The for loop

Its format is:

for (initialization; condition; increase) statement;

and its main function is to repeat statement while condition remains **true**, like the while loop. ***But*** **in addition, the for loop provides specific locations to contain an initialization statement and an increase statement**. So this loop is specially designed to perform a repetitive action with a **counter** which is **initialized** and **increased** on each iteration.

It works in the following way:

1. initialization is executed. Generally it is an initial value setting for a counter variable. This is executed only once.

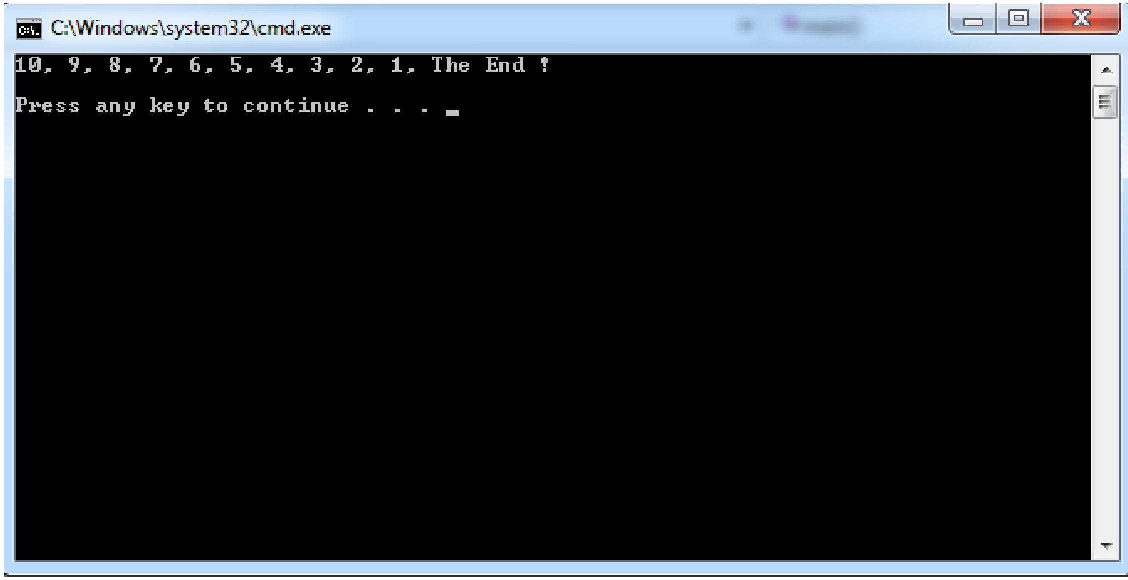
2. condition is checked. If it is **true** the loop **continues**, otherwise the loop **ends** and statement is **skipped** (not executed).

3. statement is executed. As usual, it can be either a **single** statement or a **block** enclosed in braces { }.

4. finally, whatever is specified in the **increase** field is executed and the loop gets back to step 2.

Here is an example of countdown using a for loop:

```
// countdown using a for loop
#include <iostream>
using namespace std;
int main ()
{
    for (int number=10; number>0; number--)
    {
        cout << number << ", ";
    }
cout << "The End !\n\n";
return 0;
}
```



```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, The End !

Press any key to continue . . . _
```

The initialization and increase fields are **optional**. They can remain empty, but in all cases the **semicolon** signs between them **must be** written.

For example we could write:

```
for (;n<10;)
```

if we wanted to specify **no initialization** and **no increase;**

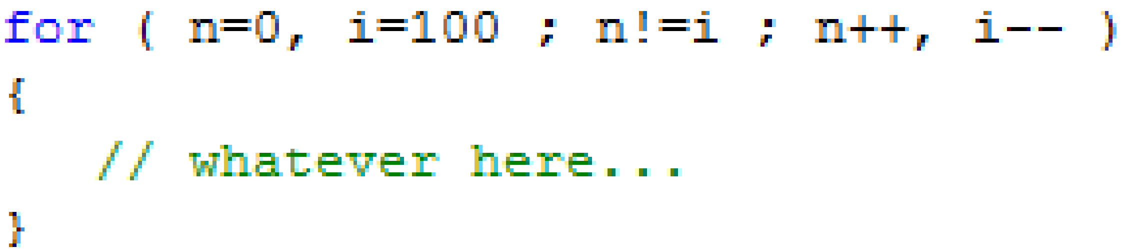
or

```
for (;n<10;n++)
```

if we wanted to include an **increase** field but **no initialization** (maybe because the variable was already initialized before).

