

WHAT ARE PROGRAMMING LANGUAGES?

Those languages which are used in the computer programming are called programming languages. Since the inception of computer, three type of computer programming languages have been evolved which are:

1. Machine language or low level language
2. Assembly level language
3. High-level languages

Machine language:

The computer language which is written in binary format (i.e. containing only 0's and 1's) as the set of instructions to the computer is called machine language. It is the only one language; a computer is able to understand directly without any modification. The earliest written programs were in machine language. The main advantages of machine language were that they were directly understood by the computer and they could be processed very fast and the results were given very quickly. However it suffered from the following disadvantages:

1. It was represented in detailed binary codes consisting of groups of 1's and 0's as such the programs were very lengthy.
2. All data and instruction had to be manually coded into machine language and later all results obtained in the machine language had to be decoded into human understandable form.

3. The binary codes of certain functions and special characters were difficult to learn.
4. The coding and decoding process was time consuming as each and every character was unique combination of 1's and 0's.
5. The lengthy programs and difficulty faced in learning and remembering the codes resulted in errors.

Thus it is seen that writing machine language programs is very bulky, time consuming process liable to errors and the need to improve this system led to the development of Assembly Language.

Assembly language:

Assembly language is a computer programming language which combines the restricted use of machine language with mnemonics or human memory aids. These aids are abbreviated forms of certain functions like *ADD* for addition, *SUB* for subtraction, *MULT* for multiplication and *HLT* for halt (exit) the program. This makes the programmer easy to remember the code of the functions. As mentioned above computer can only understand machine language, assembly language can not be understood by computer therefore it should be translated first into the machine language. Assembly language was machining depended language i.e. the code for one computer may not applicable for another computer system. Only the assembly language was not sufficient for the human beings,

therefore they developed high level language which is more user friendly programming language.

High level language:

The programming language which can be easily understood by human is called high level language. Various high level languages have been created to design coding schemes which would be least troublesome to learn and write. These languages transcribe the programs as statements, using a very limited vocabulary from English. There are various high level languages being used for specific applications like scientific, business, and mathematics. These high level languages have the following **Advantages over low level languages:**

1. The use of 'English' with proper syntax made it easy for the users to write programs quickly and efficiently. It means high level languages are user friendly.
2. They are all much faster to work with and are versatile and provide better documentation.
3. Uniformly maintained in the codes used, resulted in its applicability on any computer.
4. The programs written in high level language can be easily changed.

As in the case of assembly language; which is not directly understood by the computer, this high level language also can not be

understood by the computer. It also requires translator. Some of the popular and widely used high levels programming languages are:

FORTRAN (Formula translation, which is the first high level language), *ALGOL* (algorithm), *COBOL*, *Basic*, *Pascal*, *C*, *C++*, *Java*, *Visual Basic* etc.

Language translator:

We know that except machine language, computer can not understand other type of languages therefore other languages should be translated into the machine language. Those computer software or programs which convert the program written in one language into another language are called language translators. There are three types of language translators which are as follows:

1. Assembler
2. Compiler
3. Interpreter

Assembler:

Since a computer can not understand the assembly language, to understand the program it should be translated into machine language. A program which converts the program written in assembly language into the machine language is called Assembler. Only after the conversion of the program into the machine language,

computer can understand the program and required output can be obtained.

Compiler:

The source program written in any high level language should be translated into the machine language. The software which translates the source program written in high level language into the machine language is called Compiler. Different types of high level languages have different compiler. A compiler can translate a source program written in only one high level language. This means that a FORTRAN compiler can not used to translate a source program written in C language.

Interpreter:

An interpreter is a different type of translator. Both assembler and compiler translate the whole programs into the machine language but interpreter does not convert. An interpreter is a program which resides in a computer's memory, interprets and executes the source program. The interpreter program is stored on external memory like floppy disk or magnetic tapes and it is brought into the RAM, whenever it is required. Whereas the compiler translates the complete program into object code at one time, the interpreter evaluates and translates program statements as the program is executed, one instruction at a time.

What are the main differences between Compiler and Interpreter?

<u>Compiler</u>	<u>Interpreter</u>
<ol style="list-style-type: none">1. A compiler translates an entire program written in high level language into an object program in machine language at one time.2. The Compiler may or may not reside in the memory along with user's source program3. Compilers execute the program at a fast speed.4. After compiling if the programmer wishes to change the program, add or correct the errors, the entire process must be repeated, which becomes quite time consuming and bulky.	<ol style="list-style-type: none">1. The interpreter takes one source program instruction at a time, translates it into object code and executes it and then takes the next instruction, translate it and so on.2. The Interpreter must reside in the memory along with the user's source program.3. Interpreters have slow speed.4. With an Interpreter, changes and additions can be made interactively. Interpreters are easy to use, totally interactive and as such are widely used.

Basic introduction on computer software and its types:

Software is a complete set of instructions or collection of programs, that enables the computer to reach the solution to any problem. The software actually brings the hard ware of a computer system into operation. Software also includes any printed matter or reference manual which may be needed by the computer operator to use the computer appropriately. Computer software can be classified into three different categories which are:

- a) system software
- b) application software
- c) utility software

System software:

System software includes all the programs, languages and documentations, generally supplied by the manufacturer with the computer. System software is an indispensable part of a computer system. These programs help the programmer or user to communicate with the computer and to write his own programs. The system software is included to make the use of the computer more

effective, faster, and efficient. System software is the operating system of the computer. Without this computer is just like a bare machine. *UNIX, Window '98, Windows xp* etc are the examples of system software.

Application software:

These are the programs written by the user for performing certain specific function. That software which is applicable for our daily life is called application software. Examples of the application software are *Microsoft word, Microsoft Excel, Microsoft power point, database program, tally, Page maker* etc.

Utility software:

Utility software is also the programs normally developed by the hardware manufacturer and supplied with the system, but can also be developed by the user, depending upon his application area and can be stored in the primary memory. Thus it can be recalled by the application software when ever required. Utility programs are the set of commonly used programs for certain tasks. Some examples of utility programs are *text editor* (which is used for the generation of corrected and organized text), *debugging aids* (which is used to correct the logical mistakes) and *input-output routines*.

Introduction to C programming language

C seems a strange name for a programming language but this strange sounding language is one of the most popular computer language today because it is structured, high level, machine independent language. It allows software developers to develop programs without worrying about the hardware platforms where they will be implemented.

The root of all modern language is ALGOL, introduced in the early 1960s. ALGOL gave the concept of structured programming to the computer science community. In 1967, Martin Richards developed a language called BCPL (Basic Computer Programming Language) primarily for writing system software. In 1970, Ken Thompson created language using many features of BCPL and called simply B.

C was evolved from ALGOL, BCPL and B by Dennis Ritchie (father of the C language) at Bell Laborites in 1972. C uses many concept of this programming language and added the concept of data types and other powerful features. Since it was developed along the UNIX operating system, it is strongly associated with UNIX.

Reasons behind the popularity of the C-programming:

Due to the following characteristics of C programming language, we need this programming language:

1. Flexibility nature:

C program is a flexible language because it includes only the words of the English language which can be easily understood by the users. The meaning of words (identifier in C) of this language has the same meaning as our daily life. Due to this we can understand the meaning of the every instruction (statement) given to the computer.

2. Portable language/ Machine independent language:

Many programs written in one language run in one computer but may not run in other computer. But the program written in C language can run in any computer with little or no modifications on the program. This nature is called portability. Languages which depend upon the machine or computer system are called machine dependent language. Since C is machine independent language hence C program can be written in any computer system.

3. Easy to learn and understand:

Since C uses all English words, therefore it is easy to learn and understand the c program.

4. Modular/ Structured Programming language:

In C programming a complex or long program can be divided into many sub-programs which are called module or functions. Each

module or function does the coherent work of the main program. Due to this the understanding of the program will be very easy. Also the program will very efficient. This is the most important feature of the C language.

5. Procedure oriented programming language:

This programming language only emphasize on the logic of the program used for solving the problem rather than the object. This means it only gives rise in the procedure of the programming.

What is compiling and compiler?

Since computer can understand only binary language which includes only 0 and 1(i.e. either true or false). And the program that we write in the computer is not in the form of binary format. Therefore to make the program understandable by the computer, it should be first converted into machine language (binary language). The language translator which translates or converts the program written in high level language i.e. language that can understand by the human into the machine language is called compiler. Compiler is also a program or software.

The process of converting or translating the program code written in high level language into the machine level language is called compiling. The program code is also called source code and the machine code called as object code.

The compilation process can be easily understood by the following figure:

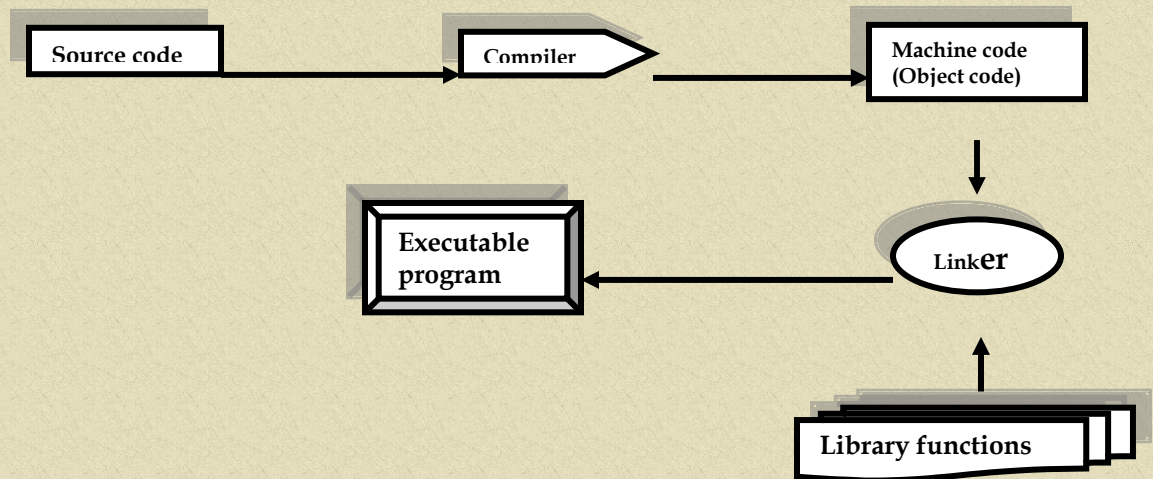


Fig.: compilation process

Source code:

Program written in a particular programming language in its original form is called source code (program). The word source differentiates code from various other forms that it can have (for example, object code and executable code). To execute the program, however, the programmer must translate it into machine language. The compiler translates the source code into a form called object code. Source code is the only format that is readable by humans. When we purchase program, we usually receive them in their machine language format. This means that we can execute them directly, but cannot read or modify them.

Object code:

Object code is the code produced by a compiler. Object code is the same as or similar to a computer's machine language. The final step

in producing an executable program is to transform the object code into machine language, if it is not already in this form. A program called linker does this job.

Basic structure of the C program

Documentation section
Linker section
Macro definition section/ definition section
Global variable declaration section
Global function declaration section
main() function <pre>{ declaration part; execution part; }</pre>
Function definition section

In this we study the sequence of the instructions to be written in any C program. For more clearly, we study about which section is to come first and which is supposed to be the last. In general all C

programs have seven sections as shown in the table above in the sequence which are described in short.

1. Documentation section:

In this section we write the question or the problem i.e. topic. This is not executable statement on the c program. This line remains in the program as comment line where we can write comment about the program.

2. Linker section:

This line starts with **#include** which is called directive. It is not the part of the actual program. It is used as a command to the compiler to direct the translation of the program. The directives included in the program is called **header file** (where the meaning of library functions are written). For example:

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

3. Definition section:

In this line we define the macro (constant or some short expression). For example:

```
#define PI 3.1416
#define sum(a+b)
```

4. Global variable declaration:

The variable which can use any function of the program is called global variable. If a variable is to be declared globally then we should declare them just below the definition section or just above the user defined function declaration.

5. Global function declaration:

Those functions or modules which can be called by any function of the program are called global function. Most of the books combine these two sections: global variable declaration and global function declaration as a single section because both section are declaration sections.

6. main() Function:

This is the entry point of any program i.e. the execution of any program starts from this point. Main () function is also a user defined function but main () function and other user defined functions are difference from each other. The difference between main () function and other function will study in later in the chapter FUNCTIONS.

In consists of two parts:

a) declaration part

b) Execution part.

The part of the main function where we declare the variable and function is called declaration part. The variable declared inside the main function is called local variable which can used only by main function. The declaration of any variable or function should be done before use them and above the execution part. The section in which

we write executable statement or instruction to the compiler is called execution part. Actual body of the program is this execution part.

7. Function definition section:

Once a function is declared and is to be used then the work of that function should be written. This is called function definition. The detail study in this topic, we will study in chapter FUNCTIONS.

Programming tools:

Before writing a program there are certain tools which makes the programming easy, these tool are called programming tools. We can write the program with out using the any programming tools but it is better to use them first and then write the programming code. Commonly used programming tools are:

1. algorithm
2. flowchart
3. pseudo code
4. decision tree

Here we concerned with only upper two tools i.e. algorithm and flow chart.

1. Algorithm

The set of the instructions of the program which are written in step wise step manner in general language is called algorithm. Algorithm is nothing but the steps to be done in writing the program code. It is written in general language in step wise step manner. Before writing any program, algorithm of the program should be known. With out proper algorithm we can not write any program code. Algorithm should be in common and simple language. While writing the algorithm we can't use any programming code. Program can be written without any application of programming tool but good program is only possible with the application of it.

Example: **to find out the greater between two numbers algorithm (steps of programming) will be:**

1. Start the program.
2. Read the two numbers.
3. Compare two numbers.
4. If first number is greater than the second then print first number is greatest otherwise print second number is greatest.
5. Stop the program.

Similarly algorithm for preparing the tea will be:

1. Lit the fire.
2. Put the tea-pot on the fire.
3. Boil the milk.

4. Pour the water on the pot.
5. Pour tea, sugar, and other thing.
6. Boil the mixture.
7. Stop the fire.
8. Now your tea is ready.

In similar way we can write the algorithm of the different programs.

2. Flowchart:

The graphical or pictorial representation of the sequence of the data or program i.e. algorithm is called flowchart. Therefore, flow chart is another important tool of programming system. Flow chart easily represents the flow of data and set of instructions of the program. By looking the flow chart of the program even non programmer (normal person) can understand the program i.e. objective of the program.

Features of the flow chart:

1. A flow chart is a pictorial representation of the flow of the operations. Hence any error in the logic can be easily detected.
2. Flow chart is very helpful in case of modification on program in future.
3. While writing big software, different programs are involved to draw different flow chart. After the completion of the flow








charts, programmers sit together and visualize the overall program.

