



CONTROL STRUCTURES-2

AFTER THIS LESSON, LEARNERS WILL BE ABLE TO:

- Understand the concept and usage of selection and iteration statements.
- Know various types of control structures available in C++.
- Analyze the problem, decide and evaluate conditions.
- To analyze and choose an appropriate combination of constructs to solve a particular problem
- Write code that employ decision structures, including those that employ sequences of decision and nested decisions.
- Design simple applications having iterative nature.

FLOW OF CONTROL AND CONTROL STRUCTURES

- ❑ The order in which a set of statements are executed in a program is known as its **flow of control**
- ❑ Generally the flow of control is **sequential**.
- ❑ However at times it is required to change this sequential flow so that only a particular statement or group of statements are executed.
- ❑ This is achieved by **control structures**
- ❑ **Control structures are statements that upon execution alter or control the sequence of execution of statements in a program.**

TYPES OF CONTROL STRUCTURES

These can be categorized into the following categories:

Selection Control Statements

- if
- if.....else
- switch.....case

Iterative Control Structures

- for loop
- while loop
- do.....while loop

Jump Statements

- break
- continue
- go to

REPETITIVE(ITERATIVE) CONTROL STRUCTURES OR LOOP

- A loop is a control structure that causes a statement or a set of statements (or code) to be repeated again and again while a condition is true.
- The piece of code or set of statements is repeated a certain no. of times till the condition becomes false, following which control is transferred to the statements following the loops.
- In C++, there are three loop structures:-
 - For loop
 - While loop
 - Do loop (do..while loop)

PARTS OF A LOOP

- Each loop consists of 3 expressions, the **initialization expression**, the **test expression** and the **update expression**, and a **Loop Control Variable** (or expression)
- The value of the loop control variable determines the number of times a loop is executed.
- The three expressions are formed using the loop control variable.
- Examples of the expressions:

Initialization

- `int i=0;`
- `int x=2;`

Test

- `i < 9;`
- `x < 4;`

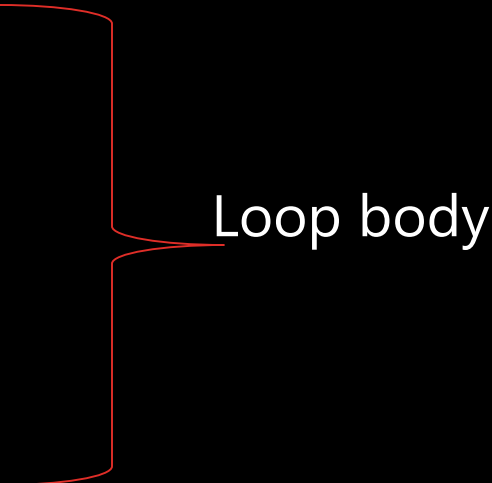
Update

- `i ++;`
- `x--;`

THE FOR LOOP

The for loop is the easiest to understand and is the most commonly used loop structure in C++. The structure of the loop is as follows:

```
for( initialization expression; test expression; update expression)
{
    Statement 1;
    Statement 2;
    ...
    Statement n;
}
```



A red bracket on the right side of the code block groups the statements from "Statement 1;" to "Statement n;" under the label "Loop body".

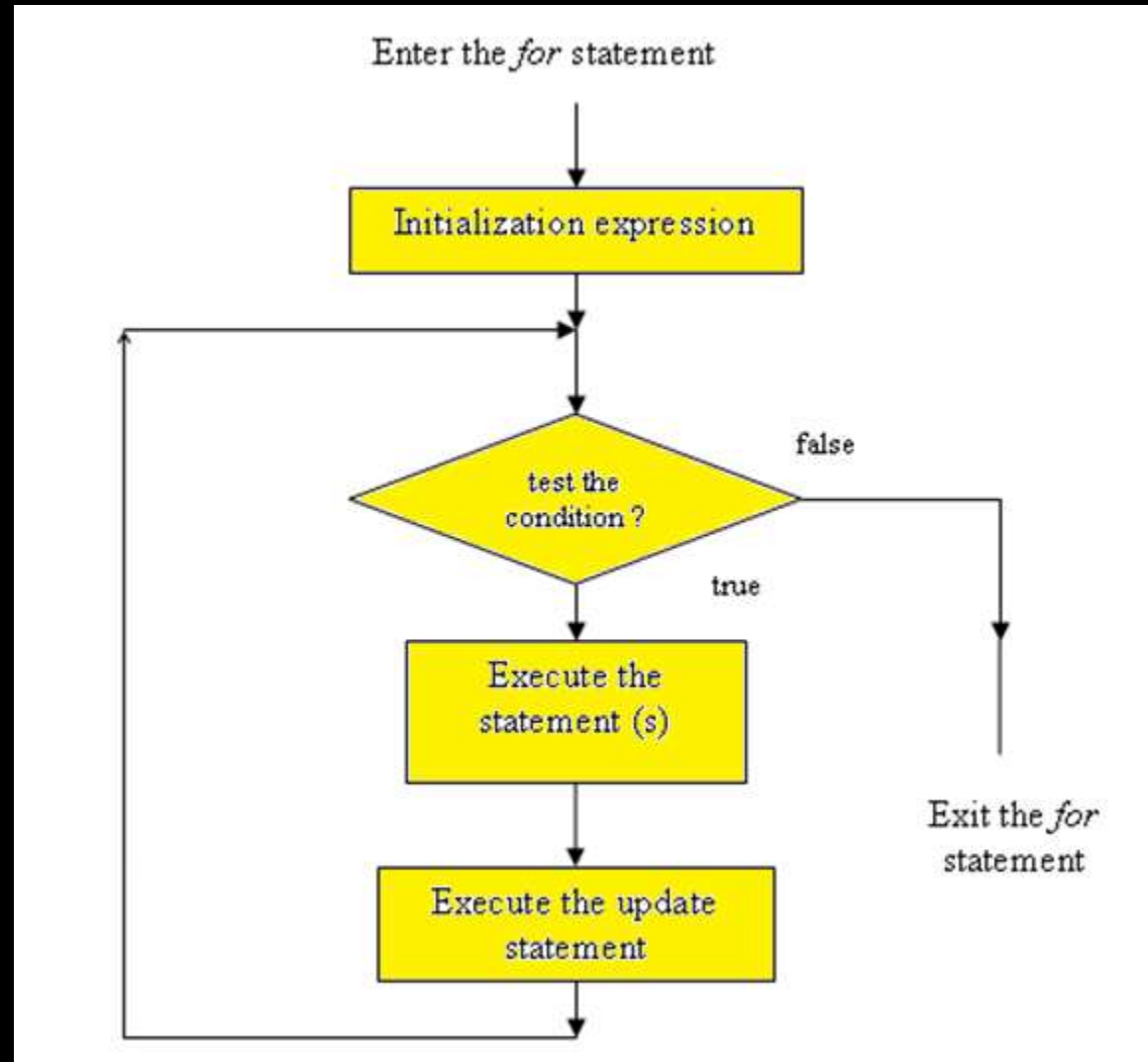
THE FOR LOOP – STRUCTURE DEFINED

for(**initialization** expression; **test** expression; **update** expression)

The three expressions usually involve the loop control variable.

- The initialization expression initializes the loop variable with some legal value.
- The test expression determines whether or not the loop is to be continued. It is actually the condition that is checked and depending upon its value (true or false) the loop is executed or terminated.
- The update expression is used to change the value of the loop variable for further iteration.

FLOWCHART : THE FOR LOOP



EXAMPLE : PROGRAM TO PRINT NUMBERS FROM 1 TO 5

```
#include <iostream.h>
```

```
void main()
```

```
{
```

Initialization
Expression

conditional
expression

update
expression

```
for ( int i=1      ; i<=5      ; i++)
```

```
cout<< i <<"\n";
```

← loop body

```
}
```

Output:

1
2
3
4
5

The loop control variable is *i* . The loop given above will execute five times. This can be verified by the number of times the value of *i* is displayed.

WORKING OF FOR LOOP: STEP 1

```
#include <iostream.h>
void main()
{
    1
    for ( int i=1      ;   i<=5      ;      i++)

    cout<< i <<"\n";


}
```

WORKING OF FOR LOOP: STEP 2

```
#include <iostream.h>
void main()
{
    for ( int i=1      ; i<=5      ;      i++)

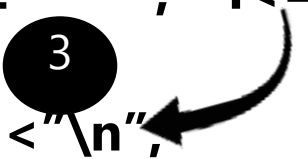
    cout<< i <<"\n";

}
```




WORKING OF FOR LOOP: STEP 3

```
#include <iostream.h>
void main()
{
    for ( int i=1      ; i<=5      ;      i++)
        cout<< i << " \n" ,
    }
}
```



WORKING OF FOR LOOP: STEP 4

```
#include <iostream.h>
void main()
{
    for ( int i=1      ; i<=5      ; i++)
        cout<< i <<"\n";
}
```




WORKING OF FOR LOOP: BACK TO STEP 2

```
#include <iostream.h>
void main()
{
    for ( int i=1      ; i<=5      ; i++)

    cout<< i <<"\n";

}
```



