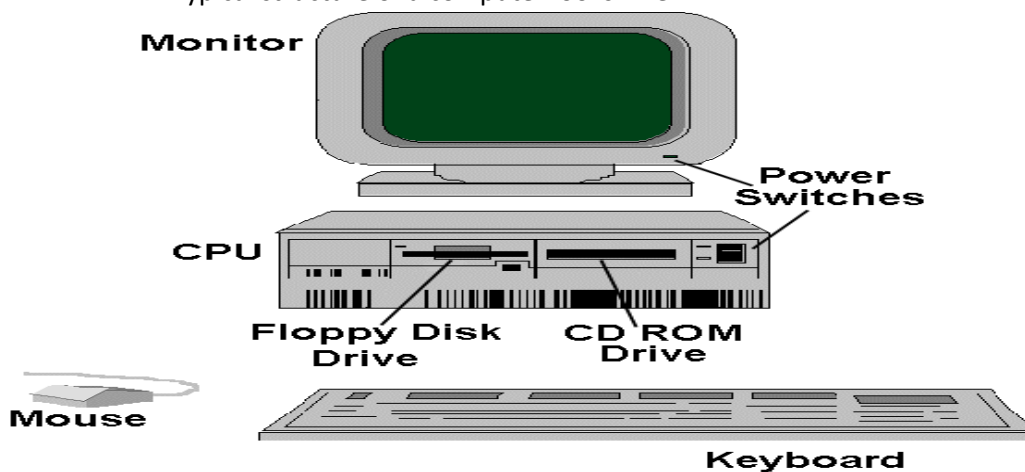


Subject: **Computer Programming in C and data Structures**

Module(1&2) e-notes

What is a Computer?

- Computer
 - Device capable of performing computations and making logical decisions
 - Computers process the data under the control of sets of instructions called computer programs
 - Computer Consists of two parts they are Hardware and Software
 - Hardware
 - Various devices comprising a computer
 - Keyboard, screen, mouse, disks, memory, CD-ROM, and processing units, mother board etc..
 - Software
 - Programs that run on a computer
- Typical structure of a computer looks like



Working principle of a computer involves following steps

Instruction phase

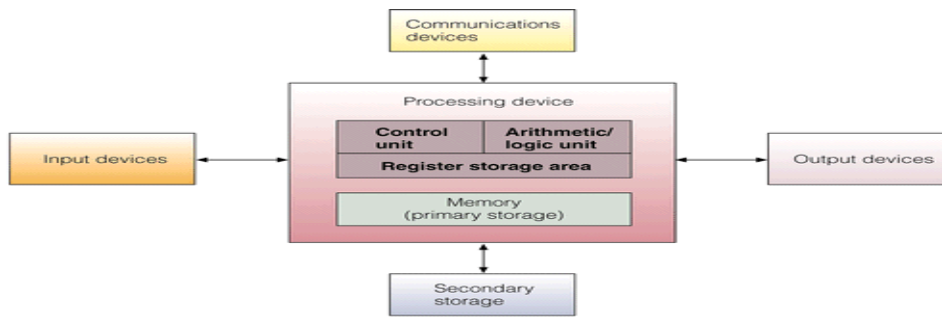
- **Step 1: Fetch instruction from the memory**
- **Step 2: Decode instruction**

Execution phase

- **Step 3: Execute the instruction**
- **Step 4: Store the results**

Now the question is how to write such Instructions?

Hence we require a programming language to communicate with machine by writing instructions. Instruction can be written in Machine language or assembly language or High level language.



Naturally a language is the source of communication between two persons, and also between person to machine like computer. The languages we can use to communicate with the computer are known as Computer programming languages.

Generally there are three major types of languages are available and as follows:

1. Machine languages

- Strings of numbers giving machine specific instructions
- Example:

+1300042774
+1400593419
+1200274027

2. Assembly languages

- English-like abbreviations representing elementary computer operations (translated via assemblers)
- Example:

Load BASIC
Add Basic,da,gross
Move gross,total

3. High level languages

- The set of commands available in high level language is very simple and easy to understand.

- Code is similar to everyday use of English sentence
- Use mathematical notations (translated via compilers)
- Example:

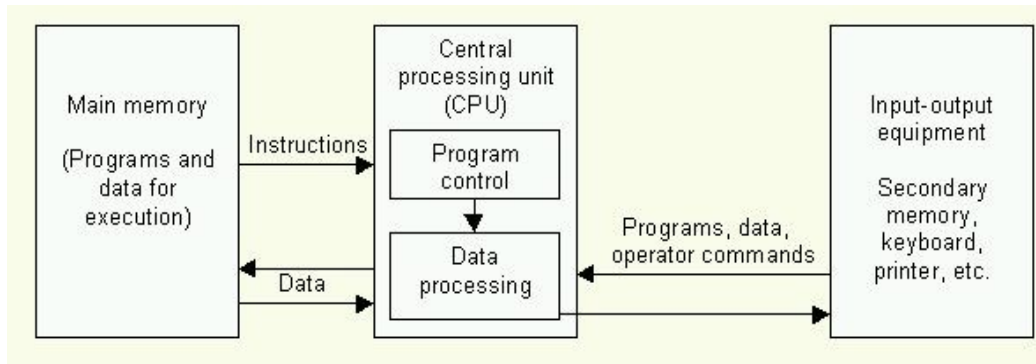
Totalsal = basic + da

“High-level” is a relative term, C programming language is a relatively low-level and also high-level language. Pascal, Fortran, COBOL, Java etc are typical examples for high-level languages. Application specific languages are Matlab, Javascript, VBScript etc.

What is Programming ?

- *A programming is a tool for developing executable models for a class of problem domains.*

- A programming language is a notational system for describing computation in a machine-readable and human-readable form.
- A vocabulary and set of grammatical rules for instructing a computer to perform specific tasks. It usually refers to high-level languages, such as BASIC, C, C++, etc...
- Computers are based on the stored program concept given by Von Neumann. It consists of Central Processing Unit(CPU) main memory, input output devices, ports, buses etc.,
- Actions performed by CPU are written through program. Hence Program is a sequence of instructions.



Tools such as flowcharts, Algorithm and pseudocodes are used to develop program

■ Pseudocodes:

Since each programming language uses a unique syntax structure, understanding the code of multiple languages can be difficult. Pseudocode helps this problem by using *conventional syntax* and *basic English* phrases that are universally understood

Pseudocode is an *informal* program description that does not contain code syntax or underlying technology considerations. It summarizes a program's steps (or flow) but excludes underlying details. Hence, By describing a program in pseudocode, programmers of all types of languages can understand the function of a program.

Example

```

If student's marks is greater than or equal to 35
    Print "pass"
else
    Print "fail "

```

• Algorithm:





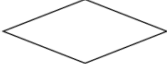



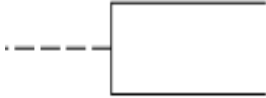
It is an effective step-by-step procedure to perform the solution to a given problem.

It can be expressed using notations. Natural language like English is used to write these steps.

- **Flowcharts:**

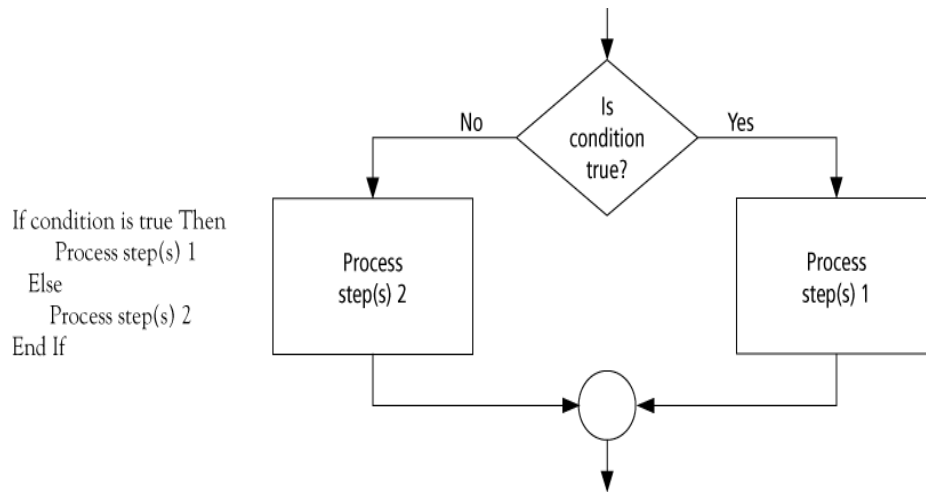
- It is a diagram showing a sequence of activities to be performed for the solution of a problem.
- A set of conventional symbols are used to draw flowcharts
- Graphically depicts the logical steps to carry out a task and shows how the steps relate to each other.
- An organized combination of shapes, lines, and text that graphically illustrates a process or structure
- Flowchart is a pictorial representation showing all the steps of a process

Some of the symbols used to design flowcharts are

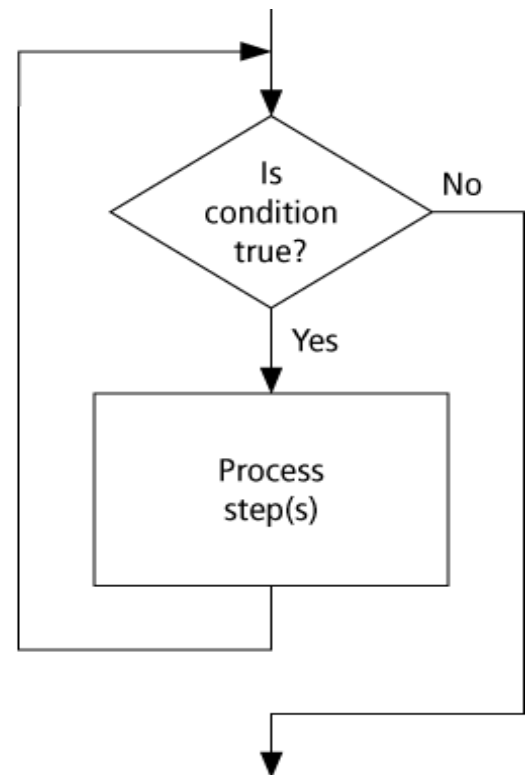
Symbol	Name	Meaning
	Flowline	Used to connect symbols and indicate the flow of logic.
	Terminal	Used to represent the beginning (Start) or the end (End) of a task.
	Input/Output	Used for input and output operations, such as reading and displaying. The data to be read or displayed are described inside.
	Processing	Used for arithmetic and data-manipulation operations. The instructions are listed inside the symbol.
	Decision	Used for any logic or comparison operations. Unlike the input/output and processing symbols, which have one entry and one exit flowline, the decision symbol has one entry and two exit paths. The path chosen depends on whether the answer to a question is "yes" or "no."
	Connector	Used to join different flowlines.
	Offpage Connector	Used to indicate that the flowchart continues to a second page.
	Predefined Process	Used to represent a group of statements that perform one processing task.
	Annotation	Used to provide additional information about another flowchart symbol.

