INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

EEC-101
Programming with C++

Module-2.6 Selection Statements and Looping

Ramanuja Panigrahi

Topics

- Concepts of algorithm & flow charts;
- Input/output,
- constants, variables
- operators;
- Naming conventions and styles;
- Conditions and selection statements;
- Looping and control structures (while, for, do-while, break and continue);
- File I/O, Header files, String processing;
- Pre-processor directives such as #include, #define, #ifdef, #ifndef;
- Compiling and linking.

Flow of Control

- Modifying the order of execution
- Four constructs:
    - sequencing
    - selection
    - looping
    - calling sub-programs

- Selection (or branching)
    - To make a choice between alternatives (two or more).
    - Conditions for selection are made up of logical expressions – expressions that evaluate to true or false.

Three types of program controls

1. Sequence control structure.
2. Selection structures such as if, if-else, nested if, if-if-else, if-else-if and switch-case-break.
3. Repetition such as for, while and do-while.

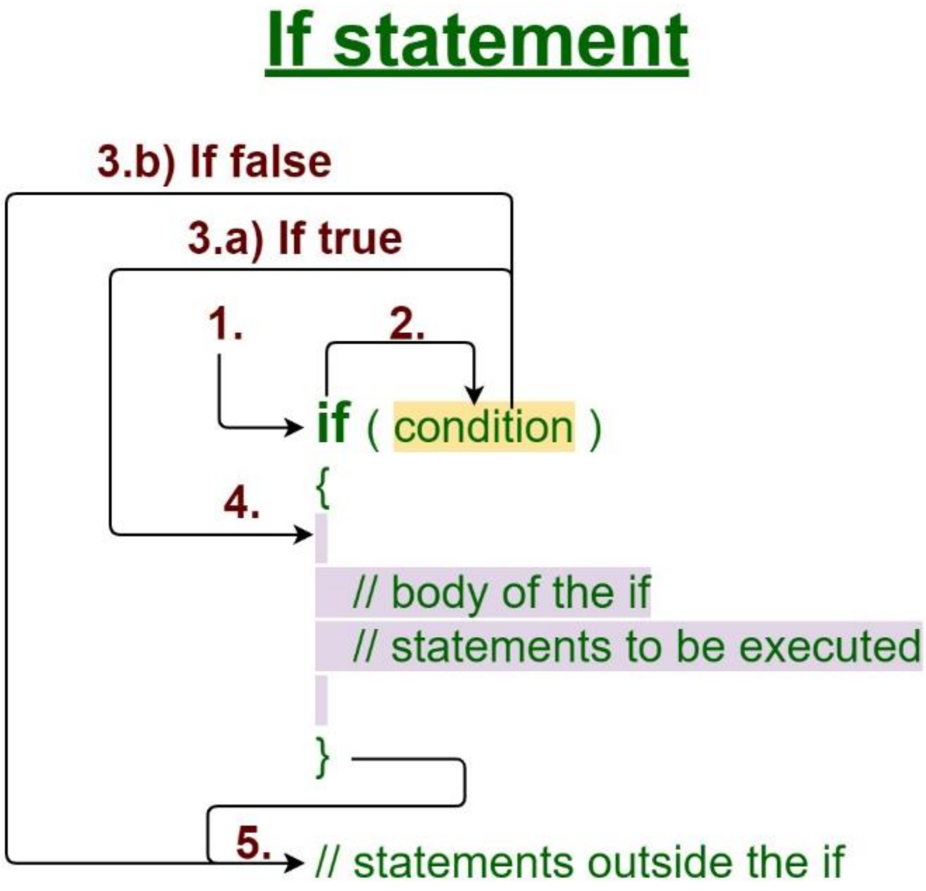
Decision Making in C++

- The **conditional statements** (also known as decision control structures) such as if, if else, switch, etc. are used for decision-making purposes in C programs.
- There come situations in real life when we need to make some decisions and based on these decisions, we decide what should we do next.
- Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code.

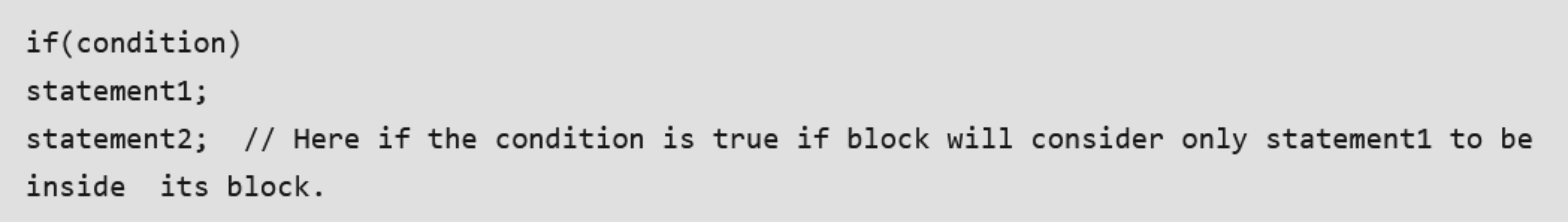
C++ if Statement

- The C++ **if statement** is the most simple decision-making statement.

- It is used to decide whether a certain statement or block of statements will be executed or not executed based on a certain type of condition.



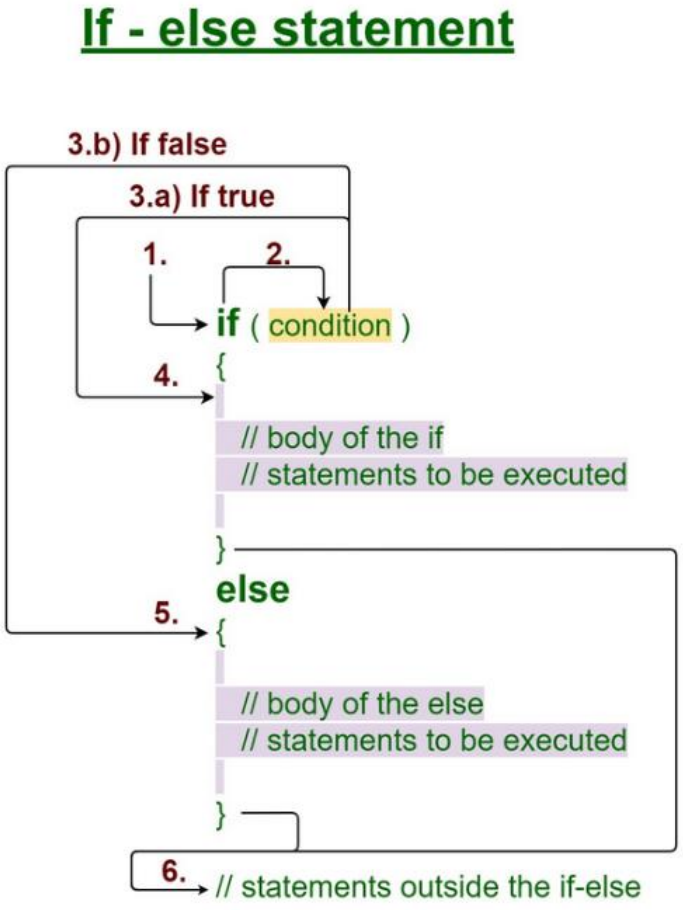
Syntax



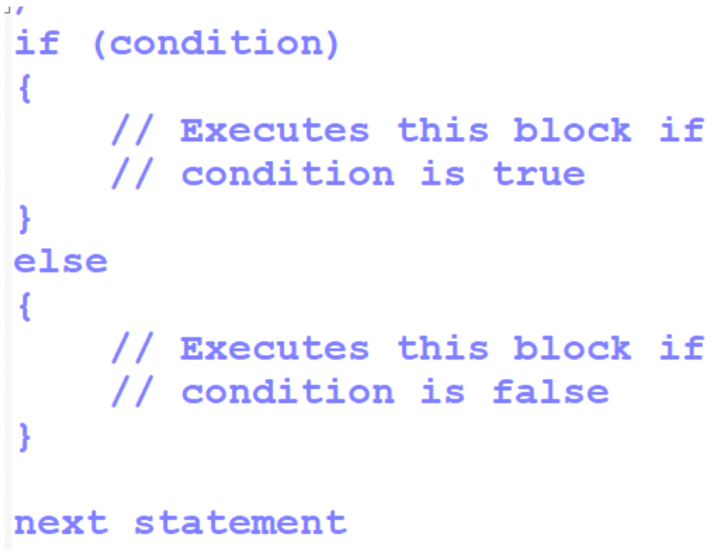
```
if(condition)
statement1;
statement2;  // Here if the condition is true if block will consider only statement1 to be
inside  its block.
```

C++ if else Statement

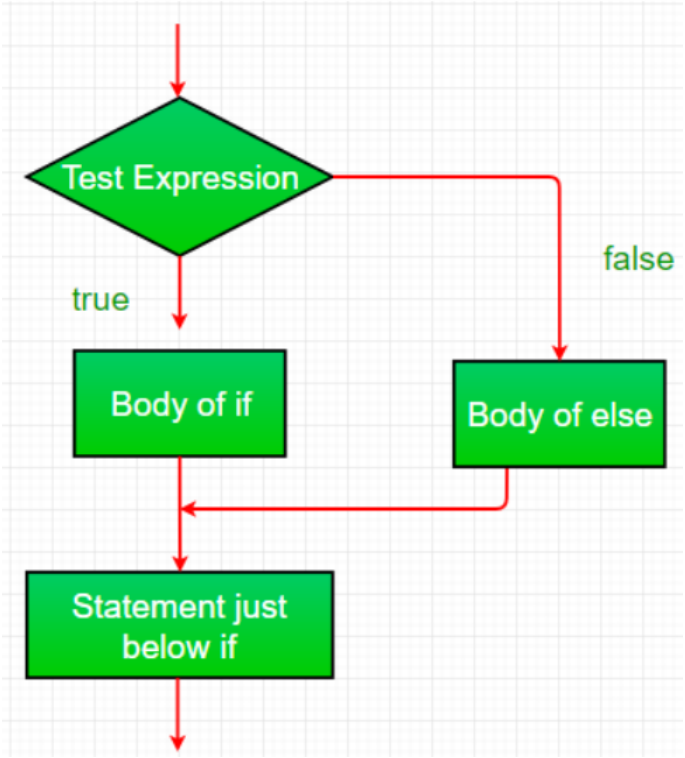
- The *if* statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't.
- But what if we want to do something else if the condition is false. Here comes the C++ **else statement**.
- We can use the else statement with if statement to execute a block of code when the condition is false.



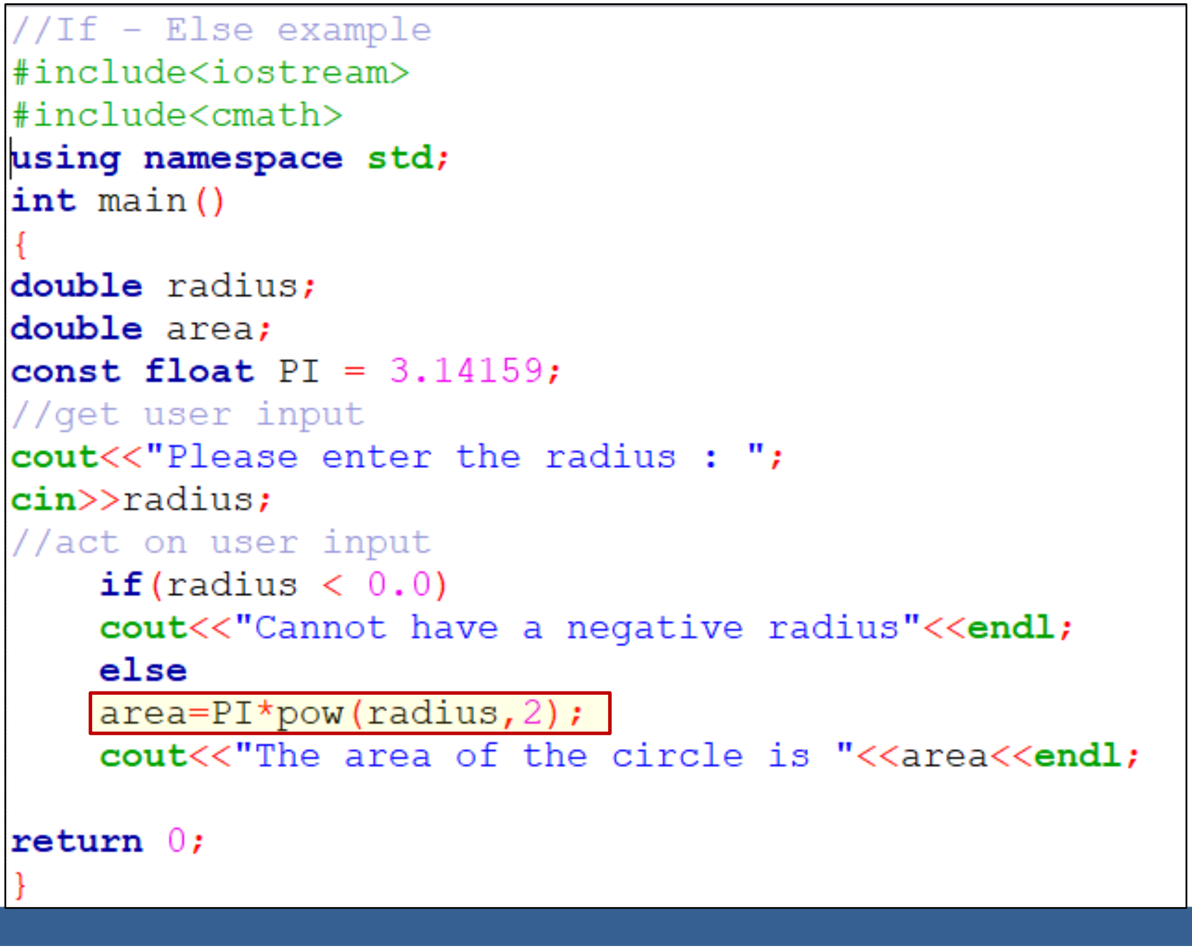
```
if ( condition )
{

    // body of the if
    // statements to be executed

}
else
{

    // body of the else
    // statements to be executed

}
// statements outside the if-else
```



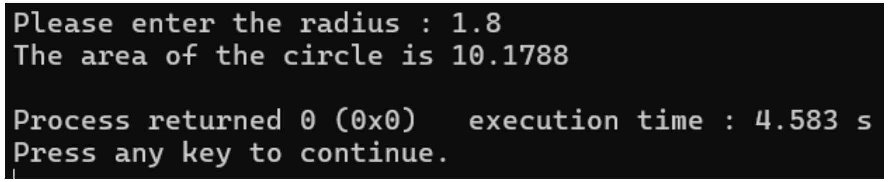
```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}

next statement
```



- WAP to evaluate the area of a circle.



