Programming in c++
Lecture_3

Dr. Sara Mohamed

Flow of Control

Flow of Control

- Flow of control
  - The order in which statements are executed
- Branch
  - Program choose between two alternatives

Branch Example

- To calculate hourly wages there are two choices
  - Regular time (up to 40 hours)
    - gross_pay = rate * hours;
  - Overtime (over 40 hours)
    - gross_pay = rate * 40 + 1.5 * rate * (hours - 40);
  - The program must choose which of these expressions to use

Designing the Branch

- Decide if (hours >40) is true
  - If it is true, then use

    ```
    gross_pay  =  rate * 40 + 1.5 * rate * (hours - 40);
    ```

  - If it is not true, then use

    ```
    gross_pay = rate * hours;
    ```

Implementing the Branch

- if-else statement is used in C++ to perform a branch

  -
    ```
    if (hours > 40)
        gross_pay  =  rate * 40 + 1.5 * rate * (hours - 40);
    else
        gross_pay = rate * hours;
    ```

Boolean Expressions

- Boolean expressions are expressions that are either true or false
- **comparison operators** such as '>' (greater than) are used to compare variables and/or numbers
  - **(hours > 40)** Including the parentheses, is the boolean expression from the wages example
  - A few of the comparison operators that use two symbols (No spaces allowed between the symbols!)
    - **>=** greater than or equal to
    - **!=** not equal or inequality
    - **= =** equal or equivalent

if-else Flow Control (1)

- if (*boolean expression*)
        true statement
  else
        false statement
- When the boolean expression is true
  - Only the true statement is executed
- When the boolean expression is false
  - Only the false statement is executed

if-else Flow Control (2)

- if (*boolean expression*)
  ```
      {
              true statements
      }
  else
      {
              false statements
      }
  ```
- When the boolean expression is true
  - Only the true statements enclosed in **{ }** are executed
- When the boolean expression is false
  - Only the false statements enclosed in **{ }** are executed

AND

- Boolean expressions can be combined into more complex expressions with
  - &&   -- The AND operator
    - True if both expressions are true
- Syntax: (*Comparison_1*) && (*Comparison_2*)
- Example: if ((2 < x) && (x < 7))
  - True only if x is between 2 and 7
  - Inside parentheses are optional but enhance meaning

OR

- | | -- The OR operator (no space!)
    - True if either or both expressions are true
- Syntax: (Comparison_1) | | (Comparison_2)
- Example: if ((x = = 1) | | (x = = y))
    - True if x contains 1
    - True if x contains the same value as y
    - True if both comparisons are true

NOT

- ! -- negates any boolean expression
  - !(x < y)
    - True if x is NOT less than y
  - !(x = = y)
    - True if x is NOT equal to y
- ! Operator can make expressions difficult to understand…use only when appropriate

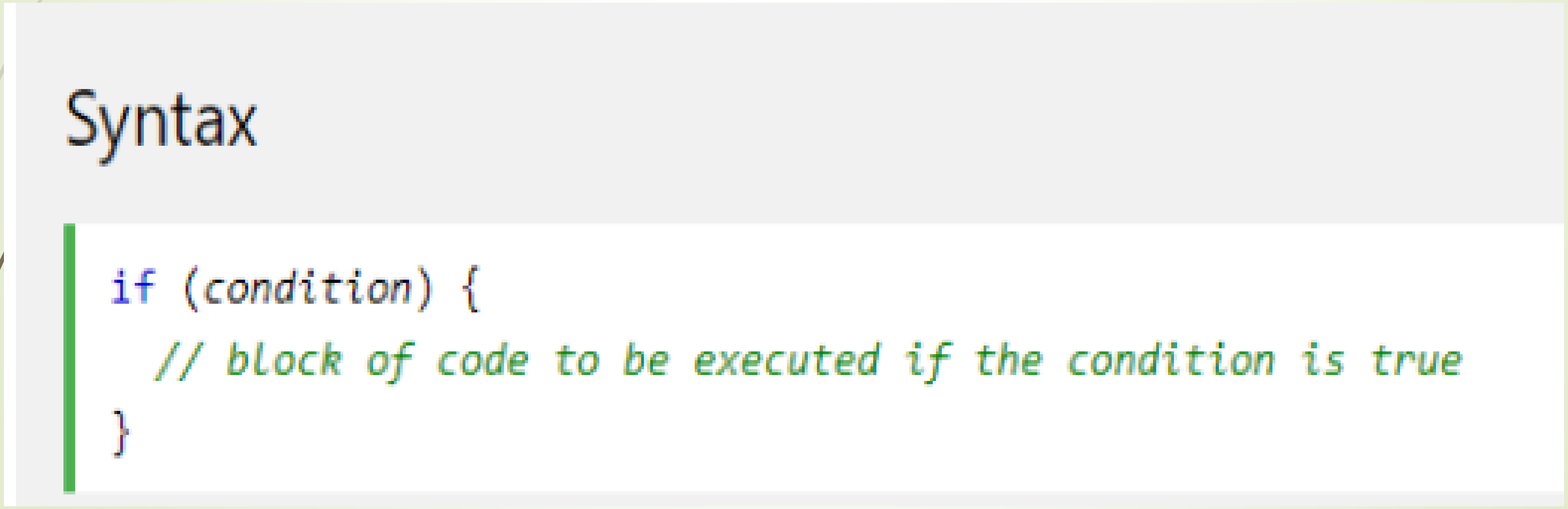
Inequalities

- Be careful translating inequalities to C++
- if x < y < z   translates as

```
if ( ( x < y )  && ( y < z ) )

NOT

if ( x < y < z )
```

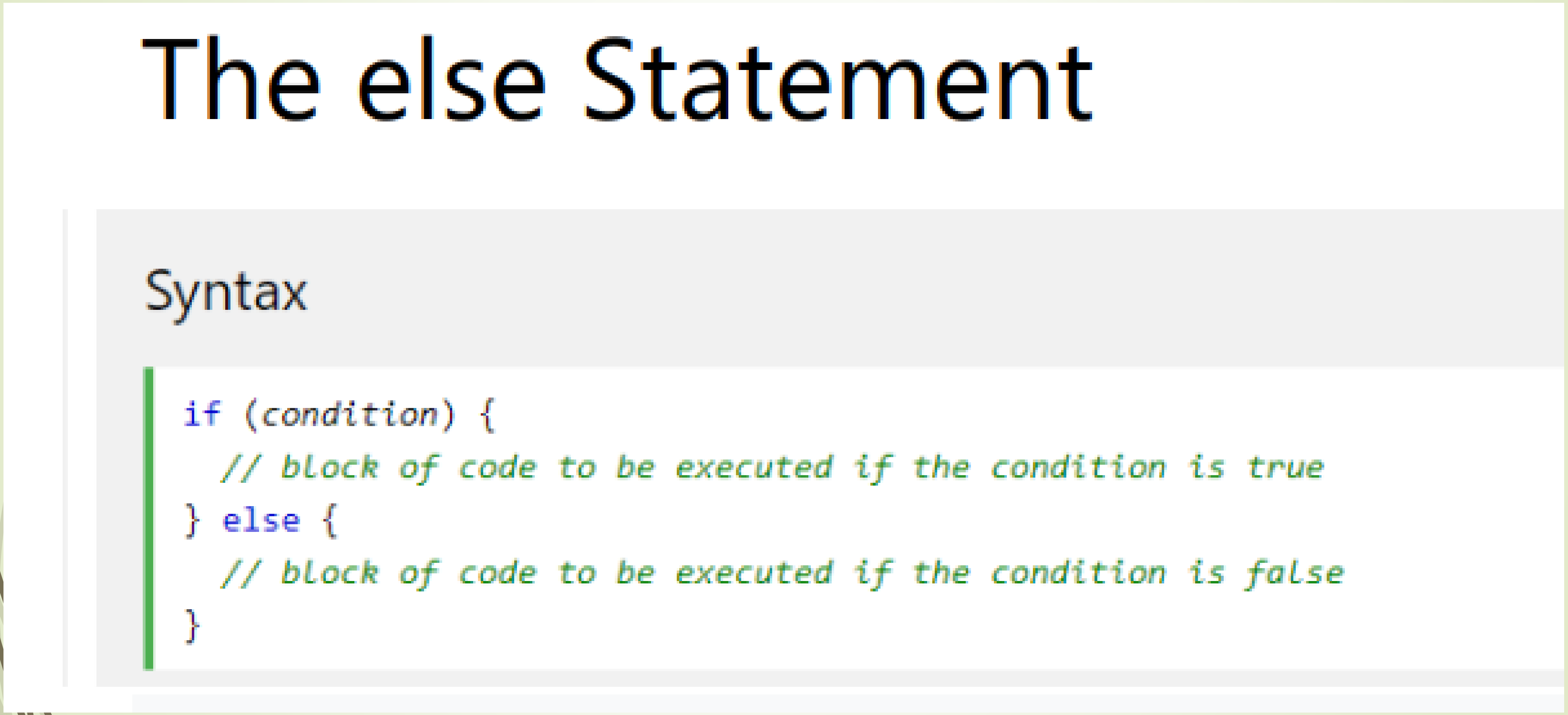
If Statements

- if to specify a block of code to be executed, if a specified condition is true



Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
}
```



