignment statement.

Example:

$x=x+5$

or

$x+=5$

similarly,

$x=x-5$ is equivalent to $x-=5$

$x=x/5$ is equivalent to $x/=5$

$x=x\%5$ is equivalent to $x\%=5$.

5).Ternary operator

Ternary operator(? and :) require three operand.
This is written as:

Condition? Operand 1: Operand 2

First condition is tested if the result is true then the value of the operand 1 is taken otherwise value of operand 2 is taken.

Example:

Maximum=a>b?a:b

First the condition a>b is evaluated, if the value is true then the value of 'a' is taken by maximum, otherwise value of 'b' is given to the maximum.

/*write a program to print the smallest between two numbers with use of ternary operators*/

```
#include<stdio.h>

void main()
{
    Int a,b,min;
    printf("Enter the first number:");
    scanf("%d",&a);
    printf("Enter the second number:");
    scanf("%d",&b);
    min=a<b?a:b;
    printf("Smallest number between %d and %d is %d",a,b,min);
    getch();
    clrscr();
}
```



```
/*Program to print the largest between two number with use  
of ternary operator*/  
#include<stdio.h>  
void main( )  
{  
int a,b,max;  
printf("Enter the first number:");  
scanf("%d",&a);  
printf("Enter the second number:");  
scanf("%d",&b);  
max=a>b?a:b;  
printf("Largest between %d and %d is %d",a,b,max);  
getch();  
clrscr();  
}
```

6).Bitwise operator

C has the ability to support the manipulation of data at the bit level. Bitwise operators are used for operations on bits.

Bitwise operators are operated on only integer. The bitwise operators are given below:

<u>Bitwise Operator</u>	<u>Meaning</u>
&	bitwise AND
	bitwise OR
~	one's complement
<<	left shift
>>	right shift
^	bitwise XOR.

7).Increment and decrement operator

C has two useful operators increment (++) and decrement(--). There are the unary operator because these operators on only single operand. The increment operator (++) increment value of the variable by 1 and decrement operator(--) decrement of the variable by 1. We can write this as:

++a	or	a=a+1
--a	or	a=a-1

/* program to understand the use of prefix increment
/decrement */

```
#include<stdio.h>
void main()
{
int a=3;
printf("a=%d\n ",a);
printf("a=%d\n",++a);
printf("a=%d\n",a);
printf("a=%d\n",- -a);
printf("a=%d\n",a);
}
```

b) Postfix operator

Here, first the value of variables is taken for operation then value of variables is incremented /decrement.

Example:

a++

a- -

let us take value of a=3

then b=a++

Hence,

a=4 b=3

b=a- -

a=3 b=4

```
/* program to understand the use of postfix  
increment /decrement*/  
#include<stdio.h>  
void mani( )  
{  
int a=3;  
printf("a=%d\n",a);  
printf("a=%d\n",a++);  
printf("a=%d\n",a);  
printf("a=%d\n",a- -);  
printf("a=%d\n",a);  
}
```

Unit 4:Input/output in C

Definition:

It takes the data as input, processes this data and gives the output. The C language has collection of library functions that include several input- output functions. Some input /output functions are given below:

- **Unformatted input/output functions:** Unformatted I/O functions works only with character datatype (char).

The **unformatted Input functions** used in C are getch(), getchar(), gets().

➤ **getchar():** getchar() accepts one character type data from the keyboard.

Syntax for getchar() in C :

variable_name = getchar();

- **gets():** gets() accepts any line of string including spaces from the standard Input device (keyboard). gets() stops reading character from keyboard only when the enter key is pressed.

Syntax for gets() in C :

```
gets(variable_name);
```


- The **unformatted output** statements in C are **putchar** and **puts**.

- **Putchar():** **putchar** displays one character at a time to the Monitor.

Syntax:

`putchar(variable_name);`

- **Puts():** **puts** displays a single / paragraph of text to the standard output device.

Syntax:

`puts(variable_name);`

//Sample program :

```
#include<stdio.h>
#include<conio.h>
void main()
{
char a[20];
gets(a);
puts(a);

getch();
}
```

○ **Formatted I/O in C :**

The formatted Input / Output functions can read and write values of all data types, provided the appropriate conversion symbol is used to identify the datatype.

➤ **scanf():** is used for receiving formatted Input.

Syntax :

```
scanf("control_string 1, - - , control_string n",&variable1,  
- - - - ,variable n);
```

➤ **printf ():**is used for displaying formatted output.

Syntax:

```
printf("control_string 1, - - , control_string n",variable1, -  
- - - , variable n);
```

//Sample Program :

```
#include<stdio.h>
#include<conio.h>
void main()
{
int no;
char ch;
char s[20];

printf("Enter a character : ");
scanf("%c",&ch);

printf("Enter a No : ");
scanf("%d",&no);

printf("Enter a Word or String : ");
scanf("%s",&s);

printf("\n Character : %c",ch);
printf("\n No : %d",no);
printf("\n String : %s",s);
getch();
}
```

EXAMPLES

First method:

```
/* Program for output fin value*/  
#include<stdio.h>  
Void main( )  
{  
int a=6;  
int b=10;  
printf("The value of a is:%d",a);  
printf("The value of b is:%d",b);  
}
```

Second methods:

```
#Include<stdio.h>  
void main()  
{  
int a,b;  
printf("Enter the value of a");  
scanf("%d",&a);  
printf("Enter the value of b");  
scanf("%d",&b);  
printf("The value of a is:%d",a);  
printf("The value of b is:%d",b);  
getch();  
clrscr();
```

Some examples;-

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

```
void main( )
```

```
{
```

```
printf("c is excellent");
```

```
}
```

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

```
void main( )
```

```
{
```

```
int basic=2000;
```

```
printf("Basic=%d",basic);
```

```
}
```

Lab Exercise

1. ***#include<stdio.h>***
void main()
{
int a,b,c;
a=125;
b=2678;
c=124589;
printf(“%5d,%5d,%5d\n”,a,b,c);
printf(“%3d,%4d,%5d\n”,a,b,c);
}

2. ***#include<stdio.h>***
void main()
{
int a=98;
char ch='b';
printf(“%c,%d\n”,a,c,h);

3. ***#include<stdio.h>***
void main()
{
printf(“%10s\n”,“Nepal”);
printf(“%4s\n”,“Nepal”);
printf(“%2s\n”,“Nepal”);
printf(“%5.25s\n”,“Nepal”);
}

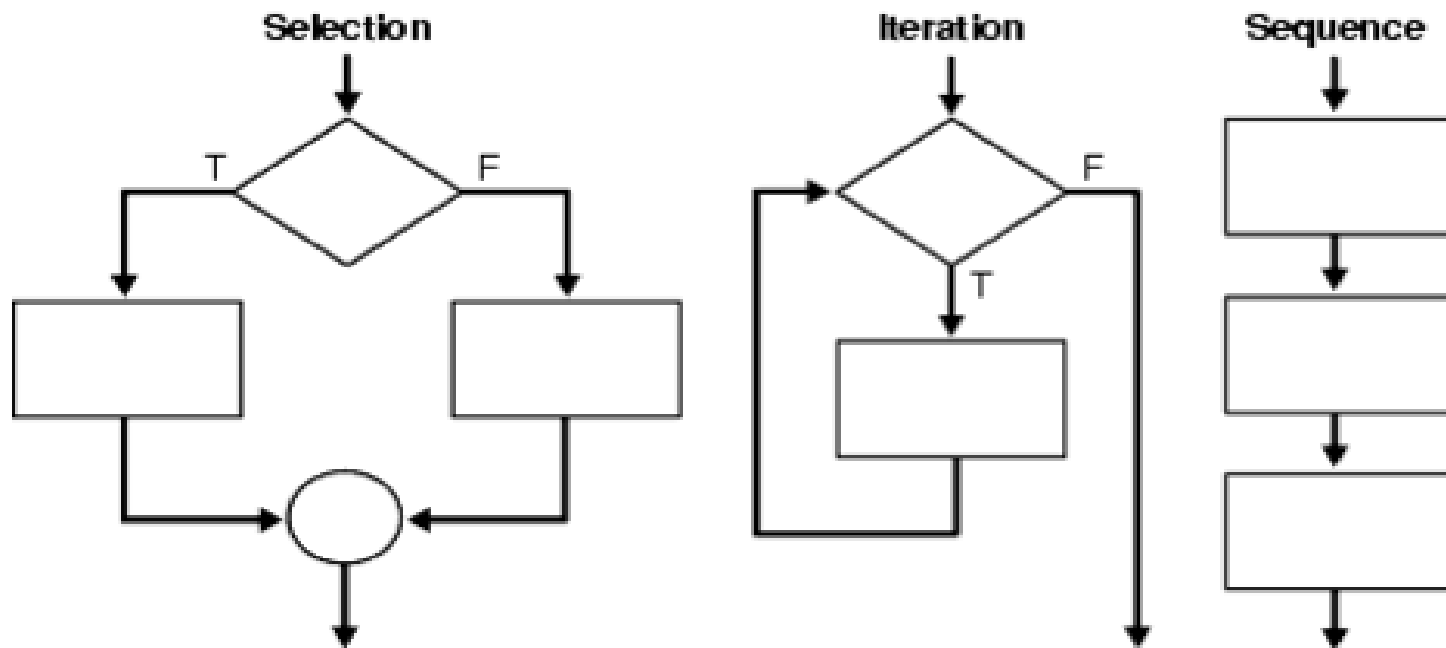
Unit 5:

