

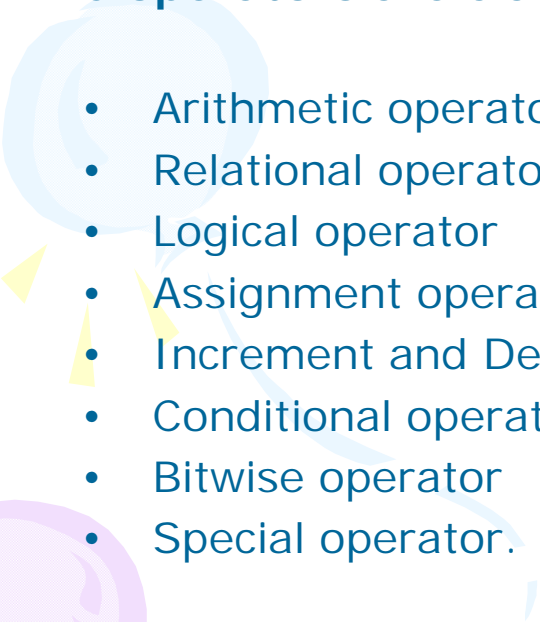



Operators and Expression:

C support a rich set of built in operator.

- An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.

C operators are classified into following categories:

- 
- Arithmetic operator
 - Relational operator
 - Logical operator
 - Assignment operator
 - Increment and Decrement operator
 - Conditional operator
 - Bitwise operator
 - Special operator.
- 



Arithmetic operator:

+ (Addition), - (Subtraction), * (Multiplication), / (Division), %(Modulo Division). All these are arithmetic operators.

ex:

a-b a+b

a*b a/b


a%b -a*b



Here a and b are variables and are known as operands.

Note : Modular operator can not be used in the floating point operation.

Arithmetic operator has three types:

- (1) Integer Arithmetic
 - (2) Real Arithmetic
 - (3) Mixed mode Arithmetic
- 



Integer Arithmetic:

When both the operands in a arithmetic expressions are integers, the expression is called an integer expression, and the operation is called integer arithmetic.

If a and b both are integer than for a=14 and b=4 we have following results:

$$a - b = 10$$

$$a + b = 18$$

$$a * b = 56$$

$$a / b = 3 \quad (\text{decimal part truncated})$$

$$a \% b = 2 \quad (\text{remainder of division})$$



During modular division, the sign of the result is always the sign of the first operand (dividend). That is

$$-14\%3 = -2$$

$$-14\%-3 = -2$$

$$14\%-3 = 2$$

Ex : /* Program to convert a given number of days into months and days*/

main()

{

int month, day;

clrscr();

scanf("%d",&day);

month = day/30;

day = day%30;

printf("Month = %d Days = %d", month,day);

getch();

}



Real Arithmetic :

An arithmetic expression involving only real operands is called real arithmetic. A real operands may assume values either in decimal or exponential notation.

If x,y,z are floats, then we will have:

$$x = 6.0/7.0 = 0.857143$$

$$y = -2.0/3.0 = -0.666667$$

the operator % can not be used with real operands.

Mixed Mode Arithmetic:

When one of the operand is real and the other is integer, the expression is called a mixed mode arithmetic expression. If either operand is of real type then only real operation is performed and the result is always a real number. Thus

$$15/ 10.0 = 1.5$$

where as

$$15/10 = 1$$

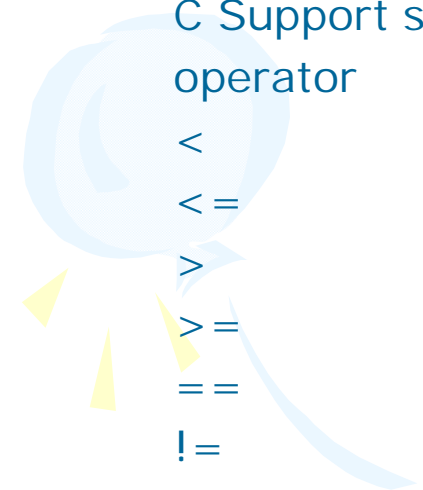


Relational operators:


Whenever you want to compare two entities at that time you can use the Relational operator.

An expression containing a relational operator is known as relational expression.

C Support six type of the relational operators:



operator	Meaning
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to



Relational operators are generally used in the decision making statements like if and while and it returns value true and false



ae-1 relational operator ae-2

Ex: 4.5 <= 10 true
 a+b == c+d true if equal else false
 10< 7+5 true
 -35 >= 0 false

Logical Operators:

Logical operators are generally used when you want to test more than one condition.

There are three types of the logical operator:

Logical And &&

Logical Or ||

Logical Not !

An example is:

a>b && x==10



An expression of this kind, which combines two or more relational expressions, is termed as a logical expression or a compound relational expression.



Truth Table:

Op-1	Op-2	value of the expression	
		op-1 && op-2	op1 op2
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0


Example :

if(age > 55 && salary < 1000)

if(number < 0 || number > 100)

Assignment Operators:

Assignment operators are used to assign the result of an expression to a variable. We have seen the usual assignment operator '='. In addition c has a '*shorthand*' assignment operators of the form.





Syntax:

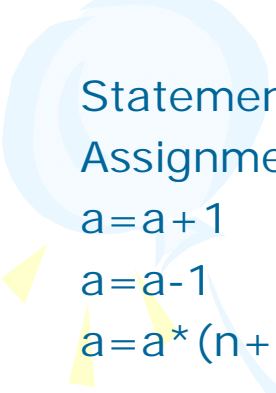
$v \text{ op} = \text{exp}$

Where v is a variable, exp is an expression and op is a c binary arithmetic operator.

Is equivalent to :

$V = v \text{ op} (\text{exp})$

Ex: $x += 1$ is equivalent to $x = x + 1$



Statement with simple
Assignment operator

$a = a + 1$

$a = a - 1$

$a = a * (n + 1)$


Statement with shorthand
operator.

$a += 1$

$a -= 1$

$a *= n + 1$

The use of the shorthand assignment operator has three advantages.

- 
1. Left hand side need not to be repeated so easier to write.
 2. Statement is more concise and easier to read.
 3. Statement is more efficient.



Increment(++) and Decrement(--) Operator:

++ operator add 1 to the operand, while -- subtracts 1. Both are unary operators and take both forms.

++m; or --m;
m++; or m--;

We generally use this statement in the for and while loop.