The else Statement

Syntax

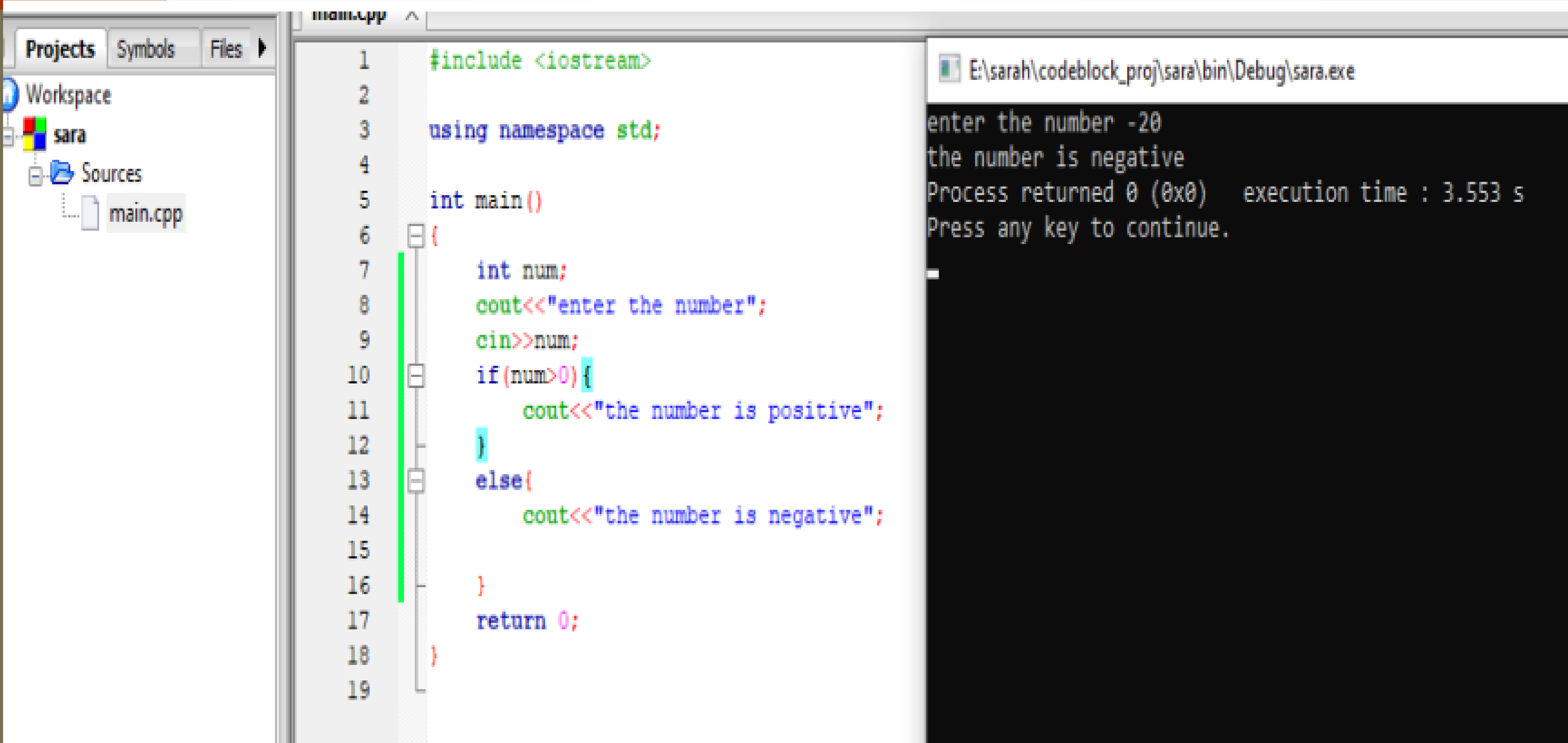
```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

Compound Statements

- A compound statement is more than one statement enclosed in { }
- Branches of if-else statements often need to execute more that one statement
- **Example:**

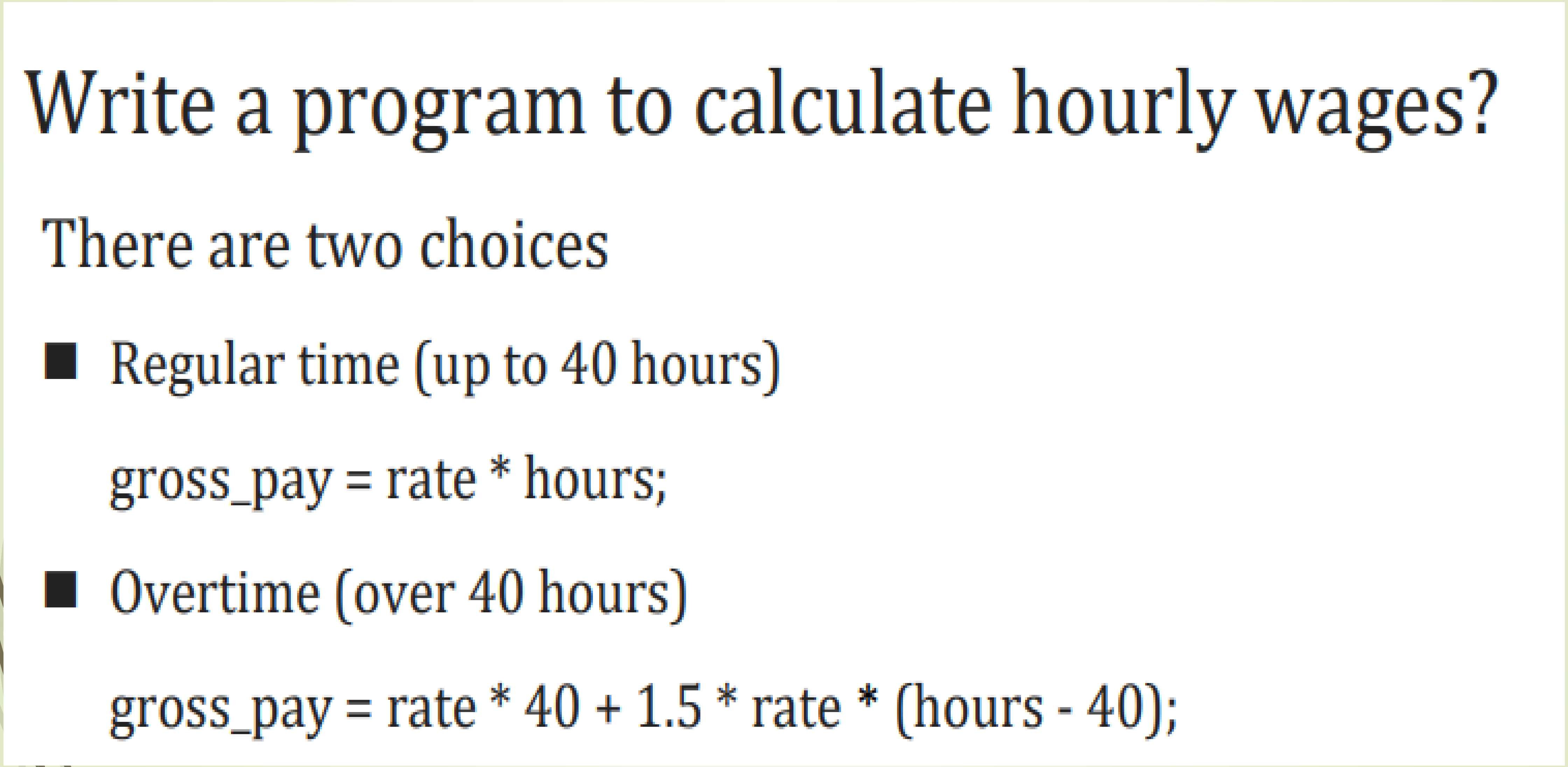
-
  ```
  if (boolean expression)
                  {
                        true statements
                  }
                 else
                  {
                        false statements
                   }
  ```

Write a c++ program to check the number negative or positive?



