

C Language

Computer:

It is an electronic device, It has memory and it performs arithmetic and logical operations.

Computer full form

Commonly Operating Machine Performing Users
Technologies and Educational Research.

Input:

The data entering into computer is known as input.

Output:

The resultant information obtained by the computer is known as output.

Program:

A sequence of instructions that can be executed by the computer to solve the given problem is known as program.

Software:

A set of programs to operate and controls the operation of the computer is known as software.

these are 2 types.

1. System software.
2. Application software.

System Software:

It is used to manages system resources.

Eg: Operating System.

Operating system:

It is an interface between user and the computer. In other words operating system is a complex set of programs which manages the resources of a computer. Resources include input, output, processor, memory, etc. So it is called as Resource Manager.

Eg:

Windows98, WindowsXp, Windows7, Windows8, Windows10, Unix, Linux , etc.

Application Software:

These are Used to develop the applications.
It is again of 2 types.

- 1 Languages
- 2 Packages.

Language:

It consists a set of executable instructions. Using these instructions we can communicate with the computer and get the required results.

Eg: C, C++, Java, etc.

Package:

It is designed by any other languages with limited resources.

Eg: MS -Office, Account Package, DTP, etc.

Hardware:

All the physical components or units which are connecting to the computer circuit is known as Hardware.

Numbering and coding systems

Numbering system:

Human beings use base 10 (decimal) arithmetic.

There are 10 distinct symbols 0, 1, 2, ...,9.

Computers use base 2 (binary) system. There are only 0 and 1. These two binary digits are commonly referred to as bits.

Conversion of decimal to binary:

Step1: Divide the decimal number by 2 repeatedly

Step2: Keep track of the remainders

Step3: Continue this process until the quotient becomes zero

Step4: Write the remainders in reverse order to obtain The binary number

Eg. Convert 25_{10} to binary

Quotient		Remainder	
25/2 =	12	1	LSB (least significant bit)
12/2 =	6	0	
6/2 =	3	0	
3/2 =	1	1	
1/2 =	0	1	MSB (most significant bit)

Therefore $25_{10} = 11001_2$

Conversion of binary to decimal

Step1: Know the weight of each bit in a binary number

Step2: Add them together to get its decimal equivalent

Eg: Convert 11001_2 to decimal

Weight: 2^4 2^3 2^2 2^1 2^0

Digits : 1 1 0 0 1

Sum : $16 + 8 + 0 + 0 + 1 = 25_{10}$

ASCII character Set

ASCII - American Standard Code for Information Interchange

There are 256 distinct ASCII characters are used by the micro computers. These values range from 0 to 255. These can be grouped as follows.

Character Type	No. of Characters

Capital Letters (A to Z)	26
Small Letters (a to z)	26
Digits (0 to 9)	10
Special Characters	32

Control Characters	34
Graphic Characters	128

Total	256

Out of 256, the first 128 are called as ASCII character set and the next 128 are called as extended ASCII character set. Each and every character has unique appearance.

Eg:

A to Z	65 to 90
a to z	97 to 122
0 to 9	48 to 57
Esc	27
Backspace	8
Enter	13
SpaceBar	32
Tab	9
etc.	

Classification of programming languages:-

- Programming languages are classifies into
2 types
- 1.Low level languages
 - 2.High level languages

Low level languages:

It is also known as Assembly language and was designed in the beginning. It has some simple instructions. These instructions are not binary codes, but the computer can understand only the machine language, which is in binary format. Hence a converter or translator is used to translate the low level language instructions into machine language. This translator is called as "***Assembler***".

High level languages:

These are more English like languages and hence the programmers found them very easy to learn. To convert high level language instructions into machine language ***compilers and interpreters*** are used.

Translators:

These are used to convert low or high level language instructions into machine language with the help of ASCII character set. There are 3 types of translators for languages.

1) *Assembler* :

It is used to convert low level

language instructions into machine language.

2) *Compiler:*

It is used to convert high level language instructions into machine language. It checks for the errors in the entire program and converts the program into machine language.

3) *Interpreter:*

It is also used to convert high level language instructions into machine language, But It checks for errors by statement wise and converts into machine language.

Debugging :

The process of correcting errors in the program is called as debugging.

Various Steps involved in program development:

There are 7 steps involved in program

development.

1) Problem definition:

Defining a problem is nothing but understanding the problem. It involves 3 specifications regarding a problem solution.

- 1 . Input specification
- 2 . Output specification
- 3 . Processing

2)Analysis and design:

Before going to make final solution for the problem, the problem must be analyzed. Outline solution is prepared for the simple problems. In case of complex problems, the main problem is divided into sub problems called as ***modules***. Each module can be handled and solved independently.

3.Algorithm:

A step by step procedure to solve the given problem is called as algorithm.

4.Flow chart:

A symbolic or graphical representation of given algorithm is called as flow chart.

5.Coding and implementation:

Coding is a process of converting the algorithmic solution or flow chart into a program of selected computer programming language. Before selecting a programming language we must follow the following 3 considerations.

- a) Nature of program.
- b) Programming language available on Computer system.
- c) Limitations of the computer.

6.Debugging and testing:

Before loading the program into computer, we must locate and correct all the errors. The process of correcting errors in a program is called as debugging.

There are 3 types of errors that generally occur in a program.

- a) Syntax errors
- b) Runtime errors
- c) Logical errors

It is very important to test the program written to achieve a specific task. Testing involves running the program with known data of which the results are known. As the results are known, the results produced by the computer can be verified.

7.Documentation:

It is the most important aspect of programming. It is a continuous process to keep the copy of all the phases involved in a problem definition,.....,debugging and testing are parts of documentation. This phase involves to producing a written document for the user.

Algorithms And Flow Charts

Algorithm:

A step by step procedure to solve the given problem is called as algorithm.

Flowchart :

A symbolic or graphical representation of given algorithm is called as flow chart.

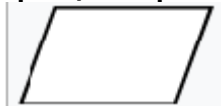
Common Shapes

The following are some of the commonly used shapes in flowcharts.

- 1) Oval shape is used to indicate start / stop of the procedure.



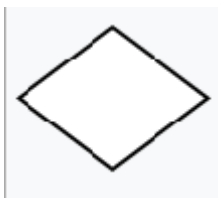
- 2) parallelograms are used to indicate the input/output operations



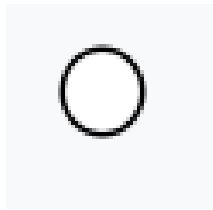
- 3) Rectangles are used to indicate the data transfer operations and arithmetic operations.



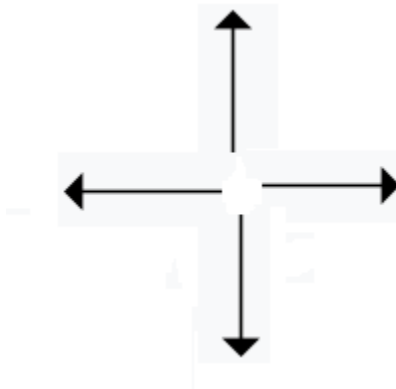
- 4) Rhombus is used to indicate the conditional statements.



- 5) Circles are used to connect different parts of a flow chart.



6) Arrows indicate the directions to be followed in a flow chart.



Write Algorithms and draw flow charts for the following problems

1) To find addition, subtraction, multiplication, division and modulus of given 2 numbers.

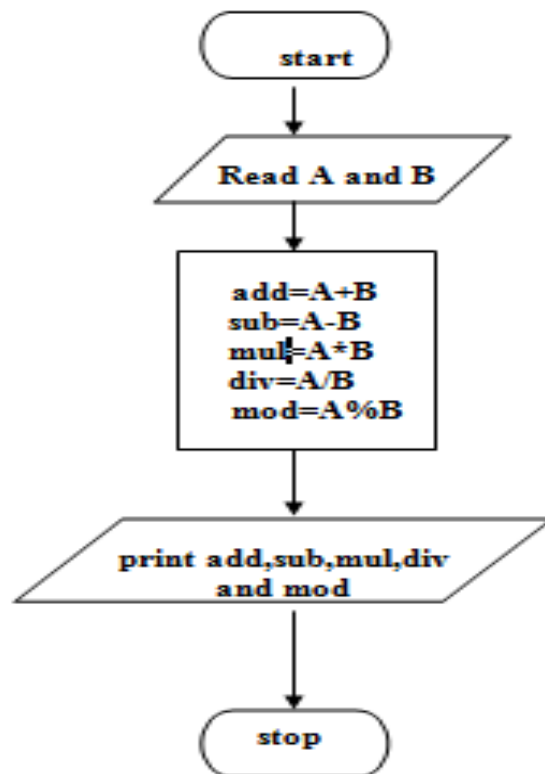
Steps:

1. start
2. Read Two Numbers A and B
3. $add = A + B$
4. $sub = A - B$
5. $mul = A * B$
6. $div = A / B$

7. $\text{mod} = A \% B$

8. print add, sub, mul, div and mod

9. stop.



2) To find maximum value of given 2 values.

Steps:

1. start

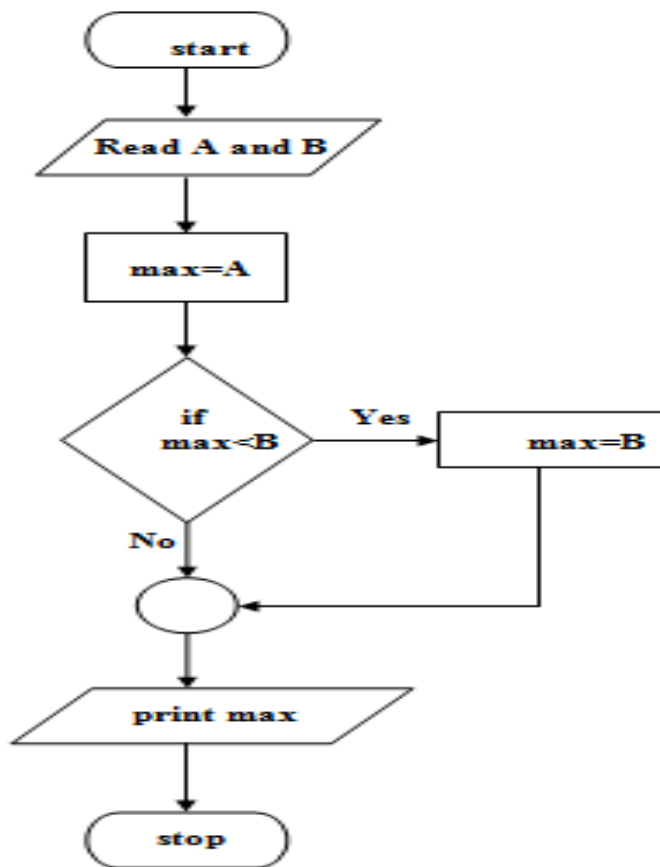
2. Read A and B

3. $\text{max} = A$

4. if ($\text{max} < B$) then $\text{max} = B$

5. print max

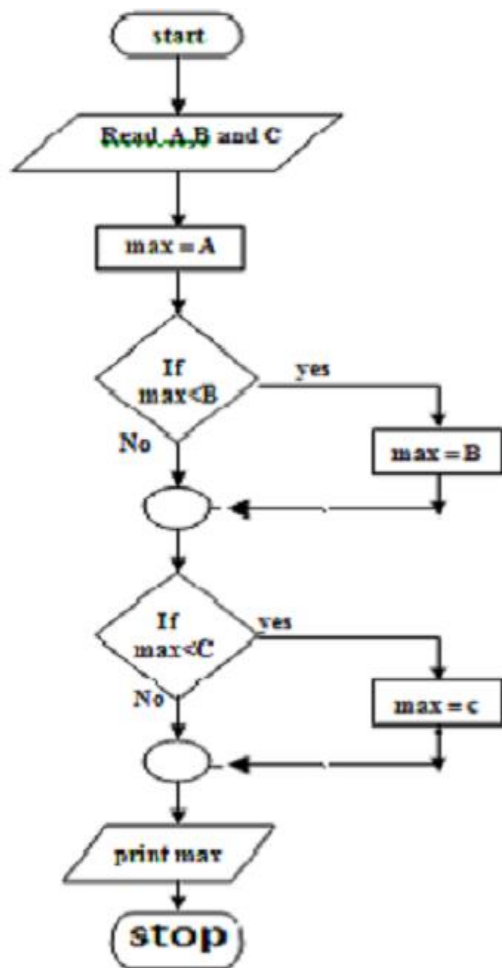
6. stop



3)To find maximum value of given 3 values.

Steps:

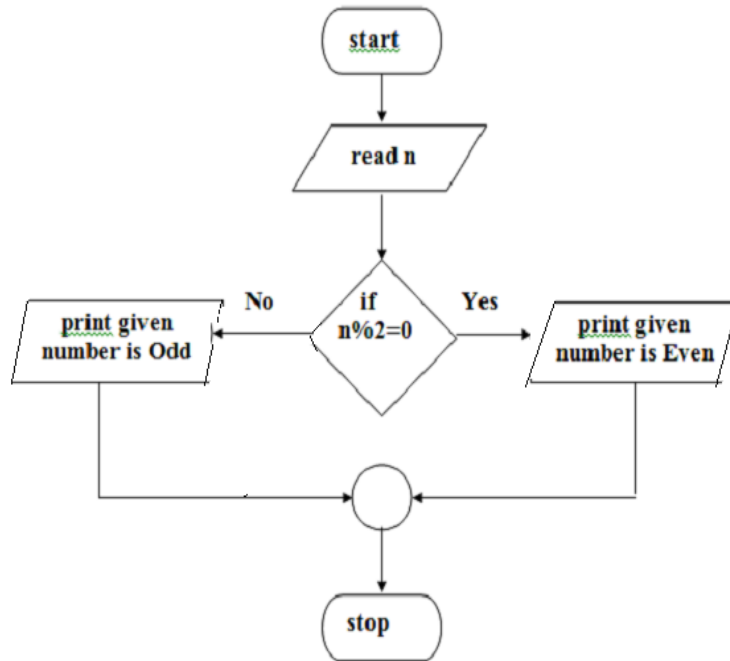
1. start
2. Read A, B and C
3. max=A
4. if(max < B) then max=B
5. if(max < C) then max=C
6. print max
7. stop



4) To check whether the given number is even or odd.

Steps:

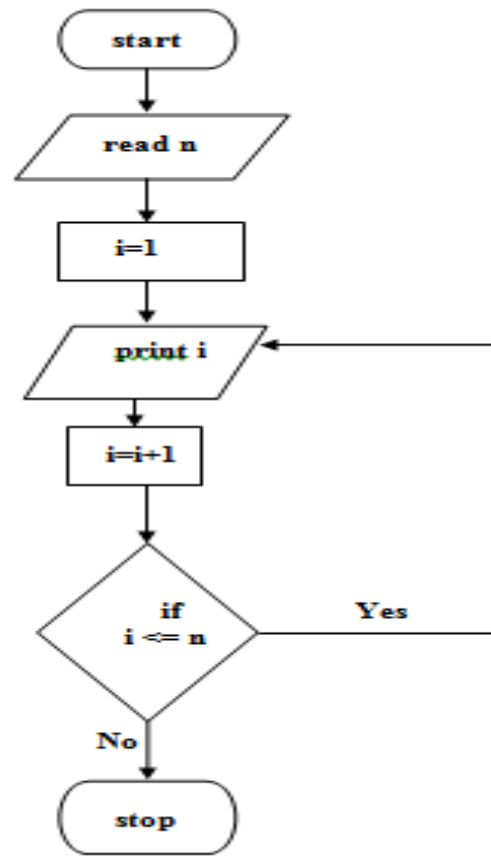
1. start
2. Read n
3. if($n \% 2 == 0$) then print "Given number is even"
else print "Given number is odd"
4. stop



5)To display natural numbers from 1 to given number.

Steps:

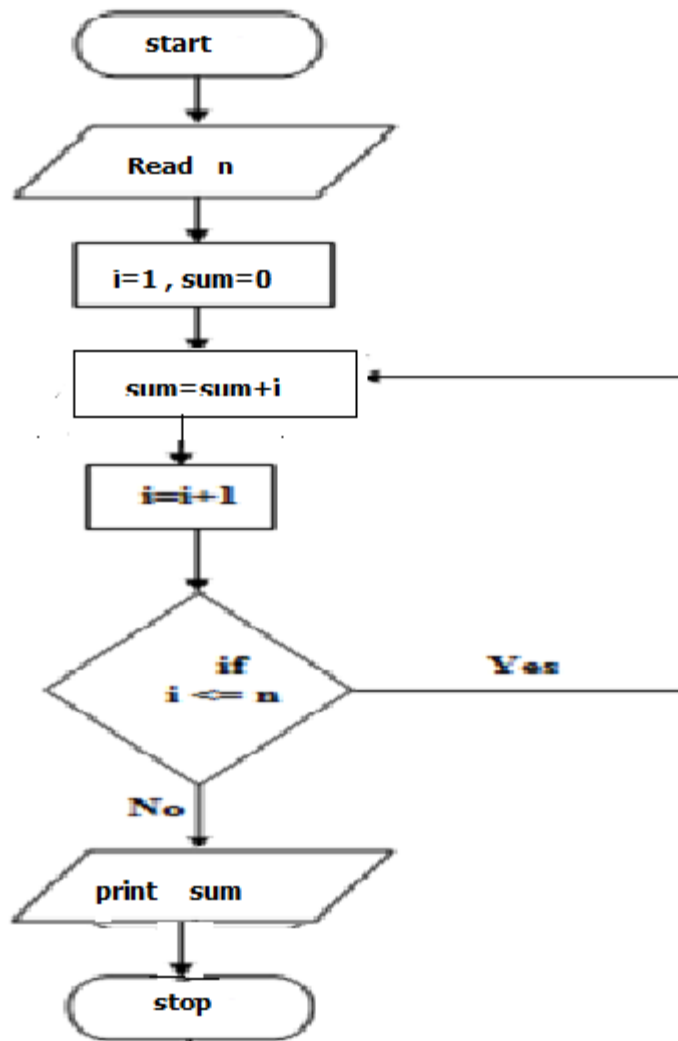
1. start
2. Read n
3. i=1
4. print i
5. i=i+1
6. if(i<=n) then goto step 4
7. stop



6)To calculate and display the sum of n natural numbers.

Steps:

1. start
2. Read n
3. sum=0,i=1
4. sum=sum+i
5. i=i+1
6. if(i<=n) goto step 4
7. print sum
8. stop



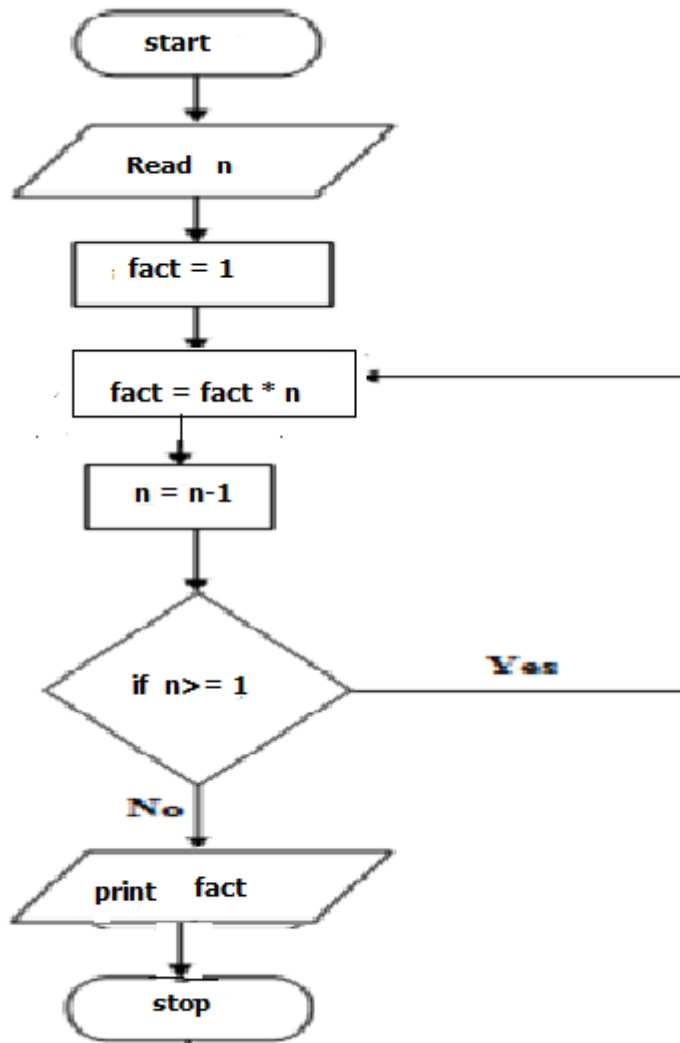
7)To find factorial of a given number.