Example to demonstrate some of the flowcharts are

1.



Do While condition is true
Process step(s)
Loop



2.

Pseudocode to calculate class average grade

: Determine the average grade of a class

Initialize Counter and Sum to 0

Do While there are more data

 Get the next Grade

 Add the Grade to the Sum

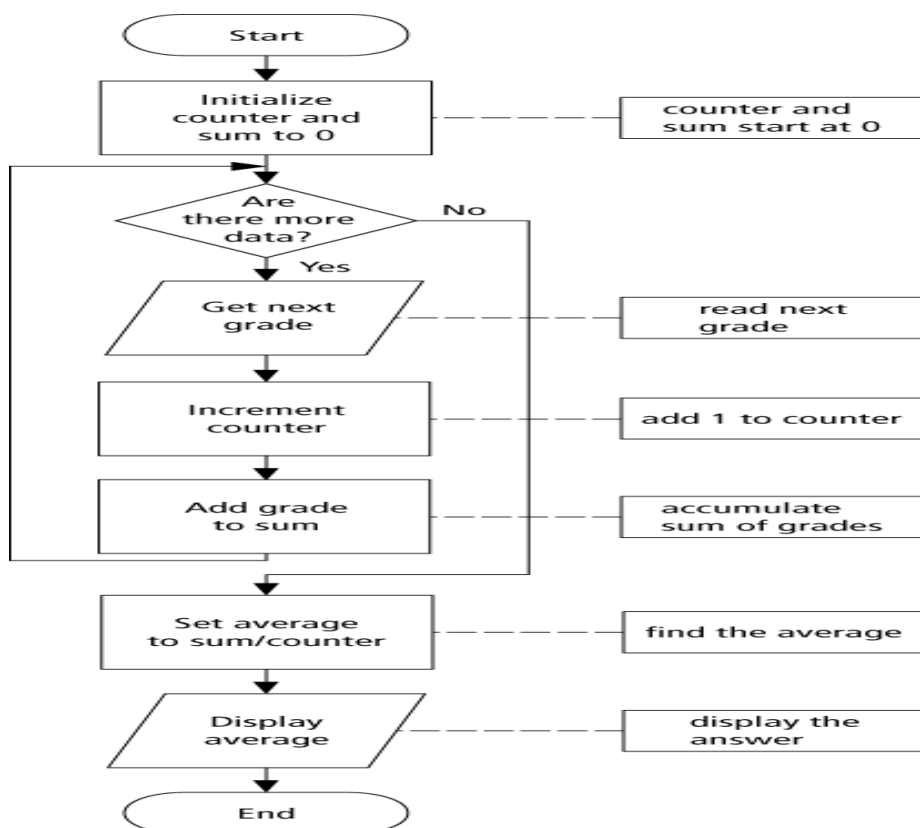
 Increment the Counter

Loop

Compute Average = Sum / Counter

Display Average

Flowchart representation



History of C Programming Language

- The C programming language was designed by Dennis Ritchie at Bell Laboratories in the early 1970s



- Influenced by following previous High level languages
 - ALGOL 60 (1960),
 - CPL (Cambridge, 1963),
 - BCPL (Martin Richard, 1967), (Basic Combined Programming Language)
 - B (Ken Thompson, 1970)

B language is modified by Ritchie and new Language is named as C

- C is a general-purpose language which has been closely associated with the UNIX OS for which it was developed - since the system and most of the programs are written in C.

C Standards

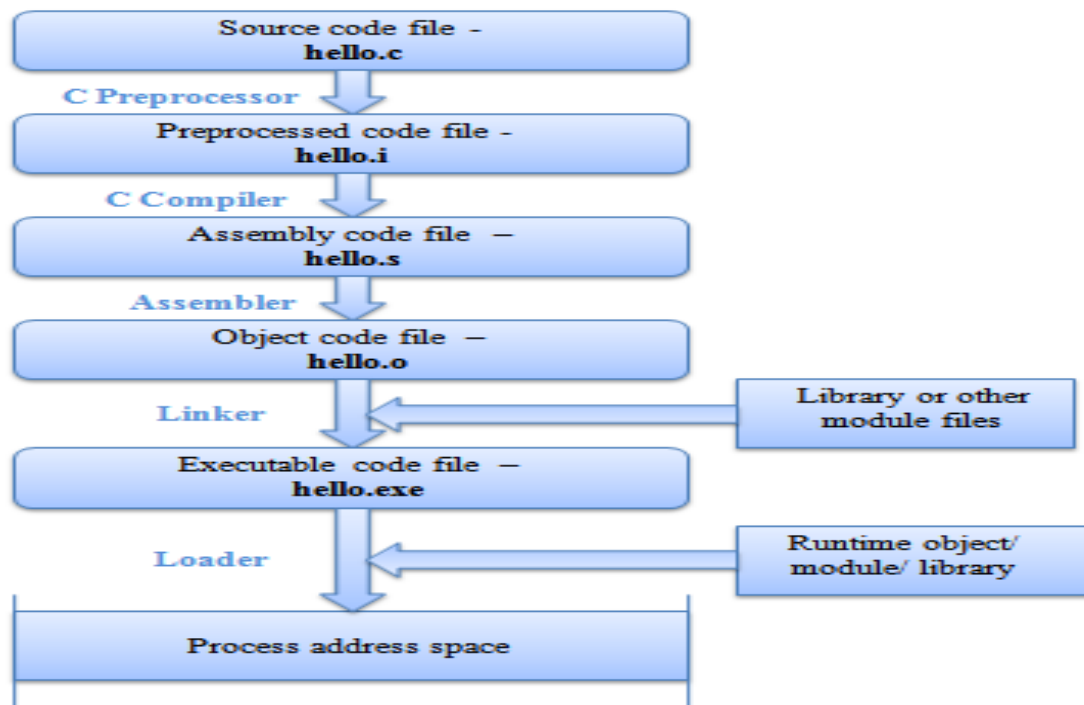
- **Standardized in 1989 by ANSI (American National Standards Institute) known as ANSI C**
- **International standard (ISO) in 1990 which was adopted by ANSI and is known as C89**
- **As part of the normal evolution process the standard was updated in 1995 (C95) and 1999 (C99)**

Characteristics of C Programming languages:

- ▶ **Direct access to memory layout through pointer manipulation**

- ▶ Concise syntax, small set of keywords
- ▶ Block structured language
- ▶ Some encapsulation of code, via functions
- ▶ Type checking (pretty weak)
- ▶ C is portable(program written for one computer can be run on another computer with little modification)
- ▶ C has an ability to extend itself.
- ▶ C was invented to write operating system called UNIX
- ▶ The language was formalized by American National Standard Institute (ANSI) 1988
- ▶ Unix is written using C
- ▶ C is widely used to develop system softwares
- ▶ C is a case sensitive program

Compilation Model:



Compilation process in UNIX/LINUX environment:

- ▶ To compile and link a C program that is contained entirely in one source file:

```
cc program.c
```

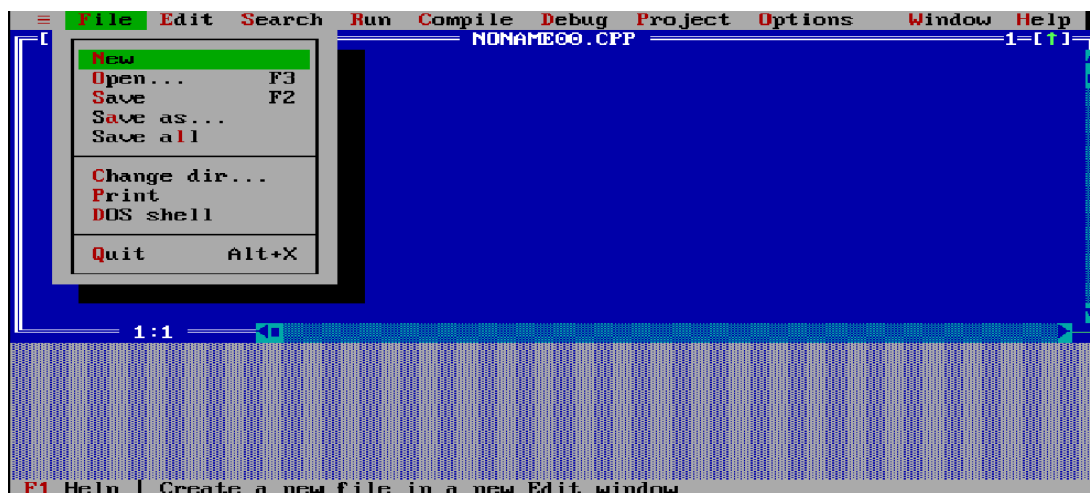
- ▶ The executable program is called a.out by default.

If you don't like this name, choose another using the `-o` option:

```
cc program.c -o exciting_executable
```

Compilation process in Turbo C environment:

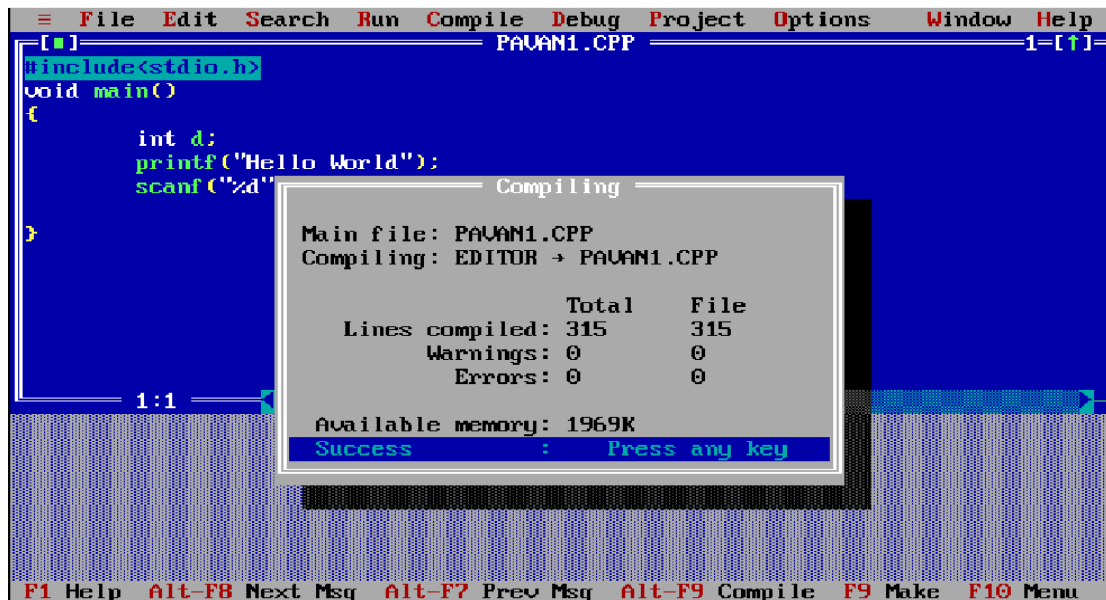
To type c Program: click File -> open -> noname.cpp is created (cpp stands for c plus plus)