Structured Programming Fundamentals:

Control statement

Control statement enable us to specify the order in which the various instructions in the program are to be executed .This determines the flow of control. Control statement defines how the control is transferred to others parts of program. Control statement defines the flow in which execution of statement should take place to achieve the requirement result.

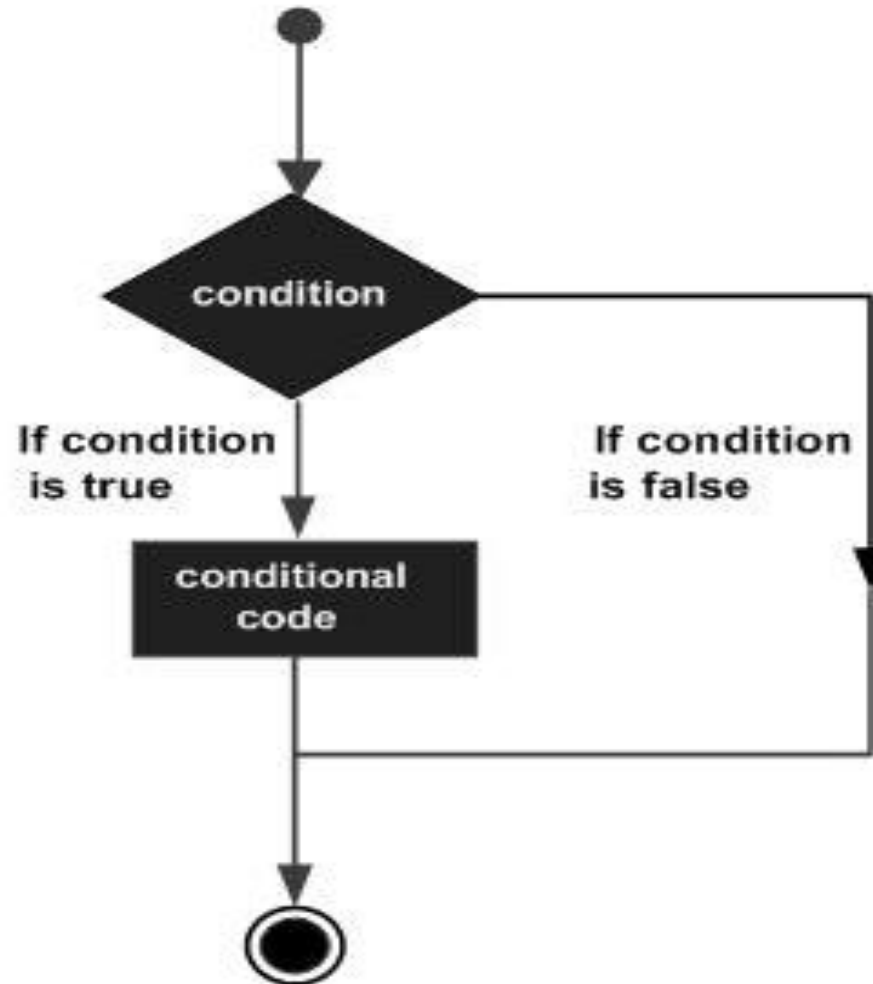
Fig: control structure



C Language support three types of control statement which are as:

a) **Decision making structures(selection structure)**: require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure:



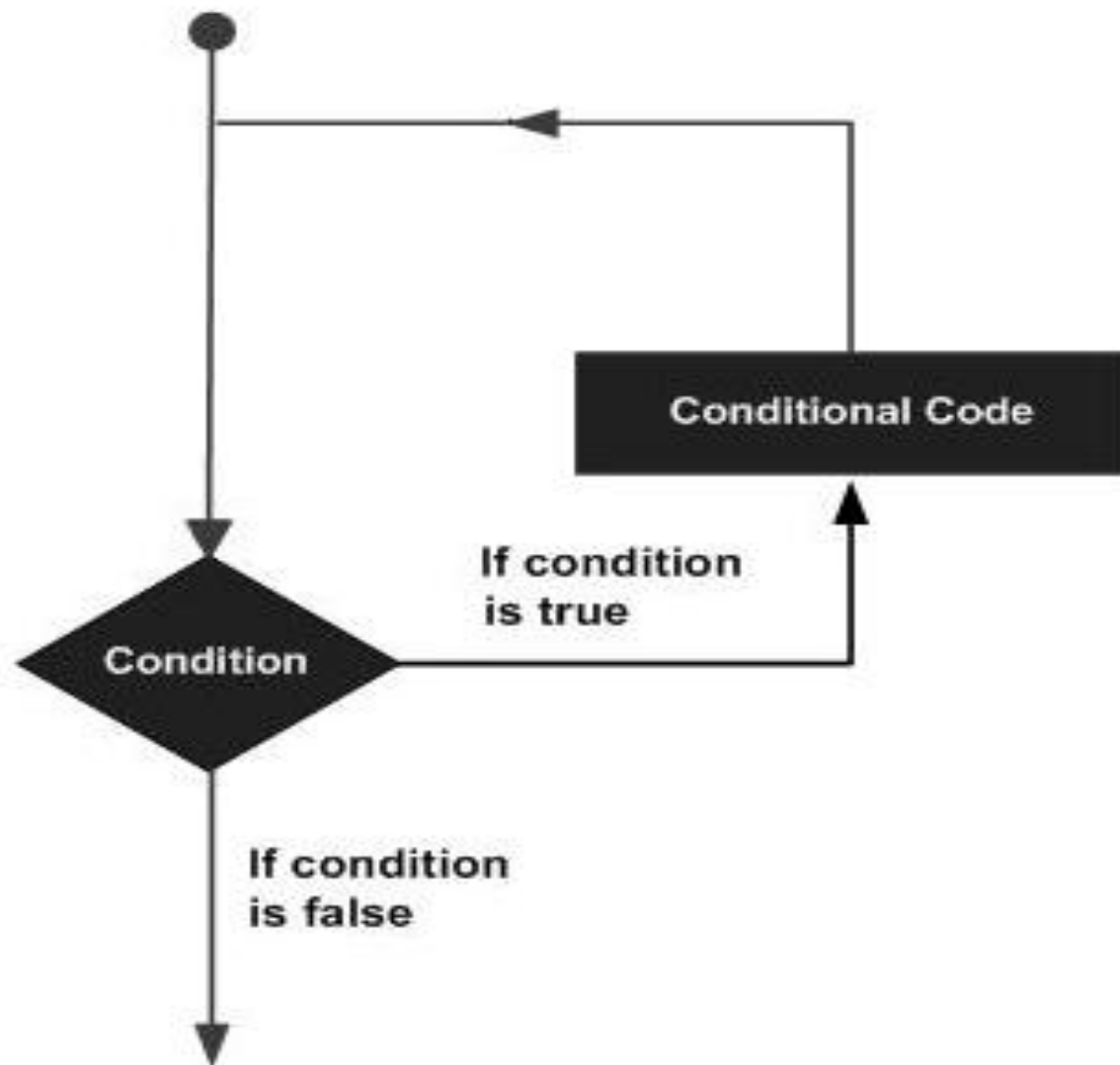
C programming language provides following types of decision making statements:

Statement	Description
if statement	An if statement consists of a boolean expression followed by one or more statements.
if...else statement	An if statement can be followed by an optional else statement , which executes when the boolean expression is false.
nested if statements	You can use one if or else if statement inside another if or else if statement(s).
switch statement	A switch statement allows a variable to be tested for equality against a list of values.
nested switch statements	You can use one switch statement inside another switch statement(s).

b) loop(Iteration structure): A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages:

- while
- do....while
- for

c) Sequence structure: The sequence structure simply executes a sequence of statements in the order in which they occur.