WORKING OF FOR LOOP : THE LOOP FORMATION

```
#include <iostream.h>
```

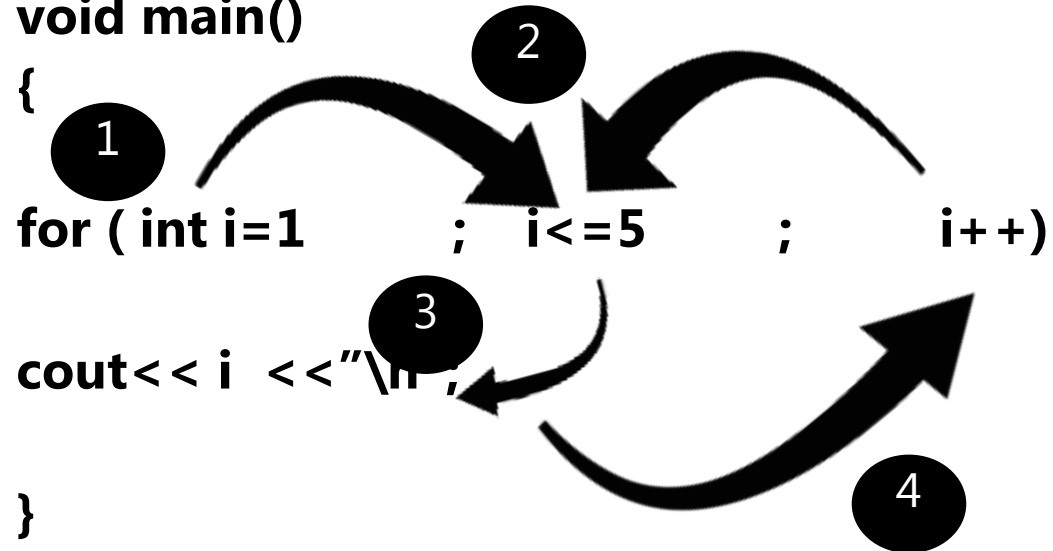
```
void main()
```

```
{
```

```
for ( int i=1      ; i<=5      ; i++)
```

```
cout<< i <<"\n";
```

```
}
```



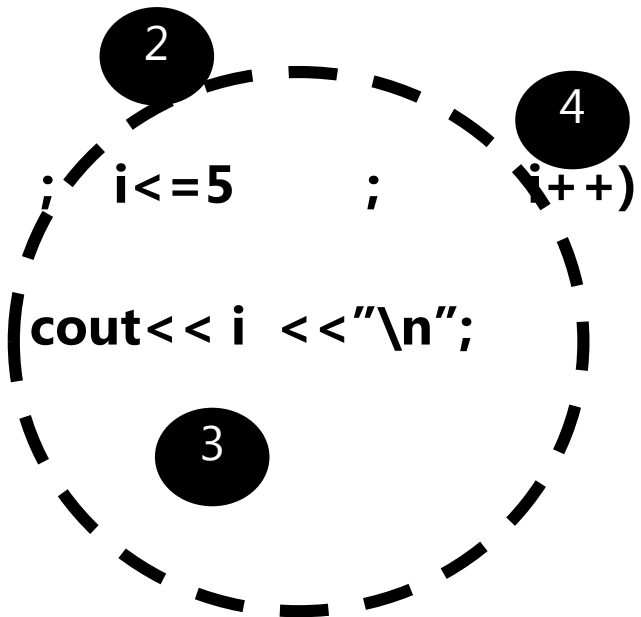
WORKING OF FOR LOOP : THE LOOP FORMATION

```
#include <iostream.h>
```

```
void main()
```

```
{
```

```
    1  
    for ( int i=1
```



```
        ; i<=5 ; i++)  
        {cout<<i <<"\n";
```

```
    }
```

Step 2 : Test
statement

Step 4 : Update
statement

Step 3 : Loop
body

EXPLANATION

In the above program, we use a for loop to print numbers from 1 to 5. Let us see how this works:

WORKING OF FOR LOOP

1. First, the variable `i` is initialized with value 1. This happens only once.
2. Next the value of the variable is checked. If it is less than or equal to 5, the control is passed to the body of the loop. If the condition is not satisfied, the control comes out of the loop.
3. The body of the loop which contains a `cout` statement is executed. The value of `i` is displayed along with a new line character.
4. Next, the value of `i` (loop variable) is incremented.
5. Now, control comes back to the step 2, where again the value of `i` is checked. If it is less than or equal to 5, the body of the loop (step 3) is executed, which prints the value of `i`. Now, step 4 is carried out, which increments `i` and so on.
6. The process continues till the value of `i` becomes 6. When `i` becomes 6 at step 4, control is transferred to step 2. Here the value of `i` is tested, the condition yields false and so the loop is terminated

MORE EXAMPLES TO DEMONSTRATE WORKING OF FOR LOOP

```
for ( int k = 6; x > 0; k -- )  
cout << k << "\t";
```

6 5 4 3 2 1

The variable k(loop control variable) is first initialized with 6. It is then tested for $k > 0$, then the body of the loop is executed which displays the value of k. Next the value of k is decremented. Thus numbers in reverse order from 6 to 1.

```
for ( int x = 1; x <= 9; x ++ )  
cout << 5*x << " ";
```

5 10 15 20 25 30 35 40 45

The above code fragment displays the first nine multiples of 5.

MORE EXAMPLES TO DEMONSTRATE WORKING OF FOR LOOP

```
for ( int i = 2; i <= 20; i += 2 )  
cout << i << " ";
```

4 6 8 10 12 14 16 18 20

This code displays all the even numbers from 2 to 20. Note the expression `i+=2`. This increments the value of `i` by 2

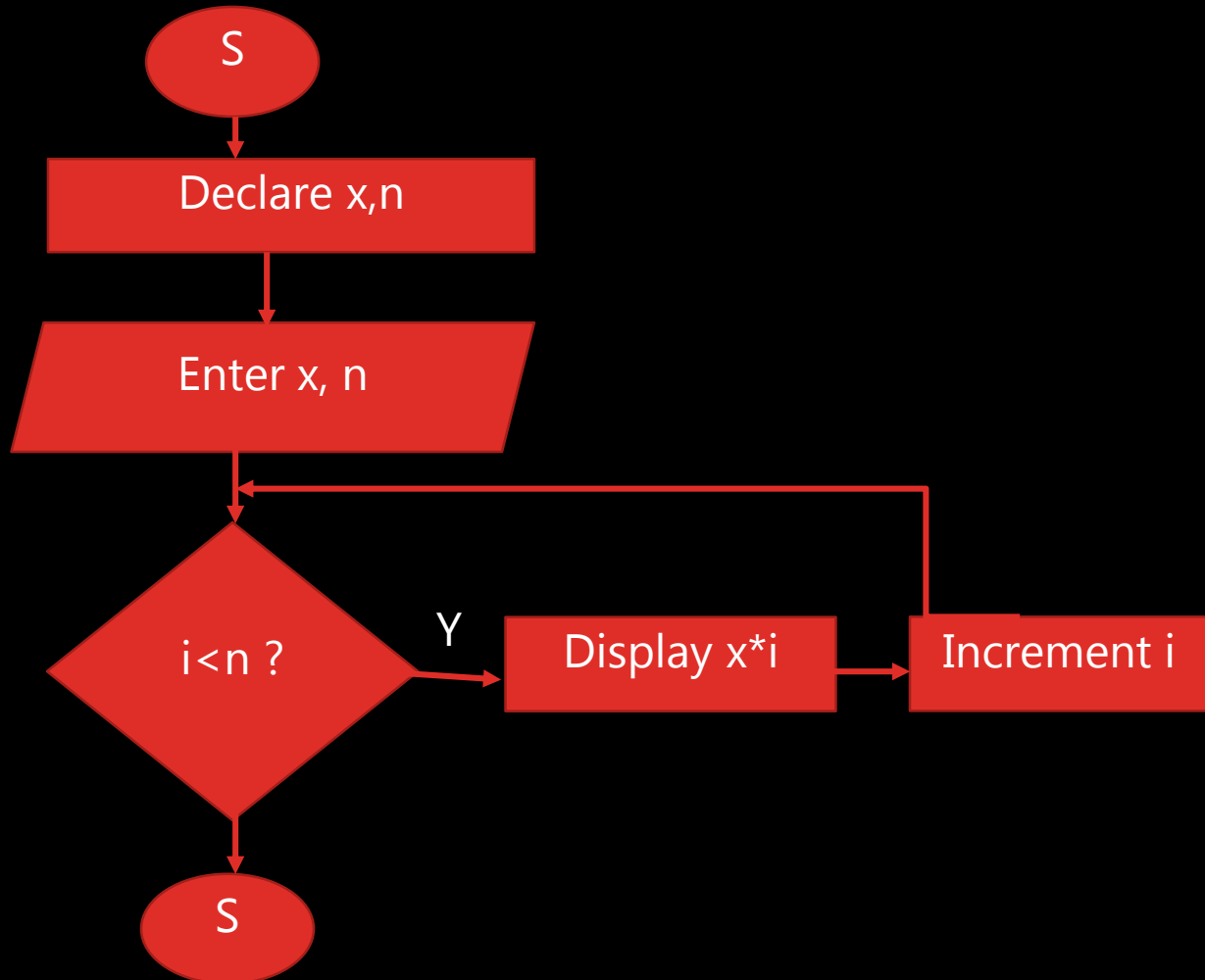
```
int k = 15;  
for ( int p = 1; p <= 5; p++ )  
{cout<<  
k<<"x"<<p<<"="<<k*p;cout<<
```

15 X 1 = 15
15 X 2 = 30
15 X 3 = 45
15 X 4 = 60
15 X 5 = 75

The above code fragment displays the first five multiples of 5.

EXAMPLE PROGRAM: WAP TO DISPLAY MULTIPLICATION TABLE OF A NUMBER

```
#include<iostream.h>
void main()
{int x, n;
cout<<"\nEnter the number ";
cin>>x;
cout<<"\nEnter the limit ";
cin>>n;
for(int i=1; i<=n; i++)
cout<<x<<" X " <<i<<" = "<<x* i<<"\n";
}
```



EXPLANATION

- Two variables are declared to store the number whose multiplication table is to be displayed and to store the limit.

```
int x, n : x for number, n for limit
```

- The for loop is formed with 1 as the initial value and n as the last.