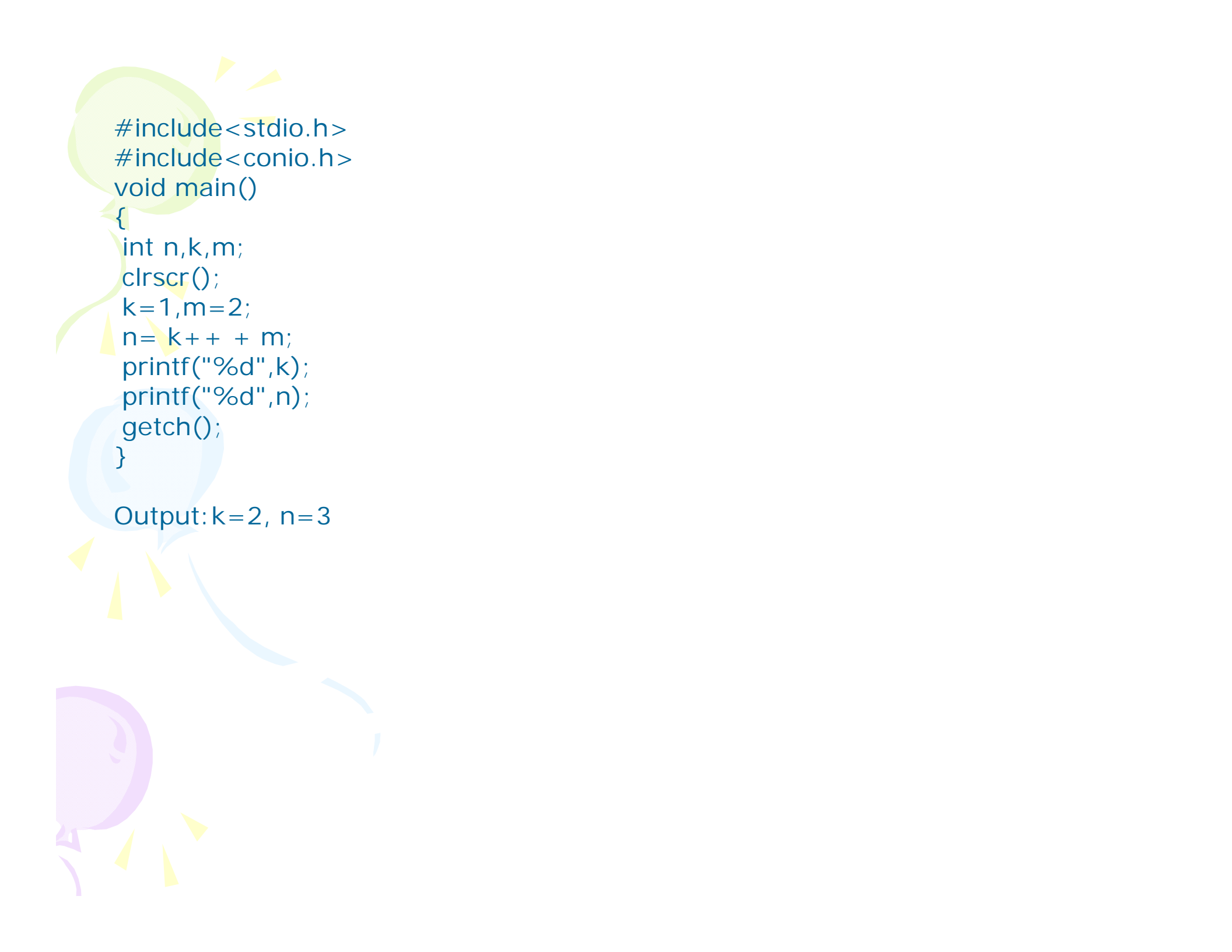


++m and m++ both are different things when they are used in the expression.
For m=5;



y= ++m In this case value of y and m would be 6

y= m++ In this case value of y would be 5 and m would be 6.



```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,k,m;
    clrscr();
    k=1,m=2;
    n= k++ + m;
    printf("%d",k);
    printf("%d",n);
    getch();
}
```

Output: k=2, n=3



Conditional operator:

Conditional operator is also called the "ternary operator".

Syntax:

`exp1 ? exp2 : exp3`

Ex:

`a = 10;`

`b = 15;`

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

Its same as

`if (a > b)`

`x = a;`

`else`

`x = b;`





Special Operator :

- C Support some special operator like comma, and sizeof operator.

Comma Operator :

Comma Operator can be used to link the related expression together.

Ex:

```
Value = (x=10, y=5, x+y);
```




The sizeof Operator:

Sizeof operator is a compile operator. It returns number of the bytes occupied by the operand. The operand may be a variable, constant or a data type.

Ex:

```
m = sizeof(sum);  
n = sizeof(long int);  
k = sizeof(235L);
```



```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,c,d;
    clrscr();
    a=15, b= 10;
    c= ++a - b; // value of a is incremented before use in expression
    printf("a=%d b=%d c=%d\n", a,b,c);
    d= b++ + a; // value of b is increment after use in expression
    printf("\na = %d b = %d d = %d",a,b,d);
    printf("\n a/b = %d",a/b);
    printf("\n a%%b = %d",a%b);
    printf("\n a*=b = %d", a*=b);
    printf("%d\n", (c>d) ? 1:0);
    printf("%d",sizeof(int));
    getch();
}
```

Output :

```
a=16 b=10 c=6
a = 16 b = 11 d = 26
a/b = 1
a%b = 5
a*=b = 1760
```

ARITHMETIC EXPRESSION :

An arithmetic expression is a combination of variables, constants, and operators arranged as per the syntax of the language.

Example :

Algebraic Expression

$a \times b - c$

$(m+n)(x+y)$

ab/c

$3x^2 + x + 1$

C Expression

$a * b - c$

$(m+n) * (x+y)$

$a*b/c$

$3*x*x + x + 1$

EVALUATION OF EXPRESSION :

Syntax : variable = expression;

Ex: $x = a * b - c; z = a - b / c + d;$

$x = a - b/3 + c*2 - 1 \quad a=9 \ b=12 \ c=3$

Precedence of Arithmetic operators:

High Priority : * / %

Low Priority : + -

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

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

```
void main()
```

```
{
```

```
    float a,b,c,x,y,z;
```

```
    clrscr();
```

```
    a = 9; b = 12; c = 3;
```

```
    x = a -b / 3 + c * 2 -1;
```

```
    y = a -b / (3 + c) * (2-1);
```

```
    z = a -(b / (3 + c) *2) -1;
```

```
    printf("x = %f y = %f z = %f", x,y,z);
```

```
    getch();
```

```
}
```

Output : x =10.000000 , y = 7. 000000 , z = 4. 000000



Rules for Evaluation Expression :

- First, parenthesized sub expression from left to right are evaluated
- If parentheses are nested, the evaluation begins with the innermost sub expressions
- The associativity rule is applied when two or more operator of the same precedence level appear in sub expression.
- The precedence rule is applied to determine the order of operator in evaluating expression.
- Arithmetic operators are evaluated left to right using the rules of precedence
- When parentheses are used, the expressions within parentheses assume highest priority.