Steps:

1. start
2. Read n
3. fact=1
4. fact=fact*n
5. n=n-1
6. if(n>=1) then goto step 4

7. print fact

8. stop



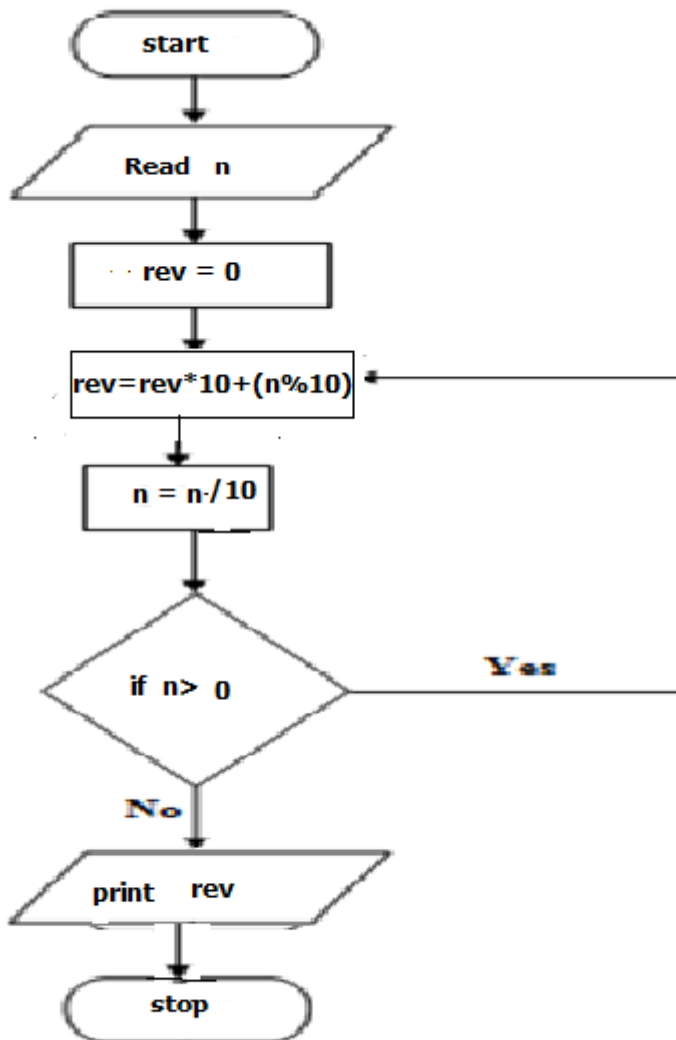
8) To find reverse number of a given number.

Steps:

1. start

2. Read n

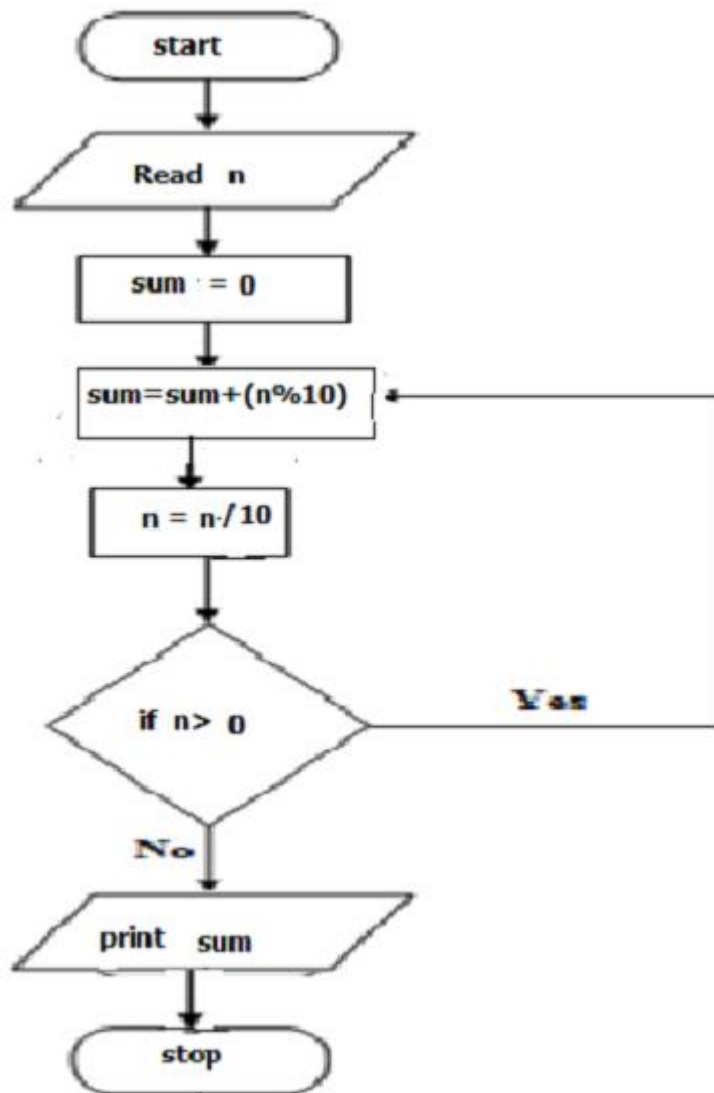
3. $rev = 0$
4. $rev = (rev * 10) + (n \% 10)$
5. $n = n / 10$
6. if $(n > 0)$ then goto step 4
7. print rev
8. stop



9) To Find Sum of digit in the given Number

Steps:

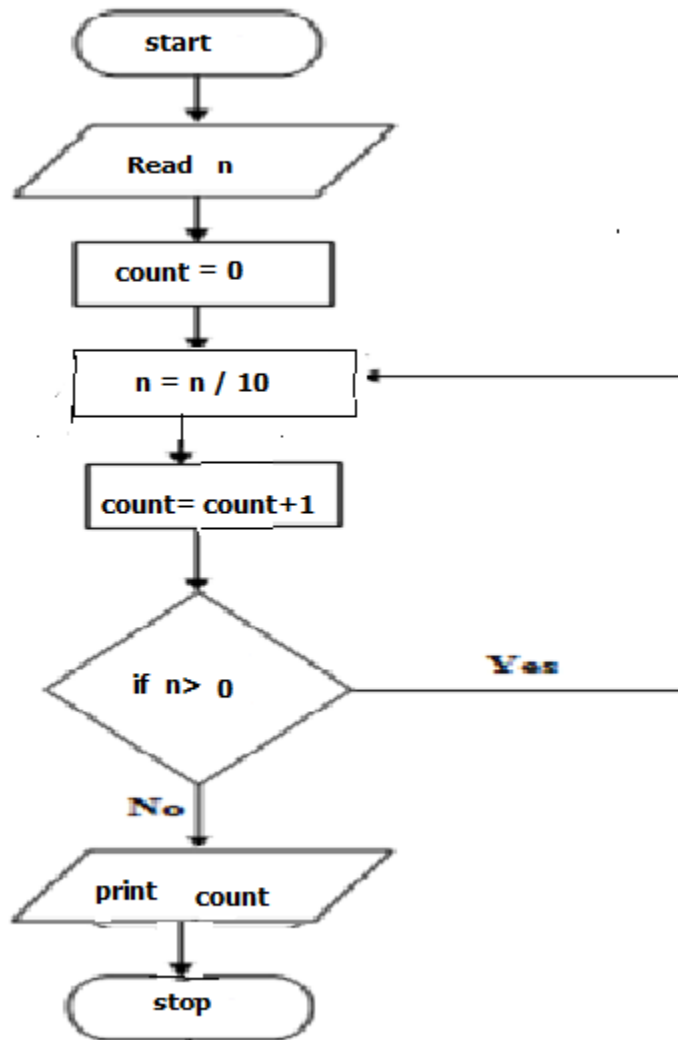
1. start
2. Read n
3. $sum=0$
4. $sum=sum+(n\%10)$
5. $n=n/10$
6. if($n>0$) then goto step4
7. print sum
8. stop



10) To Count No. of Digits in the given Number
Steps

1. start
2. Read n
3. count=0
4. $n = n / 10$
5. count=count+1
6. if $(n > 0)$ then goto srtp4

7. print count
8. stop



Introduction to C Language :

'C' is a computer programming language. It was designed by **Dennis Ritchie** in 1972 at AT &T(American Telephones and Telegraphs) BELL labs in USA.

It is the most popular general purpose programming

language. We can use the 'C' language to implement any type of applications. Mainly we are using C language to implement system softwares. These are compilers, editors, drivers, databases and operating systems.

History of C Language:

In 1960's COBOL was being used for commercial applications and FORTRAN is used for scientific and engineering applications. At this stage people started to develop a language which is suitable for all possible applications. Therefore an international committee was setup to develop such a language

"ALGOL 60" was released. It was not popular, because it seemed too general. To reduce this generality a new language CPL (combined programming language) was developed at Cambridge University. It has very less features. Then some other features were added to this language and a new language called BCPL (Basic combined programming language) developed by "Martin Richards" at Cambridge University. Then "B" language was developed by "Ken Thompson" at AT&T BELL labs.

Dennis Ritchie inherited the features of B and BCPL, added his own features and developed C language in 1972.

Features of 'C' Language:

1. C is a structured programming language with fundamental flow control construction.

2 . C is simple and versatile language.

3. Programs written in C are efficient and fast.

4. C has only 32 keywords.

5. C is highly portable programming language.

The programs written for one computer can be run on another with or without any modifications

6. C has rich set of operators.

7. C permits all data conversions and mixed mode operations

8. Dynamic memory allocation(DMA) is possible in C.

9. Extensive varieties of data types such as arrays, pointers, structures and unions are available in C.

10. C improves by itself. It has several predefined functions.

11. C easily manipulates bits, bytes and addresses.
12. Recursive function calls for algorithmic approach is possible in C.
13. Mainly we are using C language to implement system softwares. These are compilers ,editors, drivers ,databases and operating systems.
14. C compiler combines the capability of an assembly level language with the features of high level language. So it is called as middle level language.

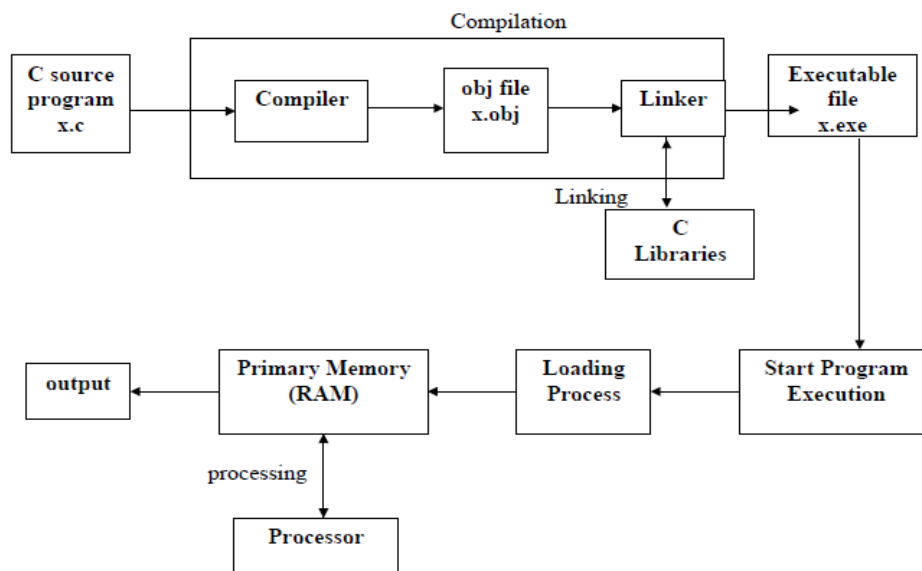
Note:

- 1 . C was basically designed for Unix operating system. 93% of instructions, which are written in C.
- 2 . C is case sensitive programming language. C statements are entered in lower case

letters only.

- 3 . C is function oriented programming language. Any C program contains one or more functions . Minimum One function is compulsory by the name called ***main***. Without main we can't execute a C program.
- 4 . Every C statement must be terminated by semicolon (;), except pre-processor statements and function definition.

Block diagram of execution of C program:



Once the program is completed, the program is feed into the computer using the compiler to produce equivalent machine language code. In C program compilation there are 2 mechanisms.

1. Compiler
2. Linker.

The Compiler receives the source file as input and converts that file into object file. Then the *Linker* receives the object file as its input and linking with C libraries. After linking it produces an executable file for the given code. After creation of executable file, then start the program execution and loads the information of the program into primary memory through 'Loading Process'. After loading the information the processor processing the information and gives output.

Basic structure of C program

[Document section]


Preprocessor section
(or)
Link section

[Global declaration section]

```

main( )
{
    [ Local declaration section ]
    Statements
}

```



Main function section

[Sub program section]
(include user defined functions)

Document section:

It consists a set of comment lines giving the name of the program, author name and some other details about the entire program.

Preprocessor Section (or) Link section:

It provides instructions to the compiler to link the functions from the 'C' library.

Global declaration Section:

The Variables that are used in more than one function are called as global variables and these are declared in global declaration section.

Main function section:

Every C program must have one function. i.e. main function. This section contains 2 parts.

1. Local declaration section.
2. Statements section.

The Local declaration section declares all the variables

used in statements. The statements part consists a sequence of executable statements. These 2 parts must appear between opening and closing curly braces. The program execution begins at the opening brace and ends at closing brace.

Sub programming section:

It contains all the user defined functions. This section may be placed before or after main function.

Comments:

Unexecutable lines in a program are called as comments. These lines are skipped by the compiler.

```
/*-----  
-----  
-----*/    Multi line comment.
```

```
//----- single line comment(C++ comment).
```

Preprocessor statements:

The preprocessor is a program that process the source code before it passes through the compiler.

#include:

It is preprocessor file inclusion directive and is used to include header files. It provides instructions to the compiler

to link the functions from the 'C' library.

Syntax:

```
#include "file_name"
```

(or)

```
#include <file_name>
```

When the file name is included within the double quotation marks, the search for the file is made first in the current directory and then the standard directories(TC(or)TurboC3). Otherwise when the file name is included within angular braces, the file is search only in standard directories.

stdio.h :- standard input and output
header file.

conio.h :- console input and output
header file.

console units: keyboard and monitor.

These 2 header files are commonly included in all C programs.

printf :

It is a function and is used to print data on the standard output device(monitor).

Syntax:

format string

/

```
int printf("control string" [,arg1,arg2,....argn] );
```

Eg: printf("Welcome to C Programming");

Program :

```
/* First Program */
```

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

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

```
void main( )
```

```
{
```

```
    printf("Welcome to C Programming");
```

```
}
```

If you want to work with Turbo c/c++ ,first install Turbo c/c++ software and then follow the following steps.

How to open a C editor(windows 7/8/10):

1. At the time of installation it creates Shortcut to C (Turboc++)on Desktop. Double click on that icon.

windows 7(or)8(or) 10

1. C : \TurboC3\Include

2. C : \TurboC3\Lib

3. -----

4. -----

3) Type C program , goto File menu → New

4) Save program , goto File menu → Save

5) Compile Program ,
goto Compile menu → Compile

6) Run the Program , goto Run menu → Run

7) See the output ,
goto window menu -> user screen

8) Exit from C editor , goto File menu → Quit

Shortcut Keys:

Open	:	F3
Save	:	F2
Close file	:	Alt + F3
Full Screen	:	F5
Compile	:	Alt + F9
Run	:	Ctrl + F9
Output	:	Alt + F5

Change to another file: F6
Help : Ctrl + F1
Tracing : F7
Quit : Alt + X

clrscr :

It is a Function and It clears text mode window.

syntax:

void clrscr();

getch:

It is a function and it gets a character from standard input device(KeyBoard), but it doesnot echo(print) to the screen.

Syntax : int getch();

Program :

```
/* Second Program */  
#include<stdio.h>  
#include<conio.h>
```

```

void main( )
{
    clrscr( );
    printf("BDPS COMPUTER EDUCATIONS");
    getch( );
}

```

Escape sequence characters :

C uses some backslash characters in output functions for formatting the output. These characters are called as Escape sequence characters.

In these characters it consists of two characters, But it is treated as a single character.

\n - new line

\t - horizontal tab(default 8 spaces)

\v - vertical tab(default 1 line)(It is Not Working in Turbo c/c++)

\b - back space

\a - alert(beep sound)

\r - Carriage return

\0 - null

\\" - double quotes

etc.

C Tokens

The smallest individual elements or units in a program

are called as Tokens. C has following tokens.

- 1 Identifiers
- 2 Keywords
- 3 Constants
- 4 Operators
- 5 Special characters

Identifiers :

Identifiers refer to the name of the variables, functions, arrays, etc. created by the programmer, using the combination of following characters.

- 1 . Alphabets : A to Z and a to z
- 2 . Digits : 0 to 9
- 3 . Underscore : _

Note :

- 1 . The first character of an identifier must be an alphabet or underscore, we cannot use digit.
- 2 . Default identifier length is 32 characters.

Keywords

Keywords are the words whose meaning has been already explained by the compiler. That means at the time of designing a language, some words are reserved to do specific tasks. Such words are called as keywords (or) reserved words. All C compilers support 32 keywords.

They are:

1. auto

2. break
3. case
4. char
5. const
6. continue
7. default
8. do
9. double
10. else
11. enum
12. extern
13. float
14. far
15. for
16. goto
17. if
18. int
19. long
20. register
21. return
22. short
23. signed
24. sizeof
25. static
26. struct
27. switch
28. typedef
29. union

30.unsigned

31.void

32.while

constants:

constants define fixed values, that donot change during the execution of a program. C supports the following constants.

- 1 Integer constants
- 2 Character constants
- 3 Real (or) floating constants
- 4 String constants

Operators:

Operator is a symbol which performs particular operation. C supports a rich set of operators. C operators can be classified into No of categories. They include arithmetic operators, logical operators, bitwise operators, etc.

Special characters:

All characters other than alphabets and digits are treated as special characters.

Eg: * , % , \$, { ,etc.

Data types in C

C language has some predefined set of data types to handle various kinds of data that we use in our program.

C data types can be classified into 3 categories.

- 1 Primary data types
- 2 Derived data types
- 3 User defined data types

Primary data types:

All C compiler supports 4 fundamental data types. Namely int, char, float and double.

int: It is positive, negative and whole values but not a decimal number.

Eg: 10, 20, 45, -30, 0 etc.

char: A single character can be treated as character data type and it is defined between single quotation marks.