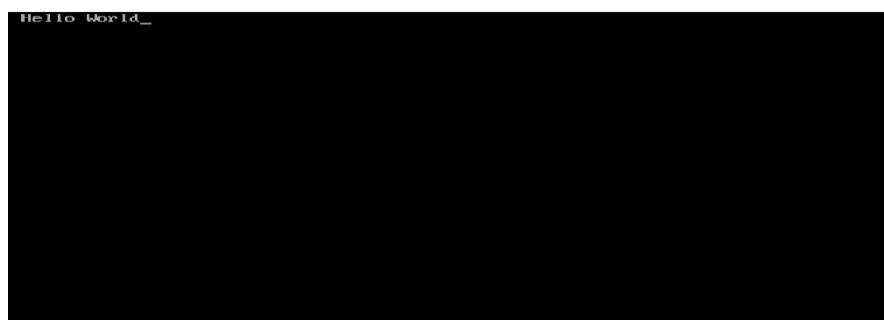
Type the program in the editor



Run the Program by pressing (Ctrl+F9)



Output of the program will be displayed as shown below:



General structure of C program

Pre-processor directives(#include..)

```
int main()
{
    Declaration statements;
    executable statements;
    return 0;
}
```

Example Programs:

1.

```
#include <stdio.h>

// program prints hello C world

Int main() {

    printf ("Hello C world!");

    return 0;

}
```

Output: **Hello C world!**

Header files:

- The files that are specified in the include section is called as header file
- These are precompiled files that has some functions defined in them
- We can call those functions in our program by supplying parameters
- Header file is given an extension .h
- C Source file is given an extension .c

Example : # include <stdio.h>

main ()

- This is the entry point of a program
- When a file is executed, the start point is the main function
- From main function the flow goes as per the programmers instructions.
- There may or may not be other functions written by user in a program
- Main function is compulsory for any c program

Comment lines in C Program:

- Single line comment (line starts with two slashes //)
 - // (double slash)
 - Termination of comment is by pressing enter key
- Multi line comment (starts with /* and ends with */)

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

This can span over to multiple lines

Output Statement printf()

printf() statement is used to output the expression values and messages on to the output screen. General structure of printf() is

```
printf(" message placeholder", variables);
```

```
printf(" value of c = %d",c);
```

- printf() is a library function declared in <stdio.h>
- Syntax: printf(*FormatString*, *Expr*, *Expr*...)
 - *FormatString*: String of text to print
 - *Exprs*: Values to print
 - *FormatString* has placeholders to show where to put the values (note: #placeholders should match #*Exprs*)
 - Placeholders: %s (print as string), %c (print as char), %d (print as integer), %f (print as floating-point)
 - \n indicates a newline character

Example: printf("Original input : %s\n", input);

Example program:

```
#include <stdio.h>  
// program prints a number of type int  
int main() {  
    int number = 10;  
    printf ("The Number is %d", number);  
    return 0;  
}
```

Output: The Number is 10

Input statement scanf()

scanf (); //used to take input from console(user).

- ❑ scanf(“%d”, &a);
- ❑ Remember to use **&** symbol along with integer/floating variable.
Does not use at the time of reading character/string type data.

Some of the format specifier

- %c for The character.
- %d for The integer format specifier.
- %f for The floating-point format specifier.
- %s for The string format specifier.

Example:

- **Input**

```
scanf(“%d”,&a);
```

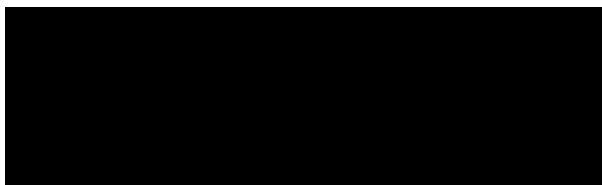
Gets an integer value from the user and stores it under the name “a”

- **Output**

```
printf(“%d”,a);
```

Prints the value present in variable a on the screen

```
1  /* Fig. 2.5: fig02_05.c
2     Addition program */
3  #include <stdio.h>
4
5  int main()
6  {
7     int integer1, integer2, sum;           /* declaration */
8
9     printf( "Enter first integer\n" );    /* prompt */
10    scanf( "%d", &integer1 );             /* read an integer */
11    printf( "Enter second integer\n" );   /* prompt */
12    scanf( "%d", &integer2 );             /* read an integer */
13    sum = integer1 + integer2;             /* assignment of sum */
14    printf( "Sum is %d\n", sum );         /* print sum */
15
16    return 0; /* indicate that program ended successfully */
17 }
```



Variables:

Variables are data that will keep on changing

Declaration

```
<<Data type>> <<variable name>>;
```

```
int a;
```

Initialization

```
<<varname>>=<<value>>;
```

```
a=10;
```

Usage (updatation)

```
<<varname>>
```

```
a=a+1; //increments the value of a by 1
```

Rules to declare a variable:

- Should not be a reserved words like int, float, sin etc..
- Should start with a letter or an underscore(_)
- Can contain letters, numbers or underscore.
 - Choose a name that reflects the role of the variable in a program, e.g.
 - Good: customer_name, ss_number;
 - Bad : cn, ss;
- No other special characters are allowed including space
- Variable names are case sensitive
 - say variable **A (Big letter)** and **a (small letter)** are different

Some properties of a variable:

- Represent storage units in a program
- Used to store/retrieve data over life of program
- Type of variable determines what can be placed in the storage unit
- *Assignment* – process of placing a particular value in a variable
- Variables must be *declared* before they are assigned
- The value of a *variable* can change; A *constant* always has the same value

Tokens:

- Token is a sequence of one or more characters that is significant as a group
- Six types of tokens are: keywords, identifiers(variables), constants, string literals, operators and other separators

keywords- reserved words for **example** `sqrt,pow,sin,int`

identifiers- names given to variables **example:** `a,sum`

Constants: integer, floating and character **example:** `10,10.0`

Operators: arithmetic, relational, logical **example:** `<,==, !=,&&,||`

Separators: white spaces, comments **example:** `//...`

Data type size and its range

- Primary : int, float, char
 - int (signed/unsigned)(2,4Bytes): used to store integers.
 - char (signed/unsigned)(1Byte): used to store characters
 - float, double(4,8Bytes): used to store a decimal number.
- User Defined:
 - typedef: used to rename a data type
 - typedef int *integer*; can use *integer* to declare an int.
 - enum, struct, union

TYPE OF DATA	DATA TYPE	SIZE (IN BYTES) 16 BIT COMPUTER	RANGE
Character	char	1	Signed -128 to 127 Unsigned: 0 to 255
integer	int	2	Signed: -32768 to 32767 Unsigned: 0 to 65535
Real number	float	4	
Double precision	double	8	

Constants

- Constant is a quantity whose value cannot be changed during program execution.

- C supports four type of constants

integer, floating, character and enumeration constants

Integer Constant

- Represents a signed integer of typically 2 or 4 or 8 bytes (16 or 32 or 64 bits)
- Precise size is machine-dependent
- It is a number that has an integer value.
- It can be specified in decimal, octal or hexadecimal form

example: 5, 125,

Floating Constant

- Floating constant have mantissa and exponent part(optional includes the letter e or E)

ddd.dddE(+/-)dd

- Mantissa(significant part) contains digit followed by decimal point(.)and then digit

453.678

Character Constant

- It is grouped into two categories: integer character, wide character

integer character is a sequence of one or more character enclosed in single quotes
'a'

- The character within the single quote may be any character (except backslash or newline or single quote)

example : 'd' 'f'

Some points to be remembered about C Programs are

- all *statements* end with a semicolon!
- Commas separate multiple declarations
- Blank lines have no effect
- Extra spaces between *tokens* has no effect.
- *Comments* are ignored by the compiler

Invisible Characters in C language

- Some special characters are not visible directly in the output stream. These all begin with an escape character (i.e. \);
 - \n newline
 - \t horizontal tab
 - \a alert bell
 - \v vertical tab

Operators in C Programming languages

Some of the operators used in c language are

- ✓ Arithmetic operators
- ✓ Conditional operators
- ✓ Bitwise operators
- ✓ Relational operators
- ✓ Logical operators
- ✓ Assignment operators
- ✓ Increment and decrement operators
- ✓ Special operators

1. Arithmetic Operator: arithmetic operations are performed on integer or floating point data i.e.

Integer Arithmetic

Operands in a single arithmetic expression
Operation is integer arithmetic

E.g. If a =10 and b=4
 a-b=6
 a+b=14
 a/b=2
 a%b=2

Real Arithmetic :

Real operator is known as real arithmetic.
Decimal and exponential notation
If x,y are floats

x=6.0/7.0=0.857143
y=-2.0/3.0=-0.666667

Mixed –mode Arithmetic

- ✓ One of the operands is real and the other is integer

For example

$$19/10.0=1.9$$

Assignment Operator:

In addition, C has a set of shorthand assignment operators of the form.
var oper = exp;

Example

x = a + b

Simple Programs

1. Write a program to calculate area of triangle

```
#include<stdio.h>

int main()
{ int len,bre,area;
  printf(" enter length and breadth of triangle");
  scanf("%d%d",&len,&bre);
  area=0.5*len*bre;
  printf("\narea of triangle=%d",area);
  return 0;
}
```

OUT PUT

enter length and breadth of triangle

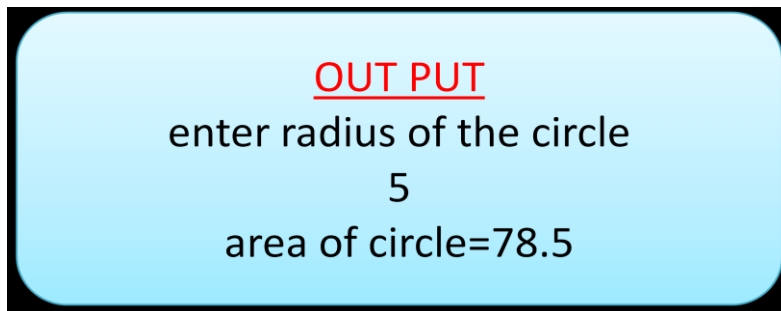
4

5

area of triangle=10

2. Write a program to calculate area of circle

```
#include<stdio.h>
#define pi 3.14
int main()
{
    int r;
    float area;
    printf(" enter radius of the circle\n");
    scanf("%d",&r);
    area= pi * r * r;
    printf("\narea of circle = %f", area);
    return 0;
}
```