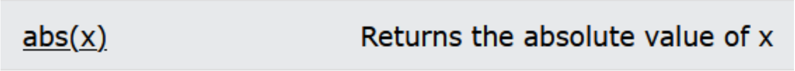
```
//If - Else example
#include<iostream>
#include<cmath>
using namespace std;
int main()
{
double radius;
double area;
const float PI = 3.14159;
//get user input
cout<<"Please enter the radius : ";
cin>>radius;
//act on user input
    if(radius < 0.0)
    cout<<"Cannot have a negative radius"<<endl;
    else
    area=PI*pow(radius,2);
    cout<<"The area of the circle is "<<area<<endl;

return 0;
}
```



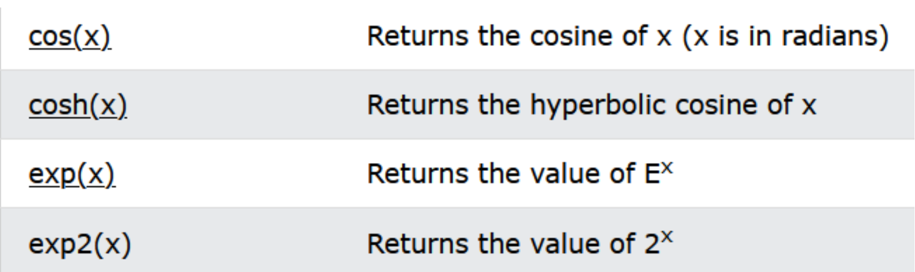
```
Please enter the radius : 1.8
The area of the circle is 10.1788

Process returned 0 (0x0)   execution time : 4.583 s
Press any key to continue.
```

C++ cmath Library

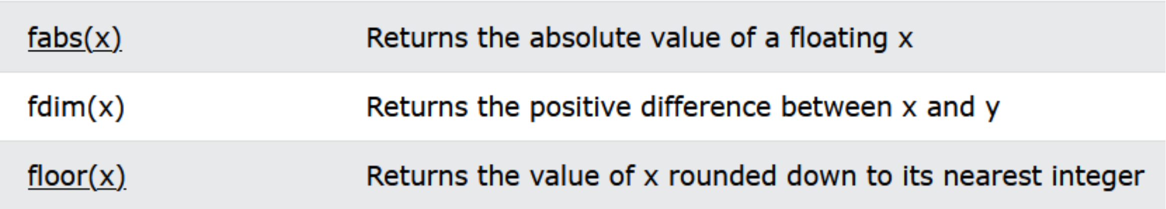
- The <cmath> library has many functions that allow you to perform mathematical tasks on numbers.



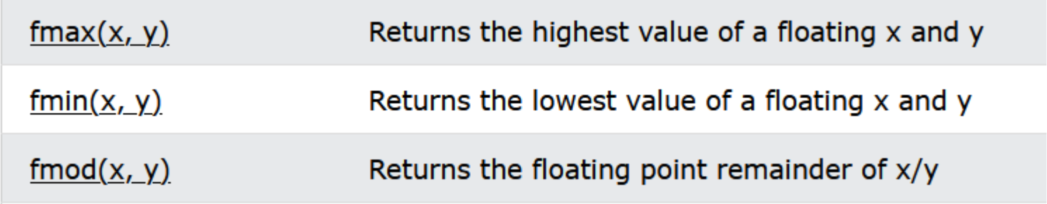
abs(x)	Returns the absolute value of x



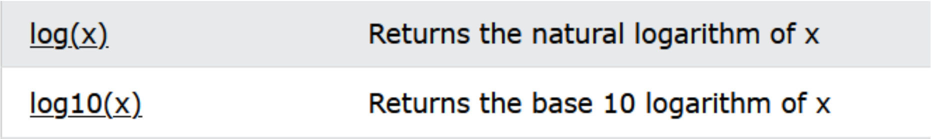
cos(x)	Returns the cosine of x (x is in radians)
cosh(x)	Returns the hyperbolic cosine of x
exp(x)	Returns the value of E^{x}
exp2(x)	Returns the value of 2^{x}



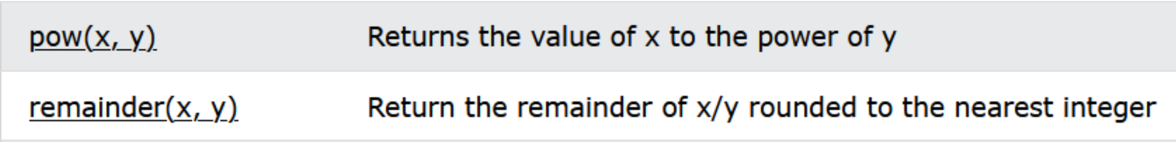
fabs(x)	Returns the absolute value of a floating x
fdim(x)	Returns the positive difference between x and y
floor(x)	Returns the value of x rounded down to its nearest integer



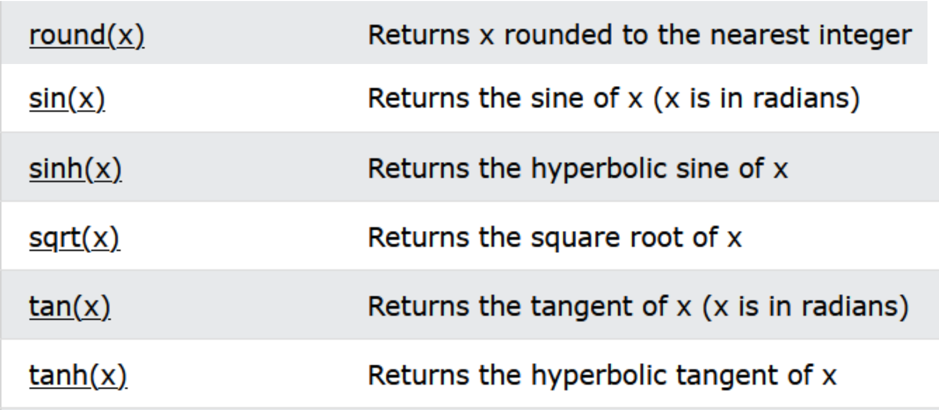
fmax(x, y)	Returns the highest value of a floating x and y
fmin(x, y)	Returns the lowest value of a floating x and y
fmod(x, y)	Returns the floating point remainder of x/y



log(x)	Returns the natural logarithm of x
log10(x)	Returns the base 10 logarithm of x



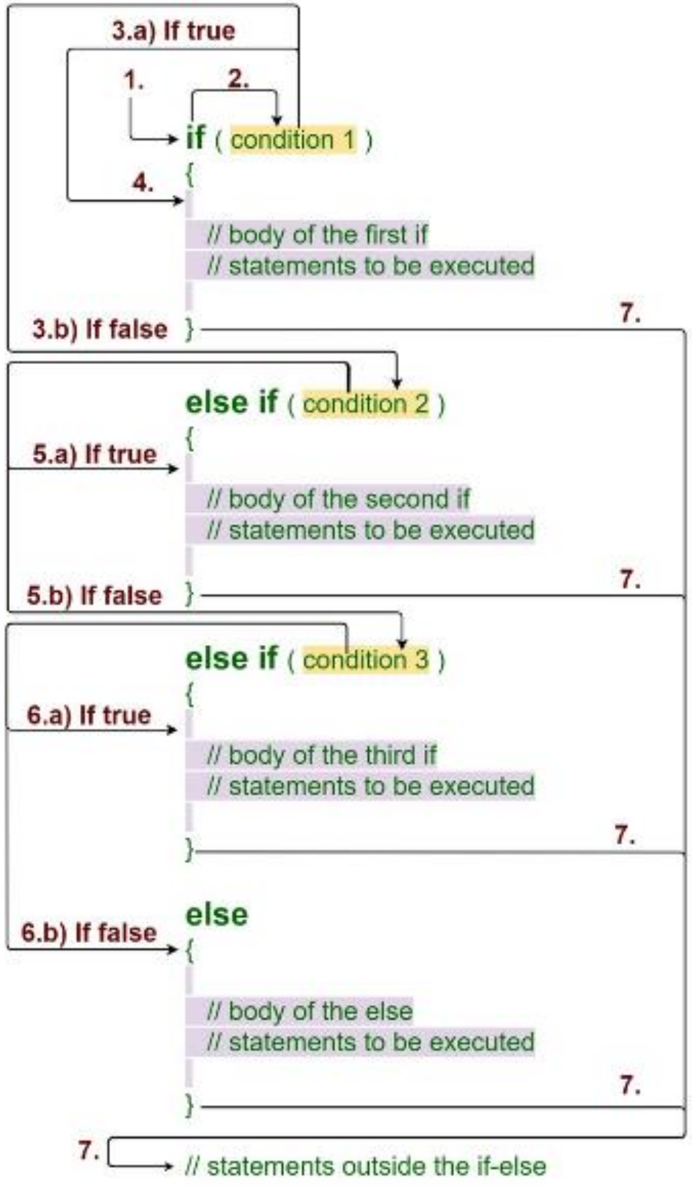
pow(x, y)	Returns the value of x to the power of y
remainder(x, y)	Return the remainder of x/y rounded to the nearest integer



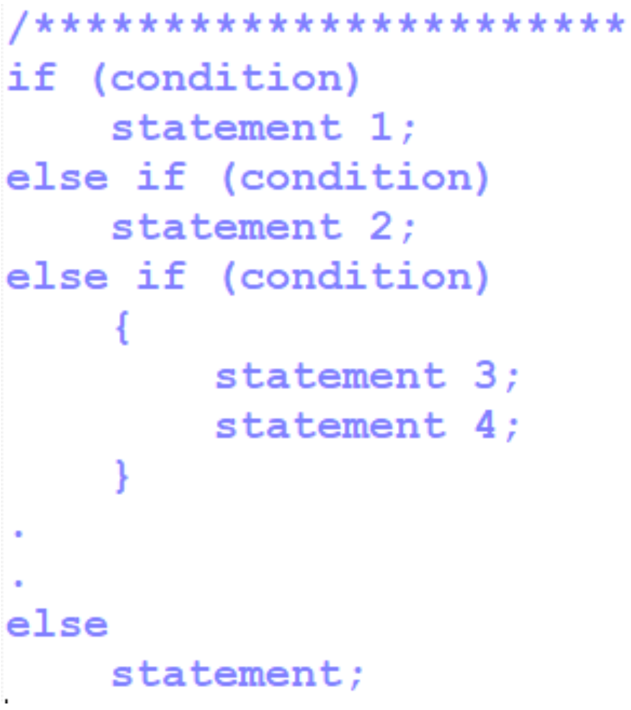
round(x)	Returns x rounded to the nearest integer
sin(x)	Returns the sine of x (x is in radians)
sinh(x)	Returns the hyperbolic sine of x
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of x (x is in radians)
tanh(x)	Returns the hyperbolic tangent of x

if-else-if ladder

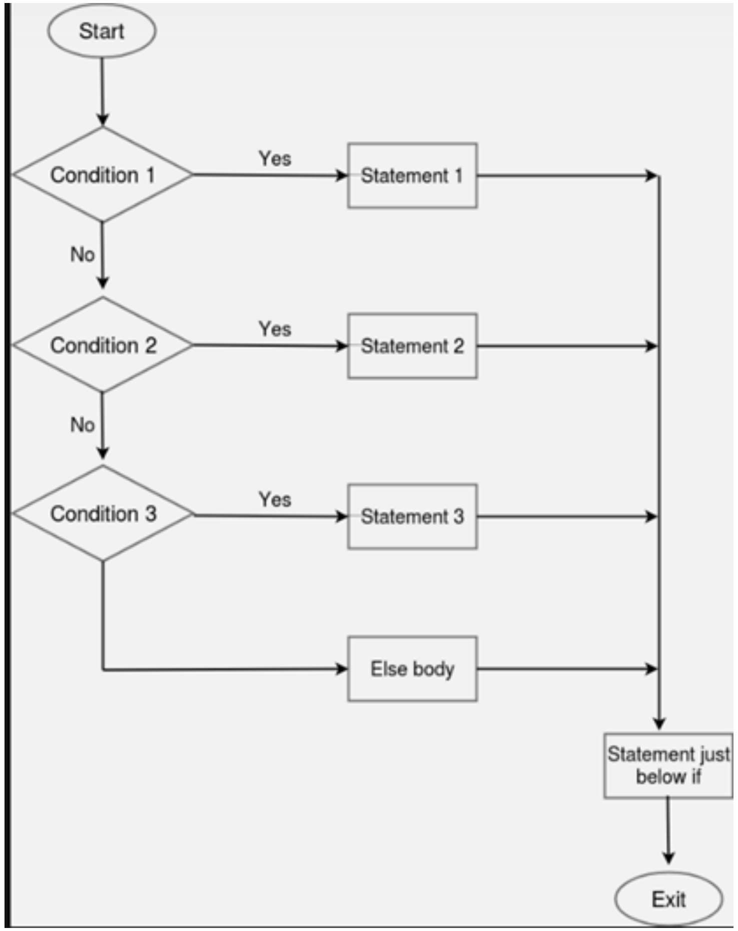
- In C++, the **if-else-if ladder** helps the user decide from among multiple options.
- The C++ if statements are executed from the top down.
- As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C++ else-if ladder is bypassed. If none of the conditions is true, then the final statement will be executed.