If...statement

This is bi-directional conditional statement .This statement is used to test condition and take one of the two possible actions. If the condition is true then a single statement or a block of statement as executed (one part of program), otherwise another single statement or a block of statement is executed (other part of the program).

Syntax: if statement

If (condition)

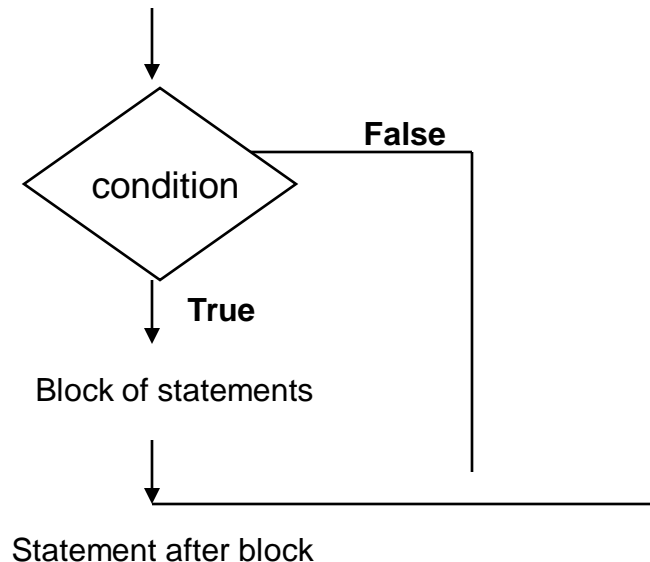
{

Statement

.....

.....

}




```
/* print the length and smallest of the two numbers*/.  
#include<stdio.h>  
void main()  
{  
int a,b;  
printf("enter the first number:");  
scanf("%d",&a);  
printf("enter the second number:");  
scanf("%d",&b);  
if(a>b)  
printf("largest number=%d and smallest number=%d\n",a,b);  
else  
printf("largest number=%d and smallest number=%d\n",b,a);  
}
```

if...else if...else Statement

An **if** statement can be followed by an optional **else if...else** statement, which is very useful to test various conditions using single if...else if statement.

Syntax2: ifelse statement

```

    If (condition)
    {
    Statement
.
    .....
    }
elseif()
{
Statement
.....
}
else
{
}
```

```
//Sample example
#include <stdio.h>
int main ()
{
    int a = 100;
    if( a == 10 )
    {
        printf("Value of a is 10\n" );
    }
    else if( a == 20 )
    {
        printf("Value of a is 20\n" );
    }
}
```

```
else if( a == 30 )
{
printf("Value of a is 30\n" );
}
else
{
printf("None of the values is matching\n" );
}
printf("Exact value of a is: %d\n", a );
return 0;
}
```

Syntax3: Nested if..else statement

It is always legal in C programming to nest if-else statements, which means you can use one if or else if statement inside another if or else if statement(s).

Syntax:

The syntax for a nested if statement is as follows:

```
if( boolean_expression 1)
{
    /* Executes when the boolean expression 1 is true */
    if(boolean_expression 2)
    {
        /* Executes when the boolean expression 2 is true */
    }
}
```

Example:

```
#include <stdio.h>

int main ()
{
    /* local variable definition */
    int a = 100;
    int b = 200;

    /* check the boolean condition */
    if( a == 100 )
    {
        /* if condition is true then check the following */
        if( b == 200 )
        {
            /* if condition is true then print the following */
            printf("Value of a is 100 and b is 200\n" );
        }
    }
    printf("Exact value of a is : %d\n", a );
    printf("Exact value of b is : %d\n", b );

    return 0;
}
```

```
/* print whether the number is even or odd*/  
#include<stdio.h>  
void main()  
{  
int n;  
printf("enter the number:");  
scanf("%d",&n);  
if(n%2==0)  
printf("number is even\n");  
else  
printf("number is odd\n");  
}
```

/* program to understand the resting of 'if.....else' statement .Find the largest number between any three number*/

```
#include<stdio.h>
void main( )
{
int a,b,c;
printf("enter the first number:");
scanf("%d",&a);
printf("enter the second number:");
scanf("%d",&b);
printf("enter the third number:");
scanf("%d",&c);
if(a>b&&b>c)
printf("%d is the largest number\n",a);
elseif(b>c&&c>a )
printf("%d is the largest number\n",b);
else
printf("%d is the largest number\n",c);
getch();
clrscr();
}
```


Switch :

This is multi-directional conditional control statement. This is a possibility in program to make a choice among number of alternatives. For making a choice, we use the switch statement. This can be written as:

```
Syntax:
switch(expression)
{
case constant1:
statement
.....
case constant 2:
statement
.....
case constant 3:
statement
.....
default:
statement
.....
}
```

```
/*program to understand the switch control statement*/
```

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

```
void main()
```

```
{
```

```
int ch;
```

```
printf("enter your choice:");
```

```
scanf("%d",&ch);
```

```
switch(ch)
```

```
{
```

```
case 1:
```

```
printf("First\n");
```

```
break;
```

```
case 2:
```

```
printf("Second\n");
```

```
break;
```

```
case 3:
```

```
printf("Third\n");
```

```
break;
```

```
default:
```

```
printf("Wrong choice\n");
```

```
}
```

```
}
```

Nested Switch

It is possible to have a switch as part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.

The syntax for a nested switch statement is as follows:

```
switch(ch1) {  
    case 'A':  
        printf("This A is part of outer switch" );  
        switch(ch2) {  
            case 'A':  
                printf("This A is part of inner switch" );  
                break;  
            case 'B': /* case code */  
        }  
        break;  
    case 'B': /* case code */  
}
```

//sample program

```
#include <stdio.h>

int main ()
{
    /* local variable definition */
    int a = 100;
    int b = 200;

    switch(a) {
        case 100:
            printf("This is part of outer switch\n", a );
            switch(b) {
                case 200:
                    printf("This is part of inner switch\n", a );
            }
    }
    printf("Exact value of a is : %d\n", a );
    printf("Exact value of b is : %d\n", b );

    return 0;
}
```

while loop: statement in C programming language repeatedly executes a target statement as long as a given condition is true.

Syntax:

```
while(condition)
{ statement(s); }
```

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any nonzero value. The loop iterates while the condition is true.

Example:

```
#include <stdio.h>

int main ()
{
    /* local variable definition */
    int a = 10;

    /* while loop execution */
    while( a < 20 )
    {
        printf("value of a: %d\n", a);
        a++;
    }

    return 0;
}
```

/# Find the factorial of any number#/

```
#include <stdio.h>
void main( )
{
int n,no,fact=1;
printf("Enter the number:");
scanf("%d",&n);
no=n;
if(n<0)
printf("No factorial of negative number\n");
else
if(n==0)
printf("factorial of Zero is 1\n");
else
while(n>1)
{
fact=fact*n;
n- -;
}
printf("factorial of %d=%d\n",no,fact);
}
```


do...while loop: is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax:

do

{ statement(s); }while(condition);

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.