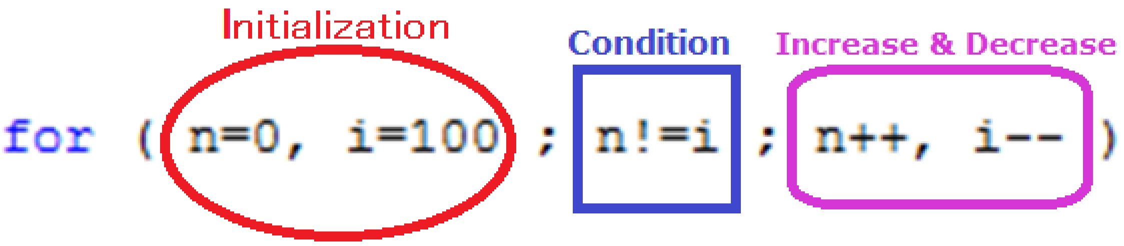
Optionally, using the **comma operator (,)** we can specify **more than one expression** in **any of the fields included in a for loop**, like in initialization, for example. The **comma operator (,)** is an expression separator, it serves to **separate** more than one expression where only one is generally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

For example, suppose that we wanted to initialize more than one variable in our loop:



```
for ( n=0, i=100 ; n!=i ; n++, i-- )
{
   // whatever here...
}
```

This loop will execute for 50 times if neither **n** or **i** are modified within the loop:



```
for ( n=0, i=100 ; n!=i ; n++, i-- )
```

n starts with a value of 0, and **i with 100**, the **condition** is ***n!=i*** (that **n is not equal to i**). Because **n** is **increased by one** and i decreased by one, the loop's condition will become **false** after the 50th loop, when both **n** and **i** will be equal to 50.

The selective structure: switch.

The syntax of the **switch** statement is a bit peculiar. Its objective is to **check several possible constant values** for an expression. Something **similar** to what we did at the previous lecture with the concatenation of several **if and else if instructions**. Its form is the following:

```
switch (expression)
{
    case constant1:
        group of statements 1;
    break;
    case constant2:
      group of statements 2;
    break;
    .
    .
    .
    default:
     default group of statements
}
```

It works in the following way: **switch** evaluates expression and **checks** if it is equivalent to constant1, if it is, it executes group of **statements1 until it finds the break** statement.

