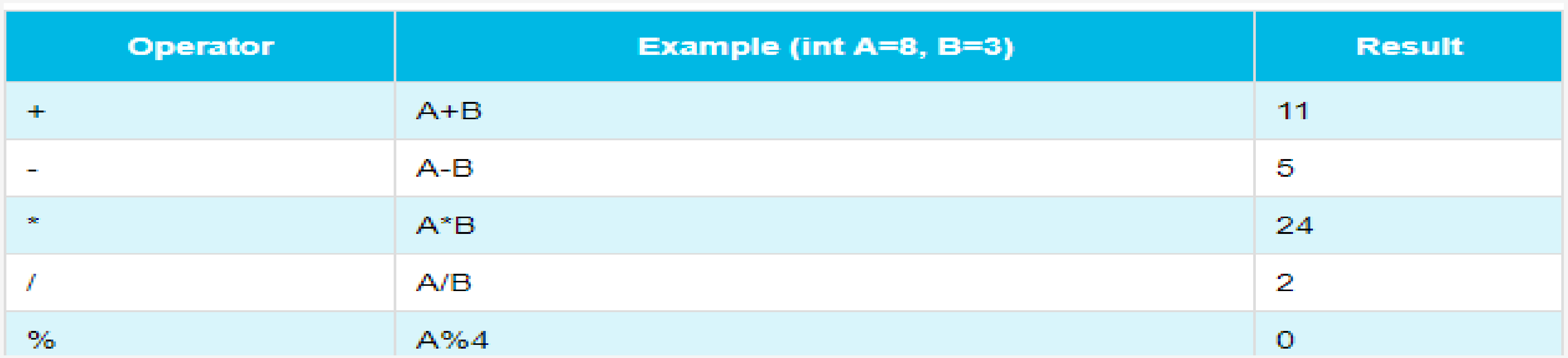
OPERATORS IN C++

What is an Operator?

-is a special symbol that tells the compiler to perform specific mathematical or logical Operation.

Arithmetic Operators

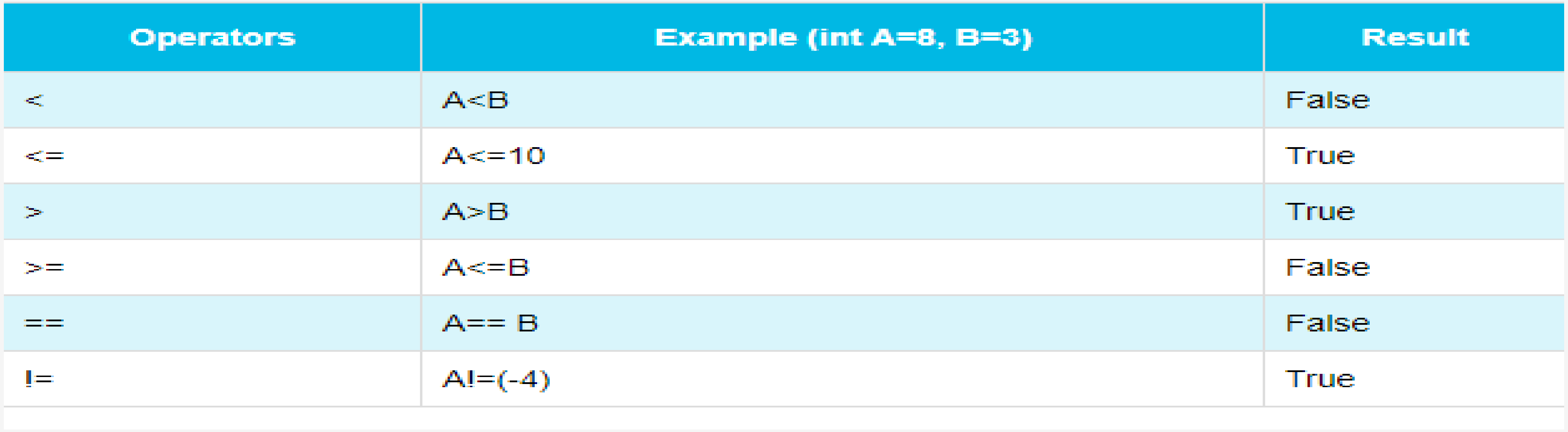
Given table shows all the Arithmetic operator supported by C Language. Lets suppose variable **A**hold 8 and **B** hold 3.



Operator	Example (int A=8, B=3)	Result
+	A+B	11
-	A-B	5
*	A*B	24
/	A/B	2
%	A%4	0

Relational Operators

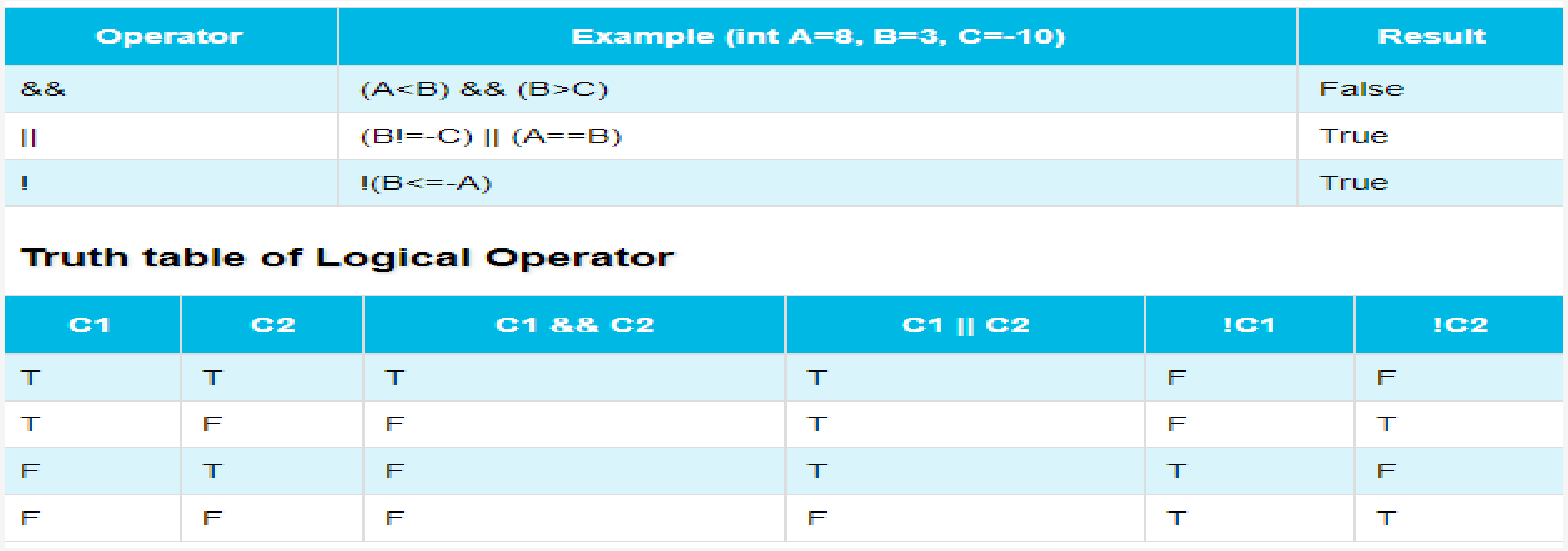
Which can be used to check the Condition, it always return true or false. Lets suppose variable **A** hold 8 and **B** hold 3.



Operators	Example (int A=8, B=3)	Result
<	A<B	False
<=	A<=10	True
>	A>B	True
>=	A<=B	False
==	A== B	False
!=	A!=(-4)	True

Logical Operator

Which can be used to combine more than one Condition?. Suppose you want to combined two conditions **A<B** and **B>C**, then you need to use **Logical Operator** like (A<B) && (B>C). Here **&&** is Logical Operator.



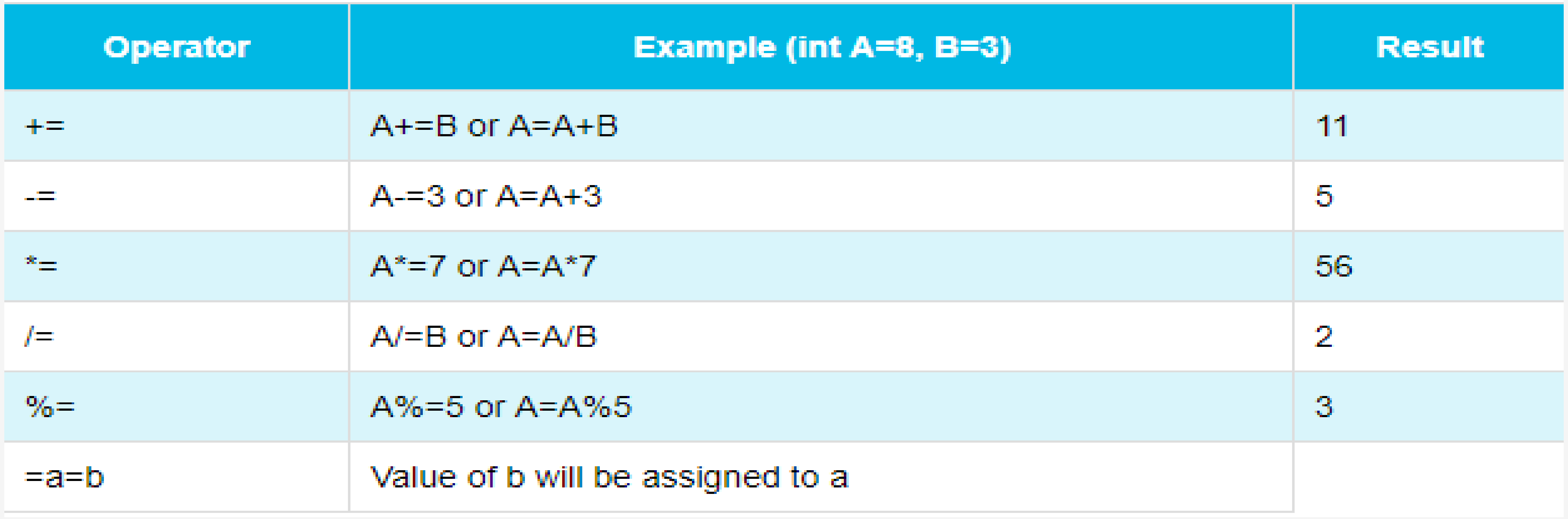
Operator	Example (int A=8, B=3, C=-10)	Result
&&	(A<B) && (B>C)	False
||	(B!=-C) || (A==B)	True
!	!(B<=-A)	True

Truth table of Logical Operator

C1	C2	C1 && C2	C1 || C2	!C1	!C2
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

Assignment operators

Which can be used to assign a value to a variable. Lets suppose variable **A** hold 8 and **B** hold 3.



Operator	Example (int A=8, B=3)	Result
+=	A+=B or A=A+B	11
-=	A-=3 or A=A+3	5
*=	A*=7 or A=A*7	56
/=	A/=B or A=A/B	2
%=	A%=5 or A=A%5	3
=a=b	Value of b will be assigned to a	

Increment and Decrement Operator in C++

- Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one.

- Both increment and decrement operator are used on single operand or variable, so it is called as unary operator. Unary operators are having higher priority than the other operators it means unary operators are execute before other operators.

Syntax

```
++   // increment operator
--   // decrement operator
```

Note: Increment and decrement operators are can not apply on constant.