Type Conversion in Expression :

There are two types of the type conversions:

- Implicit Type Conversion
- Explicit Type Conversion

Implicit Type Conversion :

- When data value automatically convert from one type to another type is called the Implicit type conversion.
- If the operands are of different types, the lower type is automatically converted to the higher type before the operation proceeds.

e.g. `int i,x;
float f;
double d;
long int l;
x = l/i + i*f - d`

Ex:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b;
    float c;
    a= 12,b=13;
    c= a+b;
    printf("%f",c);
    getch();
}
```

// automatic type conversion

/* Output 25.00000 */

Explicit Type Conversion :

When a process of the conversion done manually of locally then this process is called the explicit type conversion process or casting operation.

Syntax : (type_name) expression :

Ex:

```
x = (int) 7.5;           a = (int) 21.3 / (int)4.5
b= (double) sum /n      y = (int) (a+b);
z = (int) a + b;
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
    float a,b;
    int c;
    clrscr();
    a=12,b=13;
    c= (int) (a+b);
    printf("%d",c);
    getch();
}
```

// Type Conversion float to int

/* Output 25 */

Note : whenever you try to convert higher data type to lower type your result will be loss.



Operator Precedence and Associativity :

There are distinct levels of precedence and an operator may belong to one of these levels . The operators at the higher level of precedence are evaluated first. The operators of the same precedence are evaluated either from 'left to right' or from 'right to left', depending on the level. This is known as the associativity property of an operator.

Consider the following conditional statement.

```
if (x == 10 + 15 && y < 10)
```

Rules of Precedence and Associativity

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



Decision Making and Branching

if statement
switch statement
Conditional operator statement
goto statement

All the above statements are known as decision – making statement.


These statements 'control' the flow of the execution, they are known as control statements.



Decision Making with If statement :

If statement used to control the flow of the execution.

The if statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are :

1. Simple if statement
 2. if.....else statement
 3. Nested if.....else statement
 4. else if ladder
- 



Simple If Statement

Syntax: `if(test_expresson)`
 {
 statement-block;
 }
statement-x;

If `test_expression` condition is true, the `statement_block` will be executed.





Ex:

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

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

```
void main()
```

```
{
```

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

```
float ratio;
```

```
clrscr();
```

```
scanf("%d %d %d %d",&a,&b,&c,&d);
```

```
if(c-d != 0)
```

```
{
```

```
ratio = (float) (a+b)/(float) (c-d);
```

```
printf("Ratio = %f\n",ratio);
```

```
}
```

```
getch();
```

```
}
```




The If....Else Statement :

The if...else statement is an extension of the if statement. It is generally used whenever you want to execute the condition's true and false part.

Syntax :

```
If(test_expression)
{
    true- block statements;
}
else
{
    false – block statements;
}
statement – x;
```

If test_expression condition is true execute the true-block statements else execute the false – block statements.





Ex:

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

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

```
void main()
```

```
{
```

```
    int a,b;
```

```
    clrscr();
```

```
    scanf("%d %d", &a, &b);
```

```
    if (a>b)
```

```
        printf("A is bigger than B : %d",a);
```

```
    else
```

```
        printf("B is bigger tha A : %d",b);
```

```
    getch();
```

```
}
```



Nesting of If...Else statements:

When a series of statements are involved, we may have to use more than one if...else statements in a program in nested form is called the nesting of if...else statements.

Syntax:

```
if(test_condition 1)
{
    if(test_condition 2)
    {
        statement – 1;
    }
    else
    {
        statement – 2;
    }
}
else
{
    statement – 3;
}
Statement – x;
```

Ex:

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

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

```
void main()
```

```
{
```

```
    float a,b,c; clrscr();
```

```
    scanf("%f %f %f",&a,&b,&c);
```

```
    if(a>b)
```

```
    {
```

```
        if(a>c)
```

```
            printf("%f\n",a);
```

```
        else
```

```
            printf("%f\n",c);
```

```
    }
```

```
    else
```

```
    {
```

```
        if(c>b)
```

```
            printf("%f\n",c);
```

```
        else
```

```
            printf("%f\n",b);
```

```
    }
```

```
    getch();
```

```
}
```

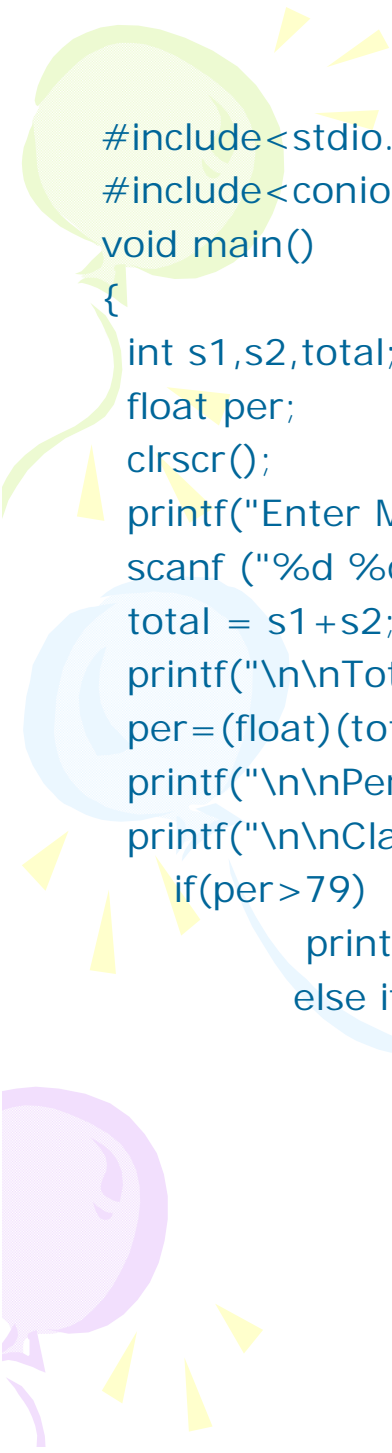
Else If Ladder:

There is another way of putting ifs together when multipath decisions are involved. A multipath decision is a chain of ifs in which the statement associated with each else is an if. This construct is known as the else..if ladder.

Syntax:

```
if (Condition 1)
    statement – 1;
else if (condition 2)
    statement – 2;
else if (condition 3)
    statement – 3;
.....
.....
else
    default -statement;
Statement x;
```

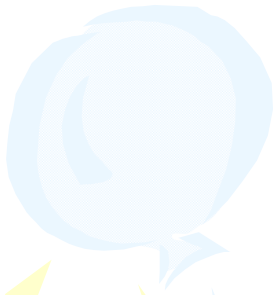
The conditions are evaluated from the top to downwards. As soon as a true condition is found, the statement associated with it is executed and the statement is transferred to the statement x. When all the condition become false default -statement will be executed.