```cpp
#include <iostream>

using namespace std;

int main()
{
    int num;
    cout<<"enter the number";
    cin>>num;
    if(num>0){
        cout<<"the number is positive";
    }
    else{
        cout<<"the number is negative";

    }
    return 0;
}

```

```
enter the number -20
the number is negative
Process returned 0 (0x0)   execution time : 3.553 s
Press any key to continue.
```



Write a program to calculate hourly wages?

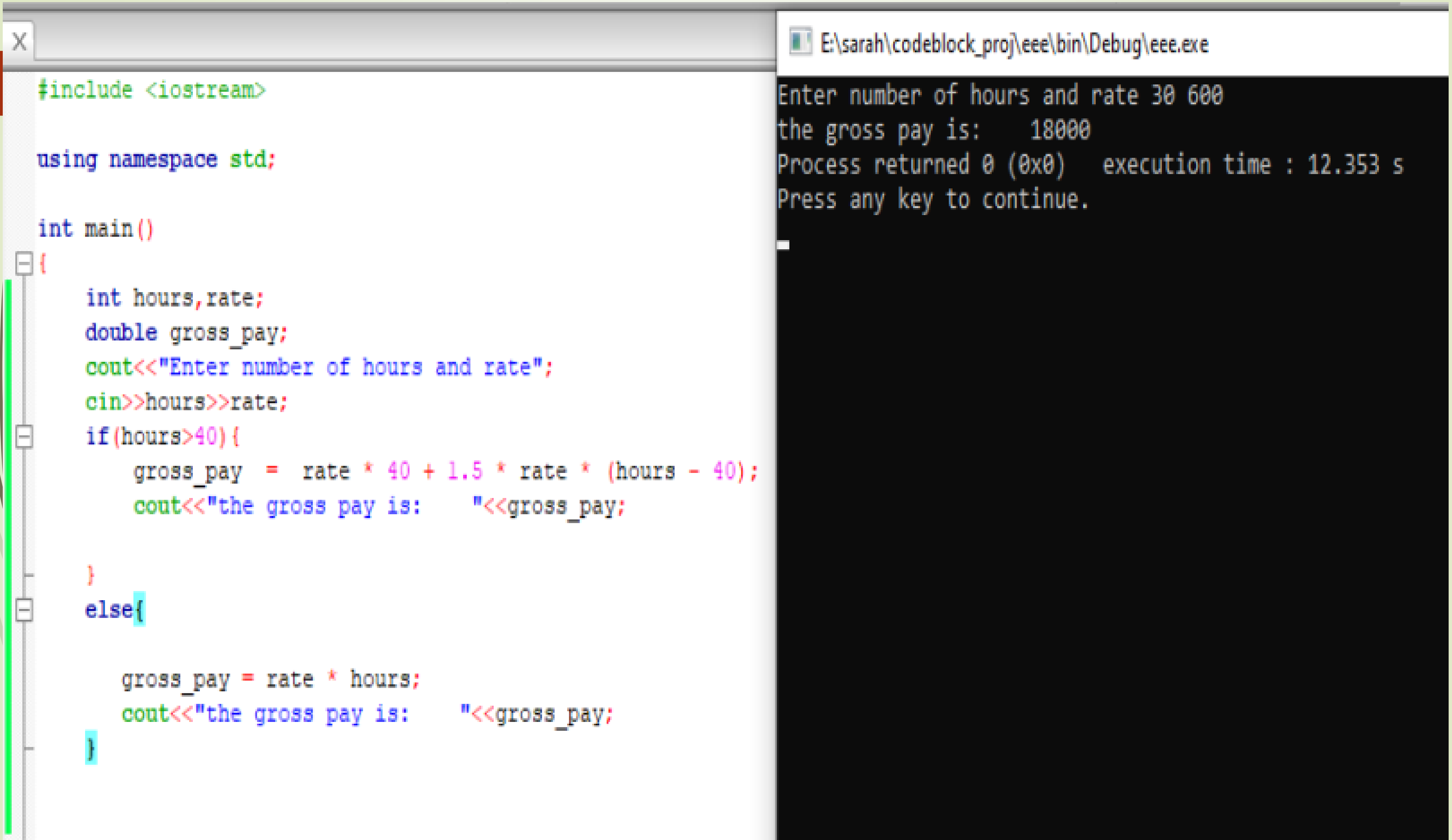
There are two choices

- Regular time (up to 40 hours)

```
gross_pay = rate * hours;
```

- Overtime (over 40 hours)

```
gross_pay = rate * 40 + 1.5 * rate * (hours - 40);
```



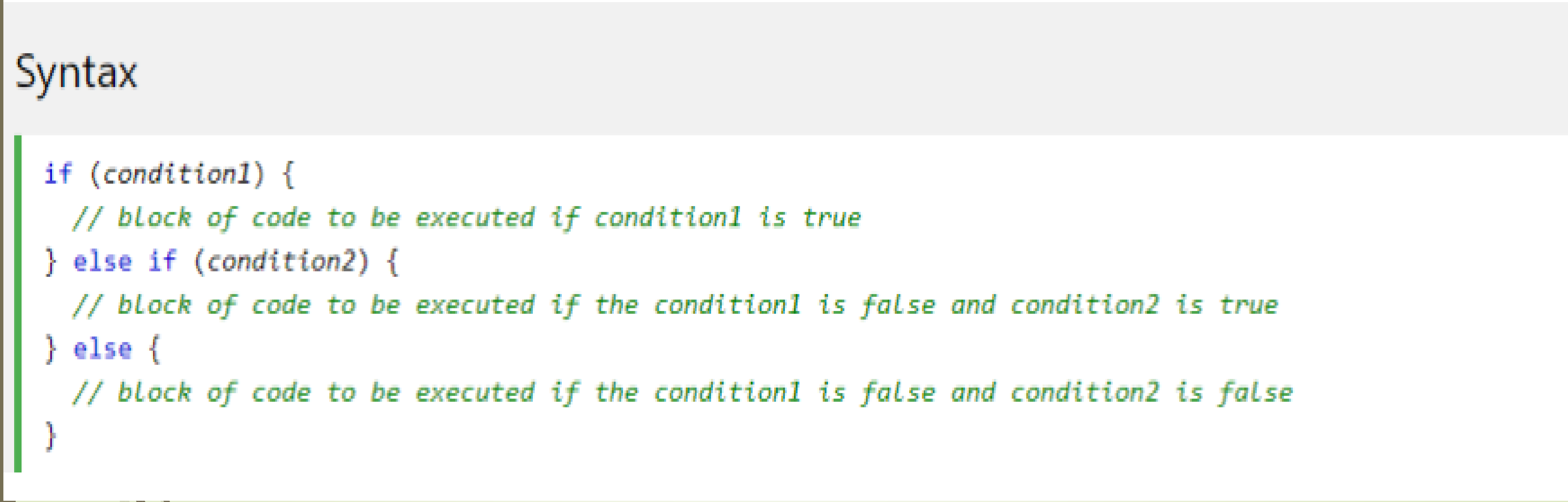
```
#include <iostream>

using namespace std;

int main()
{
    int hours,rate;
    double gross_pay;
    cout<<"Enter number of hours and rate";
    cin>>hours>>rate;
    if(hours>40){
        gross_pay  =  rate * 40 + 1.5 * rate * (hours - 40);
        cout<<"the gross pay is:    "<<gross_pay;

    }
    else{

       gross_pay = rate * hours;
       cout<<"the gross pay is:    "<<gross_pay;
    }
```

E:\sarah\codeblock_proj\eee\bin\Debug\eee.exe