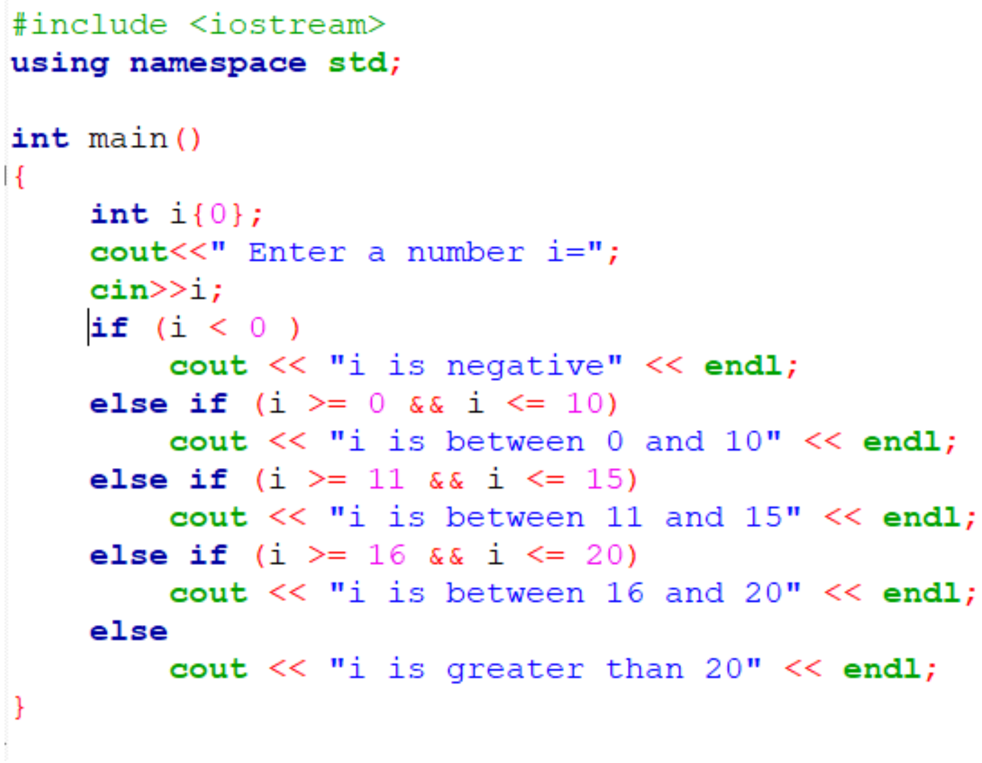


```
if ( condition 1 )
{
    // body of the first if
    // statements to be executed
}
else if ( condition 2 )
{
    // body of the second if
    // statements to be executed
}
else if ( condition 3 )
{
    // body of the third if
    // statements to be executed
}
else
{
    // body of the else
    // statements to be executed
}
// statements outside the if-else
```

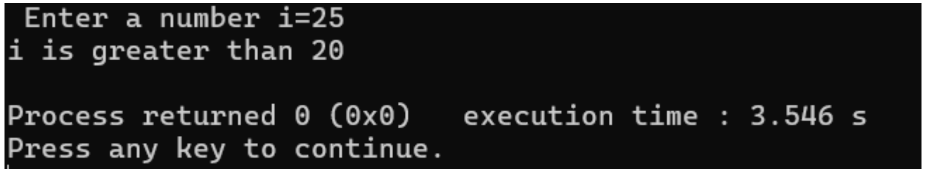


```
/**********************
if (condition)
    statement 1;
else if (condition)
    statement 2;
else if (condition)
    {
        statement 3;
        statement 4;
    }
.
.
else
    statement;
```





```cpp
#include <iostream>
using namespace std;

int main()
{
    int i{0};
    cout<<" Enter a number i=";
    cin>>i;
    if (i < 0 )
        cout << "i is negative" << endl;
    else if (i >= 0 && i <= 10)
        cout << "i is between 0 and 10" << endl;
    else if (i >= 11 && i <= 15)
        cout << "i is between 11 and 15" << endl;
    else if (i >= 16 && i <= 20)
        cout << "i is between 16 and 20" << endl;
    else
        cout << "i is greater than 20" << endl;
}
```



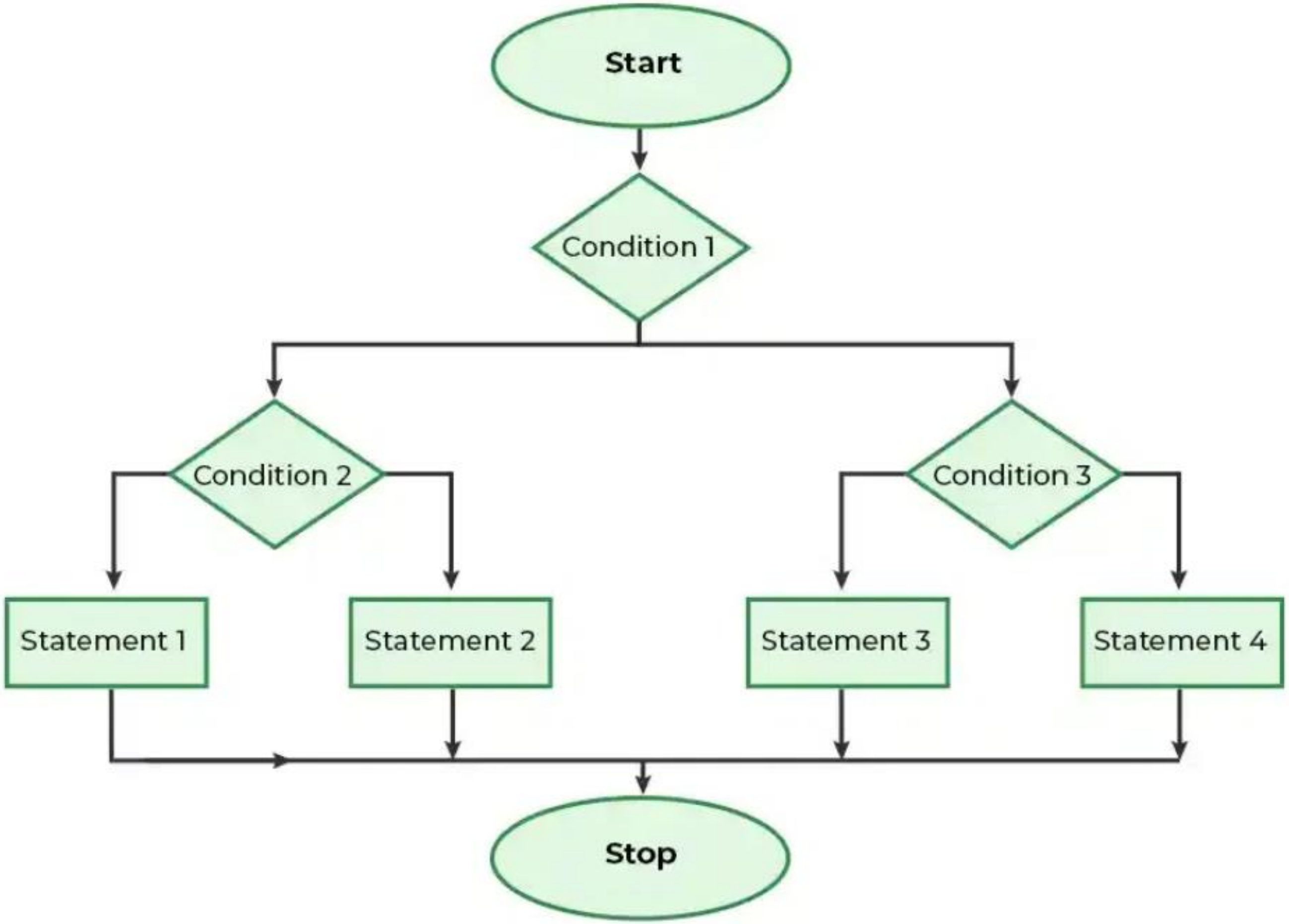
```
 Enter a number i=25
i is greater than 20

Process returned 0 (0x0)   execution time : 3.546 s
Press any key to continue.
```

Nested If-else

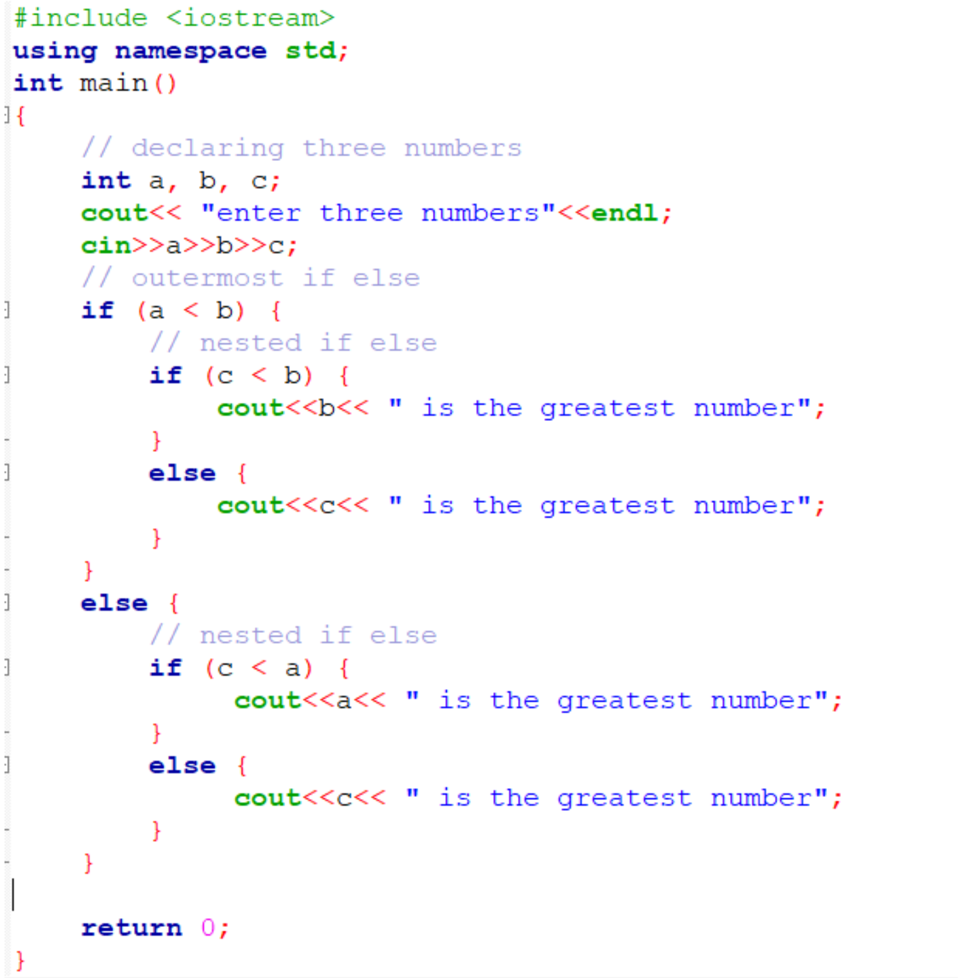
- Nested if-else statements are those statements in which there is an if statement inside another if else.
- We use nested if-else statements when we want to implement multilayer conditions(condition inside the condition inside the condition and so on).
- C++ allows any number of nesting levels.

```
if(condition1)
      {
      // Code to be executed
      if(condition2)
        {
            // Code to be executed
            }
      else
            {
            // Code to be executed
            }
        }
else
      {
      // code to be executed
      }
```

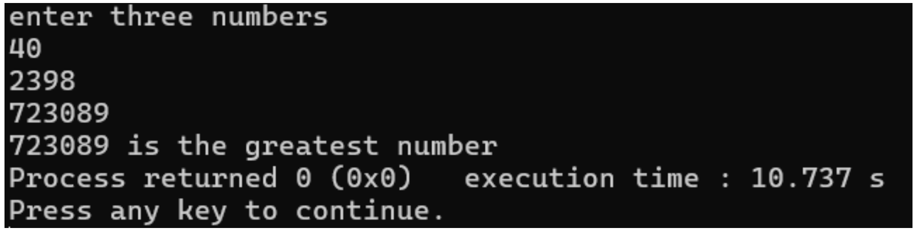
Flow Chart


Start
Condition 1
Condition 2
Condition 3
Statement 1
Statement 2
Statement 3
Statement 4
Stop

Example



```cpp
#include <iostream>
using namespace std;
int main()
{
    // declaring three numbers
    int a, b, c;
    cout<< "enter three numbers"<<endl;
    cin>>a>>b>>c;
    // outermost if else
    if (a < b) {
        // nested if else
        if (c < b) {
            cout<<b<< " is the greatest number";
        }
        else {
            cout<<c<< " is the greatest number";
        }
    }
    else {
        // nested if else
        if (c < a) {
             cout<<a<< " is the greatest number";
        }
        else {
             cout<<c<< " is the greatest number";
        }
    }

    return 0;
}
```



```
enter three numbers
40
2398
723089
723089 is the greatest number
Process returned 0 (0x0)   execution time : 10.737 s
Press any key to continue.
```

- **An ‘else’ gets paired with the closest preceding unpaired ‘if’.**

- A case of confusion

```
if (marks > 40)
   if (marks < 60)
         cout << “Passing but marginal”;
else cout << “failing”;
```

The following if statements are valid

```
if ( 3 + 2 % 5 )
cout<<"This works"  ;

if ( a = 10 )
cout<<"Even this works"  ;
```

```
if ( -5 )
cout<< "Surprisingly even this works"  ;
```

- In C++ a non-zero value is considered to be true, whereas a 0 is considered to be false. In the first if, the expression evaluates to 5 and since 5 is non-zero it is considered to be true. Hence the cout gets executed.

- In the second **if**, 10 gets assigned to **a** so the **if** is now reduced to **if (a)** or **if (10)**. Since 10 is non-zero, it is true hence again **cout** goes to work. In the third **if**, -5 is a non-zero number, hence true. So again **cout** goes to work.

- In place of -5 even if a float like 3.14 were used it would be considered to be true. So the issue is not whether the number is integer or float, or whether it is positive or negative. Issue is whether it is zero or non-zero.

The switch-case-break Statement

- The C++ Switch case statement evaluates a given expression and based on the evaluated value(matching a certain condition), it executes the statements associated with it. It is an alternative to the long if-else-if ladder which provides an easy way to dispatch execution to different parts of code based on the value of the expression.

- The switch statement in C++ is a flow control statement that is used to execute the different blocks of statements based on the value of the given expression. We can create different cases for different values of the switch expression.

- We can specify any number of cases in the switch statement but the **case value** can only be of **type int** or **char**.

- The most flexible program control statement in selection structure of program control.

- Enables the program to execute different statements based on an expression that can have more than two values. Also called multiple choice statements.

- Before this, such as if statement, were limited to evaluating an expression that could have only two values: TRUE or FALSE.

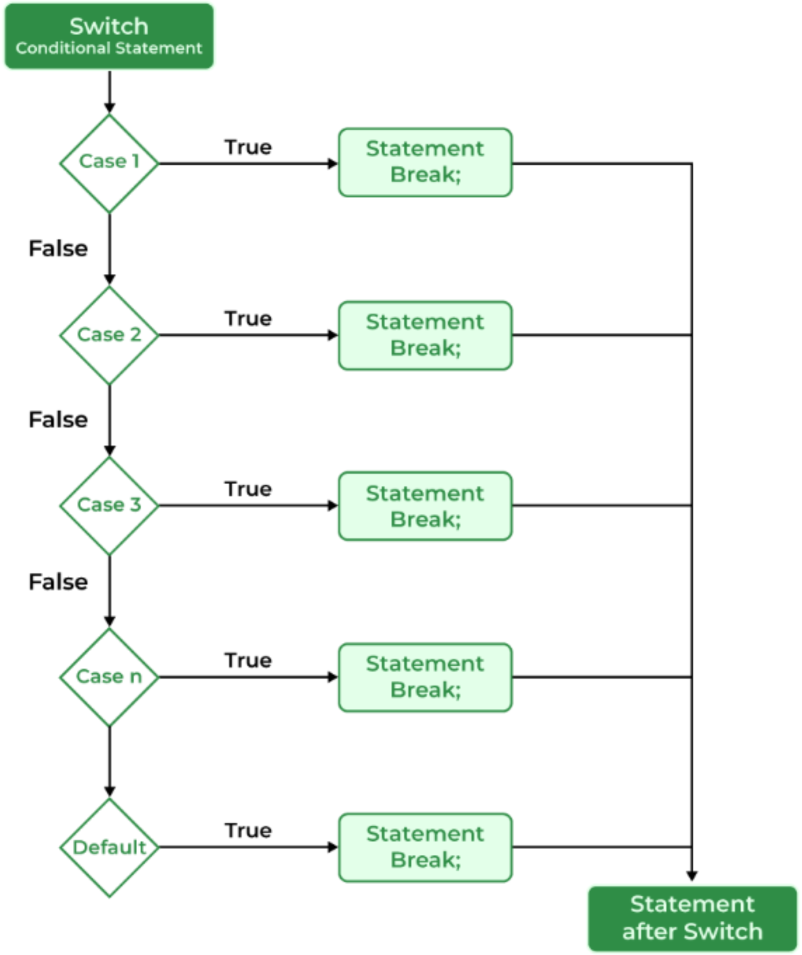
- If more than two values, have to use nested if statements. The switch statement makes such nesting unnecessary.