```
#include<stdio.h>
#include<conio.h>
void main()
{
    int s1,s2,total;
    float per;
    clrscr();
    printf("Enter Marks for s1 & s2 :");
    scanf ("%d %d",&s1,&s2);
    total = s1+s2;
    printf("\n\nTotal Mark is : %d",total);
    per=(float)(total*100/200);
    printf("\n\nPercentage is : %f",per);
    printf("\n\nClass of the Student :");
    if(per>79)
        printf("Honours:");
    else if(per>59)
        printf("First");
    else if(per>49)
        printf("Second");
    else if(per>39)
        printf("Third");
```



printf("Fail");





The Switch Statement :


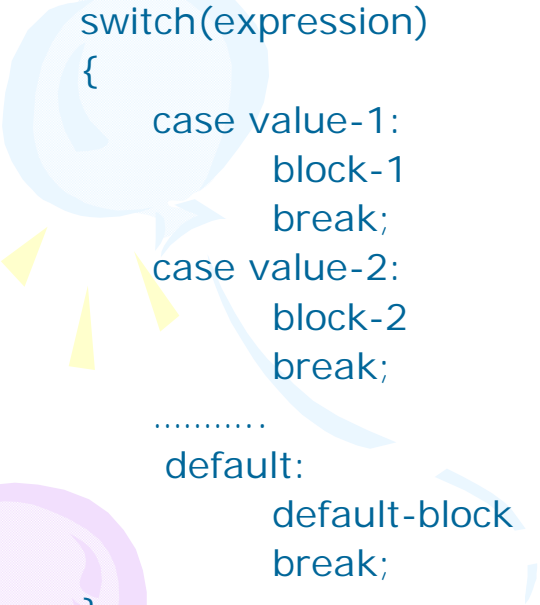
When one of the many alternative is to be selected, we can use an if statement to control the selection. However the compexity of such program increases when the number of the alternative increases.

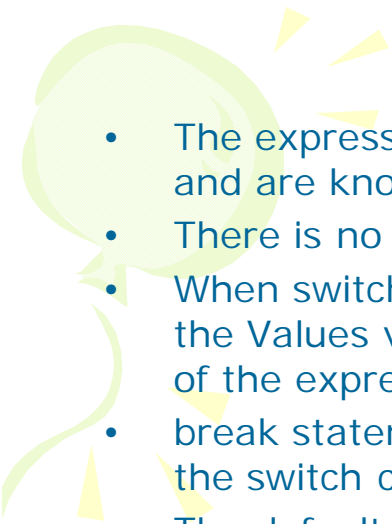
C has a built in multiway decision statement known as switch.

The switch statement tests the value of a given variable against a list of case value and when a match is found, a block of statement associated with that case is executed.

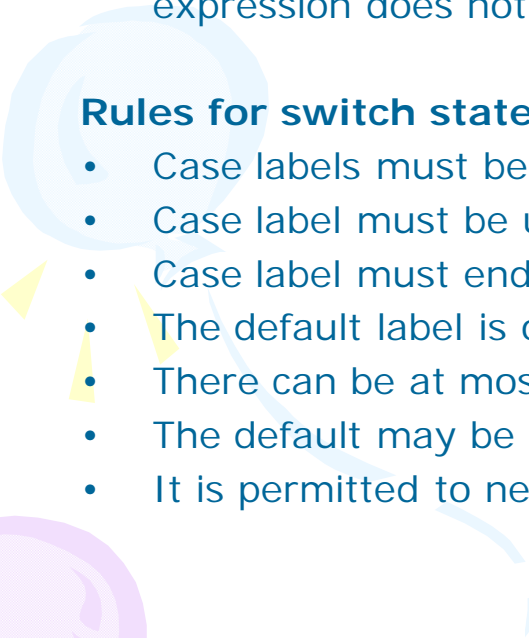

Syntax:

```
switch(expression)
{
    case value-1:
        block-1
        break;
    case value-2:
        block-2
        break;
    .....
    default:
        default-block
        break;
}
```



- 
- The expression is an integer or character expression. Value1, value2 are constants and are known as case labels.
 - There is no need to put braces around these blocks.
 - When switch is executed, the value of the expression is successfully compared against the Values value1, value2.... If the case is found whose value matches with the value of the expression, then the block of statements that follows the case are executed.
 - break statement exit the control from the switch case and transfer the control after the switch case statement.
 - The default is an optional case, when present, it will be executed if the value of the expression does not match with any of the case values.

Rules for switch statement:

- 
- Case labels must be constants or constant expression.
 - Case label must be unique. No two case labels can have the same value.
 - Case label must end with colon.
 - The default label is optional.
 - There can be at most one default label.
 - The default may be placed any where but usually placed at the end
 - It is permitted to nest switch statements.
- 



// It is the example of switch case statement

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

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

```
void main()
```

```
{
```

```
    int s1,s2,total,index;
```

```
    float per;
```

```
    clrscr();
```

```
    printf("Enter Marks for s1:");
```

```
    scanf ("%d",&s1);
```

```
    printf("Enter Marks for s2 :");
```

```
    scanf ("%d",&s2);
```

```
    total = s1+s2;
```


```
    printf("\n\nTotal Marks s : %d",total);
```

```
    per=(float)(total*100/200);
```

```
    printf("\n\nPercentage is :%f",per);
```

```
    printf("\n\nClass of the Student :");
```

```
    index=(int)(per/10);
```



```
switch(index)
{
    case 10:
    case 9:
    case 8:
        printf("Honours");
        break;
    case 7:
    case 6:
        printf("First");
        break;
    case 5:
        printf("Second");
        break;
    case 4:
        printf("Third");
        break;
    default:
        printf("Fail");
}
getch();
}
```

Go TO Statement :

C support go to statement to branch unconditionally from one point to another in the program.

Although it may not be essential to use the goto statement in a highly structured language like C. There may be occasions when the use of goto might be desirable

The goto requires a label to identify the place where the branch is to be made. Label is valid variable name and must be followed by a colon.

Label is placed before the statement where control is to be transferred.

```
goto label;
```

```
.....
```

```
.....
```

```
.....
```

```
label:
```

```
Statement;
```

[Forward Jump]

```
label:
```

```
Statement;
```

```
.....
```

```
.....
```

```
.....
```

```
goto label;
```

[Backward Jump]




Backward Jump.

If a label is before the statement goto, a loop will be formed and some statements will be Executed repeatedly. Such a jump is known as Backward Jump.

Forward Jump:

If the label is placed after the goto label, some statements will be skipped and the jump is known as a forward jump.



Ex:

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

```
void main()
```

```
{
```

```
float x,y;
```

```
clrscr();
```

```
read:
```

```
scanf("%f",&x);
```

```
if(x<0)
```

```
{
```

```
printf("This is a Negative value");
```

```
goto read;
```

```
}
```

```
else
```

```
{
```

```
y=sqrt(x);
```

```
printf("Suare root of %f is %f ",x,y);
```

```
}
```

```
goto read;
```

```
getch();
```

```
}
```

Note : Infinite loop will be occurred because of the above case.

Solution of the infinite loop in goto statement :

Ex:

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