```
Enter number of hours and rate 30 600
the gross pay is:    18000
Process returned 0 (0x0)   execution time : 12.353 s
Press any key to continue.
```

The else if Statement

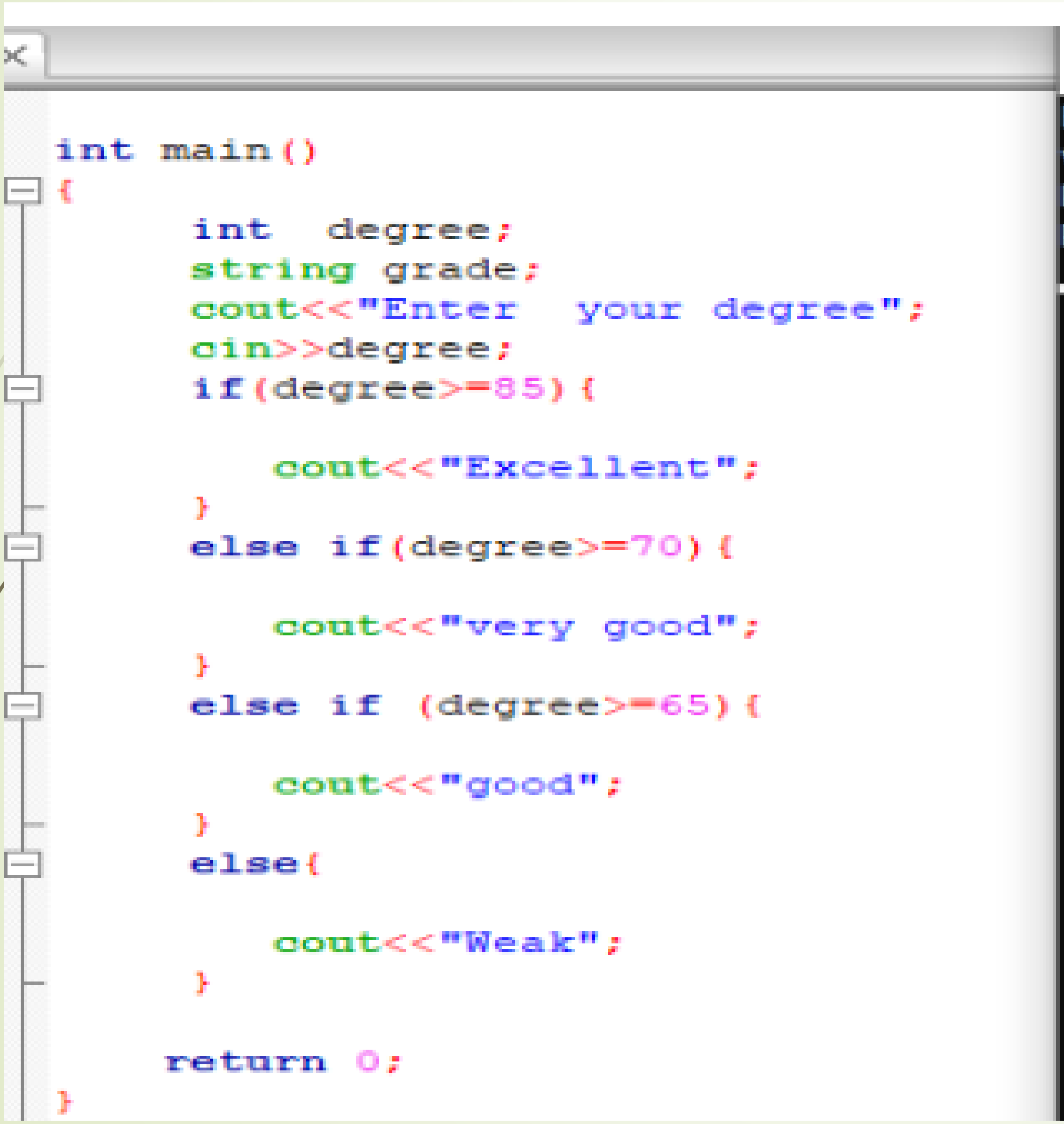
Use the else if statement to specify a new condition if the first condition is false



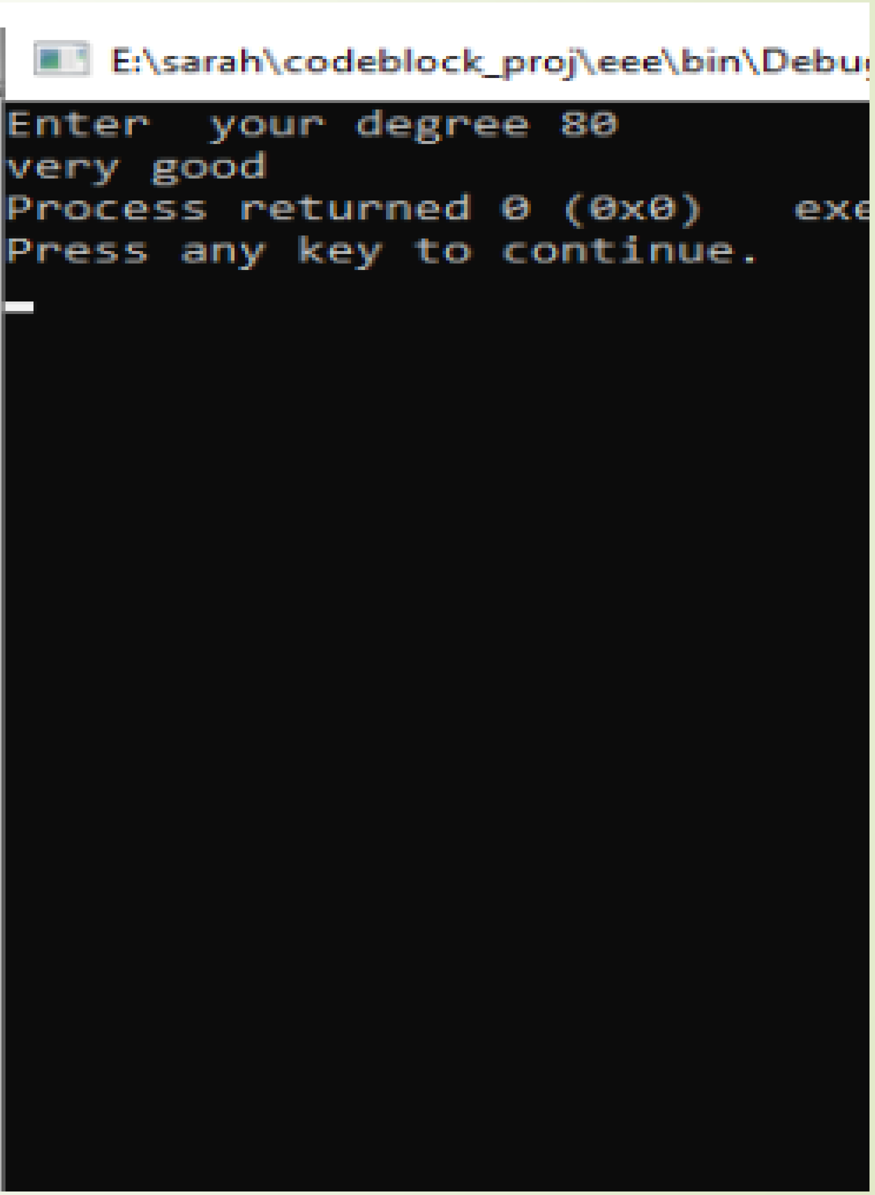
Syntax

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

Write a program to Calculates students' grades?



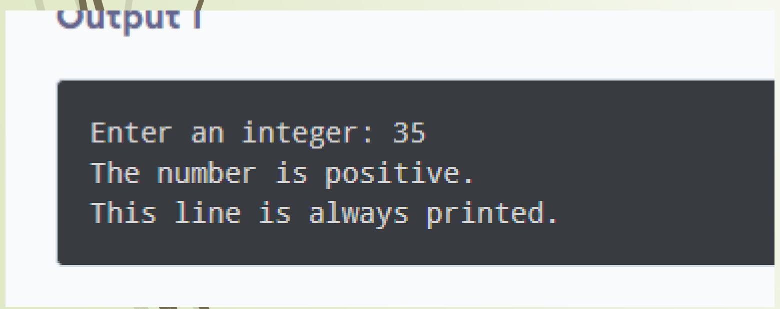
```
int main()
{
        int  degree;
        string grade;
        cout<<"Enter  your degree";
        cin>>degree;
        if(degree>=85){

            cout<<"Excellent";
        }
        else if(degree>=70){

            cout<<"very good";
        }
        else if (degree>=65){

            cout<<"good";
        }
        else{

            cout<<"Weak";
        }

      return 0;
}
```



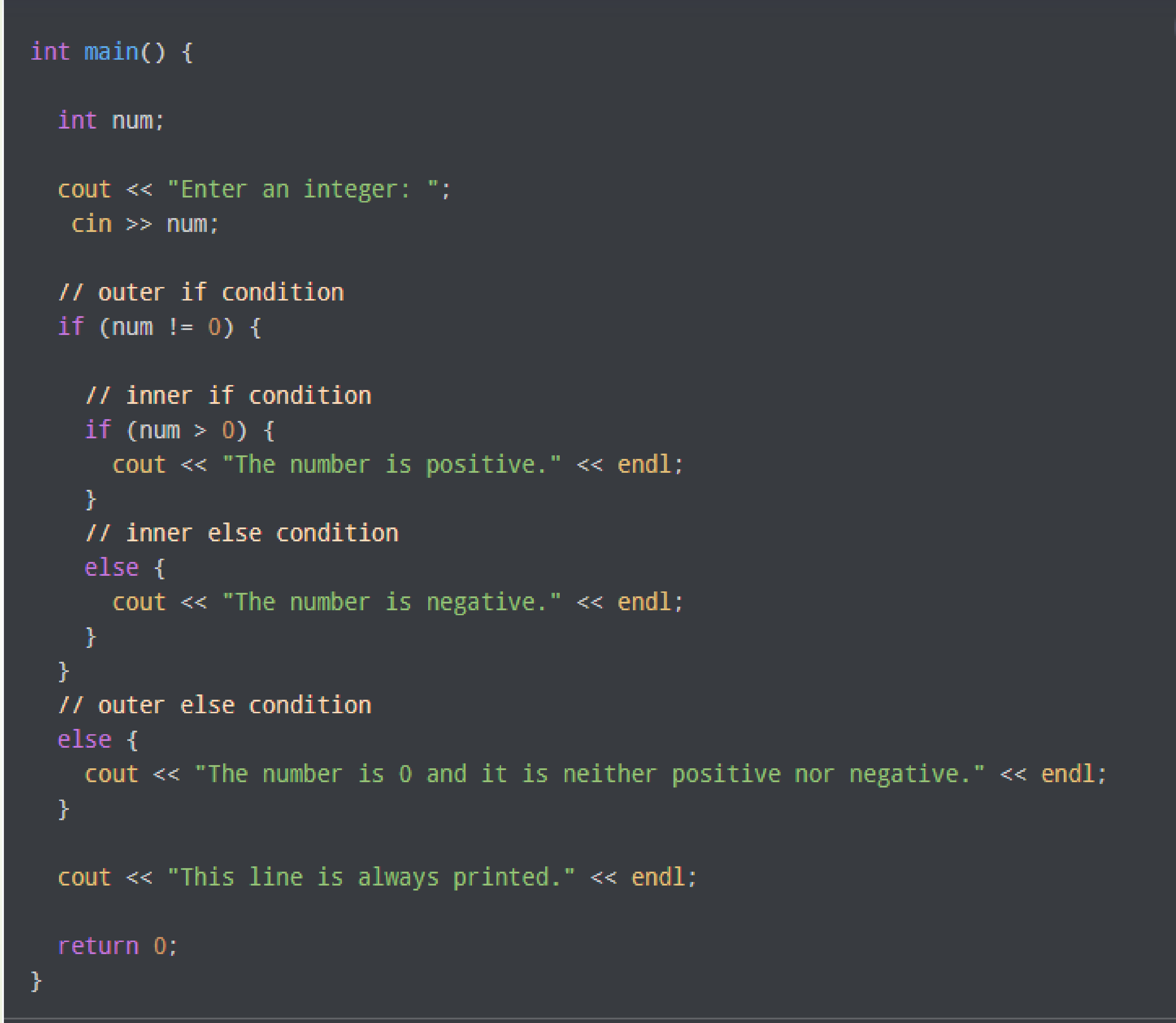
E:\sarah\codeblock_proj\eee\bin\Debu