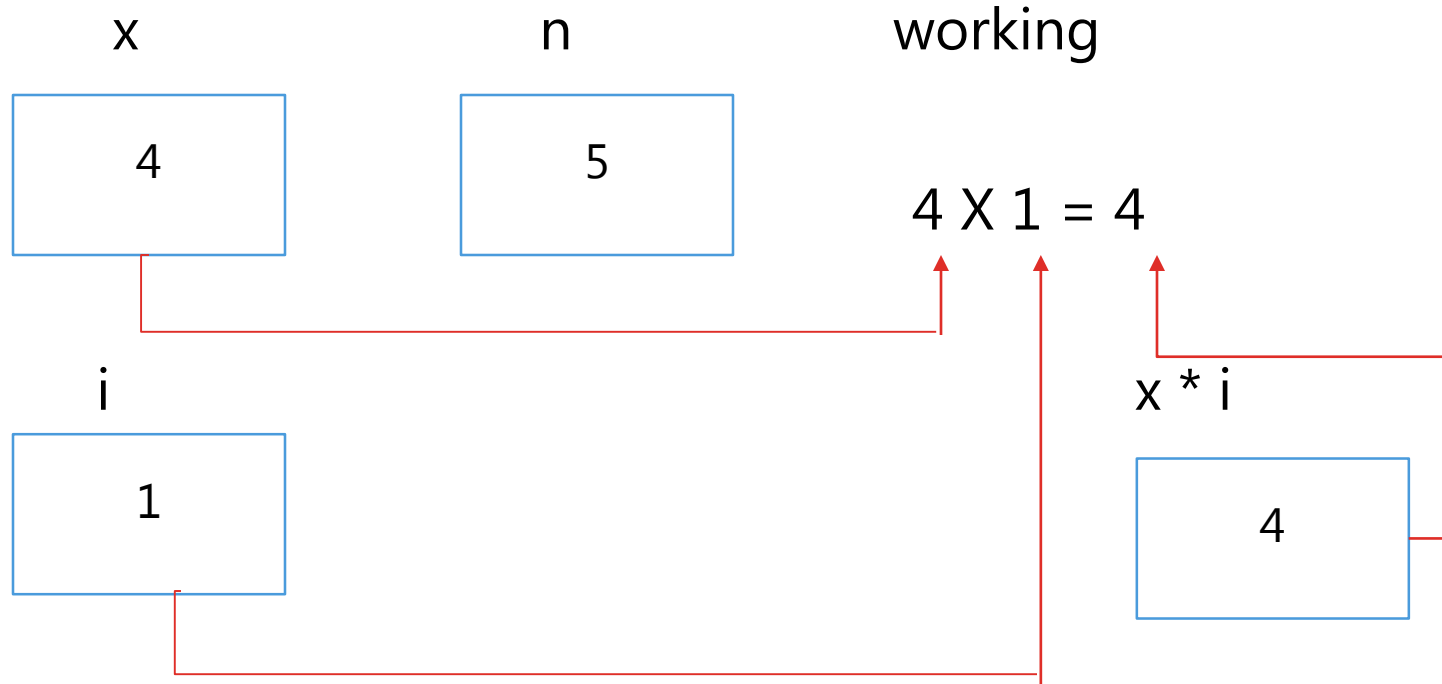
```
for(int i=1; i<=n; i++)
```

- The cout statement displays the number , x, then the multiplication symbol (X), the value of i (varies from 1 to n), the = sign and then the product of multiplication (x*i)

```
cout<<x<<" X "<<i<<" = "<<x* i<<"\n";
```