```
void main()
```

```
{
```

```
    float x,y;  int count=1;
```

```
    clrscr();
```

```
    printf("Enter FIVE real values in a LINE \n");
```

```
    read:
```

```
    scanf("%f",&x);
```

```
    if(x<0)
```

```
    {
```

```
        printf("This is a Negative value");
```

```
        goto read;
```

```
    }
```

```
    else
```

```
    {
```

```
        y=sqrt(x);
```

```
        printf("Suare root of %f is %f ",x,y);
```

```
    }
```

```
    count++;
```

```
    if(count <= 5)
```

```
        goto read;
```

```
    printf("\n End of the computation");
```

```
    getch();
```

```
}
```



Decision Making and Looping

What is looping?

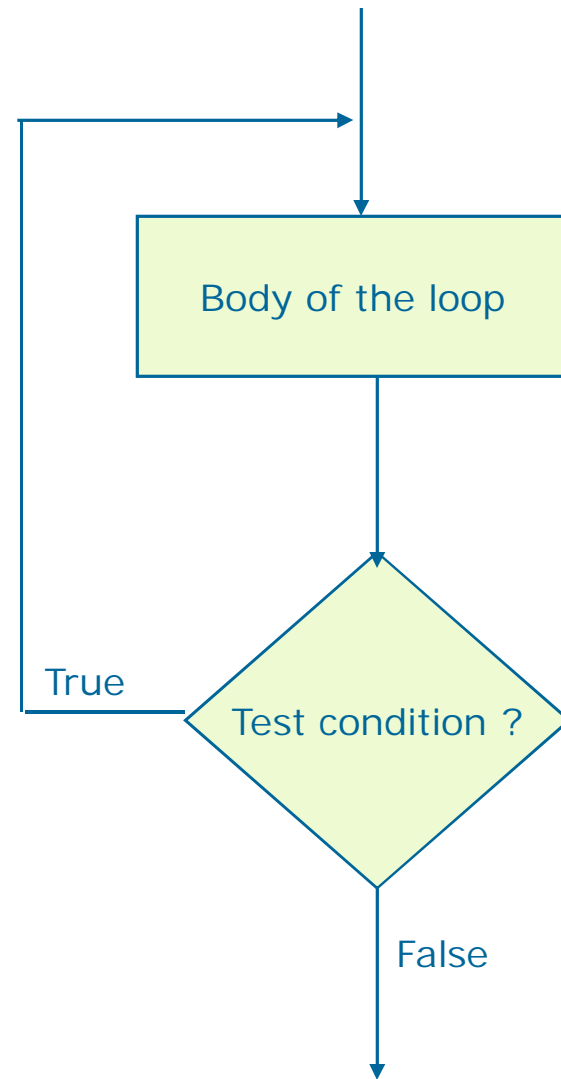
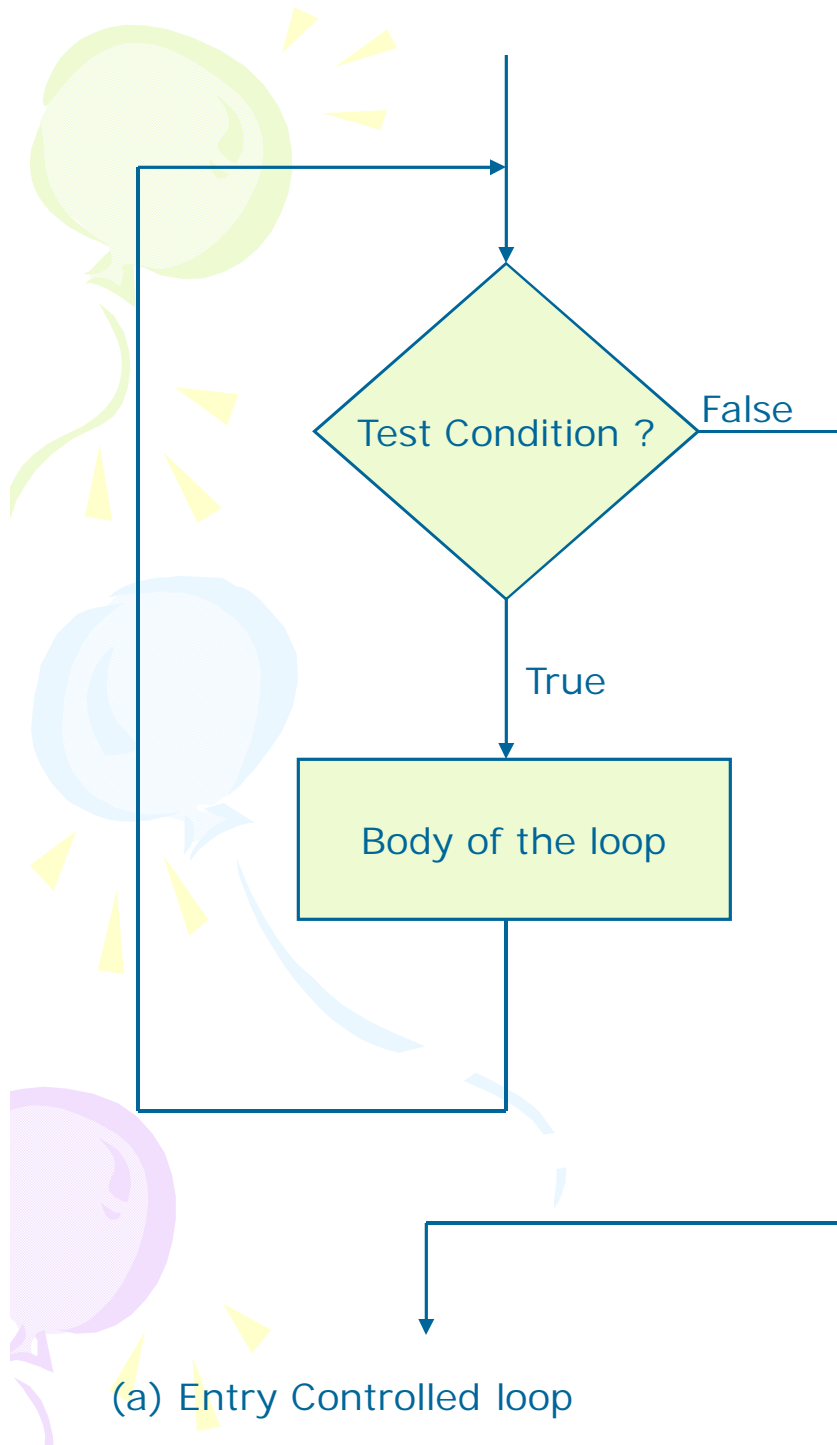
A sequence of statements are executed until some condition for termination of the loop are satisfied is called looping.

There are two types of the looping.

- (a) Entry controlled loop
- (b) Exit controlled loop

Entry controlled loop : In this looping first condition will be checked and then Body of the statements are executed is called the entry controlled loop.

Exit controlled loop: In this looping first body of the statements are executed and then condition will be checked is called the exit controlled loop.



(b) Exit Controlled loop



The c language provides for three constructs for performing loop operation.

- The while statement
- The do statement
- The for statement.

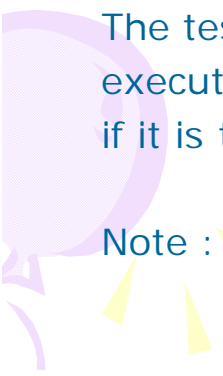
The while statement :

The while is an entry controlled loop statement.

Syntax:

```
while (test condition)
{
    Body of the loop;
}
```

Here while is a key word.



The test condition is evaluated and if the condition is true, the body of the loop is executed. After the execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again.

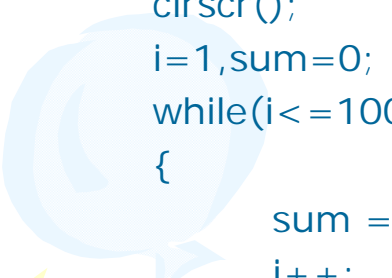
Note : Braces is needed if the body contains more than one statements.



Ex:

//program:-9 to find the sum of first 100 natural nos:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,sum;
    clrscr();
    i=1,sum=0;
    while(i<=100)
    {
        sum = sum + i;
        i++;
    }
    printf("\nSummation of first 100 is : %d",sum);
    getch();
}
```



Hw : Write the program to find the summation of first five no's square.