Example:

```
#include <stdio.h>

int main ()
{
    /* local variable definition */
    int a = 10;

    /* do loop execution */
    do
    {
        printf("value of a: %d\n", a);
        a = a + 1;
    }while( a < 20 );

    return 0;
}
```

for loop: is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax:

```
for ( init; condition; increment )  
{  
    statement(s);  
}
```

Here is the flow of control in a for loop:

- The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed.
- After the body of the for loop executes, the flow of control jumps back up to the **increment** statement.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for loop terminates.

Example:

```
#include <stdio.h>

int main ()
{
    /* for loop execution */
    for( int a = 10; a < 20; a = a + 1 )
    {
        printf("value of a: %d\n", a);
    }

    return 0;
}
```

```
/*print the number from 1 to 10*/  
#include<stdio.h>  
void main( )  
{  
int n=1;  
do  
{  
printf(“%d\n”,n);  
n=n+1;  
}  
while(n<11);  
}
```

Different between while and Do....while:

- In do.....while loop a single, statement or block of statement are executed at least once, and then the condition is evaluated.
- In a 'while' loop first the condition is evaluated and then the statement are executed.

Other Control Statement in c

break statement: in C programming language has the following two usages:

- When the **break** statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
- It can be used to terminate a case in the **switch** statement

Syntax:

```
break;
```

Example:

```
#include <stdio.h>

int main ()
{
    /* local variable definition */
    int a = 10;

    /* while loop execution */
    while( a < 20 )
    {
        printf("value of a: %d\n", a);
        a++;
        if( a > 15)
        {
            /* terminate the loop using break statement */
            break;
        }
    }

    return 0;
}
```


continue statement: in C programming language works somewhat like the **break** statement. Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between.

Syntax:

```
continue;
```

Example:

```
#include <stdio.h>

int main ()
{
    /* local variable definition */
    int a = 10;

    /* do loop execution */
    do
    {
        if( a == 15)
        {
            /* skip the iteration */
            a = a + 1;
            continue;
        }
        printf("value of a: %d\n", a);
        a++;
    }while( a < 20 );

    return 0;
}
```

goto statement: in C programming language provides an unconditional jump from the goto to a labeled statement in the same function.

Syntax:

The syntax for a **goto** statement in C is as follows:

```
goto label;
```

```
.. .
```

```
label: statement;
```

```
/*print the largest and smallest of two number*/  
#include<stdio.h>  
void main()  
{  
int a,b;  
printf("enter the first number:");  
scanf("%d",&a);  
printf("enter the second number:");  
scanf("%d",&b);  
if(a>b)  
goto large;  
else  
goto small;  
large:  
printf("largest number =%d and smallest number=%d",a,b);  
goto end ;  
small:  
printf("largest number =%d and smallest number=%d",b,a);  
goto end;  
end:  
printf("\n");  
}
```

Unit 6: Function in c

A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions. A function is known with various names like a method or a sub-routine or a procedure, etc. C program has two types functions:

- Library function
- User-defined function

Library function:

C has the facility to provides some library function to the programmer for doing some operations. As an examples, C has a mathematical function which is used for finding out square root of any number ,sting library function is used for string operations as example strlen() is used to find out the length of a string and strcmp() is used for compare two strings. These operations are programmed and stored as library function so that they can be called by any program.

```
/* Find the square root of any number*/  
#include<stdio.h>  
#include<math.h>  
void main()  
{  
int n;  
printf("enter the number");  
scanf("%d",&n);  
printf("The square root of %d is :%f",n,sqrt(n));  
}
```

User-define function

Library function provide the function for doing some predefined operations. Users can also create functions for doing any specific task of the program. Such functions are called user defined function. The user defined functions are of three types.

- Function with no argument and no return value.
- Function with argument but no return value.
- Function with argument and return value.

Defining a Function:

The general form of a function definition in C programming language is as follows:

```
return_type function_name( parameter list )  
{  
    body of the function  
}
```

Here are all the parts of a function:

- **Return Type:** A function may return a value.
- **Function Name:** This is the actual name of the function.
- **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument.
- **Function Body:** The function body contains a collection of statements that define what the function does.

Example:

Following is the source code for a function called `max()`. This function takes two parameters `num1` and `num2` and returns the maximum between the two:

```
/* function returning the max between two numbers */
int max(int num1, int num2)
{
    /* local variable declaration */
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

Function Declarations:

- A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.
- A function declaration has the following parts:
return_type function_name(parameter list);
- Following is the function declaration:
int max(int num1, int num2);

Calling a Function: A called function performs defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.

```
#include <stdio.h>

/* function declaration */
int max(int num1, int num2);

int main ()
{
    /* local variable definition */
    int a = 100;
    int b = 200;
    int ret;

    /* calling a function to get max value */
    ret = max(a, b);

    printf( "Max value is : %d\n", ret );

    return 0;
}

/* function returning the max between two numbers */
int max(int num1, int num2)
{
    /* local variable declaration */
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

Function Arguments:

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters** of the function.

The formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are two ways that arguments can be passed to a function:

- **Call by value:** The **call by value** method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument. By default, C programming language uses *call by value* method to pass arguments.


```
#include <stdio.h>

void swap(int x, int y);

int main ()
{
    int a = 100; int b = 200;
    printf("Before swap, value of a : %d\n", a );
    printf("Before swap, value of b : %d\n", b );
    swap(a, b);
    printf("After swap, value of a : %d\n", a );
    printf("After swap, value of b : %d\n", b );
    void swap(int x, int y)
    {
        int temp;
        temp = x;
        x = y;
        y = temp;
        printf("In the function values changed :%d,%d",x,y);
    }
}
```

- **call by reference** :The **call by reference** method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.

```
#include <stdio.h>

void swap(int *x, int *y);

void main ()
{
    int a = 100; int b = 200;
    printf("Before swap, value of a : %d\n", a );
    printf("Before swap, value of b : %d\n", b );
    swap(&a, &b);
    printf("After swap, value of a : %d\n", a );
    printf("After swap, value of b : %d\n", b );
    void swap(int *x, int *y)
    {
        int temp;
        temp = *x;
        *x = *y;
        *y = temp;
        printf("In the values changed :%d, %d", x,y);
    }
    getch();
    clrscr();}
```

storage classes

A storage class defines the scope (visibility) and life-time of variables and/or functions within a C Program. They precede the type that they modify. There are three types of storage classes:

- automatic
- external
- static
- register

a) **auto** : The **auto** storage class is the default storage class for all local variables.

```
{ int mount; auto int month; }
```

b) **register** storage class : The **register** storage class is used to define local variables that should be stored in a register instead of RAM.

```
register int miles;
```

c) The **extern** storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern', the variable cannot be initialized however, it points the variable name at a storage location that has been previously defined.

```
extern void write_extern();
```

d) Static Storage Class

The value of static variable persists until the end of the program. A variable can be declared static using keyword: static. For example:

```
static int i;
```

Here, *i* is a static variable.

Recursion: The special feature of C language is that it supports the recursion function. The function which calls itself (in function body) again and again is known as the recursion function. This function will call itself as long as the condition is satisfied.

Syntax:

```
main()
```

```
{
```

```
.....
```

```
function();
```

```
.....
```

```
}
```

```
fun()
```

```
{
```

```
.....
```

```
.....
```

```
fun(); recursion call
```

```
}
```

```
/* Find the factorial of any number*/  
#include<stdio.h>  
int factorial(int n);  
void main()  
{  
    int n,value;  
    printf("enter the number:");  
    scanf("%d",&n);  
    if(n<0)  
        printf("No factorial of negative number\n");  
    else  
        if(n==0)  
            printf("Factorial of Zero is 1\n");  
        else  
        {  
            value= factorial(n);
```



```
printf ("Factorial of %d=%d\n",n,value);  
}  
  
}  
  
factorial(k)  
{  
    int fact=1;  
    if(k>1)  
        fact=k*factorial(k-1);/ *recessive function call* /  
    return(fact);  
}
```

ARRAY

Array: In many situations, it may be possible that it is useful to collect the similar type of data items support concept of array for this purpose. An array is a collection of similar type of data items. The elements of array is indicated by specifying the array name followed by subscript in brackets. Let us take an array.

```
arr[10];
```

The elements of this array is arr[0],arr[1],arr[2].....arr[9].