```
Enter  your degree 80
very good
Process returned 0 (0x0)   exe
Press any key to continue.
```

C++ program to find if an integer is positive, negative or zero using nested if statements

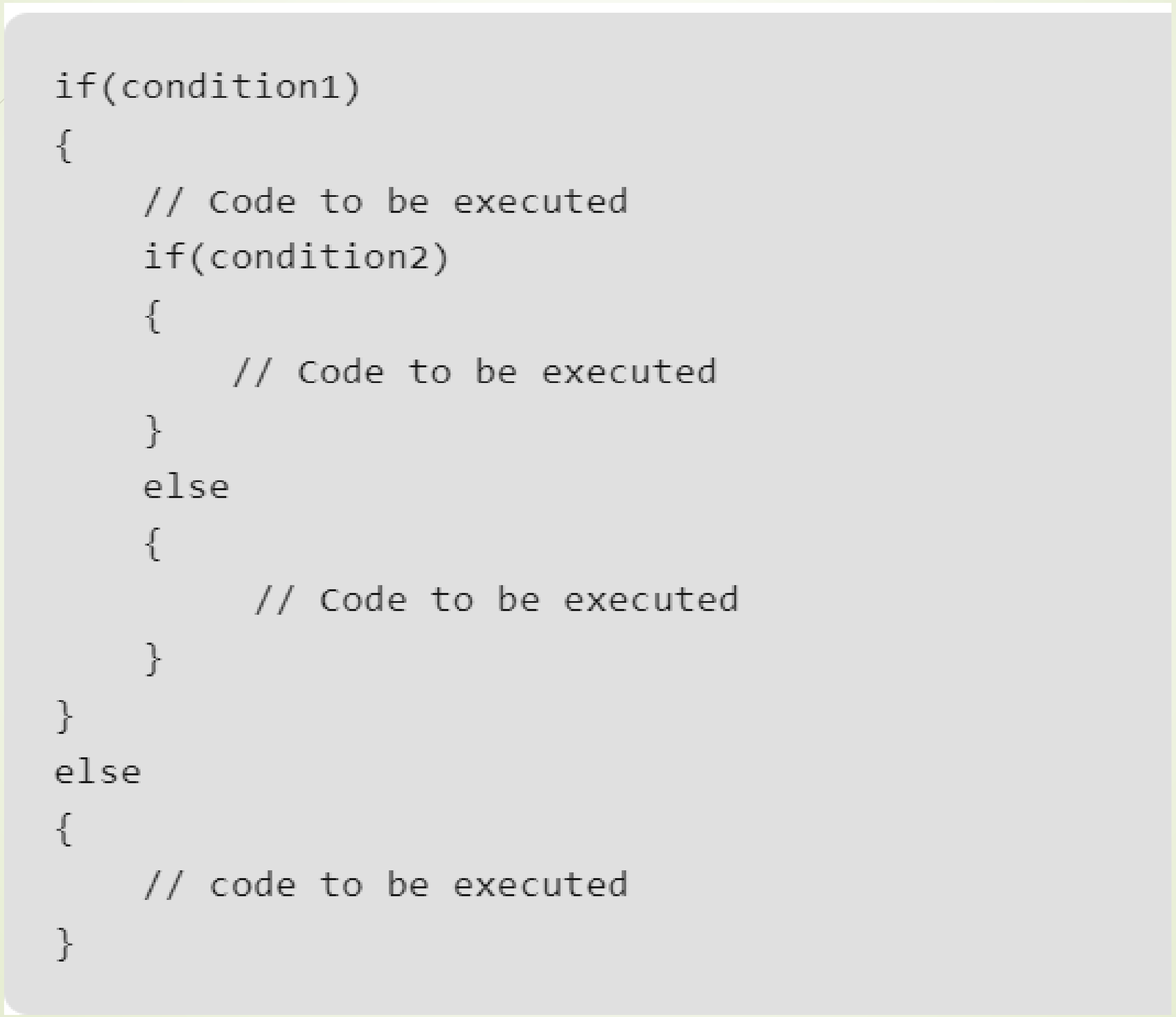

Output 1

```
Enter an integer: 35
The number is positive.
This line is always printed.
```

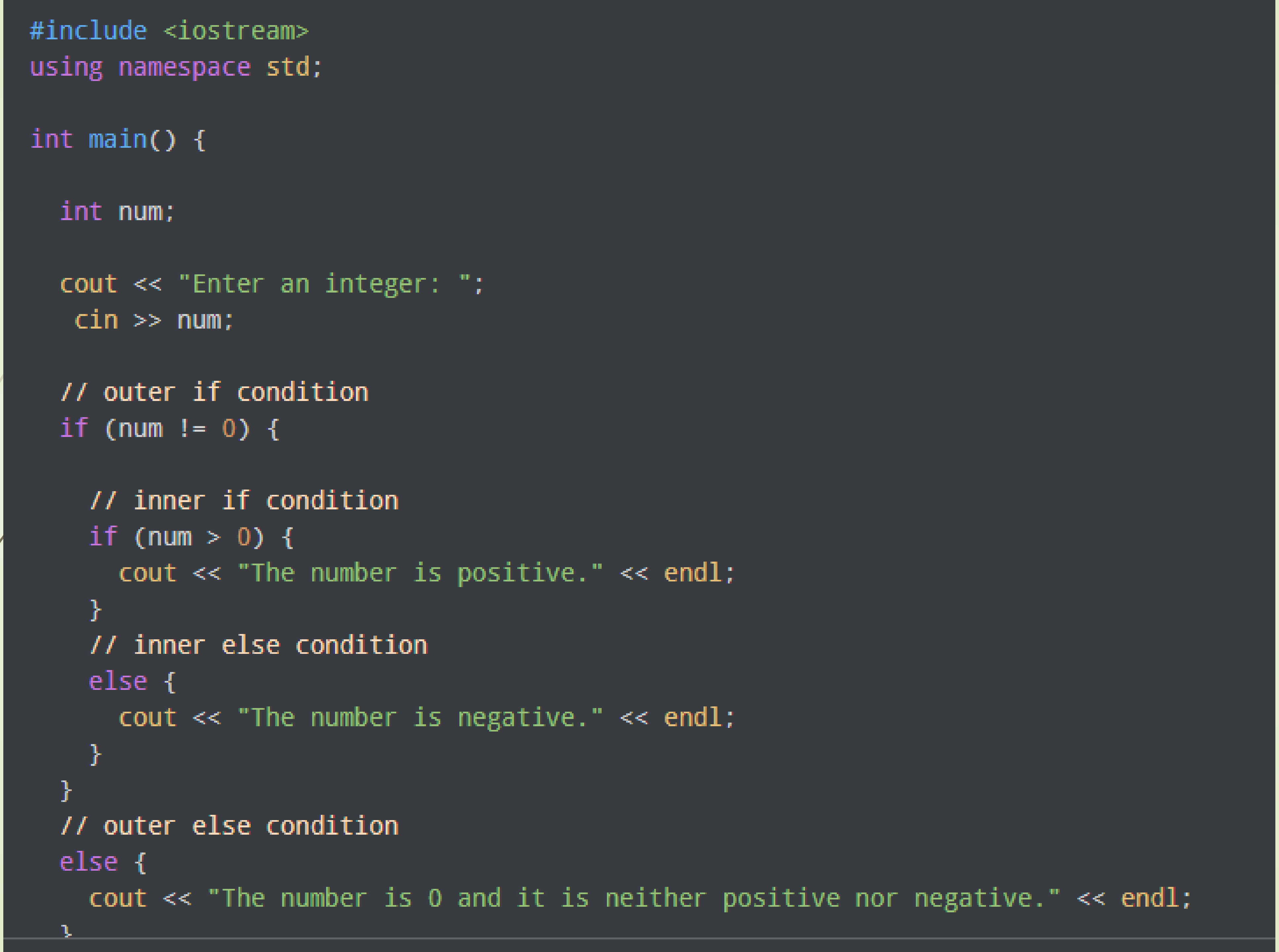


```cpp
int main() {

  int num;

  cout << "Enter an integer: ";
   cin >> num;