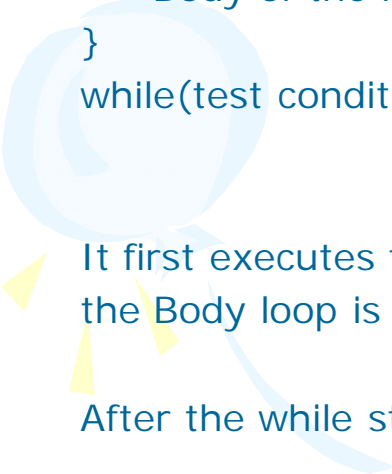


Do...While statement :

Do...while statement is exit controlled loop.

Syntax:

```
do
{
    Body of the loop;
}
while(test condition);
```



It first executes the body of loop then evaluates the test condition. If the condition is true, the Body loop is executed again and again untill the condition become false.

After the while statement there is a semicolon in do..while statement.



If fist time condition become false, though it executes the body of the loop once.


```
/* Find the factorial of any number */
#include<stdio.h>
#include<conio.h>
void main()
{
    int n, fact=1;
    clrscr();
    printf("Enter the number :");
    scanf("%d",&n);
    if(n<0)
        printf("\n factorial of negetive number is not possible :");
    else if(n==0)
        printf("Factorial of 0 is 1\n");
    else
    {
        do
        {
            fact *= n;
            n--;
        }
        while(n>0);
        printf("Factorial of the given no. is : %d", fact);
    }
    getch();
}
```



For Loop:

The for loop is another entry control loop that provides a more concise loop control structure.


Syntax :




```
for( initialization; test-condition ; increment or decrement)
{
    Body of the loop
}
```



Initialization: In Initialization part you can initialize the variable. Such as $i=0$.



Test-Condition: The value of the control variable is tested using the test – condition. Test condition is relational expression, such as $i < 10$ that determines when loop will exit. If the condition is true body of the loop is executed otherwise the loop is terminated.

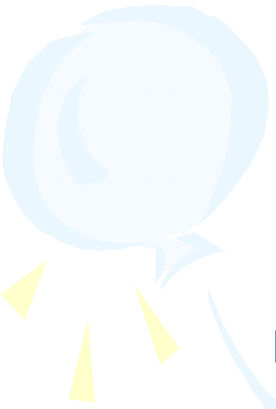


Increment or Decrement : When the Body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the loop. Now the control variable is incremented or decremented.




Ex:

/* Program to Display 1 to 10 */




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





/* Program to Display 10 to 0 */



```
#include<stdio.h>
#include<conio.h>
void main()
{
    int x;
    for(x=10; x>=0 ; x--)
    {
        printf("%d",x);
    }
    printf("\n");
}
```

Additional features of the for loop :

More than one variable can be initialized at a time in the for statement.

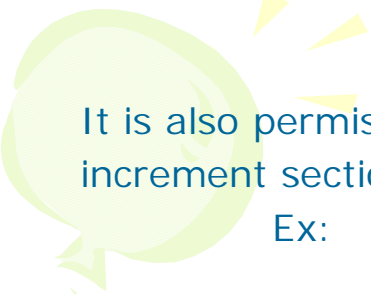
Ex: `p=1;`
 `for (n=0; n<17; n++)`
 is equivalent to
 `for(p=1,n=0; n<17; n++)`

Increment or Decrement section also has more than one statement.

Ex: `for(n=1, m=50; n<=m; n++, m--)`
 `{`
 `p=m/n;`
 `printf("%d %d %d\n", n,m,p);`
 `}`

The test condition may have any compound relation.

Ex: `sum=0;`
 `for(i=1; i<20 && sum < 100; i++)`
 `{`
 `sum = sum + i;`
 `printf("%d %d\n", i, sum);`
 `}`

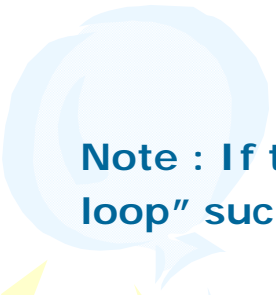


It is also permissible to use expression in the assignment statements of initialization and increment sections.

Ex: for($x = (m+n)/2$; $x > 0$; $x = x/2$)

Another feature of the for loop is that one or more section can be omitted, if necessary.

Ex: m=5;
 for(; m != 100 ;)
 {
 printf("%d\n", m);
 m++;
 }




Note : If test condition is not present, the for statement sets up an “infinite loop” such loop can be broken using break and goto statements in the loop.

We can set up the time delay loops using the null statement as follow:

 for(j=1000; j>0; j--) ;

 This loop is executed 1000 times without producing any output; it simply causes time delay.

The Body of the loop contains only a semi colon, known as *null* statement.



Note : The c compiler does not give any error message if we place semicolon by mistake at the end of a for statement.

Nesting of for loop:

Nesting of for loops, that is one for statement within another for statement.

Ex:

```
for ( i=1 ; i< 10; i++)  
{  
    for ( j=1; j != 5; j++ )  
    {  
        .....  
        .....  
    }  
    .....  
    .....  
}
```

Outer loop

Inner loop

Note : ANSI C allow up to 15 levels of nesting.



```
/* Program to display the output on the screen */
```

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

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

```
void main()
```

```
{
```

```
    int i,j,n;
```

```
    clrscr();
```

```
    printf("Enter the value of i :");
```

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

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

```
    {
```

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

```
        {
```

```
            printf("*");
```

```
        }
```

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

```
    }
```

```
    getch();
```

```
}
```



/*

Enter the value of n : 4



*

**



*/



Jumps in Loops :

There are two statements break and continue are used to performing operation jumps in loops.