Example

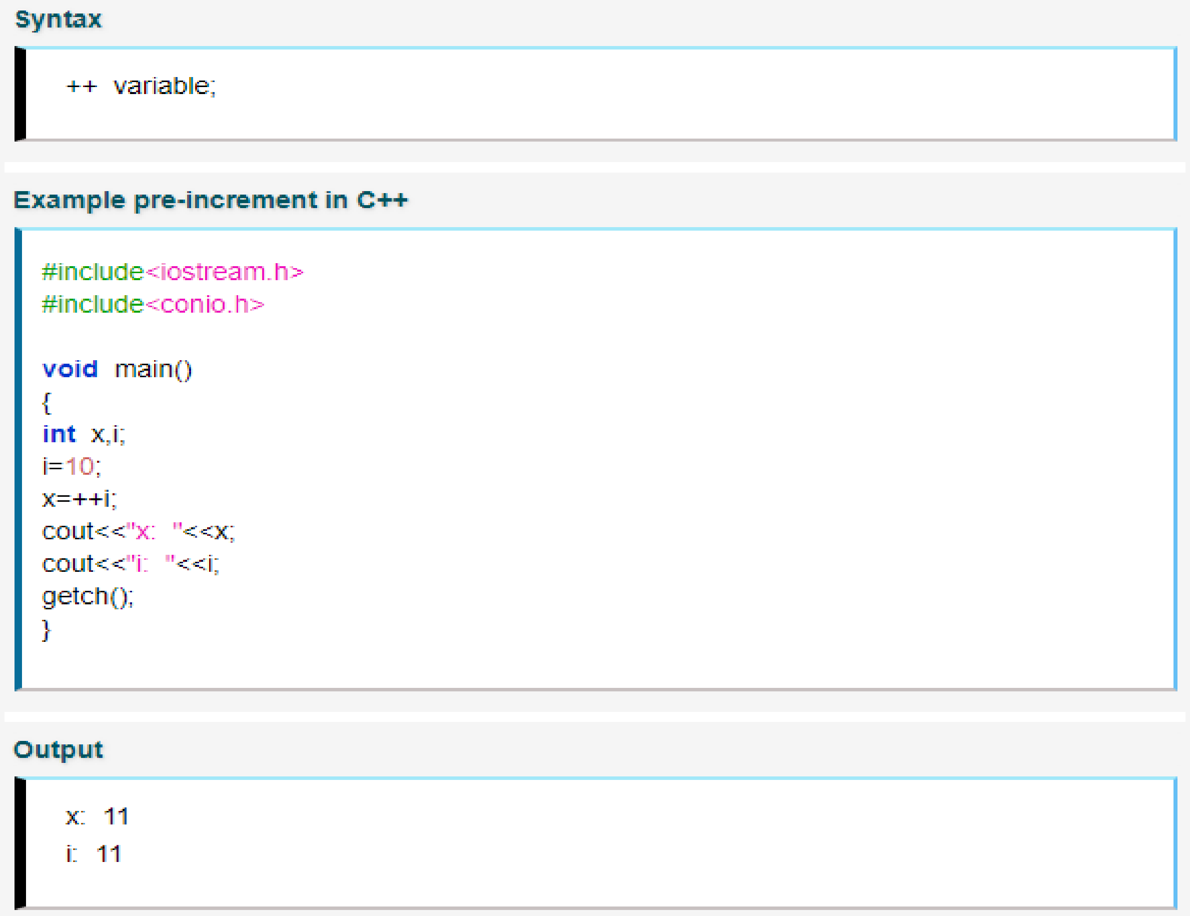
```
x= 4++;   // gives error, because 4 is constant
```

Type of Increment Operator

- pre-increment
- post-increment

pre-increment (++ variable)

In pre-increment first increment the value of variable and then used inside the expression (initialize into another variable).



Syntax

```
++ variable;
```

Example pre-increment in C++

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

void main()
{
int x,i;
i=10;
x=++i;
cout<<"x: "<<x;
cout<<"i: "<<i;
getch();
}
```

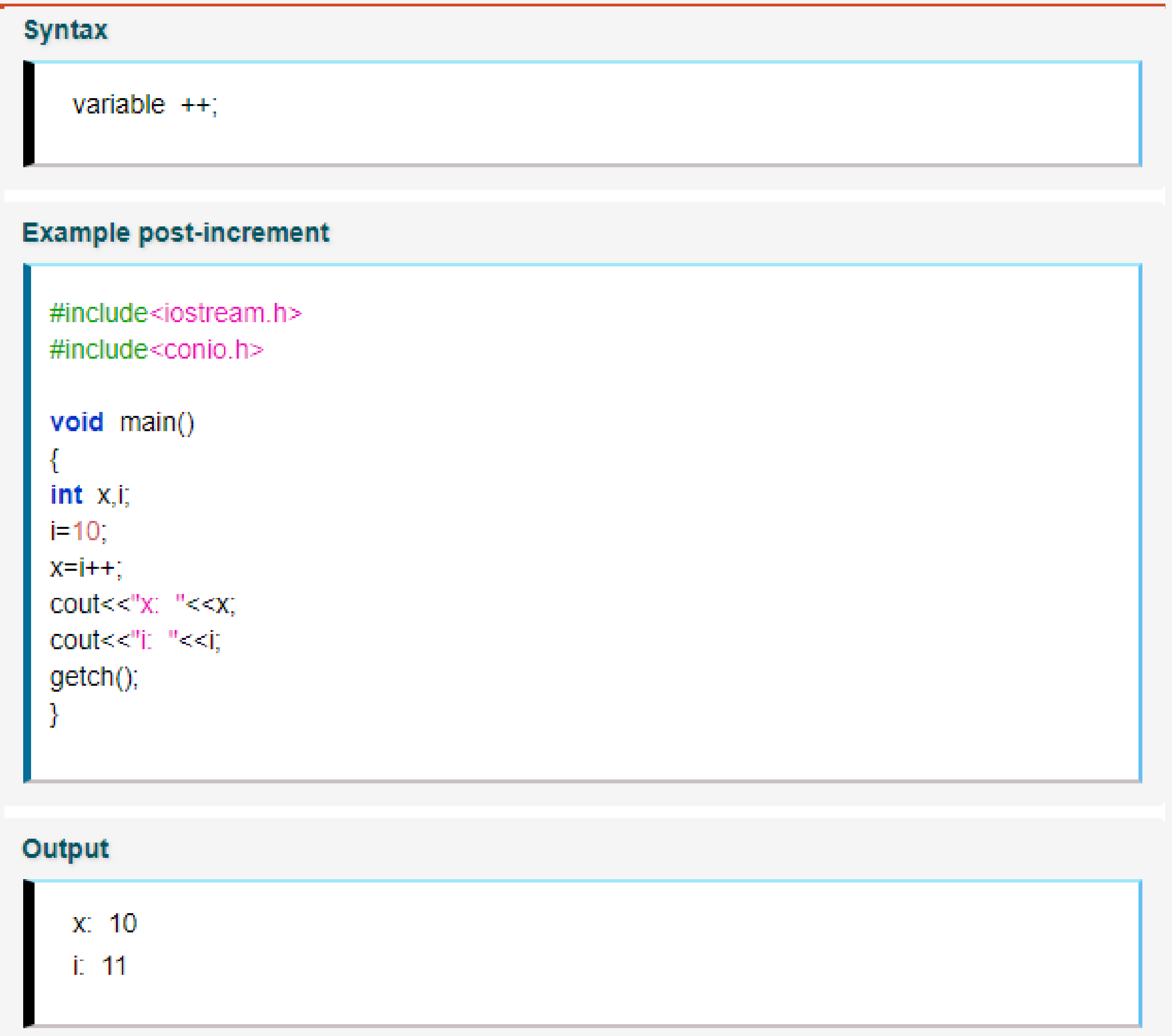
Output

```
x: 11
i: 11
```

In above program first increase the value of i and then used value of i into expression

Post-increment (variable ++)

In post-increment first value of variable is use in the expression (initialize into another variable) and then increment the value of variable.



Syntax

```
variable ++;
```

Example post-increment

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

void main()
{
int x,i;
i=10;
x=i++;
cout<<"x: "<<x;
cout<<"i: "<<i;
getch();
}
```

Output

```
x: 10
i: 11
```

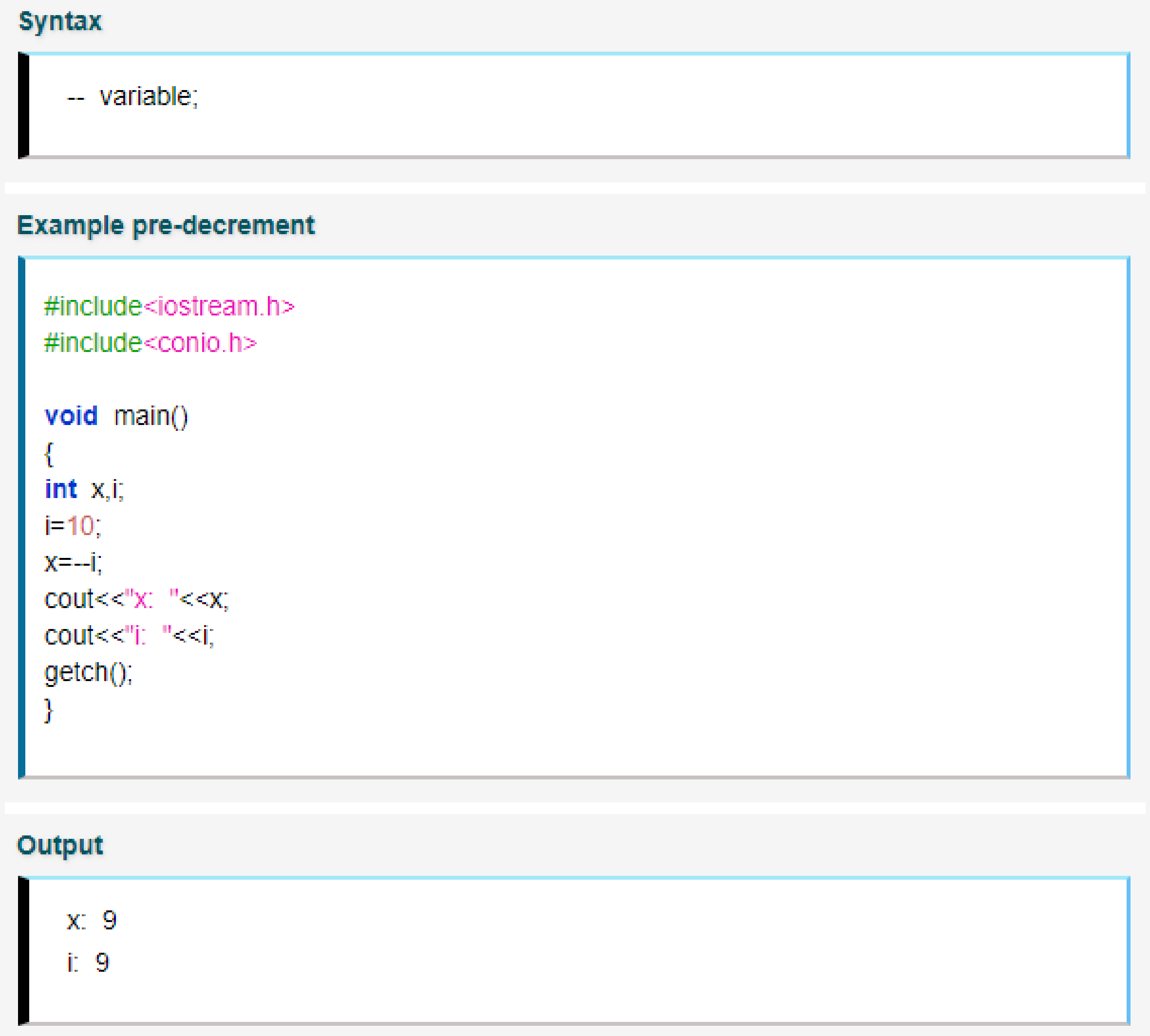
In above program first used the value of i into expression then increase value of i by 1.

Type of Decrement Operator

- pre-decrement
- post-decrement

Pre-decrement (-- variable)

In pre-decrement first decrement the value of variable and then used inside the expression (initialize into another variable).