OUT PUT
enter radius of the circle
5
area of circle=78.5

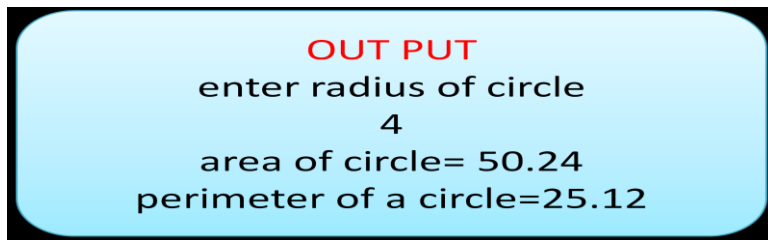
3. Write a program to calculate area and perimeter of a circle

```
#include<stdio.h>
#define pi 3.14
int main()
{
    int r;
    float area,peri;
    printf(" enter radius of the circle);
    scanf("%d",&r);
    area= pi * r * r;
```

```

peri = 2*pi*r;
printf(" area of circle = %f\n", area);
printf(" perimeter of a circle=%f",peri);
return 0;
}

```



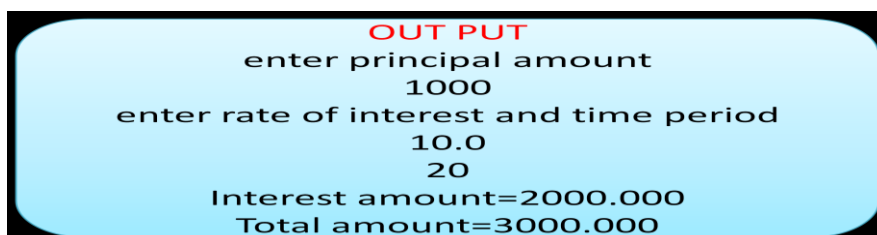
OUT PUT
enter radius of circle
4
area of circle= 50.24
perimeter of a circle=25.12

4. Write a program to calculate simple interest

```

#include<stdio.h>
int main()
{ int p;
float r,t,interest,total;
printf(" enter principal amount ");
scanf("%d",&p);
printf(" enter rate of interest and time period");
scanf("%f%f",&r,&t);
interest=(p*r*t)/100.0;
total=p+interest;
printf("interest amount=%f\n total amount=%f",interest,total);
return 0;
}

```



OUT PUT
enter principal amount
1000
enter rate of interest and time period
10.0
20
Interest amount=2000.000
Total amount=3000.000

Mathematical Library functions:

In order to calculate mathematical values of $\sin()$, $\cos()$, $\exp()$ etc. C language provides built-in library functions.

- Library facilities are usually provided in the languages to keep the language simple and to do many mathematical calculations.
- Some of the library header files available in C are

input/output facility <stdio.h> (scanf,printf,gets,puts....)

mathematical operations <math.h>(sqrt,sin,cos,log,pow...)

string manipulation oper <string.h>(strcpy, strcat, strlen..)

console input/output <conio.h> (getch,...)

- These header should be included in the program using

#include preprocessor directive statement

ex: **#include<math.h>**

- This header file gives many built in mathematical library functions such as trigonometric, logarithmic, exponential etc..

Function Name	C function call	meaning
sqrt	sqrt(x)	Square root of x
power	pow(x,y)	X rise to Y
exponential	exp(x)	
log10	log10(x)	
log	log(x)	Natural logarithm of x
sin	sin(x)	Sine of (x)
cos	cos(x)	Cosine of (x)
tan	tan(x)	Tangent of (x)
absolute	abs()	a

```

#include<stdio.h>
#include<math.h>
Void main()
{
    float x y;
    float sqrt, sinx; cosx,tanx;
    x=144.0;
    sqrt=sqrt(x);
    y=144*(3.142/180.0);
    sinx=sin(y);
    cosx=cos(y);
    tanx=tan(y);
    printf("square root of x=%f \n",sqrt);
    printf("sine value of y=%f \n",sinx);
    printf("cosine value of y=%f \n",cosx);
    printf("tangent value of y=%f \n",tanx);
    getch();
}

```

When you run the program you get the following output.

displays the computation of square, cosine, sine and tangent value of a number by using mathematical function in c.

```

Square root of x=12.000000
Sine value of y= xxxxx
Cosine value of y= xxxxx
Tangent value of y= xxxxx

```

1. Program to calculate area of triangle when sides of triangle is given

```

#include<stdio.h>
int main()
{ int a,b,c;
    float area,s;
    printf(" enter three sides length of a triangle");
    scanf("%d%d%d",&a,&b,&c);
    s=(a+b+c)/2.0
    area=sqrt(s * (s-a)* (s-b) * (s-c));
    printf(" area of triangle=%f",area);
    return 0;
}

```

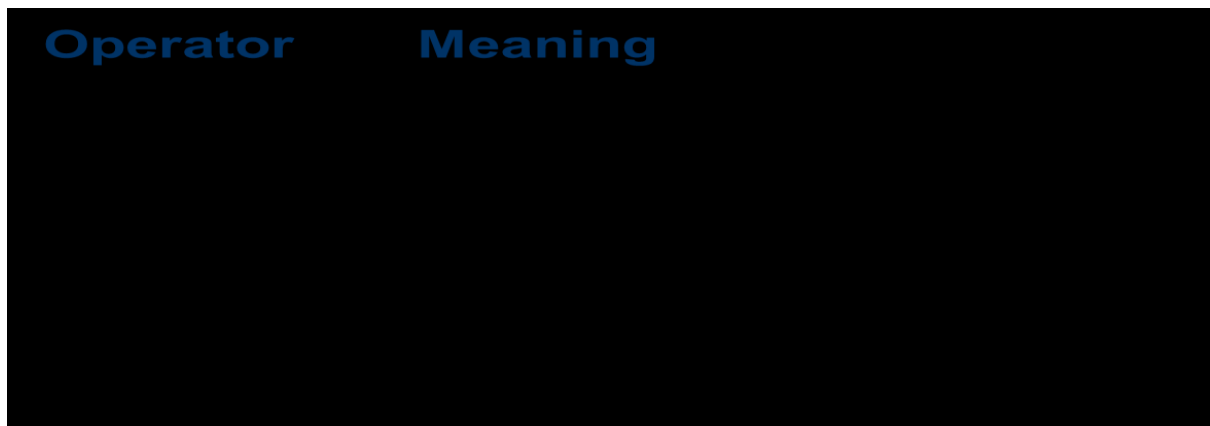
OUT PUT

```

enter three sides length of a triangle
3 4 5
area of triangle=6

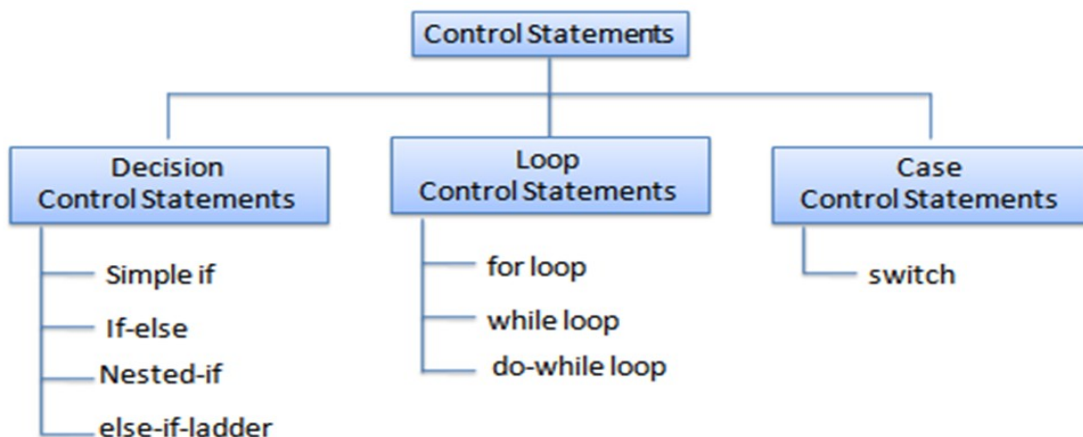
```

Relational Operator: In order to find the relation between any two values relational operator are used for example a is greater than b, result may be true or false depends on the value of a and b, some of the relational operators are:



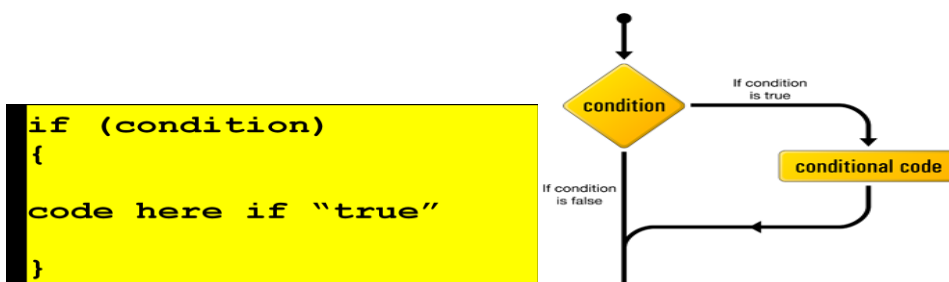
Control Statements :

- The order in which the statements are executed are called control flow
- The statements that are used to control the flow of execution of program are called control statements
- C Language supports following control statements



If Statement:

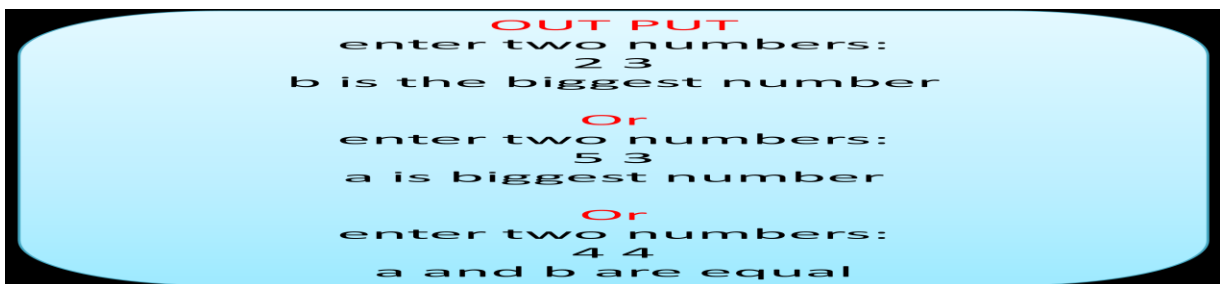
- Branching is the process of choosing the right branch for execution, depending on the result of “conditional statement”.