Eg: 'r', 'H', '5', '*', etc.

Note: String is also a character data type. A group of characters defined between double quotation marks is a String.

Eg: "HHHH", "abc985", "123*/&", etc.

float: The numbers which are stored in the form of floating point representation is called as float data type.

Eg: 10.25, 478.1234, -56.37821, etc.

double: The numbers which are stored in the form of double precision floating point representation is called as

double data type.

Eg: 1023.56789, 12345.3672, 0.456321,
-567.4556 etc.

Note :

void : Empty Data type

Derived data types:

These data types are created from the basic integers, characters and floating data types.

Eg:

arrays, pointers, structures etc.

User defined data types:

The user defined data types enable a program to invent his own data types and define what values it can taken .

C supports 2 types of user defined data types.

- 1 typedef (type definition)
- 2 enum (enumerated data type)

Type modifiers : (signed, unsigned, short, long)

A type modifier alter the meaning of the base data type.

- 1 Each of these type modifiers can be applied to the base type *int*.
- 2 Type modifiers signed and unsigned can also be applied to the base type *char*.

3 In addition, long can be applied to *double*.

Sub classification of Primary data types, format specifiers , memory size and their accessibility range :

Data type	Format Spec	Memory size	Accessbility range
unsigned char	%c	1 Byte	0 to 255
char	%c	1 Byte	-128 to 127
int	%d	2 Bytes	-32768 to 32767
unsigned int	%u	2 Bytes	0 to 65535
long (or) long int	%ld	4 Bytes	-2147483648 to 2147483647
unsigned long (or) unsigned long int	%lu	4 Bytes	0 to 4294967295
float	%f	4 Bytes	3.4*(10 power -38) to 3.4*(10 power 38)
double	%lf	8 Bytes	1.7*(10 power -308) to 1.7*(10 power 308)
long double	%Lf	10 Bytes	3.4*(10 power -4932) to 1.1*(10 power 4932)
char[] (string)	%s	-----	-----
%o	Octal Base		
%x	Hexa decimal base		

%p Memory address

Variable:

Variables are used for storing data in a program. Based on the data type of a variable, the compiler allocates memory and decides what can be stored in the allocated memory. The value of variable may vary during the program execution.

Declaration of a Variable:

datatype identifier ;

(or)

datatype identifier-1, identifier -2,, identifier-n ;

Eg:

```
int  n;
char ch;
float ft;
double db;
int a,b,c,d;
char x,y,z;
```

Initialization of variable :

At the time of declaring a variable, we can store some value into that variable is known as initialization.

Syntax:

data type identifier = value;

Eg:

```
int n=100;
char ch='H';
float ft=10.569;
double db=123.618;
int a=10,b=20,c=30;
int x=100 , y , z=200;
```

Program :

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n=100;
    clrscr();
    printf("%d",n);
    printf("\nvalue of n = %d",n);
    getch();
}
```

Output

100

value of n=100

Note:

In C language, all declarations will be done before the first executable statement.

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n=100;
    char ch='H';
    float ft=17.148;
    double db=1714.1418;
    clrscr();
    printf("n=%d",n);
    printf("\nch=%c",ch);
    printf("\nft=%f",ft);
    printf("\ndb=%lf",db);
    getch();
}
```

Output

n=100

ch='H'

ft=17.148000

db=1417.141800

Note:

Floating(float ,double and longdouble) values displays 6 digits after the decimal point, by default.

If we want to display specified number of digits after decimal point for floating values, we use the following technique

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    float ft=12.4466;
    double db=1234.56789;
    clrscr();
    printf("ft=%f",ft);
    printf("db=%lf,db);
    printf("ft=%.2f",ft);
    printf("\ndb=%.2lf",db);
    getch();
}
```

Output

```
ft=12.44660
db=1234.567890
ft=12.45
db=1234.57
```

Constants:

Constants in C refer to fixed values that do not change during the execution of a program.

const:

It is a keyword and is used to declare constants.

Syntax:

```
const datatype identifier=value;
                        (or)
const datatype identifier-1=value,...,identifier-n=value;
```

Eg:

```
const int a=100;
const int x=10,y=20,z=30;
```

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    const int a=100;
    clrscr();
    printf("a=%d",a);
```

```
    /* a=200;*/  
    printf(“\na=%d”,a);  
    getch();  
}
```

Output

a=100
a=100

In the above program we give a=200, we will get an error
can't modify a constant object

Symbolic constants:

#define :

It is a pre-processor statement and is used to define symbolic constants.

Syntax :

#define identifier value

Eg : #define pi 3.14
 #define g 9.8

Example:

```
#include<stdio.h>
#include<conio.h>
#define pi 3.14
#define g 9.8
#define val 100
void main()
{
    clrscr();
    printf("pi = %.2f ",pi);
    printf("\ng = %.1f ",g);
    printf("\nval = %d",val);
    getch();
}
```

Output :

```
pi=3.14
g=9.8
VAL=100
```

scanf :

It is a function and is used to read data from the standard input device(KeyBoard).

Syntax:

```
int  scanf("format(s)",address-1,address-2,.....address-n);
```


eg :

```
int n;  
/* address of n=&n */  
scanf("%d",&n);
```

Example :

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    int n;  
    clrscr();  
    printf("Enter any Number :");  
    scanf("%d",&n);  
    printf("Given Number : %d",n);  
    getch();  
}
```

Example :

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

```
char ch;  
int n;  
float ft;  
double db;  
clrscr();
```

```
printf("Enter any char  :");  
scanf("%c",&ch);  
printf("Enter any int   :");  
scanf("%d",&n);  
printf("Enter any float :");  
scanf("%f",&ft);  
printf("Enter any double :");  
scanf("%lf",&db);
```

```
printf("\nGiven char   :%c",ch);  
printf("\nGiven int    :%d",n);  
printf("\nGiven float   :%.2f",ft);  
printf("\nGiven double  :%.2lf",db);  
getch();  
}
```

Skipping problem :

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

```
void main()
{
    int n;
    char ch;
    clrscr();
    printf("Enter any int :");
    scanf("%d",&n);
    printf("Enter any char:");
    scanf("%c",&ch);
    printf("\nGiven int :%d",n);
    printf("\nGiven char :%c",ch);
    getch();
}
```

Output

Enter any int :100

Enter any char:

Given int :100

Given char :

In the above program, it cannot read a character into the character variable, because previous integer reading statement creates a new line character in the input stream. That character will assign to the character variable.

To avoid this problem we use ***fflush*** or ***flushall*** functions.

fflush :

It flushes the specified stream.

Syntax:

fflush(streamname);

Eg: fflush(stdin);

flushall :

It flushes all open streams.

Syntax:

flushall();

Example :

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    char ch;
    clrscr();
    printf("Enter any int :");
    scanf("%d",&n);
    printf("Enter any char:");
    fflush(stdin); /* (or) flushall(); */
    scanf("%c",&ch);
    printf("\nGiven int :%d",n);
```

```
printf("\nGiven char :%c",ch);  
getch();  
}
```

String :

A group of characters defined between double quotation marks is a string. It is a constant string. In C language, a string variable is nothing but an array of characters and terminated by a null character (\0).

Declaration :

```
char identifier[size];
```

Eg : char st[20];

Initialization of string :

At the time of declaring a string variable, we can store a constant string into that variable is called as initialization.

syntax:

```
                optional  
                |  
char identifier[size]="string";
```

Eg :

```
char st1[10]="WELCOME";  
char st2[ ]="ABCD";
```

The compiler assigns a constant string to the string variable. It automatically supplied a null character(\0) at the end of string. Therefore the size should be equal to the maximum number of characters in the string plus(+) 1.

Note: In C language, string initialization is possible, but string assigning is not possible.

Example :

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char st1[10]="WELCOME";
    char st2[]="ABCD";
    printf("st1 = %s",st1);
    printf("\nst2 = %s",st2);
    getch();
}
```

Program : To accept a string from key board and to display the string.

Example :

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char st[80];
```

```
clrscr()";  
printf("Enter any string :");  
scanf("%s",st)  
printf("Given string is :%s",st);  
getch();  
}
```

Output :

1. Enter a string: welcome
Given string:welcome
2. Enter a string: welcome to BDPS LTd
Given string:welcome

Note :

scanf statement cannot read spaces into a string variable.

gets:

It is a function and is used to read a string(including spaces) from the standard input device(KeyBoard).

Syntax :

gets(string varibale);

Eg: gets(st);

Example :

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char st[80];
    clrscr();
    printf("Enter any string :");
    gets(st);
    printf("Given string is :%s",st);
    getch();
}
```

Program :

To enter any character and display its ASCII value

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    clrscr();
    printf("Enter any character :");
    scanf("%c",&ch);
    printf("ASCII value of given character :%d",ch);
    getch();
}
```


Program :

To enter any ASCII value and display its character

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    clrscr();
    printf("enter any ASCII value from 0 to 255 :");
    scanf("%d",&n);
    printf("character of give ASCII value: %c",n);
    getch();
}
```

Program :

To enter any date in date format and to display the given date.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int d,m,y;
    clrscr();
```

```
printf("Enter any date in the format dd-mm-yyyy :");  
scanf("%d-%d-%d",&d,&m,&y);  
printf("Given date is : %.2d-%.2d-%d",d,m,y);  
getch();  
}
```

Operators in C

Operator: It is a symbol and it performs particular operation.

Operand : It is an entity, on which an operator acts.

Binary operator: It requires 2 operands.

Unary operator : It requires only a single operand.

C operators can be classified into number of categories.
They are

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment and Decrement operators
6. Bit-wise operators
7. Special operators
 - a) Ternary (or) Conditional operator
 - b) Comma operator
 - c) sizeof operator

Arithmetic operators:

These are the basic operators in C language. These are used for arithmetic calculations.

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus(Remainder)

Program : *arithm.c*

To enter any 2 numbers and test all arithmetic operations.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b;
    clrscr();
    printf("Enter 2 numbers :");
    scanf("%d%d",&a,&b);
    printf("\nAddition of %d and %d is %d",a,b,a+b);
    printf("\nSubtraction of %d and %d is %d",a,b,a-b);
```

```
printf("\nMultiplication of %d and %d is %d",a,b,a*b);  
printf("\nDivision of %d and %d is %d",a,b,a/b);  
printf("\nModulus of %d and %d is %d",a,b,a%b);  
getch();  
}
```

Relational operators :

These are used to test the relation between 2 values or 2 expressions. All C relational operators are binary operators and hence it requires 2 operands.

<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equal to
!=	not equal to

Logical operators :

These are used to combine the result of 2 or more expressions.

&& Logical AND

|| Logical OR

! Logical NOT

Exp1	Exp2	Exp1 && Exp2	Exp1 Exp2
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

1. If Exp=True → !Exp=False

2. If Exp=False → !Exp=True

Assignment Operators :

These are used to assign a constant value or values of an expression to an identifier.

They are of 2 types.

1. Simple assignment : =

Eg : n=10

2. Shorthand (or) compound assignment

+=

- =

*=

/=

%=

etc.

Eg :

int n=10, if we want to add 5 to n then we will
give
n=n+5;
(or)
n+=5; which is compound assignment.

Increment and Decrement Operators :

These operators are used to control the loops in an effective method.

1 . Increment operator :

The symbol ++ is used for incrementing by 1.

It is of 2 types.

- 1) ++ identifier ; prefix increment
- 2) identifier++ ; postfix increment

Eg :

1) int a = 10;
 ++a; (or) a++;
 a = 11

2) int a = 10,b;
 b = ++a;
 a = 11
 b = 11

3) int a = 10,b;

```
b = a++;  
    a = 11  
    b = 10
```

```
4) int x = 10  
    printf(“%d \t %d”,x,x++);  
o/p: 11    10
```

Decrement operator :

The symbol -- is used for decrementing by 1.

It is of 2 types.

- 1) --identifier; prefix decrement
- 2) identifier-- ; postfix decrement

Eg :

```
1) int a = 10  
    --a; (or) a--;  
    a = 9
```

```
2) int a = 10,b;  
    b = --a;  
    a = 9  
    b = 9
```

```
3) int a = 10,b;  
    b = a--;  
    a = 9  
    b = 10
```

4)

```
int x = 10
printf(“%d \t% d”,x,x--);
output :  9      10
```

Bit-wise Operators :

These operators are used for manipulation of data at bit level.

These are classified into 2 types. Namely,

1. Bit-wise logical operators
2. Bit-wise shift operators.

Bit-wise logical operators :

These are used for bit-wise logical decision making.

& Bit-wise logical AND

| Bit-wise logical OR

^ Bit-wise logical XOR

B1	B2	B1 & B2		B1 B2	B1 ^ B2
1	1	1	1	0	
1	0	0	1	1	
0	1	0	1	1	
0	0	0	0	0	

Eg : int a=5,b=6;

```
  a = 5  -  1  0  1
  b = 6  -  1  1  0
```


$$\begin{array}{r}
 1 \quad 0 \quad 1 \\
 1 \quad 1 \quad 0 \\
 \hline
 a \& b : \quad 1 \quad 0 \quad 0 \quad = 4 \\
 \hline
 \end{array}$$

$$\begin{array}{r}
 1 \quad 0 \quad 1 \\
 1 \quad 1 \quad 0 \\
 \hline
 a | b : \quad 1 \quad 1 \quad 1 \quad = 7 \\
 \hline
 \end{array}$$

$$\begin{array}{r}
 1 \quad 0 \quad 1 \\
 1 \quad 1 \quad 0 \\
 \hline
 a \wedge b : \quad 0 \quad 1 \quad 1 \quad = 3 \\
 \hline
 \end{array}$$

$a = 7$
 $b = 9$

Example :

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b;
    printf("Enter values int a and b :");
    scanf("%d%d",&a,&b);
    printf("\n a & b = %d",a&b);
    printf("\n a | b = %d",a|b);
    printf("\n a ^ b = %d",a^b);
    getch();
}

```

Bit-wise shift operator :

The shift operations take binary patterns and shift the bits to the left or right.

<< left shift

>> right shift

Eg :

int a=4,b,c;

a=4 → 1 0 0 (binary form)

b=a<<2; means a is to be left shifted by 2 bits and store it in b.

													1	0	0
--	--	--	--	--	--	--	--	--	--	--	--	--	---	---	---

Hence it became,

											1	0	0	0	0
--	--	--	--	--	--	--	--	--	--	--	---	---	---	---	---

then the value of b :-

$$1\ 0\ 0\ 0\ 0 = 2^{\text{power } 4} = 16$$

a=4 → 1 0 0 (binary form)

c=a>>1; means a is to be right shifted by 1 bit and store it in c.

													1	0	0
--	--	--	--	--	--	--	--	--	--	--	--	--	---	---	---

Hence it became,

														1	0
--	--	--	--	--	--	--	--	--	--	--	--	--	--	---	---

then the value of c :-

$$1\ 0 = 2^{\text{power } 1} = 2$$

Example :

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a=4,b,c;
    clrscr();
```

```

    b=a<<2;
    c=a>>1;
    printf("\n a=%d",a);
    printf("\n b=%d",b);
    printf("\n c=%d",c);
    getch();
}

```

Special operators :

1) Ternary (or) Conditional operator :

A ternary operator pair " ? " and " : " is available in C language. It is used to construct a conditional expression.

The general form of ternary operator is :

Exp1 ? Exp2 : Exp3 ;

In this operator first Exp1 is evaluated, if it is true then Exp2 is evaluated and its value becomes the value of the Expression, otherwise Exp3 is evaluated and its value becomes the value of the Expression.

Eg :

```

int a=10,b=20,c;
c = a<b? a : b;
output will be c =10

```

Program : max1_ter.c

To find the maximum and minimum values of given 2 VALUES

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,max,min;
    clrscr();
    printf("Enter 2 numbers : ");
    scanf("%d%d",&a,&b);
    max=a>b ? a : b;
    min=a<b ? a : b;
    printf("Maximim value   : %d",max);
    printf("\nMinimum value   : %d",min);
    getch();
}

```

Program : evod_ter.c

To check whether the given number is even or odd.

```

#include<stdio.h>
#include<conio.h>
void main ()
{
    int n;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    n%2==0 ? printf("Given number is even") : printf("Given number is odd");
}

```

```
    getch();  
}
```

Program : max2_ter.c

To find the maximum and minimum values of given 3 numbers.

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    int a,b,c,max,min;  
    clrscr();  
    printf("Enter 3 numbers : ");  
    scanf("%d%d%d",&a,&b,&c);  
    max=a>b && a>c ? a : (b>c ? b : c);  
    min=a<b && a<c ? a : (b<c ? b : c);  
    printf("Maximim value   : %d",max);  
    printf("\nMinimum value   : %d",min);  
    getch();  
}
```

2) comma operator :

It can be used to link the related expressions together.

The general form of comma operator is :

var=(var-1=value,var-2=value,...,var-n=value, expr);

Eg :

```
int a,b,c;  
c=(a=10,b=20,a+b);
```

Here first assigns the value of 10 to a, then assigns 20 to b and finally assigns 30(a+b) to c.

Example : comma.c

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    int a,b,c;  
    clrscr();  
    c=(a=10,b=20,2*a+b);  
    printf("a=%d",a);  
    printf("\nb=%d",b);  
    printf("\nc=%d",c);  
    getch();  
}
```

3) sizeof operator :

It is used to get the memory size of specified variable or expression or data type.

Syntax :

```
sizeof (variable / expression / datatype);
```

Eg 1:

1. sizeof(int); 2
2. sizeof(float); 4

Eg2:

```
int a,b;  
sizeof(a); 2  
sizeof(b); 2  
sizeof(a+b); 2
```

Example : 1 *sizeof_1.c*

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    int n;  
    char ch;  
    float ft;  
    double db;  
    clrscr();  
    printf("size of int    : %d bytes",sizeof(n));  
    printf("\nsize of char   : %d byte",sizeof(ch));  
    printf("\nsize of float  : %d bytes",sizeof(ft));  
    printf("\nsize of double : %d bytes",sizeof(db));  
    getch();  
}
```


Type casting (or) Type conversion :

The process of converting from one data type to another is called as type casting or type conversion.

In C language type conversion is an arithmetic expression will be done automatically. This is called implicit conversion. If we want to store a value of one type into a variable of another type, we must caste the value to be stored by preceding it with the type name in parenthesis. This is called explicit conversion or type casting.

Eg :

```
int a,b;
float c,d;
a=5;
b=2;
c=a/b;  c=2.00   Automatic (or) implicit conversion
d=(float)a/b;    d=2.50   Type casting (or) explicit
conversion
```

Example : *typecast.c*

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

```
int a,b;
float c,d;
clrscr();
printf("Enter 2 numbers : ");
scanf("%d%d",&a,&b);
c=a/b;
d=(float)a/b;
printf("c=%.2f",c);
printf("\nd=%.2f",d);
getch();
}
```

Program : tri_area.c

To enter base and height of a triangle and to calculate its area.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    float b,h,area;
    clrscr();
    printf("Enter base of triangle  : ");
    scanf("%f",&b);
    printf("Enter height of triangle : ");
```

```
scanf("%f",&h);
area=(float)1/2*b*h; /* or area=0.5*b*h; */
printf("Area of triangle : %.2f square units",area);
getch();
}
```

Program : circl_ar.c

To enter radius of a circle and to calculate its area.

```
#include<stdio.h>
#include<conio.h>
#define pi 3.14
void main()
{
    float r,area;
    clrscr();
    printf("Enter radius of circle : ");
    scanf("%f",&r);
    area=pi*r*r;
    printf("Area of circle : %.2f square units",area);
    getch();
}
```

Program : simp_int.c

To enter principle amount, time , and rate of interest and to calculate and display simple interest and total amount.