  // outer if condition
  if (num != 0) {

    // inner if condition
    if (num > 0) {
      cout << "The number is positive." << endl;
    }
    // inner else condition
    else {
      cout << "The number is negative." << endl;
    }
  }
  // outer else condition
  else {
    cout << "The number is 0 and it is neither positive nor negative." << endl;
  }

  cout << "This line is always printed." << endl;

  return 0;
}
```

Nested if-else

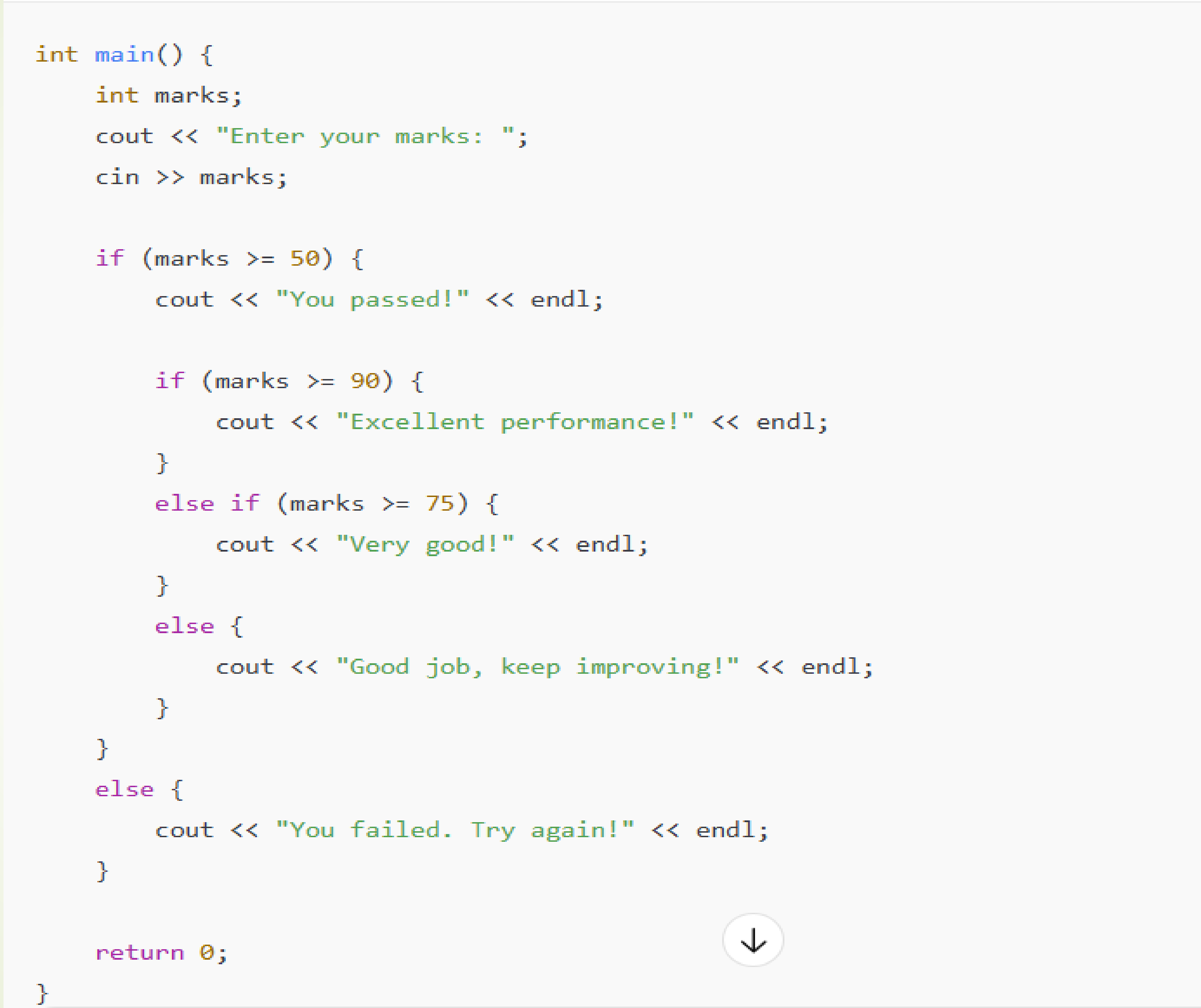


```
if(condition1)
{
    // Code to be executed
    if(condition2)
    {
        // Code to be executed
    }
    else
    {
         // Code to be executed
    }
}
else
{
    // code to be executed
}
```

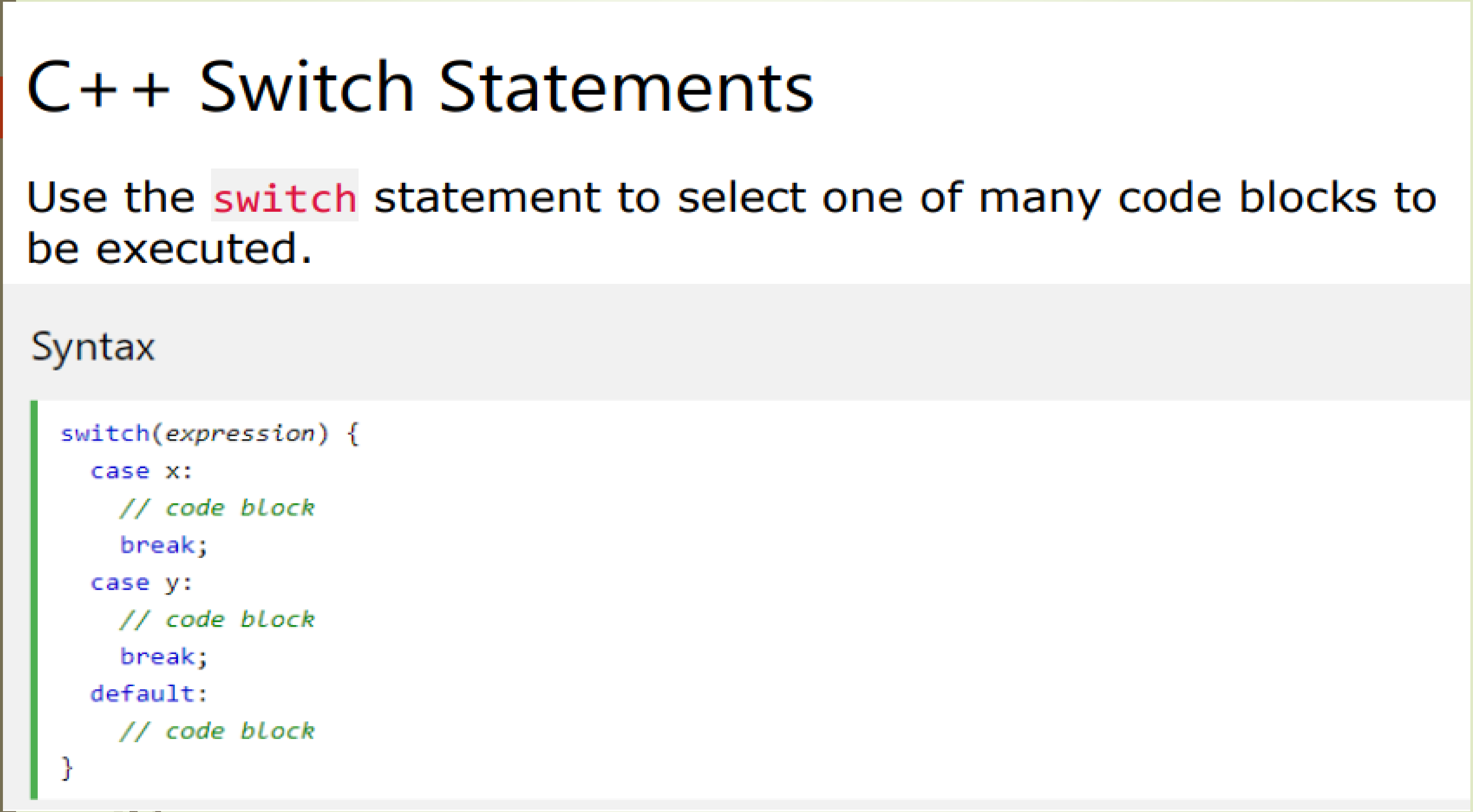


```cpp
#include <iostream>
using namespace std;

int main() {

  int num;

  cout << "Enter an integer: ";
   cin >> num;

  // outer if condition
  if (num != 0) {

    // inner if condition
    if (num > 0) {
      cout << "The number is positive." << endl;
    }
    // inner else condition
    else {
      cout << "The number is negative." << endl;
    }
  }
  // outer else condition
  else {
    cout << "The number is 0 and it is neither positive nor negative." << endl;
  }
```

Another example