Here number of subscripts determine the dimension of array. The value in the first bracket is size of the array.

If the size of array is n, for example arr[n]; Then arr[6] is the first element and arr[n-1] is the last element of array.

Declaration of Array

The array should be declare with the data type, array name and number of subscript in the bracket. By declaring an array, the number of memory locations (size of array)is allocated in memory.

The syntax for declaration of array is

data type array-name [expression].

Example:

The array can be declare as-

```
Int age[10];
```

```
Float sal[10];
```

```
Char grade[10];
```

Here first one is the integer type array. Every element of array can hold an integer value .Second one is the floating type array, can hold a float value and third one is the character type array, can hold one character value.

Processing with array:

We know the elements of array is stored in contiguous memory locations.

As example;

```
Int arr[5];
```

This is stored in memory as

100	500	800	101	200
-----	-----	-----	-----	-----

Arr[1]	arr[2]	arr[3]	arr[4]
--------	--------	--------	--------

We can take the value in the array as other variables by scanf()statement

```
printf(“%d”,arr[1]);
```

Let us take the character type array.

```
Char arr[10];
```

We can take the value in character type array as

```
scanf(“%c”,arr[1]);
```

```
scanf(“%s”,arr);
```

Similarly we can print the character type array by printf() statement as

```
printf(“%c”,arr[1]);
```

Or

```
printf(“%5”,arr);
```

/* program to print sum of 10 numbers*/

```
#include<stdio.h>
void main()
{
int arr[10],i, sum=0;
for(i=0;i<10;i++)
{
printf("Enter the %d number \n",i+1);
scanf("%d",&arr[i]);
}
for (i=0;i<10;i++)
{
sum=sum+arr[i];
printf("sum=%d\n",sum);
}
}
```

/* program to accept 10 number and print the number is odd or even*/

```
#include<stdio.h>
void main()
{
    int arr[10],i;
    for(i=0;i<10;i++)
    {
        printf("Enter the %d number ",i+1);
        scanf("%d",& arr[i]);
    }
    for(i=0;i<10;i++)
    {
        if(arr[i]%2==0)
            printf("%d number is even\n",arr[i]);
        Else
            printf("%d number is odd/n",arr[i]);
    }
}
```

/* program to accept 10 characters and print them*/

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

```
void main()
```

```
{
```

```
char arr[10];
```

```
int i;
```

```
for(i=0; i<10; i++)
```

```
scanf("%c",&arr[i]);
```

```
printf("Enter characters are :\n");
```

```
for (i=0;i<10;i++)
```

```
printf("%c\n", arr[i]);
```

```
}
```

/* program to accept any string and print them*/

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

```
void main()
```

```
{
```

```
char str[15];
```

```
printf("Enter the string");
```

```
scanf("%s",str);
```

```
}
```


Two Dimensional Array:

As we know one dimensional array use one bracket, two dimensional array use two brackets. Similarly-dimensional array use n brackets.

Example:

Arr[i][J] means the starting element of array is arr[0][0] and the last element of array is arr[i-1][J-1]. Here in arr[i][j], i denotes the row in the array and J denotes the column in array or elements in each row.

```
/* program to input and print the elements of matrix */
#include<stdio.h>
void main()
{
int mat[3][3],i,j,row,col;
printf("Enter the row of matrix:");
scanf("%d",&row);
printf("Enter the column of matrix:");
scanf("%d",&col);
print("Enter matrix elements");
for(i=0;i<row;i++)
{
for(j=0;j<col;j++)
{
scanf("%d",&mat[i][j]);
}
}
}
```

```
printf("matrix is:\n");  
for(i=0;i<row;i++)  
{  
  for(j=0; j<col;j++)  
  {  
    printf("%d\t",mat [i][j]);  
    printf("\n");  
  }  
}
```

```
/* Program to addition of two matrix */
#include<stdio.h>
void main()
{
int mat1[3][3],mat2[3][3],mat3[3][3],l,j,row,col;
printf("Enter the row and column of matrix:");
scanf("%d%d",&row,&col);
printf("For the first matrix:\n");
for(i=0;i<row;i++)
{
for(j=0;j<col;j++)
{
print("Enter the elements of first matrix:");
scanf("%d",&mat1[i][j]);
}
}
}
```

```
printf("Enter the elements of second  
matrix:");  
for(i=0;i<row;i++)  
{  
for(j=0;j<col;j++)  
{  
scanf("%d",&mat2[i][j]);  
}  
}
```

/*For addition*/

```
for(i=0,i<row;i++)
{
for(j=0;j<col;j++)
{
mat3[i][j]=mat1[i][j]+mat2[i][j];
}
}
printf("Matrix1 is:\n");
for(i=0;i<row;i++)
{
for(j=0;j<col;j++)
{
printf("%d\t";mat1[i][j]);
}
printf("\n");
}
```

```
printf("matrix2 is:\n");
for(i=0;i<row;i++)
{
for(j=0;j<col;j++)
{
printf("%d\t",mat2[i][j]);
}
printf("\n");
printf("After addition matrix is:n");
for(i=0;i<row;i++)
{
for(j=0;j<col;j++)
{
printf("%d\t",mat3[i][j]);
}
printf("\n");
}
}
```

/*program to transpose of matrix*/

```
#include<stdio.h>
void main()
{
    int mat1[3][3],mat2[3][3],i,j,row,col;
    printf("Enter row and column of matrix:\n");
    scanf("%d%d",&row&clo);
    printf("Enter the elements of matrix:\n");
    for(i=0;i<row;i++)
    {
        for(j=0;j<col;j++)
        {
            scanf("%d",&mat1[i][j]);
        }
    }
    printf("matrix is:\n");
    for(i=0;i<row;i++)
    {
        for(j=0;j<col;j++)
        {
            printf("%d\t",mat1[i][j]);
        }
        printf("\n");
    }
}
```


/*for transpose of matrix*/

```
    for(i=0;i<row;i++)
    {
        for(j=0;j<col;j++)
        {
            mat2[i][j]=mat1[j][i];
        }
    }
printf("Transpose of matrix is:\n");
    for(i=0;i<row;i++)
    {
        for(j=0;j<col;j++)
        {
            printf("%d\t",mat2[i][j]);
        }
        printf("\n");
    }
}
```

#/*program to multiplication of two matrix*/

```
#include<stdio.h>
void main()
{
int mat1[3][3],mat2[3][3],mat3[3][3],i,j,row,col,row2,col2,k;
printf("Enter the row and column for first matrix:\n");
scanf("%d%d",&row1,&col1);
printf("Enter the row &column for second matrix:\n");
scanf("%d%d",&row2,&col2);
if(col1!=row2)
{
printf("Multiplication of two matrix not possible:");
}
else
printf("enter the elements of first matrix:\n");
for(i=0;i<row1;i++)
{
for(j=0;j<col;j++)
{
scanf("%d",&mat1[i][j]);
}
}
}
CONT.....
```

```

.....
printf("enter the element of second matrix:");
    for(i=0;i<row2;i++)
    {
        for(j=0;j<col2;j++)
        {
            scanf("%d",&mat2[i][j]);
        }
    }
    for(i=0;i<row1;i++)
    {
        for(j=0;j<col2;j++)
        {
            mat3[i][j]=0;
            for(k=0;k<col1;k++)
mat3[i][j]=mat3[i][j]+mat1[i][k]*mat2[k][j];
        }
    }
    printf("Element of first matrix is:\n");
    for(i=0;i<row1;i++)
    {
        cont.....

```

```
for(j=0;j<col1;j++)
{
printf("%d\t",mat1[i][j]);
}
printf("\n");
}
printf("Element of the second matrix is:\n");
for(i=0;i<row2;i++)
{
for(j=0;j<col2;j++)
{
printf("%d",mat2[i][j]);
}
printf("\n");
}
```

```
printf("Multiplication of two matrix is:\n");  
for(i=0;i<row1;i++)  
{  
for(j=0;j<col1;j++)  
{  
printf("%d\t",mat3[i][j]);  
}  
printf("\n");  
}  
}
```

Arrays and String

- Strings are array of character .i.e. they are character arrange one after another in memory. Thus, character array is also known as string. Strings is used in c for manipulating text such as word or sentence. A string is always terminated by a null character (\0).