- Used together with case and break.

Flowchart

The working of the switch statement in C is as follows:

1. **Step 1:** The switch expression is evaluated.
2. **Step 2:** The evaluated value is then matched against the present case values.
3. **Step 3A:** If the matching case value is found, that case block is executed.
4. **Step 3B:** If the matching code is not found, then the default case block is executed if present.
5. **Step 4A:** If the break keyword is present in the block, then program control comes out of the switch statement.
6. **Step 4B:** If the break keyword is not present, then all the cases after the matching case are executed.
7. **Step 5:** Statements after the switch statement is executed.



```
switch(expression)
{
case  ConstExp_1  : statement(s);
                    break;
case  ConstExp_2  : statement(s);
                    break;
...................
...................
case  ConstExp_n : statement(s);
                    break;
default : statement(s);
}
next_statement;
```

•Evaluates the (expression) and compares its value with the ConstExps following each case label.

•If a match is found between (expression) and one of the ConstExp_n, execution is transferred to statement(s) that follows the case label.

•If no match is found, execution is transferred to the statement(s) following the optional default label.

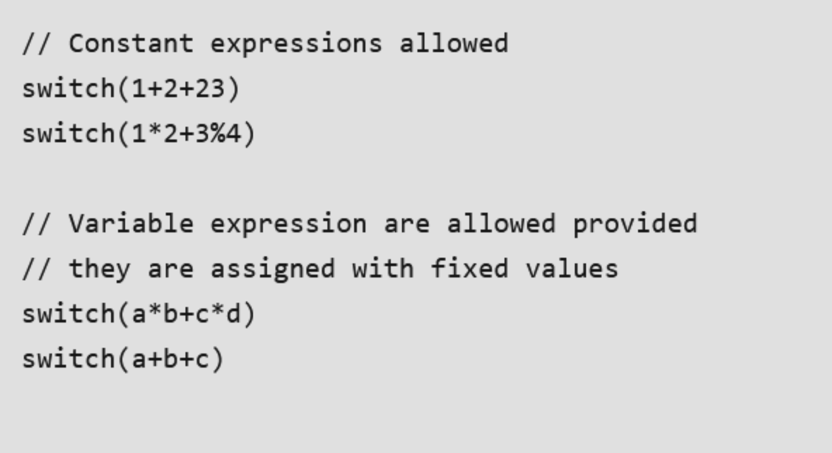
•If no match is found and there is no default label, execution passes to the first statement following the switch statement closing brace, the next_statement.

•To ensure that only the statements associated with the matching ConstExp are executed, include a break statement where needed, which terminates the entire switch statement.

•As usual the statement(s) can also be a block of code put in curly braces.

Rules

- There are some rules that we need to follow when using switch statements in C++. They are as follows:
  - The case value must be either int or char type.
  - There can be any number of cases.
  - No duplicate case values are allowed.
  - Each statement of the case can have a break statement. It is optional.
  - The default Statement is also optional.



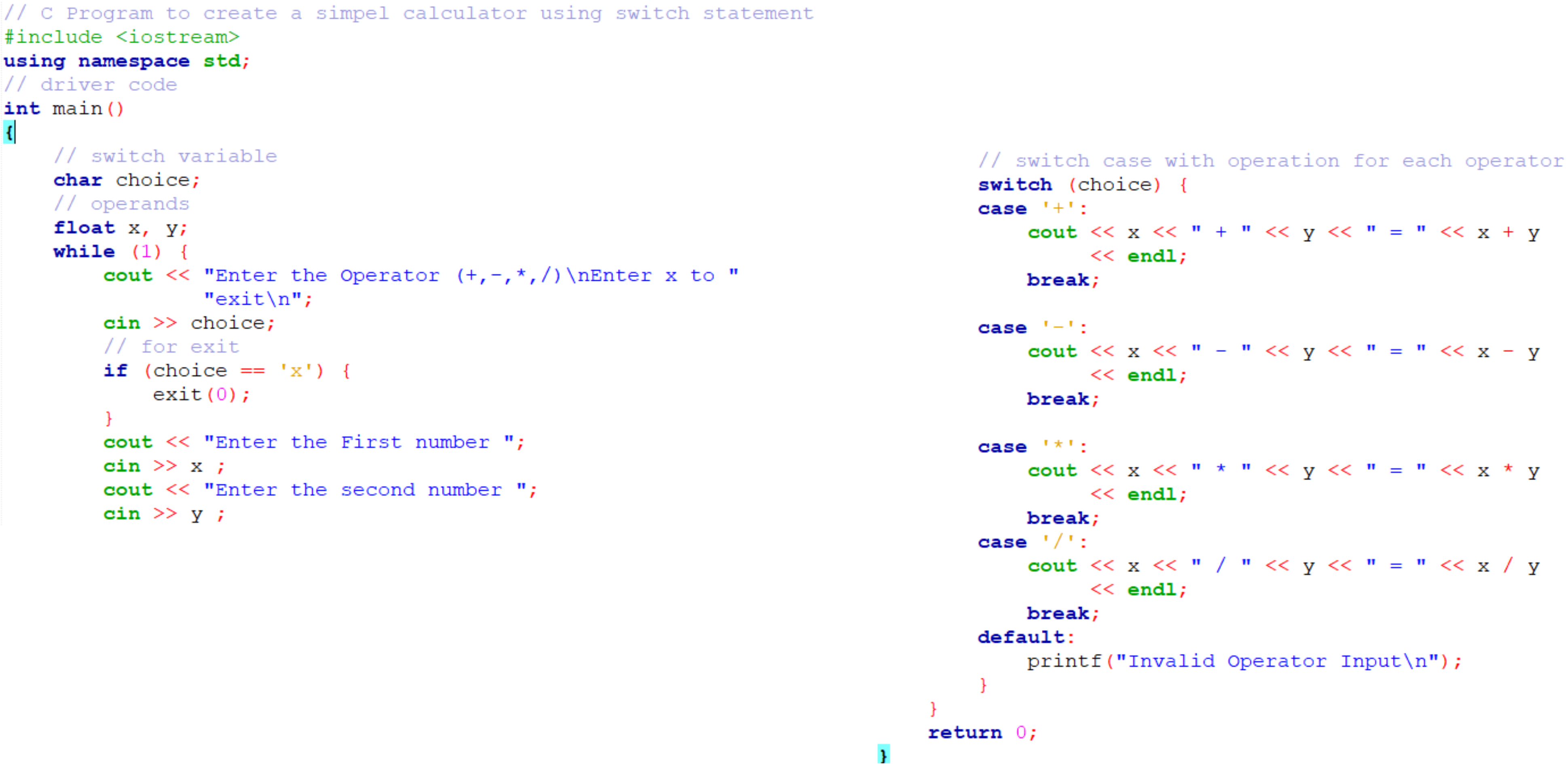
```
// Constant expressions allowed
switch(1+2+23)
switch(1*2+3%4)

// Variable expression are allowed provided
// they are assigned with fixed values
switch(a*b+c*d)
switch(a+b+c)
```

- **Advantages of switch Statement in C++**
- Easier to debug and maintain for a large number of conditions.
- Easier to read than if else if.

- **Disadvantages of switch Statement in C++**
- Switch case can only evaluate int or char type.
- No support for logical expressions.
- Have to keep in mind to add a break in every case.

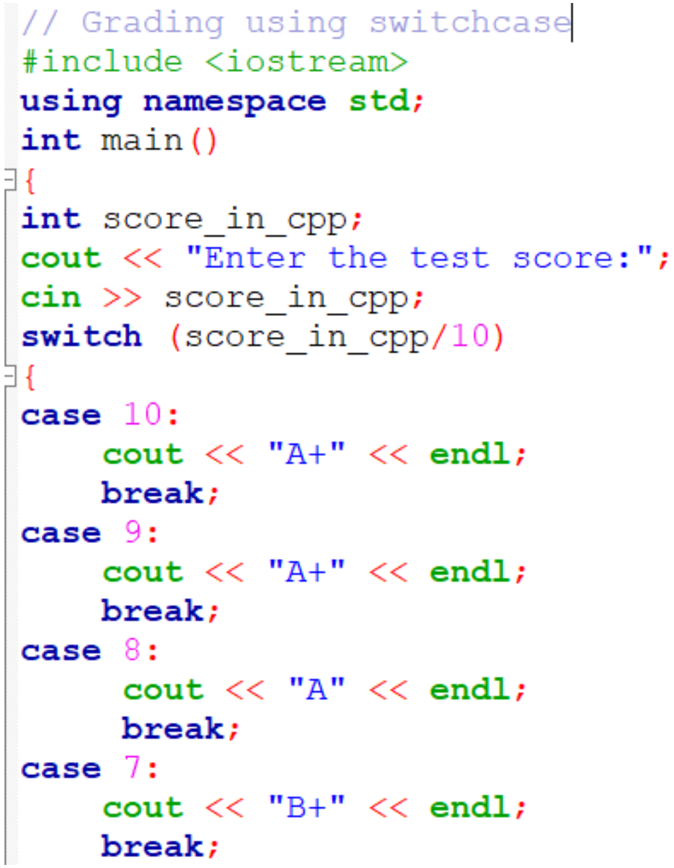
- WAP to implement a simple calculator using Switch case.

Example: A simple calculator

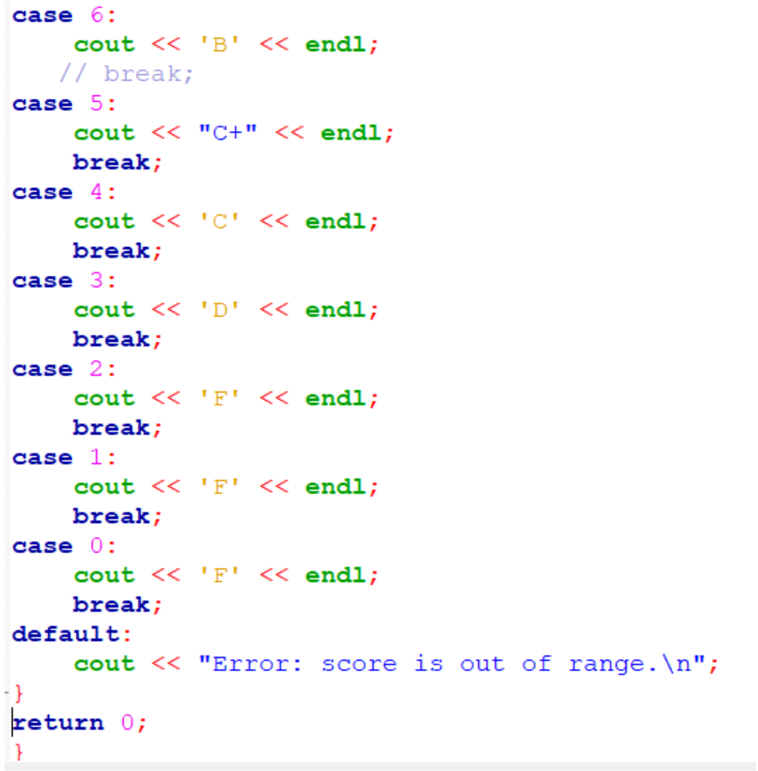


```
// C Program to create a simpel calculator using switch statement
#include <iostream>
using namespace std;
// driver code
int main()
{
    // switch variable
    char choice;
    // operands
    float x, y;
    while (1) {
        cout << "Enter the Operator (+,-,*,/)\nEnter x to "
                "exit\n";
        cin >> choice;
        // for exit
        if (choice == 'x') {
            exit(0);
        }
        cout << "Enter the First number ";
        cin >> x ;
        cout << "Enter the second number ";
        cin >> y ;
        // switch case with operation for each operator
        switch (choice) {
        case '+':
            cout << x << " + " << y << " = " << x + y
                 << endl;
            break;

        case '-':
            cout << x << " - " << y << " = " << x - y
                 << endl;
            break;

        case '*':
            cout << x << " * " << y << " = " << x * y
                 << endl;
            break;
        case '/':
            cout << x << " / " << y << " = " << x / y
                 << endl;
            break;
        default:
            printf("Invalid Operator Input\n");
        }
    }
    return 0;
}
```

Example



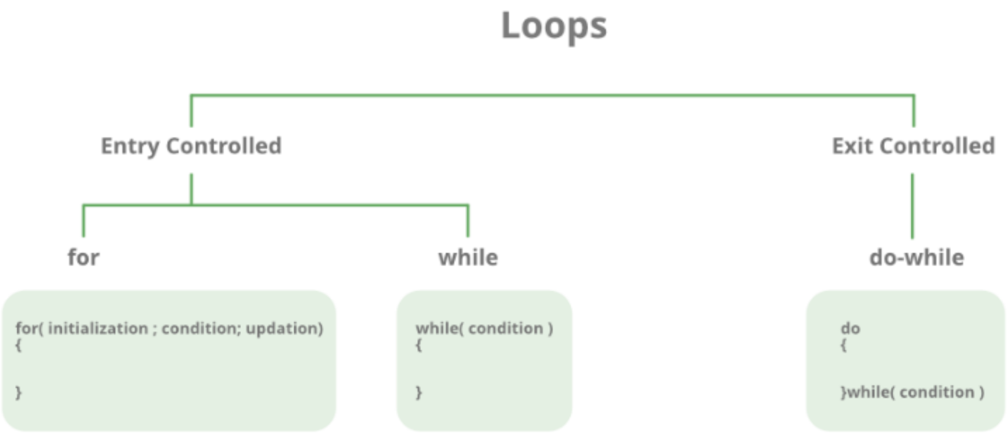
```
// Grading using switchcase
#include <iostream>
using namespace std;
int main()
{
int score_in_cpp;
cout << "Enter the test score:";
cin >> score_in_cpp;
switch (score_in_cpp/10)
{
case 10:
    cout << "A+" << endl;
    break;
case 9:
    cout << "A+" << endl;
    break;
case 8:
     cout << "A" << endl;
     break;
case 7:
    cout << "B+" << endl;
    break;
```



```
case 6:
    cout << 'B' << endl;
   // break;
case 5:
    cout << "C+" << endl;
    break;
case 4:
    cout << 'C' << endl;
    break;
case 3:
    cout << 'D' << endl;
    break;
case 2:
    cout << 'F' << endl;
    break;
case 1:
    cout << 'F' << endl;
    break;
case 0:
    cout << 'F' << endl;
    break;
default:
    cout << "Error: score is out of range.\n";
}
return 0;
}
```

Looping

- In Programming, sometimes there is a need to perform some operation **more than once** or (say) **n number** of times. Loops come into use when we need to repeatedly execute a block of statements.
- **There are mainly two types of loops:**
- **Entry Controlled loops**: In this type of loop, the test condition is tested before entering the loop body. **For Loop** and **While Loop** is entry-controlled loops.
- **Exit Controlled Loops**: In this type of loop the test condition is tested or evaluated at the end of the loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is true or false. the do-while **loop** is exit controlled loop.