4. Once the flow chart is completed, it acts as a guideline to write a program.

Disadvantages of the flow chart:

1. Drawing the flow chart is time consuming.
2. Flow charts are difficult to amend. Redrawing of flow chart is required.
3. When there are complex branches and loops, flow charts become more complicated.

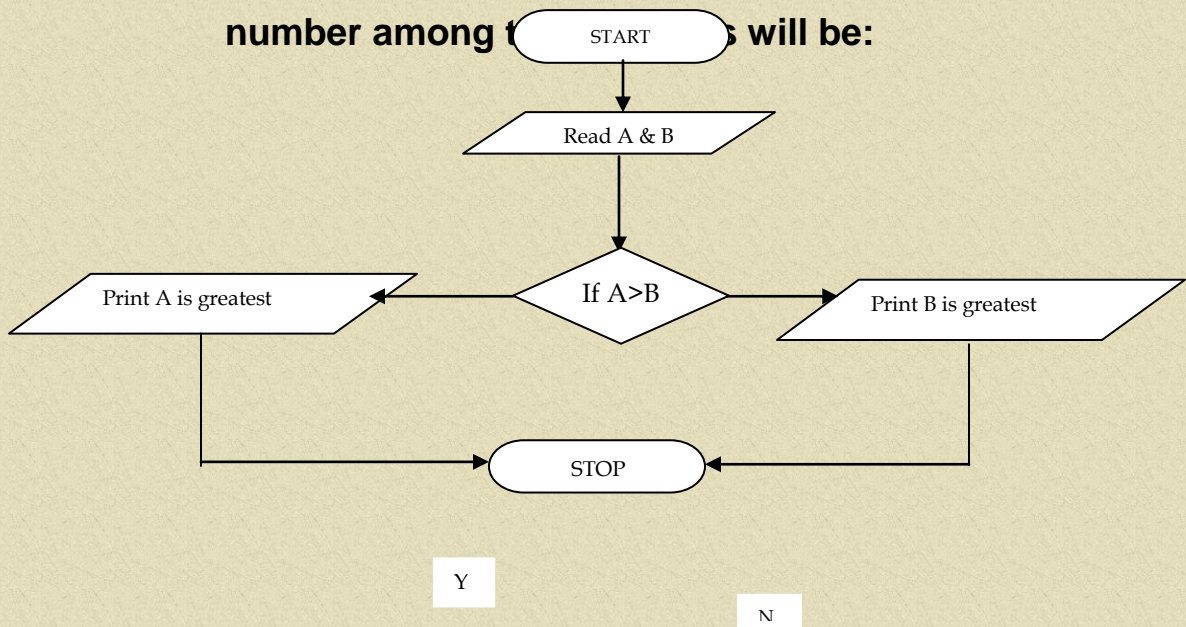
There are some symbols used in flow charts which are given below:

Symbols	Purpose
	Start or stop
	Output /display
	Input/ read the data/output
	Process
	Decision making
	External file/ function
	Flow of the program



Connector

For example the flow chart for finding the greatest number among two numbers will be:



3. Pseudo Code:

It is one of the important and famous tools among all the programming analysis tools. Pseudo means “false” or “imitation and code means “instructions to the computer”. Therefore, it is useful to analysis whether instructions are systematically given to the computer or not. In conclusion, it is imitation of actual computer

instructions, it is written in different general language like English, and French etc. in modern programming algorithm also acts as pseudo code.

Fundamental of C programming:

Header file:

Sentence that begins with a hash sign (#) are directives for the preprocessors. These directives are called header file. They are not executable part of the program but they should be included while writing any program because they contain the definition or meaning of the library functions, key words, and pre-defined constants. The header file can be included by following syntax:

`#include<header file name>` OR

`#include"header file name"`

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

`#include"string.h"`

The other header files which we will require for this level are listed below:

- 1) `stdio.h`
- 2) `conio.h`
- 3) `string.h`

- 4) math.h
- 5) stdlib.h
- 6) ctype.h
- 7) time.h
- 8) alloc.h etc.

Statements:

The sentence which is written in the program to do one work separately is called statement. All the statements in C should be terminated by semicolon (;) which is called terminator. For example: $a = b + c$; is a statement.

Braces { }:

The opening and closing braces are used in the beginning and end of the compound statement respectively. Every functions or the compound statement should contain this braces with appropriate form.

Compound statements:

The statements written inside the braces which act as a single statement are called compound statement. Generally it consists of more than two statements. For example:

```
{  
a= b+c;  
d= a+b;
```

} this over all statement which is enclosed by braces is compound statement.

Input/output put functions:

Those functions which are used to send the data from the user or from the file are called input functions and the functions used to display the result or output are called out put functions. Most common output function is printf () function which is used to display the text in standard output device i.e. monitor. Similarly the most common input function is scanf () function which is used to read the data from the standard input device i.e. keyboard.

Library functions:

Those functions whose definitions or job are pre-written in the library of the C program are called library functions. printf (), scanf (), clrscr (), getch() etc. all are example of the library functions.

Comments:

The lines written within a program to make the program user friendly are called comments. Comments are non-executable lines in the program. It helps to understand the logic of the program to solve the problem. We can add the comment in the program in two ways:

a) Single line comment:

If we want to add only one line/single line comment in the program then we can use double division sign for commenting. For example:

```
// I am going to learn C programming.
```

b) Multiple-line comment:

If we want to add the comment more than more than one line then use of single line comment is not efficient. In this case we can use this:

```
/* .....comments..... */
```

Variable:

It is the name given to the memory location to hold some value. It is an entity. During the execution of the program value of the variable may changes. For example:

```
int total;
```

char name; etc. here total and name are two variables.

Constants:

A value that doesn't change during the execution of the program is called constant. Constant is sometimes referred to as literal. Whatever the content is there in the variable which is fixed is called constant. It cannot be changed during the execution of the program. Types of the constant can be understood by the following chart:

An integer constant like 1234 is an int. A long constant is written with a terminal l (ell) or L, as in 123456789L; an integer constant too big to fit into an int will also be taken as a long.

Unsigned constants are written with a terminal u or U, and the suffix ul or UL indicates unsigned long.

Floating-point constants contain a decimal point (123.4) or an exponent (1e-2) or both; their type is double, unless suffixed. The suffixes f or F indicates a float constant; l or L indicates a long double.

A character constant is an integer, written as one character within *single quotes*, such as 'x'. The value of a character constant is the numeric value of the character in the machine's character set. For example, in the ASCII character set the character constant '0' has the value 48, which is unrelated to the numeric value 0. If we write '0' instead of a numeric value like 48 that depends on the character set, the program is independent of the particular value and easier to read. Character constants participate in numeric operations just as any other integers, although they are most often used in comparisons with other characters.

The complete set of escape sequences is

\a	alert (bell) character	\\	Backslash
\b	Backspace	\?	question mark
\f	form feed	\'	single quote
\n	new line	\"	Double quote
\r	carriage return	\ooo	Octal number
\t	horizontal tab	\xhh	hexadecimal number
\v	vertical tab		

The character constant '\0' represents the character with value zero, the null character. '\0' is often written instead of 0 to emphasize the character nature of some expression, but the numeric value is just 0. A *constant expression* is an expression that involves only constants. Such expressions may be evaluated at during compilation rather than run-time, and accordingly may be used in any place that a constant can occur, as in

```
#define MAXLINE 1000  
char line[MAXLINE+1];
```

or

```
#define LEAP 1 /* in leap years */  
int days[31+28+LEAP+31+30+31+30+31+31+30+31+30+31];
```

A *string constant*, or *string literal*, is a sequence of zero or more characters surrounded by *double quotes*, as in

```
"it is a string"
```

or

```
"" /* the empty string */
```

The quotes are not part of the string, but serve only to delimit it. The same escape sequences used in character constants apply in strings; \" represents the double-quote character. String constants can be concatenated at compile time:

```
"hello, " "world"
```

is equivalent to

```
"hello, world"
```

This is useful for splitting up long strings across several source lines.

Technically, a string constant is an array of characters. The internal representation of a string has a null character '\0' at the end, so the physical storage required is one more than the number of characters written between the quotes. This representation means that there is no limit to how long a string can be, but programs must scan a string completely to determine its length. The standard library function `strlen(s)` where `s` is the string and this library function is included into the header file `string.h` returns the length of its character string argument `s` which is an integer, excluding the terminal '\0'. For example: `Length=strlen(s);`

Be careful to distinguish between a character constant and a string that contains a single character: 'x' is not the same as "x". The former is an integer, used to produce the numeric value of the letter x in the machine's character set. The latter is an array of characters that contains one character (the letter x) and a '\0'.

Identifiers:

Identifiers can be defined as the name of the variable, functions, arrays, structures etc created by the programmer. They are the fundamental requirement of any programming language. It consists of letters (a-z, A-Z) and digit (0-9) in any order but the first character must be a alphabet or underscore (`_`) and should contain at least one letter. Both the lower case and upper case letters are permitted. The following are the valid identifiers:

A, abc, ab12, `_name`, `roll_no`, SCHOOL, etc.

The following names are not valid identifiers for the reason stated:

1st the first character must be letter.

“name” illegal character (“)

roll-no illegal character (-)

roll no illegal character (blank space)

Keywords:

Keywords are the reversed identifiers and they can not used as name for the program variable or other user defined program element. The meanings of the key words have already been given to the compiler. The key words are also called reserved words. There are 32 key words available in C which is listed below:

auto	const	double	float	Int	short	struct	unsigned
break	continue	else	for	Long	signed	switch	void
case	default	enum	goto	Register	sizeof	typedef	while
char	do	extern	if	Return	static	union	far

Note that keywords all are should in lower case. Since upper case is not equivalent to lower case therefore upper case keywords can be used as other identifiers. Some compiler contains more than 32 keywords also.

Data types:

A data type is defined as the set of possible values that a variable can hold on it. This means data type identifies the type of the data i.e. whether the variable is character type or integer type or whether it contains decimal values or not etc. There are mainly three data types which are:

- a) Built in data type
- b) User defined data type
- c) derived data type

We study the user defined data type and derived data type later in the chapters' structures and arrays respectively. In this part we will study only built in data type.

It is also called **basic** or **standard** or **fundamental data type**. There are mainly five built-in data types.

1) Integer (int) data type:

This type of data can store only whole numbers. The key word int is used for integer data type. The valid integers are **446, 421, -123, 2, 3333** etc. The size of integer type data is **2** byte i.e. **16** bits.

2) Character (char) data type:

The data types which are used to store data of character are called character data type. The key word char is used to represent the character data type. The size of character data type is **1** byte i.e. **8** bits.

3) Floating point (float) data type:

If a number contains the fractional part then it is called floating point number. This type of data is usually stored on float where float is the key word for the floating point number. Size of this type of data is **4** bytes. For example **12.344** are floating point number. This type data can store the six digits after decimal.

4) Double (double) data type:

Double data type is used to represent double precision floating point number. It is called double data type because it occupies twice the memory than that of the floating point i.e. **8** bytes and it can stores **12** digits after decimal.

5) empty (void) data type:

The void data type specifies as empty set of values. It is used as the return type for function that does not return any value. Generally we don't declare the variable of type void. We study in more detail about this in **FUNCTIONS**.

Rules for declaring the variables:

1. First character of the variable should be letter or underscore. Other extra character is not allowed at the beginning. Then other type of character can be included.

For example:

int roll; this is valid.

float _hello; this is valid.

char 12name; this is not valid because beginning character must be letter.

2. The name of the variable should not match with the name of the functions or the key words. For example main, int, char, static etc. are invalid variable names because they are key words.
3. While writing the name of the variable name both upper case and lower case are permitted.
4. Upper case and lower case are not interchangeable.
5. Space is not allowed in between the name of the variable.

The printf() function:

This is one of the most common output functions used in C programming. Its main usage is to display the text, or value or both on the standard output device i.e. on the monitor.

Depending upon what is to be displayed, the general syntax of the printf () are as follows:

1. for displaying text only:

```
printf("your text");
```

2. for displaying value only:

```
printf("formal string/ type specifier",variablename);
```

3. for displaying both text and value:

```
printf("text type specifier ",variablename);
```

Examples of each type of syntax are given below:

MY FIRST PROGRAM IN C:

1. Program for first type syntax:

```
/*write a program to display the text "This is  
my first program in C." */
```

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

```
main()
```

Documenting section

Linker section

Main function

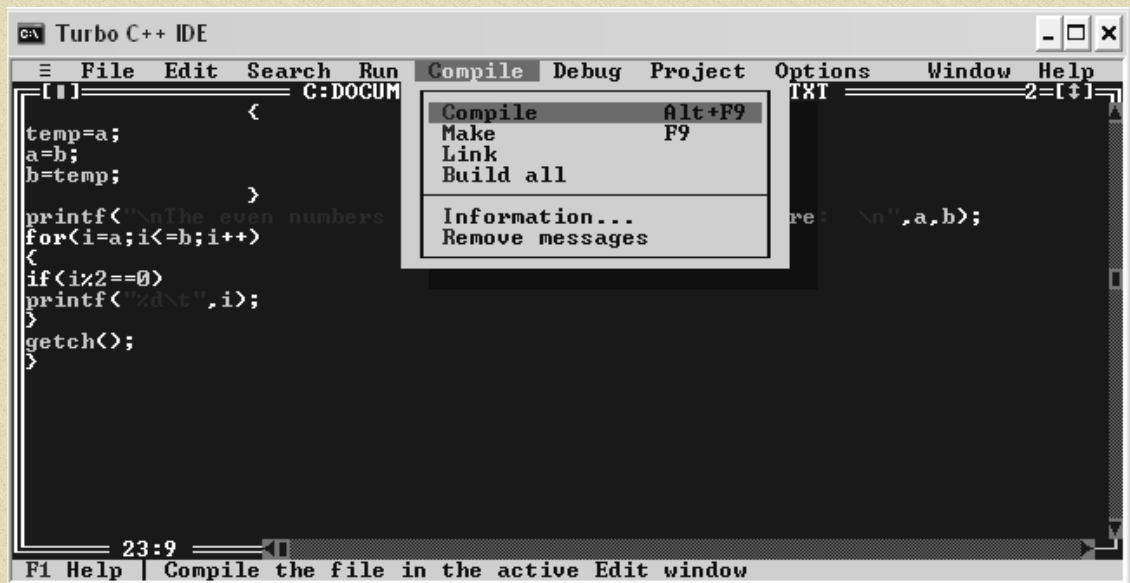
```
{  
    printf("This is my first program in C.");  
}
```

Executable statement

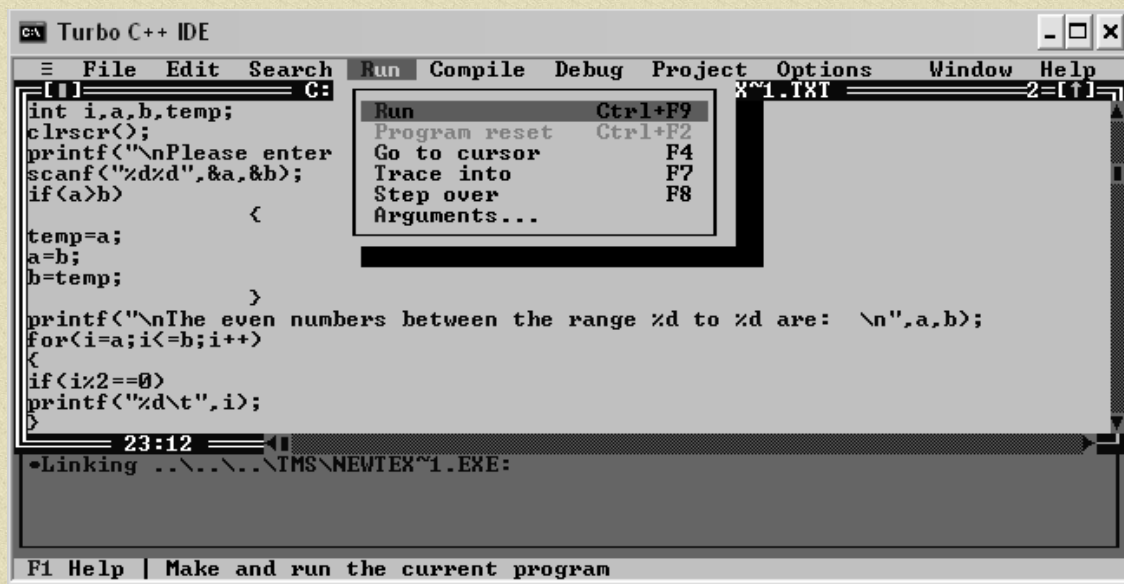
Out put of the program:

This is my first program in C.