DRY RUN

memory diagram of variables

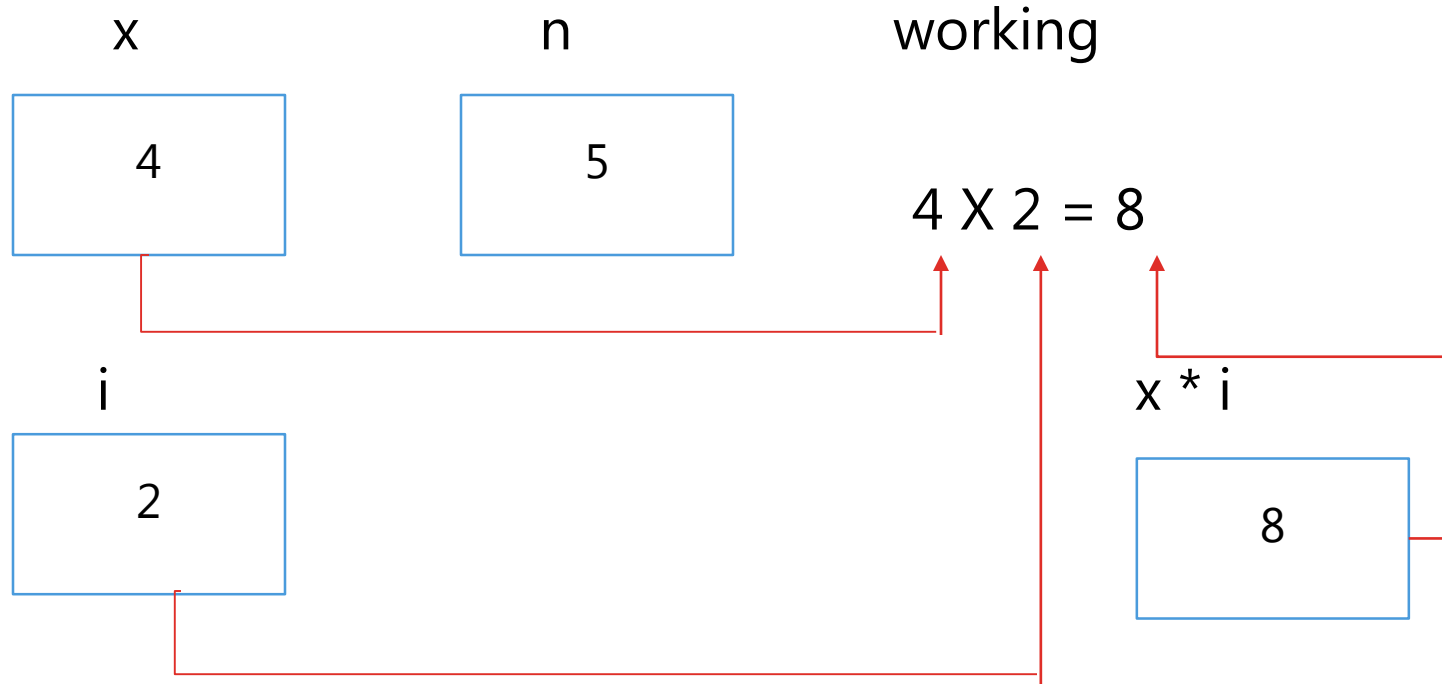


Output

$4 \times 1 = 4$

DRY RUN

memory diagram of variables



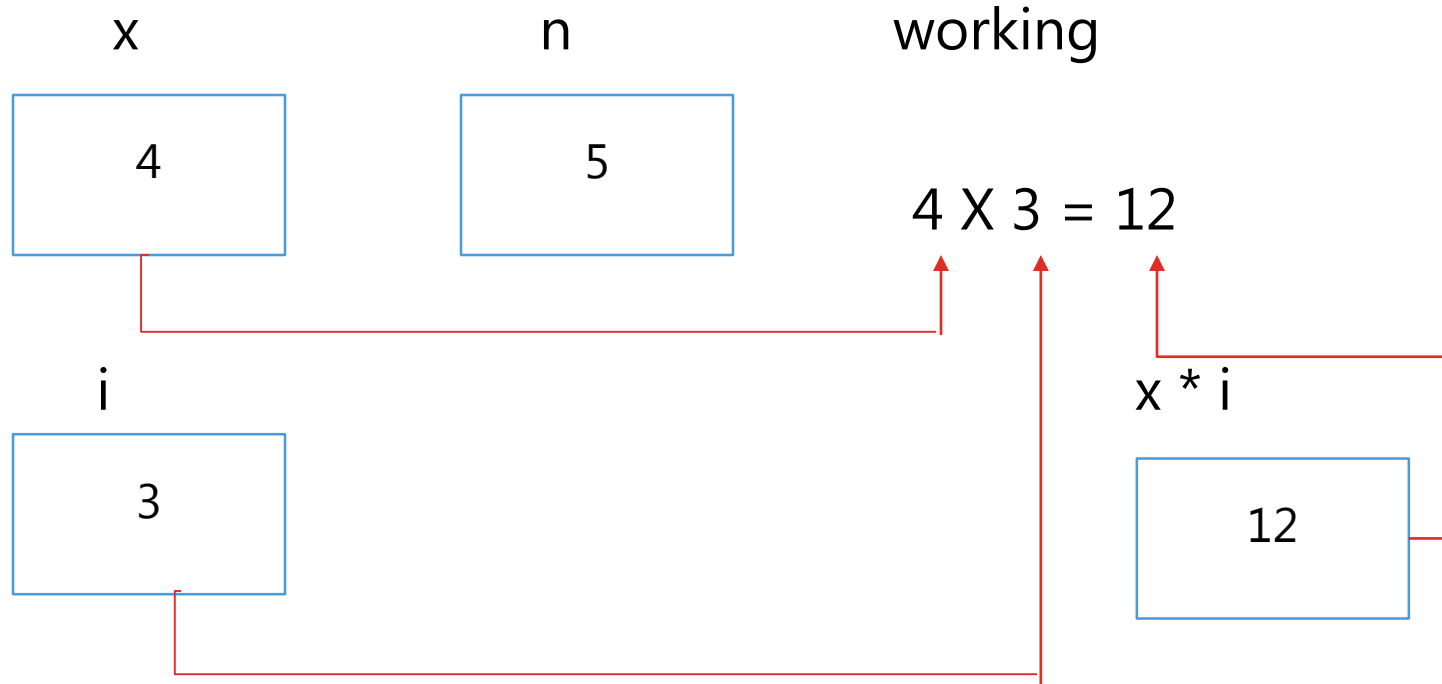
Output

$$4 \times 1 = 4$$

$$4 \times 2 = 8$$

DRY RUN

memory diagram of variables



Output

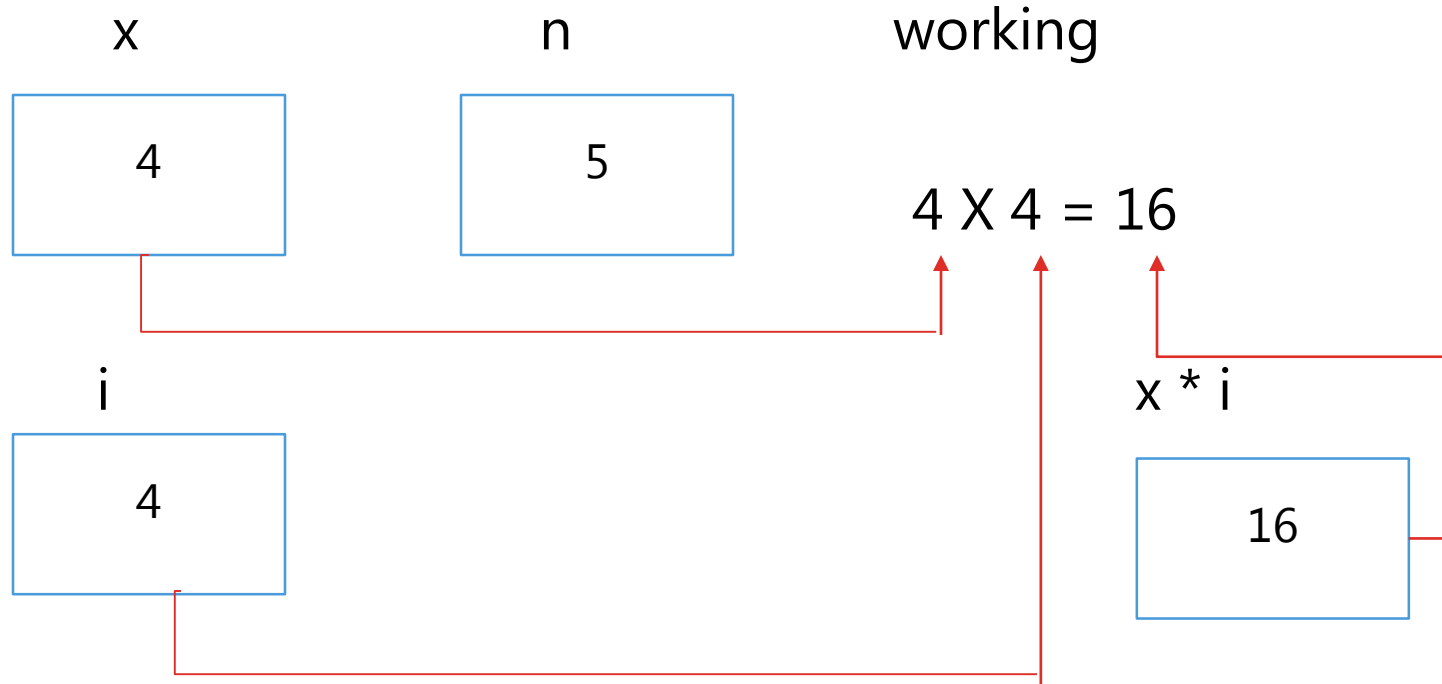
$$4 \times 1 = 4$$

$$4 \times 2 = 8$$

$$4 \times 3 = 12$$

DRY RUN

memory diagram of variables



Output

$$4 \times 1 = 4$$

$$4 \times 2 = 8$$

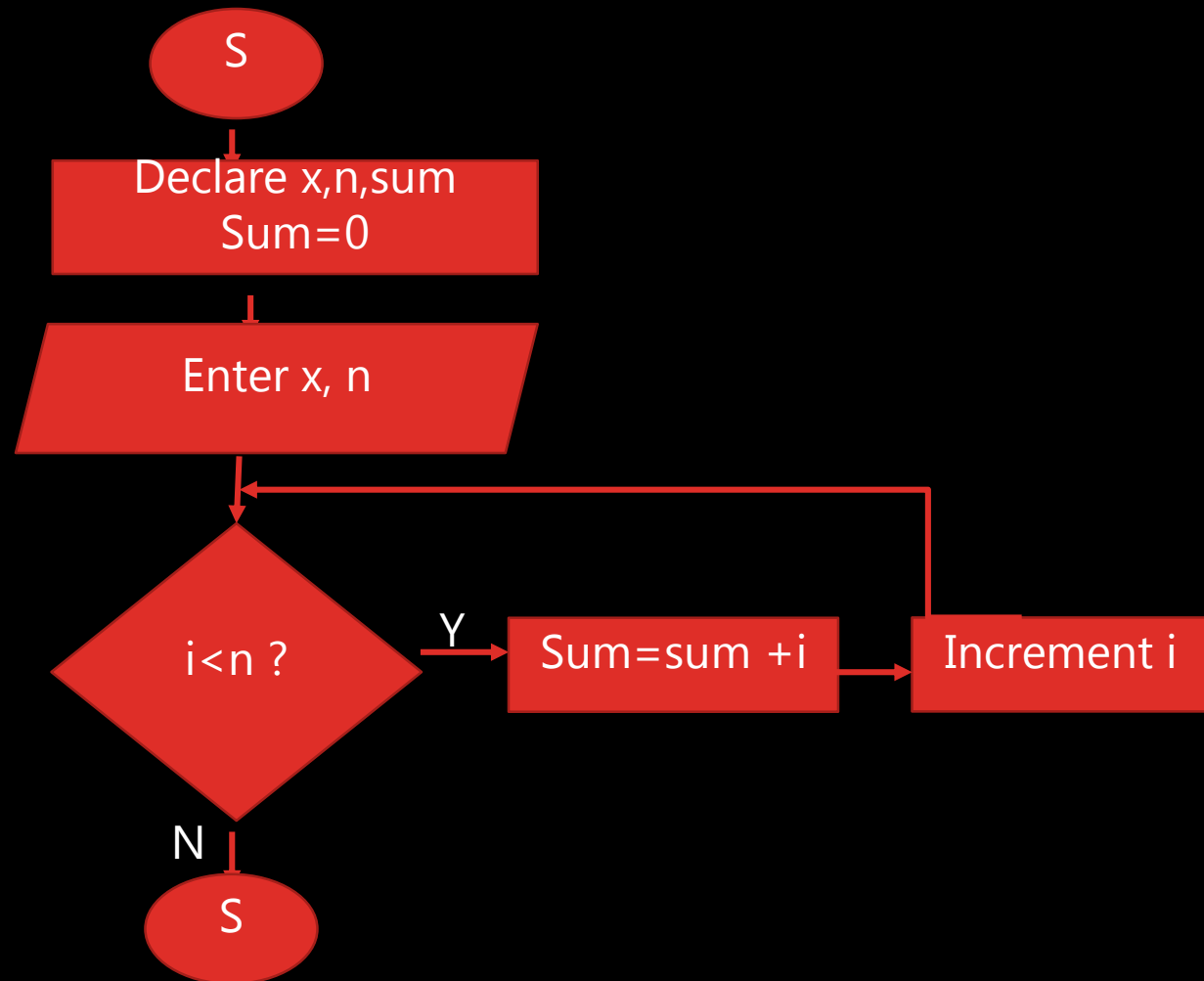
$$4 \times 3 = 12$$

$$4 \times 4 = 16$$

EXAMPLE PROGRAM: DISPLAY THE SUM OF THE SERIES $1+X+X^2+X^3+....X^N$

```
#include<iostream.h>
void main()
{int x, n, sum=0;
cout<<"\nenter the value of x ";
cin>>x;
cout<<"\n enter the value of n";
cin>>n;
for(int i=1; i<=n; i++)
sum+=pow(x,i);
cout << "\nsum of series is "<<1+sum;
}
```

Here `sum` is an accumulator variable. It accumulates and stores the numbers from 1 to the value of `n`



PROGRAM TO CALCULATE FACTORIAL OF A NUMBER

```
#include<iostream.h>
void main()
{int x, fact=1;
cout<<"\nEnter the number :";
cin>>x;
for(int i=x; i>0; i- -)
fact= fact * i;;
cout << "\nFactorial is:" << fact;
"<<1+sum;
}
```

Output

Enter the number : 4
Factorial is:24

VARIATIONS IN FOR LOOP

Variation	Loop example	Output	Explanation
Multiple initialization, update, test expressions	for(int i=0, j=2 ; i< j , i++ , j- -) cout<<i*j<<"\ n";	0 1	There are two initializations and updates here. The loop runs till i<j. when I becomes 2 and j becomes 0 loop breaks.
Placement of Expressions	<pre>int i=0; for(; i<5;) {cout<<i++; cout<<" "; }</pre> <div>initialization</div> <div>update</div>	0 1 2 3 4	The initialization statement is above the loop and update inside the loop. This is allowed but we must write the semicolon in the respective places.
Empty loop	for(int x=0; x<9;x++) cout<<x<<" ";	9	This loop will execute till the value of x becomes 9. The semicolon prevents control from entering loop body. Thus the cout is executed only after loop terminates which displays the final value of x(9).
Infinite loop	int a = 0; for(; ;) cout<< a ++<<" " ;	0 1 2 3 4	The above for loop does not have a test expression, therefore the value of a keeps on incrementing which is displayed after each pass . As there is no test expression the loop never breaks.