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int y,m,t;
    float p,r,si,tamt;
    clrscr();
    printf("Enter principal amount          : ");
    scanf("%f",&p);
    printf("Enter rate of interest          : ");
    scanf("%f",&r);
    printf("Enter number of years and months : ");
    scanf("%d%d",&y,&m);
    t=m+(y*12);
    si=(p*t*r)/100;
    tamt=p+si;
    printf("Simple Interest                :%.2f",si);
    printf("\nTotal Amount                :%.2f",tamt);
    getch();
}

```

Program : swap1.c

swapping of given two values using temporary variable

```

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

```

```

void main()
{
    int a,b,t;
    clrscr();
    printf("Enter 2 numbers : ");
    scanf("%d%d",&a,&b);
    printf("\nBefore swapping ");
    printf("\na=%d",a);
    printf("\nb=%d\n\n",b);
    t=a;
    a=b;
    b=t;
    printf("After swapping ");
    printf("\na=%d",a);
    printf("\nb=%d",b);
    getch();
}

```

Program : swap2.c

swapping of given variable with out using temporay variable

```

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

```

```

clrscr();
printf("Enter 2 numbers : ");
scanf("%d%d",&a,&b);
printf("\nBefore swapping ");
printf("\na=%d",a);
printf("\nb=%d\n\n",b);
a=a+b;
b=a-b;
a=a-b;
printf("After swapping ");
printf("\na=%d",a);
printf("\nb=%d",b);
getch();
}

```

block :

A group of statements enclosed within curly braces is called as block.

```

{
-----
-----
}

```

Control statements (or) Control structures

C is a structured programming language. One of the

reasons for this is having various program control statements. C process the decision making capabilities and supports the statements known as control statements.

C supports 3 types of control statements. They are :

1. Conditional control statements.
2. Unconditional control statements.
3. Loop control statements.

Conditional control statements :

C supports 5 types of conditional control statements.

- 1) simple **if** statement
- 2) **if-else** statement
- 3) **Nested if** statement
- 4) **else-if** ladder statement
- 5) **switch** statement

simple if statement :

It is a conditional controlled statement and is used to control the flow of execution.

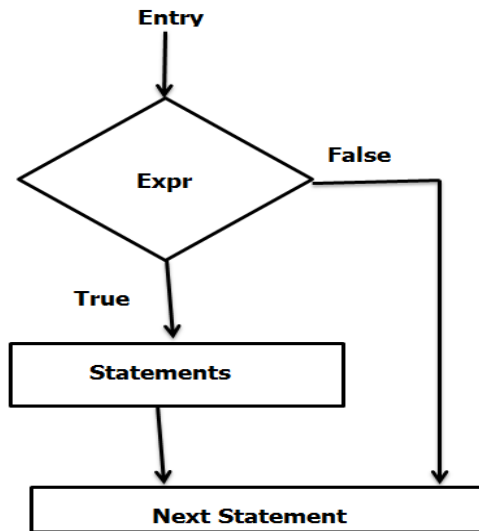
Syntax :

```
if( expr )
{
    statements;
}
```

In this statement, first the expression will be evaluated. If it is true then the statement block will be

executed and the control transfer to the next statement. Otherwise if the expression is false, then the control directly goes to the next statement.

flow chart



Note :

In any control statement, the statement block contains only a single statement, curly braces are not necessary.

Example : *sim_if1.c*

To find maximum value of given 2 values using simple if

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,max;
    clrscr();
```



```
printf("Enter any two values : ");
scanf("%d%d",&a,&b);
max=a;
if(max<b)
{
    max=b;
}
printf("Maximum Value : %d",max);
getch();
}
```

Example : *sim_if2.c*

To find maximum of 3 numbers using simple if

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,c,max;
    clrscr();
    printf("Enter 3 numbers : ");
    scanf("%d%d%d",&a,&b,&c);
    max=a;
    if(max<b)
        max=b;
```

```
if(max<c)
    max=c;
printf("Maximum Value  : %d",max);
getch();
}
```

if-else statement :

It is an extension of simple if statement.

Syntax :

```
if(expr)
{
    Statements-1;
}
else
{
    Statements-2;
}
```

In this statement, first the expression will be evaluated, and if it is true then the if block statements(statements-1) are executed and the else block statements(statements-2) are ignored. Otherwise if the expression is false, then else block statements are executed and if block statements are ignored.

Example : *if_ else.c*

To find maximum of 2 numbers using if_ else

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,max;
    clrscr();
    printf("Enter 2 numbers : ");
    scanf("%d%d",&a,&b);
    if(a>b)
        max=a;
    else
        max=b;
    printf("Maximum value : %d",max);
    getch();
}
```

Example : if_ else2.c

To check whether the given number is even or odd

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    if(n%2==0)
        printf("Given number is Even");
    else
        printf("Given number is Odd");
    getch();
}
```

Nested if statement :

Using a if statement within another if is called as nested if. If a series of decisions are involved, we use nested if statement.

Form : 1

```
if(expr-1)
{
    if(expr-2)
    {
```

```
.....  
if(expr-n)  
{  
    statements;  
}  
.....  
}  
}
```

Form : 2

```
if(expr-1)
{
    if(expr-2)
    {
        Statement-1;
    }
    else
    {
        Statement-2;
    }
}
else
{
    if(expr-3)
    {
        Statement-3;
    }
    else
    {
        Statement-4;
    }
}
```

Example : nest_if.c

To find maximum of 3 numbers using nested if

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,c,max;
    printf("Enter any three numbers : ");
    scanf("%d%d%d",&a,&b,&c);
```

```

if(a>b)
{
    if(a>c)
        max=a;
    else
        max=c;
}
else
{
    if(b>c)
        max=b;
    else
        max=c;
}
printf("Maximum value : %d",max);
getch();
}

```

else-if ladder :

This statement is also used for a series of decisions are involved.

Syntax :

```

if(expr-1)
{
    Statements-1;
}
else if(expr-2)
{

```



```
    Statement-2;
}
.....
else if(expr-n)
{
    Statement-n;
}
else
{
    else block statements;
}
```

In this statement, the expressions are evaluated from top to bottom, if the condition is true then the statements associated that block is executed and the control transfers to the next statement. Otherwise when all expressions are false then the final else block statements will be executed.

Example : *elif_lad.c*

To find maximum of 3 numbers using else if ladder

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,c,max;
    printf("Enter any three numbers : ");
    scanf("%d%d%d",&a,&b,&c);
    if(a>b && a>c)
        max=a;
    else if(b>c)
        max=b;
    else
        max=c;
    printf("Maximum value : %d",max);
    getch();
}
```

Example : ck_char.c

To check whether the given number is alphabet or digit or special character

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    clrscr();
    printf("Enter any character : ");
    scanf("%c",&ch);

    if((ch>=65 && ch<=90) || (ch>=97 && ch<=122))
    //if((ch>='A' && ch<='Z') || (ch>='a' && ch<='z'))
        printf("Given character is a alphabet");
    else if(ch>=48 && ch<=57)
    // else if(ch>='0' && ch<='9')
        printf("Given character is a digit");
    else
        printf("Given character is a special character");
    getch();
}
```

Example : vow_cons.c

To check whether the given character is vowel or consonant

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    clrscr();
    printf("Enter any character : ");
    scanf("%c",&ch);
    if((ch>='a' && ch<='z') || (ch>='A' && ch<='Z'))
    {
        if(ch=='a' || ch=='A' || ch=='e' || ch=='E' || ch=='i' ||
           ch=='I' || ch=='o' || ch=='O' || ch=='u' || ch=='U')
            printf("Given character is a vowel");
        else
            printf("Given character is a consonant");
    }
    else
        printf("Given character is not an Alphabet");
    getch();
}
```

Example : *st_det.c*

To enter student number, name, marks in C,C++ and java

. calculate and display total marks, average, result and grade.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int sno,c,cpp,java,tot;
    char sname[20],res[5],div[10];
    float avg;
    clrscr();
    printf("Enter student number    : ");
    scanf("%d",&sno);
    printf("Enter student name      : ");
    fflush(stdin);
    gets(sname);
    printf("Enter marks in C          : ");
    scanf("%d",&c);
    printf("Enter marks in CPP          : ");
    scanf("%d",&cpp);
    printf("Enter marks in Java         : ");
    scanf("%d",&java);
    tot=c+cpp+java;
    avg=(float)tot/3;
    clrscr();
    printf("Student Number    : %d",sno);
    printf("\nStudent Name    : %s",sname);
    printf("\nMarks in C        : %d",c);
    printf("\nMarks in CPP       : %d",cpp);
```

```

printf("\nMarks in Java    : %d",java);
printf("\nTotal Marks      : %d",tot);
printf("\nAverage Marks    : %.2f",avg);

if(c>=50 && cpp>=50 && java>=50)
{
    printf("\nResult        : PASS ");
    if(avg>=60)
        printf("\nDivision    : FIRST");
    else
        printf("\nDivision    : SECOND");
}
else
{
    printf("\nResult        : FAIL");
    printf("\nDivision    : NO DIVISION");
}
getch();
}

```

exit : <process.h>

It terminates the program.

Syntax: void exit(int status)

Status :

Typically value of zero , indicates a normal exit, and a non

zero indicates some error.

Example : *cbill.c*

To enter consumer number, name, present month reading, last month reading. calculate and display total units and bill amount by the following table.

Units	
rate/unit	
0 – 50	1.45(min)
51-100	2.80
101-200	3.05
201-300	4.75
>300	5.50

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
{
    int cno,pmr,lmr,tu;
    char cname[20];
    float bamt;
    clrscr();
    printf("Enter consumer number : ");
```

```
scanf("%d",&cno);
printf("Enter consumer name  :");
fflush(stdin);
gets(cname);
printf("Enter present month reading :");
scanf("%d",&pmr);
printf("Enter last month reading  :");
scanf("%d",&lmr);
if(pmr<lmr)
{
    printf("Invalid reading");
    getch();
    exit(0);
}
tu=pmr-lmr;
if(tu<=50)
    bamt=50*1.45;
else if(tu<=100)
    bamt=50*1.45+(tu-50)*2.80;
else if(tu<=200)
    bamt=(50*1.45)+(50*2.80)+(tu-100)*3.05;
else if(tu<=300)
    bamt=(50*1.45)+(50*2.80)+(100*3.05)+(tu-200)*4.75;
else
    bamt=(50*1.45)+(50*2.80)+(100*3.05)+(100*4.75)+(tu-300)*5.50;
clrscr();
```



```

printf("\nConsumer Number      : %d",cno);
printf("\nConsumer Name        : %s",cname);
printf("\nPresent month reading : %d",pmr);
printf("\nLast month reading    : %d",lmr);
printf("\nTotal units           : %d",tu);
printf("\nTotal Bill Amount      : %.2f",bamt);
getch();
}

```

Switch statement :

It is a multiway conditional control statement used in C language. It is mainly used in situations where there is need to pick one alternative among many alternatives.

Syntax :

```

switch(variable / expression)
{
    case value-1 :
        statements – 1;
        break;

    case value-2 :
        statements – 2;
        break;

    .....

    .....
    case value-n :
        statements – n;
        break;
}

```

```
        default :  
            default statements;  
    }
```

The switch statement tests the value of given variable or expression against a list of case values. When a match is found, the statement associated to that case is executed and the control transfer to the next statement, otherwise the default statements will be executed.

break :

It is an unconditional control statement and used to terminate a switch statement or a loop statement.

Syntax :

```
break;
```

Note :

1. In switch statement the variable or expression is an integral value(integer or character), we cannot use floating and string values.
2. In switch statement, the default case is optional.

Program : *switch_1.c*

To test all arithmetic operations using switch statement

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,opt;
    clrscr();
    printf("Enter two numbers : ");
    scanf("%d%d",&a,&b);
    printf("1-Addition\n2-Subtraction\n3-Multiplication\n4-
Division\n5-Modulus\nEnter your option : ");
    scanf("%d",&opt);
    switch(opt)
    {
        case 1:
            printf("Addition of %d and %d is %d",a,b,a+b);
            break;

        case 2:
            printf("Subtraction of %d and %d is %d",a,b,a-b);
            break;

        case 3:
```

```

        printf("Multiplication of %d and %d is
%d",a,b,a*b);
        break;

    case 4:
        printf("Division of %d and %d is
%.2f",a,b,(float)a/b);
        break;
    case 5:
        printf("Modulus of %d and %d is %d",a,b,a%b);
        break;

    default:
        printf("Invalid Option");
    }
    getch();
}

```

Program : *switch_2.c*

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b;
    char opt;

```

```

clrscr();
printf("Enter 2 numbers : ");
scanf("%d%d",&a,&b);
printf("A-Addition\nS-Subtraction\nM-
Multiplication\nD-Division\nR-Modulus\nEnter      your
option : ");
fflush(stdin);
scanf("%c",&opt);
switch(opt)
{
    case 'A':
    case 'a':

        printf("Addition of %d and %d is %d",a,b,a+b);
        break;

    case 'S':
    case 's':
        printf( "Subtraction of %d and %d is %d",a,b,a-b);
        break;

    case 'M':
    case 'm':
        printf("Multiplication    of    %d    and    %d    is
%d",a,b,a*b);
        break;

    case 'D':

```

```
case 'd':
    printf("Division of %d and %d is %d",a,b,a/b);
    break;

case 'R':
case 'r':
    printf("Modulus of %d and %d is %d",a,b,a%b);
    break;

default:
    printf("Invalid Option");
}
getch();
}
```

Unconditional control statements :

C supports 3 types of unconditional control statements.
They are

- 1)break
- 2)continue
- 3)goto

break :

It is an unconditional control statement and is used to terminate a switch statement or a loop statement.

Syntax :

break;

continue :

It passed the control

Syntax :

continue;

Causes the control to pass to the end of the innermost enclosing while, do or for statement, at which point the loop continuation condition is reevaluated.

goto :

It is an unconditional control statement, which is used to alter the execution of the program sequence by transfer of control to some other part of the program.

Syntax :

goto label;

Where label is valid C identifier used to the label of the destination such that the control could be transferred.

Syntax of label :

identifier :

Program : nat_goto.c

To display natural numbers from 1 to given number using

default Columns (80) and Rows(25)

Note : If the coordinates are invalid, the call to gotoxy is ignored.

Program : gotoxy.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
    clrscr();
    gotoxy(35,12);
    printf("WELCOME");
    getch();
}
```

Loop control statements :

loop:

The process of repeatedly executing a block of statement up to specified number of times is called as loop. C supports 3 types of looping statements. They are :

- 1) while loop
- 2) do-while loop

3) for loop

While loop (Entry control loop statement) :

It is a conditional controlled loop statement in C language.

Syntax :

```
while(test condition)
{
    Statements;
}
```

In this loop first the test condition will be evaluated. If it is true, then the statement block will be executed. After the execution of statements, the test condition will be evaluated once again, if it is true then the statement block will be executed once again. This process of repeated execution continued until the test condition becomes false.

Program : nnos_w.c

To print natural numbers from 1 to given number

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,i=1;
    printf("Enter a number : ");
    scanf("%d",&n);
    printf("Natural number from 1 to %d\n\n",n);
    while(i<=n)
    {
        printf("\t%d",i);
        i++;
    }
    getch();
}
```

Program : ev_od_w.c

To display even and odd numbers for 1 to given number

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

```

printf("Enter a number : ");
scanf("%d",&n);
printf("\n\nEven numbers from 1 to %d\n\n",n);
i=1;
while(i<=n)
{
    if(i%2==0)
        printf("\t%d",i);
    i++;
}
printf("\n\nOdd numbers from 1 to %d\n\n",n);
i=1;
while(i<=n)
{
    if(i%2==1) // if(i%2!=0)
        printf("\t%d",i);
    i++;
}
getch();
}

```

Program : nsum_w.c

To print sum of natural numbers for 1 to given number

```

#include<stdio.h>
#include<conio.h>
void main()

```

```

{
    int n,i=1,sum=0;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    while(i<=n)
    {
        sum=sum+i;
        i++;
    }
    printf("Sum of %d natural numbers is %d",n,sum);
    getch();
}

```

Note :

We can also find the sum of ***n*** digits by the following formula :

$$\text{Sum} = n * (n+1)/2$$

Program : facts_w.c

To display factors of given number

```

#include<stdio.h>
#include<conio.h>
void main()

```

```

{
    int n,i=1;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    printf("\n\nFactors of given number : \n\n");
    while(i<=n)
    {
        if(n%i==0)
            printf("\t%d",i);
        i++;
    }
    getch();
}

```

Program : facto_w.c

To find factorial of given number

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    unsigned long fact=1;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);

```

```

while(n>=1)
{
    fact=fact*n;
    n--;
}
printf("\n\nFactorial of %d is %lu ",n,fact);
getch();
}

```

Program : rev_w.c

To find reverse number of a given number using while loop

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    unsigned long rev=0;
    printf("Enter a number : ");
    scanf("%d",&n);
    while(n>0)
    {
        rev=(rev*10)+(n%10);
        n=n/10;
    }
}

```

```
printf("Reverse number of given number : %lu",rev);  
getch();  
}
```

Program : pal_w.c

To check whether the given number is palindrome or not

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    int n,m,rev=0;  
    printf("Enter a number : ");  
    scanf("%d",&n);  
    m=n;  
    while(m>0)  
    {  
        rev=(rev*10)+(m%10);  
        m=m/10;  
    }  
    if(n==rev)  
        printf("Given number is a Palindrome");  
    else
```



```
    printf("Given number is not Palindrome");
    getch();
}
```

prime numbers are numbers that have only 2 factors: 1 and themselves

Program : prime_w.c

To check whether the given number is prime or not

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,i=1,count=0;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    while(i<=n)
    {
        if(n%i==0)
            count++;
        i++;
    }
    if(count==2)
        printf("Given number is Prime");
    else
        printf("Given numbr is not Prime");
    getch();
}
```

```
}
```

Program : fibo_w.c

To generate fibonacci series upto n terms

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int m,n,a=1,b=0,c=0;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    printf("Fibonacci series :");
    while(n>=1)
    {
        printf("\t%d",c);
        c=a+b;
        a=b;
        b=c;
        n--;
    }
    getch();
}
```

do-while loop : (exit control loop statement)

It is an alternative form of while loop. The only

difference between while and do-while is the minimum number of execution of while is zero and the minimum number of execution of do-while is one.

Syntax :

```
do
{
    statements;
}
while(test condition) ;
```

In this loop first the statement block will be executed, after the execution of statements, the test condition will be evaluated. If it is true, then the statement block will be executed once again, this process of repeated execution continuous until the test condition becomes false.