```
int main() {
    int marks;
    cout << "Enter your marks: ";
    cin >> marks;

    if (marks >= 50) {
        cout << "You passed!" << endl;

        if (marks >= 90) {
            cout << "Excellent performance!" << endl;
        }
        else if (marks >= 75) {
            cout << "Very good!" << endl;
        }
        else {
            cout << "Good job, keep improving!" << endl;
        }
    }
    else {
        cout << "You failed. Try again!" << endl;
    }

    return 0;
}
```

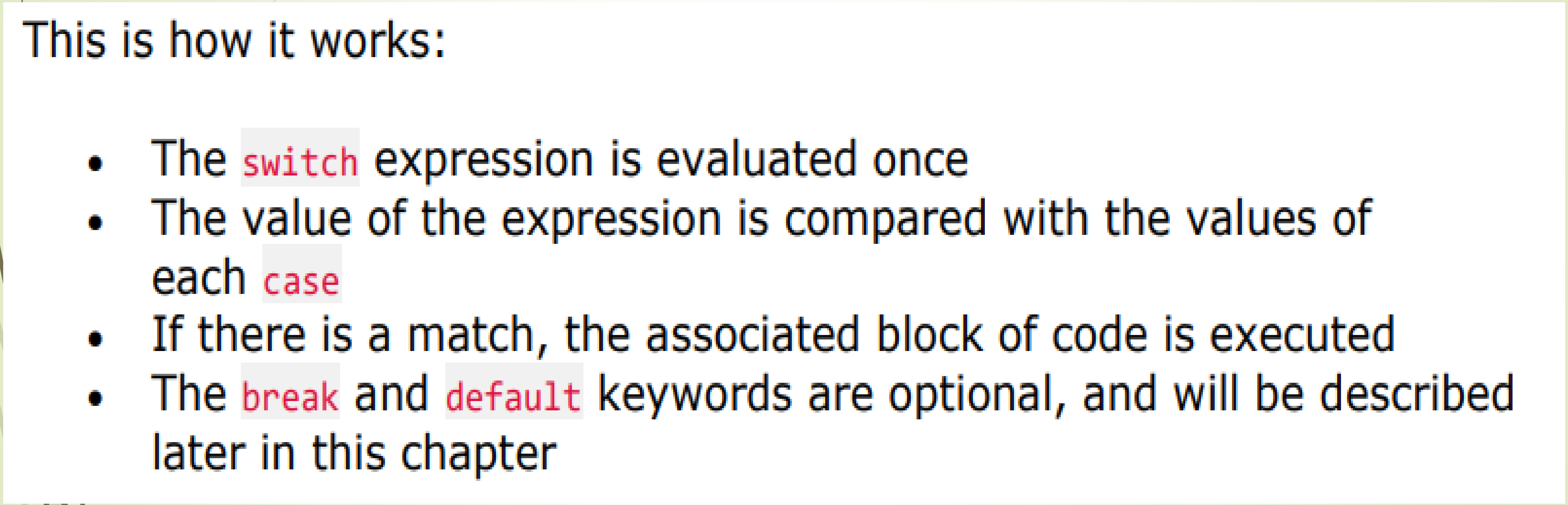


C++ Switch Statements

Use the `switch` statement to select one of many code blocks to be executed.

Syntax

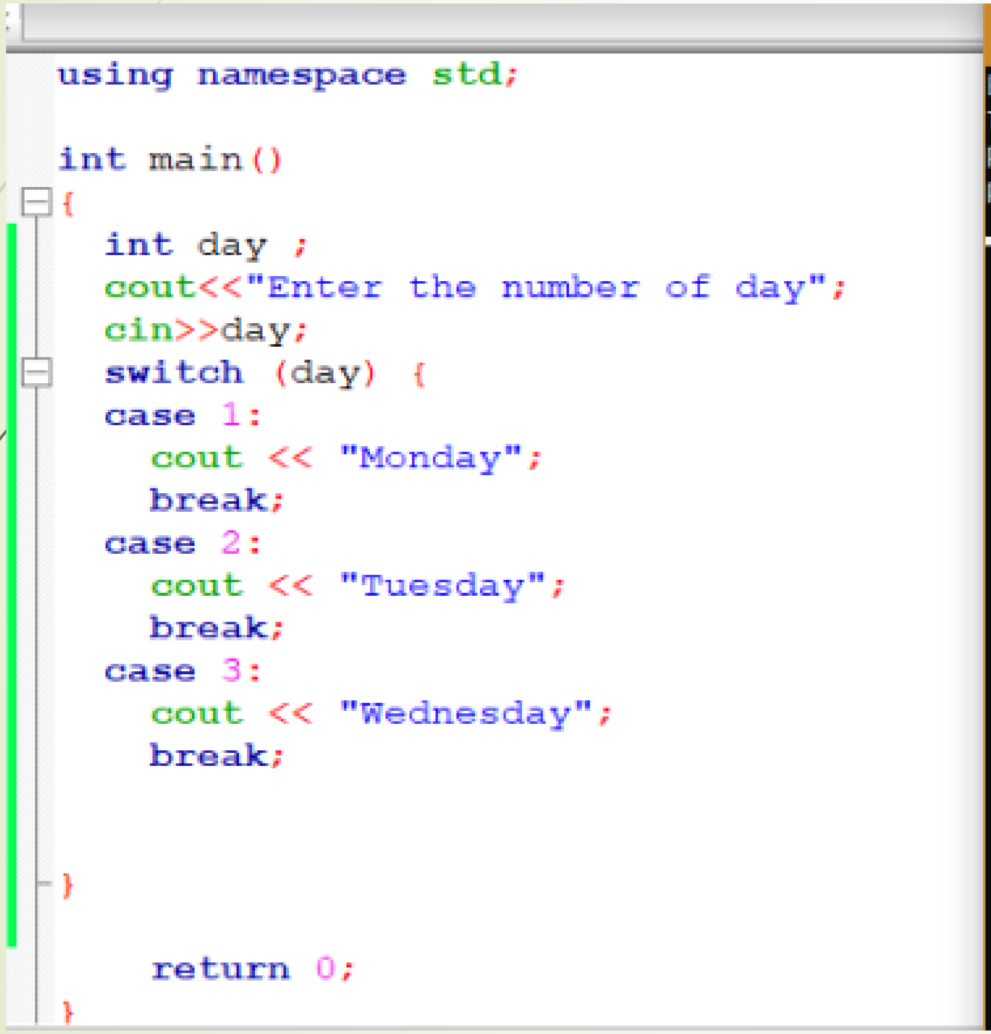
```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```



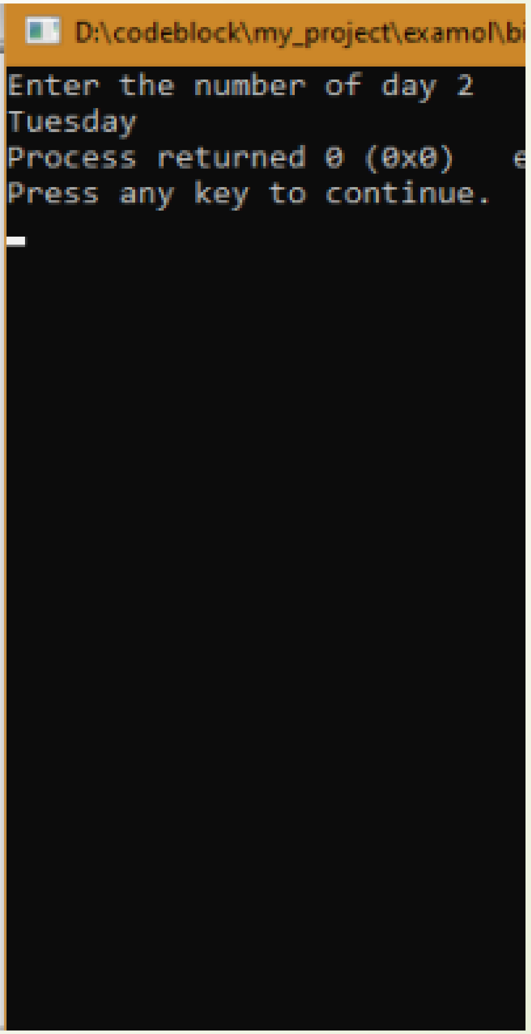
This is how it works:

- The `switch` expression is evaluated once
- The value of the expression is compared with the values of each `case`
- If there is a match, the associated block of code is executed
- The `break` and `default` keywords are optional, and will be described later in this chapter