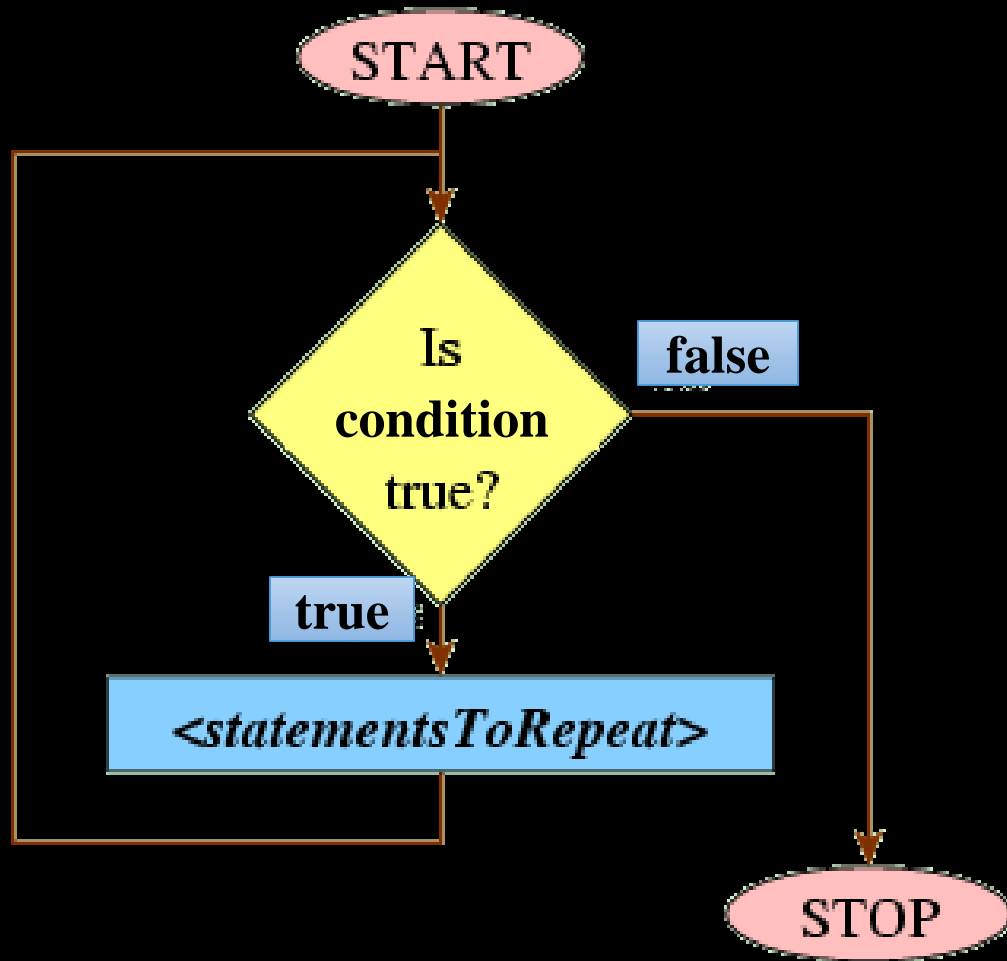
WHILE LOOP

- ❑ While loop is usually used when we do not know how many times we need to execute the loop.
- ❑ It repeats a statement or a set of statements on the basis of a condition.
- ❑ If the condition is true, the statements in the loop are repeated again.
- ❑ If the condition is false, the loop is terminated.
- ❑ Unlike the for loop the syntax of while loop does not require to place the initialization, test and update expressions in any order.
- ❑ The syntax requires a condition or test expression as a part of it. The other two statements can be placed according to logic.

SYNTAX OF WHILE LOOP

while having one statement in loop body	while having multiple statements in loop body
<pre>while(condition) Statement;</pre>	<pre>while (condition) { statement block; }</pre>

FLOWCHART: WHILE LOOP



WHILE LOOP DETAILED SYNTAX

Test expression/ condition

```
while ( i<9 )  
{  
    statement 1;  
    statement 2;  
    ....  
    ....  
}
```

No semicolon

Loop body

No semicolon

❑ The test expression is any valid conditional statement formed by using variables, constants and operators including arithmetic and relational operators.

❑ We can also combine two conditions using logical operators.

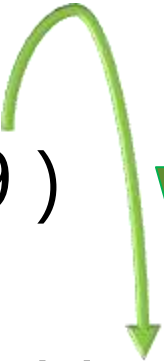
❑ There can be single or multiple statements in Loop body.

❑ These can be any valid c++ statement.

WORKING OF WHILE LOOP

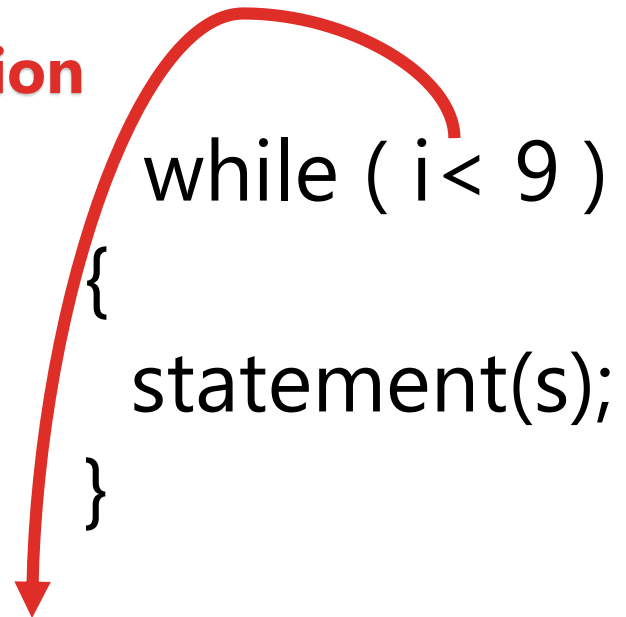
```
while ( i < 9 )  
{  
    statement(s);  
}
```

when condition is true



WORKING OF WHILE LOOP

**When condition
is false**



WHILE LOOP EXAMPLE

- In order to be able to run the loop properly we need the initialization as well as an update statement also.
- So we place the initialization statement **before** the loop and the update statement **inside** the loop.

Initialization expression

```
int i= 2,  
while( i< 9 )  
{  
    cout<< i<<"\n";  
    i++;
```

update expression

output

2
3
4
5
6
7
8

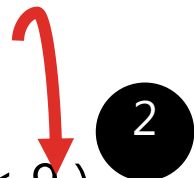
WORKING OF WHILE LOOP

```
1 int i= 2;  
  
while( i< 9 )  
{  
    cout<< i<<"\n";  
  
    i++;  
  
}
```

Initialization

WORKING OF WHILE LOOP

```
int i= 2;  
while( i< 9 )  
{  
    cout<< i<<"\n";  
  
    i++;  
}
```

A red curved arrow points from the value '2' in the initialization 'int i= 2;' to the variable 'i' in the while loop condition 'while(i< 9)'. A black circle containing the number '2' is positioned next to the 'i' in the condition, indicating the current value of the loop variable.

Test Expression

WORKING OF WHILE LOOP

```
int i= 2;
```

```
while( i < 9 )
```

```
{
```

```
    cout << i << "\n";
```

```
    i++;
```

```
}
```

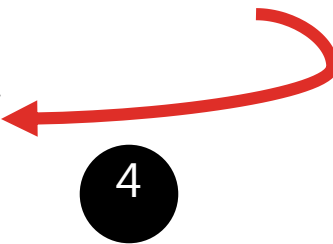


3

Loop body

WORKING OF WHILE LOOP


```
int i= 2;  
  
while( i< 9 )  
{  
    cout<< i<<"\n";  
    i++;  
}
```