Program : nat_dw.c

Programs to display natural number from 1 to given number is using do-while loop

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,i=1;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    printf("\nNatural numbers from 1 to %d :",n);
    do
    {
        printf("\t%d",i);
        i++;
    } while(i<=n);

    getch();
}
```

for loop :

It is the most commonly used loop statement in C language. It is consisting of 3 expressions.

Syntax :

```
for(exp1 ; exp2 ; exp3)
{
    Statements;
}
```

In this loop first expression is used to initialize the index, second expression is used to test whether the loop is to be continued or not (test condition) and the third expression is used to change the index for further iteration.

Program : nat_for.c

Programs to display natural number from 1 to given number is using for loop

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,i;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    printf("\nNatural numbers from 1 to %d\n\n",n);
    for(i=1;i<=n;i++)
    {
        printf("\t%d",i);
    }
    getch();
}
```

Program : rev_for.c

To find reverse number of a given number using for loop

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    unsigned long rev;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    for(rev=0;n>0;n=n/10)
    {
        rev=(rev*10)+(n%10);
    }
    printf("Reverse number of given number : %lu",rev);
    getch();
}
```

Program : perfect.c

To check whether the given number is perfect number or not

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

```

int n,i,sum=0;
clrscr();
printf("Enter a number : ");
scanf("%d",&n);
for(i=1;i<=n/2;i++)
{
    if(n%i==0)
        sum=sum+i;
}
if(n==sum)
    printf("Given number is Perfect number");
else
    printf("Given number is not Perfect number");
getch();
}

```

Perfect number :

If a given number = sum of its divisibles except that number, is called as perfect number.

Eg : 6, 28, 496 etc

Program : armst.c

*To check whether the given number is armstrong number
of not*


```

#include<stdio.h>
#include<conio.h>
void main()
{
    int n,sum=0,m,r;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    m=n;
    while(m>0)
    {
        r=m%10;
        sum=sum+(r*r*r);
        m=m/10;
    }
    if(n==sum)
        printf("Given number is Armstrong number");
    else
        printf("Given number is not Armstrong number");
    getch();
}
1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407,

```

Program : table.c

To display multiplication table of given number from 1 to 20

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,i;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    for(i=1;i<=20;i++)
    {
        printf("\n%d * %d = %d",n,i,n*i);
    }
    getch();
}
```

Nested loops :

Using a loop statement, within another loop is called as nested loop.

Program : prime_nl.c

To display prime numbers from 1 to given number

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

```

void main()
{
    int n,i,j,count;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    printf("Prime Numbers for 1 to %d\n",n);
    for(i=1;i<=n;i++)
    {
        count=0;
        for(j=1;j<=i;j++)
        {
            if(i%j==0)
                count++;
        }
        if(count==2)
            printf("\t%d",i);
    }
    getch();
}

```

Program : prm2_nl.c

To display prime numbers between given 2 numbers

```

#include<stdio.h>
#include<conio.h>
void main()

```

```

{
    int n1,n2,i,j,count;
    clrscr();
    printf("Enter starting number : ");
    scanf("%d",&n1);
    printf("Enter ending number  : ");
    scanf("%d",&n2);
    for(i=n1;i<=n2;i++)
    {
        count=0;
        for(j=1;j<=i;j++)
        {
            if(i%j==0)
                count++;
        }
        if(count==2)
            printf("\t%d",i);
    }
    getch();
}

```

Program : pal_nl.c

To display Palindrome numbers from 1 to n

```

#include<stdio.h>
#include<conio.h>
void main()

```

```

{
    int n,rev,m,i;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        rev=0;
        m=i;
        while(m>0)
        {
            rev=(rev*10)+(m%10);
            m=m/10;
        }
        if(rev==i)
            printf("\t%d",i);
    }
    getch();
}

```

Program : perf_nl.c

To display perfect numbers from 1 to given number

```

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

```

```

int n,i,j,sum=0;
clrscr();
printf("Enter a number : ");
scanf("%d",&n);
printf("Perfect Numbers between 1 and %d : ",n);
for(i=1;i<=n;i++)
{
    sum=0;
    for(j=1;j<=i/2;j++)
    {
        if(i%j==0)
            sum=sum+j;
    }
    if(sum==i)
        printf("\t%d",i);
}
getch();
}

```

Program : armst_nl.c

To display Armstrong numbers from 1 to given number

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int n,m,r,arm,i;

```

```
clrscr();
printf("Enter a number : ");
scanf("%d",&n);
printf("Armstrong numbers from 1 to %d : \n",n);
for(i=1;i<=n;i++)
{
    arm=0;
    m=i;
    while(m>0)
    {
        r=m%10;
        arm=arm+(r*r*r);
        m=m/10;
    }
    if(arm==i)
        printf("\t%d",i);
}
getch();
}
```

Formatting the output :

Integer :

```
1. int n=100
    printf("%d",n);
```

Output : 100

2. int n=100
printf(“%5d”,n);

Output : 100

2 spaces

3. int n=100
printf(“%-5d”,n);

Output : 100

2 spaces

4. int n=5
printf(“%.2d”,n);

Output : 05

5. int n=5
printf(“%.3d”,n);

Output : 005

Character :

1. char c='R'
printf("%c",c);
Output : R

2. char c='R'
printf("%4c",c);

Output : R

3 spaces

3. char c='R'
printf("%-4c",c);

Output : 'R'

3 spaces

Float :

1. float ft=14.1718
printf("%f",ft);

Output : 14.171800

2. float f=14.1718
printf(“%12f”,f);

Output : 14.171800

3 spaces

3. float f=14.1718
printf(“%-12f”,f);

Output : 14.171800

3 spaces

4. float f=14.1718
printf(“%.2f”,f);

Output : 14.17

5. float f=14.1718
printf(“%7.2f”,f);

Output : 14.17

2 spaces

In 7.2, 7 indicates total number of digits including decimal point, 2 represents number of digits after decimal point.

String :

1. `char st[10]="welcome"`
`printf("%s",s);`

Output : welcome

2. `char st[10]="welcome"`
`printf("%9s",s);`

Output : welcome

2 spaces

3. `char st[10]="welcome"`
`printf("%-9s",s);`

Output : welcome

2 spaces

4. `char st[10]="welcome"`
`printf("%.3s",s);`

Output : wel

Other important formats :

1. `int n=100, x=5;`
`printf("%*d",x,n);` \leftrightarrow `printf("%5d",n);`

Output : 100

2 spaces

2. `char st[10] = "welcome";`
`int x=3;`
`printf("%.*s",x,st);` \leftrightarrow `printf("%.3s",st);`

Output : wel

Program : *patrnl.c*

To generate the given pattern

w
we
wel
welc
welco
welcom
welcome

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int x;
    char st[10]="welcome";
    clrscr();
    for(x=1;x<=7;x++)
    {
        printf("\n%.*s",x,st);
    }
    getch();
}
```

textmode() :

It changes screen mode (in text mode).

Syntax : void textmode(int newmode);

Constant	Value	Text mode	
LASTMODE	-1	Previous text mode	
BW40 columns	0	Black and White	40
C40 columns	1	Color	40
BW80 columns	2	Black and White	80
C80 columns	3	Color	80
MONO columns	7	Monochrome	80
C4350 43 line	64	EGA (Enhanced graphic adapter)	

line

graphic adapter)

Program : txtmode1.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
    clrscr();
    textmode(1); // or textmode(C40);
    gotoxy(17,12);
    printf("RAJI");
    getch();
}
```

Program : txtmode2.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i;
    clrscr();
    textmode(64); // or textmode(C4350);
    for(i=1;i<=45;i++)
    {
```

```

    printf("\n BDPS");
}
getch();
}

```

textcolor() :

It selects a new character color in text mode.

Syntax : void textcolor(int newcolor);

textbackground() :

It selects a new text background color

Syntax : void textbackground(int new color);
color (textmode) :

Constant	Value	Background	Foreground
BLACK	0	Yes	Yes
BLUE	1	Y	Y
GREEN	2	Y	Y
CYAN	3	Y	Y
RED	4	Y	Y
MAGENTA	5	Y	Y
BROWN	6	Y	Y

LIGHT GREY	7	Y	Y
DARK GREY	8	N	Y
LIGHT BLUE	9	N	Y
LIGHT GREEN	10	N	Y
LIGHT CYAN	11	N	Y
LIGHT RED	12	N	Y
LIGHT MAGENTA	13	N	Y
YELLOW	14	N	Y
WHITE	15	N	Y

cprintf() :

It is same as printf(). If we want to display colors in text mode, we use cprintf().

Program : txtmode3.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
    clrscr();
    textmode(1);
    textcolor(RED+BLINK);
    textbackground(WHITE);
```

```
gotoxy(17,12);
cprintf("RAJI");
getch();
}
```

Program : patrn2.c

To generate the following pattern

n=5

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,n;
    clrscr();
    printf("Enter value of n : ");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            printf("%2c",'*');

```

```
    }  
    printf("\n");  
}  
getch();  
  
}
```

Program : patrn3.c

To generate the following pattern

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

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    int i,j,n;  
    clrscr();  
    printf("Enter value of n : ");  
    scanf("%d",&n);  
    for(i=1;i<=n;i++)  
    {
```

```

        for(j=1;j<=i;j++)
        {
            printf("%2c",'*');
        }
        printf("\n");
    }
    getch();
}

```

Program : patrn4.c

To generate the following pattern

```

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

```

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,n;
    clrscr();
    printf("Enter value of n : ");
    scanf("%d",&n);
}

```

```
for(i=1;i<=n;i++)
{
    for(j=1;j<=i;j++)
    {
        printf("%3d",i);
    }
    printf("\n");
}
getch();
}
```

Program : patrn5.c

To generate the following pattern

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,n;
    clrscr();
    printf("Enter a number : ");
```

```

scanf("%d",&n);
for(i=1;i<=n;i++)
{
    for(j=1;j<=i;j++)
    {
        printf("%3d",j);
    }
    printf("\n");
}
getch();
}

```

Program : patrn6.c

To generate the following pattern

```

      *
    *  *
  *  *  *
*  *  *  *
*  *  *  *  *

```

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,k,n,s;

```

```

clrscr();
printf("Enter value of n : ");
scanf("%d",&n);
s=n*2;
for(i=1;i<=n;i++)
{
    /*
        for(k=1;k<=s;k++)
        {
            printf(" ");
        }

        or */

    printf("%*c",s,32);
    for(j=1;j<=i;j++)
    {
        printf("%4c",'*');
    }
    printf("\n");
    s=s-2;
}
getch();
}

```

Program : patrn7.c

To generate the following pattern

```

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

```

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,k,n,s;
    clrscr();
    printf("Enter value of n : ");
    scanf("%d",&n);
    s=n*2;
    for(i=1;i<=n;i++)
    {
        for(k=1;k<=s;k++)
        {
            printf(" ");
        }
        for(j=1;j<=i;j++)
        {
            printf("%4d",i);
        }
        printf("\n\n");
    }
}

```



```

    s=s-2;
}
getch();
}

```

Program : patrn8.c

To generate the following pattern

```

      *
    *  *
  *  *  *
*  *  *  *
*  *  *  *  *
  *  *  *  *
    *  *  *
      *  *
        *

```

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,k=1,n,s;
    clrscr();
    printf("Enter value of n : ");
    scanf("%d",&n);

```

```

s=n*2;
for(i=1;i<=n*2;i++)
{
    printf("%*c",s,32);
    for(j=1;j<=k;j++)
    {
        printf("%4c",'*');
    }
    if(i<n)
    {
        s=s-2;
        k++;
    }
    else
    {
        s=s+2;
        k--;
    }
    printf("\n");
}
getch();
}

```

Program : patrn9.c

To generate the following pattern

n=5

1 2 3 4 5

10	9	8	7	6
11	12	13	14	15
20	19	18	17	16
21	22	23	24	25

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,n,k=1;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            printf("%4d",k);
            if(j<n)
            {
                if(i%2==0)
                    k--;
                else
                    k++;
            }
        }
    }
```

```

        printf("\n");
        k=k+n;
    }
    getch();
}

```

Program : patrn10.c

To generate the following pattern

```

    n=5
    1 2 3 4 5
    5 1 2 3 4
    5 4 1 2 3
    5 4 3 1 2
    5 4 3 2 1

```

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,k,n,m;
    clrscr();
    printf("Enter value of n : ");
    scanf("%d",&n);
    m=n;
    for(i=1;i<=n;i++)
    {
        for(j=n;j>m;j--)

```

```
{  
    printf("%3d",j);  
}  
for(k=1;k<=m;k++)  
{  
    printf("%3d",k);  
}  
printf("\n");  
m--;  
}  
getch();  
}
```

Functions

Function :

Function is a self contained block of statements and it performs a particular task . It can be used at several times in a program, but defined only once.

They are of 2 types.

1. Library functions (or) pre-define functions
2. User defined functions.

Library functions:

The functions which are In-built with the compiler are

known as Library functions (or) Pre-defined functions.

Eg:

- 1 printf
- 2 scanf
- 3 getch
- 4 clrscr, etc.

Userdefined functions:

User can define functions to do a task relevant to their programs. Such functions are called as userdefined functions.

Note: "main" is pre-declared user defined function

- Any function(either pre-defined (or) user defined) has 3 things.

1. Function declaration
2. Function definition
3. Function calling

In case of Library functions(pre-defined), the function declaration is in header files, function definition is in C libraries and function calling is in source program. But in case of user defined functions all three things are in source program.

Function declaration:

Syntax:

```
returntype  function_name ( [argument_list] );
```

Function definition:

Syntax:

```
returntype  function_name ( [argument_list] )  
{  
    statements;  
}
```

Function calling :

Syntax:

```
function_name ( [parameter_list] );
```

Note:

The arguments which are given at the time of function declaration or function definition are called as arguments or formal arguments. The arguments which are given at the time of function calling are called as parameters or actual arguments.

Eg:

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

```

void func( );    // declaration
clrscr();
func();          // calling
getch();
}
void func()      // definition
{
    printf("\nWelcome to C Functions");
}

```

Rules for Creating and accessing functions:

1. A function can be called by any number of times.
2. A function may or maynot receive arguments.
3. A function may or may not return a value
4. If a function doesnot return any value, the function return data type will be specified as void.
5. If A function returns a value ,only one value it can be returned.
6. If a function returnrs a value, the returning value must be returned with statement "return".
7. If a function returns a value ,the execution of the return statement should be last.
8. If we cannot specify any returntype, the function returns an int by default.
9. A function returns a value, the returning value should match with the function return data type.
10. A function is defined after or before the main

function.

11. Before calling a function, the function declaration or definition is must and should
12. if a function definition is specified before the function call, then the function declaration is not necessary.
13. A function is executed when the function is called by its name.
14. If a function returns a value, the return value is replaced by function calling statement.
15. The function definition should not be terminated with a semicolon(;

return(keyword):

Exits immediately from the currently executing function to the calling routine, optionally returning a value.

Syntax:

```
return [expr];
```

Example : ret_max.c

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

```

#include<conio.h>
void main()
{
    int maxval(int,int);
    int a,b,m;
    clrscr();
    printf("Enter any two numbrs : ");
    scanf("%d%d",&a,&b);
    m=maxval(a,b);
    printf("Maximum Value : %d",m);
    getch();
}
int maxval(int x,int y)
{
    if(x>y)
        return x;
    else
        return y;
}

```

Function categories (or) Function prototypes:

A function ,depending on whether arguments are present or not and whether a value is returning or not,may belongs to one of the following categories.

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

Category-1:Function with no arguments and no return value:

In this type the function(called) has no arguments, it doesnot receive any data from the calling function. Similarly it doesnot return any value, the calling function doesnot receive any data from called function. So there is no data communication between calling function and called function.

Example : *cat1.c*

```
#include<stdio.h>
#include<conio.h>
void main()
{
    void sum();
    clrscr();
    sum();
    getch();
}
```

```
void sum()
{
    int a,b;
    printf("Enter any two numbers:");
    scanf("%d%d",&a,&b);
    printf("sum= %d",a+b);
}
```

category-2 Function with arguments and no return value:

In this type the function has some arguments, it receives data from the calling function. But it doesnot return any value, the calling function doesnot receive any data from the called function. So there is one way data communication between calling function and called function.

Example : cat2.c

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

```

    int a,b,s;
    clrscr();
    printf("Enter any two numbers :");
    scanf("%d%d",&a,&b);
    sum(a,b);
    getch();
}
void sum(int x,int y)
{
    printf("sum= %d",x+y);
}

```

Category 3:Function with arguments and return value:

In this type the function has some arguments, it receives data from the calling function. Similarly it returns a value, the calling function receives data from the called function. So there is two way data communication between calling function and called function.

Example : cat3.c

```

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

```

```

    clrscr();
    printf("Enter any two numbers :");
    scanf("%d%d",&a,&b);
    s=sum(a,b);
    printf("sum= %d",s);
    getch();
}
int sum(int x,int y)
{
    return x+y;
}

```

category-4: Function with no arguments and return value:

In this type the function has no arguments, it doesnot receive any data from the calling function. But it returns a value, the calling function receives data from the called function. So there is one way data communication between called function and calling function.

Example : cat4.c

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int sum();
    int s;

```

```
    clrscr();
    s=sum();
    printf("sum = %d",s);
    getch();
}
int sum()
{
    int a,b;
    printf("Enter any two numbers:");
    scanf("%d%d",&a,&b);
    return a+b;
}
```

Program : fn_max.c

To find maximum value of given 3 numbers

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int maxval(int,int);
    int a,b,c,m;
    clrscr();
```

```

printf("Enter 3 numbrs : ");
scanf("%d%d%d",&a,&b,&c);
m=maxval(a,b);
m=maxval(m,c);
printf("Maximum Value : %d",m);
getch();
}
int maxval(int x,int y)
{
    if(x>y)
        return x;
    else
        return y;
}

```

Program : nnos_fn.c

To display natural numbers from 1 to n

```

#include<stdio.h>
#include<conio.h>
void main()
{
    void disp(int);
    int n;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    disp(n);
}

```



```
    getch();
}
void disp(int x)
{
    int i;
    for(i=1;i<=x;i++)
    {
        printf("%d\t",i);
    }
}
```

Program : fact_fn.c

To calculate factorial of given number

```
#include<stdio.h>
#include<conio.h>
void main()
{
    unsigned long fact(int);
    int n;
    unsigned long f;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    f=fact(n);
    printf("Factorial of %d : %lu",n,f);
    getch();
}
```

```
}  
unsigned long fact(int x)  
{  
    unsigned long f=1;  
    while(x>=1)  
    {  
        f=f*x;  
        x--;  
    }  
    return f;  
}
```

Program : rev_fn.c

To display reverse number of given number

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    unsigned long rev(int);  
    int n;  
    unsigned long r;  
    clrscr();  
    printf("Enter a number : ");  
    scanf("%d",&n);  
    r=rev(n);  
    printf("Reverse number of %d : %lu",n,r);
```

```
    getch();
}
unsigned long rev(int x)
{
    unsigned long r=0;
    while(x>0)
    {
        r=r*10+(x%10);
        x=x/10;
    }
    return r;
}
```

Program : fibo_fn.c

To generate fibonacci series

```
#include<stdio.h>
#include<conio.h>
void main()
{
    void fibo(int);
    int n;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    printf("Fibonacci series \n");
```

```

    fibo(n);
    getch();
}
void fibo(int x)
{
    int i,a=0,b=1,c;
    printf("%d\t%d",a,b);
    for(i=1;i<=x-2;i++)
    {
        c=a+b;
        printf("\t%d",c);
        a=b;
        b=c;
    }
}

```

Storage classes :

By the declaration statement the memory is allocated temporarily for all the variables. The availability of the variables for access depends on its declaration type.

The storage class specifiers are used to specify the life and scope of the variables within blocks, functions and the entire program.

There are 4 types of storage classes supported by C language. Namely

- 1) automatic variables
- 2) static variables
- 3) external (or) global variables

4)register variables

automatic variables :

These variables are declared inside a function block.If there is no storage class specifier before the declaration of any variable inside a function block, by default it takes auto storage class.

Storage : main memory

Default value : garbage value

Scope : Local to the block in which it is defined

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

Keyword : auto

Program : auto1.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
    auto int a,b;
    clrscr();
    printf("\na = %d",a);
    printf("\nb = %d",b);
    getch();
}
```

}

static variables :

The memory of static variables remains unchanged until the end of the program.