//wap to illustrate string initilation.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char name[]="Shyam Prasad Sharma";
int i=0;
printf("\n The name in discrit form:\n");
while(name[i]!='\0')
{
printf("\t%c",name[i]);
i++;
}
getch();
clrscr();
}
```

//wap to illustrate string initilation.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char name[10];
int i;
printf("Enter the name of 5 student");
for(i=0;i<5;i++)
scanf("%s",&name[i]);
printf("\nNames are:\n");
for(i=0;i<5;i++)
printf("%s\t",name[i]);
getch();
clrscr();
}
```


String handling functions

strlen()

This function returns an integer which denotes the length of string passing. The length of string is the number of characters present in it, excluding the terminating null character.

Syntax:-

```
Integer_varriable= strlen(string);
```

Example:-

//wap to illustrate string initilation.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
char name[10];
int len;
printf("Enter your name ");
gets(name);
len=strlen(name);
printf("\n the number your name is :\t%d",len);
getch();
clrscr();
}
```

strcpy()

The `strcpy()` function copies one string to another. The function accept two string as parameter and copies second string character by character in to the frist one upto and including the null character of the second string . it's syntax is : `strcpy(distination_string, source_string);`

//wap to copy one string to another.

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

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

```
#include<string.h>
```

```
void main()
```

```
{
```

```
char name[]="Saroj Bhatta",s[15];
```

```
strcpy(s,name);
```

```
printf("The value in two strings :\n s=%s",s,name);
```

```
getch();
```

```
clrscr();
```

```
}
```

strcat()

This function concatenates two strings .i.e. it appends one string at the end of another. This function accepts two string as parameter and sorts the contents of the second string at the end of the first. Syntax:-

```
Strcat(string1,string2);
```

//wap to using strcat() function.

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

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

```
#include<string.h>
```

```
void main()
```

```
{
```

```
char fname[20]="Saroj",lname[]=" Bhatta";
```

```
strcat(fname,lname);
```

```
printf("The full name is:\n%s",fname);
```

```
getch();
```

```
clrscr();
```

```
}
```

strcmp()

This function compares two strings to find out whether they are same or different. This function is useful for constructing and searching strings as arranged in dictionary. This function accept two strings as parameters and returns an integer whose value is

- ✓ Less than 0 if the frist string is less than the second.
- ✓ Equals to 0 if the both are same.
- ✓ Grater than 0 if the first string is grater then second

The two string is compared character by character until there is mismatch or end of one string is reached. It's syntax is:-

```
Integer_variable= strcmp(string1,string2);
```

//wap to illustrate the use of strcmp() function.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char name[20],lname[20];
int diff;
printf("\n Enter the frist string\t");
gets(name);
printf("\nEnter the second string\t");
gets(lname);
diff=strcmp(name,lname);
if(diff>0)
printf("%s is grater then %s by value %d",name,lname,diff);
else if(diff<0)
printf("%s is grater than %s by value %d",lname,name,diff);
else
printf("%s is same as %s",name,lname);
getch();
clrscr();
}
```


strrev()

This function used to reverse all characters in a string except null character at the end of string . syntax:- strrev(string);

//wap to illustrate the use of strrev() function.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char name[20]="Shyam Sharma",lname[20];
strcpy(lname,name);
strrev(name);
printf("The reverse string of original string %s is %s",lname,name);
getch();
clrscr();
}
```

POINTER

Pointer:

A pointer is a variable which contains the memory(locations)of the another variable and is declared as the pointer type. As an example if one variable is a data type and second variable is the pointer type which points to the first variables type, then the contents of the second variable is the address of the first variable.

Here, the second variable contains the address of the first variable.5 is the value at address2000 which is the address of the first variable.

Uses of pointer

- Pointer is used for saving memory space.
- Use of pointer assigns the memory space and also releases it.
- With pointer data manipulation is done with address, so the addition time is faster.
- The two-dimensional and multi-dimensional array representation easy with pointer
- Pointer concept is used with data structure such as linked list

The & and * operator

The ‘&’ is the address operator, it represents the address of variable.

Example:

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

```
Void main()
```

```
{
```

```
int a=5;
```

```
printf("value of a=%d\n",a);
```

```
printf("Address of a=%u\n",&a);
```

```
}
```

The %u is used for obtaining the address.

The * operator is the value at address operator. It represent the value at the specified address.

Example:

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

```
void main()
```

```
{
```

```
int a=5;
```

```
printf("value of a=%d\n",a);
```

```
printf("address of a=%u\n",&a);
```

```
printf("value at address %u=%d\n",&a,*(&a));
```

```
}
```

Declaration of pointer: Let us see, how we use the pointer in expression. The & operator represents the address of the variable. If we want to give the address of the variable to another variable then we can write.

When a pointer variable is declared then an asterisk (*) symbol should precede the variable name. Here b is the pointer type variable, which points to the variable a. Hence it must be declared as-

```
int a=5;
```

```
int*b;
```

Similarly,

```
char*p;
```

```
float*q;
```

Example:-

```
#include<stdio.h>

void main()
{
int a=5;
int*b;
b=&a;
printf("value of a=%d\n",a);
printf("value of a=%d\n",*(&a));
printf("value of a=%d\n",*b);
printf("Address of a=%u \n",&a);
printf("Address of a=%u \n",b);
printf("Address of b=%u \n",&a);
printf("Value of b=address of a=%u \n",b);
}
```

Pointer to pointer: We know that pointer is a variable that contains the address of the another variable. Similarly, another pointer variable can store the address of this pointer variable. Hence we can say this is a pointer to pointer variable .

/* Program to understand pointer to pointer variable*/

```
#include<stdio.h>
void main()
{
    int a=5;
    int*b;
    int**c;
    b=&a;
    c=&b;
    printf("value of a=%d\n",a)
    printf("value of a=%d\n",*(&a));
    printf("value of a=%u \n",*b);
    printf("value of a=%u \n",**c);
}
```


***Pointer Arithmetic:** Arithmetic operation can also be done with pointer variable. Postfix, prefix increment and decrement operation are also possible with pointer.

Example:

```
/* program to understand the postfix , prefix increment and decrement is the pointer variable */  
#include<stdio.h>  
void main()  
{  
    int a=5;  
    int*b;  
    b=&a;  
    printf("address of a=%u\n",b);  
    printf("address of a=%u\n",++b);  
    printf("address of a=%u\n",b++);  
    printf("address of a=%u\n",--b);  
    printf("address of a=%u\n", b--);  
}
```

Pointer and function:

The arguments or parameters to the Function are passed in two ways:

- *Call by value
- *Call by reference

In call by value the values of the variables are passed. In this value of variable are not affected by changing the value of the formed parameter.

/*program to explain call by value */

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

```
void main()
```

```
{
```

```
int a=5;
```

```
int b=8;
```

```
printf("Before calling the function a and b are %d %d\n",a,b);
```

```
value(a,b);
```

```
printf("Before calling the function a and b are%d%d\n",a,b);
```

```
value(a,b);
```

```
printf("After calling the a and b are %d%d\n",a,b);
```

```
value(a,b);
```

```
}
```

```
value(p,q)
```

```
{
```

```
int p,q;
```

```
{
```

```
P++;
```

```
q++;
```

```
printf("In function changes are%d%d\n",p,q);
```

```
}
```

(2).Call by reference:-

In call by reference the address of the variable are passed .In this value of variable are affected by changing the value of the format parameter.

*/*Program to explain call by reference*/*

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

```
void main()
```

```
{
```

```
int a=5;
```

```
int b=8;
```

```
printf("Before calling the function a and b are %d%d\n",a,b);
```

```
ref(a,b);
```

```
printf("After calling the function a and b are %d%d\n",a,b);
```

```
}
```

```
ref(p,q)
```

```
int*p,*q;
```

```
{
```

```
(*p)++;
```

```
(*q)++;
```

```
printf("In function changes are%d%d\n",*p,*q);
```

```
}
```

/*Program to interchange the value of variable from cell by reference*/