Jumping Out of a Loop : break statement

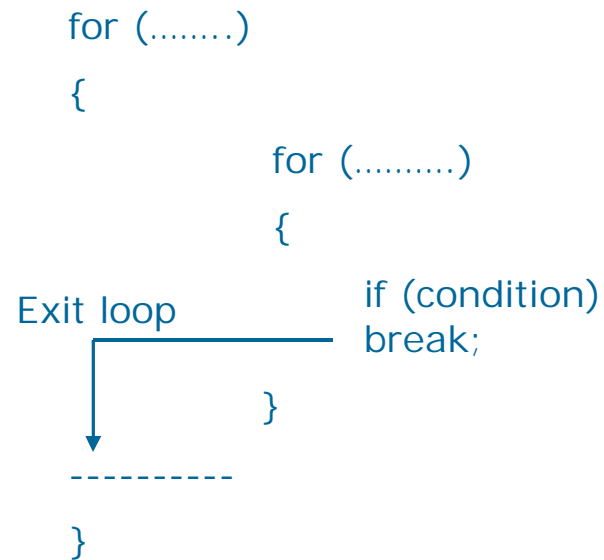
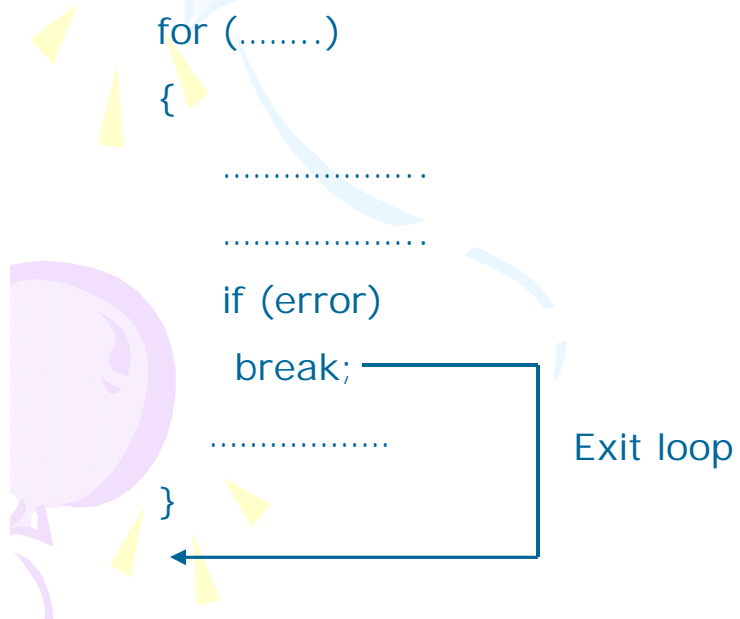
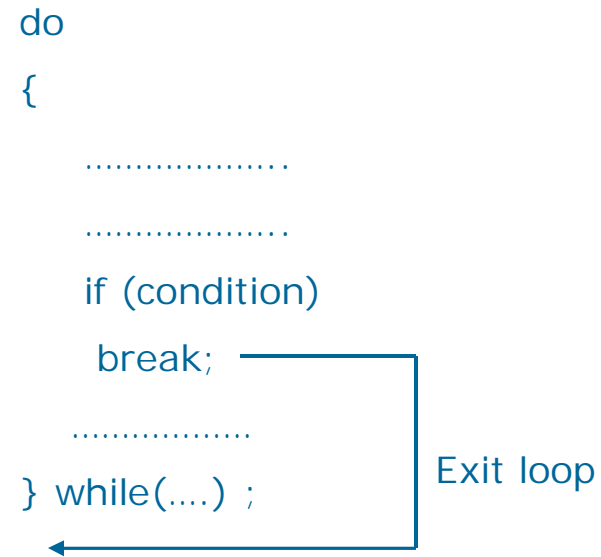
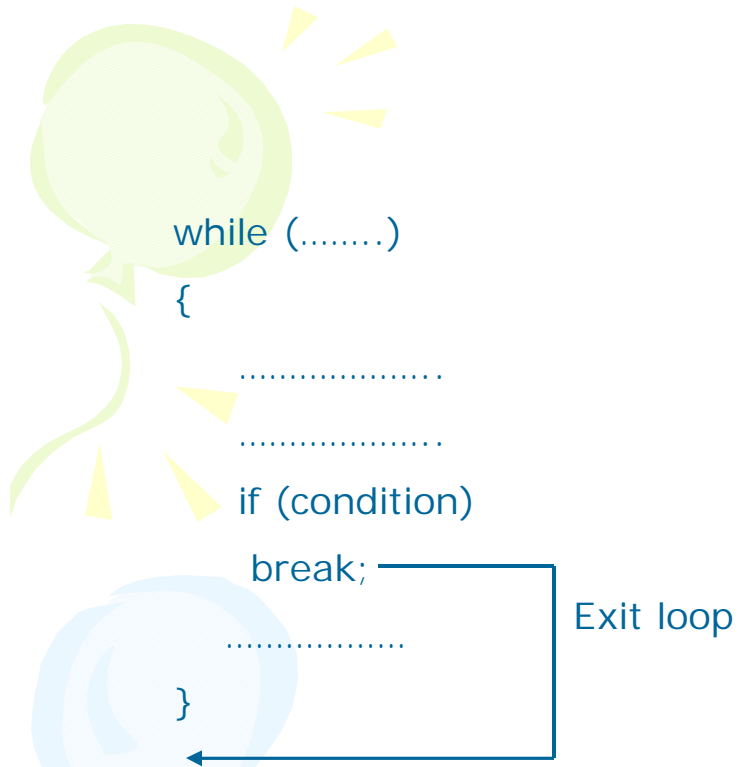
When the break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop. When the loops are nested, the break would only exit from the loop containing it. That is break will exit only a single loop.

Syntax :

`break;`

Since a goto statement can transfer the control to any place in a program . Another important use of goto is to exit from deeply nested loops when an error occurs.








Skipping Part of a Loop : continue statement

The use of the continue statement is to "skip the following statements and continue with the next iteration".

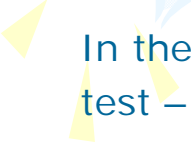
Syntax : `continue;`




Unlike the break which causes the loop to be terminated, the continue as the name implies, causes the loop to be continued with the next iteration after skipping any statements in between.