Storage : main memory

Default value : zero

Scope : Local to the block in which it is defined

Life : The value of static variable persists between different function calls , that means it cannot reinitialize between the different function calls

Keyword : static

Program : static1.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
    static int a,b;
    clrscr();
    printf("\na = %d",a);
    printf("\nb = %d",b);
```

```
    getch();  
}
```

Program : static2.c

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    void disp();  
    int i;  
    clrscr();  
    for(i=1;i<=10;i++)  
    {  
        disp();  
    }  
    getch();  
}  
void disp()  
{  
    static int n=1;  
    printf("%d\t",n);  
    n++;  
}
```

Output

```
1  2  3  4  5  6  7  8  9  10
```

Here n is static, it cannot reinitializes between different

function calls,.

Also if we declare n as automatic variable, we will get

1 1 1 1 1 1 1 1 1 1

external (or) global variables :

The variables that are both alive and active throughout the entire program are known as global variables.

Storage : main memory

Default value : zero

Scope : global

Life : As long as the program execution doesnot come to end.

Keyword : extern

Program : extern1.c

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

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

```
int a,b;
```

```
void main()
```

```
{
```

```
clrscr();
```

```
printf("\na = %d",a);
```



```
printf("\nb = %d",b);  
getch();  
}
```

Program : extern2.c

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    extern int a;  
    clrscr();  
    printf("a = %d",a);  
    getch();  
}  
int a=100;
```

Note :

Without the keyword **extern** in main() block, we will get garbage values of **a**. This is because **a** becomes local to main().

Register variables :

We can use register variables for frequently used variables to improve the faster execution of program. These variables are also declared inside a function block.

Storage : CPU register
Default value : Garbage value
Scope : Local to the block in which it is defined
Life : Till the control within the block in which it is defined
Keyword : register

Note :

It supports only integral data type (int and char)

Program : register.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
    register int a,b;
    clrscr();
    printf("\na = %d",a);
    printf("\nb = %d",b);
    getch();
}
```

Recursive Function :

Calling a function within the same function definition is called as recursive function.

If we want to work with recursive function, we must follow the following 2 aspects.

- 1) Calling by itself
- 2) Termination condition

Program : nnos_rfn.c

To display natural numbers from 1 to n using recursive function

```
#include<stdio.h>
#include<conio.h>
void main()
{
    void disp(int);
    int n;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    printf("Natural numbers from 1 to %d : \n\n",n);
    disp(n);
    getch();
}
void disp(int x)
```

```
{  
    if(x>1)  
        disp(x-1);  
    printf("%d\t",x);  
}
```

Program : fact_rfn.c

To find factorial of given number using recursive function

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    unsigned long fact(int);  
    int n;  
    unsigned long f;  
    clrscr();  
    printf("Enter a number : ");  
    scanf("%d",&n);  
    f=fact(n);  
    printf("Factorial of given Number : %lu",f);  
    getch();  
}  
unsigned long fact(int x)  
{  
    if(x<=1)
```

```
    return 1;
else
    return x*fact(x-1);
}
```

Math.h Functions :

sqrt :

Calculates the square root of a given number.

Syntax :

```
double sqrt(double x);
```

Program : sqrt.c

To find square root of given number

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    int n;
    double s;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
```

```
s=sqrt(n);  
printf("Square root of %d = %.3lf",n,s);  
getch();  
}
```

Pow :

It Calculates exponential value of given base and power.

Syntax :

```
double pow(double x,double y);
```

Here x is base and y is power.

Program : power.c

To find exponential value of given base and power

```
#include<stdio.h>  
#include<conio.h>  
#include<math.h>  
void main()  
{  
    int b,p;  
    double e;  
    printf("Enter base and power values : ");  
    scanf("%d%d",&b,&p);  
    e=pow(b,p);
```

```
printf("Exponential value = %.3lf",e);  
getch();  
}
```

floor : It rounds down the given value

Syntax : double floor(double x);

ceil : It rounds up the given value.

Syntax : double ceil(double x);

Program : **flor_ceil.c**

To find floor and ceil values of a given number

```
#include<stdio.h>  
#include<conio.h>  
#include<math.h>  
void main()  
{  
    double n,f,c;  
    clrscr();  
    printf("Enter a number : ");  
    scanf("%lf",&n);
```

```

f=floor(n);
c=ceil(n);
printf("Floor Value = %.2lf",f);
printf("\nCeil Value = %.2lf",c);
getch();
}

```

Program : triangle.c

To find area of given triangle with 3 sides

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    float a,b,c,s,area;
    printf("Enter sides of a triangle : ");
    scanf("%f%f%f",&a,&b,&c);
    if(a+b<=c || a+c<=b || b+c<=a)
    {
        printf("unable to form a triangle");
        getch();
        exit(0);
    }
    s=(a+b+c)/2;
    area=sqrt(s*(s-a)*(s-b)*(s-c));
}

```



```
printf("Area of triangle = %.2f sq units",area);  
getch();  
}
```

Program : compound.c

To enter principal amount, rate of interest and time, then calculate total amount with compound interest

```
#include<stdio.h>  
#include<conio.h>  
#include<math.h>  
void main()  
{  
    int y,m,n;  
    float p,r,i,tot;  
    clrscr();  
    printf("Enter Principal Amount      : ");  
    scanf("%f",&p);  
    printf("Enter Rate of Interest      : ");  
    scanf("%f",&r);  
    printf("Enter time(years and months) : ");  
    scanf("%d%d",&y,&m);  
    n=y*12+m;  
    tot=p*pow(1+(r/100),n);  
    i=tot-p;  
    printf("Interest      : %.2f",i);  
    printf("\nTotal Amount : %.2f",tot);  
    getch();  
}
```

}

Arrays

Array is a collection of data items (variables) of same datatype, stored in a continuous memory locations and it can be referred as a common name. A particular value in the array is accessed by using a number called index or subscript in square braces after the array name.

Types of arrays:

C supports 3 types of arrays, they are

- 1) One/single dimensional arrays
- 2) Two /double dimensional arrays
- 3) Multi-dimensional arrays

One/single dimensional array:

A group of data items can be given one variable name, using only one index, such a variable is called as single dimensional array.

Declaration:

datatype arrname[size];

Eg : int a[5];

Elements are: a [0], a [1], a [2], a [3], a[4];

Note :

In any array, the array index is from 0 to size - 1

Initialization :

optional
|
datatype arraya_name[size] = { val-1, val-2, Val-n};

Eg : int a[5]={ 1,2,3,4,5};
 (or)
 int a[]={ 1,2,3,4,5,6,7};

Program : arr1.c

To initialize 1D array and to display elements

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,a[5]={ 1,2,3,4,5};
    clrscr();
    for(i=0;i<5;i++)
    {
        printf("\n a[%d] = %d",i,a[i]);
    }
    getch();
}
```

Program : arr2.c

To accept array elements and to display the elements

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,n,a[20];
    clrscr();
    printf("Enter number of elements : ");
    scanf("%d",&n);
    if(n>20)
    {
        printf("Invalid size");
        getch();
        exit(0);
    }
    printf("\nEnter Elements : ");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    printf("\nGiven Elements : ");
    for(i=0;i<n;i++)
    {
        printf("%d\t",a[i]);
    }
    getch();
}
```

Program : arr3.c

To accept 1D array and to display maximum and minimum elements

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,n,a[20],max,min;
    clrscr();
    printf("Enter number of Elements : ");\
    scanf("%d",&n);
    printf("\nEnter Elements : ");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    max=min=a[0];
    for(i=1;i<n;i++)
    {
        if(a[i]>max)
            max=a[i];
        if(a[i]<min)
            min=a[i];
    }
    printf("\nMaximum Element : %d",max);
```

```
printf("\nMinimum Element : %d",min);  
getch();  
}
```

Program : arr4.c

To accept 1D array and to search specified element in the given array

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    int a[20],n,i,se,ck=0;  
    clrscr();  
    printf("Enter number of Elements : ");  
    scanf("%d",&n);  
    printf("\nEnter Elements \n\n");  
    for(i=0;i<n;i++)  
    {  
        printf("Enter element in a[%d] : ",i);  
        scanf("%d",&a[i]);  
    }  
    printf("\n\nEnter Element to Search : ");  
    scanf("%d",&se);  
    for(i=0;i<n;i++)  
    {  
        if(a[i]==se)
```

```

    {
        ck=1;
        printf("\n%d is found at a[%d]",se,i);
    }
}
if(ck==0)
    printf("%d is not found",se);
getch();
}

```

Program : arr5.c

To accept 1D array and to sort the given elements in ascending order/

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int a[20],n,i,j,t;
    clrscr();
    printf("Enter number of Elements : ");
    scanf("%d",&n);
    printf("\nEnter Elements : \n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
}

```

```

printf("\n\nElements Before Sorting : ");
for(i=0;i<n;i++)
{
    printf("\na[%d]=%d",i,a[i]);
}
// sorting //
for(i=0;i<n;i++)
{
    for(j=i+1;j<n;j++)
    {
        if(a[i]>a[j])
        {
            t=a[i];
            a[i]=a[j];
            a[j]=t;
        }
    }
}
printf("\n\nElements After Sorting : ");
for(i=0;i<n;i++)
{
    printf("\na[%d]=%d",i,a[i]);
}
getch();
}

```

Program : arr6.c

To accept and display 1D array using functions

```
#include<stdio.h>
#include<conio.h>
void main()
{
    void accept(int [],int);
    void disp(int [],int);
    int a[20],n;
    clrscr();
    printf("Enter number of elements : ");
    scanf("%d",&n);
    printf("Enter Elements : ");
    accept(a,n);
    printf("Given Elements : ");
    disp(a,n);
    getch();
}
void accept(int x[],int s)
{
    int i;
    for(i=0;i<s;i++)
    {
        scanf("%d",&x[i]);
    }
}
void disp(int x[],int s)
{
```

```
int i;
for(i=0;i<s;i++)
{
    printf("\t%d",x[i]);
}
}
```

Program : arr7.c

To accept 1D array and display elements in ascending order using functions

```
#include<stdio.h>
#include<conio.h>
void main()
{
    void accept(int[],int);
    void disp(int[],int);
    void sort(int[],int);
    int a[20],n;
    printf("Enter number of elements : ");
    scanf("%d",&n);
    printf("Enter elements : ");
    accept(a,n);
    printf("\nElements Before Sorting :");
```

```
    disp(a,n);
    sort(a,n);
    printf("\nElements After Sorting : ");
    disp(a,n);
    getch();
}
```

```
void accept(int a[],int n)
{
    int i;
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
}
```

```
void disp(int a[],int n)
{
    int i;
    for(i=0;i<n;i++)
    {
        printf("\t%d",a[i]);
    }
}
```

```
void sort(int a[],int n)
{
    int i,j,t;
    for(i=0;i<n-1;i++)
    {
```

```

for(j=i+1;j<n;j++)
{
    if(a[i]>a[j])
    {
        t=a[i];
        a[i]=a[j];
        a[j]=t;
    }
}
}
}

```

Program : arr8.c

To accept 1D array and search specified element using functions

```

#include<stdio.h>
#include<conio.h>
void main()
{
    void accept(int[],int);
    void disp(int[],int);
    int search(int[],int,int);
    int a[20],n,se,p;
    printf("Enter number of elements : ");
    scanf("%d",&n);
    printf("Enter elements : ");
}

```

```

accept(a,n);
printf("\nGiven Elements :");
disp(a,n);
printf("\nEnter element to search : ");
scanf("%d",&se);
p=search(a,n,se);
if(p==-1)
    printf("%d is not found",se);
else
    printf("%d is found at position %d",se,p);
getch();
}
void accept(int a[],int n)
{
    int i;
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
}
void disp(int a[],int n)
{
    int i;
    for(i=0;i<n;i++)
    {
        printf("\t%d",a[i]);
    }
}

```

```

int search(int a[],int n,int se)
{
    int i;
    for(i=0;i<n;i++)
    {
        if(a[i]==se)
            return i;
    }
    return -1;
}

```

Two (or) Double dimensional arrays :

A **Two dimensional** array can store a table of values which contains rows and columns. In this array we use 2 index values. The first index is for rows and second index is for columns.

The general form of **Two dimensional** array is :

datatype arr_name[row size][column size];

Eg : int a[3][3];

Elements are :

a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2]

Initialization :

Form : 1

optional
|
datatype arr_name[row size][col size]={value-1,.....,value-n};

Eg :

1) int a[3][3] = { 1,2,3,4,5,6,7,8,9};

2) int a[3][3]={ 1,2,3,4,5,6,7,8,9,10}; not possible

3) int a[][3] = { 1,2,3,4,5,6,7,8,9,10}; is possible

Here the rows are allocated according to the given elements and it will assign zeros for remaining values.

1	2	3
4	5	6
7	8	9
10	0	0

Form : 2

optional
|
datatype arr_name[row size][col size]={ {value-1,...,value-n},
{value-1,...,value-n},

{value-1,...,value-n} };

Eg :

1) int a[3][3]={ { 1,2,3},
{4,5,6},

{7,8,9} };

1	2	3
4	5	6
7	8	9

2) int a[3][3]={ {1},
 {4,6},
 {8,9}};

1	0	0
4	6	0
8	9	0

If the values are missing it will assign zeros in that element place.

Program : arr11.c

To initialize 2D array and display elements

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



```

{
//int a[3][3]={ 1,2,3,4,5,6,7,8,9};
int a[3][3]={ { 1,2,3},
               {4,5,6},
               {7,8,9}};

int i,j;
clrscr();
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
        printf("%3d",i,j,a[i][j]);
    }
    printf("\n");
}
getch();
}

```

Program : arr12.c

To accept 2D array and display elements

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int a[10][10],r,c,i,j;
    clrscr();

```

```
printf("Enter number of rows and columns : ");
scanf("%d%d",&r,&c);
if(r>10 || c>10)
{
    printf("Invalid Size");
    getch();
    exit(0);
}
printf("\nEnter Elements : ");
for(i=0;i<r;i++)
{
    for(j=0;j<c;j++)
    {
        scanf("%d",&a[i][j]);
    }
}
printf("\nGiven Elements : \n");
for(i=0;i<r;i++)
{
    for(j=0;j<c;j++)
    {
        printf("%3d",a[i][j]);
    }
    printf("\n");
}
getch();
}
```

Program : arr13.c

To accept 2 dimensional array and display elements using functions

```
#include<stdio.h>
#include<conio.h>
void main()
{
    void accept(int [][][10],int,int);
    void disp(int [][][10],int,int);
    int a[10][10],r,c;
    clrscr();
    printf("Enter number of rows and columns : ");
    scanf("%d%d",&r,&c);
    printf("\nEnter Elements : ");
    accept(a,r,c);
    printf("\nGiven Elements : ");
    disp(a,r,c);
    getch();
}
void accept(int a[][][10],int r,int c)
{
    int i,j;
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
```

```

        scanf("%d",&a[i][j]);
    }
}
}
void disp(int a[][10],int r,int c)
{
    int i,j;
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            printf("%3d",a[i][j]);
        }
        printf("\n");
    }
}

```

Program : arr14.c

Addition of two Matrices using arrays and functions

```

#include<stdio.h>
#include<conio.h>
void main()
{
    void accept(int [][][10],int,int);
    void disp(int [][][10],int,int);
    void addt(int [][][10],int[][10],int[][10],int,int);
}

```

```

int a[10][10],b[10][10],add[10][10],r,c;
clrscr();
printf("Enter number of rows and columns : ");
scanf("%d%d",&r,&c);
printf("\nEnter First Matrix Elements   : ");
accept(a,r,c);
printf("\nEnter Second Matrix Elements   : ");
accept(b,r,c);
printf("\nElements of Given First Matrix : \n");
disp(a,r,c);
printf("\nElements of Given Second Matrix : \n");
disp(b,r,c);
addt(a,b,add,r,c);
printf("\n Addition of given 2 Matrices   :\n");
disp(add,r,c);
getch();
}
void accept(int a[][10],int r,int c)
{
    int i,j;
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
}
}

```

```

void disp(int a[][10],int r,int c)
{
    int i,j;
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            printf("%3d",a[i][j]);
        }
        printf("\n");
    }
}

void addt(int a[][10],int b[][10],int add[][10],int r,int c)
{
    int i,j;
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            add[i][j]=a[i][j]+b[i][j];
        }
    }
}

```

pogram : arr15.c

Multiplicaton of two Matrices using arrays and functions

```

#include<stdio.h>
#include<conio.h>
void main()
{
    void accept(int [][][10],int,int);
    void disp(int [][][10],int,int);
    void mul(int [][][10],int[][][10],int[][][10],int,int,int);
    int a[10][10],b[10][10],c[10][10],m,n,p,q;
    clrscr();
    printf("Enter number of rows and columns of First Matrix
: ");
    scanf("%d%d",&m,&n);
    printf("Enter number of rows and columns of Second
Matrix : ");
    scanf("%d%d",&p,&q);
    if(n!=p)
    {
        printf("\nMatrix Multiplication is not Possible");
        getch();
        exit(0);
    }
    printf("\nEnter First Matrix Elements    : ");
    accept(a,m,n);
    printf("\nEnter Second Matrix Elements    : ");
    accept(b,p,q);
    printf("\nElements of Given First Matrix : \n");

```

```

    disp(a,m,n);
    printf("\nElements of Given Second Matrix : \n");
    disp(b,p,q);
    mul(a,b,c,m,n,q);
    printf("\nMultiplication of given Matrices :\n");
    disp(c,m,q);
    getch();
}
void accept(int a[][10],int r,int c)
{
    int i,j;
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
}
void disp(int a[][10],int r,int c)
{
    int i,j;
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            printf("%3d",a[i][j]);
        }
    }
}

```



```

    printf("\n");
}
}
void mul(int a[][10],int b[][10],int c[][10],int m,int n,int q)
{
    int i,j,k;
    for(i=0;i<m;i++)
    {
        for(j=0;j<q;j++)
        {
            c[i][j]=0;
            for(k=0;k<n;k++)
            {
                c[i][j]=c[i][j]+(a[i][k]*b[k][j]);
            }
        }
    }
}

```

Program : arr17.c

To find Transpose of the given matrix

```

#include<stdio.h>
#include<conio.h>
void main()
{
    void accept(int[][10],int,int);

```

```

void disp(int [][][10],int,int);
void trans(int [][][10],int[][10],int,int);
int a[10][10],at[10][10],r,c;
clrscr();
printf("Enter number of rows and columns : ");
scanf("%d%d",&r,&c);
printf("\nEnter Elements : ");
accept(a,r,c);
printf("\nGiven Elements : \n");
disp(a,r,c);
trans(a,at,r,c);
printf("Transpose Matrix : \n");
disp(at,c,r);
getch();
}
void accept(int a[][10],int r,int c)
{
    int i,j;
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
}
void disp(int a[][10],int r,int c)
{

```

```

int i,j;
for(i=0;i<r;i++)
{
    for(j=0;j<c;j++)
    {
        printf("%3d",a[i][j]);
    }
    printf("\n");
}
}
void trans(int a[][10],int at[][10],int r,int c)
{
    int i,j;
    /* for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            at[j][i]=a[i][j];
        }
    }
    */
    for(i=0;i<c;i++)
    {
        for(j=0;j<r;j++)
        {
            at[i][j]=a[j][i];
        }
    }
}

```

}

Multi dimensional array :

C supports arrays of 3 or more dimensions. In Multi dimensional arrays we use more than 2 index values.

Syntax :

Datatype arr_name [s1][s2].....[sn];

Eg : int a[2][2][3];

Elements :

a[0][0][0]	a[0][0][1]	a[0][0][2]
a[0][1][0]	a[0][1][1]	a[0][1][2]
a[1][0][0]	a[1][0][1]	a[1][0][2]
a[1][1][0]	a[1][1][1]	a[1][1][2]

Program : arr18.c

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

```

#include<conio.h>
void main()
{
    int a[2][2][3],i,j,k;
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
        {
            for(k=0;k<3;k++)
            {
                printf("Enter Value into a[%d][%d][%d] : ",i,j,k);
                scanf("%d",&a[i][j][k]);
            }
        }
    }
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
        {
            for(k=0;k<3;k++)
            {
                printf("\nGiven Value in a[%d][%d][%d] = %d",i,j,k,
                    a[i][j][k]);
            }
        }
    }
    getch();
}

```

Strings

A group of characters defined between double quotation marks is a string. It is a constant string ,But in C language, a string variable is nothing but an array of characters and terminated by a null character (\0).

Declaration :

char identifier [size] ;

Eg : char st[20] ;

Initialization :

At the time of declaring a string variable we can store a constant string into that variable is called as initialization of a string.

syntax:

optional

|

char identifier[size] = "string";

Eg : char st[10] = "WELCOME";

char str[] = "ABCD" ;

Note :

- 1.The compiler assigns a constant string to a string variable, it automatically supplies a null character (\0)at the end of a string. Therefore the size should be equal to number of characters in the string plus 1.
- 2.In C language, string initialization is possible, but string assigning is not possible.

Program : str1.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char st[10]="WELCOME";
    char str[]="ABCD";
    clrscr();
    printf("st = %s",st);
    printf("\nstr = %s",str);
    getch();
}
```

gets :

It gets a string from stdin(Keyboard).