Syntax

```
-- variable;
```

Example pre-decrement

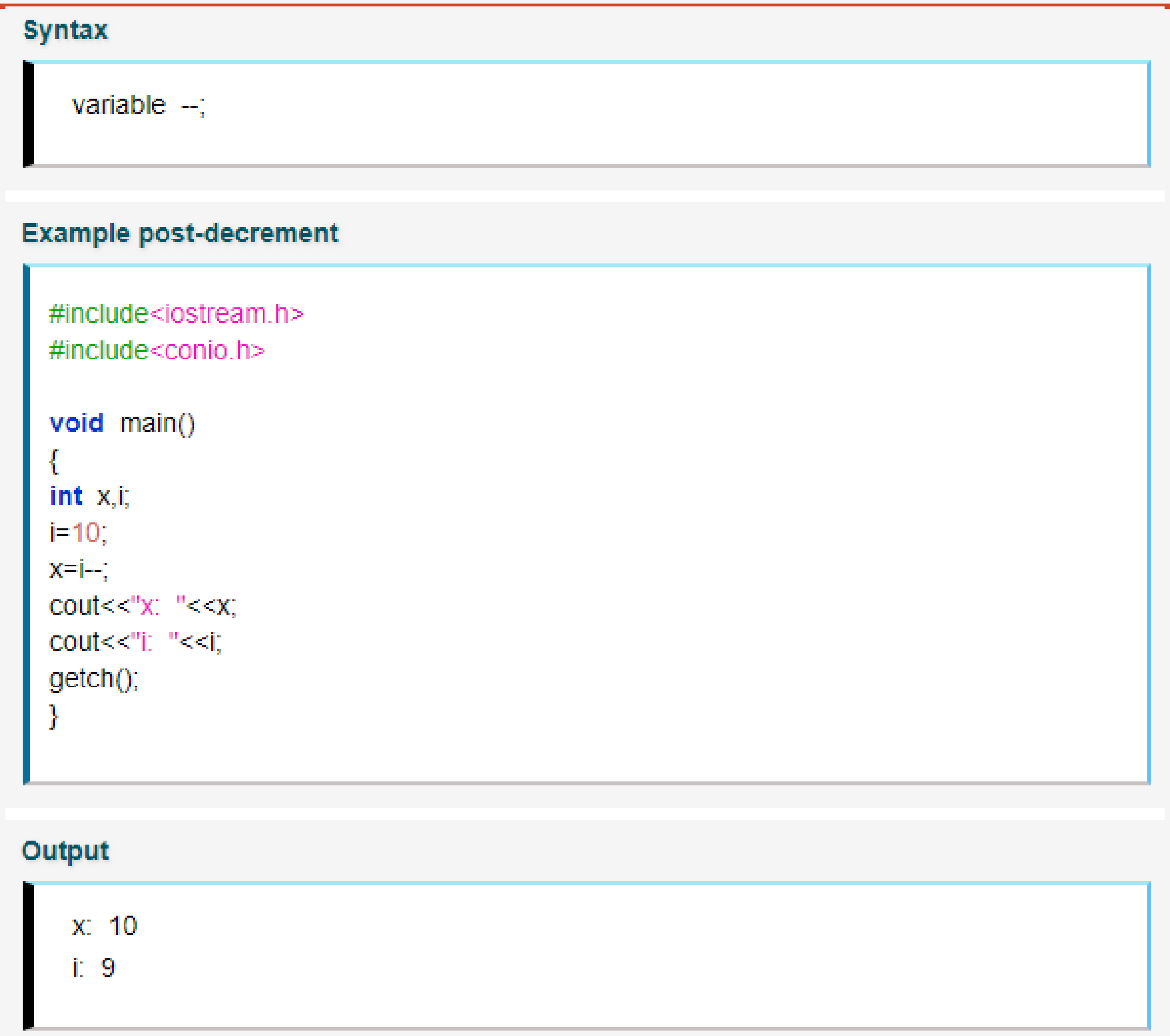
```
#include<iostream.h>
#include<conio.h>

void main()
{
int x,i;
i=10;
x=--i;
cout<<"x:  "<<x;
cout<<"i:  "<<i;
getch();
}
```

Output

```
x:  9
i:  9
```

post-decrement (variable --)

In Post-decrement first value of variable is use in the expression (initialize into another variable) and then decrement the value of variable.



Syntax

```
variable --;
```

Example post-decrement

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

void main()
{
int x,i;
i=10;
x=i--;
cout<<"x: "<<x;
cout<<"i: "<<i;
getch();
}
```

Output

```
x: 10
i: 9
```

In above program first used the value of x in expression then decrease value of i by 1.

DECISION MAKING STATEMENT

Decision Making Statement

Decision making statement is depending on the condition block need to be executed or not which is decided by condition. If the condition is "true" statement block will be executed, if condition is "false" then statement block will not be executed.

These are the following types of decision making statement.

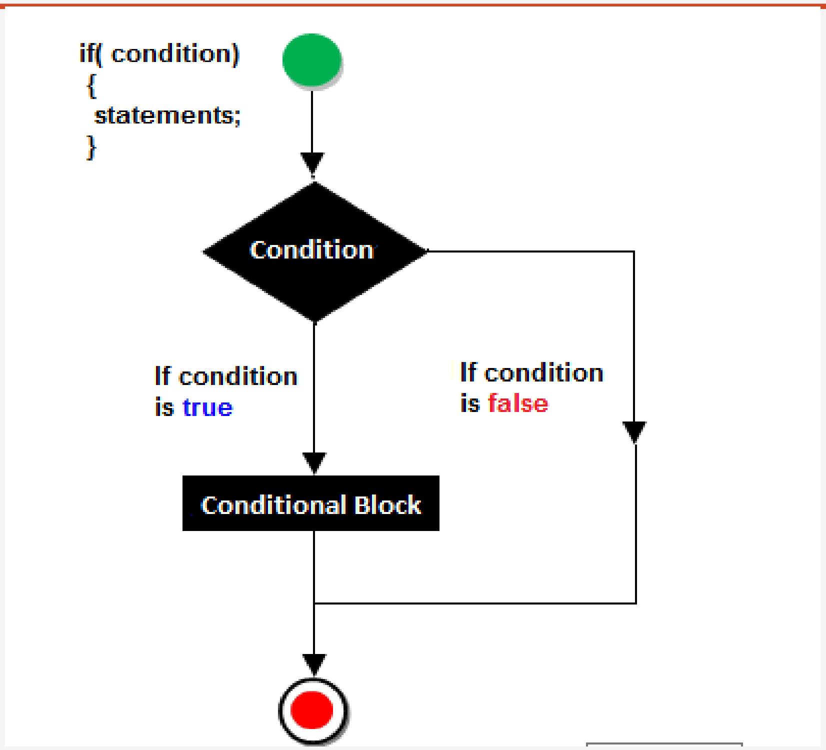
- if
- if-else
- if –else if
- Nested if
- Switch
- Nested switch

if Statement

if-then is most basic statement of Decision making statement. It tells to program to execute a certain part of code only if particular condition is true.

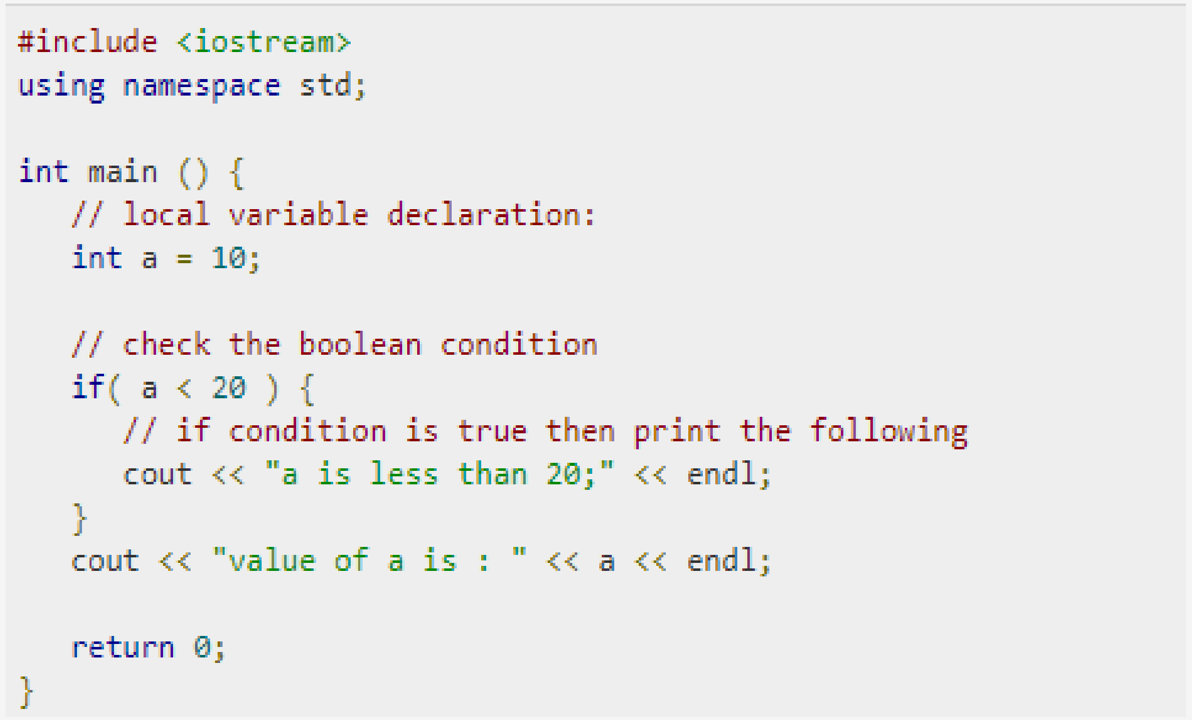
Syntax

```
If(condition) //Boolean expression
{
..........
//code to be executed
..........
}
```



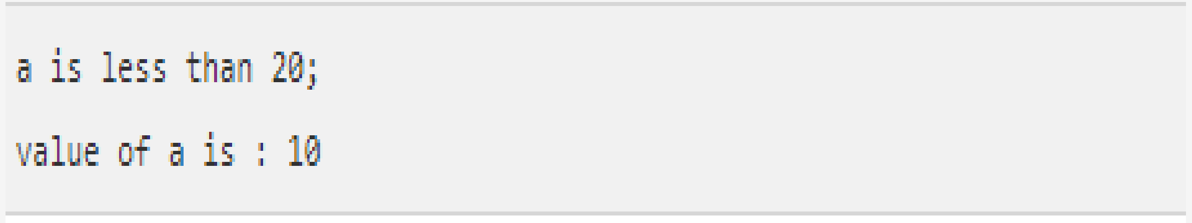
•Constructing the body of "if" statement is always optional, Create the body when we are having multiple statements.
•For a single statement, it is not required to specify the body.
•If the body is not specified, then automatically condition part will be terminated with next semicolon (;).

If Statement Examples



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

int main () {
   // local variable declaration:
   int a = 10;

   // check the boolean condition
   if( a < 20 ) {
      // if condition is true then print the following
      cout << "a is less than 20;" << endl;
   }
   cout << "value of a is : " << a << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result –



```
a is less than 20;

value of a is : 10
```

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

int main ()
{
  int grade = 85;
        if (grade >= 75)
        {
           cout<<"Remarks: Passed";
        }
  return 0;
}
```