To run the program please press **Ctrl+F9** and after this to see the output of the program press **Alt+F5**. Before run the program check the compile the program whether the program contains error or not by pressing **Alt+F9**.



To compile the program



To run the program

1. Program for second type of syntax:

Before writing the program for this syntax, again look at the syntax once again. There is the term type specifier / formal string. What this means? Let's know this term. The symbol which specifies the type of the data is called type specifier and this is also called as the formal string. Various type specifier for different data types are shown in the table:

Data type	Formal string
Integer	%d

Floating point	%f
Character	%c
Double	%lf

/*write a program to assign a value on A and print the value of A*/

```
#include<stdio.h>
void main()
{
    int A;
    A=10;           //assigning the value 10
for A
    printf("%d",A);
}
```

The output of the above program is just only 10. This syntax is not user friendly, it means only seeing the out put of the program 10 we can not say what 10 is. Therefore for the more user-friendly the third syntax is used.

2. Example of the third type of the syntax:

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

```
void main()
{
    int a,b,c;
    a=5;
    b=10;
    c=12;
    printf("the value of the a is %d", a);
    printf("the value of b is %d", b);
    printf("the value of c is %d",c);
}
```

Sample output:

the value of the a is 5the value of b is 10the
value of c is 12

We can write the above program as follows:

```
#include<stdio.h>
void main()
{
    int a=5,b=10,c=12;
    printf("the value of a is %dthe value of b is  
%dthe value of c is %d",a,b,c);
}
```

Output of this program is same as the above. In these two programs all output will be displayed as the line. If we want to escape the line then the printf(); statement can be written as:

```
printf("the value of a is %d\nthe value of b is  
%d\n the value of c is %d",a,b,c);
```

The output of the program is:

```
the value of a is 5  
the value of b is 10  
the value of c is 12
```

Qualifier:

Those words which are used to increase the quality of the data is called qualifier. These words are preserved words i.e. they are key words.

They are the words preceding the data type. Syntax:

qualifier data_type variable;

Example:

long int sum;

The function scanf ():

This is the most common data input function used in C. If we want to send the data to the computer then we use this function. It reads the data from the standard input device i.e. keyboard. General syntax of this function is:

```
scanf("type specifier",& variable_name);
```

Here the symbol & is called address operator. The function of this symbol can be understood by this example. If you have a bag having many pockets and you have to put a book in the book then where you would put this book? Naturally where the enough space is in the bag for the book, there you will be put this book. For this you have to search for the space. In the same way to

hold a value on the variable, the symbol '&' searches the space /memory location in the computer memory. For simplicity we can say that '&' represents the memory location of the variable and therefore called as address operator.

Scanf ()/printf () format codes:

Code	use for
%d	for integer number
%c	for single character
%f	for floating point number
%h	for short integer
%s	for string
%l	for long
%ld	for long integer

Write a program which takes a numeric data from user and displays the data on the monitor.

```
#include<stdio.h>
void main( )
{
int a;
printf("enter the value of a:  ");
scanf("%d",&a);          //reading the data from the
keyboard.
printf("\n\nValue of a is %d.",a);
}
```

Sample out put of the program:

enter the value of a: 12

Value of a is 12._

Other useful library functions:

<u>Library function</u>	<u>purpose</u>
-------------------------	----------------

- | | |
|-----------------|---|
| 1. clrscr(); | to clear the screen |
| 2. getch(); | to hold the output of the program of the program unless one key is pressed. |
| 3. getche(); | to read a character from the standard input device |
| 4. strcpy(); | to copy the string and assign |
| 5. sqrt (); | to find out the square root |
| 6. abs(); | to find the absolute value |
| 7. strlen (); | to find out the no. of characters in the string |
| 8. strcmp(); | to compare two strings |
| 9. toupper(); | to change a character in upper case |
| 10. tolower(); | to change the character into lower case |

Operators and types:

Operators are nothing but are the symbols which are used to derive certain operations. An operator is a symbol which tells the computer to do certain mathematical or logical task. Operators are used in programs to manipulate and compute the data and variables. Therefore, operators are sometimes called as the driver of the operation. C contains large number of the operators.

Depending upon the type of operator, operators are classified into numbers of categories among them some important categories are as follows:

A) Binary Operator:

1. Arithmetic operators
2. Relational operators
3. Logical Operators
4. Assignment operators

B) Unary operators:

1. Increment and decrement operators
2. Unary minus operator

C) Conditional operator

Binary operators:

The word binary represents the meaning of this type of operator. Binary means “two” therefore that type of operator which takes two operands to compute the operation is called binary

operators. There are many types of the binary operators among them some are explained below:

1. Arithmetic Operators:

C includes all the fundamental arithmetic operators. The symbol which is used to manipulate the various type of the arithmetic operation is called arithmetic operator. The arithmetic which are available in C are listed below:

Operators	Meaning	Example
+	Addition	$a + b$
-	Subtraction	$a - b$
*	Multiplication	$a * b$
/	Division	a / b
%	Modulo division/ remainder operator	$a \% b$

It should be noted that, modulo division (%) is only possible between two integer but all other arithmetic operators can operate with float and double also.

Example: use of the arithmetic operators

```
/*write a program which uses all the arithmetic  
operators.*/  
  
#include<stdio.h>  
#include<conio.h>  
  
void main()  
{
```



```
int a,b,sum,sub,mul,mod;
float div;
clrscr();
printf("Pliz enter the value of a:\t");
scanf("%d",&a);
printf("\nPliz enter the value of b:\t");
scanf("%d",&b);
sum=a+b;
sub=a-b;
mul=a*b;
mod=a%b;
div=(float)a/b;
printf("\n sum of the given numbers %d and %d is
%d", a,b,sum);
printf("\n difference between given nnumbers is
%d",sub);
printf("\n multiplication of given numbers is
%d",mul);
printf("\n remainder after dividing the %d by %d
is %d",a,b,mod);
printf("\n division of given numbers is %f", div);
getch();
}
```

Out put of the program:

```
Pliz enter the value of a:    12
Pliz enter the value of b:    5
```

Sum of the given numbers 12 and 5 is 17

Difference between given numbers is 7

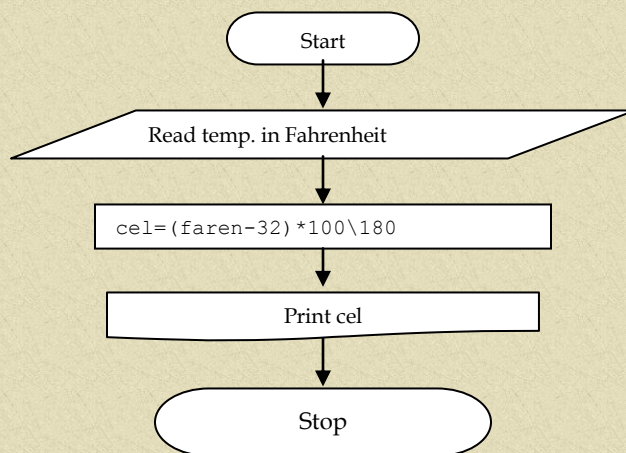
Multiplication of the given numbers is 60

Remainder after dividing the 12 by 5 is 2

Division of given numbers is 2.400000

Here in the program we are written the statement *div= (float) a/b;* because here div is a variable of float type but a and b are variables of type int. While dividing int by int it also produces the result in int i.e. it neglects the fractional part of the result and prints only 2.000000 which is wrong result. For this we have to tell the computer that the result is float, therefore we are writing float in front of the expression a/b. This process is called type caste because here we change the caste of the result of the program first the result was int further we change it into float.

*/*Write flow chart and program code which converts the Fahrenheit temperature into the Celsius degree.*/*



Program code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    float faren,cel;
    printf("\nplease enter the temperature in
    Fahrenheit degree:\t");
    scanf("%f",&faren);
    cel=(faren-32)*100\180;
    printf("\n\nthe given temperature in Celsius
    degree is %f.",cel);
    getch();
}
```

2. Relational operators:

The symbols which are used to compare two expressions or values or data are called relational operators. The result of the comparison is either true or false which is used to make a certain decision. If the relational expression is true then it returns *one*

otherwise if the expression is false then it returns zero. C supports six relational operators which are as follows:

<u>Operator</u>	<u>Meaning</u>	<u>Example</u>
<	Is less than	X<Y
<=	Is less or equal to	X<=Y
>	Is greater than	X >Y
>=	Is greater or equal to	X >= Y
==	Is equal to	X==Y
!=	Not equal to	X !=Y

Note: When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the result is compared. That is, arithmetic operators have a higher priority than relational operators. Relational operators are used in decision making statements such as if and while to decide the course of action of a running program. On these topics we study in next chapter.

Example of use of relational operator:

```
/*write a program that finds out the greatest  
number between two given integer.*/  
  
#include<stdio.h>  
#include<conio.h>  
  
void main()
```



```
{  
    int a,b;  
    printf("please enter the value of a:  ");  
    scanf("%d",&a);  
    printf("\nenter the value of b:  ");  
    scanf("%d",&b);  
    if(a>b)  
        printf("\n%d is greater than %d.",a,b);  
    else  
        printf("\n%d is greater than %d.",b,a);  
    getch();  
}
```

sample output of the program:

```
please enter the value of a:  12  
enter the value of b:  10  
12 is greater than 10.
```

3. LOGICAL OPERATORS:

If we have to combine two or more relational operations into a single expression then we use a logical operator which evaluates either true or false. Since it combines more than two relational expressions therefore it is also called compound relational operator. There are mainly three types of the logical operators which are:

&& meaning logical **AND**

|| meaning logical **OR**

! meaning logical **NOT**

The AND and OR operators are widely used but NOT operator is rarely used. Therefore we are not going to study this operator in detail.

If 1 represents true and 0 represents false then truth table of these operators will be as:

<u>X</u>	<u>Y</u>	<u>Z=X & Y</u>
0	0	0
0	1	0
1	0	0
1	1	1

Table: truth table of **AND** operator

<u>X</u>	<u>Y</u>	<u>Z=X Y</u>
0	0	0
0	1	1
1	0	1
1	1	1

Table: truth table of

OR operator

From truth it is clear that in the logical AND operator if all the expressions are true only then it produces true result other wise result will be false. Similarly in case of logical OR operator if at least one expression is true then it produces true result. It produces false result only when all the expressions are false. Therefore in the case where all the conditions should be true then we have to use logical AND operator and the case where at least one condition is to be true then we have to use logical OR operator.

Examples:

/*Write a program that checks whether the number entered by the user is exactly divisible by 4 but not by 11.*/

TIPS: Before writing the code for this program we should know type of the operator is to be used. For this, problem should be known very clearly. In this problem both the conditions mentioned (number is should be divisible by 4 and number should not be divisible by 11) must be true. Therefore in this case we have to use logical AND operator.

Source code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int number;
    printf("Please enter a number:  ");
    scanf("%d",&number);
    if((number%4==0) && (number%11!=0))
    printf("\n%d is exactly divisible by 4 but not by 11.");
    else
    printf("\nThe condition is false,");
    getch();
}
```

Sample output:

```
Please enter a number:  12
12 is exactly divisible by 4 but not by 11.
```

Please enter a number: 44

The condition is false.

/*write a program which find out whether the given year is leap year is or not. */<Bex> [6] marks

Before writing the code for this problem, we should know what type of year is called leap year or what is leap year? Leap year the year which is perfectly divisible by 4 but not by 100 or those year which is perfectly divisible by 400. Before writing the code for any program we should know the problem i.e. definition of the problem, what is the problem and what are the logic should be implemented to solve that problem. Only then we can write the proper program.

```
Solution:
#include<stdio.h>
#include<conio.h>
void main()
{
    int year;
    printf("please enter the year:  ");
    scanf("%d",&yeat);
    if ((year % 4 == 0 && year % 100 != 0) || year %
400 == 0)
        printf("%d is a leap year\n", year);
    else
```



```
printf("%d is not a leap year\n", year);  
getch();  
}
```

In the program the conditions $year \% 4 == 0$ and $year \% 100 != 0$ should be true. Therefore they are combined by the logical AND (&&) operator. In solving this problem another condition $year \% 400 == 0$ also valid. It means either upper condition or lower condition, both conditions are applicable to check whether the year is leap year or not. Therefore these conditions are combined by the logical OR (||) operator.

4. ASSIGNMENT OPERATOR:

The name of operator “assignment” says it is the operator which assigns something for a variable. Simple equal to sign (=) is called assignment operator which assigns a value or expression for a variable. It should take care that double equal to sign (= =) and assignment operator (=) are not same.

Example:

$a=b+c;$

$b=a\%b;$

$x=10;$

$y='a';$ etc.

C shorthand:

C has a set of shorthand assignment operators. Shorthand assignment operator is the combination of two different arithmetic operators. The assignment of shorthand takes the form:

$V_{ao} = \text{exp};$

Where v represents variable, ao represents arithmetic operator and exp represents expression. Here $V_{ao} = \text{exp};$ is equivalent to $V = V_{ao} \text{exp};$

Different types of shorthand operators are listed in the table:

Short hand	Equivalent meaning
$A += B;$	$A = A + B;$
$A -= B;$	$A = A - B;$
$A *= B;$	$A = A * B;$
$A /= B;$	$A = A / B;$
$A \% = B;$	$A = A \% B;$

Unary operator:

Unary means one. Therefore that type of operator which takes only one operand is called unary operator. There are different types of unary operators among them some important are as follows:

Increment and decrement operator:

The operator which increases the value of a variable by unity is called increment operator and the operator which decrease the value of variable by unity is called decrement operator. Increment operator is ++ and decrement operator is --. The increment operator ++ adds 1 to its operand where as the decrement operator -- subtracts 1 from its operand.