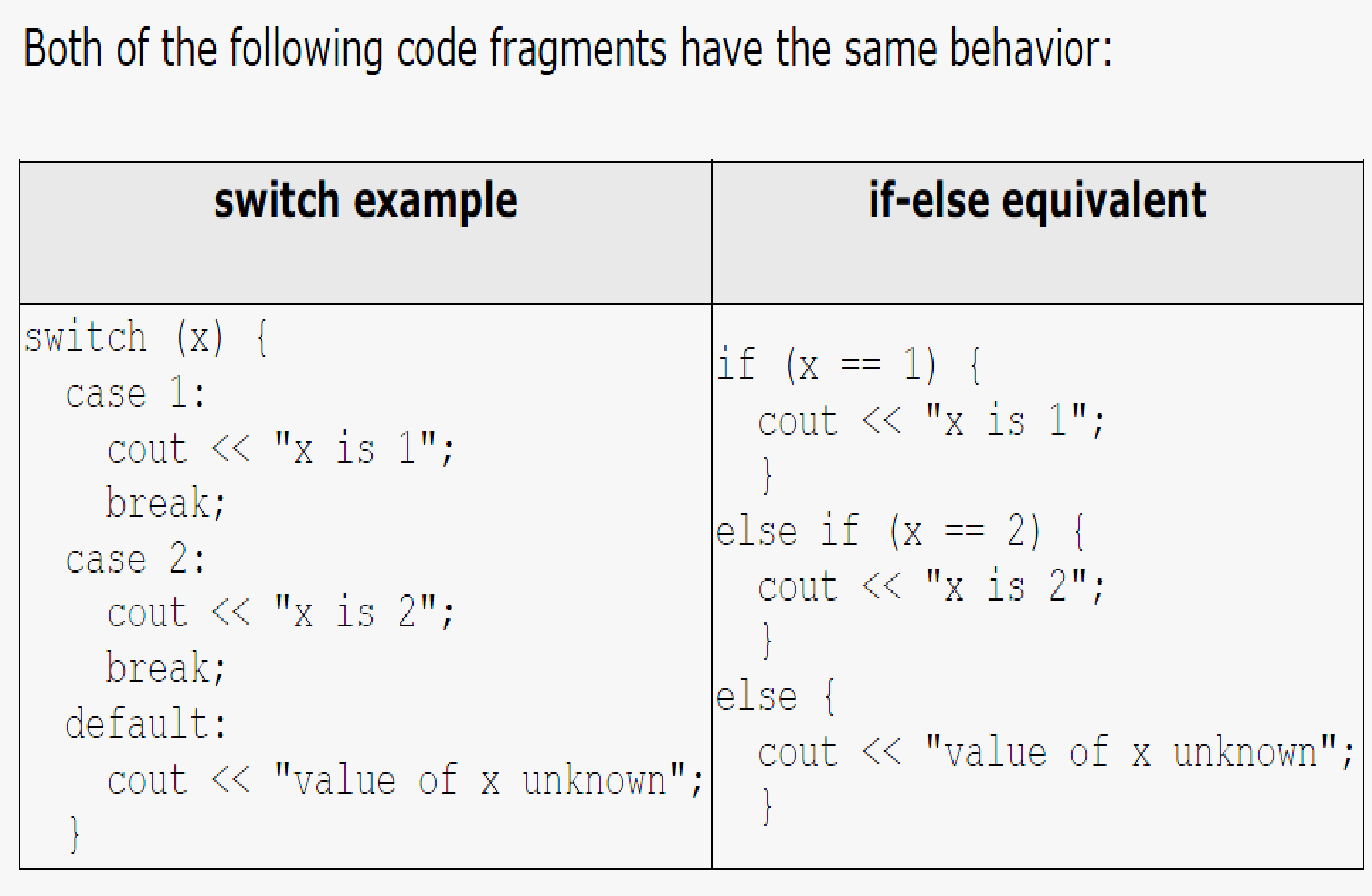
When it finds this break statement the program jumps to the end of the switch selective structure.

If expression was **not equal to constant1** it will be **checked** against **constant2.**

If it is equal to **this**, it will execute group of statements 2 until a **break** keyword is found, and then will jump to the end of the switch selective structure.

Finally, if the value of expression did not match any of the previously specified constants (you can include as many case labels as values you want to check),

the program will execute the statements included after the default: label, if it exists (since it is optional).