Output

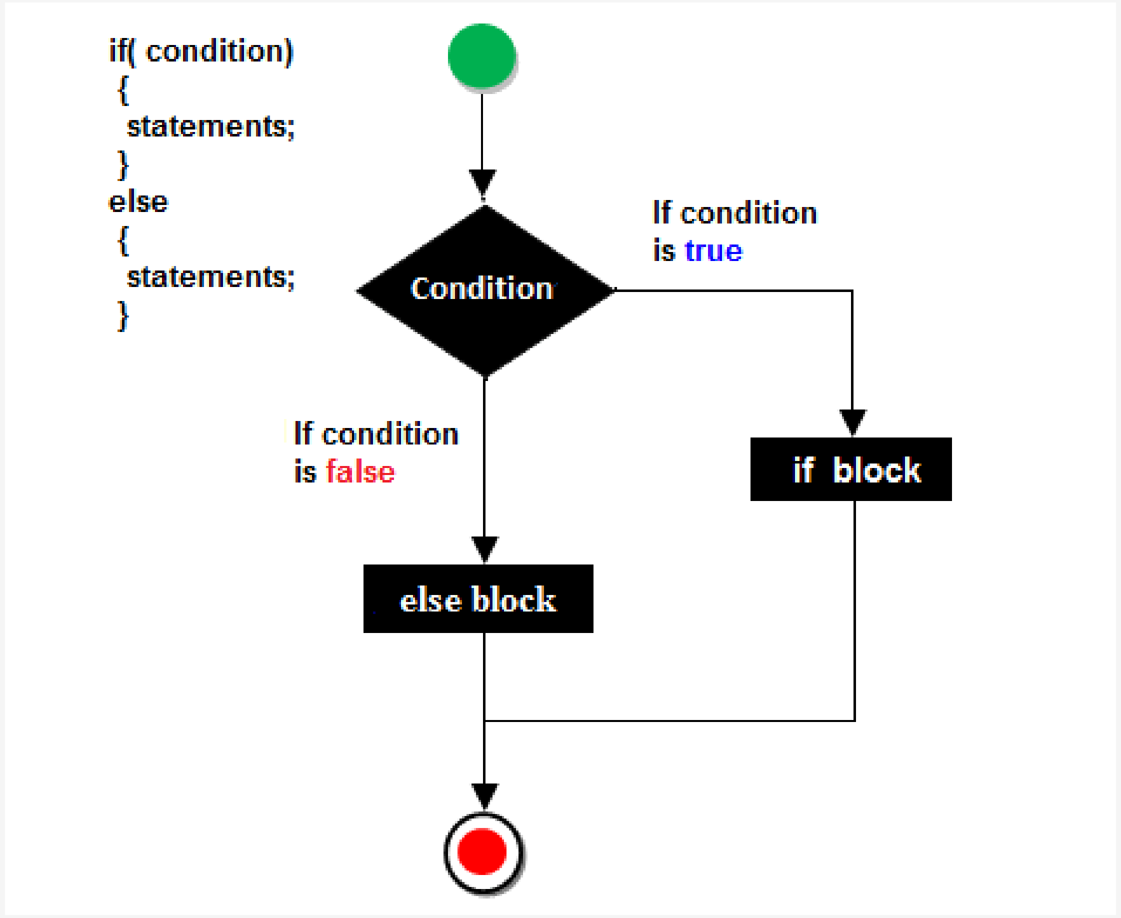
Remarks: Passed

If-else statement

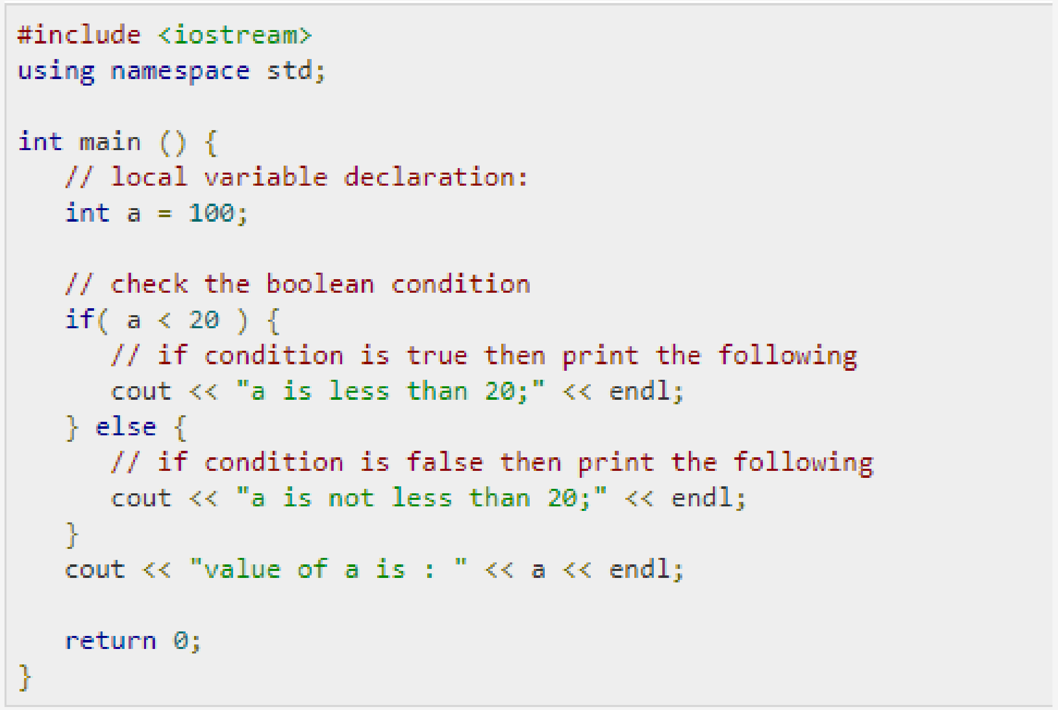
Else - It is a keyword, by using this keyword we can create a alternative block for "if" part. Using else is always optional i.e, it is recommended to use when we are having alternate block of condition.

In any program among if and else only one block will be executed. When if condition is false then else part will be executed, if part is executed then automatically else part will be ignored.
An ‘if’ statement can be followed by an optional ‘else’ statement, which executes when the Boolean expression is false. It executes if block if condition is true otherwise else block is executed.

Syntax and Diagram


```
if( condition)
 {
  statements;
 }
else
 {
  statements;
 }
```

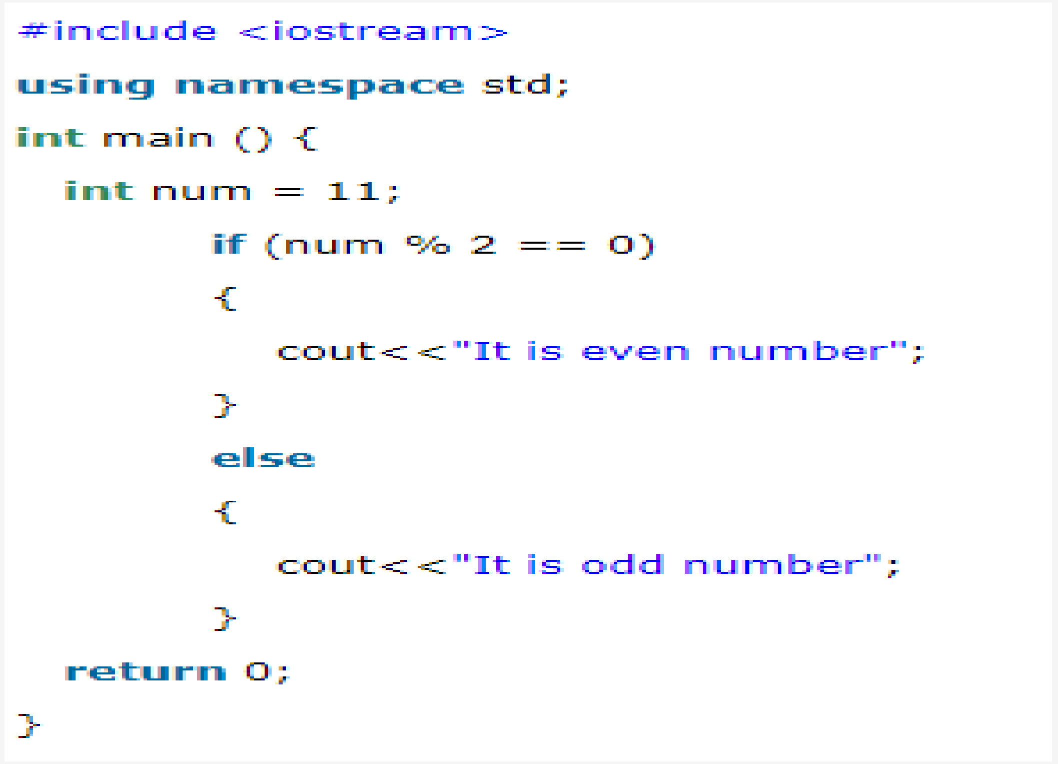
Examples



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

int main () {
   // local variable declaration:
   int a = 100;

   // check the boolean condition
   if( a < 20 ) {
      // if condition is true then print the following
      cout << "a is less than 20;" << endl;
   } else {
      // if condition is false then print the following
      cout << "a is not less than 20;" << endl;
   }
   cout << "value of a is : " << a << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
a is not less than 20;

value of a is : 100
```



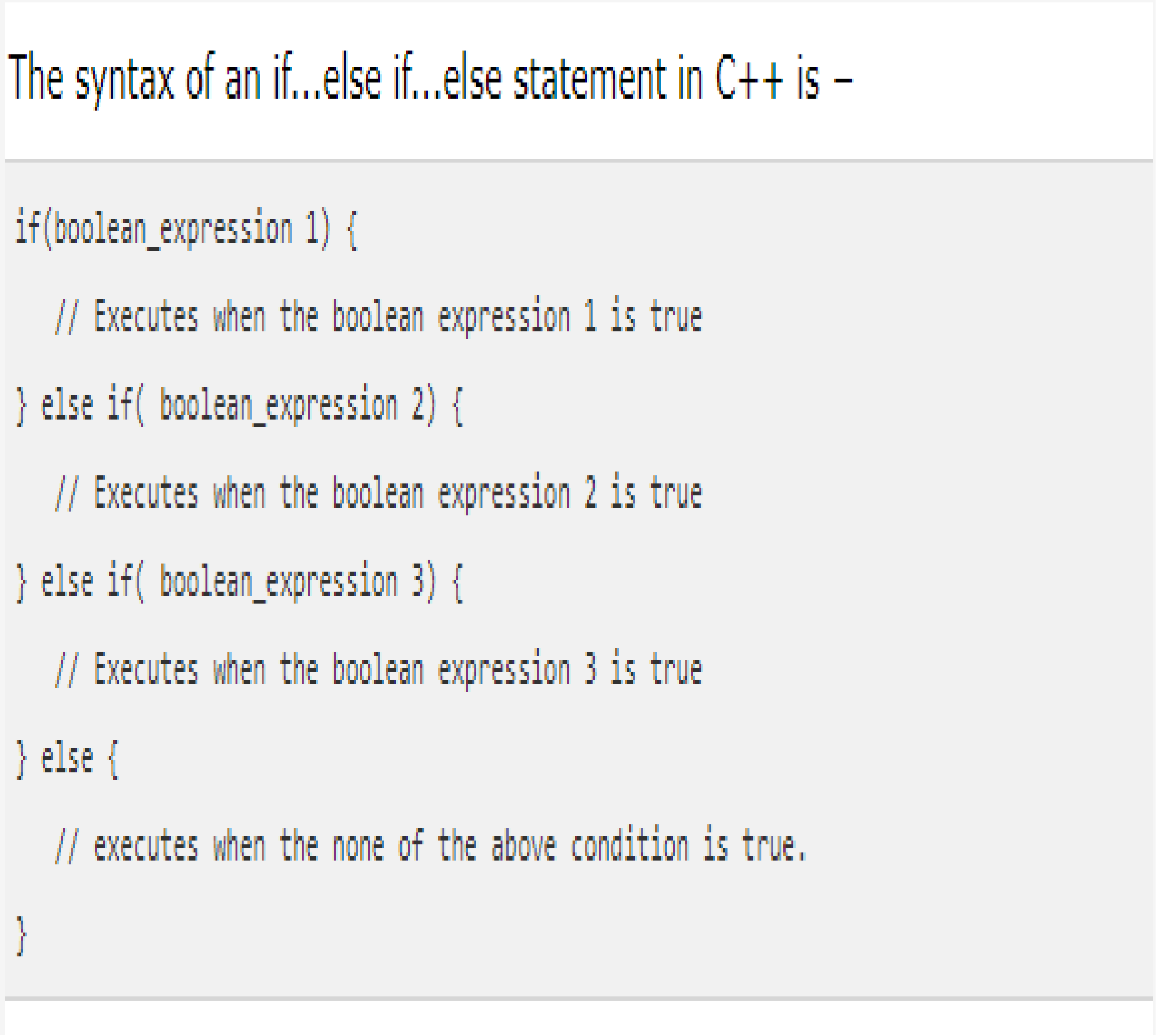
```
#include <iostream>
using namespace std;
int main () {
   int num = 11;
            if (num % 2 == 0)
            {
                cout<<"It is even number";
            }
            else
            {
                cout<<"It is odd number";
            }
   return 0;
}
```

Output:

```
It is odd number
```

if...else if...else Statement

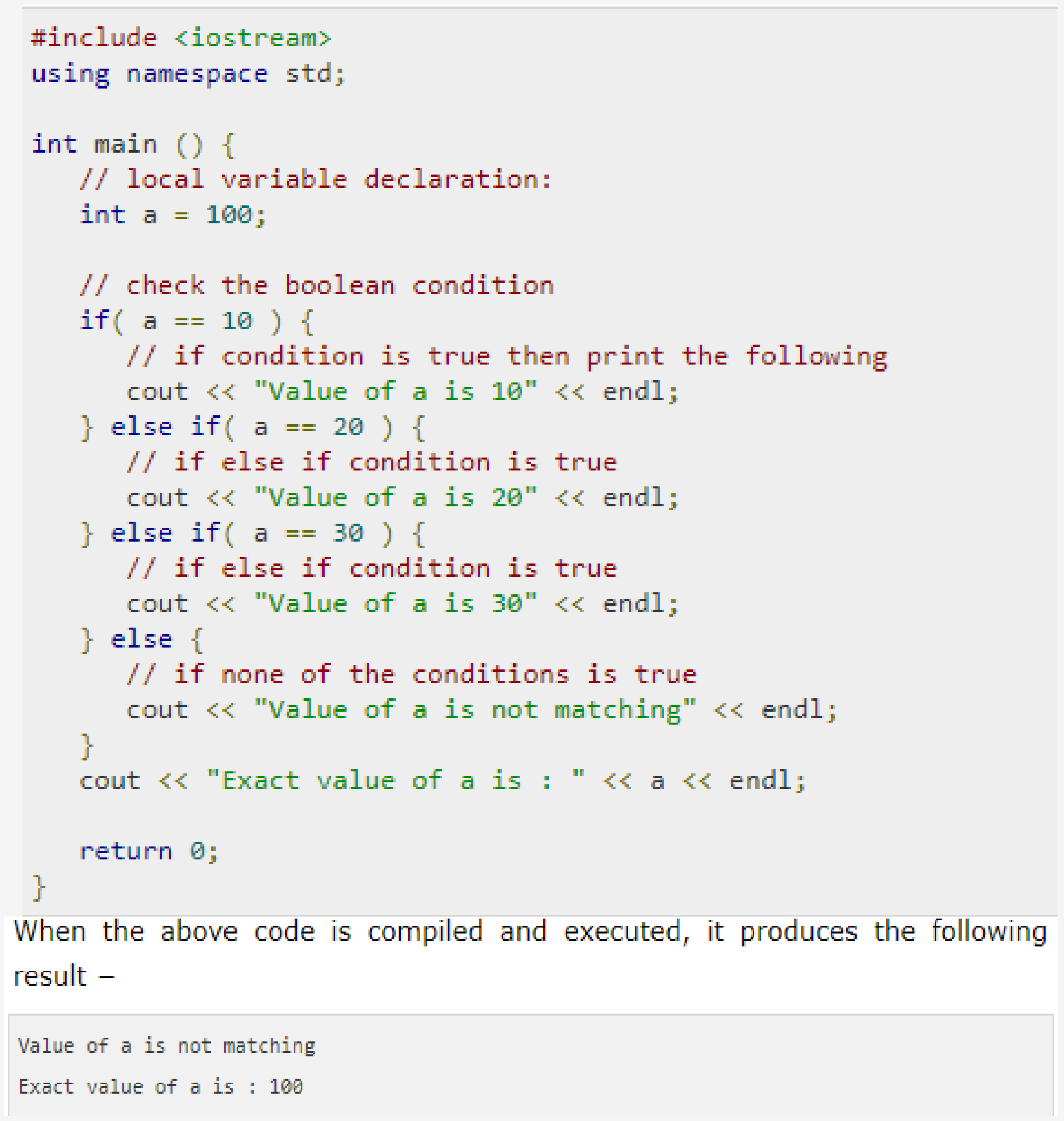
An **if** statement can be followed by an optional **else if...else** statement, which is very useful to test various conditions using single if...else if statement.



The syntax of an if...else if...else statement in C++ is −

```
if(boolean_expression 1) {
   // Executes when the boolean expression 1 is true
} else if( boolean_expression 2) {
   // Executes when the boolean expression 2 is true
} else if( boolean_expression 3) {
   // Executes when the boolean expression 3 is true
} else {
   // executes when the none of the above condition is true.
}
```

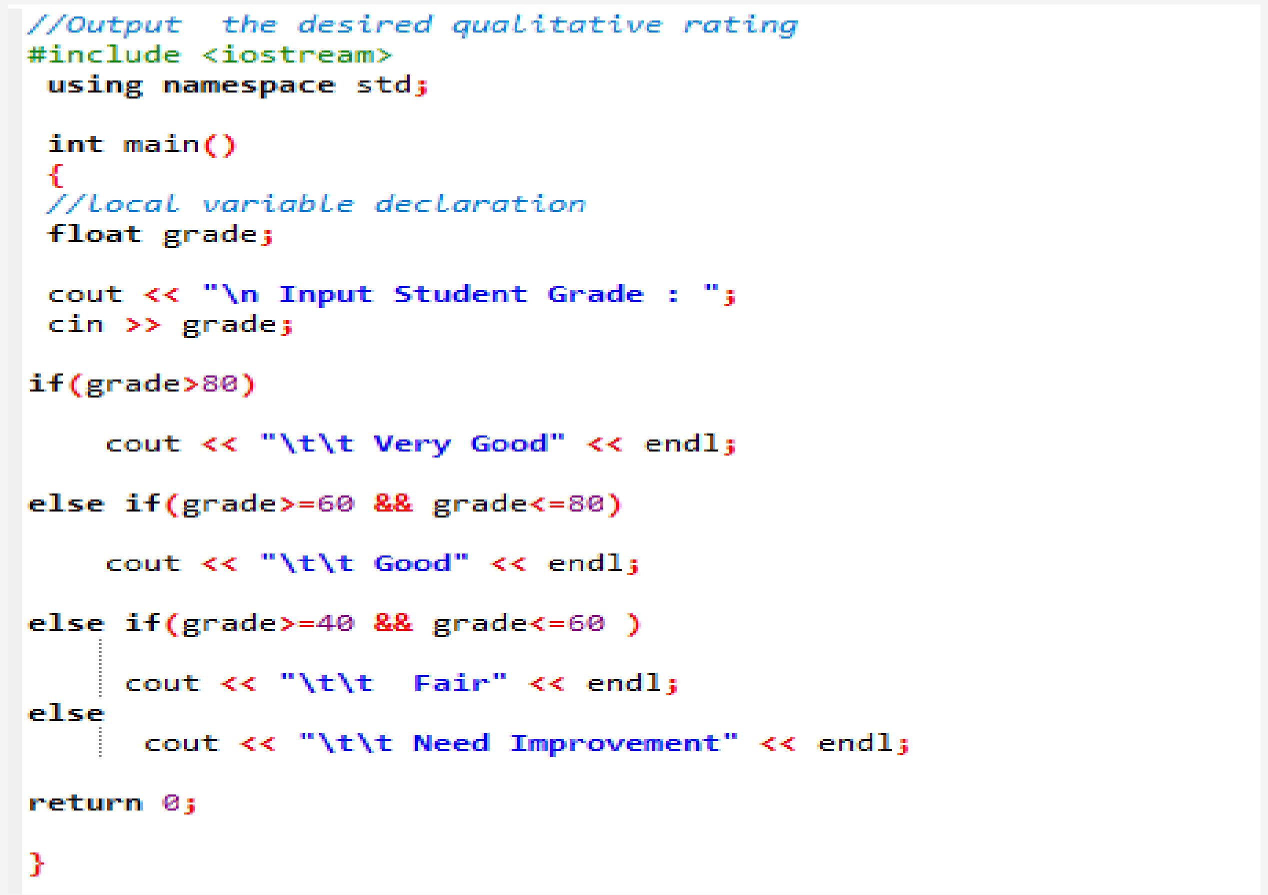
Examples



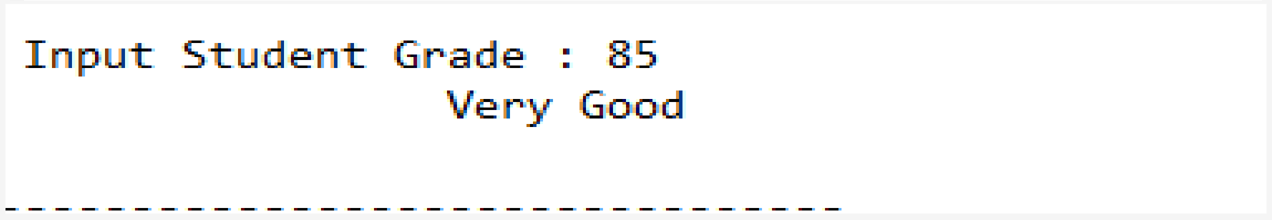
```cpp
#include <iostream>
using namespace std;

int main () {
   // local variable declaration:
   int a = 100;

   // check the boolean condition
   if( a == 10 ) {
      // if condition is true then print the following
      cout << "Value of a is 10" << endl;
   } else if( a == 20 ) {
      // if else if condition is true
      cout << "Value of a is 20" << endl;
   } else if( a == 30 ) {
      // if else if condition is true
      cout << "Value of a is 30" << endl;
   } else {
      // if none of the conditions is true
      cout << "Value of a is not matching" << endl;
   }
   cout << "Exact value of a is : " << a << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
Value of a is not matching
Exact value of a is : 100
```



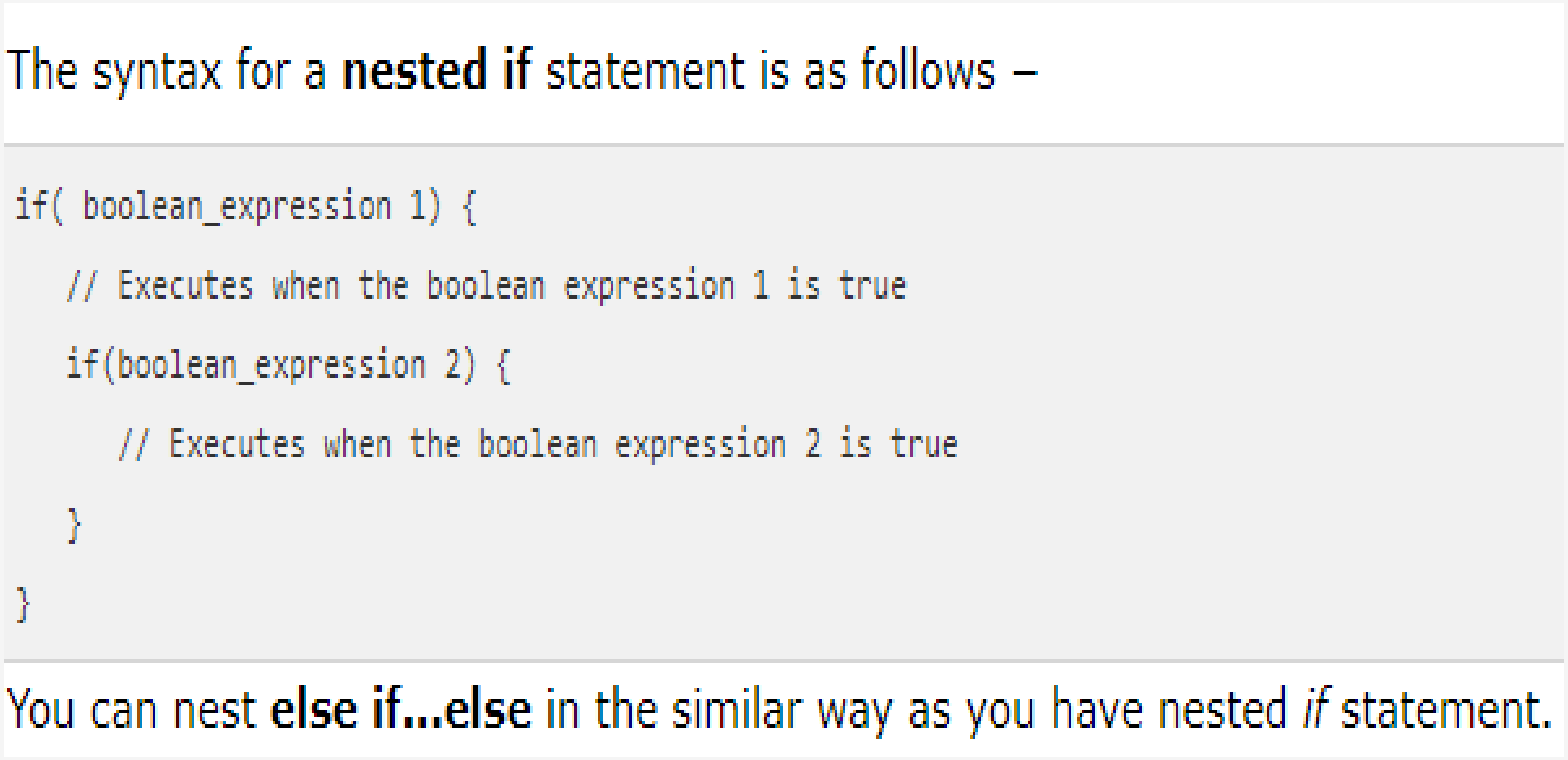
```cpp
//Output  the desired qualitative rating
#include <iostream>
 using namespace std;

 int main()
 {
 //local variable declaration
 float grade;

 cout << "\n Input Student Grade : ";
 cin >> grade;

if(grade>80)

    cout << "\t\t Very Good" << endl;

else if(grade>=60 && grade<=80)

    cout << "\t\t Good" << endl;

else if(grade>=40 && grade<=60 )

     cout << "\t\t  Fair" << endl;
else
      cout << "\t\t Need Improvement" << endl;

return 0;