Eg. i++; which is equivalent to i=i+1; and i--; which is equivalent to i=i-1;

The unusual aspect is that ++ and -- may be used either as **prefix operators** (before the variable, as in ++n), or **postfix operators** (after the variable: n++). In both cases, the effect is to increment n. But the expression ++n increments n *before* its value is used, while n++ increments n *after* its value has been used. This means that in a context where the value is being used, not just the effect, ++n and n++ are different. If n is 5, then

```
x = n++;
```

sets x to 5, but

```
x = ++n;
```

sets x to 6. In both cases, n becomes 6.

NOTE: The increment and decrement operators can only be applied to variables; an expression like (i+j)++ is illegal.

To understand this matter let's take an example:

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

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

```
void main()  
{  
    int a=5,b;  
    b=a++;  
    printf("\nb=%d",b) ;  
    b=++a;  
    printf("\nb=%d",b) ;  
    getch() ;  
}
```

output of this program will be:

b=5

b=7

Explanation of the program: In the program at first 5 is assigned to **a**. The statement **b=a++**; firstly assign the value of **a** and then increases the value of **a** by 1. Therefore 5 is stored on **b** and value of **a** will be 6 and **b=5** will be printed. But in the statement **b=++a**; firstly value of **a** is increased by 1 i.e. value of **a** will be 7 which further assigned to **b**. therefore value of **b** will be 7 and hence **b=7** will be printed.

Unary minus sign:

This is simply minus sign which changes the value of variable with negative. In the binary minus i.e. in arithmetic minus sign it takes two operands which subtract the second operand from first one but in the case of unary it takes only one operand. For example:

-n; is a statement.

C) **CONDITIONAL OPERATOR** (ternary operator? :):

This is different type of operator. It is combination of if – else statement as a condition operator. Therefore it is also called conditional statement. The statements

```
if (a > b)
    z = a;
else
    z = b;
```

compute in z the greatest of a and b. The *conditional expression*, written with the ternary operator "?:", provides an alternate way to write this and similar constructions. The general syntax of this type of expression is:

expr₁? expr₂: expr₃

Here the expression *expr₁* (here which is conditions) is evaluated first. If it is non-zero (true), then the expression *expr₂* is evaluated, and that is the value of the conditional expression. Otherwise *expr₃* is evaluated, and that is the value. Only one of *expr₂* and *expr₃* is evaluated not both at a time. Thus to set z to the maximum of a and b,

```
z = (a > b)? a: b;
```

The conditional operator takes three operands; therefore we can not say that it is a binary operator, it is ternary operator.

It should be noted that the conditional expression is indeed an expression, and it can be used wherever any other expression can be. If *expr₂* and *expr₃* are of different types, the type of the result is

determined by the conversion rules. For example, if *f* is a float and *n* an int, then the expression

`(n > 0) ? f : n`

is of type float regardless of whether *n* is positive. Parentheses are not necessary around the first expression of a conditional expression, since the precedence of `?:` is very low, just above assignment. They are advisable anyway, however, since they make the condition part of the expression easier to see.

Write a program that finds out the greatest number among two numbers using conditional operator.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b;
    int greater;
    printf("Please enter any two numbers:  ");
    scanf("%d%d",&a,&b);
    (a>b)? greater = a : greater = b;
    printf("\n\ngreater number is %d",greater);
    getch();
}
```

Explanation of the program:

In this program if *a* is greater than *b* then it assigns *a* as *greater* otherwise *b* is assigned to *greater*. And finally *greater* in which is greatest number is assigned is printed.

Among these operators which are mentioned there are other types of operators are also available in C program such as bitwise operator, special operators (sizeof operator, pointer operator, comma operator, member selection operator etc). But important operators are explained above. Therefore we are not going to study other operators.

Precedence and Associativity of operators in C:

Each operator in C has its own precedence associated with it. The Precedence means in the expression containing more than one operator, which one is evaluated first and then after. There are distinct levels of precedence and operator belongs to one of the level. The operators at higher level of precedence are evaluated first. The operators of the same precedence are evaluated either from left or from right depending from the level. This is known as Associativity of an operator.

Precedence of Arithmetic operators and unary operators:

Operators	Associativity	Precedence
++ , --	Right to left	Highest
- (unary minus)	left to right	
*, /, %	Left to right	
+, -	Left to right	Lowest

Precedence of Relational operators and Logical operators:

Operators	Associativity	Precedence
-----------	---------------	------------

!	Right to left	Highest
> , >=, < , <=	left to right	
= , !=	Left to right	
&&	Left to right	
	Left to right	Lowest

Summary of precedence of operators:

Operators	Associativity	Precedence
() [] -> .	left to right	Highest
! ~ ++ -- + - * (type) sizeof	right to left	
* / %	left to right	
+ -	left to right	
<< >>	left to right	
< <= > >=	left to right	
== !=	left to right	
&	left to right	
^	left to right	
	left to right	
&&	left to right	
	left to right	
?:	right to left	
= += -= *= /= %= &=	right to left	
^= = <<= >>=		
,	left to right	Lowest

Data input and output:

The important of C programming is its ability to fascinate the user as far as possible. To do this we should able to handle input and output (I/O). All the input/output operations are carried out through function call such as `printf()`, `scanf()` etc.

These types of functions which are used to read and display the data are collectively known as standard I/O library functions. Each program that uses these types of functions should include the standard header file `<stdio.h>` which stands for standard input output header file. It is included in the program by writing the code `#include<stdio.h>` which tells the compiler to search a file named `stdio.h` and place its content at the point of the program. The contents of the file became part of the source code when it is compiled.

For inputting the data (which is called reading of data in C program), the input function `scanf()` is used which can read the data from a terminal. In the same way for displaying (outputting) the output function '`printf ()`' is used. Hence the `printf()` function moves the data from computer memory to standard output device i.e monitor whereas `scanf()` function enters the data from standard output device i.e. keyboard and stores it in computer main memory.

We have studied about this earlier, therefore we are not going to discuss further on this topic.

More data input and out put functions are as follows:

Data input functions:

- a) `getch ()`
- b) `getche ()`
- c) `getchar ()`
- d) `gets ()` etc.

Data output functions:

- a) `putchar ()`
- b) `putc ()`
- c) `puts ()` etc.

Escape sequence characters:

Those characters which escape the sequence of the data are called escape sequence characters. The complete set of escape sequences is

Character	Function	Character	Function
<code>\a</code>	alert (bell) character	<code>\\</code>	backslash
<code>\b</code>	Backspace	<code>\?</code>	To print question mark
<code>\f</code>	form feed	<code>\'</code>	to print single quote

<code>\n</code>	new line	<code>\"</code>	to print double quote
<code>\r</code>	Carriage return	<code>\ooo</code>	octal number
<code>\t</code>	horizontal tab	<code>\xhh</code>	hexadecimal number
<code>\v</code>	Vertical tab		

The character constant `'\0'` represents the character with value zero, the null character. `'\0'` is often written instead of `0` to emphasize the character nature of some expression, but the numeric value is just `0`.

Use of `getchar ()` and `putchar ()` functions:

Both functions are character holding functions that means they can process only one character at a time. By the name of the function `getchar ()`, it can be guess that this function is used to read a character from standard input device i.e. keyboard and actually it is. Similarly the function `putchar ()` is used to put or print a character on the monitor. The general syntax of these functions is:

```
variable = getchar( );
```

```
putchar(variable);
```

Example: write a program which reads a character from the keyboard and print in upper case.

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main( )
{
char ch;
printf("please      enter      a
character:  ");
ch=getchar( );    /*reading a
character */
ch=toupper(ch);    /*changing
the character in to upper
case*/
putchar(ch);    /*printing the
character*/
getch( );
}
```

We can write in place of
putchar(ch); the statement
printf("%c",ch); both result same
output.

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main( )
{
char ch;
int var;
printf("please
enter          a
character:  ");
ch=getchar( );
var=toascii(ch);
var-=32;
putchar(var);
getch( );
}
```

Use of gets() and puts():

As in the case of getchar() and putchar() where 'char' represents character in the same way here in gets () and puts (), 's' represents string. String is set of character. In detail about the string we will study later in the section Arrays. For this instant let us understand string as a set of character only. These functions are string functions which are used in manipulation of strings i.e. reading and writing the string.

Control statement:

C program consists of set of instructions which are executed in sequentially order as written in any high level programming language. But many cases may arise where we have to choose the statement to be executed. At this instant we require the instruction which transfer the control of the program from one point to another point in the program and sometimes we have to repeat the same task for certain number of times. C has this type of facility. This type of statement (instruction) which transfers the control of the execution of the program from one point to another point is called control statement.

C supports following types of control statements:

- a) Sequential control structure
- b) Decision making control statement
 - 1) simple if statement
 - 2) if – else statement
 - 3) if-else-if statement
 - 4) nested if statement
 - 5) conditional operator statement
 - 6) switch statement
- c) Iteration / looping statements
 - 1. while loop
 - 2. do-while loop

3. for loop

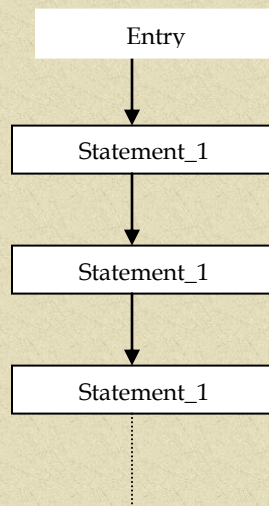
d) special control statement

1. goto statement
2. break statement
3. continue statement

Sequential control statement:

It is the flow of the statements in sequential order that means as written in the source code. The sequence control statement ensures that the instructions in the program are executed in the same order as they appeared in the program.

Flow chart of sequential control statement is:



Decision making control statement:

Those statements or instructions which are used to make some decisions are called decision making statement. These statements are also called as selection statement. Using these statements we can select the required statement to be executed. In this type of structure the test condition decides the flow of the execution. Here we describe the different types of decision making statements in detail:

a) Simple if statement:

By name of this statement, we can guess that this statement is used for making decision and therefore this statement is also known as decision making statement. Here **if** is key word (reversed word). The general syntax of this function is:

```
if(condition)           // Here if should be in lower case.

{
true statement(s);
}
next statement;
```

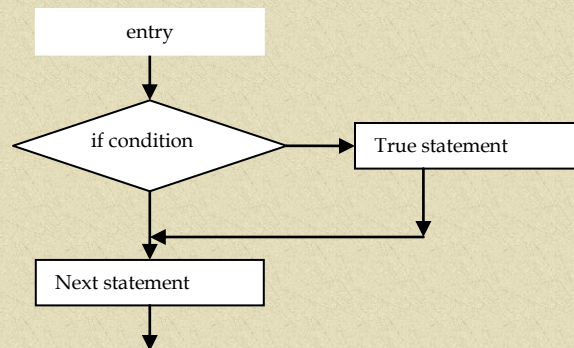


fig: flow chart of simple if statement

Explanation of the syntax and flow chart:

Here, initially condition (which is logical condition) is checked. If the condition is true then **true statement** enclosed by a pair of braces is executed and then **next statement** will be executed otherwise (i.e. if the condition is false then) without executing the true statement directly **next statement** will be executed as indicated in the syntax or flow chart. Here if the condition is false then it leaves the some statement and transfers the control of the program from one point to another point. Therefore this statement is called control statement.

Examples:

`/*write a program to check whether the number entered by the user is even or odd*/`

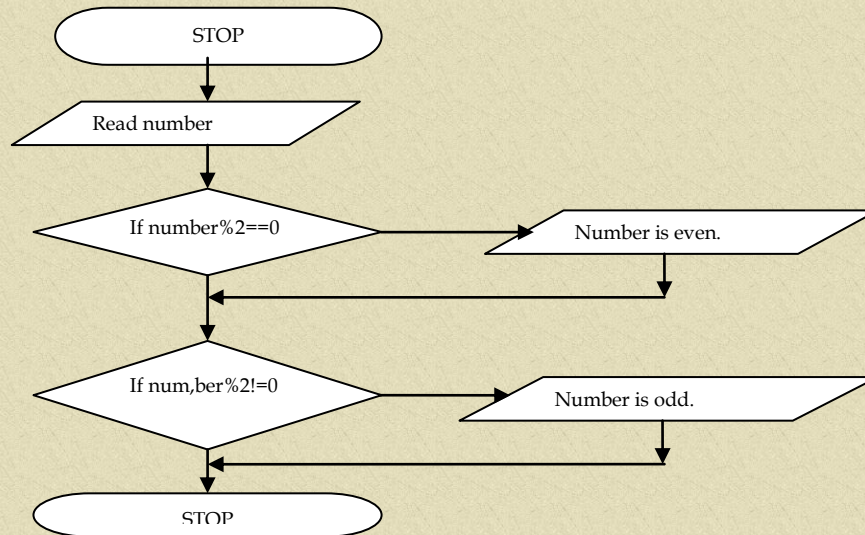
Before writing the code for this problem we should know what type of number is called even number and what type of number is called odd number. The number which is perfectly divisible by 2 is called even number and remaining other are called odd number. On the basis of

this clue we have to write a complete program. Before writing the code, let us write algorithm and flow chart of the program.

Algorithm of the program:

4. START the program.
5. Read a number.
6. If number is perfectly divisible by 2 then print “the number is even.”
7. If number is not divisible by 2 then print “the number is odd.”
8. STOP the program.

Flowchart of the program:



Source code:

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



```
void main()
{
int number;
printf("enter a number:  ");
scanf("%d",&number);
if(number%2==0)
printf("\nentered number is even.");
if(number%2!=0)
printf("\nentered number is odd);
getch();
}
```

Sample output of the program:

```
enter a number:    12
entered number is even.
```

```
enter a number:    3
entered number is odd.
```

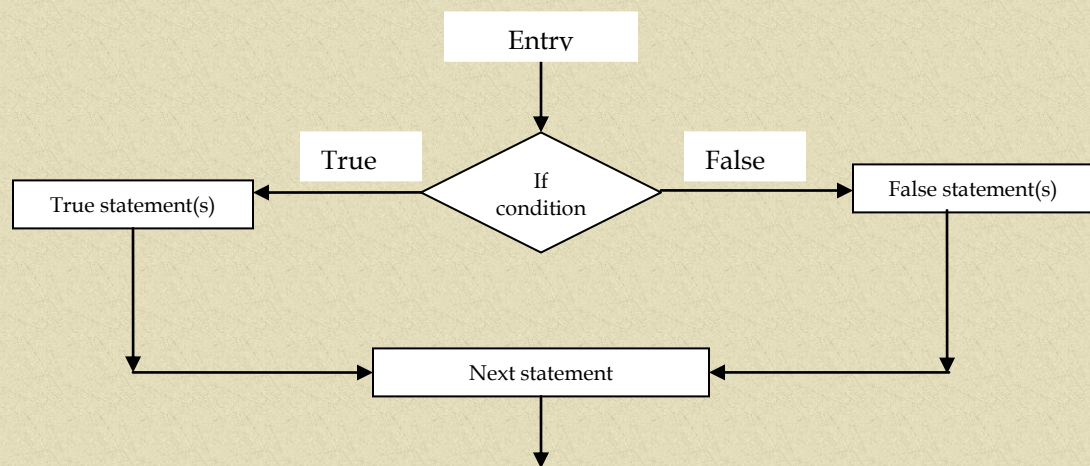
b) if-else statement:

“if– else” statement is the real decision making statement. In simple if statement there is no alternative i.e. if the condition is false then there is no any alternative but in case of if else statement, there is alternative. Here else is key word. The general syntax of this statement is:

```
if(condition)
{
```

```
        true_statement(s);  
    }  
    else  
    {  
        false_statement(s);  
    }  
    next_statement;
```

Flow chart of this statement:



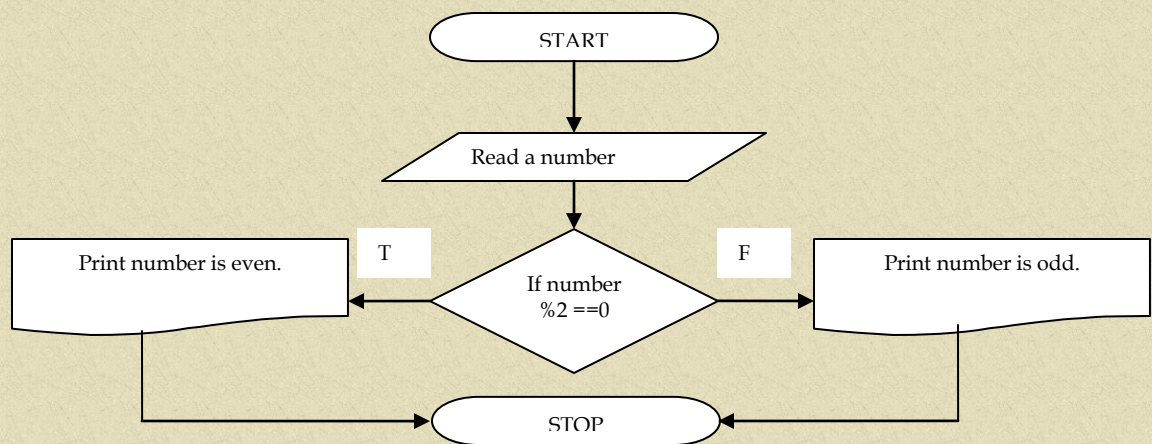
Explanation of syntax and flow chart:

Here, firstly the condition is checked. If the condition is true then the true statement(s) will be executed as in the simple if statement otherwise false statement(s) will be executed. That means either true or false, only one statement will execute not both at a time. After executing one of the statements, then next statement will be executed.

/*draw a flow chart and write source code of a program which finds out the whether the entered number is odd or even.*/

Answer:

Flowchart of the program:



Source code for program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int number;
    printf("please enter a number:  ");
    scanf("%d",&number);
    if(number%2==0)
    printf("\nEntered number is even.");
    else
```

```
printf("\nEntered number is odd.");  
getch();  
}
```

Sample output:

```
please enter a number: 13  
Entered number is odd.
```

```
please enter a number: 18  
Entered number is even.
```

c) if-else-if statement (if-else ladder):

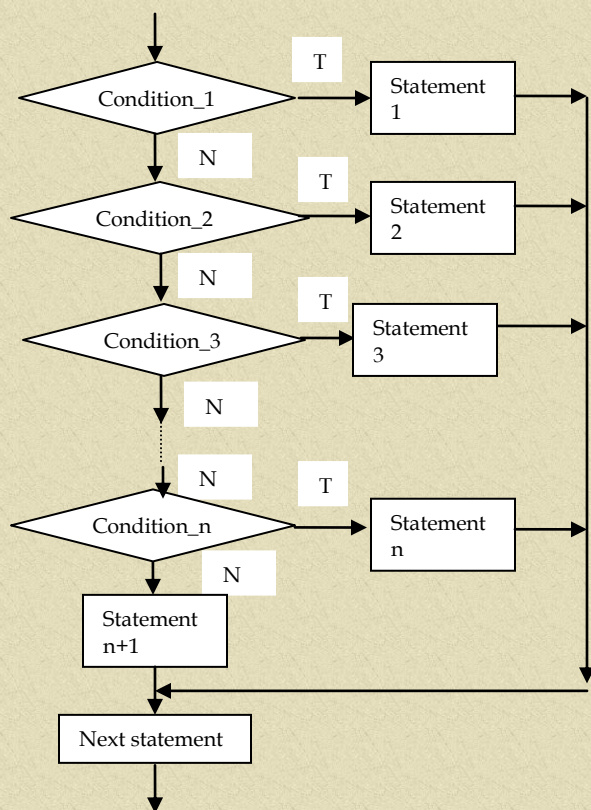
In this form of if statement, conditions are checked first. If the condition is found to be true then the statements associated with the condition will be executed and other conditions will be skipped. If none of the conditions are true, then it will execute the statement of else side.

It takes the following form of syntax:

```
if(condition1)  
{  
statement1;  
}  
else if(condition2)  
{  
statement2;  
}  
else if(condition3)  
{
```

```
statement3;  
    }  
else .....  
...  
...  
else if(condition_n)  
    {  
statement_n;  
    }  
else  
    {  
statement_n+1;  
    }  
next_statement;
```

Flow chart of if-else-if statement:



Write a program that takes average marks obtained by a student. Your program should display the result and division of that student.

If average marks ≥ 80 , distinction

If average marks ≥ 60 but < 80 , first division

If average marks ≥ 45 but < 60 , second division

If average marks ≥ 35 but < 45 , first division

To pass the average marks should be greater or equal to 35.

Solution:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    float ave;
    printf("enter the average marks obtained by
    the student:\t");
    scanf("%f",&ave);
    if(ave>=80)
        printf("\nresult=                pass.\ndivision=
        distinction.");
    else if(ave<80&&ave>=60)
        printf("\nresult=                pass.\ndivision=    first
        division.");
    else if(ave<60&&ave>=45)
```

```
printf("\nresult=      pass.\ndivision=" second
division.");
else if(ave<45&&ave>=35)
printf("\nresult=      pass.\ndivision=" third
division.");
else
printf("\nresult=      fail.\ndivision=" no
division.");
getch();
}
```

d) Nested if statement:

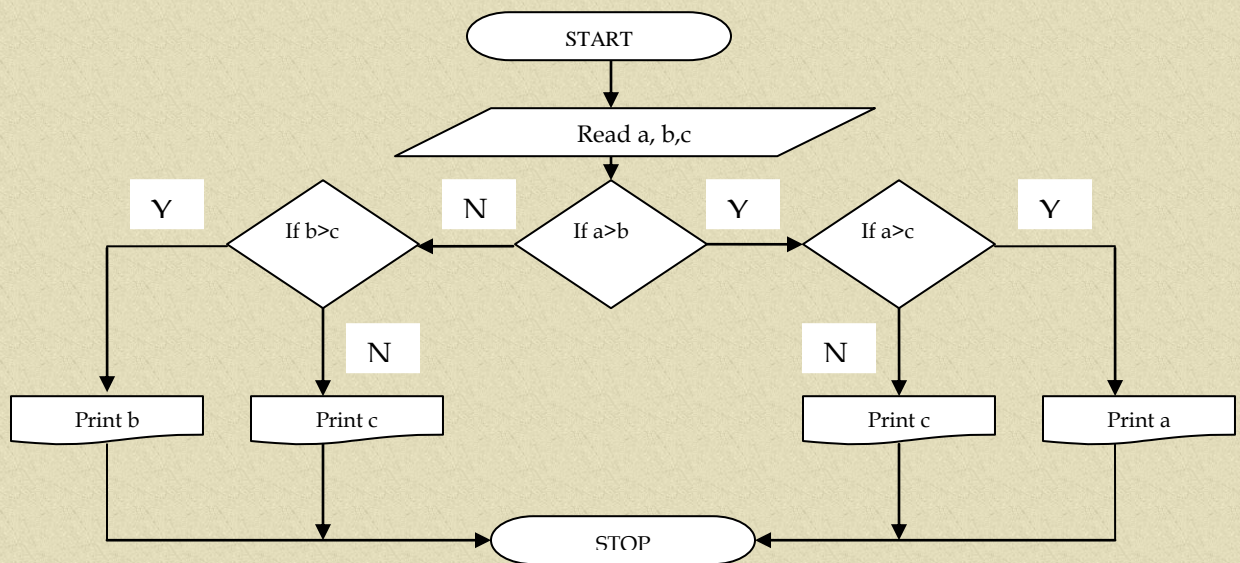
If an “if” –statement contains if statement within the statement then such type of condition is called nested if condition. Nested if is an if that has another if in its if’s body or in its else’s body. In other word, “if else” statement within another “if-else” body is known as nested if. The general syntax of this type of statement is:

```
if(condition)
{
    if(condition)
    {
        statement(s);
    }
    else
    {
        statement(s);
    }
}
else
{
}
```

```
statement(s);  
}
```

Examples:

Design a program in C with flow chart that reads three numbers a, b, c and prints the largest number among them using nested if structure.



Source program:

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

```
if(a>b)
{
    if(a>c)
        printf("\n%d is largest",a);
    else
        printf("\n%d is largest.",c);
}
else if(b>a)
    printf("\n%d is largest.",b)
else
    printf("%d is largest.", c);
getch();
}
```

e) Conditional operator statement:

The name of the statement type suggests that it is that type of statement which uses conditional operator for the operation. Simply this is the combination of simple if else statement. Syntax of this type of statement is:

Expression1 ? expression 2: expression 3;

In this if the expression1 is true then expression2 is evaluated otherwise expression3 will be evaluated. This is equivalent to:

```
if(expression1)
    expression2;
else
    expression3;
```

Example:

Write a program that finds out the greatest number among two numbers using conditional operator statement.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b;
    int greater;
    printf("Please enter any two numbers:  ");
    scanf("%d%d",&a,&b);
    (a>b)? greater = a : greater = b;
    printf("\n\ngreater number is %d",greater);
    getch();
}
```

f) Switch statement:

If we have to create a program of menu system then we can use switch statement. This is similar to if-else-if statement. Use of this statement makes the program more users friendly and more standard. The general syntax of this statement is:

```
switch( choice)
{
    case 1:
```



```
        statement(s);  
        break;  
case 2:  
        statements(S);  
        break;  
.....  
.....  
.....  
default:  
        statement(S);  
        break;  
}
```

if the choice is of character type i.e. alphabet then incase of writing case 1, case 2,..... we have to write case 'a', case 'b', etc.

Explanation of the syntax of the switch statement:

If the choice is 1, then it only executes the statements written in case 1 block, then it directly transfers the control of the program out of the switch statement. If choice is not matched then it executes the statements written in default block and transfers the control out of statement.

Example: write a menu type program that reads two integer's value and asks for the choice. If you press 1 then your program should add them and display, and similarly on pressing 2 subtraction, on pressing 3 multiplication, on pressing 4

division and on pressing 5 remainder should display. If any other than 1-5 is pressed then your program should display an appropriate message.

```
#include<stdio.h>
#include<conio.h>
void mian()
{
    int a,b,res, choice;
    flaot div;
    printf("please enter any two integer numbers:
    ");
    scanf("%d%d",&a,&b);
    printf("\n\n\n\n\n");
    printf("\n1.      addition\n2.      subtraction\n3.
    multiplication\n4. division\n5. remainder");
    printf("\n\nplease enter your choice(1-5):\t");
    scanf("%d",&choice);
    switch(choice)
    {
    case 1:
        res=a+b;
        printf("\naddition    of    given    numbers    is
        %d",res);
        break;
    case 2:
```

```
        res=a-b;
        printf("\n subtraction of given numbers is
%d",res);
        break;
case 3:
        res=a*b;
        printf("\n multiplication of given numbers is
%d",res);
        break;
case 4:
        res1=(float)a/b; //type casting
        printf("\ndivision of given numbers is
%d",res1);
case 5:
        res=a%b;
        printf("\nremainder after dividing %d by %d is
%d",a,b,res);
        break;
default:
        printf("\n your choice id out of range. Please
try again.");
    }
    getch();
}
```

sample output of the program:

please enter any two integer numbers: 12 4

- 1.addition
- 2.subtraction
- 3.multiplication
- 4.division
- 5.remainder

please enter your choice(1-5): 3

multiplication of given numbers is 48

please enter any two integer numbers: 12 4

- 1.addition
- 2.subtraction
- 3.multiplication
- 4.division
- 5.remainder

please enter your choice(1-5): 7

your choice is out of range. Please try again.

Write a program to calculate area of triangle,
circle and sphere using switch-case statements.

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

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

```
void main()
{
char ch;
float area;
printf("a.    area    of    triangle.\nb.    area    of
circle.\nc. area ofrectangle\n\n");
printf("\n enter your choice (a-c):");
ch=getchar();
switch(ch)
{
case 'a':
float base, height;

printf("enter base and height of the triangle:
");
scanf("%f%f",&base,&height);
area=.5*base*height;
printf("\narea of the triangle is : %f",
area);
break;
case 'b':
float radius;
printf("\nenter the radius of the circle: ");
scanf("%f",&radius);
area=3.14159*radius*radius;
printf("\narea of the circle is %f",area);
```



```
        break;
case 'c':
    float length,breadth;
    printf("\nenter the lenght and breadth of the
rectangle:  ");
    scanf("%f%f",&length,&breadth);
    area=length*breadth;
    printf("\narea    of    the    rectanlge    is:
%f",area);
    break;
default:
    printf("\n Wrong entry. Please enter valid
choice.");
    break;
}
getch();
}
```

sample output:

```
    a.area of triangle
    b.area of circle
    c.area of rectangle
please enter your choice (a-c): 2
Wrong entry. Please enter valid choice.
```

```
a.area of triangle
b.area of circle
c.area of rectangle
please enter your choice (a-c): b
enter radius of the circle 12.15
area of the circle is 490.873437
```

Explanation of the program:

There is confusion in the program which is that we are saying that no declaration can be done on the execution part of the program but here we are declaring the various variables inside the execution part of the program. We can do this but it should take care that they work only inside that block only.