<code>if (condition)</code>	<code>if (condition)</code>
<code>statement;</code>	<code>{ statements; }</code>
<code>next_statement;</code>	<code>next_statement;</code>

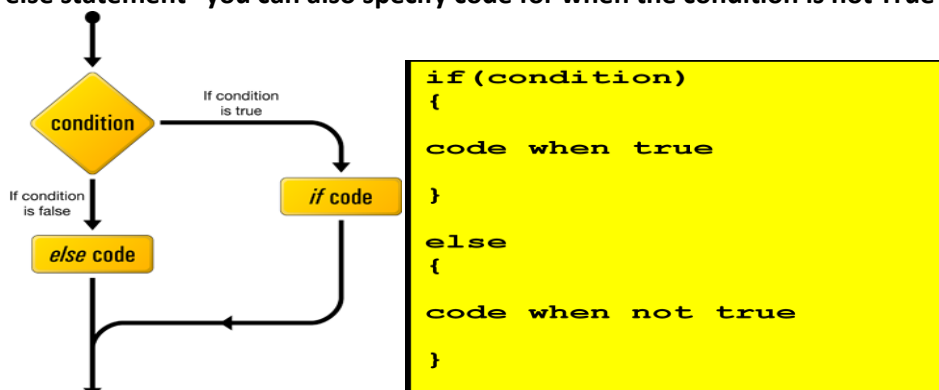
1. Write a Program to find biggest among two numbers

```
#include <stdio.h>
int main()
{
    int a, b;
    printf( " enter two numbers:");
    scanf("%d%d",&a,&b);
    if (a>b) printf(" a is the biggest number");
    if (b>a) printf(" b is the biggest number");
    if (a==b) printf (" a and b are equal");
    return 0;
}
```



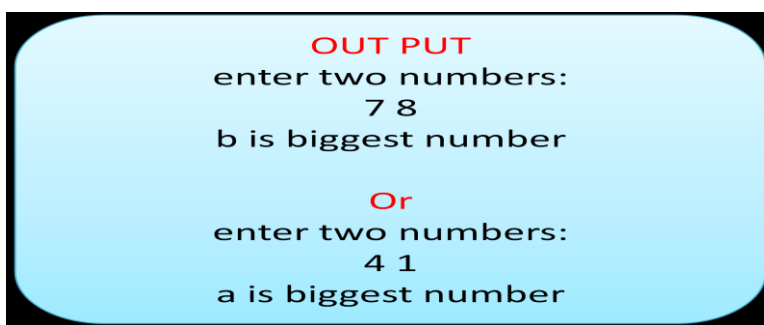
2. If-else statement:

Unlike "if statement" where you could only specify code for when condition is true; for "if else statement" you can also specify code for when the condition is not True (false)



Program to find biggest among two numbers using (if-else)

```
#include <stdio.h>
int main()
{ int a, b;
  printf( " enter two numbers:");
  scanf("%d%d",&a,&b);
  if (a>b) printf(" a is the biggest number");
  else printf(" b is the biggest number");
  return 0;
}
```



OUT PUT

enter two numbers:
7 8
b is biggest number

Or

enter two numbers:
4 1
a is biggest number

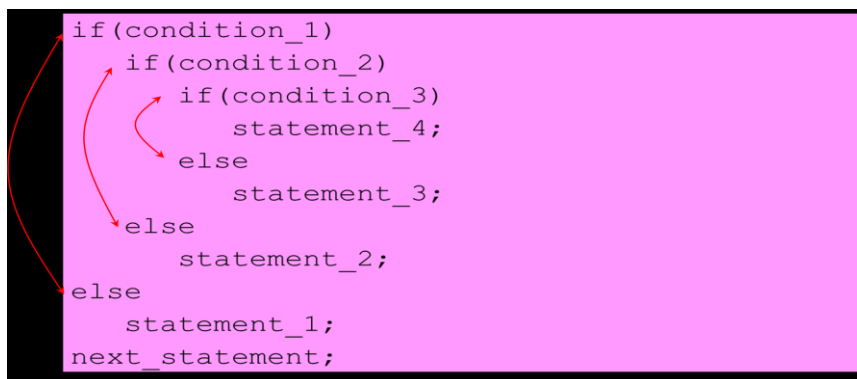
3. Nested- if statement:

- Using "if...else statement" within another "if...else statement" is called 'nested if statement'.

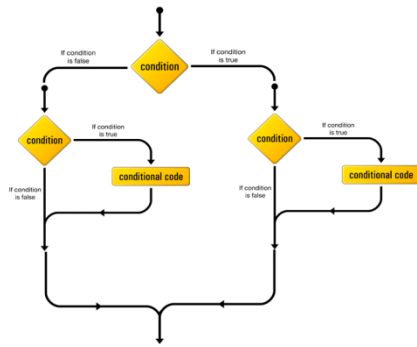
"Nested if statements" is mainly used to test multiple conditions

- The if-else constructs can be nested (placed one within another) to any depth.
- General forms: if-if-else and if-else-if.

The if-if-else constructs has the following form (3 level of depth example



```
if (condition_1)
  if (condition_2)
    if (condition_3)
      statement_4;
    else
      statement_3;
  else
    statement_2;
else
  statement_1;
next_statement;
```



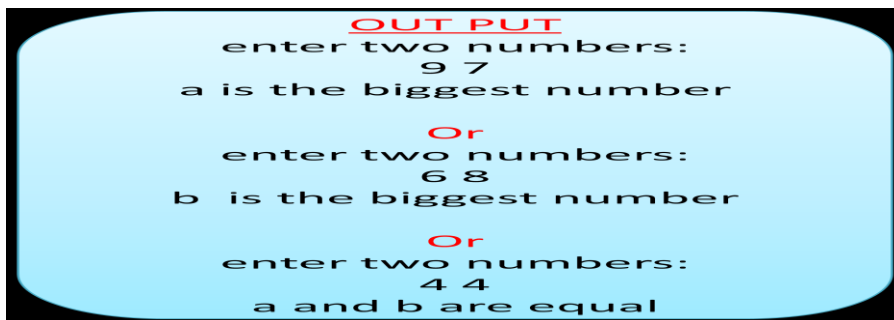
```

if(condition)
{
    if (condition)
    {
        code when true
    }
}
else
{
    if (condition)
    {
        code when true
    }
}
  
```

Program to find biggest among two numbers using nested-if statement:

```

#include <stdio.h>
int main()
{ int a, b;
  printf( " enter two numbers:");
  scanf("%d%d",&a,&b);
  if (a>b) printf(" a is the biggest number");
  else if (b>a) printf(" b is the biggest number");
  else printf(" a and b are equal");
  return 0;
}
  
```



4. Logical Operator

Operator	Meaning
&&	Logical AND
	Logical OR
!	Logical NOT

Inputs		and	or
a	b	a & b	a b
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

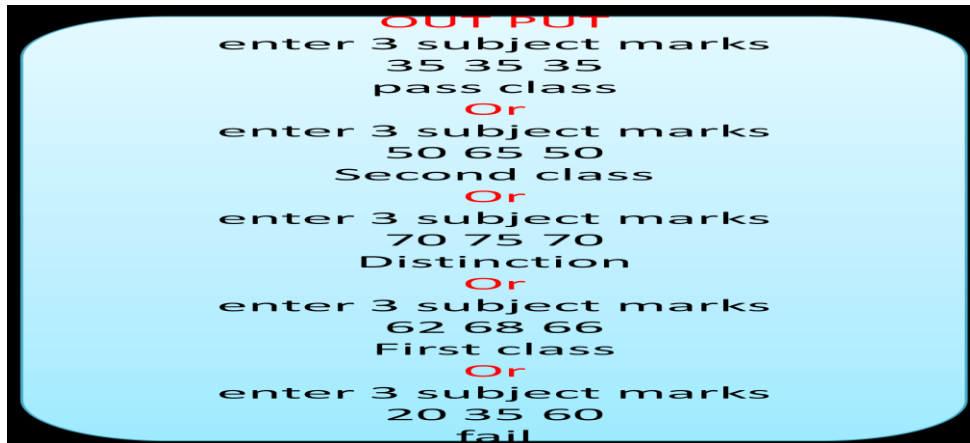
1. Program to find quadrant for the given co-ordinates

```
#include<stdio.h>
int main();
{
  int x,y;
  printf("Enter the coordinates for quadrants:");
  scanf("%d%d",&x,&y);
  if(x>0 && y>0) printf("the co-ordinate lies on 1st quadrant");
  else if(x<0 && y>0) printf("the co-ordinate lies in 2nd quadrant");
  else if(x<0 && y<0) printf("the co-ordinate lies in 3rd quadrant");
  else if(x>0 && y<0) printf("the co-ordinate lies in 4th quadrant");
  else if(x==0 && y==0) printf("the co-ordinate is on origin");
  else if(x==0) printf("the co-ordinate lies in y-axis");
  else printf("the co-ordinate lies in x-axis");
}
```

Input co-ordinates	OUT PUT
25,78	the co-ordinate lies on 1 st quadrant
-105,40	the co-ordinate lies on 2 nd quadrant
-105,-150	the co-ordinate lies on 3 rd quadrant
85,-125	the co-ordinate lies on 4 th quadrant
0,0	the co-ordinate lies in origin
0,25	the co-ordinate lies in y-axis
30,0	the co-ordinate lies in x-axis

2. Program to declare student result

```
int main()
{
    int s1,s2,s3;
    float avg;
    printf( "enter 3 subject marks");
    scanf("%d%d%d",&s1,&s2,&s3);
    avg=(s1+s2+s3)/3.0;
    if ( (s1<35) || (s2<35) || (s3<35)) printf(" fail");
    else if (avg>=70) printf(" Distinction");
    else if ((avg>=60) && (avg < 70)) printf(" First class");
    else if ((avg>=50) &&(avg < 60)) printf(" Second class");
    else printf (" pass class");
    return 0;
}
```



OUTPUT

```
enter 3 subject marks
35 35 35
pass class
Or
enter 3 subject marks
50 65 50
Second class
Or
enter 3 subject marks
70 75 70
Distinction
Or
enter 3 subject marks
62 68 66
First class
Or
enter 3 subject marks
20 35 60
fail
```

1. Program to calculate Roots of Quadratic equation (Lab program)

- Quadratic equation of the form $ax^2+bx+c=0$ is having two roots. Root is a value of x when it is substituted to the above equation it satisfies (i.e. LHS=RHS ($F(X)=y=0$))
- Root is a point on the x -axis where $y=0$. In quadratic equation it cuts x -axis at two points. (i.e. two roots)
- Root value depends on the discriminant value (b^2-4ac)
- If discriminant is zero -- roots are equal (i.e. $r_1=r_2$)
- If discriminant is greater than zero – roots are distinct
- If discriminant is less than zero – roots are imaginary

```

#include<stdio.h>
#include<math.h>
int main()
{ float a,b,c;
  float d,r1,r2,re,ri;
  printf("enter the co-efficients");
  scanf("%f%f%f",&a,&b,&c);
  d=b*b-4*a*c;
  if (d==0){
    r1=-b/(2*a);
    printf("roots are real and equal")
    printf("root1=%f\t root2=%f",r1,r1);
  }
  else if (d>0){
    r1=-b+sqrt(d)/(2*a);
    r2=-b-sqrt(d)/(2*a);
    printf("roots are distinct\n");
    printf("root1=%f\t root2=%f",r1,r2);
  }
  else {
    re=-b/(2*a);
    ri=sqrt(abs(d))/(2*a);
    printf("roots are complex conjugates\n");
    printf("real part=%f\t img part=%f",re,ri);
  }
  return 0;
}