Update

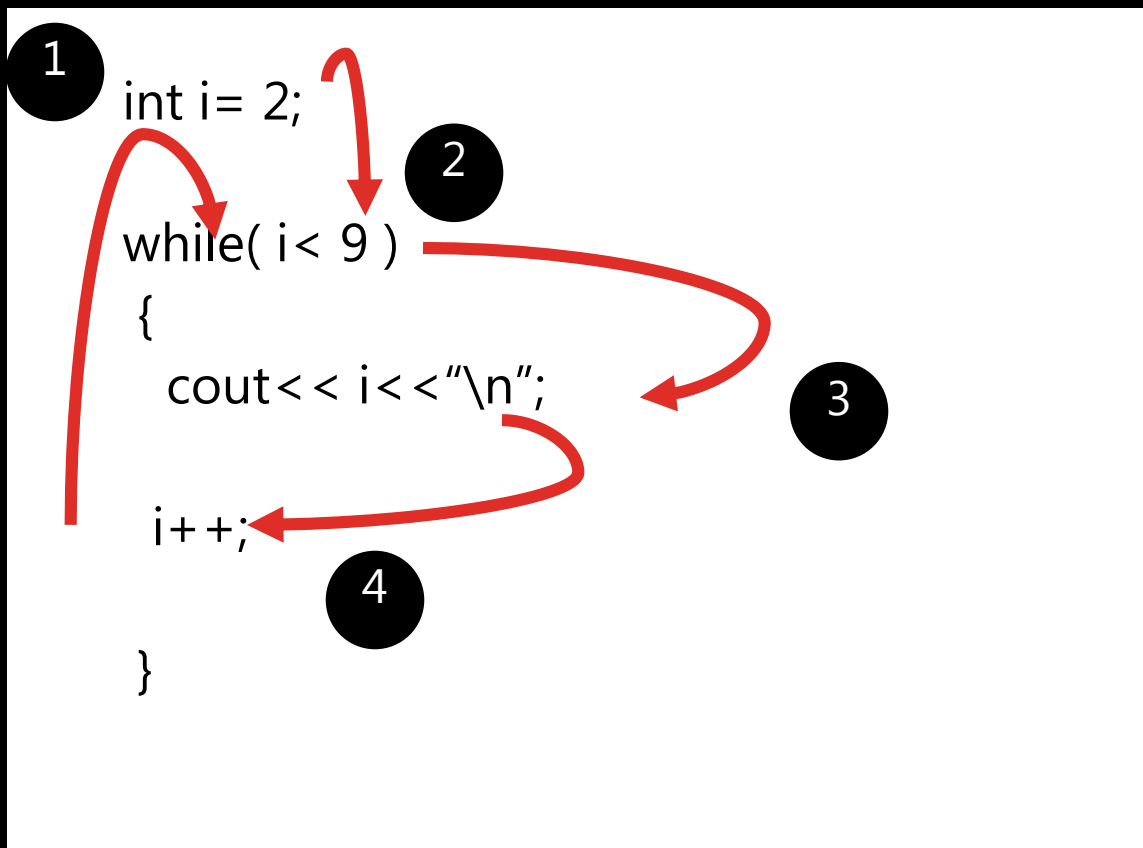
WORKING OF WHILE LOOP

```
int i= 2;  
while( i< 9 )  
{  
    cout<< i<<"\n";  
  
    i++;  
  
}
```



Control transferred
to test expression

WORKING OF WHILE LOOP



1. Loop control variable(*i*) is initialized
2. The test expression is evaluated
3. Loop body is executed if it is true
4. Loop control variable is updated and control is transferred back to test expression.

If test expression evaluates to true loop is executed otherwise control is transferred to statement following the loop

SOME POINTS TO NOTE

- While loop is usually used when we do not know how many times we need to execute it. Such loops are known as sentinel controlled loops and do not require a counter.
- The loop control variable acquires its value from user input and not by assignment/initialization.
- The updating takes place within the loop and depends on user input .
- The number of times the loop executes also depends on the user input

A PROGRAM TO COUNT THE NUMBER OF DIGITS IN A NUMBER

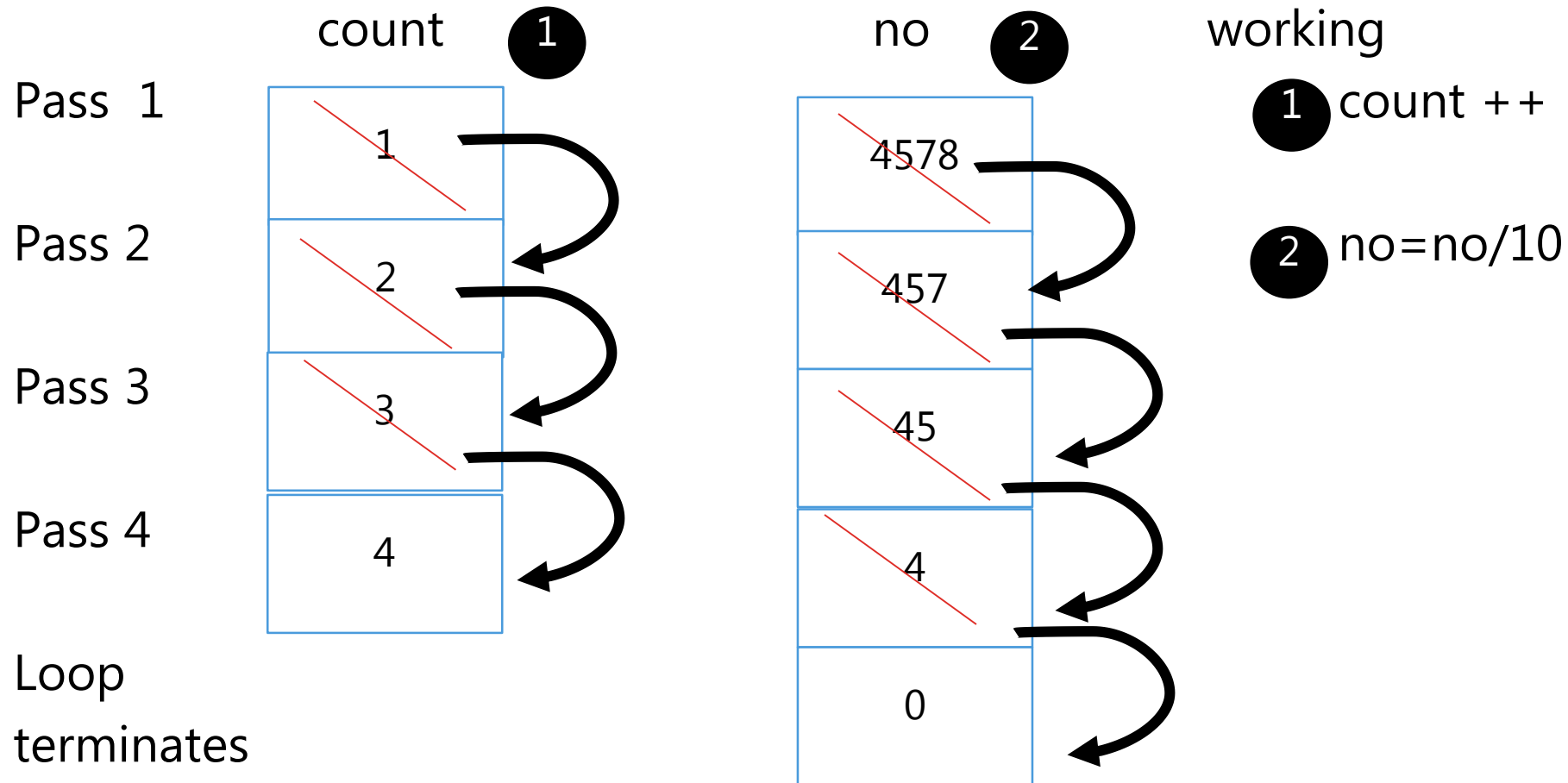
```
#include <iostream.h>
void main()
{ int no, count=0;
cout<<"\n Enter the number :";
cin>>no;      // user input : loop control variable (LCV)
while(no!=0)   // condition is formed using the LCV
{ count ++;    // counter
  no/=10;      // the update statement , changing value of LCV
}
cout<<"\n Number of digits is:"<<count;
}
```

Output

```
Enter the number :
4578
Number of digits is : 4
```


DRY RUN

memory diagram of variables



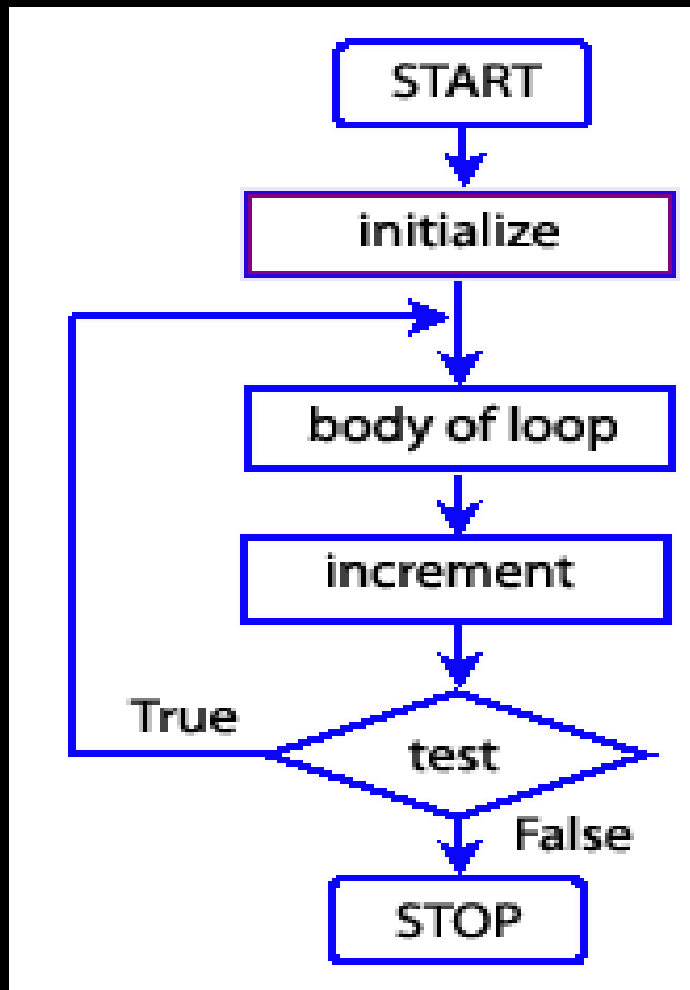
WORKING

Pass	no	Condition (no!=0)	Count ++	Update (no/=10)	Output
1	4568	4568 != 0 True	1	no=4568/10 → no=456	
2	456	456 != 0 True	2	no=456/10 →no=45	
3	45	45 != 0 True	3	no=45/10 →no=4	
4	4	4 != 0 True	4	no=4/10 N0	
0 != 0 False → loop terminates					Number of digits is : 4

DO WHILE LOOP

- The do.... while loop is a control structure in which the loop is first executed and then the test expression is evaluated.
- This loop is also known as an exit controlled loop as the test expression/condition is checked at the end of the loop i.e. after the loop body is executed.
- This ensures that the loop executes at least once even if the condition is false.