}
```



```
Input Student Grade : 85
                     Very Good
```

Nested if statements

It is always legal to **nest** if-else statements, which means you can use one if or else if statement inside another if or else if statement(s).

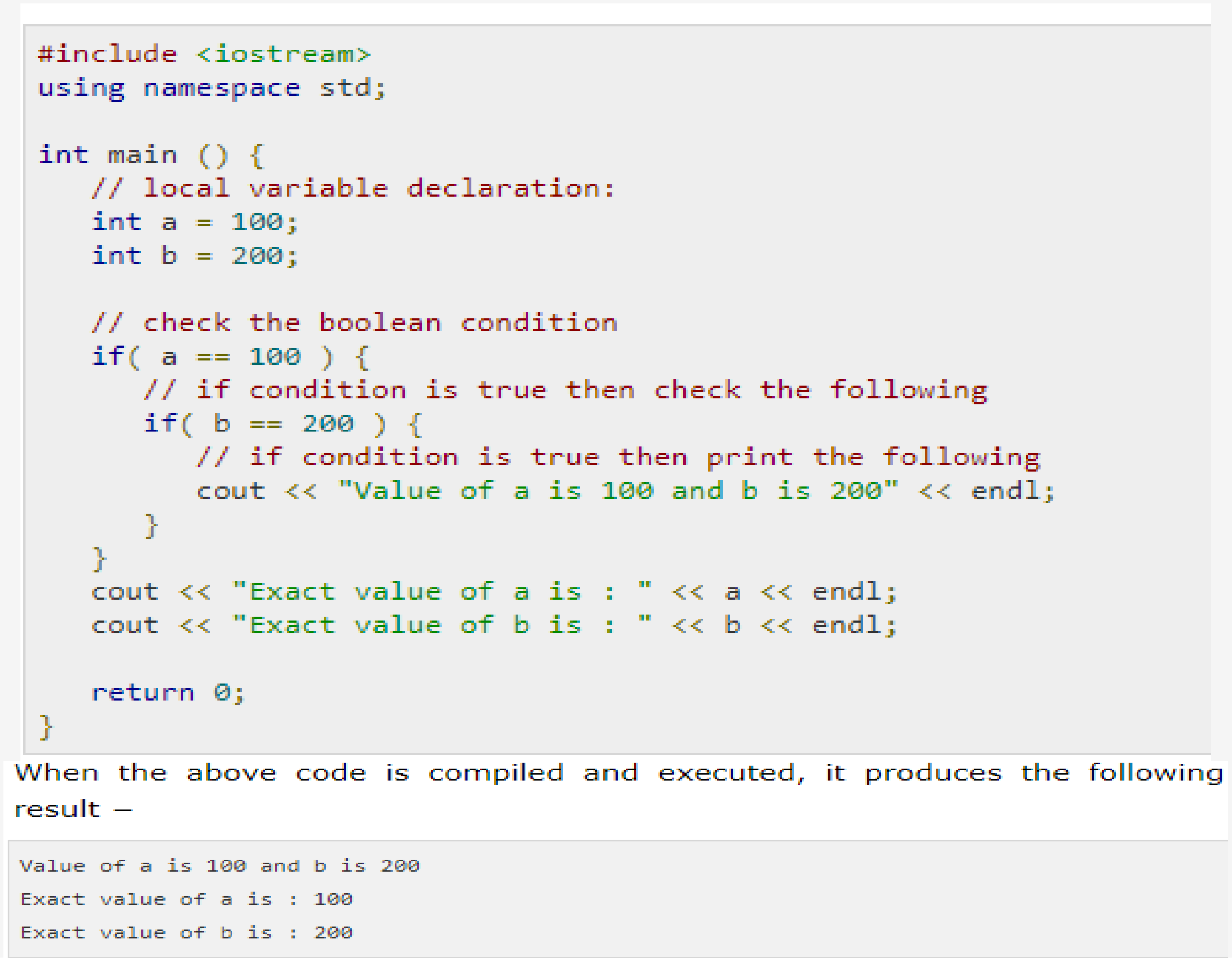


The syntax for a **nested if** statement is as follows −

```
if( boolean_expression 1) {
   // Executes when the boolean expression 1 is true
   if(boolean_expression 2) {
      // Executes when the boolean expression 2 is true
   }
}
```

You can nest **else if...else** in the similar way as you have nested *if* statement.

Example



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

int main () {
   // local variable declaration:
   int a = 100;
   int b = 200;

   // check the boolean condition
   if( a == 100 ) {
      // if condition is true then check the following
      if( b == 200 ) {
         // if condition is true then print the following
         cout << "Value of a is 100 and b is 200" << endl;
      }
   }
   cout << "Exact value of a is : " << a << endl;
   cout << "Exact value of b is : " << b << endl;

   return 0;
}
```

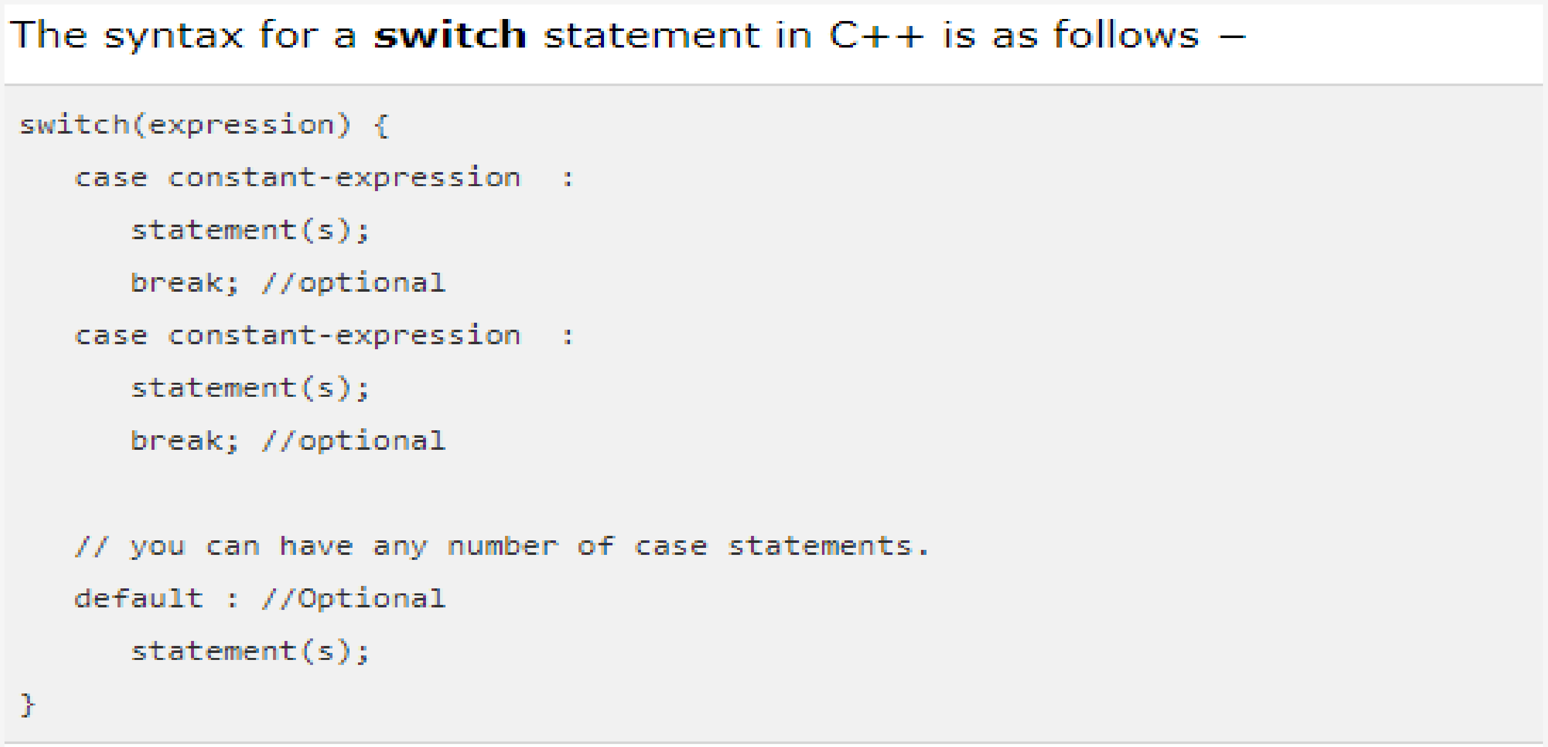
When the above code is compiled and executed, it produces the following result −

```
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```

SWITCH STATEMENT

switch statement

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.



The syntax for a **switch** statement in C++ is as follows −

```
switch(expression) {
   case constant-expression  :
      statement(s);
      break; //optional
   case constant-expression  :
      statement(s);
      break; //optional