```

```

Output 1: enter the co-efficients
          1 2 1
          roots are real and equal
          root = -1.00

Output 2: enter the co-efficients
          1 3 2
          roots are distinct
          root1= -1.00  root2= -2.00

Output 3: enter the co-efficients
          1 2 3
          roots are complex conjugates
          real part = -1.00  img part= 1.41

```

2. Program to check whether the given year is Leap year or not (Lab Program)

A year is leap if it is divisible by 4 but not by 100 or is divisible by 400

ex: 1996,2000,2004 -- leap year

1900,2002,2100 -- not a leap year

Algorithm leap(year)

read(year)

if(year is divisible by 4 and year is not divisible by 100) or is divisible by 400

print " given year is leap year"


else

print " given year is not a leap year"

End of the algorithm

((year%4==0) &&(year%100 != 0) || year%400 == 0)

- #include<stdio.h>
int main()
{
int year;
printf("enter the year");
scanf("%d",&year);
if(((year%4==0) &&(year%100 != 0)) || year%400 == 0)
printf("given year %d is leap year",year);
else printf ("given year %d is not a leap year",year);
return 0;
}



```
OUT PUT
enter the year
2010
given year 2010 is leap year
Or
enter the year
2013
given year 2013 is not a leap year
```

Arithmetic Operators:

- **Prefix Increment : ++a**
 - example:
 - » int a=5;
 - » b=++a; // value of b=6; a=6;
 - » **Postfix Increment: a++**
 - example

- » `int a=5;`
 - » `b=a++;` //value of `b=5; a=6;`
- **Modulus (remainder): %**
 - **example:**
 - » `12%5 = 2;`
 - » **Assignment by addition: +=**
 - **example:**
 - » `int a=4;`
 - » `a+=1;` //(means `a=a+1`) value of `a` becomes 5

We Can use `-`, `/`, `*`, `%` also

- **Comparison Operators: `<`, `>`, `<=`, `>=`, `!=`, `==`, `!`, `&&`, `||`.**
 - **example:**
 - » `int a=4, b=5;`
 - » `a<b` returns a true(non zero number) value.
 - » **Bitwise Operators: `<<`, `>>`, `~`, `&`, `|`, `^`.**
 - **example**
 - » `int a=8;`
 - » `a = a>>1;` // value of `a` becomes 4
- **Meaning of `a + b * c` ?**
is it `a+(b*c)` or `(a+b)*c` ?
- **All operators have precedence over each other**
- **`*`, `/` have more precedence over `+`, `-`.**
 - If both `*`, `/` are used, associativity comes into picture. (more on this later)
 - **example :**
 - » `5+4*3 = 5+12= 17.`

- **Precedence rules decides the order in which different operator are applied.**
- **Associativity rule decides the order in which multiple occurrences of the same level operator are applied**
-

Precedence Table:

Highest on top		
++	--	(Postfix)
++	--	(Prefix)
*	/	%

+	-
<<	>>
<	>
&	
&&	

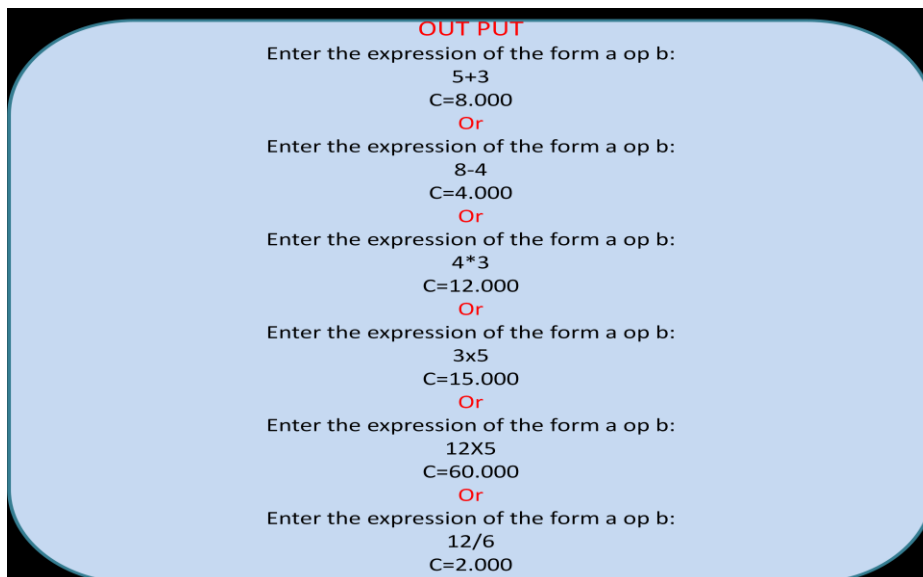
Switch Statement: switch statement begins with switch keyword. It is the combination of multiple cases separated by break statement. Only one case will be executed at any time and break statement takes the control outside the switch statement. If there is no match for the case statement default statement will be executed.

```
switch(condition)
{
    case template_1 : statement(s);
                    break;
    case template_2 : statement(s);
                    break;
    case template_3 : statement(s);
                    break;
    ...
    ...
    case template_n : statement(s);
                    break;

    default : statement(s);
}
next_statement;
```

1. Program to Simulate simple calculator

```
#include<stdio.h>
int main()
{
    float a,b,c;
    char op;
    printf("Enter the expression in the form of a op b: ");
    scanf("%f %c %f",&a,&op,&b);
    switch(op)
    {
        case '+': c=a+b;
                break;
        case '-': c=a-b;
                break;
        case '*':
        case 'x': c=a*b;
                break;
        case '/': c=a/b;
                break;
    }
    printf("c=%f\n",c);
    return 0;
}
```



OUT PUT

Enter the expression of the form a op b:
5+3
C=8.000
Or
Enter the expression of the form a op b:
8-4
C=4.000
Or
Enter the expression of the form a op b:
4*3
C=12.000
Or
Enter the expression of the form a op b:
3x5
C=15.000
Or
Enter the expression of the form a op b:
12X5
C=60.000
Or
Enter the expression of the form a op b:
12/6
C=2.000

2. Program to calculate area of geometric objects as per user request

```
#include<stdio.h>
int main()
{
    float a,b,area;
    int choice;
    printf("Enter the choice(1:square,2:rectangle:3:circle ");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1 : printf("enter side length");
                scanf("%f",&a);
                area=a*a;
                break;
        case 2 : printf("enter length and
                breadth");
                scanf("%f%f",&a,&b);
                area=a*b;
        case 3 : printf("enter the radius")
                scanf("%f",&a)
                area=3.14*a*a;
                break;
        default: printf("in valid choice");
                break;
    }
    printf(" area=%f\n",area);
    return 0;
}
```

While LOOP : It executes the statement with in the loop until the condition is satisfied.

- Executes a block of statements as long as a specified condition is TRUE.
- The general while loop construct,

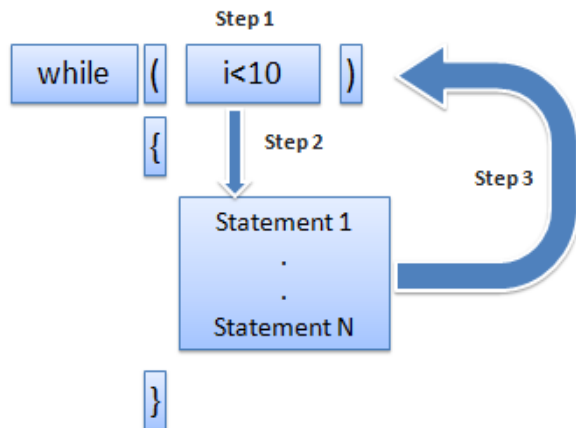
```
while (condition)
    statement(s);
next_statement;
```

- The (condition) may be any valid C expression.
- The statement(s) may be either a single or a compound (a block of code) C statement.
- When while statement encountered, the following events occur:
 1. The (condition) is evaluated.
 2. If (condition) evaluates to FALSE (zero), the while loop terminates and execution passes to the next_statement.
 3. If (condition) evaluates as TRUE (non zero), the C statement(s) is executed.
 4. Then, the execution returns to step number 1 until condition becomes FALSE.

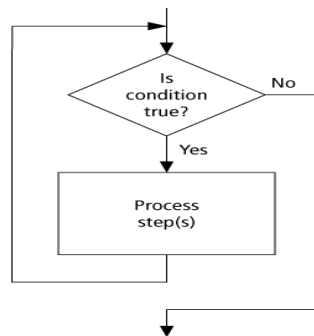
```

initialization
while (condition)
{
    //body
    iteration
}

```



Do While condition is true
Process step(s)
Loop



```

simple while loop example
#include <stdio.h>
int main(void)
{
    int n = 1;
    // set the while condition
    while(n <= 12)
    {
        // print
        printf("%d ", n);
    }
}

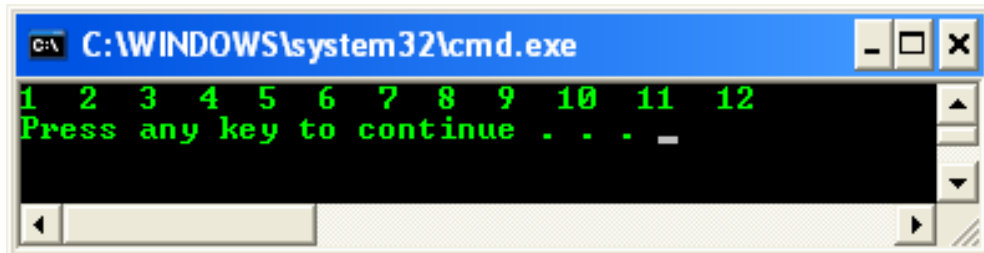
```

```

        // increment by 1, repeats
        n++;
    }
    // a newline
    printf("\n");
    return 0;
}

```

Output:



```

C:\WINDOWS\system32\cmd.exe
1 2 3 4 5 6 7 8 9 10 11 12
Press any key to continue . . . _

```

```

// simple while loop example
#include <stdio.h>
int main(void)
{
    int n = 1,sum=0;
    // set the while condition
    while(n <= 50)
    {
        sum=sum+n;
        // increment n by 1, repeats
        n++;
    }
    // a newline
    printf("sum upto 1 to 50 is =%d",sum);
    return 0;
}