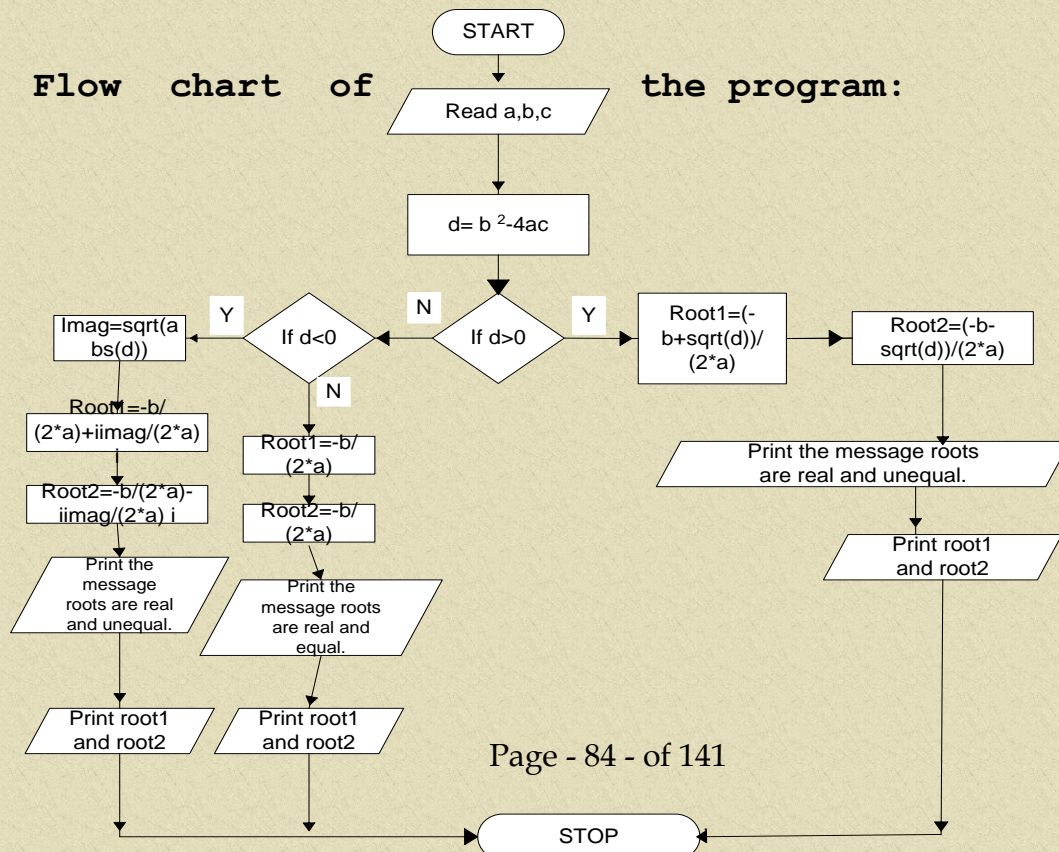
Write algorithm, flow chart and program to find roots of a quadratic equation. Suppose that (b^2-4ac) has three different conditions. <2062Baishakh DCT> [10]

Answer:

Algorithm of the program:

1. START the program.
2. Read the coefficient of x^2 , x and constant and store them in a, b, c .
3. calculate $d=b^2-4ac$
4. If d is greater than 0, then do this otherwise go to step 5.

- a. $\text{root1} = (-b+d)/(2*a)$
 - b. $\text{root2} = (-b-d)/(2*a)$
 - c. Print the message roots are real and unequal.
 - d. Print root1 and root2
5. If d is lesser than 0 then do this otherwise go to step 6.
- a. $\text{Imag} = \text{sqrt}(\text{abs}(d))$
 - b. Print the roots are imaginary and they are:
 - i. $\text{Root1} = -b/(2*a) + (\text{imag}/(2*a))i$
 - ii. $\text{Root2} = -b/(2*a) - (\text{imag}/(2*a))i$
6. if d is equal to 0 then do this:
- a. $\text{root1} = -b/(2*a)$
 - b. $\text{root1} = \text{root2}$
 - c. print the message roots are equal and real
 - d. print root1 and root2
7. STOP the program.



SOURCE CODE OF THE PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    flaot a,b,c;
    flaot root1,root2,imag;
    pritnf("\nPlease enter the coefficient of x^2,
x and constant:\t");
    scanf ("%f%f%f",&a,&b,&c);
    d=b*b-4*a*c;
```

```
        if(d>0)
    {
        root1= (-b+sqrt(d))/(2*a);
        root2= (-b-sqrt(d))/(2*a);
        printf("\n\nroots are real and unequal. They
are %f and %f",root1,root2);
    }
else if(d<0)
{
    imag=sqrt(abs(d))/(2*a);
    real=-b/(2*a)
    printf("\n\nroots are imaginary and they are
%f+%fi and %f-%fi",real,imag,real,imag);
}
else
{
    root1=(-b/(2*a));
    root1=root2;
    printf("\n\nroots are real and equal. They are %f
and %f.",root1,root2);
}
getch();
}
```

c) Iteration / looping statements:

If we have to do same task for several times then use of sequential programming approach becomes more difficult, time consuming and less use friendly. In this situation we need a type of statement which can perform the same task for specified times. This type of statement is called looping statement and the process is called iteration. In general, these types of statements include four things: *1. setting or initializing a condition variable. 2. Execution of the statements inside the loop. 3. Updating the condition variable and 4. Test the specified value of the condition variable.* Here the condition may be either to determine whether the loop has been repeated the specified number of times or to determine whether a particular condition has been reached.

There are mainly three types of looping statements used in C and they are:

- a) **while loop/statement**
- b) **do-while statement**
- c) **for statement**

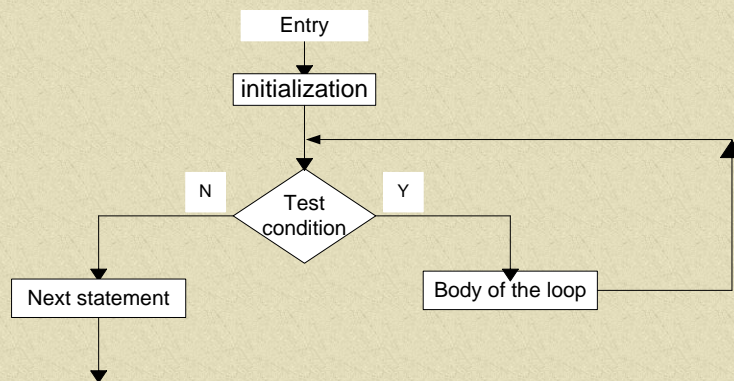
1. The while statement:

This is a looping statement where the statements that written inside the loop executed till the condition is true. The general syntax of this looping statement is:

```
Initiliazation;  
while (test_expression/condtion)  
{  
statements;           //body of the loop
```

```
expression;  
    }  
next_statement;
```

Flow chart of this statement:



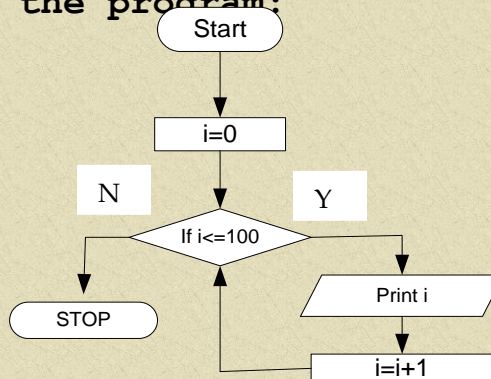
The “**while statement**” is an *entry-controlled loop* statement. That means the execution of the loop is wholly depends upon the entry condition. At first the **test condition** is evaluated and if the condition is true, then the body of the loop is executed. After the execution of the body, the test condition is again tested and again it yields true then again the loop will be executed. This process will go on continuously until the test condition yields result as false. When the test condition become false then the control of the program will transfer just outside the loop and then **next_statement** will be executed. The body of the loop may or may not contain more than one statement. The pair of braces will only needed of the body of the loop contains two or more than two statement i.e. if the body of the loop is

compound statement. But for good practice it is better to put pair of braces even for a single statement.

Example: **write a program to print the numbers from 0 to 100.**

Ans:

Flow chart of the program:



Source code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i;
    printf("the      required      series      is      given
    below:\n\n\n");
```

```
i=0;           //initialization
while(i<=100)   //test condition
{
printf("%d\t",i); // body of the loop
i++;           //expression
}
getch();
}
```

Using while loop only, write a program to print the following series untill the term value is less than 750. The series is: 1,2,5,10,17,26,.....<BEX 2063, Baishakh> [8]

Answer:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i=0,term;
term=0;
while (term<=750)
{
term=i*i+1;
i++;
printf("%d\t",term);
}
```

```
    }  
    getch();  
}
```

Using while statement, write a program to find out the factorial value of a number entered by the user.

ANSWER:

```
#include<stdio.h>  
#incldue<conio.h>  
void main()  
{  
    int num,i;  
    long int fact=1;  
    pritrnf("\n  please  enter  the  number  whose  
    factorial value is to be calculated:\t");  
    scanf("%d",&num);  
    i=1;  
    while(i<=num)  
    {  
        fact=fact*i;  
        i++;  
    }  
    printf("\n\n\nfactorial    value    of    %d    is  
    %ld.",num,fact);  
    getch();  
}
```



```
}
```

Sample output:

```
please enter the number whose factorial value is  
to be calculated:10  
factorial value of 10 is 3628800.
```

Using while statement, write a program to evaluate the equation $y=x^n$ where n is non-negative integer.

Answer:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int x,y,n;
    int i;
    printf("\nenter the value of x and n:\t");
    scanf("%d%d",&x,&n);
    i=1;
    y=1;
    while(i<=n)
    {
        y=y*x;
        i++;
    }
    printf("\n\n\nRequired result is %d",y);
    getch();
}
```

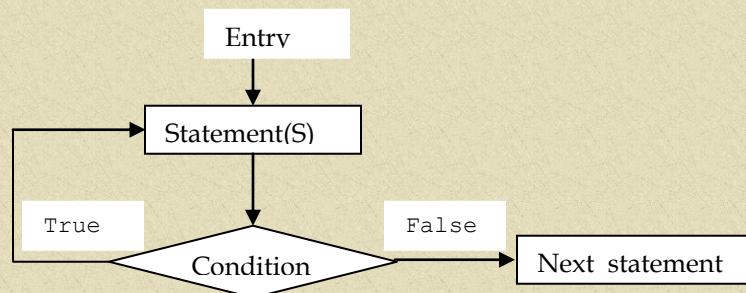
}

2. do-while statement:

This statement is also used to repeat some statements for specified no of times. In many cases even the condition is failed, we have to do that work at least one time. This problem is fascinated by **do-while** statement. In the name of this statement, “**do**” means perform the work or execute the statements and “**while**” means until the condition is satisfied. That means in “**do-while**” statement, **statements are executed first and then condition is checked**. Therefore this statement is exit controlled statement, because the condition for the loop is evaluated after the execution of the statements. The general syntax of this looping structure is:

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

The **flow chart** for this statement is:



In the syntax of **do-while** statement it should be take care that, the while statement should be end with semicolon. This is also the difference between while and do-while loop.

The differences between the while and do-while statement:

While loop	Do-while loop
1. It is an entry controlled loop. That means the entry point decides whether to enter into the loop or not.	1. It is exit controlled loop. That means the exit point of the loop decides whether to again enter into the loop or not.
2. The test condition is tested outside the loop before entering into the loop.	2. The test condition is tested within the loop before exiting the loop.
3. If the condition is not satisfied then the statements inside the loop will not be	3. Even the condition is not satisfied at least one time the loop will be executed.

executed.	
-----------	--

Examples of do-while statement:

Write a program that prints the given series and their sum.

1,4,9,16,25,..... up to the term value less than 1500.

Answer:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,sum=0,term;
printf("\nThe required series is:\n\n");
i=1;
do
{
term=i*i;
sum+=term;
i=i+1;
printf("%d\t",term);
}while(term<=1500);
printf("\n\nsum of the series is:  %d",sum);
getch();
}
```

3. The For loop/statement:

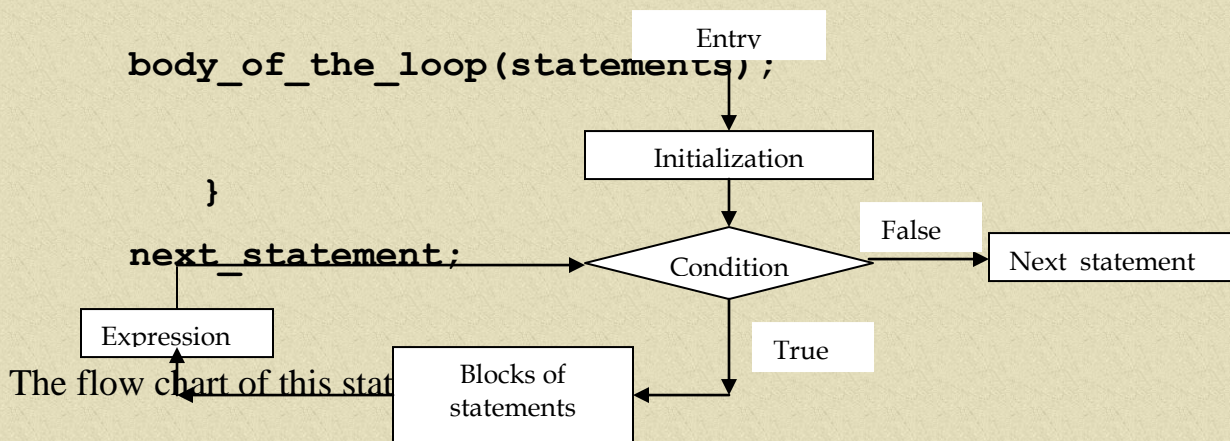
This “**for**” loop is one of the most popular loops among other two loops in C programming. As in the while loop, it also allows to define condition to repeat the block of statements for number of times. In **while** loop we can not initialize the value of condition variable but in **for** loop we can do it. For many cases while and for loop are very similar to each other. This is also entry controlled statement but in some cases it is neither entry controlled nor exit controlled. This is also called as counter controlled statement. The general **syntax of for** loop is:

```
for(initialization;test_condition;expression)
{
```

```
body_of_the_loop(statements);
```

```
}
```

```
next_statement;
```



Explanation of the syntax and flow chart of for loop:

When the control of the program enters in for loop, at first it initializes a value on a variable and then checks the condition. If the condition is true then it

enters into the loop and body of loop (i.e. blocks of statements) will be executed. After the execution of body of loop, the expression will be executed and then again checks the condition. And same job will repeat until the condition is failed. If the condition is failed then the control of the program doesn't reach the body of loop, and directly transfer to the next_statement.

Examples:

Write a program that generates the following series using for loop.

1,2,5,10,17,26,.....up to 20th term.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,term=20,value;
for(i=0;i<term;i++)
{
value=i*i+1;
printf("%d\t",value);
}
getch();
}
```

WRITE A PROGRAM THAT FINDS OUT THE GREATEST NUMBER AMONG 100 NUMBERS.