   // you can have any number of case statements.
   default : //Optional
      statement(s);
}
```

The following rules apply to a switch statement

•You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.

•The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.

•When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.

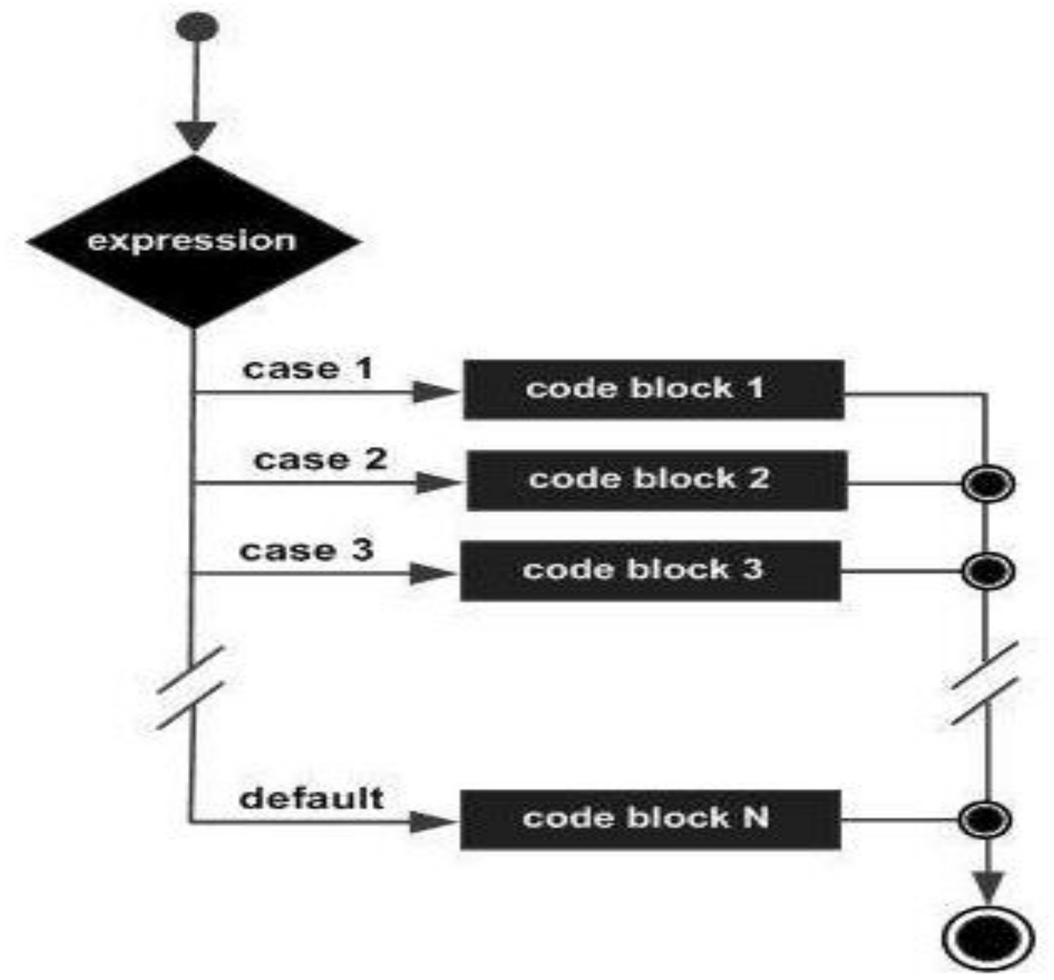
•When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.

•Not every case needs to contain a break. If no break appears, the flow of control will *fall through* to subsequent cases until a break is reached.

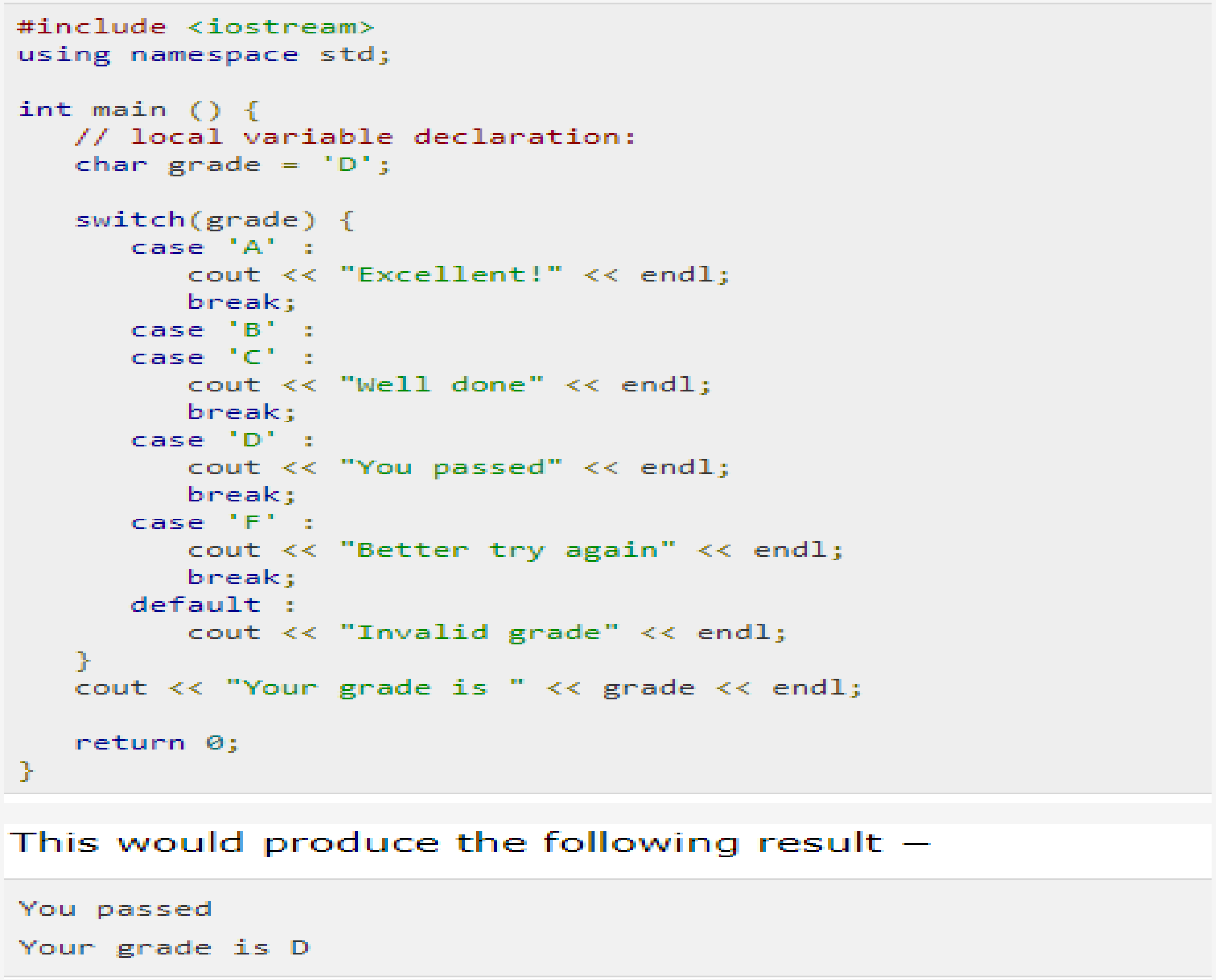
•A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Diagram

Flow Diagram



Example

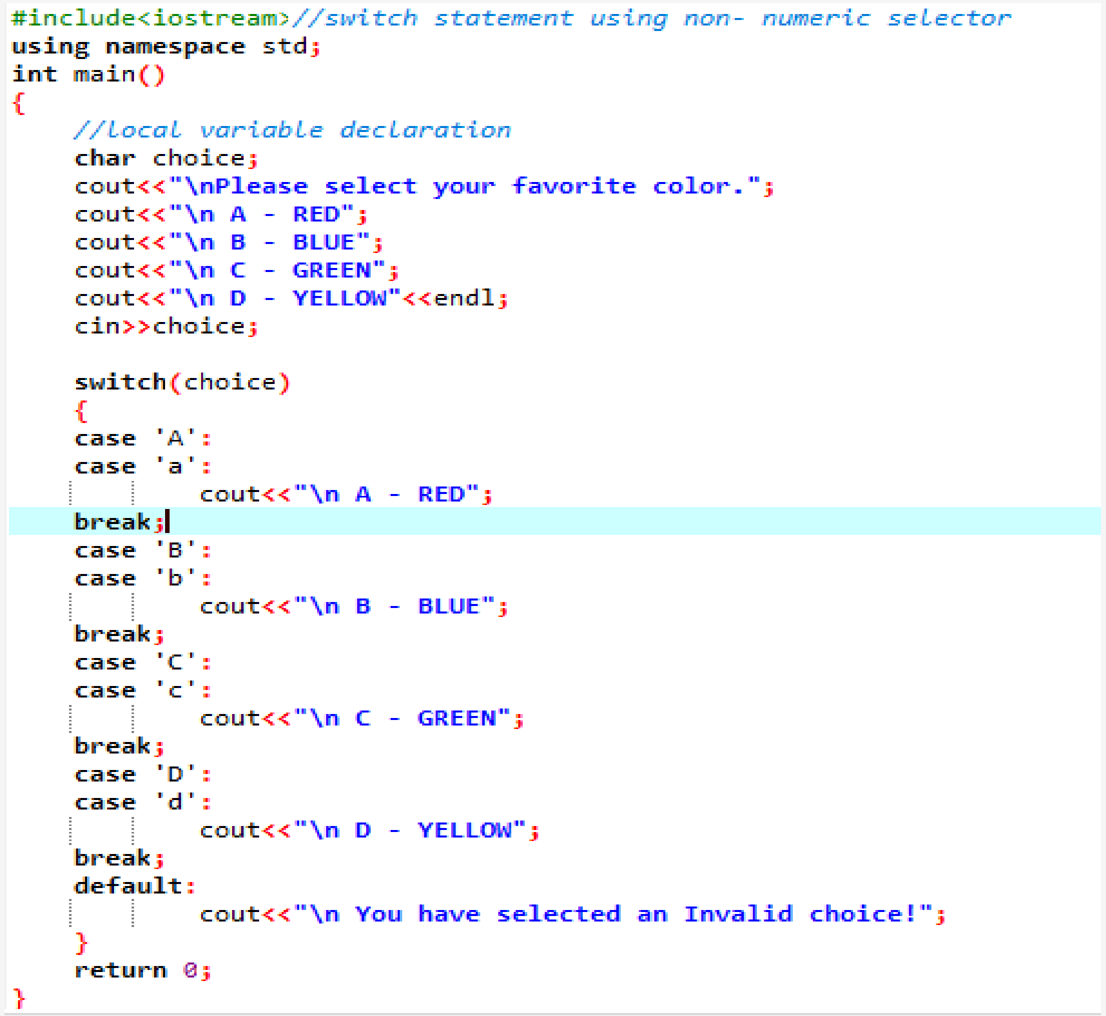


```cpp
#include <iostream>
using namespace std;

int main () {
   // local variable declaration:
   char grade = 'D';

   switch(grade) {
      case 'A' :
         cout << "Excellent!" << endl;
         break;
      case 'B' :
      case 'C' :
         cout << "Well done" << endl;
         break;
      case 'D' :
         cout << "You passed" << endl;
         break;
      case 'F' :
         cout << "Better try again" << endl;
         break;
      default :
         cout << "Invalid grade" << endl;
   }
   cout << "Your grade is " << grade << endl;

   return 0;
}
```

This would produce the following result −

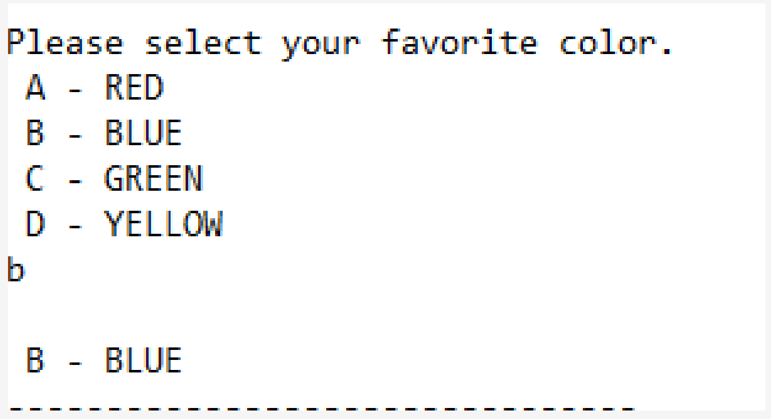
```
You passed
Your grade is D
```

Example using a character selector



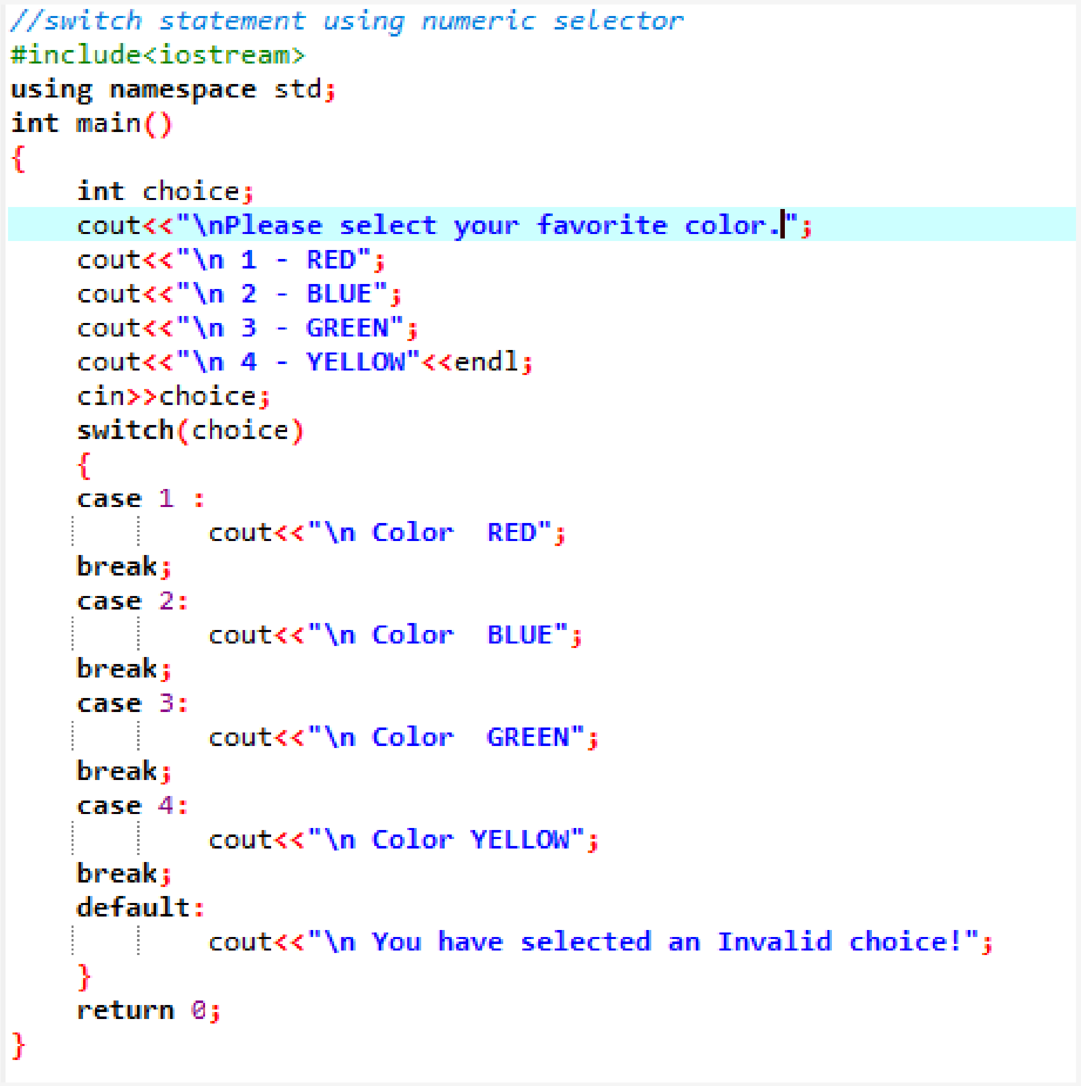
```cpp
#include<iostream>//switch statement using non- numeric selector
using namespace std;
int main()
{
    //local variable declaration
    char choice;
    cout<<"\nPlease select your favorite color.";
    cout<<"\n A - RED";
    cout<<"\n B - BLUE";
    cout<<"\n C - GREEN";
    cout<<"\n D - YELLOW"<<endl;
    cin>>choice;

    switch(choice)
    {
    case 'A':
    case 'a':
            cout<<"\n A - RED";
    break;
    case 'B':
    case 'b':
            cout<<"\n B - BLUE";
    break;
    case 'C':
    case 'c':
            cout<<"\n C - GREEN";
    break;
    case 'D':
    case 'd':
            cout<<"\n D - YELLOW";
    break;
    default:
            cout<<"\n You have selected an Invalid choice!";
    }
    return 0;
}
```

OUTPUT



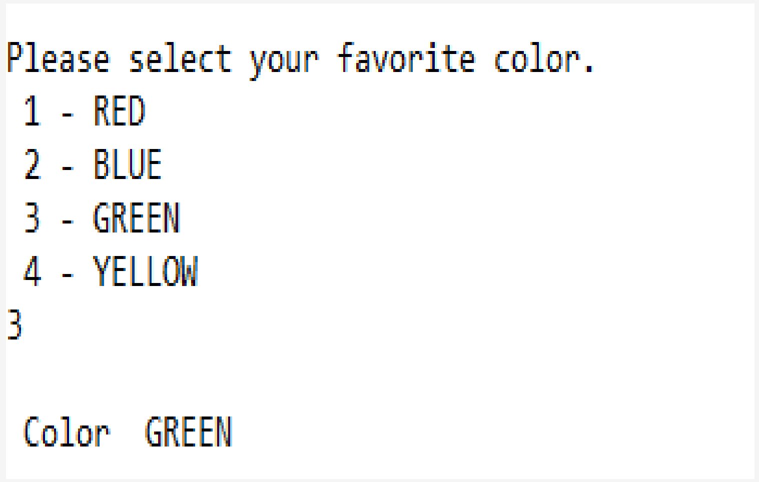
```
Please select your favorite color.
 A - RED
 B - BLUE
 C - GREEN
 D - YELLOW
b