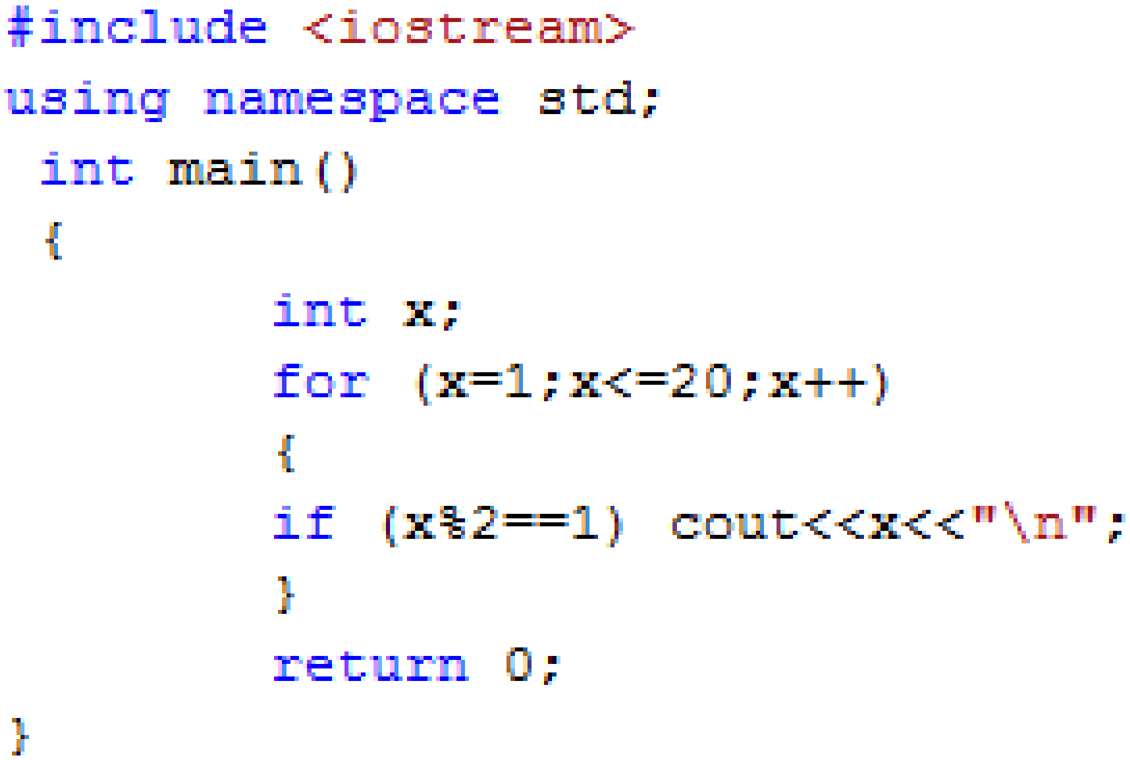


Both of the following code fragments have the same behavior:

switch example	if-else equivalent

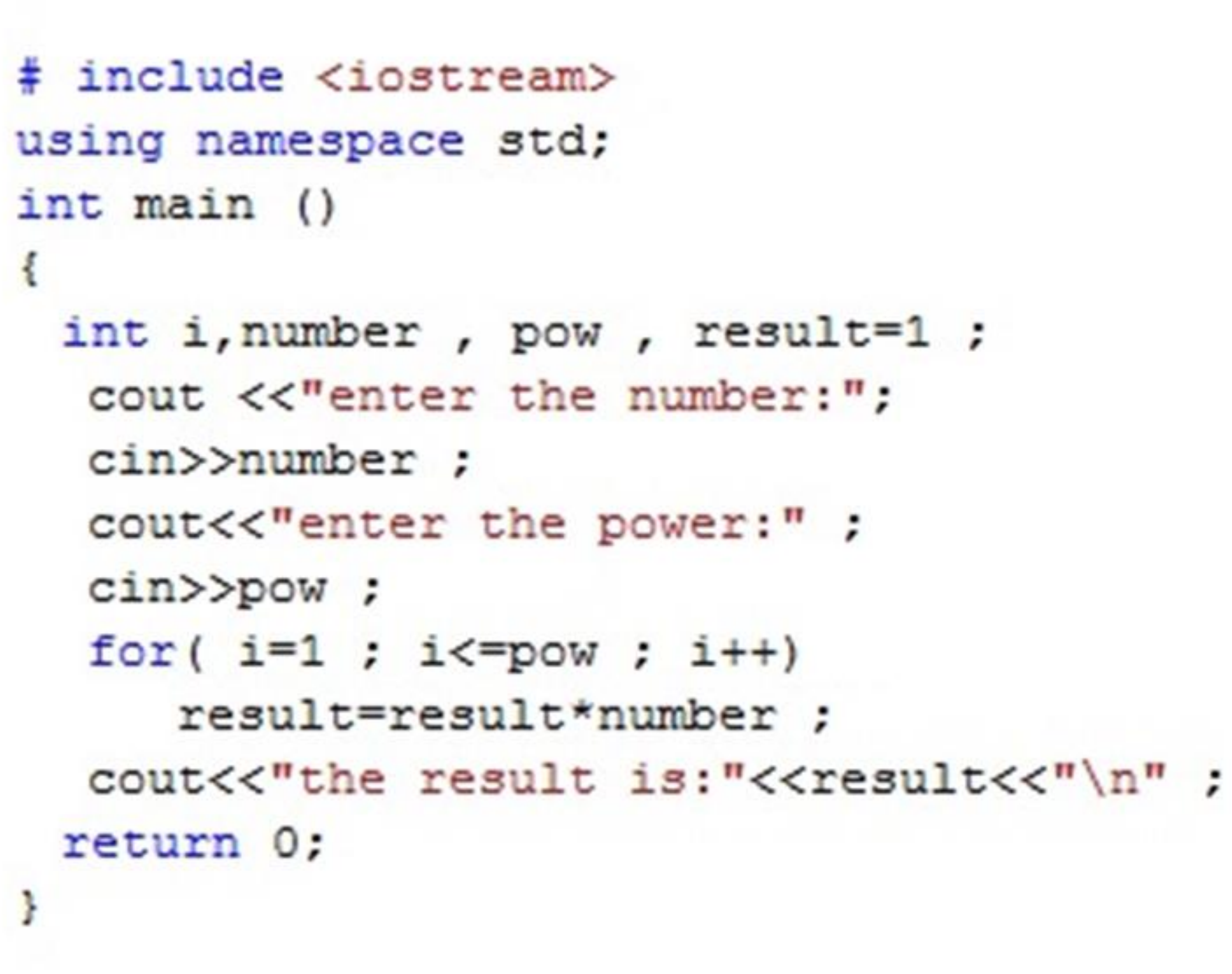
switch (x) {
  case 1:
    cout << "x is 1";
    break;
  case 2:
    cout << "x is 2";
    break;
  default:
    cout << "value of x unknown";
  }
	if (x == 1) {
	  cout << "x is 1";
	  }
	else if (x == 2) {
	  cout << "x is 2";
	  }
	else {
	  cout << "value of x unknown";
	  }