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

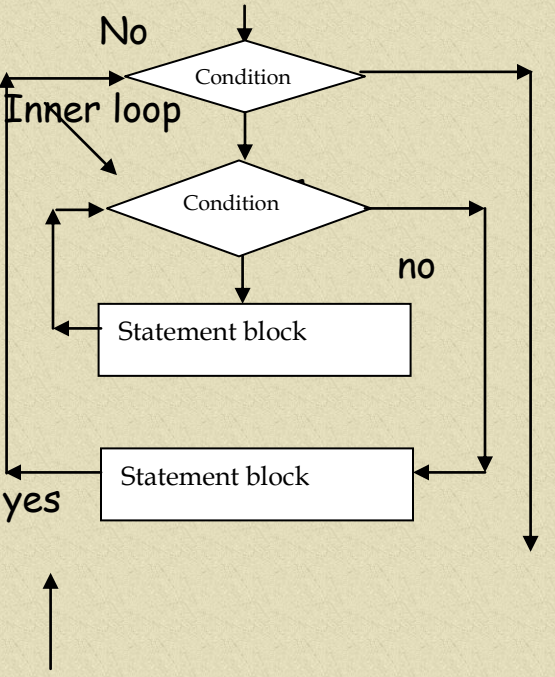
```
#include<conio.h>
void main()
{
int i,num,greatest;
for(i=1;i<=100;i++)
{
printf("\nPlease enter %dth number:\t");
scanf("%d",&num);
if(i==1)
{
greatest=num;
continue;
}
if(greatest<num)
greatest=num;
}
printf("\n\nthe greatest number is %d.",greatest);
getch();
}
```

Nested loops:

If a looping statement contains another looping statement in its body, then this type form of loop is called nested loop. The occurrence of one loop within another loop is known as nested loop.

Rules for constructing the nested loop:

1. As outer loop and inner loop can not have same control variable.
2. The outer loop starts first and then only inner loop starts.
3. The inner loop must terminate before the termination of the outer loop.

Flowchart	Syntax:
 <pre> graph TD Start(()) --> OuterCond{Condition} OuterCond -- No --> InnerCond{Condition} OuterCond -- yes --> OuterSBlock[Statement block] InnerCond -- No --> OuterSBlock InnerCond -- yes --> InnerSBlock[Statement block] InnerSBlock --> InnerCond OuterSBlock --> OuterCond </pre> <p>Outer loop</p> <p>Inner loop</p>	<pre> For(condition) { for(condition) { statement_block; } statement_block; } </pre> <p>inner loop</p> <p>outer loop</p>

e) Special control statement/ Jumping statement:

By the name of this statement we can understand, what does this statement? Simply these statements are used to jump the statements as our requirement. Therefore this is called as jumping statement. There are mainly three types of jumping statements which are:

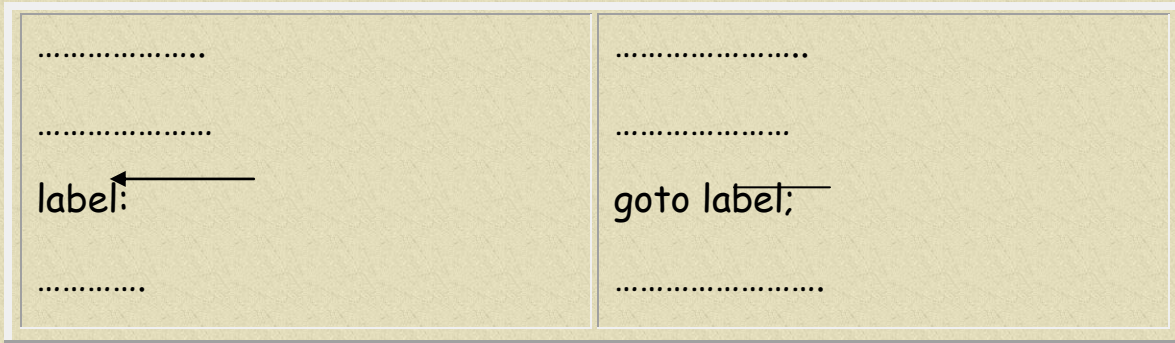
1. goto statement
2. break statement
3. continue statement

1. The goto statement

This statement is used to transfer the control of the program anywhere within the program. Using this statement we can break the sequence of execution of the program without having any test condition and transfers the control to the statement having the statement label rather than any other statements. On the basis of transference of the control within the program, we can assume the two types of goto statement. They are:

- i) forward goto statement
- ii) backward goto statement

Syntax of forward goto statement	Syntax of backward goto statement
<pre>goto label;</pre>	<pre>label: ←</pre>



While programming it is better not to use goto statement because use of this statement makes the program poor. The use of goto statement in the program is considered as poor programming. Use of this statement is accepted in only few statement. In the examination you are requested to don't use this statement, if you use this statement then you will not awarded full marks.

Example:

Write a program that generates the multiplication table of a number entered by the user without using any looping structure. (Use goto statement).

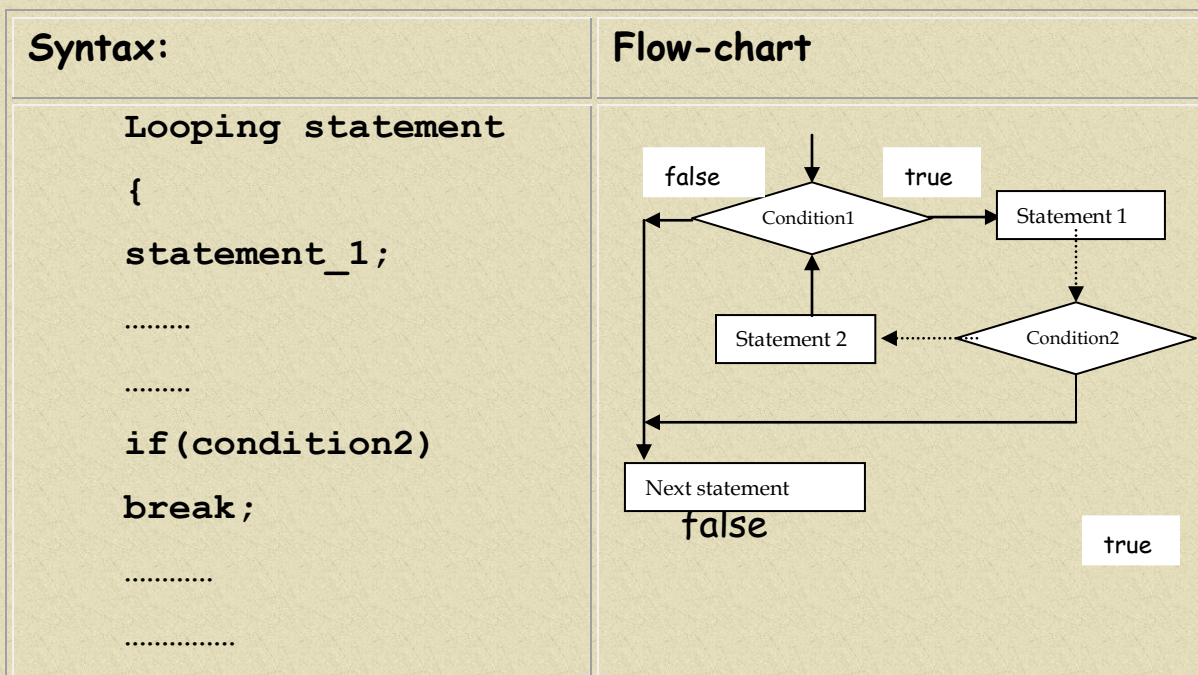
```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i=1,num,mul;
    printf("\nplease enter the number:\t");
    scanf("%d",&num);
    up:
    mul=i*num;
```



```
i++;
printf("\n%d*%d=%d", num, i, mul);
if(i<=10)
goto up;
getch();
}
```

2. The break statement:

This statement is used to break or stop the loop or to escape from switch () statement. Especially the break statement is used to end the infinite loop whose number of execution is not known. Generally this statement comes after if statement because in which condition we have to stop the loop that condition is must be checked and if the condition is satisfied then loop is to be stopped. The general syntax of the break statement is:



```
statement_2;  
}  
next_statement;
```

Explanation of the syntax or flow chart:

When the execution of the program reach the loop firstly condition1 is checked, if the condition is true only then it enters into the loop. After entering into the loop it executes the statement_1 and other statements. In flow-chart, the dashed arrow head indicates that there may be other set of statements. Then it checks the condition2. If this condition is true then it executes the break statement which transfers the control of execution directly outside the loop i.e. to next_statement with out executing the remaining statements after the break statement. If the condition is failed then it executes the statements up to the statement_2, after the break statement. And then again condition1 is evaluated and similar task will be repeated. Those looping structures in which break statement is used, there the number of times of execution of loop depends up on the loop condition i.e. condition1 or break condition i.e. condition2.

Example:

Write a program that displays the first natural numbers until a number is encountered which is multiple of both 5 and 4.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i;
    for(i=1 ; ;i++ )
    {
        if(i%4==0&& i%5==0)
            break;
        printf("%d\t",i);
    }
    getch();
}
```

Output of the program:

1	2	3	4	5	6	7
8	9	10				
11	12	13	14	15	16	17
18	19					

3. The continue statement:

This is also another jumping statement. The “continue” statement and the “break” statement are seemed to be similar but there is huge difference between them. The “continue” statement is used to escape the certain statement

if some specified condition is satisfied and again return to the loop. The break statements stops the loop if the condition for the break statement is true but in the case of continue statement loop will not be stopped. This is the major difference between these two statements. The general syntax and flow chart of the continue statement is:

Syntax	Flowchart
<pre> Loop statement { statement_1; if(condition2) continue; statement_2; } next_statement; </pre>	<pre> graph TD Entry(()) --> Cond1{Condition1} Cond1 -- yes --> S1[Statement 1] S1 -.-> Cond2{Condition2} Cond2 -- yes --> S2[Statement 2] S2 --> Next[Next_statement] Next --> Cond1 Cond2 -- no --> Exit(()) Cond1 -- no --> Exit </pre>

Explanation of the flow chart or syntax:

As in the break statement when the control of the execution reaches the loop statement, firstly it evaluates condition1. If the condition1 is true only then it

enters into the loop. After entering into the loop, it executes the statement1 block and checks the condition2. If the condition2 is true then it executes the continue statement and transfers the control directly to the loop without executing the remaining statements after continue statement. And again loop will be continued. If the condition2 is false then the statements after the continue statements will be executed only then loop will be continued. In this type of looping where only continue statement is used the number is execution time of loop is depends only on the loop condition i.e. condition1.

Example:

Write a program that prints the numbers from 1 to 15 except 7 and 11.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i;
    for(i=1;i<=15;i++)
    {
        if(i==7||i==11)
            continue;
        else
            printf("%d\t",i);
    }
    getch();
}
```



```
}
```

There are also other jump statements such as **return** and **exit**. The detail on return statement will be studied in function and exit statement is used to terminate the whole program at our required point.

Write a program that reads two numbers. If the numbers are multiple of both 3 and 4 terminate the program with an appropriate message other wise display the sum and difference between the numbers.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>          //for exit() statement
void main()
{
    int a,b;
    int sum,dif;
    printf("\nplease enter any two integer number:\t");
    scanf("%d%d",&a,&b);
    if(a%3==0&&a%4==0&&b%3==0&&b%4==0)
    {
        printf("\nYou have entered the numbers which are
multiple both 3 and 4");
        printf("\n\nPlease enter any to exit the program:
");
        getch();
        exit();
    }
}
```

```
    }  
else  
{  
    sum=a+b;  
    dif=a-b;  
    printf("\nsum of the numbers is:  %d",sum);  
    printf("\ndifference between the entered numbers is  
%d",dif);  
    getch();  
}
```

UPTO HERE

- 1. Write a program to calculate the volume of a pool. Assume that the pool has length of 25 ft, a width of 10 ft and depth of 6 ft.**

Solution:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int length, width, depth;
    int volume;
    length=25;
    width=10;
    depth=6;
    volume=length*width*depth;
    printf("\n\nvolume of the pool is %d",volume);
    getch();
}
```

- 2. Write a program to print sum, product and quotient of two numbers entered by the user.**

Solution:

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

```
#include<conio.h>
void main()
{
float a,b;
float sum, quotient, product;

printf("\nplease enter any two numbers: ");
scanf("%f%f",&a,&b);
sum=a+b;
product=a*b;
quotient=a/b;
printf("\n\nsum of %f and %f is %f",a,b,sum);
printf("\n\nproduct of %f and %f is %f",a,b,product);
printf("\n\nquotient of %f and %f is %f",a,b,quotient);
getch();
}
```

3. Write a program to compute the area of sphere by reading the radius of the sphere.

Solution:

```
#include<stdio.h>
#include<conio.h>
#define PI 3.14159
```

```
void main()
{
float radius;
float area;
printf("enter the radius of the sphere:  ");
scanf("%f",&radius);
area=4*PI*radius*radius;
printf("\n\nArea  of  sphere  having  radius  %f  is
%f.",radius, area);
getch();
}
```

4. Write a program that will ask for the name and marks obtained by a student in seven different subjects and display the percentage score of that student.

Solution:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char name[15];
int marks, total;
float percentage;
printf("enter the name of the student:  ");
```



```
gets(name);
total=0;
for(i=1;i<=7;i++)
{
    printf("\nEnter the marks in %dth subject:
    ",i);
    scanf("%d",&marks);
    total+=marks;
}
percentage=(int) total/7;
printf("\n Percentage scored by %s is
%f.",name,percentage);
getch();
}
```

5. Write a program to input length and breadth of a room and print its area and perimeter.

Solution:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    float length, breadth;
    float area, perimeter;
```

```
printf("\nPlease enter the length and breadth of  
the room:\t");  
scanf("%f%f",&length,&breadth);  
area=length*breadth;  
perimeter=2*(length+breadth);  
printf("\n\nArea of the room is %f.",area);  
pritrnf("\n\nperimeter of the room is  
%f.",perimeter);  
getch();  
}
```

6. Write a program to calculate simple interest for the total amount p kept in bank for n years at the rate of r per annum.

Solution:

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
float p,n,r,simple_interest;  
printf("\nenter the total amount kept in bank:\t");  
scanf("%f",&p);  
printf("\nenter the time for which amount is  
kept:\t");  
scanf("%f",&n);
```

```
printf("\nenter the annum rate of the interest:\t");
scanf("%f",&r);
simple_interest=p*n*r/100;
printf("\n\nsimple      of      the      deposit      is
%f",simple_interest);
getch();
}
```

7. Write a program to calculate compound interest.

Solution

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    float p,n,r,I,temp;
    printf("\nplease enter value of p,n and r:\t");
    scanf("%f%f%f",&p,&n,&r);
    temp=1+r/100;
    I=p*pow(temp,n);
    printf("\n\ncompound interest of amount %f is
%f",p,I);
    getch();
}
```

value and then print them.