 B - BLUE
--------------------------------
```

Example using numeric selector



```
//switch statement using numeric selector
#include<iostream>
using namespace std;
int main()
{
    int choice;
    cout<<"\nPlease select your favorite color.";
    cout<<"\n 1 - RED";
    cout<<"\n 2 - BLUE";
    cout<<"\n 3 - GREEN";
    cout<<"\n 4 - YELLOW"<<endl;
    cin>>choice;
    switch(choice)
    {
    case 1 :
            cout<<"\n Color  RED";
    break;
    case 2:
            cout<<"\n Color  BLUE";
    break;
    case 3:
            cout<<"\n Color  GREEN";
    break;
    case 4:
            cout<<"\n Color YELLOW";
    break;
    default:
            cout<<"\n You have selected an Invalid choice!";
    }
    return 0;
}
```

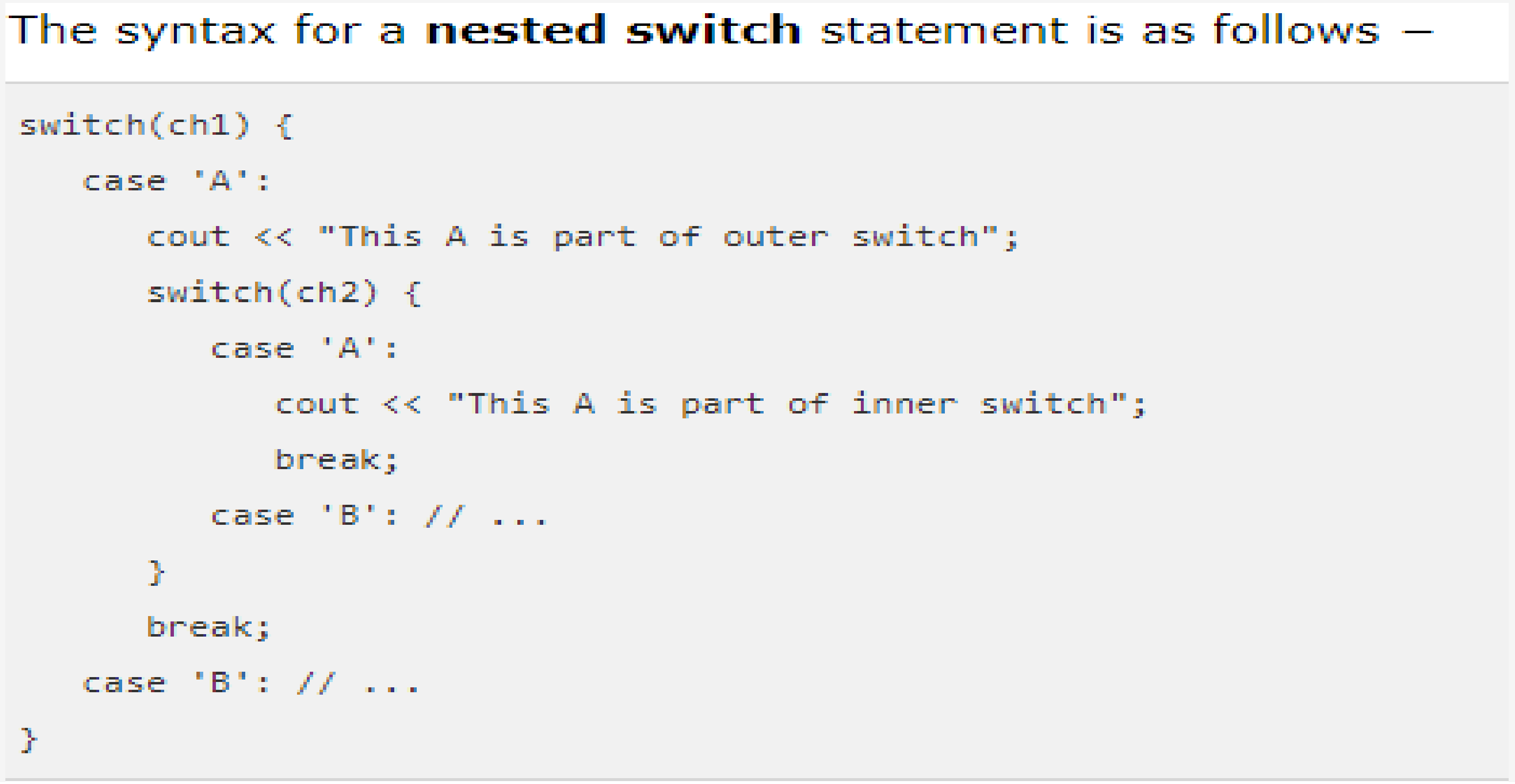
OUTPUT



```
Please select your favorite color.
 1 - RED
 2 - BLUE
 3 - GREEN
 4 - YELLOW
3

 Color  GREEN
```

Nested switch statements

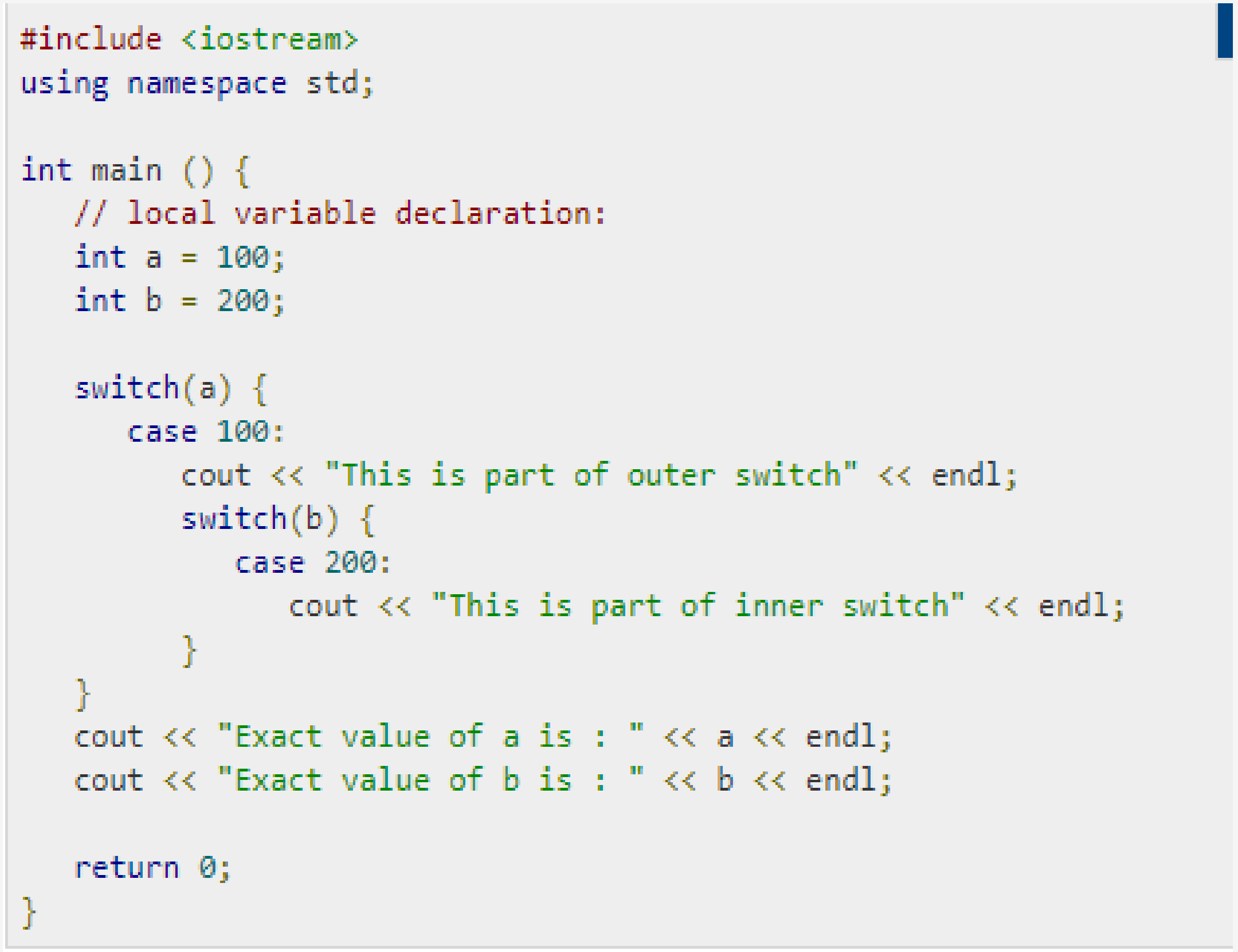
It is possible to have a switch as part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.



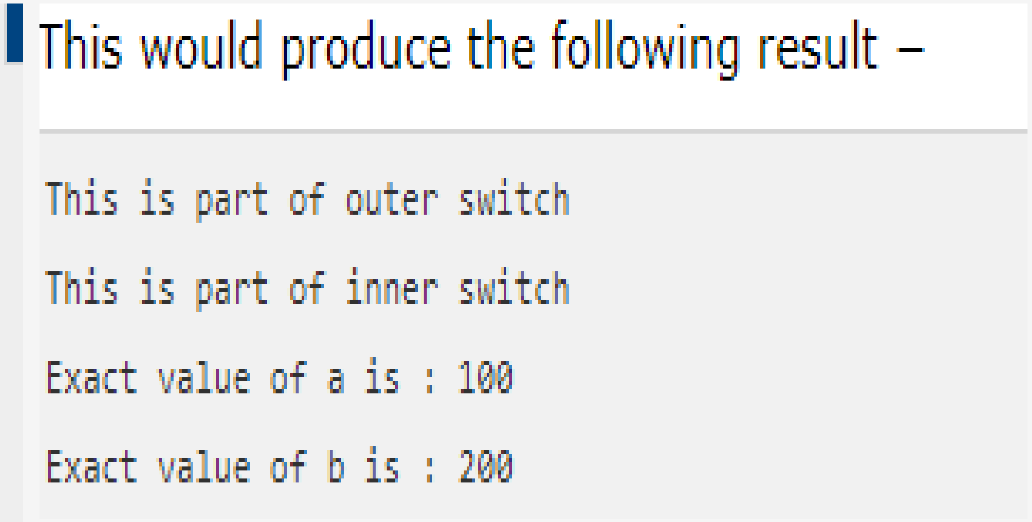
The syntax for a **nested switch** statement is as follows −

```
switch(ch1) {
   case 'A':
      cout << "This A is part of outer switch";
      switch(ch2) {
         case 'A':
            cout << "This A is part of inner switch";
            break;
         case 'B': // ...
      }
      break;
   case 'B': // ...
}
```

Example



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

int main () {
   // local variable declaration:
   int a = 100;
   int b = 200;

   switch(a) {
      case 100:
         cout << "This is part of outer switch" << endl;
         switch(b) {
            case 200:
               cout << "This is part of inner switch" << endl;
         }
   }
   cout << "Exact value of a is : " << a << endl;
   cout << "Exact value of b is : " << b << endl;

   return 0;
}
```



This would produce the following result –

```
This is part of outer switch

This is part of inner switch

Exact value of a is : 100

Exact value of b is : 200
```