اكتب برنامج بلغة C++ يقوم بطباعة الاعداد الفردية Odd من 1الى 20

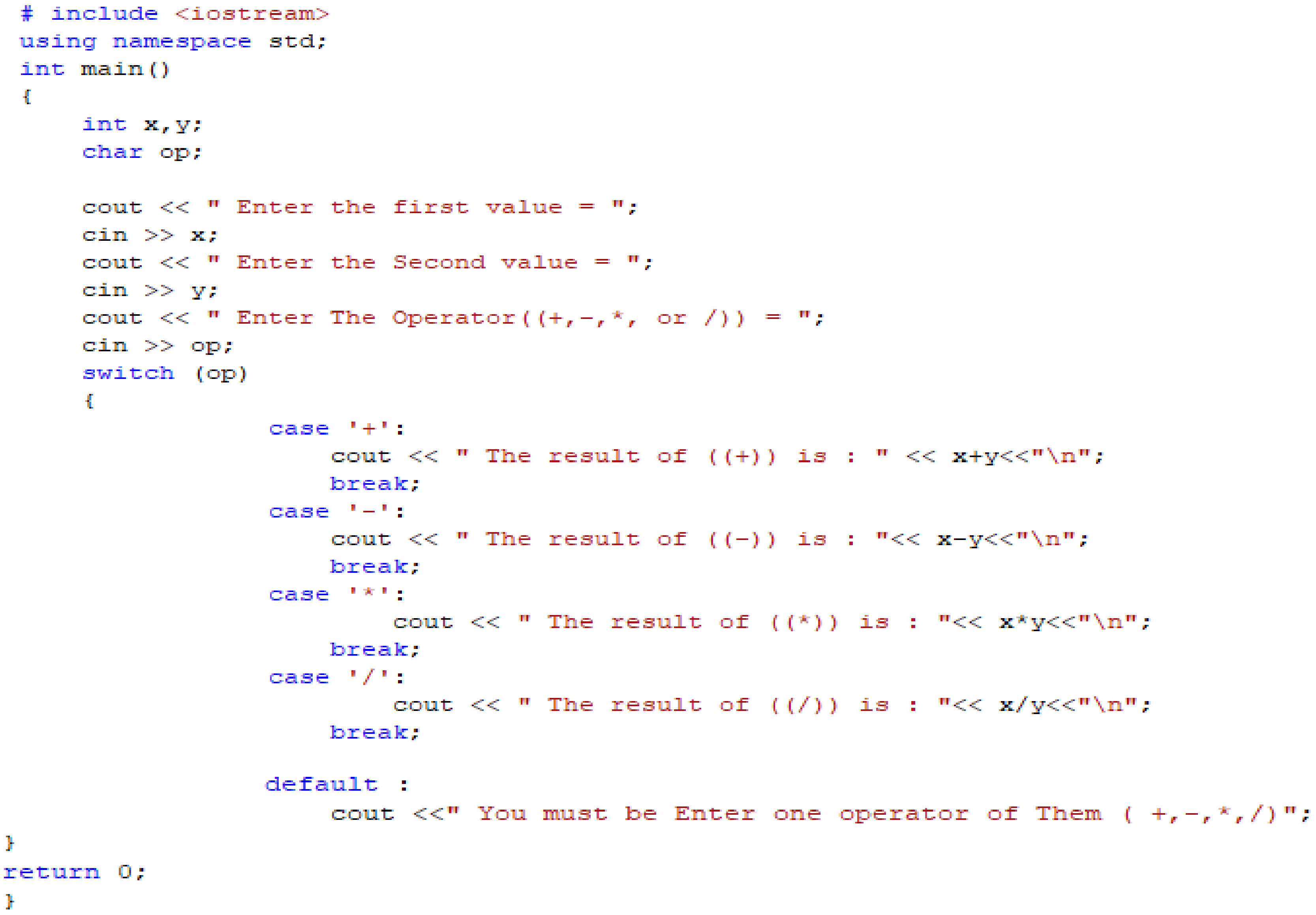


```
#include <iostream>
using namespace std;
 int main()
 {
        int x;
        for (x=1;x<=20;x++)
        {
        if (x%2==1) cout<<x<<"\n";
        }
        return 0;
}
```

أكتب برنامج بلغة ال C++ يقوم بحساب (X^Y Power)



```
# include <iostream>
using namespace std;
int main ()
{
  int i,number , pow , result=1 ;
   cout <<"enter the number:";
   cin>>number ;
   cout<<"enter the power:" ;
   cin>>pow ;
   for( i=1 ; i<=pow ; i++)
      result=result*number ;
   cout<<"the result is:"<<result<<"\n" ;
  return 0;
}
```



```
 # include <iostream>
 using namespace std;
 int main()
 {
     int x,y;
     char op;

     cout << " Enter the first value = ";
     cin >> x;
     cout << " Enter the Second value = ";
     cin >> y;
     cout << " Enter The Operator((+,-,*, or /)) = ";
     cin >> op;
     switch (op)
     {
                 case '+':
                     cout << " The result of ((+)) is : " << x+y<<"\n";
                     break;
                 case '-':
                     cout << " The result of ((-)) is : "<< x-y<<"\n";
                     break;
                 case '*':
                         cout << " The result of ((*)) is : "<< x*y<<"\n";
                     break;
                 case '/':
                         cout << " The result of ((/)) is : "<< x/y<<"\n";
                     break;

                 default :
                     cout <<" You must be Enter one operator of Them ( +,-,*,/)";
}
return 0;
}
```