```
    #include<stdio.h>
    void main()
    {
        int a=5;
        int b=8;
        printf("Before swapping a=%d,b=%d\n",a,b);
        swap(&a,&b);
        printf("After calling swap function a=%d, b=%d\n",a,b,);
    }
    swap(p,q)
    int*p,*q;
    {
        int temp;
        temp=*p;
        *p=*q;
        *q=temp;
        printf("In swapping function p=%d,q=%d\n",*p,*q);
    }
```

Pointer and Array: Array is a collection of similar data type elements. When we declare an array then consecutive memory locations are allocated to the array elements. The base address of array is the address of the 0th element of the array.

Example:

```
int arr[4]={5,10,15,20}
```

Here arr[4] means that array has 4 elements and is of int data type.

Arr[6]	arr[1]	arr[2]	arr[3]
--------	--------	--------	--------

2000	2002	2004	2006
------	------	------	------

```
Int arr[4]={5,10,15,20}
```

```
Int*a;
```

```
a=arr;
```

/* Use pointer to print the value and address of array elements */

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

```
void main()
```

```
{
```

```
int arr[4]={s,10,15,20},a;
```

```
int*b;
```

```
b=arr;
```

```
for(a=0;a<4;a++)
```

```
{
```

```
printf("value of arr[%d]=%d\n",a,*b);
```

```
printf("value of arr[%d]=%u \n",a,b);
```

```
b=b+1;
```

```
}
```

Pointer and string: A string is a collection of characters that are stored in the character array. Every string is terminated with the '0\' (null character). It is automatically inserted at the end of the string.

/* Program to print the character of any string and also address of each character */

```
#include<stdio.h>
void main()
{
int J;
char arr[]="Vijay";
int J;
for(J=0; arr[j]!='\0'; J++)
{
printf("address=%u\t",&arr[J]);
printf("address=%c\n",&arr[J]);
}
}
```


Dynamic memory allocation:-

In an array, it is must to declare the size of the array. There are two possibilities for declaring the array size. The first case is ,the records that are stored is less than the size of the array, second case, we want to store more records than size of the array. In first case, there is a wastage of memory. In second case we can't store more records that the size of the array. The C language has the facility to allocate memory at the time of execution . The process of allocating memory at the time of execution is called as dynamic memory allocation.

- The allocation and releasing of this memory space can be done with the use of some built in function which are find in alloc.h file.

Let us take these function in details.

1.Size of (): Size of is an unary operator. This operator gives the size of its argument in the term of byte. The argument can be a variable, array or any data type. This operator also gives the size of any structure.

Syntax;

size of (int)

This gives the bytes occupied by the in data type, that is 2

/*program to understand the size of operator */

```
#include<stdio.h>
void main()
struct{
char name[10];
int age;
float sal;
}
sec;
int arr[10];
printf("size of structure=%d",size of (rec));
printf("size of int=%d",size of (int));
printf("size of array =%d",size of (arr));
}
```

2. Mallo(); This function is used to allocate memory space. The mallo() function reserved a memory space of specified size and gives the starting address to pointer variable.

This can be written as-

```
ptr=(datatype,*)mallo()(specified size);
```

Here data types says the type of pointer and specified size is the size which is required to reserve in memory .An example:

```
ptr=(int*) malloc(10);
```

This allocates 10 bytes of memory space to the pointer ptr of type int and base address is stored in the pointer ptr.

```
ptr=(int*) mallo(10* size of(int));
```

This allocates the memory space 10 times, that is hold an int datatype.

/*Program to enter the 5member and print them*/

```
#include<stdio.h>
void main()
{
int J;
int*a;
a=(int*)malloc(5*size of (int));
for(J=0;J<5;J++)
{
printf("number%d=",J+1);
scanf("%d",(a+J));
}
for(J=0;j<5;J++)
printf("%d\n",*(a+J));
}
```

3.Calloc(): The calloc() function is used to allocate multiple block of memory .This has two arguments .

As Example:

```
ptr=(int*) calloc(5,2);
```

This allocates 5 block of memory each block contains 2 bytes of memory and the starting address is stored in the pointer variable ptr which is of type int.

The calloc() function is generally used for allocating the memory space for array and structure .

As example:

```
struct record{  
    char name[10]  
    int age;  
    float sal;  
    }
```

```
int tot-rec=100;
```

```
ptr=(record*)calloc(tot-rec,size of (record));
```

Structure and Union

Structure:

- Structure is a collection of data item. The data item can be different type, some can be int, some can be float, some can be char and so on. The data item of structure is called member of the structure.
- In other words we can say that heterogeneous data types can be grouped to form a structure. In some languages structure is known as record. The different between array and structure is the element of an array has the same type while the element of structure can be of different type. Another different is that each element of an array is referred to by its position while each element of structure has a unique name.

We can define the structure as-

```
Struct tag{  
  Member 1;  
  Member 2;  
  .....  
  .....  
  Member n;  
}
```

Here, struct is a keyword for structure, we can also defined as-

```
Struct{  
  Member 1;  
  Member 2;  
  .....  
  .....  
  Member n;  
}  
Var;
```

Here var is the structure variable.

/*Program to accept name, age and address of any person and display it*/

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

```
void main()
```

```
{
```

```
struct{
```

```
char name[20];
```

```
int age;
```

```
char address[20];
```

```
}rec;
```

```
printf("Enter the name");
```

```
scanf("%s\n",rec.name);
```

```
printf("Enter the age");
```

```
scanf("%d",rec.age);
```

```
printf("Enter the address");
```

```
scanf("%s",rec.address);
```

```
printf("Name:%s\n",rec.name);
```

```
printf("Age:%d\n",rec.age);
```

```
printf("Address:%s\n",rec.address);
```

```
}
```

/* Same program for understanding how we define any structure to another name and how it is used*/

```
#include<stdio.h>
void main()
{
    Struct rec{
        char name[20];
        int age;
        address[20];
    }
    Struct rec person;
    printf("Enter the name");
    scanf("%s",person. name);
    printf("Enter the age:");
    scanf("%d",person .age);
    printf("Enter the address");
    scanf("%s",person.address);
    printf("Name:%s\n",person.name);
    printf("Age:%d\n", person.age);
    printf("Address:%s\n", person.address);
}
```

*Initialize value to the structure:

We can initialize the value to the members of structure.

```
struct{  
char name[20];  
int age;  
char address[20]  
}  
sec={"Bijay",19,"Bardaghat"};
```

Another type of initialization value to structure.

```
struct rec{  
char name[20];  
int age;  
char address[20];  
}  
struct rec person={"Bijay",19,"Bardaghat"};
```

Array of Structure: As we know array is a collection of same datatype of elements. Similarly we can declare the array of structure where every element of array is of structure type. Array of structure can be declared as:-

```
struct rec{  
char name[20];  
int age;  
char address[20];  
}  
Person[10];
```

```
/* Program to accept record of 10 person which has name,  
age and address and also display them */
```

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

```
void main(){
```

```
    struct rec{
```

```
        char name[20];
```

```
        int age;
```

```
        char address[20];
```

```
    }
```

```
    Person[10];
```

```
    int J;
```

```
    for(J=0;J<10;J++){
```

```
printf("Enter the record%d\n",j+1);
printf("Name;");
scanf("%s",person[J]name);
printf("Age:");
scanf("%d",&person[J]age);
printf("Address:");
scanf("%s",person[J].address);
}
printf("Name \t Age\t Address\n");
for(J=0;J<10;J++){
printf("%s\t",person[J].name);
printf("%d\t",person[J].age);
printf("%s\t",person[J].address);
}}
```

Union: Union is same as structure. As the structure contains members of different data types. In the structure each member has its own memory location whereas members of union has same memory location whereas members of union has same memory location we can assign values to only one member at a time so assigning value to another member that time has no meaning .

Syntax:

```
Union union-name{  
    Member1;  
    Member2;  
}
```

Here union is the keyword, it is necessary for declaration of union .

- The declaration of union variable is same as structure.