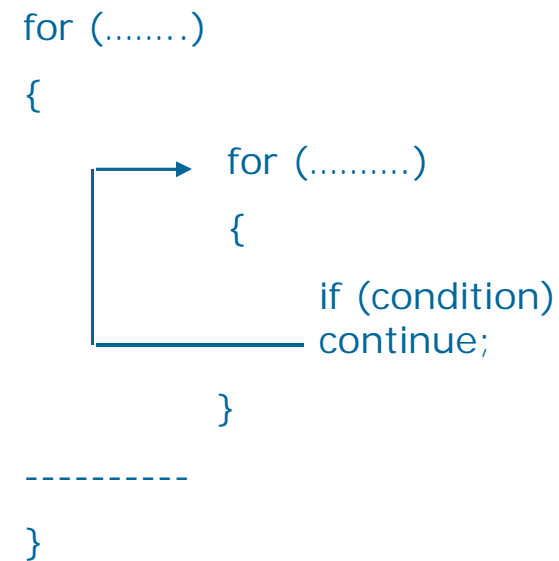
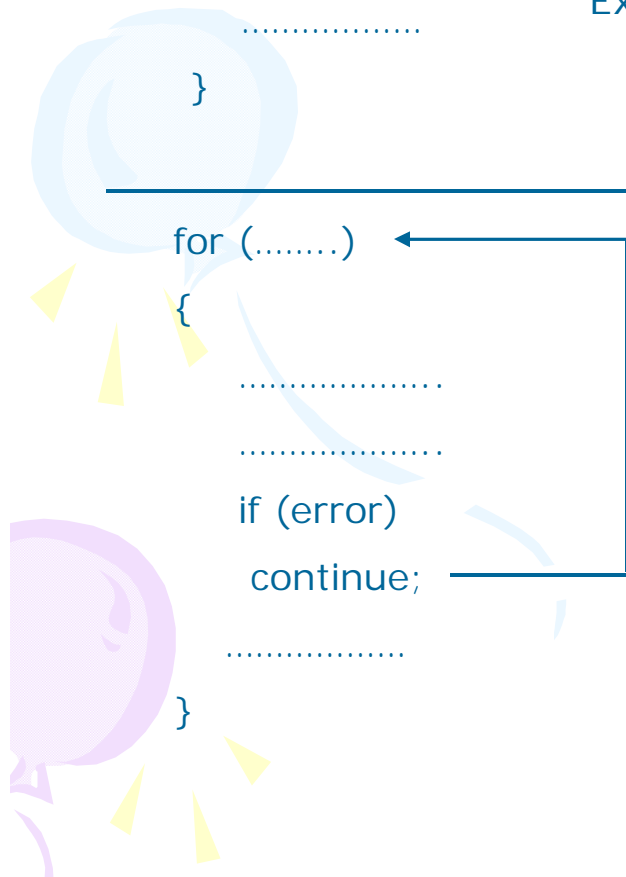
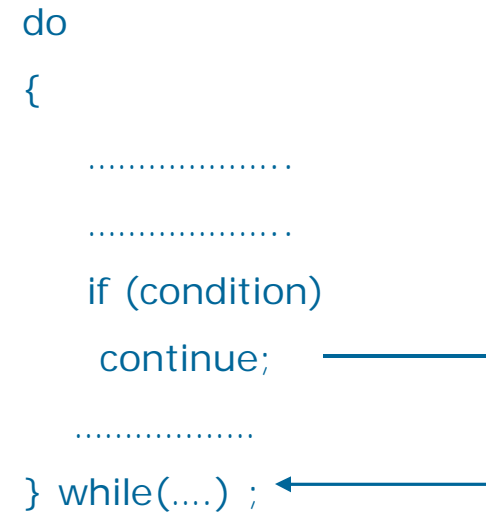
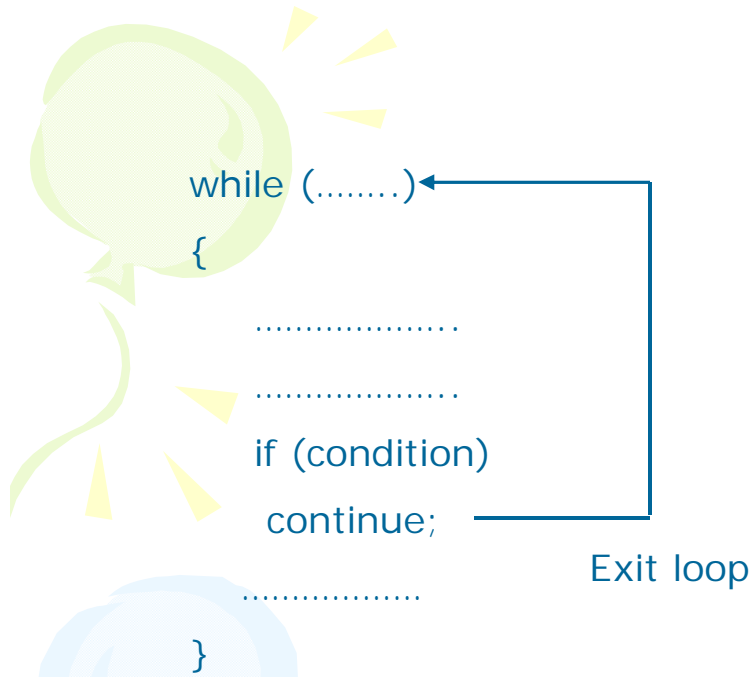


In while and do loops, continue causes the control to go directly to the test condition and then do continue the iteration process.



In the case of for loop increment section of the loop is executed before the test – condition is evaluated.







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

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

```
#include<math.h>
```

```
void main()
```

```
{
```

```
    int count,negative;
```

```
    float no,sqroot;
```

```
    clrscr();
```

```
    printf("Enter 9999 To Stop\n");
```

```
    count=0;
```

```
    negative=0;
```

```
    while(count<=20)
```

```
    {
```

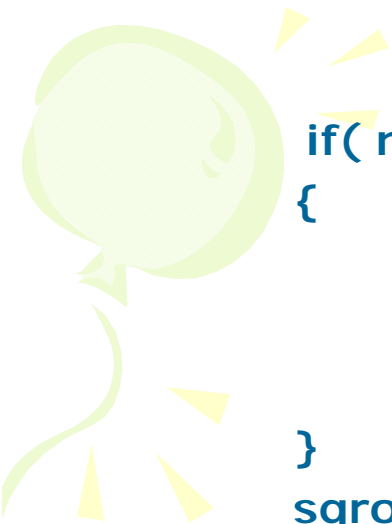
```
        printf("\nEnter a number :");
```

```
        scanf("%f",&no);
```

```
        if(no==9999)
```


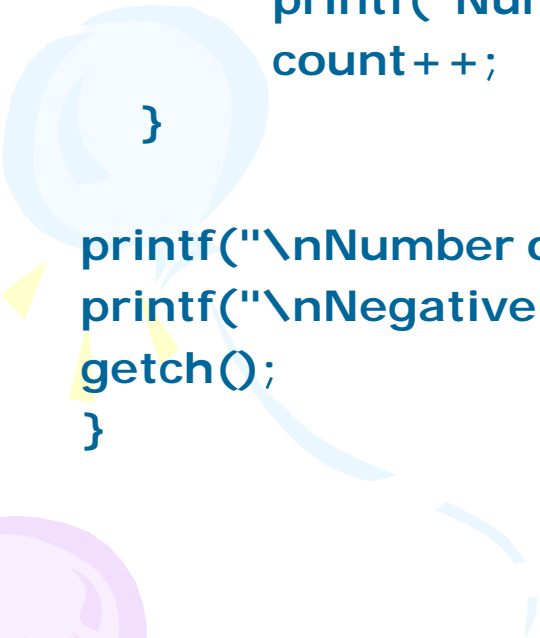
```
            break;
```

```
        //Break statement
```



```
if( no < 0 )
{
    printf("\nNumber is negative");
    negative+ +;
    continue;           // continue statement
}
sqroot = sqrt(no);
printf("Numer = %f\n Square root = %f\n\n",no,sqroot);
count+ +;
}

printf("\nNumber of item done = %d\n",count);
printf("\nNegative item = %d\n",negative);
getch();
}
```





Jumping out of the Program

We can jump out of a program by using the library function `exit()`. In case, due to some reason, we wish to break out of a program and return to the operating system, we can use the `exit()` function, as shown below:

```
.....
```

```
.....
```

```
if (test condition) exit(0);
```

```
.....
```

```
.....
```