Loops

Entry Controlled

Exit Controlled

for

while

do-while

```
for( initialization ; condition; updation)
{

}
```

```
while( condition )
{

}
```

```
do
{

}while( condition )
```

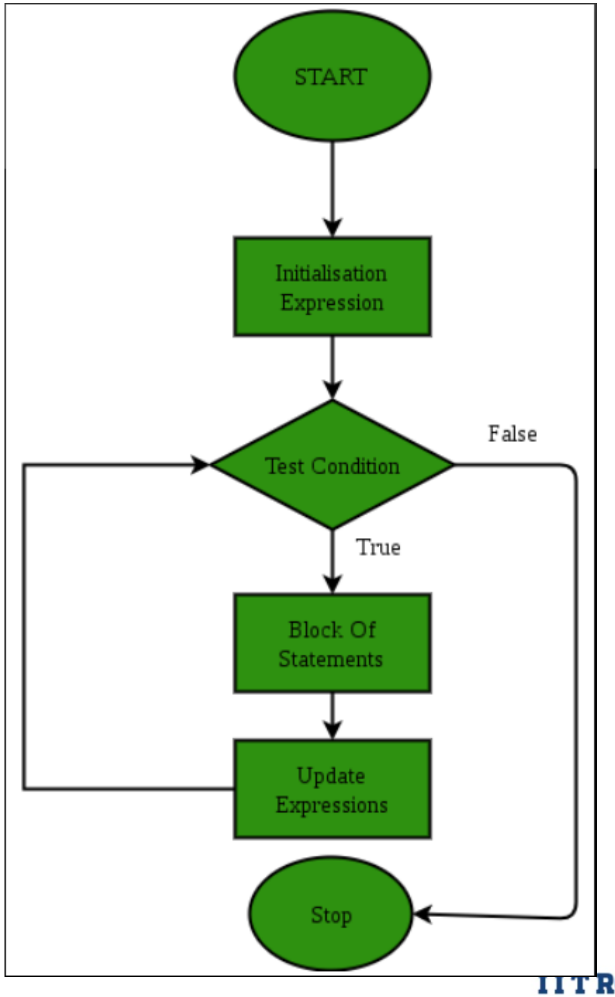
Sl. No.	Loop Type	Description
1.	while loop	– First checks the condition, then executes the body.
2.	for loop	– firstly initializes, then, condition check, execute body, update.
3.	do-while loop	– firstly, execute the body then condition check

For Loop

- A *For loop* is a repetition control structure that allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.

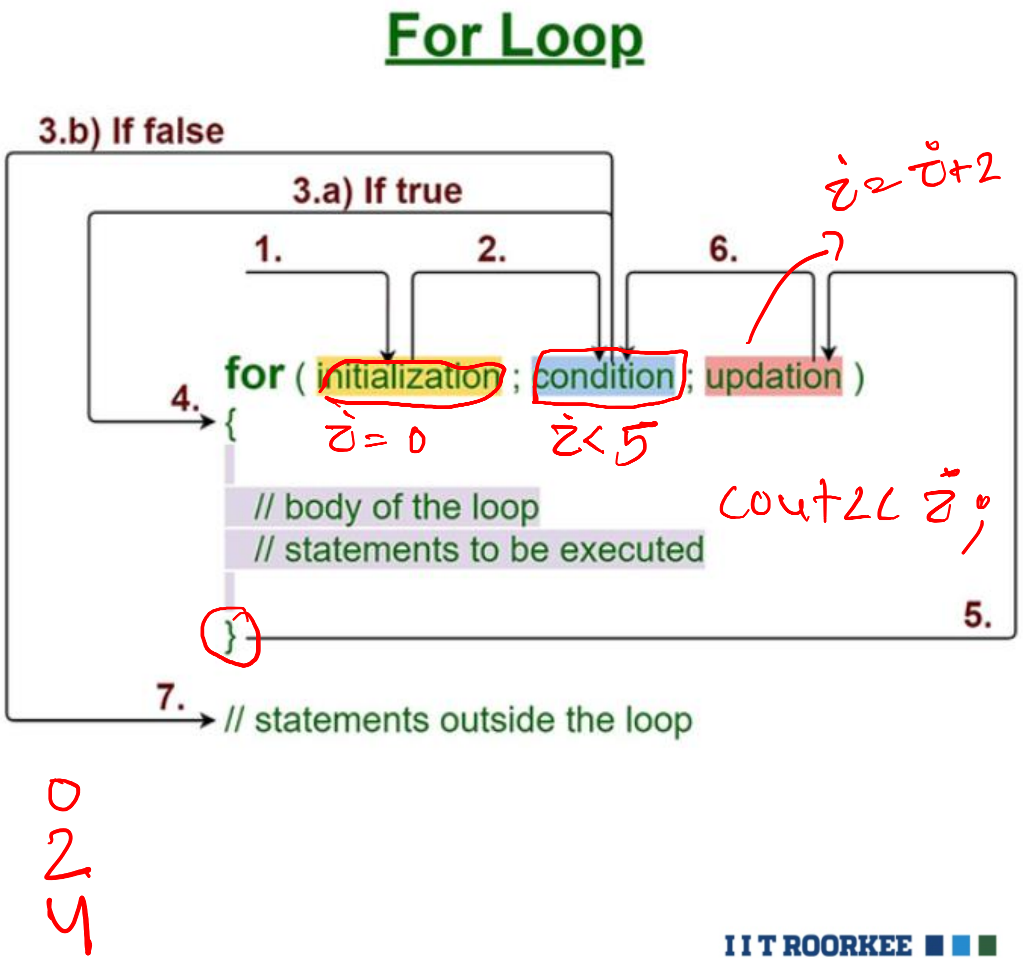
Flowchart Syntax

```
//syntax – for loop

for (initialization expr; test expr; update expr)
{
// body of the loop
// statements we want to execute
}
```



Execution Flow

1. Control falls into the for loop. Initialization is done.
2. The flow jumps to Condition.
3. Condition is tested.
4. If the Condition yields true, the flow goes into the Body.
5. If the Condition yields false, the flow goes outside the loop.
6. The statements inside the body of the loop get executed.
7. The flow goes to the update.
8. Updation takes place and the flow goes to Step 3 again.
9. The for loop has ended and the flow has gone outside


For Loop
3.b) If false
3.a) If true
1.
2.
6.
4.
5.
7.
for (initialization ; condition ; updation)
{
// body of the loop
// statements to be executed
}
// statements outside the loop
i = i + 2
i = 0
i < 5
cout << i;
0
2
4

Example

- WAP to find the factorial of a number using For Loop.

```
int main ( )
{
 cout << "enter a +ve number";
  int n;
  cin >> n;     int i ~~=0~~;
                int F=1;
for ( i=~~0~~1; i<=n; i++ )
{
    F = F x i;

}
```

$n! = 1 \times 2 \times 3 \times \dots \quad n$

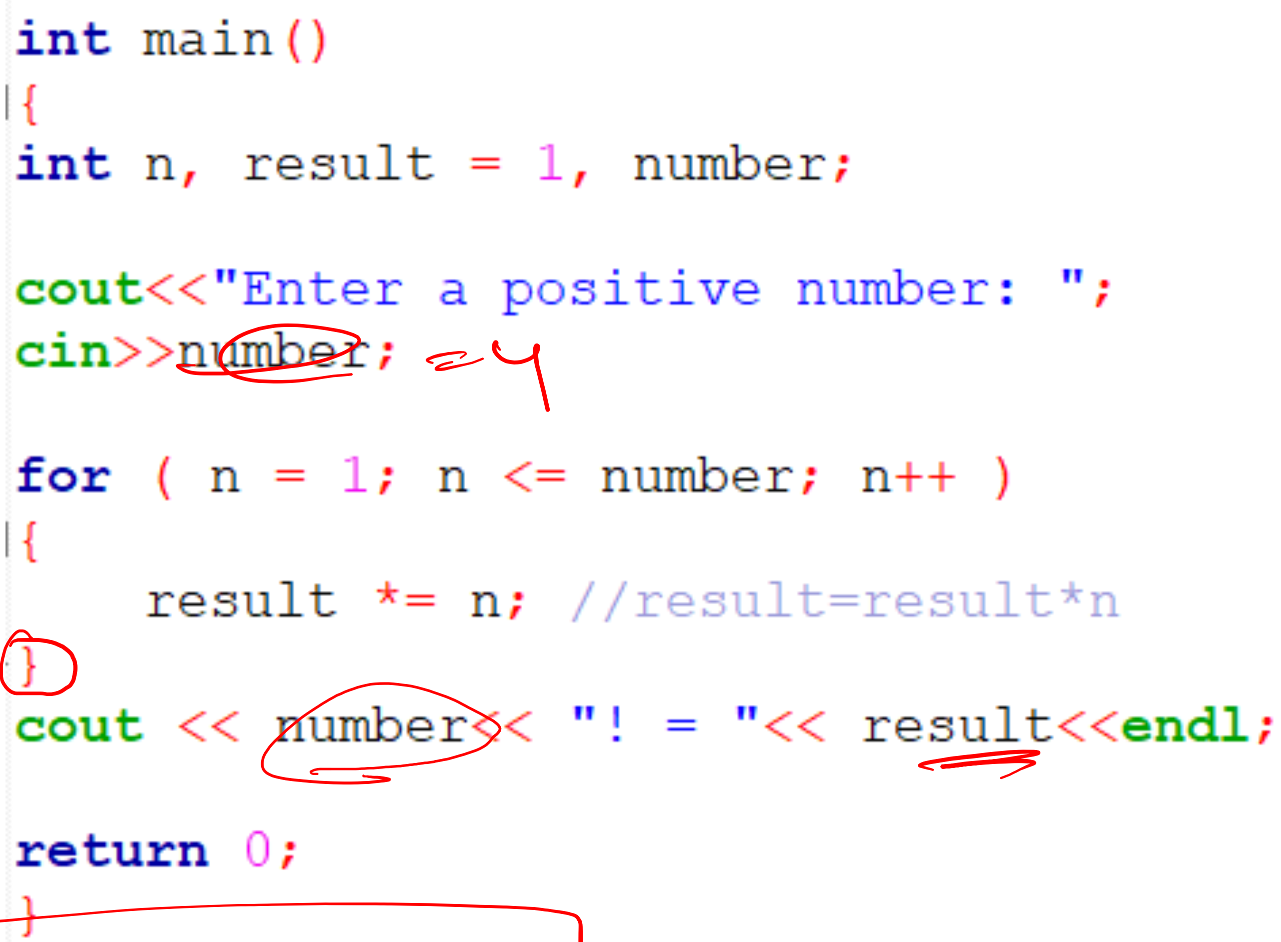
$\downarrow \quad \downarrow$

$1 \quad (1+1) \rightarrow (1+1+1) \dots$

Example

- WAP to find the factorial of a number using For Loop.

n=1 (✓)
result = 1×1
→ n=2 (✓)
result =1×2
→ n=3 (✓)
result=6
→ n=4 (✓)
result = 4×6 =24
→ n=5

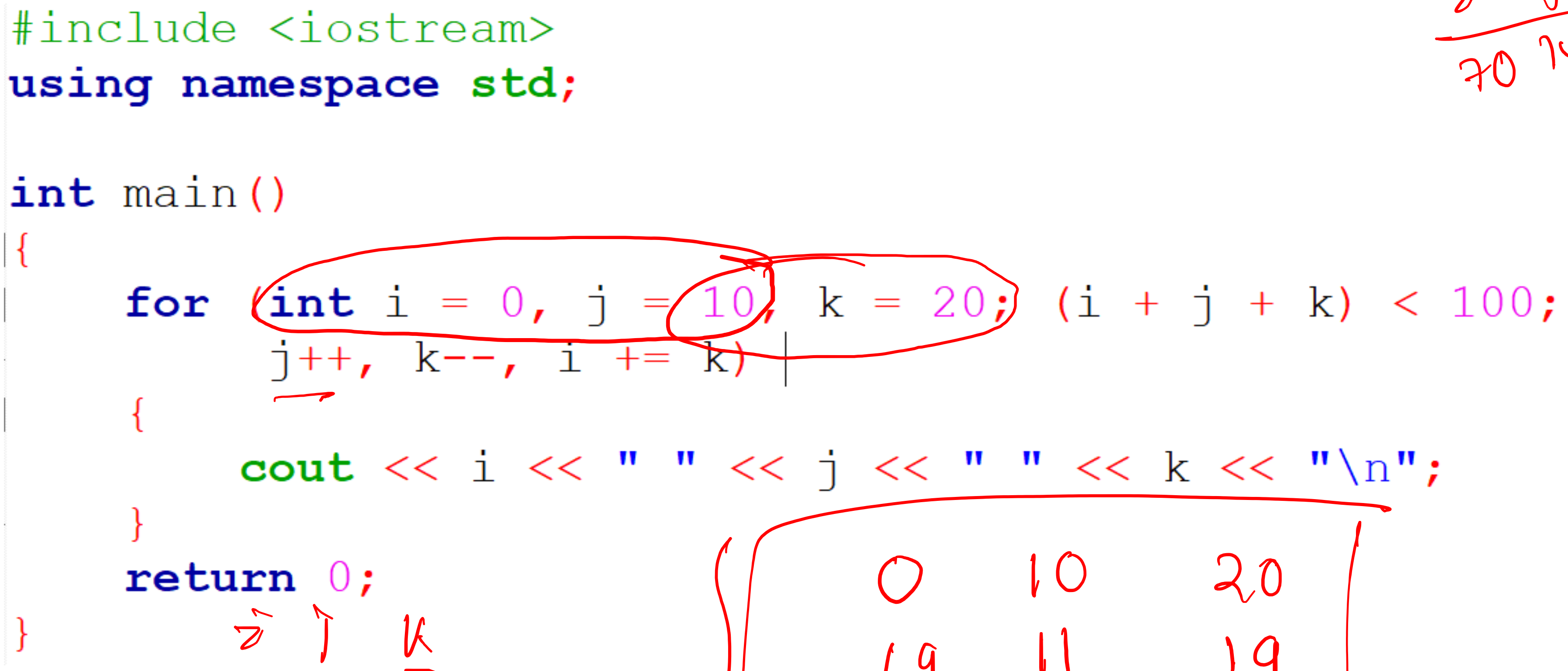


```
int main()
{
int n, result = 1, number;

cout<<"Enter a positive number: ";
cin>>number; =4

for ( n = 1; n <= number; n++ )
{
    result *= n; //result=result*n
}
cout << number<< "! = "<< result<<endl;

return 0;
}
```