SYNTAX AND FLOWCHART OF DO WHILE LOOP



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

WORKING OF DO WHILE LOOP

```
int x = 0;
```

**initialization of loop
control variable**

```
do
```

```
{
```

```
    cout << x++;
```

```
    cout << "\n";
```

```
}while (x < 6);
```

loop body

Loop condition

- ❑ The loop control variable is x, which is initialized to 0.
- ❑ The loop body is then executed which displays and then increments the value of x.
- ❑ After this the loop condition is tested.
- ❑ If the condition is false, the loop is terminated, otherwise loop continues.
- ❑ When the value of x becomes 6, the condition becomes false and the loop is terminated

WORKING OF DO WHILE LOOP

In a do loop the loop will execute at least once even if the condition is false.

```
int x = 8;  
do  
{  
    cout << x++;  
    cout << "\n";  
}while(x < 8);
```

Output
8

- ❑ The value of x is 8 which contradicts the condition ($x < 8$).
- ❑ But since there is no check the loop body is executed, which displays the value of x and increments it.
- ❑ After the increment x becomes 9; the condition is checked in the end which is false and the loop is terminated.
- ❑ Thus we can see that the loop executes at least once even if the condition is false.

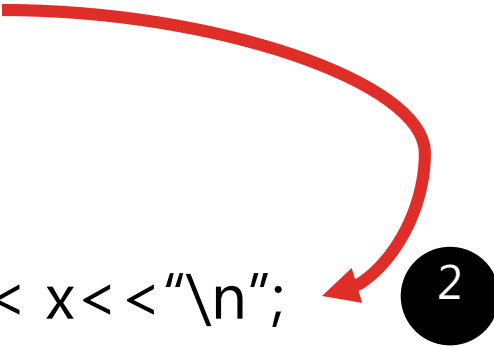
DO LOOP: WORKING

```
int x = 8; ①  
do  
{  
    cout << x<<"\n";  
  
    x++;  
  
} while(x < 8);
```

Initialization

WORKING OF DO LOOP

```
int x = 8;  
do  
{  
    cout << x<<"\n";  
  
    x++;  
} while(x < 8);
```

A red curved arrow originates from the opening curly brace of the do loop and points to a black circle containing the number 2, which is positioned next to the first line of the loop body (cout << x<<"\n";). This indicates the start of the loop body's execution.

2

Loop Body

WORKING OF DO LOOP


```
int x = 8;  
do  
{  
    cout << x<<"\n";  
    x++;  
} while(x < 8);
```

3

Update

WORKING OF DO LOOP

```
int x = 8;  
do  
{  
    cout << x<<"\n";  
  
    x++;  
} while(x < 8);
```




4

Test Expression

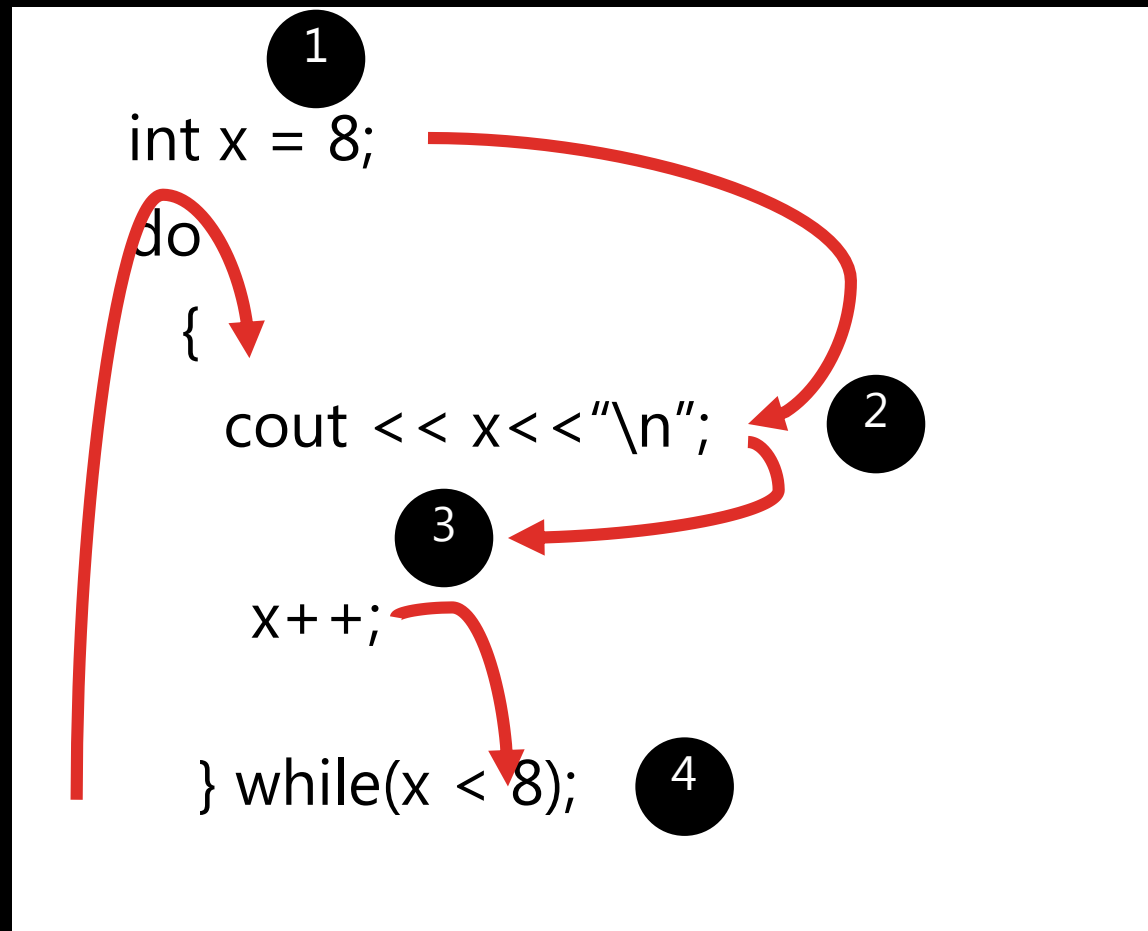
WORKING OF DO LOOP

```
int x = 8;  
do  
{  
    cout << x<<"\n";  
  
    x++;  
} while(x < 8);
```



Control transferred
to loop body if condition
is true

WORKING OF DO LOOP



1. Loop control variable(x) is initialized
2. Loop body is executed
3. Loop control variable is updated
4. The test expression is evaluated

If test expression evaluates to true loop is executed otherwise control is transferred to statement following the loop

PROGRAM TO CHECK WHETHER A NUMBER IS A PALINDROME

```
#include <iostream.h>
void main()
{
    long int number, d, numrev, storednum;
    numrev=0;                                // declare a variable to accumulate reversed number
    cout<<"Enter the number";
    cin>>number;                             // Enter number to check (LCV)
    storednum=number;                         // store a copy of the original number as the value will change in the
    loop
    do
    {
        d= number % 2;                       // extract the last digit
        numrev=numrev*10 + d;                 // accumulate the digit in reverse
        number=number/10;                     // shorten the number (update LCV)
    }while(number !=0);
    if(numrev==storednum)                     // numrev will contain the reversed number, compare with
        cout<<"\n It is a Palindrome";       // original number, if they are equal, it is a palindrome
    else
        cout<<"\n It is not a Palindrome";
    }
```

SAMPLE RUNS OF THE PROGRAM

Sample run 1 :

Enter the number : 99099

It is a palindrome

Sample run 2:

Enter the number : 1818

It is not a palindrome

ENTRY CONTROLLED AND EXIT CONTROLLED LOOPS

The for and while loop are entry controlled loops whereas the do while is an exit controlled loop.

Entry Controlled Loop	Exit Controlled Loop
Condition is checked before the loop body.	Condition is checked after the loop body.
If the condition is false, loop will not be executed at all	If the condition is false, even then the loop is executed at least once
<code>while(condition)</code> ← Loop Entry point Loop body	do loop body <code>while(condition);</code> ← Loop Exit point

EXAMPLE OF ENTRY CONTROLLED LOOP VS EXIT CONTROLLED LOOP

While loop	Do while loop
<pre>int i = 15; while (i < 6) { cout<< i--<<"\n"; }</pre>	<pre>int i = 15; do { cout<< i--<<"\n"; } while (i < 6);</pre>
Output	Output
No output	15
Explanation	Explanation
The loop body will not be entered(executed) since condition is false and checked before the body.	The loop body will be entered(executed) and then the condition will be checked. Since it is false, loop will terminate.
loop will not be executed even once as the condition is false	loop will be executed at least once even if the condition is false

WHICH LOOP TO USE WHEN

- All the loops are interchangeable. We can replace any loop with another. But there are some situations which can determine the choice:
 - ❑ We use the for loop when we know exactly how any times the statements are to be repeated.
 - ❑ If we do not know, exactly how any times the statements are to be repeated then we use while loop.
 - ❑ the for loop is used when we do not know exactly how any times the statements are to be repeated but you need to execute the loop at least once.

JUMP STATEMENTS

- ❑ C++ provides us with some jump statements that forcibly transfer flow of control according to a given condition.
- ❑ There are three jump statements and a function which we shall study
 - break
 - continue
 - goto
 - exit()

THE BREAK STATEMENT

- ❑ The break statement causes a loop to exit ie, it terminates the loop as soon as it is encountered. The control is transferred outside the loop.
- ❑ The break statement can be used with any of the loops- for loop, while loop or do..... while loop.