Solution:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,c;
    printf("\nenter the value of a:\t");
    scanf("%d",&a);
    printf("\nenter the value of b:\t");
    scanf("%d",&b);
    clrscr();
    printf("\nbefore      interchanging:\na=%d\t      and
b=%d.",a,b);
    c=a;
    a=b;
    b=c;
    printf("\nafter      interchanging:\na=%d\t      and
b=%d.",a,b);
    getch();
}
```

9. Write a program to read n numbers and find out their sum and average.

Solution:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int number, sum;
    int i,n;
    float average;
    sum=0;
    printf("\nEnter the value of n:  ");
    scanf("%d",&n);
    i=1;
    while(i<=n)
    {
        printf("\nEnter %dth number:  ",i);
        scanf("%d",&number);
        sum+=number;
    }
    average=(float) sum/number;
    printf("\n\nSum of the entered is %d",sum);
    printf("\nAverage of the number is %f",average);
    getch();
}
```


10. Write a program that count the number of digits present in the integer number entered by the user and sum of the digits.

Solution:

```
#include<stdio.h>
#include<stdio.h>
void main()
{
    int num,count=1,sum=0,rem;
    printf("\nenter a integer number:\t");
    scanf("%d",&num);
    do
    {
        rem=num%10;
        num=num/10;
        sum+=rem;
        if(num>0)count++;
    }
    while(num>0);
    printf("\n\nno. of digits in the number entered:
    %d",count);
    printf("\nsum of the digits =%d",sum);
    getch();
```

```
}
```

11. Design a program in C to prepare multiplication table according to the value given from the key board using do-while loop.

Solution:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,i;
    printf("enter a number whose multiplication table is
    to be displayed:\t");
    scanf("%d",&n);
    clrscr();
    printf("\nmultiplication of %d is:\n\n\n");
    i=1;
    do
    {
        printf("\n%d*d=*d",i,n,i*n);
        i++;
    }while(i<=10);
    getch();
}
```

12. Design a program in C to print the series 5, 10, 15,.....10th term using for loop.

Solution:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i=;
    printf("\nthe required series is:\n");
    for(i=1;i<=10;i++)
        printf("%d\t",i*5);
    getch();
}
```

13. Write a program that reads two integers a & b and then compute a^b .

Solution:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,i,result=1;
    printf("\nenter the value of a and b:\t");
    scanf("%d%d",&a,&b);
    for(i=1;i<=b;i++)
```

```
result*=a;
printf("\nThe result of the program is
                                         %d",result");
                                         getch();
                                         }
```

14. Write a program for Nepal electricity Board, where user can supply name, address, and total unit of electricity consumption. Your program should print the bill using the following rates:

<u>Unit</u>	<u>rates</u>
-------------	--------------

For the first 20units rs 75

For the next 50 unit Rs 6.5 per unit for next next units Rs 8 per unit. Service charge is Rs 100, which is compulsory.

Solution:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    float units, rs;
    char name[15],address[15];
    printf("\nenter the name: ");
```

```
scanf("%s",name);
printf("\nenter the address:  ");
scanf("%s",address);
printf("\nenter the total units consumed:  ");
scanf("%f",&units);
rs=100;
if(units<=20)
    rs+=75;
else if(units>20 &&units<=70)
    {
units=units-20;
rs+=75+units*6.5;
    }
else
    {
units=units-70;
rs+=75+6.5*50+units*8;
    }
clrscr();
printf("\nname of the consumer : %s",name);
printf("\nAddress of the consumer:  %s",address);
printf('\ntotal amount to be paid:  %f',rs);
getch();
}
```


14. Design a program that finds out whether the given number is Prime or not.

/* a number is said to be prime if the number is not multiple of other any number except 1 and that number. */

Solution:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int num,i;
    clrscr();
    printf("please enter the number:\t");
    scanf("%d",&num);
    for(i=2;i<=num;i++)
    {
        if(num%i==0)
        {
            printf("\n\nThe entered number %d is not prime
            number.",num);
            break;
        }
    }
    if(i==num)
    printf("\n\n\nThe entered number %d is prime
    number.");
}
```

```
getch();  
}
```

15. Write a program to input an integer number and find the sum of all digits of number. Suppose supplied number is 1234, then the sum should be $1+2+3+4=10$.

solution:

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    int num,dig,sum=0;  
    printf("\nPlease enter an integer:\t");  
    scanf("%d",&num);  
    for( ; ; )  
    {  
        dig=num%10;  
        num=num/10;  
        sum+=dig;  
        if(num==0)  
            break;  
    }  
    printf("\n\n the sum of digits is %d",sum);  
    getch();  
}
```

16. Write a program that reverses and sums up the digits of an integer.

Solution:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int num,dig,sum=0,rev=0;
printf("\nPlease enter an integer:\t");
scanf("%d",&num);
for( ; ; )
{
dig=num%10;
num=num/10;
sum+=dig;
rev*=10+dig;
if(num==0)
break;
}
printf("\n\n the reverse of the given number is %d
and the sum of its digits is %d",reverse, sum);
getch();
}
```

//here the "for loop" is constructed for looping only.

Sample output of the program:

Please enter an integer: 2345

The reverse of the given number is 5432 and the sum of its digits is 14

17. Write a program using nested loop to print the multiplication table for even numbers from 2 to 10.

Solution:

```
#include<conio.h>
#include<stdio.h>
void main()
{
int i,j,mul;
for(j=1;j<=10;j++)
{
for(i=2;i<=10;i=i+2)
{
mul=i*j;
printf("%d*d=%d\t",i,j,mul);
}
printf("\n");
}
getch();
```

}

SAMPLE OUTPUT OF THE PROGRAM

2*1=2	4*1=4	6*1=6	8*1=8	10*1=10
2*2=4	4*2=8	6*2=12	8*2=16	10*2=20
2*3=6	4*3=12	6*3=18	8*3=24	10*3=30
2*4=8	4*4=16	6*4=24	8*4=32	10*4=40
2*5=10	4*5=20	6*5=30	8*5=40	10*5=50
2*6=12	4*6=24	6*6=36	8*6=48	10*6=60
2*7=14	4*7=28	6*7=42	8*7=56	10*7=70
2*8=16	4*8=32	6*8=48	8*8=64	10*8=80
2*9=18	4*9=36	6*9=54	8*9=72	10*9=90
2*10=20	4*10=40	6*10=60	8*10=80	10*10=100

18. Write a program to display the following series:

**1 5 9 13 Up to the term
value is less than 75.**

Solution:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
for(i=1; ;i=i+4)
{ if(i>75)
```



```
        break;
        printf("\t%d",i);
    }
    getch();
}
```

sample output:

	1	5	9	13	17	21
25	29	33	37			
	41	45	49	53	57	61
65	69	73				

19. Write a program to find factorial of a supplied number.

Solution:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int num;
    int i;
    long int fact=1;
    printf("\nPlease enter a number whose
    factorial is to be determined:\t");
    scanf("%d",&num);
    printf("\n\n\nfactorial of %d is :\n");
```

```
for (i=1;i<=num;i++)
{
    fact*=i;
    if (i<num)
        printf("%d*",i);
    else
        printf("%d",i);
}
printf("=%ld",fact);
getch();
}
```

Sample output of the program:

Please enter a number whose factorial is to be
determined: 6

factorial of 1326 is :

1*2*3*4*5*6=720_

20. Write a program to display this pattern:

```
111111
11111
1111
111
11
1
```

Solution:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,k=1;
    for(i=1;i<=6;i++)
    {
        printf("\n");
        for(j=6-i;j>=0;j--)
            printf("%d",k);
        getch();
    }
}
```

21. Write a program to display this pattern:

```
1
12
123
1234
12345
123456
```

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

```
{
    int i,j;
    for(i=1;i<=6;i++);
    {
        printf("\n");
        for(j=1;j<=i;j++)
            printf("%d",j);
    }
    getch();
}
```

22. Write a program that read two numbers and display all the even numbers between them.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,a,b,temp;
    clrscr();
    printf("\nPlease enter the two numbers:\t");
    scanf("%d%d",&a,&b);
    if(a>b)
    {
        temp=a;
        a=b;
        b=temp;
    }
```

```
    }  
    printf("\nThe even numbers between the range %d to  
    %d are:  \n",a,b);  
    for(i=a;i<=b;i++)  
    {  
        if(i%2==0)  
            printf("%d\t",i);  
    }  
    getch();  
}
```

sample output of the program:

Please enter the two numbers: 17 124

The even numbers between the range 17 to 124 are:

18	20	22	24	26	28	30
32	34	36				
38	40	42	44	46	48	50
52	54	56				
58	60	62	64	66	68	70
72	74	76				
78	80	82	84	86	88	90
92	94	96				
98	100	102	104	106	108	110
112	114	116	118	120	122	124

23. Write a program that displays alphabets from a to z.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i;
    clrscr();
    for(i='a';i<='z';i++)
    {
        printf("%c\t",i);
    }
    getch();
}
```

sample output of the program:

a	b	c	d	e	f	g
h	i	j				
k	l	m	n	o	p	q
r	s	t				
u	v	w	x	y	z	

24. Write a program that displays alphabets from A to Z.

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

```
void main()
{
int i;
clrscr();
for(i='A';i<='Z';i++)
{
printf("%c\t",i);
}
getch();
}
```

sample output of the program:

A	B	C	D	E	F	G
H	I	J				
K	L	M	N	O	P	Q
R	S	T				
U	V	W	X	Y	Z	

25. Write a program that assigns a word westernregioncampus and display it in the following pattern.

W
We
Wes
West
Weste
Wester

Western

Western

Western r

Western re

Western reg

Western regi

Western regio

Western region

Western region

Western region c

Western region ca

Western region cam

Western region camp

Western region campu

Western region campus

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

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

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

```
void main()
```

```
{
```

```
int i,j,len;
```

```
char string[]="western region campus";
```

```
len=strlwr(string);
```

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

```
{
```

```
for(j=0;j<i;j++)  
printf("%c",string[j]);  
printf("\n");  
}  
getch();  
}
```

26. Write a program to calculate the total salary of the employee with the following conditions:

basic salary	HRA	DA
--------------	-----	----

Rs<1500	0	0
>=1500 & <2300	10%	3%
>=2300 & <3500	15%	5%
>=3500 & <5000	20%	7%
>=5000	25%	700

Total Salary= Bs+HRA+DA */

```
#include<stdio.h>  
#include<conio.h>  
#include<string.h>  
void main()  
{  
char nam[50],post[50];  
int bs;  
float hra,da,ts;  
clrscr();
```

```
printf("Enter the employee name:  ");
scanf("%s",nam);
printf("\nEnter the employee's post:  ");
scanf("%s",post);
printf("\nEnter basic salary of the employee:
");
scanf("%d",&bs);
    if(bs<1500)
    {
        hra=0;
        da=0;
    }
if (bs>=1500&&bs<2300)
    {
        hra=bs*10/100;
        da=bs*3/100;
    }
if (bs>=2300&&bs<3500)
    {
        hra=bs*15/100;
        da=bs*5/100;
    }
if (bs>=3500&&bs<5000)
    {
        hra=bs*20/100;
        da=bs*7/100;
```



```
    }  
    if (bs>=5000)  
    {  
        hra=bs*25/100;  
        da=700;  
    }  
    ts=bs+hra+da;  
    clrscr();  
    printf("\nName          of          employee:  
    %s",strupr(nam));  
    printf("\nPost:      %s",strupr(post));  
    printf("\nBasic Salary:  %d",bs);  
    printf("\nHRA:    %.2f",hra);  
    printf("\nDA:     %.2f",da);  
    printf("\nTotal salary:   %.2f",ts);  
}
```

27. write a program to read the age of 100 persons and count the no.
of peresons

in the group as below:

0-10	40-50
10-20	50-60
20-30	60-70
30-40	70-80 */

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,n,a=0,b=0,c=0,d=0,e=0,f=0,g=0,h=0,age[150];
for(i=1;i<=100;i++)
{

printf("\nEnter the %dth age:  ");
scanf("%d",&age[i]);
if(age[i]>=0&&age[i]<10)
a++ ;
if(age[i]>=10&&age[i]<20)
b++;
if(age[i]>=20&&age[i]<30)
c++;
if(age[i]>=30&&age[i]<40)
d++;
if(age[i]>=40&&age[i]<50)
e++;
if(age[i]>=50&&age[i]<60)
f++;
if(age[i]>=60&&age[i]<70)
g++;
if(age[i]>=70&&age[i]<80)
h++;
}
clrscr();
printf("\nNo. of persons in the group (0-10):  %d",a);
printf("\nNo. of persons in the group (10-20):
%d",b);
printf("\nNo. of persons in the group (20-30):
%d",c);
printf("\nNo. of persons in the group (30-40):
%d",d);
printf("\nNo. of persons in the group (40-50):
%d",e);
printf("\nNo. of persons in the group (50-60):  %d",f);
```

```
printf("\nNo. of persons in the group (60-70): %d",g);  
printf("\nNo. of persons in the group (70-80): %d",h);  
getch();  
}
```

Some important theory questions for the final examination:

1. What is programming language? Describe briefly different types of programming language and their major features.
2. What do you mean by computer software? Briefly describe different types of software.

3. Write down the differences between compiler and interpreter. Explain in detail about compiling process.
4. What are the steps that need to be followed for developing the application software?
5. What is flow chart? List out the various commonly used flow chart symbols. How does a flow chart help a computer programming?
6. What are the basic features of the C programming language? What are its advantages compared to other high level languages?
7. What is an identifier? Describe about valid and invalid identifier. Why it is necessary to use meaningful identifier in programming?
8. What is an operator in C language? Discuss the different types of operators used in C.
9. What data type and qualifiers are available in C? Explain with examples.
10. Explain in brief about the different types of operators available in C language.
11. What is conditional statement? Explain with examples.
12. Explain syntax, working and uses of switch statement in C? What are the restrictions on case labels?

13. What do you mean by repetitive structure? Differentiate between definite and indefinite loops with an example of each.
14. Draw the flow chart and syntax of *if.....else* statement and explain how it works?
15. Why c is called a structured programming language? Write about control statement in C language.
16. Explain along with flow chart, how do-while (exit controlled) loop differs from while (entry controlled) loop?
17. What is type casting? Explain with example.
18. Differentiate between source code and object code.
19. Draw the flow chart and syntax of *do.....while* loop, explain how it works?
20. Explain syntax, working and uses of for loop in C.
21. Explain the differences between *break* and *continue* statement with examples.
22. What is operator precedence and associativity? Explain the precedence and associativity of arithmetic operators with examples.
23. What do you mean by logical and relational operator? Differentiate between them.
24. What do you mean by nested loop? Explain about it.