O/P = 4! = 24

Output?

i	j	k
70	14	16



```
#include <iostream>
using namespace std;

int main()
{
    for (int i = 0, j = 10, k = 20; (i + j + k) < 100;
         j++, k--, i += k)
    {
        cout << i << " " << j << " " << k << "\n";
    }
    return 0;
}
```

i	j	k
0	10	20
19	11	19
37	12	18
54	13	17

```
0   10   20
19  11   19
37  12   18
54  13   17
```

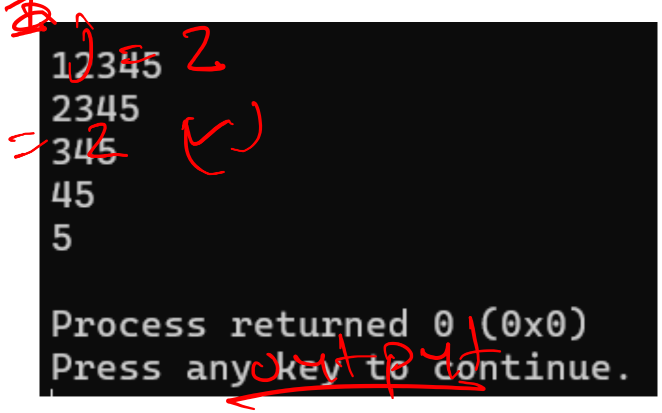
Nested Loop

```
#include <iostream>
using namespace std;
int main()
{
        int   i, j;    // variables for counter…
        // outer loop, execute this first...
        for(i=1; i<6; i++)
        {
                cout<<"\n"<<i;
                // then...execute inner loop with loop index j
                for(j=i+1; j<6; j++)
              { cout<<j; }  // display the result…
                // increment counter by 1 for inner loop…
        }
        // increment counter by 1 for outer loop…
        cout<<"\n";
        return 0;
}
```

i=1 (✓)

j= 2

i=2 (✓)



```
12345
2345
345
45
5

Process returned 0 (0x0)
Press any key to continue.
```

output

12345
2345
345
45
5

- The initialization and update fields are optional.
- They can remain empty, but in all cases, the semicolon signs between them must be written.
- for (;n<10;) if required to specify no initialization and no increase; or
- for (;n<10;n++) if required to include an increase field but no initialization (maybe because the variable was already initialized before).

- Optionally, using the comma operator (,) we can specify more than one expression in any of the fields included in a for loop
- The comma operator (,) is an expression separator, it serves to separate more than one expression where only one is generally expected.

Example

```
int n =0,i=10;

 for (     ; n!=i ; n++, i-- )
      {
           cout<<n;
      }
```

0
1
2
3
4

While Loop

- **While Loop in C++** is used in situations where we do not know the exact number of iterations of the loop beforehand.

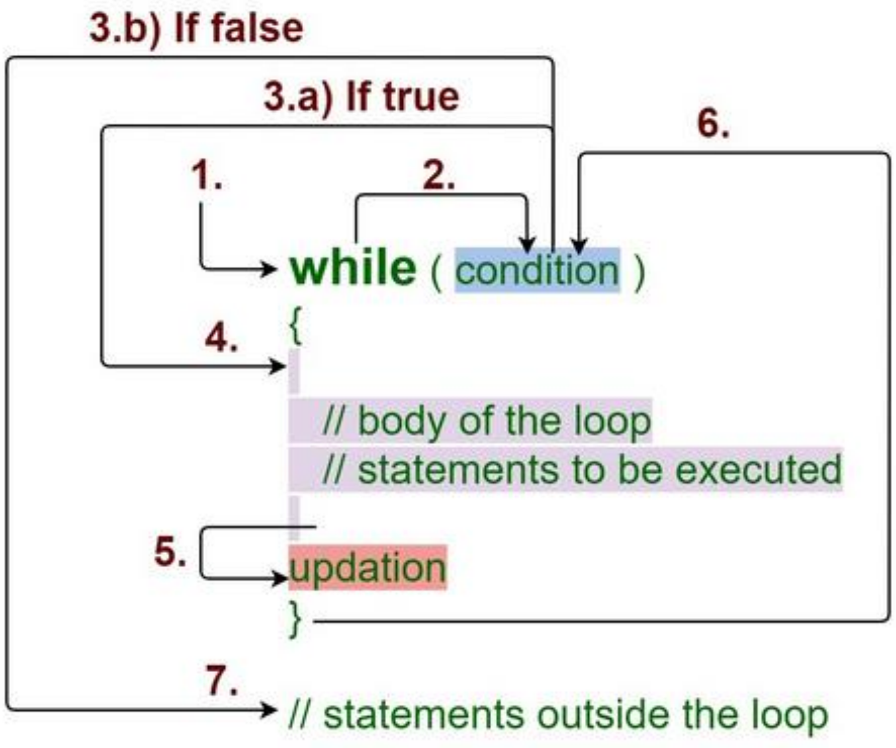
- The loop execution is terminated on the basis of the test condition. **Loops in C++** come into use when we need to repeatedly execute a block of statements.

- During the study of the **‘for’ loop in C++**, we have seen that the number of iterations is usually known beforehand, i.e. the number of times the loop body is needed to be executed is known to us.

Flowchart

How does a While loop execute?

1. Control falls into the while loop.
2. The flow jumps to Condition
3. Condition is tested.
    1. If the Condition yields true, the flow goes into the Body.
    2. If the Condition yields false, the flow goes outside the loop
4. The statements inside the body of the loop get executed.
5. Updation takes place.
6. Control flows back to Step 2.
7. The while loop has ended and the flow has gone outside.



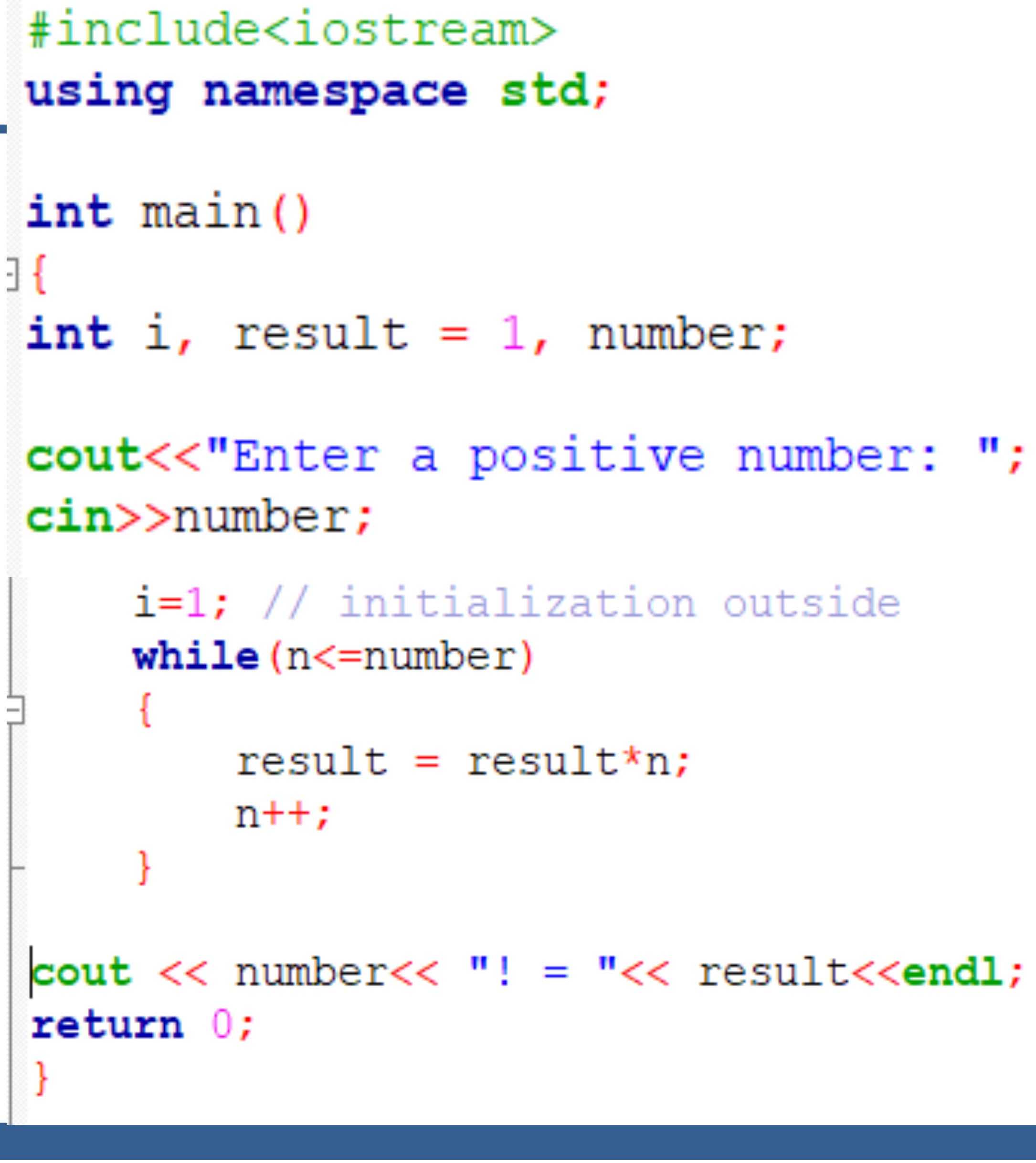
- The **while statement** has the form:

```
while(condition)
        {
                // code to execute
         }
```

- The loop variable needs initialization outside.
- *condition* is a boolean statement that is checked each time after the final "}" of the while statement executes.
- If the *condition* is true then the while statement executes again.
- If the *condition* is false, the while statement does not execute again.

Example

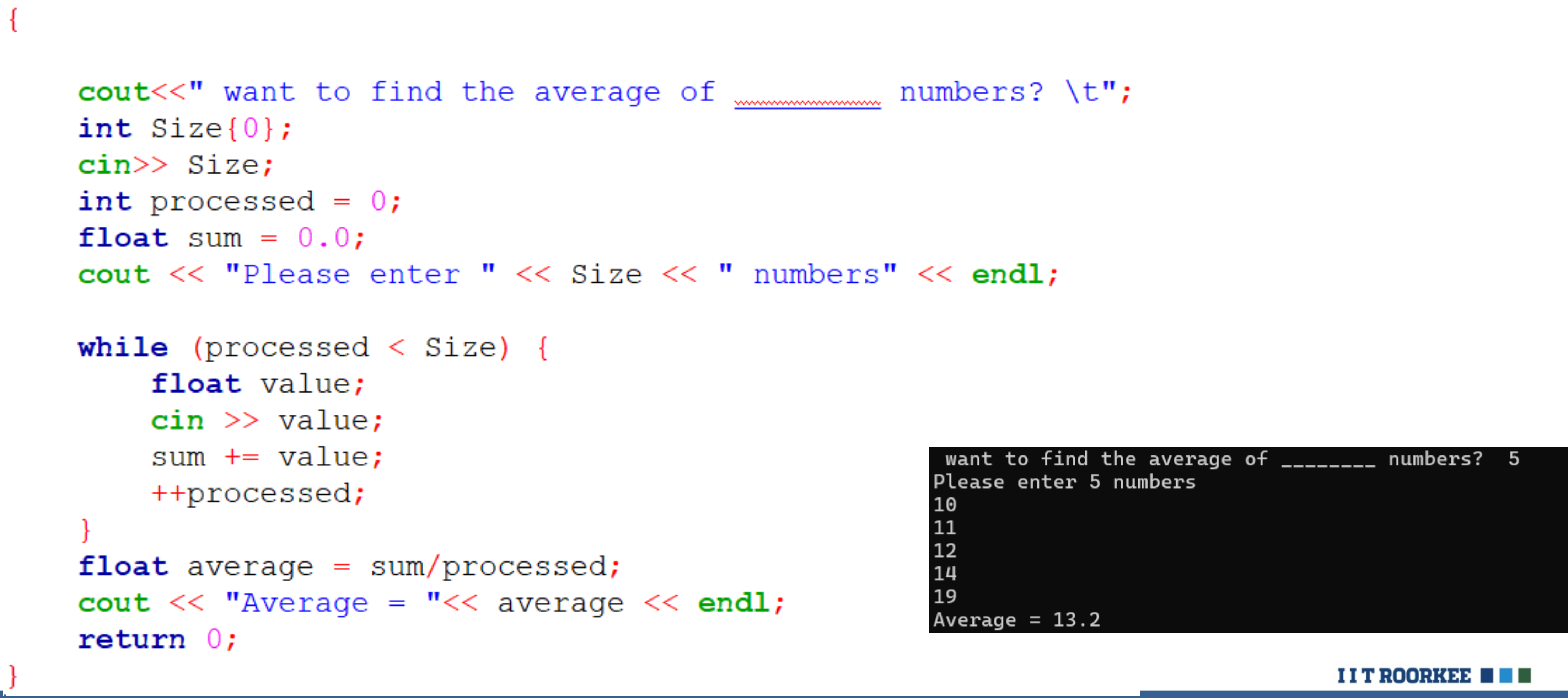
WAP to find the factorial of a number using While Loop.



```
#include<iostream>
using namespace std;

int main()
{
int i, result = 1, number;

cout<<"Enter a positive number: ";
cin>>number;
    i=1; // initialization outside
    while(n<=number)
    {
        result = result*n;
        n++;
    }

cout << number<< "! = "<< result<<endl;
return 0;
}
```

Example

WAP to find the average of ‘n’ numbers. Where ‘n’ is user input.



```
{

    cout<<" want to find the average of ________ numbers? \t";
    int Size{0};
    cin>> Size;
    int processed = 0;
    float sum = 0.0;
    cout << "Please enter " << Size << " numbers" << endl;

    while (processed < Size) {
        float value;
        cin >> value;
        sum += value;
        ++processed;
    }
    float average = sum/processed;
    cout << "Average = "<< average << endl;
    return 0;
}
```