Syntax :

```
char * gets(char *s);
```

Eg :

- 1) gets(st);
- 2) gets("welcome"); -> not possible

puts :

It outputs a string to stdout(Monitor) and appends a new line character.

Syntax :

```
char * puts(const char *s);
```

Eg :

- 1) puts(st);
- 2) puts("welcome"); -> possible

Program : str2.c


```
#include<stdio.h>
#include<conio.h>
void main()
{
    char st[20];
    clrscr();
    puts("Enter a String : ");
    gets(st);
    puts("Given String : ");
    puts(st);
    getch();
}
```

getch:

It gets a character from the console(keyboard) but doesnot echo (print) to the screen.

Syntax : int getch();

getche :

It gets a character from the console and echoes to the screen.

Syntax : int getche();

Note :

No need to press enter key after entering a character.

Program : str3.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    char che;
    clrscr();
    printf("Enter first character : ");
    ch=getch();
    printf("Enter second character : ");
    che=getche();
    printf("First character : %c"ch);
    printf("Second character : %c"che);
    getch();
}
```

getchar :

It is a macro that gets a character from stdin(keyboard).

Syntax : int getchar();

putchar :

It is a macro that outputs a character on stdout(monitor).

Syntax : int putchar(int c);

Program : str4.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    clrscr();
    printf("Enter a character : ");
    ch=getchar();
    printf("Given character : ");
    putchar(ch);
```

```
    getch();  
}
```

String handling functions :

These are included in the header file
<string.h>

strlen :

It calculates length of given string.

Syntax :

```
size_t  strlen(const char *s);
```

Program : str5.c

```
#include<stdio.h>  
#include<conio.h>  
#include<string.h>  
void main()
```

```
{  
    int len;  
    char s[20];  
    clrscr();  
    printf("Enter a string : ");  
    gets(s);  
    len=strlen(s);  
    printf("Length of given string : %d",len);  
    getch();  
}
```

strcpy :

It copies one string to another .

Syntax :

```
char * strcpy(char *dest, const char *src);
```

Program : str6.c

```
#include<stdio.h>  
#include<conio.h>  
#include<string.h>  
void main()
```

```
{  
    int len;  
    char s1[20],s2[20];  
    clrscr();  
    printf("Enter a string : ");  
    gets(s1);  
    strcpy(s2,s1);  
    printf("\nGiven string : %s",s1);  
    printf("\nCopied String : %s",s2);  
    getch();  
}
```

strrev :

It reverses all characters in the given string except for the terminating null(\0).

Syntax :

char * strrev(char *s);

Program : str7.c

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

```
void main()
{
    char s[80];
    clrscr();
    printf("Enter a string : ");
    gets(s);
    printf("Given string   : %s",s);
    strrev(s);
    printf("\nReverse string : %s",s);
    getch();
}
```

strcat :

It appends one string to another.

Syntax :

```
char * strcat(char *dest, const char *src);
```

Program : str8.c

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

```
void main()
{
    char s1[80],s2[80];
    clrscr();
    printf("Enter first string  : ");
    gets(s1);
    printf("Enter second string : ");
    gets(s2);
    strcat(s1,s2);
    printf("\nConcatenation of 2 strings : %s",s1);
    getch();
}
```

strupr :

It converts lower case(a to z) letters in the given string to upper case(A to Z).

Syntax : char * strupr(char *s);

strlwr :

It converts upper case(A to Z) letters in the given string to lower case(a to z).

Syntax : char * strlwr(char *s);

Program : str9.c

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
    char s[80];
    clrscr();
    printf("Enter a string : ");
    gets(s);
    printf("Given string   : %s",s);
    strlwr(s);
    printf("\nGiven string in lower case: %s",s);
   strupr(s);
    printf("\nGiven string in upper case: %s",s);
    getch();
}
```

strcmp : It compares 2 strings with case

sensitivity.

Syntax :

```
int strcmp(const char *s1, const char *s2);
```

stricmp :

It compares 2 strings without case sensitivity.

syntax:

```
int stricmp(const char *s1, const char *s2);
```

strcmpi :

It is a macro which compares 2 strings without case sensitivity.

Syntax :

```
int strcmpi(const char * s1, const char * s2);
```

Return value :

These routines(functions) return an int value, that is

< 0 -> s1 < s2

== 0 -> s1 == s2

> 0 -> s1 > s2

Program : str10.c

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
    char s1[80],s2[80];
    int n;
    clrscr();
    printf("Enter first string : ");
    gets(s1);
    printf("Enter second string : ");
    gets(s2);
    // n=strcmp(s1,s2);
    // n=stricmp(s1,s2);
    n=strcmpi(s1,s2);
    if(n==0)
        printf("\n2 strings are equal");
    if(n<0)
        printf("\nFirst string is less than Second string");
    if(n>0)
        printf("\nFirst string is greater than Second string");
    getch();
}
```

Program : str11.c

To check weather the given string is Plindrome

or Not

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
    char s1[80],s2[80];
    int n;
    clrscr();
    printf("Enter a string : ");
    gets(s1);
    strcpy(s2,s1);
    strrev(s2);
    n=strcmp(s1,s2);
    if(n==0)
        printf("\nGiven string is a Palindrome");
    else
        printf("\nGiven string is not a Palindrome");
    getch();
}
```

Program : str12.c

To find character frequency of given String

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

```

{
    static int a[256],i;
    char st[80];
    clrscr();
    printf("Enter a String : ");
    gets(st);
    for(i=0;i<strlen(st);i++)
    {
        a[st[i]]=a[st[i]]+1;
    }
    printf("\nCharacter  Frequency");
    printf("\n-----\n");
    for(i=0;i<256;i++)
    {
        if(a[i]!=0)
            printf("\n%c          %d",i,a[i]);
    }
    getch();
}

```

Two dimensional character array (String array) :

A list of strings can be treated as a table of

strings and a two dimensional character array can be used to store the entire list.

For example :

char st[20][20] may be used to store a list of 20 strings, each of length not more than 20 characters.

Program : strarr1.c

To enter group of strings and to display them

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char st[20][20];
    int n,i;
    clrscr();
    printf("Enter number of strings : ");
    scanf("%d",&n);
```

```

printf("Enter %d strings : ",n);
fflush(stdin);
for(i=0;i<n;i++)
{
    gets(st[i]);
}
printf("\nGiven strings : ");
for(i=0;i<n;i++)
{
    puts(st[i]);
}
getch();
}

```

Program : strarr2.c

To enter group of strings and to display them in alphabetical order

```

#include<stdio.h>
#include<conio.h>
void main()
{
    char st[20][20],t[20];

```

```
int n,i,j,k;
clrscr();
printf("Enter number of strings : ");
scanf("%d",&n);
printf("Enter %d strings : \n",n);
fflush(stdin);
for(i=0;i<n;i++)
{
    gets(st[i]);
}
printf("\nGiven strings : \n");
for(i=0;i<n;i++)
{
    puts(st[i]);
}
// sorting //
for(i=0;i<n-1;i++)
{
    for(j=i+1;j<n;j++)
    {
        k=strcmp(st[i],st[j]);
        if(k>0)
        {
            strcpy(t,st[i]);
```



```

        strcpy(st[i],st[j]);
        strcpy(st[j],t);
    }
}
}
printf("\nGiven strings in Alphabetical order :
\n");
for(i=0;i<n;i++)
{
    puts(st[i]);
}
getch();
}

```

Preprocessor statements (or) Preprocessor directives

The processor is a program that process the source code before it passes through the compiler. The commands used to control the pre-processor are known as preprocessor directives.

These directives are divided into 3 categories.

1. File inclusion directives
2. Macro substitution directives
3. Compiler control directives

File inclusion directive:

#include:

It is a pre-processor file inclusion directive and is used to include header files. It provides instructions to the compiler to link the functions from the 'C' library.

Syntax:

#include "file name"

(or)

#include < file name >

When the file name is included within " ", the search for the file is made first the current directory and then the standard directories (software directory). Otherwise the file name is included within < >, the file is searched only in the standard directories.

Macro substitution directives:

Macro is a process where an identifier in a program is replaced by a predefined string or value.

#define :

It is a preprocessor statement and is used to define macros.

Syntax:

Form 1 : (simple macro)

#define identifier predefined_string (or) value

eg:

#define pf printf

#define sf scanf

#define val 100

Program : sim_mac.c

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

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

```
#define pf printf
```

```
#define sf scanf
```

```
#define vm void main
```

```
#define cls clrscr
```

```
#define gt getch
```

```

#define val 100
vm()
{
    int n;
    cls();
    pf("Enter a number : ");
    sf("%d",&n);
    pf("Given Number = %d",n);
    p("\nval = %d",val);
    gt();
}

```

Form 2 :
(complex macro or macro with arguments)

Syntax:

#define identifier(arg-1,arg-2,.....arg-n) definition

Eg:
#define square(x) x*x

Program : comx_mac.c

```

#include<stdio.h>
#include<conio.h>
#define square(x)  x*x

int square(int x)
{
    return x*x;
}

void main()
{
    int n,s;
    clrscr();
    printf("Enter a number : ");
    scanf("%d",&n);
    s=square(n);
    printf("Square of given number = %d",s);
    getch();
}

```

Differences between function and macro :

Function	Macro
1)It is a self contained block of statements.	1)It is a preprocessor statement.

2)It replaces its return value

3)We can use only specified data types.

4)Execution speed is less.

2)It replaces its definition.

3)Data types are generic.

4)Execution speed is more

Program : macr_def.c

```
#include<stdio.h>
#include<conio.h>
#define square(x) x*x
void main()
{
    int n;
    n=100/square(5);
    printf("n = %d",n);
    getch();
}
```

Compiler control directives:

These directives are used to control the compiler. The following compiler control directives are used in C.

- 1 #if
- 2 #else
- 3 #elif
- 4 #endif etc.

Program : comp_dir.c

```
#include<stdio.h>
#include<conio.h>
#define n 10
#if n<=10
    #define val 100
#else
    #define val 200
#endif
void main()
{
    clrscr();
```

```
printf("Val = %d",val);  
getch();  
}
```

Pointers

C provides the important feature of data manipulations with the address of the variables. Hence the execution time is very much reduced. Such concept is possible with the special data type called pointers.

Definition :

A pointer is a variable which stores the address of another variable.

Declaration :

Pointer declaration is similar to normal variable but preceded by * symbol.

syntax:

datatype *identifier;

eg:

int *p;

char *ch1,*ch2;

int x,y,z,*p;

Initialization :

At the time of declaring a pointer, we can store some address into that pointer is called as initialization of

pointer.

Syntax :

```
datatype *identifier = address;
```

Example :

```
int a;  
int *p = &a;
```

Assigning Address to a pointer:

```
datatype *identifier;  
identifier=Address;
```

eg:

```
int *p;  
int a;  
p=&a;
```

void * :

It is a generic pointer, it refers the address of any type of variable and also it will convert to any type of pointer.

eg:

```
void *p;
```

NULL :

Null pointer value (empty address)

Note :

Any type of pointer, it allocates only 2 bytes of memory ,Because it stores the address of memory location. In C language the memory address is in unsigned integer. (2 bytes for unsigned int, in the range of 0 to 65535)

Program : ptr1.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int *p1;
    char *p2;
    float *p3;
    double *p4;
    clrscr();
    printf("size of int pointer  = %d bytes",sizeof(p1));
    printf("\nsize of char pointer  = %d bytes",sizeof(p2));
    printf("\nsize of float pointer = %d bytes",sizeof(p3));
    printf("\nsize of double pointer = %d bytes",sizeof(p4));
    getch();
}
```

Accessing pointers :

```
int a=100;  
int *p=&a;      (or)
```

```
int a=100;  
int *p;  
p = &a;
```

```
a - 100  
&a - 5000  
a = 200
```

```
*p - 100  
p - 5000  
*p=200
```

Program : ptr2.c

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    int a=100;  
    int *p=&a;  
    clrscr();
```

```
printf("value of a = %d",a);  
printf("\naddress of a = %lu",&a);  
printf("\nvalue of a using p = %d",*p);  
printf("\naddress of a using p = %lu",p);  
a=200;  
printf("\n *p = %d",*p);  
*p=300;  
printf("\n a = %d",a);  
getch();  
}
```

Pointers and Functions :

Calling mechanism of argument function (or) Parameter passing techniques

- 1)Call by value (or) Pass by value
- 2)Call by reference (or) Pass by reference.

Call by value :

The process of passing the value to the function call is known as call by value.

Program : fn1.c

Passing constant value

```
#include<stdio.h>
#include<conio.h>
void main()
{
    void fun(int);
    clrscr();
    fun(100);
    getch();
}
void fun(int x)
{
    printf("%d",x);
}
```

Program : fn2.c

Passing a variable

```
#include<stdio.h>
#include<conio.h>
void main()
{
    void fun(int);
    int n;
    clrscr();
    printf("Enter any value : ");
    scanf("%d",&n);
```

```
    fun(n);
    getch();
}
void fun(int x)
{
    printf("Given value = %d",x);
}
```

Program : fn3.c

Passing an expression

```
#include<stdio.h>
#include<conio.h>
void main()
{
    void fun(int);
    int a,b;
    clrscr();
    printf("Enter any two numbers : ");
    scanf("%d%d",&a,&b);
    fun(a+b);
    getch();
}
void fun(int x)
{
    printf("Sum = %d",x);
}
```

Call by reference :

The process of calling a function using pointers to pass the address of the variables is called as call by reference.

Program : ptrfn1.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
    void setdata(int *);
    int a;

    clrscr();
    setdata(&a);
    printf("a=%d",a);
    getch();
}
void setdata(int *p)
{
    *p=100;
}
```

Program : ptrfn2.c

To accept a value and display it using call by reference

```
#include<stdio.h>
#include<conio.h>
void main()
{
    void accept(int *);
    int n;
    clrscr();
    accept(&n);
    printf("Given value = %d",n);
    getch();
}
void accept(int *p)
{
    printf("Enter a value : ");
    scanf("%d",p);
}
```

Program : ptrfn3.c

Swaping of 2 values using call by reference

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



```

#include<conio.h>
void main()
{
    void swap(int *,int *);
    int a,b;
    clrscr();
    printf("Enter 2 values : ");
    scanf("%d%d",&a,&b);
    printf("\nGiven values before swaping : ");
    printf("\na=%d",a);
    printf("\nb=%d",b);
    printf("\n\nGiven values after swaping : ");
    swap(&a,&b);
    printf("\na=%d",a);
    printf("\nb=%d",b);
    getch();
}
void swap(int *x,int *y)
{
    int t;
    t=*x;
    *x=*y;
    *y=t;
}

```

Pointers and Arrays :

When an array is declared, the compiler allocates a base

address and sufficient amount of storage to contain all the elements of array in continuous memory locations. The base address is the location of the first element (index = 0) of the array. The compiler also defines the array name as a constant pointer to the first element.

Program : ptrarr1.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[5];
    clrscr();
    printf("Base address : %u",&a[0]);
    printf("\nBase address : %u",a);
    getch();
}
```

Suppose we declare an array *a* as follows :-

```
int a[5] = {1,2,3,4,5};
```

Suppose the base address of a=1000 and each integer requires 2 bytes of memory,the 5 elements will be stored as follows :-

The name 'a' is defined as a constant pointer pointing to the first element $a[0]$ and therefore the value of a is 1000, the location where $a[0]$ is stored.

Hence $a = \&a[0] = 1000$

If we declare p as an integer pointer, then we can make the pointer p to point to the array a by the following assignment.

```
int *p;
```

```
p = a ; (or) p = &a[0];
```

Now we can access every element of a using $p++$ or $p+0$, $p+1$, to move one element to another. The relationship between p and a is shown below.

$$p+0 = \&a[0] = 1000$$
$$p+1 = \&a[1] = 1002$$
$$p+2 = \&a[2] = 1004$$
$$p+3 = \&a[3] = 1006$$
$$p+4 = \&a[4] = 1008$$

When handling arrays instead of using array index we can use pointers to access array elements.

Note that

$*(p+i)$ gives the value of $a[i]$.

To get the values of array elements :

$*(p+0) = a[0] = 1$

$*(p+1) = a[1] = 2$

$*(p+2) = a[2] = 3$

$*(p+3) = a[3] = 4$

$*(p+4) = a[4] = 5$

Note :

The pointer accessing method is much faster than array indexing.

Program : ptrarr2.c

To accept Single dimension array and display array elements using pointer accessing method

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

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

```
void main()
```

```

{
    int a[20],n,i,*p;
    p=a;    //p=&a[0];
    clrscr();
    printf("Enter number of elements : ");
    scanf("%d",&n);
    printf("Enter array elements : ");
    for(i=0;i<n;i++)
    {
        scanf("%d",(p+i));

    }
    printf("\nGiven array elements : ");
    for(i=0;i<n;i++)
    {
        printf("%d\t",*(p+i));

    }
    getch();
}

```

Pointers and Strings

A pointer can also point to a string .

Program : ptrstr1.c

```

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

```

```
void main()
{
    char *st;
    clrscr();
    printf("Enter a String : ");
    gets(st);
    printf("Given String : %s",st);
    getch();
}
```

String copying using Pointers :

Program : ptrstr2.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char *st1,*st2;
    clrscr();
    printf("Enter a String : ");
    gets(st1);
    st2=st1;
    printf("Given String   : %s",st1);
    printf("\nCopied String : %s",st2);
    getch();
}
```

Dynamic Memory Allocation :

C language requires the number of elements in an array to be specified at compile time. But we may not be able to do so always. Our initial judgement of size, if it is wrong may cause failure of program or wastage of memory space. At this situation we can use **Dynamic Memory Allocation**.

Definition :

The process of allocating memory at run time is known as **Dynamic Memory Allocation**.

malloc : <alloc.h>

It is used to allocate memory at runtime.

Syntax :

```
void * malloc(size_t size);
```

size_t : It is the data type used for memory object size. Also it is another name for unsigned integer.