❑ Syntax:

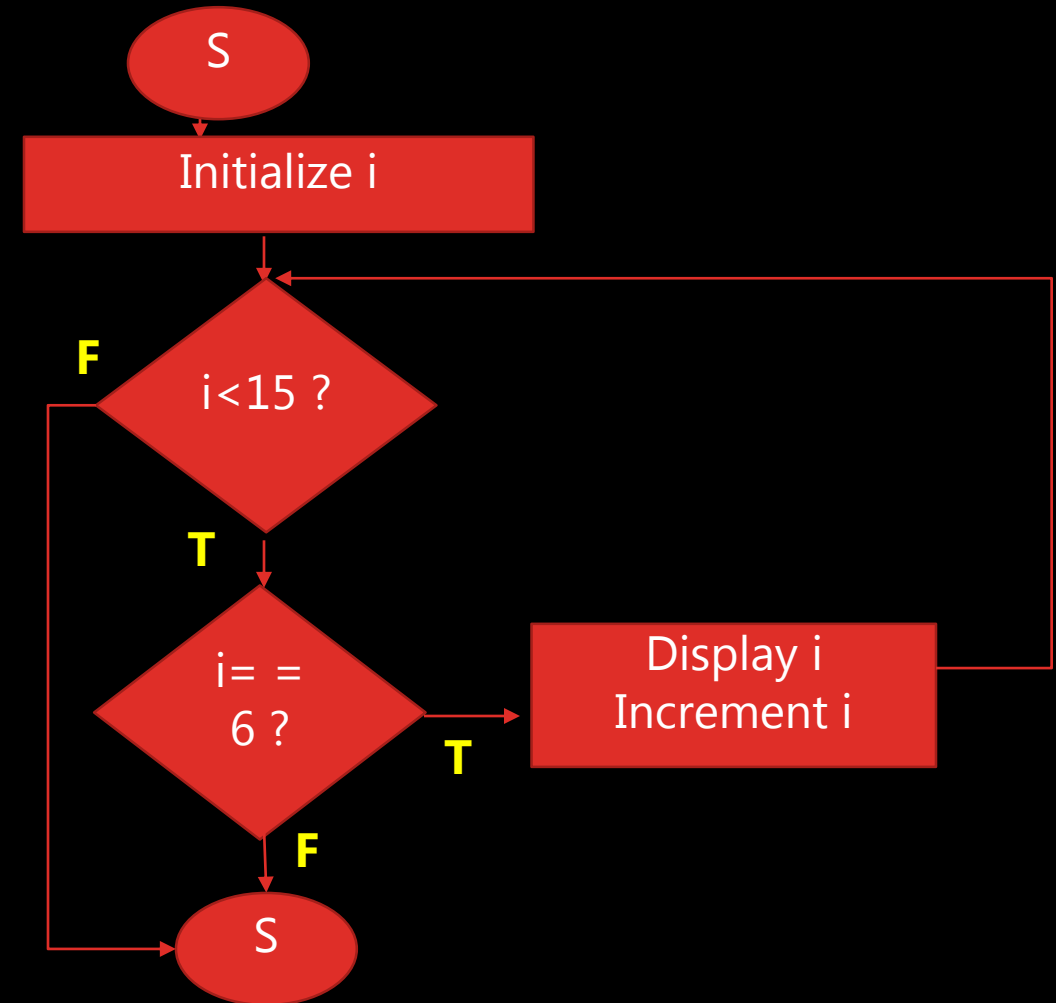
```
break;
```

❑ Usage :

```
for(int i=0; i<15; i++)  
{ if ( i % 11 ==0)  
    break;  
  cout<<i<<"\n";  
}
```

EXAMPLE AND FLOWCHART

Code	Output & Explanation
<pre>int i = 0; while(i < 15) { if(i == 6) break; cout << i << "\n"; i = i + 2; } cout << "outside of loop"; }</pre>	<p>Output</p> <p>0 2 4 Outside of loop</p> <p>Explanation</p> <p>The loop runs only till the values of i becomes 6. When i is 6 the loop terminates and control is transferred to the statement outside the loop.</p>



THE CONTINUE STATEMENT

- ❑ The continue statement is used to skip the current iteration (or pass) of a loop.
- ❑ This statement is valid only in loops.
- ❑ When it is encountered it skips the rest of the statements in the loop body and causes the next iteration of the loop.

❑ Syntax:

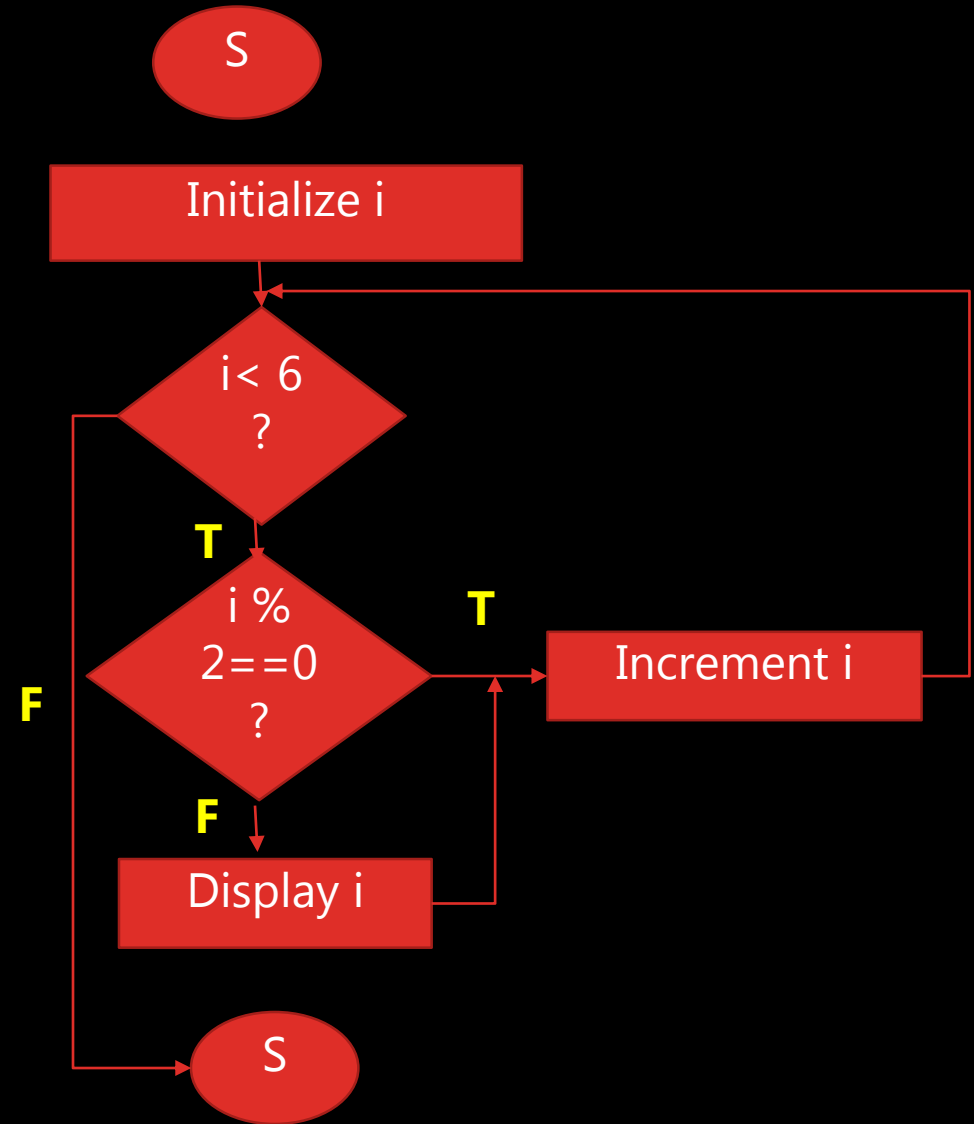
`continue;`

❑ Usage

```
for(int i=0; i<15; i++)  
    { if ( i % 11 ==0)  
        continue;  
        cout<<i<<"\n";  
    }
```

EXAMPLE AND FLOWCHART

code	Output and explanation
<pre>int i=1; while(i<6) { if(i%2==0) continue; cout<<i++<<"\n"; } cout<<"\n out of loop";</pre>	<p>1 3 5 out of loop</p> <p>Explanation The loop will not display any even numbers as whenever the value of i is even, the continue is executed and control is transferred to the update statement.</p>



THE GOTO STATEMENT

- ❑ The goto statement is used to transfer control from one part of the program to some other part.
- ❑ The syntax of goto is:
 goto label;
- ❑ Where label is any valid C++ identifier which is used to label the part of the program where the control is to be transferred

EXAMPLE AND EXPLANATION

```
#include<iostream.h>
void main()
{
    int n = 6;
    int k;
    cout<< " Enter a number";
    cin>> k;
    if(n > k)
        goto path1;
    else
        cout<<n<<"is not more than"<< k;
path1:
    cout<<"The larger number is"<< n;
}
```

Output

Enter a number 3
The larger number is 6

- ❑ We can see from the code that the goto statement simply transfer control to some other part of the program.
- ❑ Here n is 6.
- ❑ Suppose we enter 3 into k, then the if condition (n > k) becomes true and the goto is executed which is goto path1
- ❑ Control is transferred to the statement labelled path1 where cout is executed

WHY WE SHOULD NOT USE GOTO?

- It is best to avoid using goto as it leads to the program complexity. Programs having goto are hard to debug and follow.
- It is bad programming practice to use goto as it may lead to system failure.