```

while – loop example

To sum= $2^2+4^2+\dots +50^2$

simple while loop example

```

#include <stdio.h>
int main(void)
{
    int n = 2;
    float sum=0.0;
    // set the while condition
    while(n <= 50)
    {
        sum=sum+pow(n,2);
        // increment n by 1, repeats
        n=n+2;
    }
}

```

```
}
```

```
printf("sum upto 2^2 to 50^2 is =%d",sum);  
return 0;
```

3. GCD using Euclidian's algorithm

- Let m and n represent two numbers and variable r represent remainder of the division

```
    r=m%n  
Algorithm GCD(m,n)  
read (m,n)  
while(n>0)  
    { r= m % n;  
      m = n;  
      n= r;  
    }  
print("gcd of m and n is", m)  
#include<stdio.h>  
int main()  
{  
    int m,n,r;  
    printf("enter two integer number");  
    scanf("%d%d",&m,&n);  
    while(n > 0)  
    { r=m%n;  
      m=n;  
      n=r;  
    }  
    printf("GCD of m and n is :%d", m);  
    return 0;  
}
```

step	m	n	m%n
1	50	35	15
2	35	15	5
3	15	5	0
4	5	0(s)	

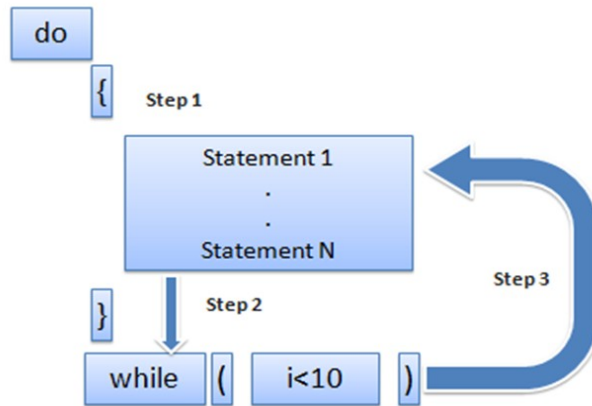
```
OUT PUT
enter two integer number
50 35
GCD of m n is :5
```

Do-While Loop: This statement is used when we want to execute the body of the loop at least once.

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

- Executes a block of statements as long as a specified condition is true at least once.
- Test the condition at the end of the loop rather than at the beginning, as demonstrated by the for and while loops.
- (condition) can be any valid C expression.
- When the program encounter the do-while loop, the following events occur:
 - The statement(s) are executed.
 - The (condition) is evaluated. If it is TRUE, execution returns to step number 1. If it is FALSE, the loop terminates and the next_statement is executed.
 - This means the statement(s) in the do-while will be executed at least once.

```
initialization
do
{
    //body
    iteration
}while(condition) ;
```



```
#include <stdio.h>
main()
{
    int i = 10;
    do{
        printf("Hello %d\n", i);
        i = i -1;
    }while ( i > 0 );
}
```

Output:

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

1. Program to check the given number is palindrome or not

If the given number and its reverse is also same then the given number is called palindrome ex: 1221, 343


```

#include<stdio.h>
int main()
{
    int num,rev=0,temp;
    printf("Enter a number: ");
    scanf("%d",&num);
    temp=num;
    while(num!=0){
        digit=num%10;
        num=num/10;
        rev=rev*10+digit;
    }
    if(rev==temp)
        printf("%d is palindrome",temp);
    else
        printf("%d is not a palindrome",temp);
    return 0;
}

```

num	digit	rev
3467		0
3467	7	7
346	6	76
34	4	764
3	3	7643
0		

1. Check the given number is Armstrong number or not using C program

Definition of Armstrong number:

Those Numbers Which **Sum Of The Cube Of Its Digits Is Equal To That Number Are Known As Armstrong Numbers.**

For Example 153 Since

$$1^3 + 5^3 + 3^3 = 1 + 125 + 9 = 153$$

```
#include<stdio.h>
int main()
{
    int num,r,sum=0,temp;
    printf("Enter a number: ");
    scanf("%d",&num);
    temp=num;
    while(num!=0){
        r=num%10;
        num=num/10;
        sum=sum+(r*r*r);
    }
    if(sum==temp)
        printf("%d is an Armstrong number",temp);
    else
        printf("%d is not an Armstrong number",temp);
    return 0;
}
```

For statement:

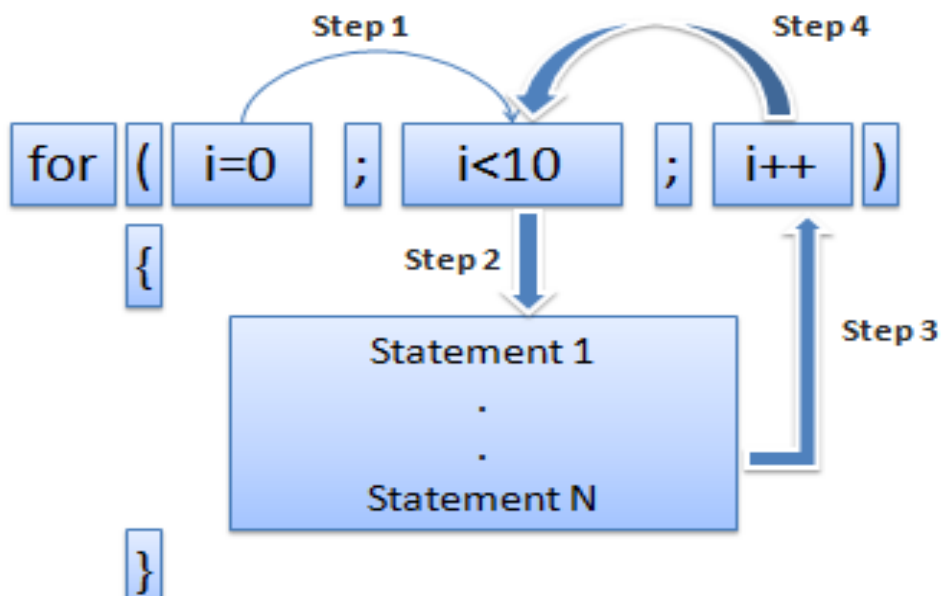
- for loop is used for repetitive execution of a statement or group of statements.

```
for(initial_expr; final_expr; update_expr)
{
    statements;
}
```

example:

```
for(i=1; i<=10; i++)
{
    sum= sum+i;
}
```

```
for initialization      condition      iteration
//body
```



- Initial_expr is usually an assignment expression.
- Update_expr is an increment/decrement expression
- Final_expr is the relational expression results in T or F

Print the numbers in the given range say m to n

```
scanf("%d%d",&m,&n);
for (i=m;i<=n;i++)
    printf("%d \t",i);
• for( x=0.1;x<=1.0;x+=0.2)
printf("%f\t %f\n",x,x*x);
```

0.1 0.01
0.3 0.09
0.5 0.25
0.7 0.49
0.9 0.81

1. Program to find factorial of a given number

Algorithm fact

```
read n
fact=1
for i=n down to 2
    fact= fact*i
print (fact)
end
```

```
#include<stdio.h>
Int main()
{ int n,i,fact=1
  printf("enter the positive number:");
  scanf("%d",&n)
  for(i=n;i<=2;i--)
    fact=fact*i;
  printf ("factorial of %d is %d",n,fact);
  getch();
}
```

Output:

```
enter the positive number : 3
factorial of 3 is 6
```

- for loop is a very flexible construct.
- Can use the decrementing counter instead of incrementing. For example,
for (nCount = 100; nCount > 0; nCount--)
- Can use counter other than 1, for example 3,
for(nCount = 0; nCount < 1000; nCount += 3)
- initial_value can be omitted if the test variable has been initialized beforehand.
- However the semicolon must still be there. For example,
nCount=1;
for(; nCount < 1000; nCount ++)
- The for statement(s) can be followed by a null (empty) statement, so that task is done in the for loop itself.

Null statement consists of a semicolon alone on a line. For example

```
for(count = 0; count < 20000; count++)  
;
```

- This statement provides a pause (delay) of 20,000 milliseconds.

for loop – Example

```
sum=0+1+2+.....+20
```

```
nsum = 0;
```

```
for(irow = 1; irow <=20; irow++)
```

```
    nsum = nsum + irow;
```

```
printf("\n Sum of the first 20 natural numbers = %d",nsum);
```

The above **program segment will compute and display the sum of the first 20 natural numbers**

Nested for loop

- Example:
- for(i=1;i<=2,i++)
- { for(j=1; j<=3;j++)
- c=i*j;
- }

for i=1 j varies from 1 to 3

for i=2 j varies from 1 to 3

```
• #include <stdio.h>  
• #include <conio.h>  
• void main()  
• {  
•     int i,j;  
•     for( i=1; i<=4;i++)  
•         { for (j=1; j<=10; j++)  
•             {  
•                 printf("%d X %d=%d\t",i,j,i*j);  
•             }  
•         }  
•     printf("\n");  
•     getch();  
• }
```

Output:

```
1X1=1    1X2=2    1X3 =3    1X4=4    1X5 =5  
1X6=6    1X7=7    1X8=8    1x9=9    1x10=10
```

```
2X1=2  2X2=4  .....
```

```
3X1=3  3X2=6  3x3=9 .....
```

```
4X1=4  4X2=8  4X3=12 .....
```

- Write a program to print the below structure using "*"

```
*  
*  *  
*  *  *  
*  *  *  *  
  
#include <stdio.h>  
#include <conio.h>  
void main()  
{  
    int i,j;  
    for( i=1; i<=4;i++)  
    { for (j=1; j<=i; j++)  
        {  
            printf("*\t");  
        }  
        printf("\n");  
    }  
    getch();  
}
```

- Write a program to print the below structure using “*”

```

*   *   *   *
*   *   *
*   *
*

#include <stdio.h>
#include <conio.h>
void main()
{
    int i,j;
    for( i=4; i>=1;--i)
        { for (j=1; j<=i; ++j)
            {
                printf(“*\t”);
            }
            printf(“\n”);
        }
    getch();
}

```

Continue statement:

- continue keyword forces the next iteration to take place immediately, skipping any instructions that may follow it.
- The continue statement can only be used inside a loop (for, do-while and while) and not inside a switch-case selection.
- When executed, it transfers control to the condition (the expression part) in a while or do-while loop, and to the increment expression in a for loop.