```
pointer = (typecast) malloc(memory size);
```

Eg :

```
int *p;
```

```
p = (int *) malloc(sizeof(int)) ;           // 1 location
```

```
p = (int *) malloc(5 * sizeof(int)) ; // 5 locations
```

It allocates specified number of locations in the heap(main memory) and initialized to garbage values.

calloc : <alloc.h>

It is also used to allocate memory at runtime.

Syntax :

```
void * calloc(size_t nitems , size_t size);  
pointer = (typecast) calloc(number of locations, size of each location);
```

Eg :

```
int *p;  
p = (int *) calloc(1, sizeof(int)) ; // 1 location  
p = (int *) calloc(5, sizeof(int)) ; // 5 locations
```

It allocates specified number of locations in the heap(main memory) and initialized to zeroes.

free : <alloc.h>

It frees the allocated blocks.

Syntax :

```
void free(void * block);
```


Program : dma1.c

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
void main()
{
    int *p1,*p2,i;
    p1=(int *)malloc(5*sizeof(int));
    p2=(int *)calloc(5,sizeof(int));
    clrscr();
    printf("MALLOC : ");
    for(i=0;i<5;i++)
    {
        printf("%d\t",*(p1+i));
    }
    printf("\nCALLOC : ");
    for(i=0;i<5;i++)
    {
        printf("%d\t",*(p2+i));
    }
    free(p1);
    free(p2);
    getch();
}
```

Program : dma2.c

```

#include<stdio.h>
#include<conio.h>
#include<alloc.h>
void main()
{
    int *p,n,i;
    clrscr();
    printf("Enter number of elements : ");
    scanf("%d",&n);
    p=(int *)malloc(n*sizeof(int));
    //p=(int *)calloc(n,sizeof(int));
    printf("Enter elements : ");
    for(i=0;i<n;i++)
    {
        scanf("%d",(p+i));
    }
    printf("\nGiven elements : ");
    for(i=0;i<n;i++)
    {
        printf("%d\t",*(p+i));
    }
    free(p);
    getch();
}

```

realloc : <alloc.h>

It reallocates main memory at run time.

Syntax :

```
void * realloc(void * block, size_t size);
```

realloc adjusts the size of allocated block to size, copying the contents to a new location.

Program : realloc1.c

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<string.h>
void main()
{
    char *str;
    clrscr();
    str=(char *)malloc(10);
    strcpy(str,"Hello ");
    printf("Given String : %s",str);
    str=(char *)realloc(str,25);
    strcat(str," Welcome to BDPS");
    printf("\nNew String  : %s",str);
    free(str);
    getch();
}
```

Structures

We have seen that arrays can be used to represent a group of data items that belongs to same data type. If we want to represent a collection of data items of different data types using a single name, then we cannot use an array. C supports a constructed data type known as Structure, which is a method for packing data of different data types.

A structure is a convenient tool for handling a group of logically related data items. Structures help to organize complex data in a more meaningful way. It is a powerful concept that we may often need to use in our program design.

Definition :

A group of data items that belongs to different data types is known as Structure.

struct : It is a keyword and is used to declare a Structure.

Declaration :

```
struct struct_name
{
    data item-1;
    data item-2;
    .....
    .....
    data item-n;
```

}[var_list];

Eg :

```
struct emp
{
    int eno;
    char ename[20];
    float sal;
}e; (or) e1,e2,...,en;
```

Declaration of structure variable :

```
struct struct_name identifier;
      (or)
struct struct_name identifier-1,identifier-2,.....,identifier-n;
```

Eg:

```
struct emp e;
      (or)
struct emp e1,e2,...,en;
```

. (Access operator):

It is used to access the data items of a structure with the help of structure variable.

Syntax :

```
struct_variable . dataitem;
```

Eg:

```
e.eno;
e.ename;
```

e.sal;

Initialization :

```
struct    struct_name
{
    data item-1;
    data item-2;
    .....
    .....
    data item-n;
}identifier={value-1,value-2,...,value-n};
```

Eg :

```
struct emp
{
    int eno;
    char ename[20];
    float sal;
} e = {100,"HHHH",20000.00};
```

```
struct struct_name identifier = {value-1,value-2,.....value-n};
struct emp e = {100,"HHHH",20000.00};
```

Program : struct1.c

To initialize a structure and display its data

```
#include<stdio.h>
#include<conio.h>
struct emp
{
    int eno;
    char ename[20];
    float sal;
} e={ 1,"HHHH",10000.00};

void main()
{
    /* struct emp e={ 1,"HHHH",90000}; */
    clrscr();
    printf("\nEmp number : %d",e.eno);
    printf("\nEmp name    : %s",e.ename);
    printf("\nSalary       : %.2f",e.sal);
    getch();
}
```

Program : struct2.c

To accept a structure and display the data

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

```

#include<conio.h>
struct emp
{
    int eno;
    char ename[20];
    float sal;
};
void main()
{
    struct emp e;
    clrscr();
    printf("Enter emp number : ");
    scanf("%d",&e.eno);
    printf("Enter emp name   : ");
    fflush(stdin);
    gets(e.ename);
    printf("Enter salary       : ");
    scanf("%f",&e.sal);
    printf("\n\nGiven Employ data ");
    printf("\n-----");
    printf("\nEmp number : %d",e.eno);
    printf("\nEmp name   : %s",e.ename);
    printf("\nSalary      : %.2f",e.sal);
    getch();
}

```

Program : struct3.c

To accept student details and to calculate and display result

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

```
struct student
{
    int sno,c,cpp,java,tot;
    char sname[20],res[5],gr;
    float avg;
};
```

```
void main()
{
    struct student s;
    clrscr();
    printf("Enter Student Number :");
    scanf("%d",&s.sno);
    printf("Enter Student Name :");
    fflush(stdin);
    gets(s.sname);
    printf("Enter Marks in C,C++ and Java :");
    scanf("%d%d%d",&s.c,&s.cpp,&s.java);
    s.tot=s.c+s.cpp+s.java;
    s.avg=(float)s.tot/3;
    if(s.c>=50 && s.cpp>=50 && s.java>=50)
    {
```

```

    strcpy(s.res,"Pass");
    if(s.avg>=60)
        s.gr='A';
    else
        s.gr='B';
    }
else
{
    strcpy(s.res,"Fail");
    s.gr='-';
}
clrscr();
printf("\tStudent Result");
printf("\n-----");
printf("\nStudent Number    :%d",s.sno);
printf("\nStudent Name       :%s",s.sname);
printf("\nMarks in C           :%d",s.c);
printf("\nMarks in C++          :%d",s.cpp);
printf("\nMarks in Java          :%d",s.java);
printf("\nTotal Marks           :%d",s.tot);
printf("\nAverage               :%.2f",s.avg);
printf("\nResult                :%s",s.res);
printf("\nGrade                  :%c",s.gr);
printf("\n-----");
getch();
}

```

Array of Structures :

Like any other data type, structure arrays can be defined, so that each array element can be of structure data type.

For Example,

```
struct student s[100];
```

which defines an array called *s*, that contains 100 elements, each element is defined to be of type **struct student**.

program : struct_ar1.c

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

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

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

```
struct student
```

```
{
```

```
int sno,c,cpp,java,tot;
```

```
char sname[20],res[10],gr;
```

```
float avg;
```

```
};
```

```
void main()
```

```
{
```

```
struct student s[100];
```

```
int id=0,i;
```

```
char ch;
```

```
do
{
    clrscr();
    printf("Enter Student Number :");
    scanf("%d",&s[id].sno);
    printf("Enter Student Name :");
    fflush(stdin);
    gets(s[id].sname);
    printf("Enter Marks in C,Cpp and Java :");
    scanf("%d%d%d",&s[id].c,&s[id].cpp,&s[id].java);
    s[id].tot=s[id].c+s[id].cpp+s[id].java;
    s[id].avg=(float)s[id].tot/3;
    if(s[id].c>=50 && s[id].cpp>=50 && s[id].java>=50)
    {
        strcpy(s[id].res,"Pass");
        if(s[id].avg>=60)
            s[id].gr='A';
        else
            s[id].gr='B';
    }
    else
    {
        strcpy(s[id].res,"Fail");
        s[id].gr='-';
    }
    id++;
    printf("Enter Another Record(y/n) :");
    fflush(stdin);
```

```

scanf("%c",&ch);
}while(ch!='n' && id<100);

for(i=0;i<id;i++)
{
clrscr();
printf("\nRecord Number :%d",i+1);
printf("\nStudent Number   :%d",s[i].sno);
printf("\nStudent Name      :%s",s[i].sname);
printf("\nMarks in C         :%d",s[i].c);
printf("\nMarks in Cpp        :%d",s[i].cpp);
printf("\nMarks in Java        :%d",s[i].java);
printf("\nTotal Marks         :%d",s[i].tot);
printf("\nAverage             :%.2f",s[i].avg);
printf("\nResult               :%s",s[i].res);
printf("\nGrade                 :%c",s[i].gr);
printf("\n\nPress Any Key.....");
getch();
}
}

```

Passing structure as function argument(or) parameter :

Like any other data type a structure may also be used as function argument.

Returning structure :

A function can, not only receive a structure as its argument, but also can return them.

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

```
struct num
{
    int n;
};
```

```
struct num accept()
{
    struct num x;
    printf("Enter any Number :");
    scanf("%d",&x.n);
    return x;
}
```

```
void disp(struct num x)
{
    printf("Given Number :%d",x.n);
}
```

```
void main()
{
    struct num nb;
    clrscr();
    nb=accept(); /* Returning structure */
    disp(nb);    /* Passing structure */
    getch();
}
```

Pointers and Structures :

A pointer can also point to a structure.

For example :

```
struct student
{
    int sno;
    char sname[20], course[20];
    float fee;
};
```

```
struct student s;
struct student *p;
```

Here p is defined to be a pointer, pointing to student structure,

we can write

```
p = &s;
```

After making such an assignment we can access every data item of student structure through **p**.

→(Arrow)

-> is the combination of - followed by > .

It is used to access the data items of a structure with the help of structure pointer.

syntax:

Struct_pointer -> data item;

Eg:

p→ sno;

p→ sname;

p→ course;

p→ fee;

program

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

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

```
struct student
```

```
{
```

```
int sno;
```

```
char sname[20],course[20];
```

```
float fee;
```

```
};
```

```
void main()
```

```
{
```

```
struct student s;
```

```
struct student *p;
```

```
float *f,f1;
```

```
f=&f1;
```

```
*f=f1;
```



```

clrscr();
p=&s;
printf("Enter Studet Number :");
scanf("%d",&p->sno);
printf("Enter Studet Name :");
fflush(stdin);
gets(p->sname);
printf("Enter Course  :");
gets(p->course);
printf("Enter Fees :");
scanf("%f",&p->fee);
printf("\nStudent Number  :%d",p->sno);
printf("\nStudent Name    :%s",p->sname);
printf("\nCourse         :%s",p->course);
printf("\nFees           :%.2f",p->fee);
getch();
}

```

structures and dynamic memory allocation

```

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

```

```

struct student
{
    int sno;

```

```
char sname[20],course[20];  
float fee;  
};
```

```
void main()
```

```
{  
    struct student *p;
```

```
  
    float *f,f1;  
    f=&f1;  
    *f=f1;
```

```
  
    p=(struct student *)malloc(sizeof(struct student));  
    clrscr();  
    printf("Enter Studet Number :");  
    scanf("%d",&p->sno);  
    printf("Enter Studet Name :");  
    fflush(stdin);  
    gets(p->sname);  
    printf("Enter Course  :");  
    gets(p->course);  
    printf("Enter Fee  :");  
    scanf("%f",&p->fee);  
    printf("\nStudent Number  :%d",p->sno);  
    printf("\nStudent Name    :%s",p->sname);  
    printf("\nCourse          :%s",p->course);  
    printf("\nFee              :%.2f",p->fee);  
    free(p);
```

```
    getch();  
}
```

Pre defined structures :

1. date :

It is Pre-defined structure and is used to get current system date. It is included in the header file <DOS.H>

Syntax :

```
struct date  
{  
    int da_year;    // current year  
    char da_day;    // day of month  
    char da_mon;    // month (Jan=1)  
};
```

getdate :

It gets the current system date.

Syntax : void getdate(struct date *d);

2. time :

It is a pre-defined structure and is used to get current system time. It is included in the header file <DOS.H>

Syntax :

```
struct time  
{
```

```
    unsigned char ti_min;    // minutes
    unsigned char ti_hour;  // hours
    unsigned char ti_sec;    // seconds
};
```

gettime :

It gets the current system time.

Syntax :

```
void gettime(struct time *t);
```

Union:

A union is similar to struct, except it allows us to define variables that share storage space.

Syntax :

```
union union_name
{
    data item1;
    data item2;
    .....
    data item-n;
} [var_list];
```

Eg :

```
union test
{
    int n;
```

```
        char ch;  
        float ft;  
    } t;
```

Declaration of union variable in a separate statement

```
union  union_name identifier;
```

eg:

```
    union test t;
```

Program : un_stru.c

```
#include<stdio.h>  
#include<conio.h>  
struct test1  
{  
    int n;  
    char c;  
    float f;  
}t1;  
union test2  
{  
    int n;  
    char c;
```

```

float f;
}t2;

void main()
{
clrscr();
printf("Memory size of struct : %d bytes",sizeof(t1));
printf("\nMemory size of union : %d bytes",sizeof(t2));
getch();
}

```

Output

- 1) Memory size of struct : 7 bytes
Memory size of union : 4 bytes

STRUCT	UNION
1.A group of data items that belongs to different data types	1.It is also same as struct, but the only major difference is memory allocation.
2.Allocates memory of all declared data items in it	2.Allocates memory of biggest data items in it
3.We can access all data items at a time	3.We can access only one data items at a time

4.Each and every data items has its own storage space	4.All the items shared common storage space.
---	--

Program : union.c

```
#include<stdio.h>
#include<conio.h>
union emp
{
    int eno;
    char ename[20];
    float sal;
};
void main()
{
    union emp e;
    printf("Enter empno : ");
    scanf("%d",&e.eno);
    printf("Emp number : %d",e.eno);
    printf("\n\nEnter empname : ");
    fflush(stdin);
    gets(e.ename);
    printf("Emp name : %s",e.ename);
    printf("\n\nEnter empsal : ");
    scanf("%f",&e.sal);
    printf("Salary : %.2f",e.sal);
}
```

FILES

C language permits the usage of limited input and output functions to read and write data. These functions are used for only smaller volumes of data and it becomes difficult to handle longer data volumes. Also the entire data is lost when the program over.

To overcome these difficulties a flexible method was developed by employing the concept of files to store, to read and write data and to return them even when the program over.

Definition :

A file is one, which enable the user to store, to read and write a group of related data.

(or)

A file is a collection of records .

C supports a number of functions to have the ability to perform the basic file operations, which include :

- naming a file
- opening a file
- reading data from file
- writing data to file
- closing a file

Types of files :

Files are two types, Namely

1. Text files (or) Sequential files
2. Binary files

Text files :

Used for reading and writing data in the form of characters. The memory size of each and every character is 1 byte.

Binary files :

Used for reading and writing data in the form of data blocks. The memory size of each and every data block depends on its data type.

FILE(keyword):

It is a pre-defined structure and is used to declare a file pointer. Using this pointer, we can perform all file operations.

Declaration of FILE pointer :

Syntax :

FILE *identifier.

Eg:

FILE *fp;

File handling functions :

fopen : It opens a file .

Syntax :

```
FILE * fopen(const char *filename , const char *mode);
```

filename : file that the function opens

mode string :

There are various modes and are described in the following table.

String	Description
r	Open for reading only.
w	Create for writing. If a file by that name already exists, it will be overwritten
a	Append : open for writing at end of file, or create for writing if the file does not exist.

r+	Open an existing file for update (reading and writing)
w+	Create a new file for update (writing and reading). If a file by that name already exists, it will be overwritten.
a+	Open for append : open for update (writing and reading). at the end of the file, or create if the file does not exist.

- 1 To specify that a given file is being opened or created in text mode, append "t" to the string (rt, w+t, etc.).
- 2 To specify binary mode, append "b" to the string (wb, a+b, etc.).

fclose :

It closes a file .

Syntax :

```
int    fclose(FILE *fp);
```

fcloseall :

It closes all open files.

Syntax :

```
int fcloseall();
```

Program : file1.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    clrscr();
    fp=fopen("a.txt","w");
    if(fp==NULL)
        printf("Unable to open a file");
    else
        printf("File is opened");
    getch();
}
```

If we specify the mode “a”, we will get the same output (i.e. File is opened), if we specify the mode “r”, we will get the output as Unable to open a file.

Textfile operations**getc :**

It is a macro that gets one character from a file.

Syntax :

```
int getc(FILE *stream);
```

putc : It is a macro that outputs a character to a file .

Syntax :

```
int putc( int c,FILE *stream);
```

fgetc : It is a function that gets one character from a file .

Syntax :

```
int fgetc(FILE *stream);
```

fputc : It is a function that outputs a character to a file.

Syntax :

```
int fputc(int c,FILE *stream);
```

EOF :

It is a constant indicating that end of file has been reached on a File. To get this character from keyboard press Ctrl+Z

Program : file2.c

To create a text file and to store data into that file through keyboard

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    FILE *fp;
    clrscr();
    fp=fopen("a.txt","wt");
    printf("Enter data (Ctrl+Z to stop) : ");
    ch=getchar();
    while(ch!=EOF)
    {
        putc(ch,fp);
        ch=getchar();
    }
    fclose(fp);
    printf("Data stored successfully");
    getch();
}
```

Program : file3.c

To open a file and to read data from the file and display

on the monitor

```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    char ch;
    clrscr();
    fp=fopen("a.txt","rt");
    printf("Reading data from the file\n\n");
    ch=getc(fp);
    while(ch!=EOF)
    {
        printf("%c",ch);
        ch=getc(fp);
    }
    fclose(fp);
    getch();
}
```

Program : file4.c

To enter a file and display its contents on the monitor

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

```
FILE *fp;
char ch,st[20];
clrscr();
printf("Enter File name : ");
gets(st);
fp=fopen(st,"rt");
if(fp==NULL)
{
    printf("\nFile not found");
    getch();
    exit(0);
}
printf("\n\nReading data from the file\n\n");
ch=getc(fp);
while(ch!=EOF)
{
    printf("%c",ch);
    ch=getc(fp);
    delay(10);
}
fclose(fp);
getch();
}
```

Command line arguments :

These are the parameters supplied to a program, when

the program is invoked. This parameter may represent a file name that the program should process. Command line arguments are typed by the user. The first argument is always the file name.

We know that, every C program should have one main function and it can take arguments like other functions. If we want to work with command line arguments, the main function can take 2 arguments called argc and argv and the information contained in the command line is processed onto the program through these command line arguments.

The variable argc is an argument counter that counts no. of arguments on the command line. The argument argv is an argument vector that represents an array of character pointers that points to the command line arguments. The size of this array is equal to the value of argc.

Program : cla1.c

```
#include<stdio.h>
#include<conio.h>
void main(int argc,char *argv[])
```

```

{
    int i;
    clrscr();
    printf("Number of argument : %d",argc);
    for(i=0;i<argc;i++)
    {
        printf("\nArgument[%d] : %s",i,argv[i]);
    }
    getch();
}

```

Program : cla2.c

```

#include<stdio.h>
#include<conio.h>
void main(int argc,char *argv[])
{
    FILE *fp;
    char ch;
    clrscr();
    if(argc!=2)
    {
        printf("Invalid arguments");
        exit(0);
    }
    fp=fopen(argv[1],"wt");

```

```
ch=getchar();
while(ch!=EOF)
{
    putc(ch,fp);
    ch=getchar();
}
fclose(fp);
printf("File created");
getch();
}
ch=getc(fp);
while(ch!=EOF)
{
    printf("%c",ch);
    ch=getc(fp);
}
fclose(fp);
getch();
}
```