```
Union union-name {
```

```
    Member 1;
```

```
    Member 2;
```

```
    .....
```

```
} variable name;
```

Example

```
Union emp {
```

```
    Int age;
```

```
    Char code[4]
```

```
}
```

In union emp both int age and char code[4] share the same memory location. Here code occupy 4 byte and age use only 2 bytes. We can access the union member same as structure if we access directly then we use the dot(.) operator if we access through pointer then we use arrow() operator.


```
/*Program for accessing union member */  
#include<stdio.h>  
void main()  
{  
    union result{  
        int marks  
        char grade;  
    } res;  
    printf("size of union:%d\n",sizeof(res));  
    res.marks=90;  
    printf("Marks =%d,grade=%c\n",res.marks,res.grade);  
    res.grade='A';  
    printf("marks=%d,grade:%c\n",res.marks,res.grade);  
}
```

File handling function in c

A **file** is a place on the disk where a group of related data is stored. The data file allows us to store information permanently and to access and alter that information whenever necessary. Programming language C has various library functions for creating and processing data files. Mainly, there are two types of data files:

1. High level (standard or stream oriented) files.
2. Low level (system oriented) files.

In high level data files, the available library functions do their own buffer management where as the programmer should do it explicitly in case of lower level files. The standard data files are again subdivided into text files and binary files. The text files consist of consecutive characters and these characters can be interpreted as individual data item. The binary files organize data into blocks containing contiguous bytes of information. For each, binary and text files, there are a number of formatted and unformatted library functions in C.

Opening and closing a data file:

Before a program can write to a file or read from a file, the program must open it. Opening a file established a link between the program and the operating system. This provides the operating system, the name of the file and the mode in which the file is to be opened.

While working with high level data file, we need buffer area where information is stored temporarily in the course of transferring data between computer memory and data file. The process of establishing a connection between the program and file is called **opening the file**.

A file pointer is a pointer to a structure of type FILE. Whenever a file is opened, a structure of type FILE is associated with it, and a file pointer that points to this structure identifies this file. The function `fopen()` is used to open a file.

The buffer area is established by

```
FILE *ptr_variable;
```

And file is opened by using following syntax

```
ptr_variable = fopen( file_name, file_mode);
```

where `fopen()` function takes two strings as arguments, the first one is the name of the file to be opened and the second one is `file_mode` that decides which operations (read, write, append, etc) are to be performed on the file. On success, `fopen()` returns a pointer of type FILE and on error it returns NULL.

For example:

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

```
fp1 = fopen ("myfile.txt", "w");
```

```
fp2 = fopen ("yourfile.dat", "r");
```

In other word, it specifies the purpose of opening a file. They are:

1. “w” (write):

If the file doesn't exist then this mode creates a new file for writing, and if the file already exists, then the previous data is erased and the new data entered is written to the file.

2. “a” (append):

If the file doesn't exist then this mode creates a new file, and if the file already exists then the new data entered is appended at the end of existing data. In this mode, the data existing in the file is not erased as in “w” mode.

3. “r” (read):

This mode is used for opening an existing file for reading purpose only. The file to be opened must exist and the previous data of file is not erased.

4. “w+” (write + read):

This mode is same as “w” mode but in this mode we can also read and modify the data. If the file doesn't exist then a new file is created and if the file exists then previous data is erased.

5. “r+” (read + write):

This mode is same as “r” mode but in this mode we can also write and modify existing data. The file to be opened must exist and the previous data of file is not erased. Since we can add new data and modify existing data so this mode is also called update mode.

Closing a file: The file which are open from the fopen c function must be closed at the end of the program. This is written as:-

```
fclose(fp);
```

Structure of the file program

```
void main()
```

```
FILE *fp;
```

```
fp=fopen("file name","mode");
```

```
.....
```

```
.....
```

```
fclose(fp);
```

```
}
```

file I/O:-

fprintf(): This function writes the data into the file, so it has one more parameter is the file pointer.

Syntax:-

fprintf(fp,"control character", variable names);

fscanf(): This function is same as the scanf() function but this reads the data from the file, so this has one more parameter that is the file pointer.

Syntax:

fscanf(fp,"control character", variable names);

eof():- The macro eof() is used for detecting whether the file pointer is at the end of file or not. It returns nonzero if the file pointer is at the end of file, otherwise it return zero.

//Writing a file

```
#include<stdio.h>
#include<conio.h>
void main
{
int age;
char name;
FILE *fp;
fp=fopen("info.dat","w");
printf("Enter name:");
gets(name);
printf("Enter age:");
scanf("%d",&age);
fprintf("%s%d",name,age);
fclose(fp);
return 0;
}
```

//Reading a file

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

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

```
void main()
```

```
{
```

```
int age;
```

```
char *name;
```

```
FILE *fp;
```

```
fp=fopen("info.dat","r");
```

```
fscanf(fp,"%s%d",name,&age);
```

```
printf("Your name:%s",name);
```

```
printf("Your age:%d",age);
```

```
fclose(fp);
```

```
return 0;
```

```
}
```

//Write a program to read data from the keyboard, write it to a file called INFO, again read the same data from INFO file, and display it on the screen.

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

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

```
void main()
```

```
{
```

```
FILE *fp;
```

```
char ch;
```

```
printf("\nData input\n\n\n");
```

```
fp=fopen("INFO.TXT","w");
```

```
while(ch=getchar()!=EOF)
```

```
printf("%c",ch);
```

```
fclose(fp);
```

```
//data output
```

```
fp=fopen("INFO.TXT","r");
```

```
while(ch=getc(fp)!=EOF)
```

```
printf("%c",ch);
```

```
fclose(fp);
```

```
getch();
```

```
}
```

//Write a program to read name, address and telephone number 10 number of students write them //on STUD.DAT file and display them reading from file.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char name[30], add[20], long int phone;
FILE *fp;
fp=fopen("STUD.DAT",w+);
for(i=0;i<10;i++)
{
printf("Enter name:\n");
scanf("%s",name[i]);
printf("Enter address:\n");
scanf("%s",add[i]);
```

```
printf("Enter phone:\n");
scanf("%ld",&phone[i]);
fprintf(fp,"%s%s%ld",name,add,phone);
}
rewind(fp);
for(i=0;i<10;i++)
{
fscanf(fp,"%s%s%ld",name,add,&phone);
printf("Name=%s\tAddress=%s\tPhone=%ld",name[i],add[i], phone[i]);
}
fclose(fp);
getch();
}
```

```
#include<stdio.h>
#include<conio.h>
//#include<lib.h>
void main()
{
    struct student
    {
        char name[30];
        int roll;
        float marks;
    };
    struct student s[3],st[3];
    int i;
    FILE *fptr;
```

```
clrscr();
fptr=fopen("student.text","w+b");
for(i=0;i<3;i++)
{
printf("\n\nEnter information of student no %d\n",i+1);
printf("Name:\t");
scanf("%s",s[i].name);
printf("\nRoll:\t");
scanf("%d",&s[i].roll);
printf("\nMarks:\t");
scanf("%f",&s[i].marks);
}
printf("\n\nWriting Information to File.....\n");
fwrite(&s,sizeof(s),3,fptr);
rewind(fptr);
```



```
printf("\nReading same content form file.....\n");
fread(&st,sizeof(st),3,fptr);
printf("Student Name\t Roll\t Marks");
printf("\n.....\n");
for(i=0;i<3;i++)
printf("%s\t\t%d\t%.2f\n",st[i].name,st[i].roll,st[i].marks);
fclose(fptr);
getch();
}
```

A program to print ascii table

```
#include<stdio.h>
int main()
{
int i;
for(i=0;i<=255;i++)
printf("ASCII value of character %c: %d\n",i,i);
return 0;
}
```

```
/* C program to display character from A to Z
   using loops. */
#include <stdio.h>

int main()
{
    char c; for(c='A'; c<='Z'; ++c)
        printf("%c ",c);
    return 0;
}
```