Unlike the break statement, continue does not force the termination of a loop, it merely transfers control to the next iteration.

```

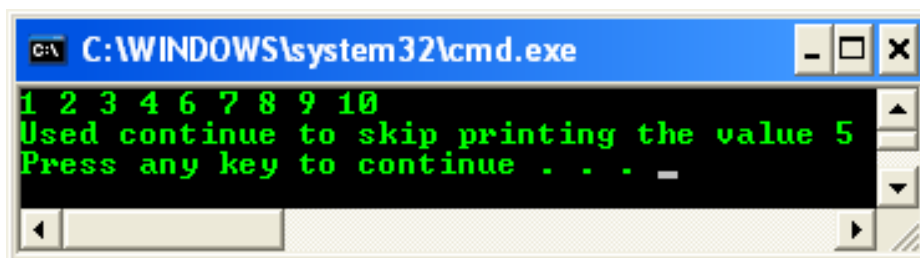
// using the continue in for structure
#include <stdio.h>
int main(void)

```

```

{
    int inum;
    for(inum = 1; inum <= 10; inum++)
    {
        // skip remaining code in loop only if iNum == 5
        if(inum == 5)
            continue;
        printf("%d ", iNum);
    }
    printf("\nUsed continue to skip printing the value 5\n");
    return 0;
}

```



```

C:\WINDOWS\system32\cmd.exe
1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5
Press any key to continue . . . -

```

Goto Statement:

- The goto statement is one of C unconditional jump or branching.
- When program execution encounters a goto statement, execution immediately jumps, or branches, to the location specified by the goto statement.
- The statement is unconditional because execution always branches when a goto statement is came across, the branching does not depend on any condition.
- A goto statement and its target label must be located in the same function, although they can be in different blocks.
- Use goto to transfer execution both into and out of loop.
- However, using goto statement strongly not recommended.
- Always use other C branching statements.
- When program execution branches with a goto statement, no record is kept of where the execution is coming from.

Example:

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

```
int main () {  
  /* local variable definition */  
  int a = 10;  
  /* do loop execution */  
  LOOP:do {  
  
    if( a == 15) {  
      /* skip the iteration */  
      a = a + 1;  
      goto LOOP;  
    }  
  
    printf("value of a: %d\n", a);  
    a++;  
  
  }while( a < 20 );  
  
  return 0;  
}
```

Output:

```
value of a: 10  
value of a: 11  
value of a: 12  
value of a: 13  
value of a: 14  
value of a: 16  
value of a: 17  
value of a: 18
```

value of a: 19

1. Program to check the given number is prime or Not

```
#include<stdio.h>
#include<conio.h>
void main()
{ int n,l,r;
  scanf("%d",&n)
  for (i=2; i<=n/2;i++)
  { r= n % i ;
    if ( r== 0) {
      printf ("given number is not prime");
      getch();
      exit(0);
    }
  }
  printf(" given number is prime");
  getch();
  return 0;
```

Remarks: if we check the number up to half of the given number whether it is divisible or not is enough.

- **Conditional operator: (? : Operator)**

The conditional operator consists of 2 symbols the question mark (?) and the colon (:)

Syntax:

exp1 ? exp2: exp3

example:

```
a=16
b=25;
x=(a>b) ? a : b;
```

working of above expression can be analyzed as follows:

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

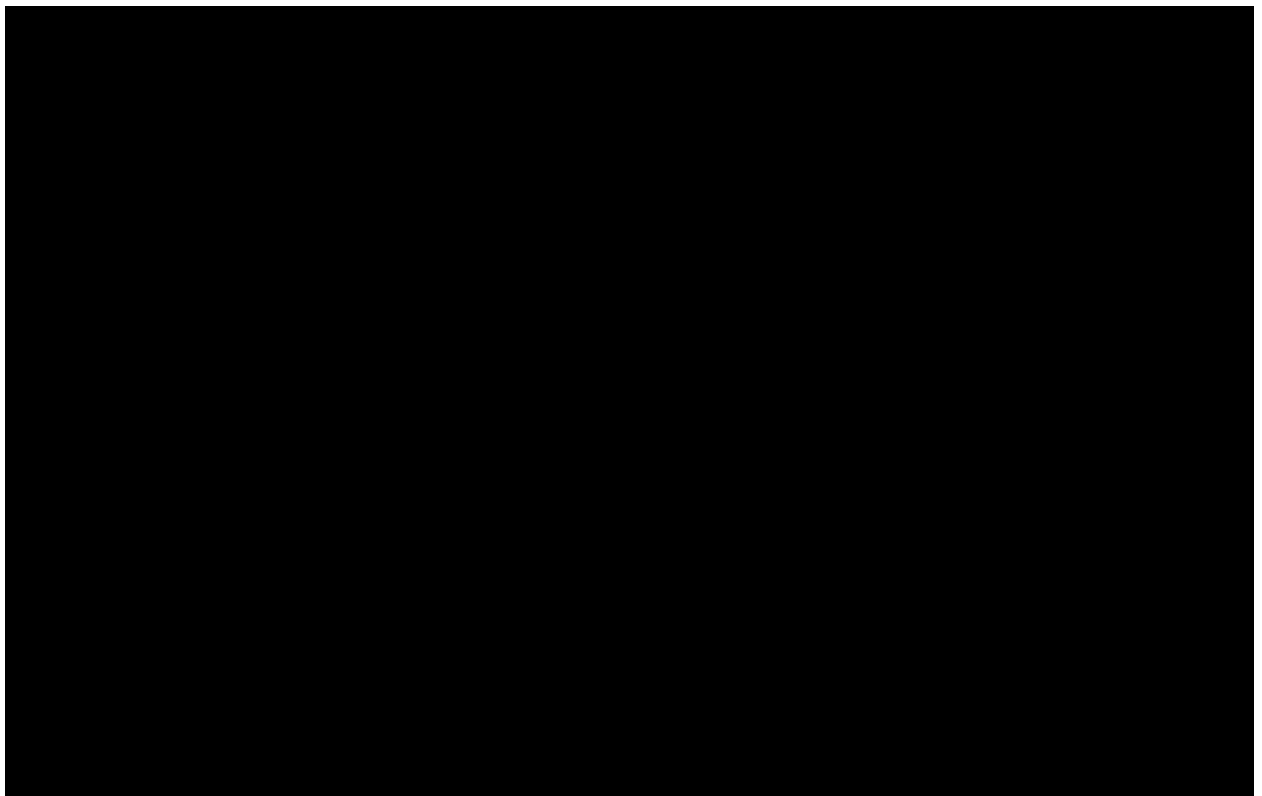
- The `?:` operator is just like an `if ... else` statement except that because it is an operator you can use it within expressions.
- `?:` is a ternary operator in that it takes three values, this is the only ternary operator C has.
- `?:` takes the following form:
if condition is true ? then X return value : otherwise Y value;

```
#include <stdio.h>
main()
{
    int a , b;
    a = 10;
    printf( "Value of b is %d\n", (a == 1) ? 20: 30 );
    printf( "Value of b is %d\n", (a == 10) ? 20: 30 );
}
```

Output:

Value of b is 30

Value of b is 20



```

Step 1: n=x=4
        x-n/x=4-1=3 >0.001
        x=(4+1)/2=2.5

Step 2: 2.5-4/2.5=0.9>0.001
        x=(2.5+1.6)/2=2.05

Step 3: 2.05-4/2.05=0.098
        x=(2.05+1.95)/2=2.0

Step4: 2.0-4/2.0= 0.00 >0.001

square root of 4 is = 2.0

```

Break statement:

The **break** statement in C programming has the following two usages –

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

If you are using nested loops, the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

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

```

int main () {
    int a = 10;

    while( a < 20 )
    {
        printf("value of a: %d\n", a);
        a++;

        if( a > 15) {
            /* terminate the loop using break statement */
            break;
        }
    }
    return 0;
}

```

Output:

value of a: 10

value of a: 11
 value of a: 12
 value of a: 13
 value of a: 14
 value of a: 15

2. Bitwise operator

- It takes the operand as string of bits
- Bit operations are carried out on the data
 - & bitwise AND operator
 - | bitwise OR operator
 - ~ NOT operator
 - << left shift operator
 - >> right shift operator

- X=4
 $x = x \ll 3$

	64	32	16	8	4	2	1
	0	0	0	0	1	0	0
First shift	0	0	0	1	0	0	0
Second shift	0	0	1	0	0	0	0
Third shift	0	1	0	0	0	0	0

Answer $x = 32$

- X=96
 $x = x \gg 3$

	64	32	16	8	4	2
98	1	1	0	0	0	0
First shift(48)	0	1	1	0	0	0
Second shift(24)	0	0	1	1	0	0
Third shift (12)	0	0	0	1	1	0

Answer x= 12

```
#include<stdio.h>
int main()
{ int a,b;
  a=98;
  b=4;
  a= a>>3;
  b=b<<3;
  printf(" right shift of a for 3 times results in=%d\n",a);
  Printf("left shift of b for 3 times results in =%d",b);
  return 0;
}
```

Output:

right shift of a after 3 times results in=12
left shift of b after 3 times results in =32

Type Casting:

- Process of converting an expression of a given type to another type
example: int to float
float to int
- When two operands in an expression are different, user explicitly changes the data type, this is known type conversion

```
#include<stdio.h>
int main()
{ int a=3;
  float b=0.0;
  b=1/a;
  printf(“”b=%f”,b);
}
```

output
b=0.0

```
#include<stdio.h>
int main()
{ int a=3;
  float b=0.0;
  b=1/(float)a;
  printf(“”b=%f”,b);
}
```

output
b=0.3333

3. Special operator

Comma operator: combines multiple expressions into single expression. The value of right most expression is assigned to left expression

```
x=(a=4,b=5,c=2,c+a*b);
```

x=22

4. Size of operator : sizeof()

used to determine size of the operand based on its data type

```
float r= 12.45;
```

```
int y;
```

```
y=sizeof(r);
```

value of y is 4 because size of float is 4 bytes.

Formatted output statement:

```
a= 14
```

```
printf("%d", a) → 14
```

```
printf("%4d",a) → bb14
```

```
printf("%.3d",a) → b014
```

```
printf("%-4d",a) → 14bb
```

```
printf("%-4.3d",a) → 014b
```

Note:

b -> stands for blank

```
float a= 10.437
```

```
printf("%f",a) → 10.437000( it provides 6 digit  
after decimal point)
```

```
printf("%w.df",a)
```

w-> number of columns for integer part

d-> number of digits to be limited after
the decimal point

```
Printf("%4.2f",a) -> 10.44
```

Comparison of getchar(),getche(),getch() functions

- getchar() -> used to read single character at run time. Given value is displayed on the screen and the compiler wait for another character to be typed.
- getche()-> it is used to get a character from console and echoes to the screen. The given value is displayed on the screen and the compiler does not wait for another character to be typed
- getch() -> is used to get a character from console but does not echo to the screen.

Exercises:

- What is a variable and constant? List the rules to be followed while declaring a variable.
- What are data types? List the data types available in C programming language
- List and explain with appropriate example the various operators used in C language.
- Compare while loop with do-while loop of C programming language.
- Write a program to exchange (swap) two numbers.
- Write a program to check the given number is prime or not
- Write a program to generate fibonacci series

1 1 2 3 5 8 13