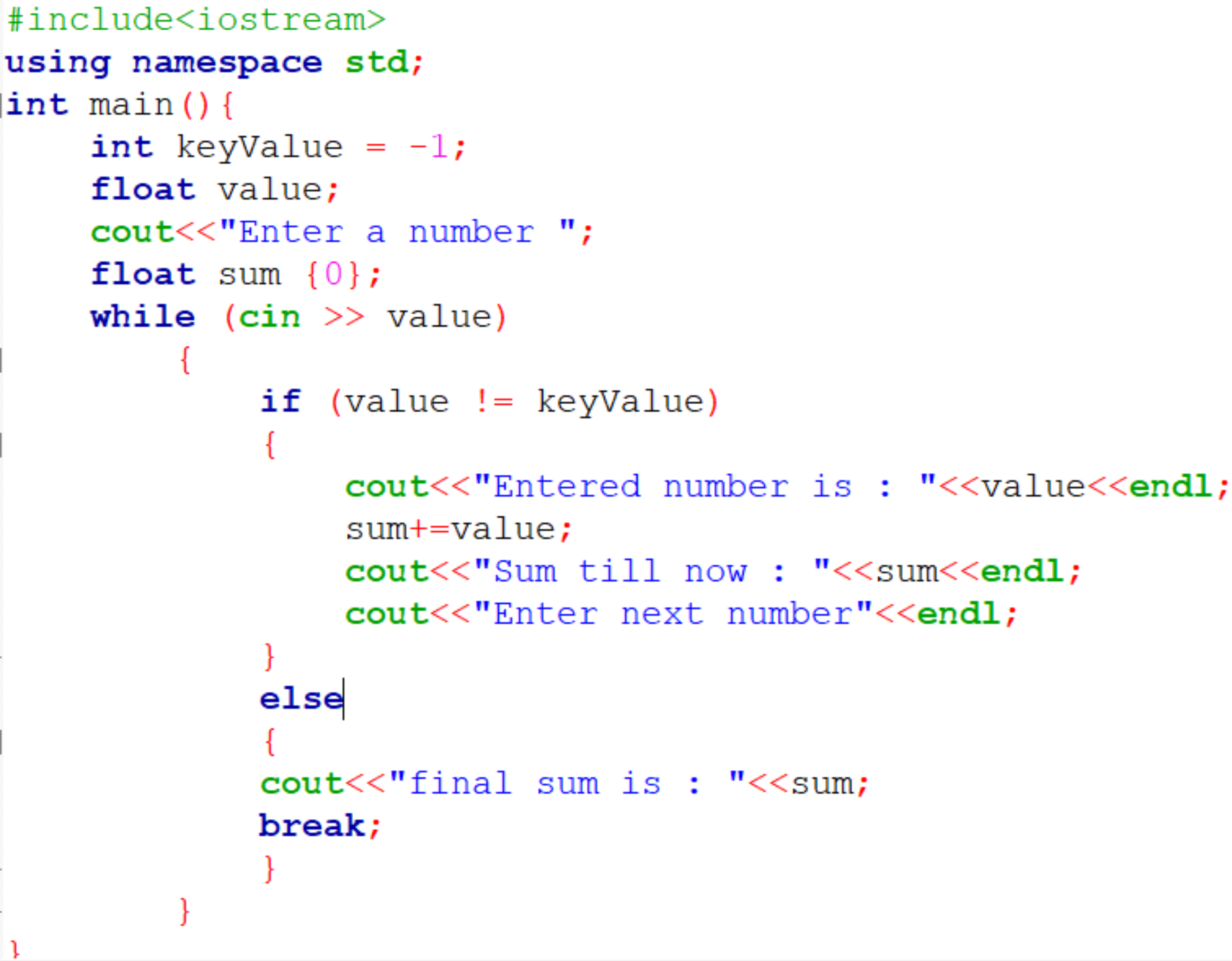
```
 want to find the average of ________ numbers?  5
Please enter 5 numbers
10
11
12
14
19
Average = 13.2
```

‘while’ usage

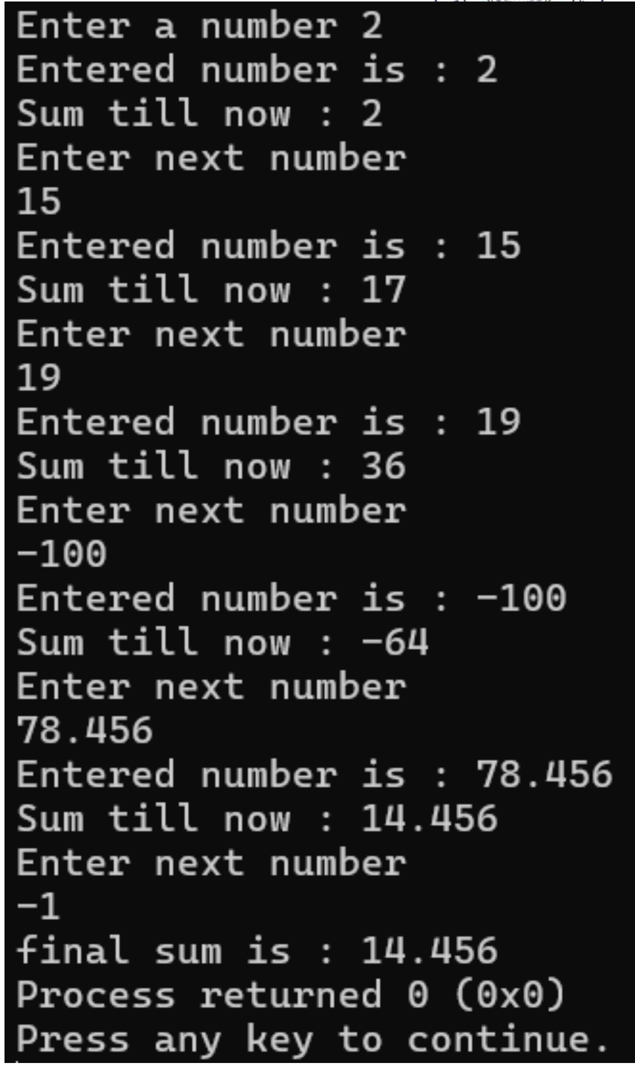
- a) Processing an arbitrary number of inputs:

```
while (cin >> value)
```

- The condition here serves to read the input value, as well as act as a continuation condition.
- The value of (cin >> value) is a reference to input stream. Its value is non-zero only if a value has been extracted from the input stream.



```cpp
#include<iostream>
using namespace std;
int main(){
    int keyValue = -1;
    float value;
    cout<<"Enter a number ";
    float sum {0};
    while (cin >> value)
        {
            if (value != keyValue)
            {
                cout<<"Entered number is : "<<value<<endl;
                sum+=value;
                cout<<"Sum till now : "<<sum<<endl;
                cout<<"Enter next number"<<endl;
            }
            else
            {
            cout<<"final sum is : "<<sum;
            break;
            }
        }
}
```



```
Enter a number 2
Entered number is : 2
Sum till now : 2
Enter next number
15
Entered number is : 15
Sum till now : 17
Enter next number
19
Entered number is : 19
Sum till now : 36
Enter next number
-100
Entered number is : -100
Sum till now : -64
Enter next number
78.456
Entered number is : 78.456
Sum till now : 14.456
Enter next number
-1
final sum is : 14.456
Process returned 0 (0x0)
Press any key to continue.
```

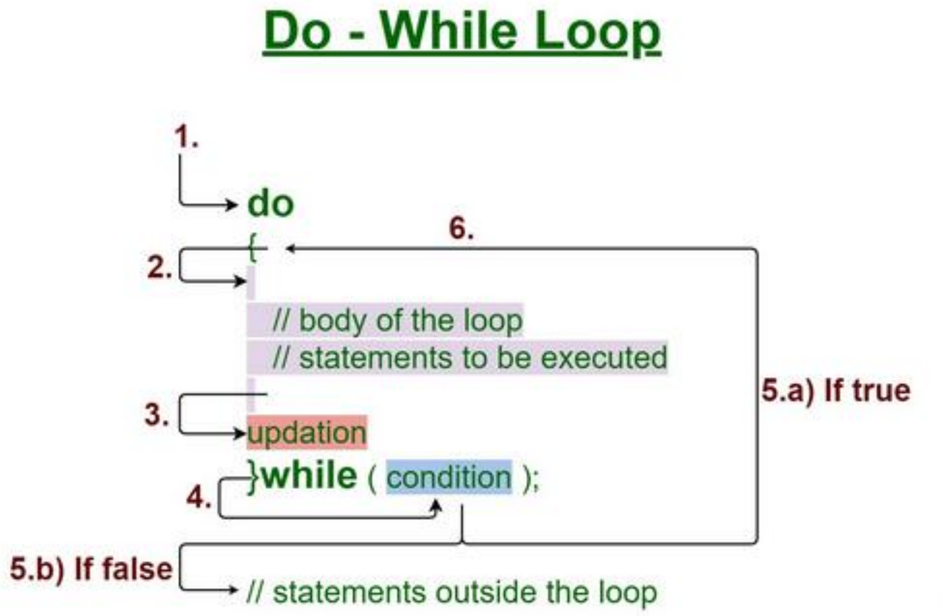
Do-while loop

- Loops come into use when we need to repeatedly execute a block of statements.
- Like while the do-while loop execution is also terminated on the basis of a test condition.
- The main difference between a do-while loop and a while loop is in the do-while loop, the condition is tested at the end of the loop body, i.e. do-while loop is exit controlled, whereas the other two loops are entry-controlled loops.

- **Note**: In the do-while loop, the loop body will execute at least once irrespective of the test condition.

How does a do-While loop execute?

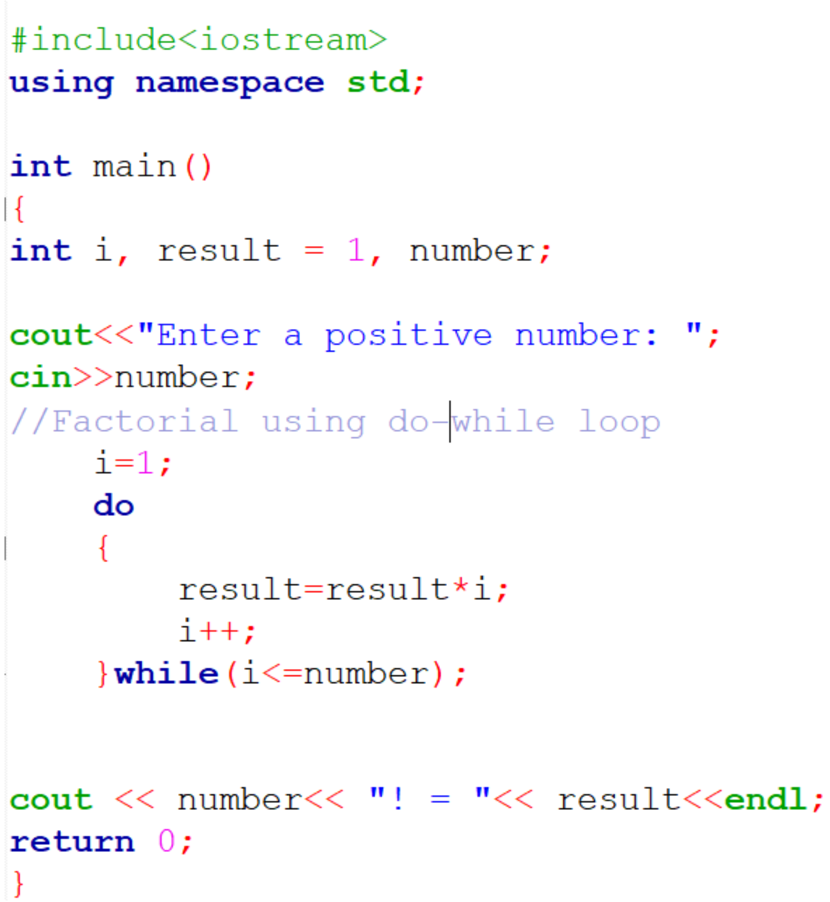
1. Control falls into the do-while loop.
2. The statements inside the body of the loop get executed.
3. Updation takes place.
4. The flow jumps to Condition
5. Condition is tested.
    1. If the Condition yields true, go to Step 6.
    2. If the Condition yields false, the flow goes outside the loop
6. The flow goes back to Step 2.
7. The do-while loop has been ended and flow has gone outside the loop.



```
do
{
  // body of the loop
  // statements to be executed

updation
}while ( condition );
// statements outside the loop
```

Example

- WAP to find the factorial of a number using a do-while loop.



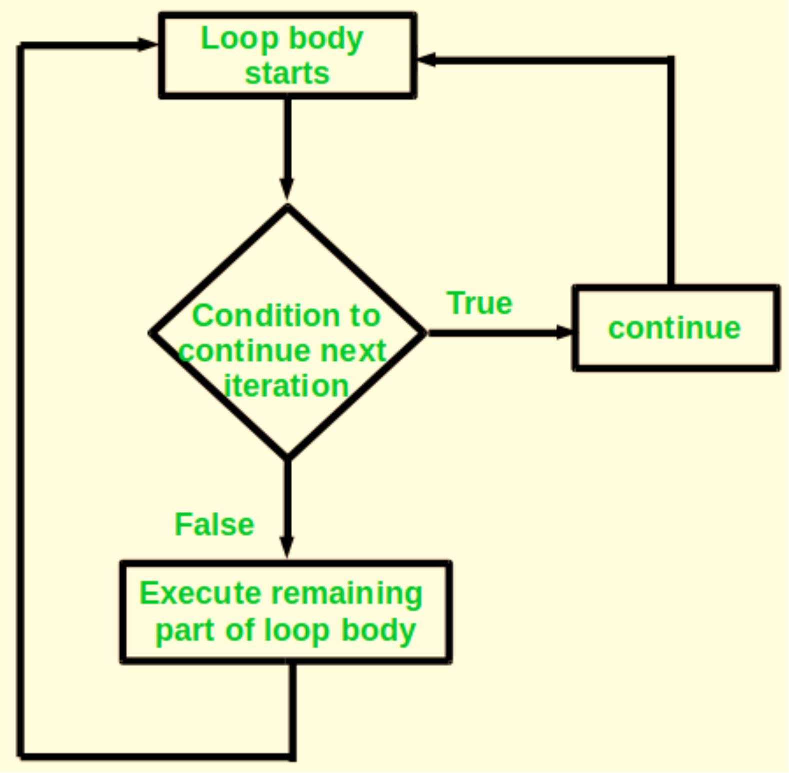
```
#include<iostream>
using namespace std;

int main()
{
int i, result = 1, number;

cout<<"Enter a positive number: ";
cin>>number;
//Factorial using do-while loop
    i=1;
    do
    {
        result=result*i;
        i++;
    }while(i<=number);


cout << number<< "! = "<< result<<endl;
return 0;
}
```

Jump statements in C++

- Jump statements are used to manipulate the flow of the program if some conditions are met.
- It is used to terminate or continue the loop inside a program or to stop the execution of a function.
- **Types of Jump Statements in C++**
  - continue
  - break
  - goto
  - return (will cover in functions)

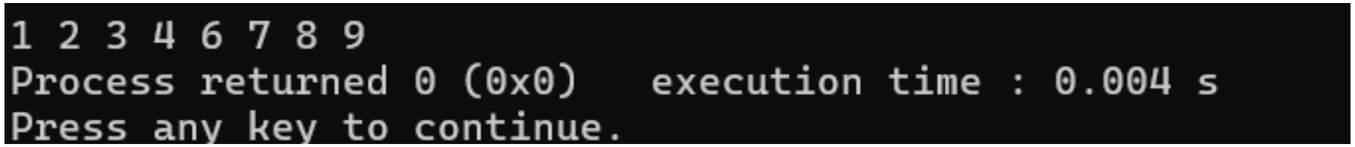
continue;

- The C++ continue statement is used to execute some parts of the loop while skipping some other parts.
- Rather than terminating the loop, it continues to execute the next iteration of the same loop.
- It is used with a decision-making statement which must be present inside the loop.
- This statement can be used inside for loop or while or do-while loop.


Loop body starts
Condition to continue next iteration
True
continue
False
Execute remaining part of loop body

Example

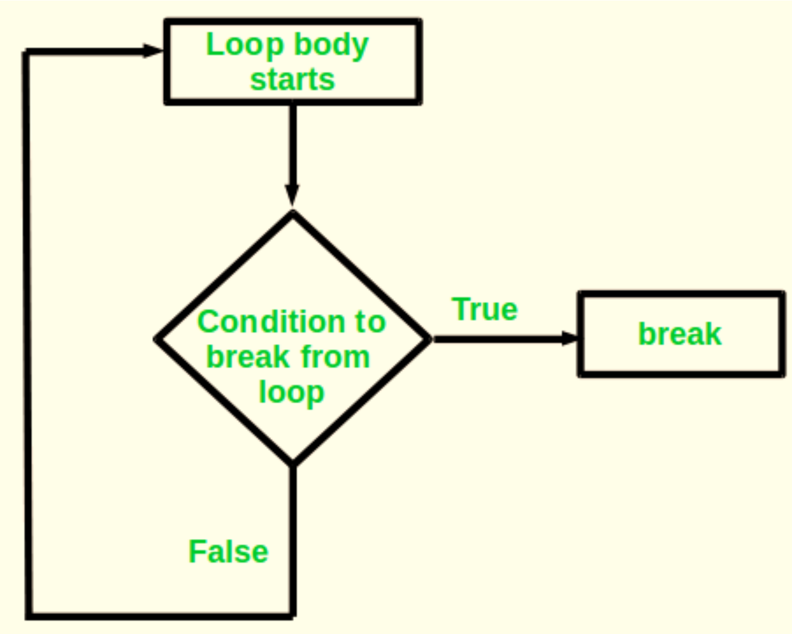
```cpp
// C++ program to demonstrate the
// continue statement
#include <iostream>
using namespace std;

int main()
{
    for (int i = 1; i < 10; i++) {

        if (i == 5)
            continue;
        cout << i << " ";
    }
    return 0;
}
```



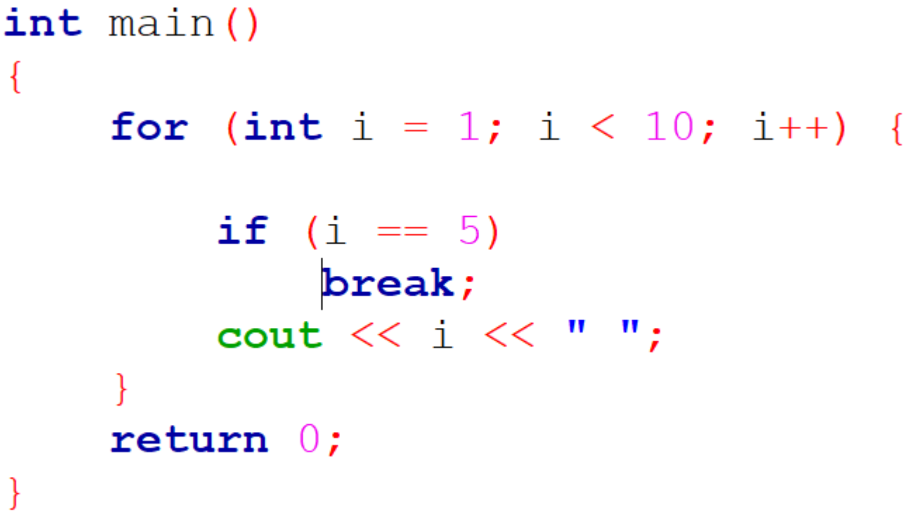
```
1 2 3 4 6 7 8 9
Process returned 0 (0x0)   execution time : 0.004 s
Press any key to continue.
```

break;

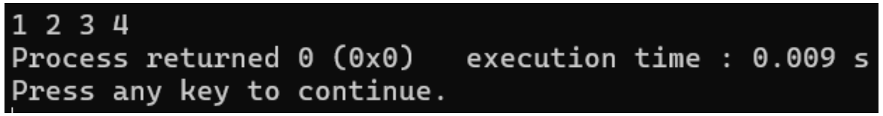
- The C++ break statement is used to terminate the whole loop if the condition is met.
- Unlike the continue statement after the condition is met, it breaks the loop and the remaining part of the loop is not executed.
- The break statement is used with decision-making statements such as if, if-else, or switch statements, which are inside a loop, It forces the loop to stop the execution of the further iteration.



Example



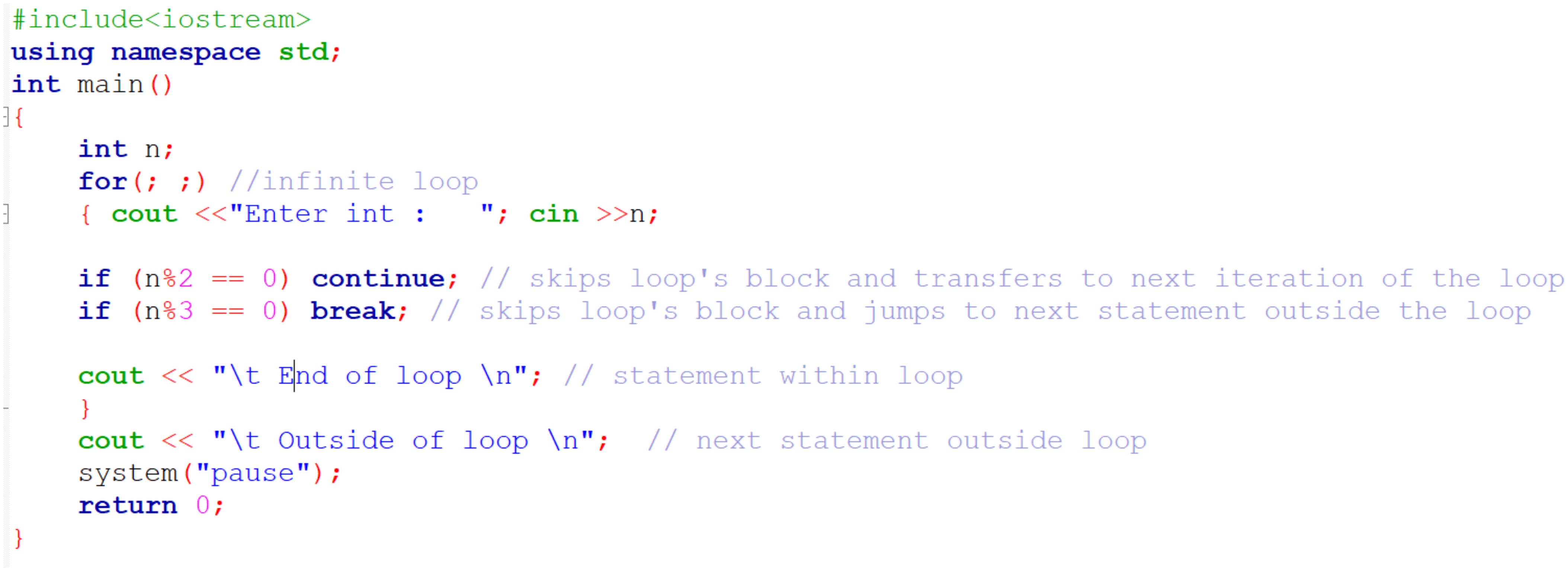
```
int main()
{
    for (int i = 1; i < 10; i++) {

        if (i == 5)
            break;
        cout << i << " ";
    }
    return 0;
}
```



```
1 2 3 4
Process returned 0 (0x0)   execution time : 0.009 s
Press any key to continue.
```

Example

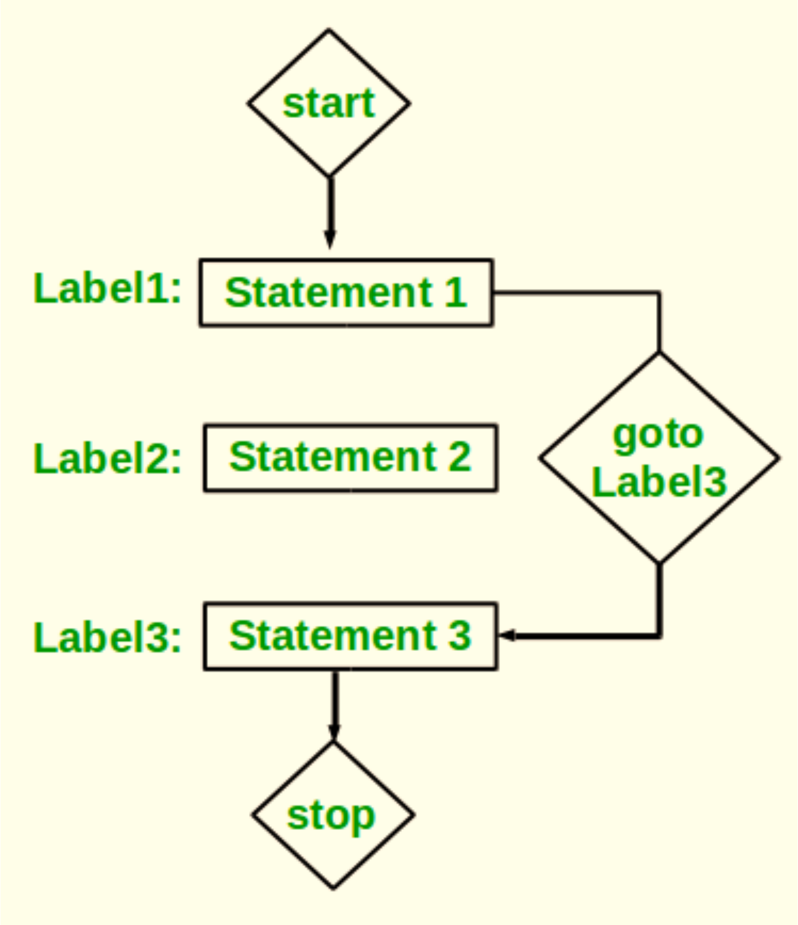
- What will be the output if the entered inputs are



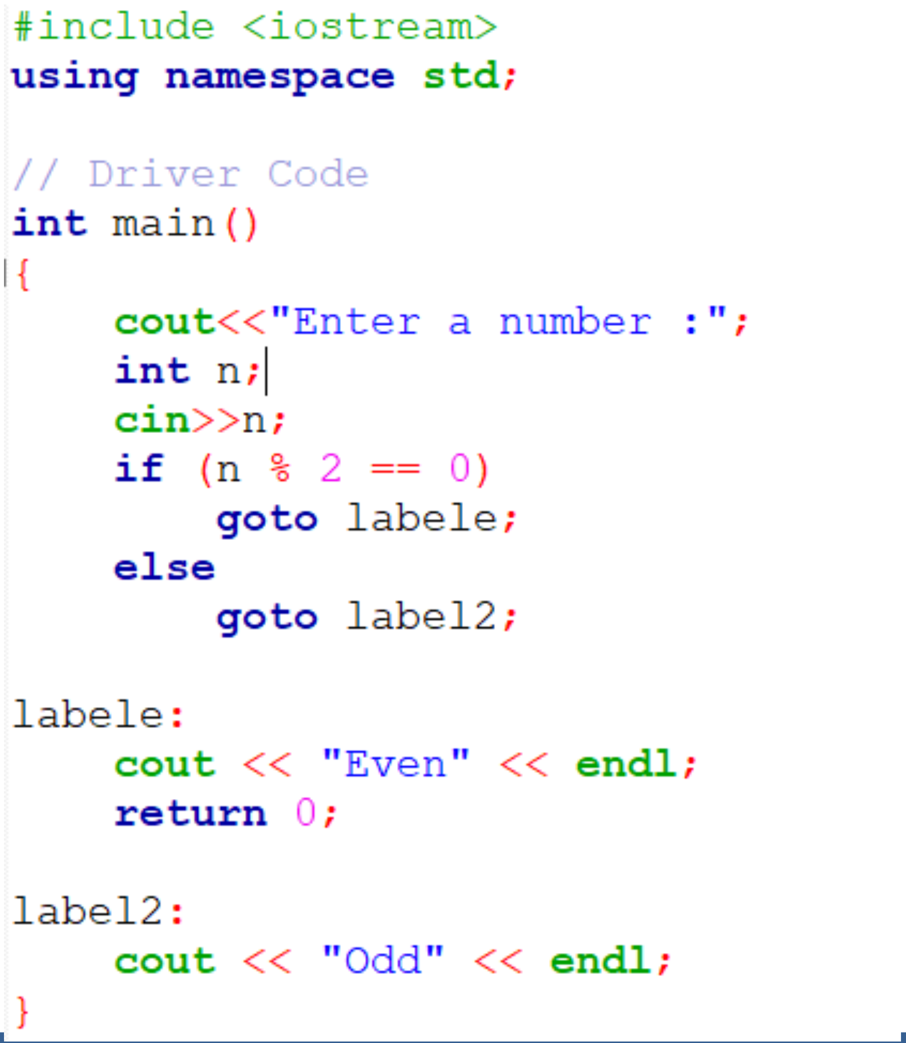
```
#include<iostream>
using namespace std;
int main()
{
    int n;
    for(; ;) //infinite loop
    { cout <<"Enter int :   "; cin >>n;

    if (n%2 == 0) continue; // skips loop's block and transfers to next iteration of the loop
    if (n%3 == 0) break; // skips loop's block and jumps to next statement outside the loop

    cout << "\t End of loop \n"; // statement within loop
    }
    cout << "\t Outside of loop \n";  // next statement outside loop
    system("pause");
    return 0;
}
```

Goto statement

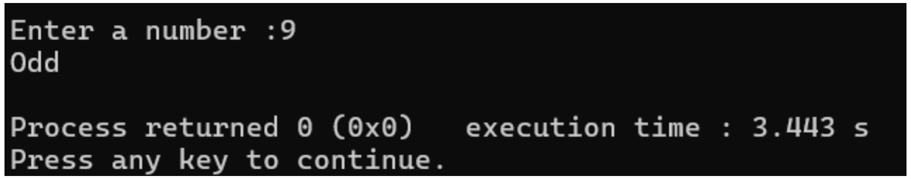
- The C++ goto statement is used to jump directly to that part of the program to which it is being called.
- Every goto statement is associated with the label which takes them to part of the program for which they are called.
- The label statements can be written anywhere in the program it is not necessary to use them before or after the goto statement.



Example



```cpp
#include <iostream>
using namespace std;

// Driver Code
int main()
{
    cout<<"Enter a number :";
    int n;
    cin>>n;
    if (n % 2 == 0)
        goto labele;
    else
        goto label2;

labele:
    cout << "Even" << endl;
    return 0;

label2:
    cout << "Odd" << endl;
}
```



```
Enter a number :9
Odd

Process returned 0 (0x0)   execution time : 3.443 s
Press any key to continue.
```