Example_1



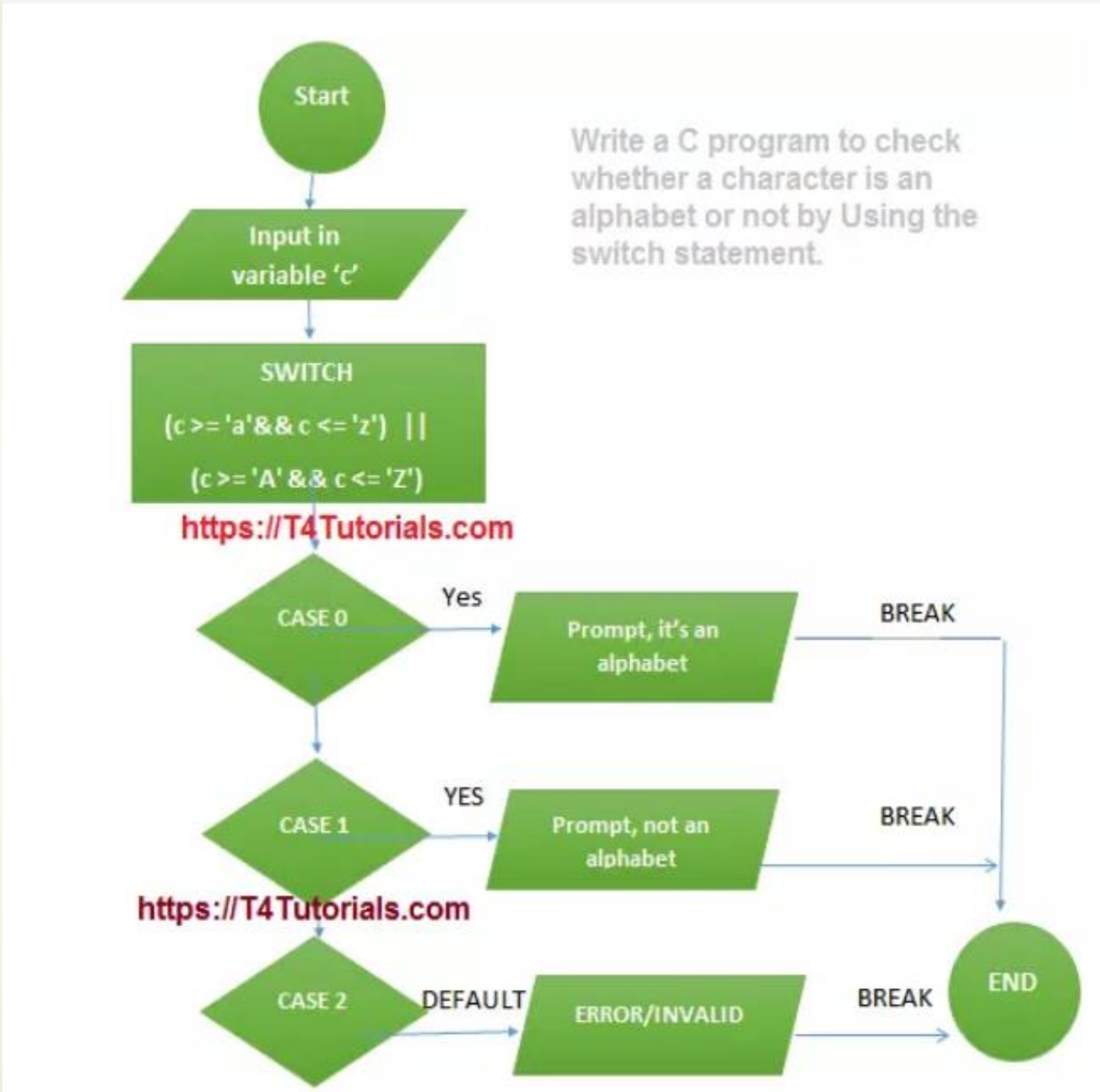
```
using namespace std;

int main()
{
  int day ;
  cout<<"Enter the number of day";
  cin>>day;
  switch (day) {
  case 1:
    cout << "Monday";
    break;
  case 2:
    cout << "Tuesday";
    break;
  case 3:
    cout << "Wednesday";
    break;


}


    return 0;
}
```



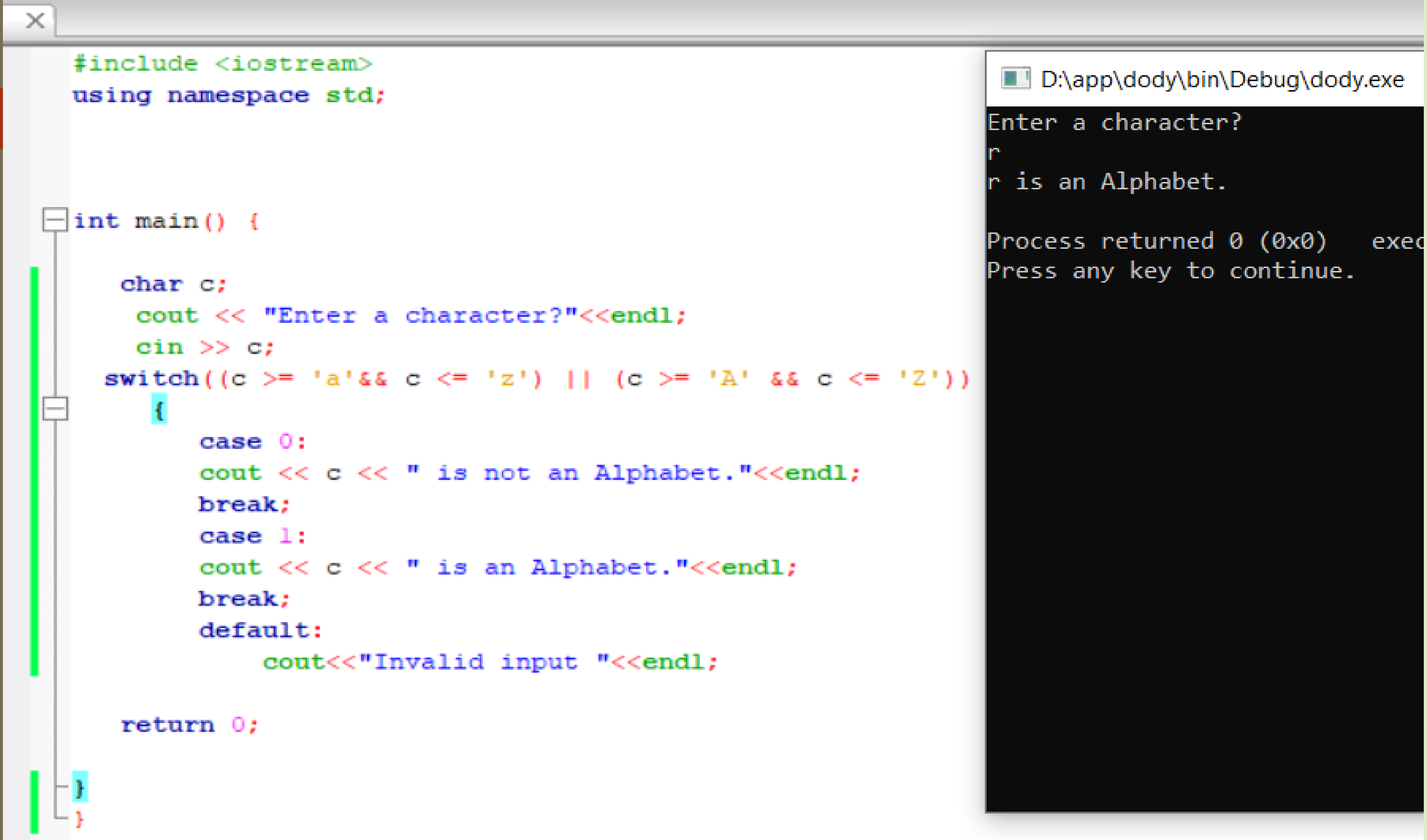
D:\codeblock\my_project\examol\bi

```
Enter the number of day 2
Tuesday
Process returned 0 (0x0)   e
Press any key to continue.
```

Example_2:
C program to check whether a character is an alphabet or not



Write a C program to check whether a character is an alphabet or not by Using the switch statement.

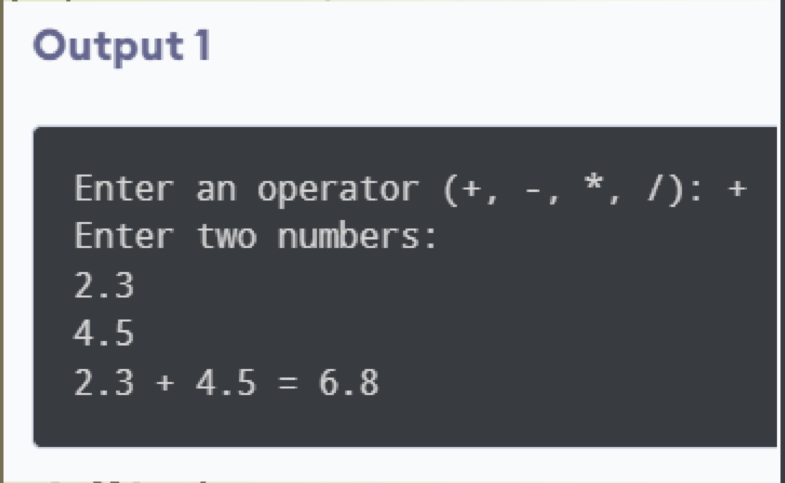


```cpp
#include <iostream>
using namespace std;



int main() {

    char c;
     cout << "Enter a character?"<<endl;
     cin >> c;
   switch((c >= 'a'&& c <= 'z') || (c >= 'A' && c <= 'Z'))
      {
         case 0:
         cout << c << " is not an Alphabet."<<endl;
         break;
         case 1:
         cout << c << " is an Alphabet."<<endl;
         break;
         default:
             cout<<"Invalid input "<<endl;

    return 0;

}
}
```

D:\app\dody\bin\Debug\dody.exe

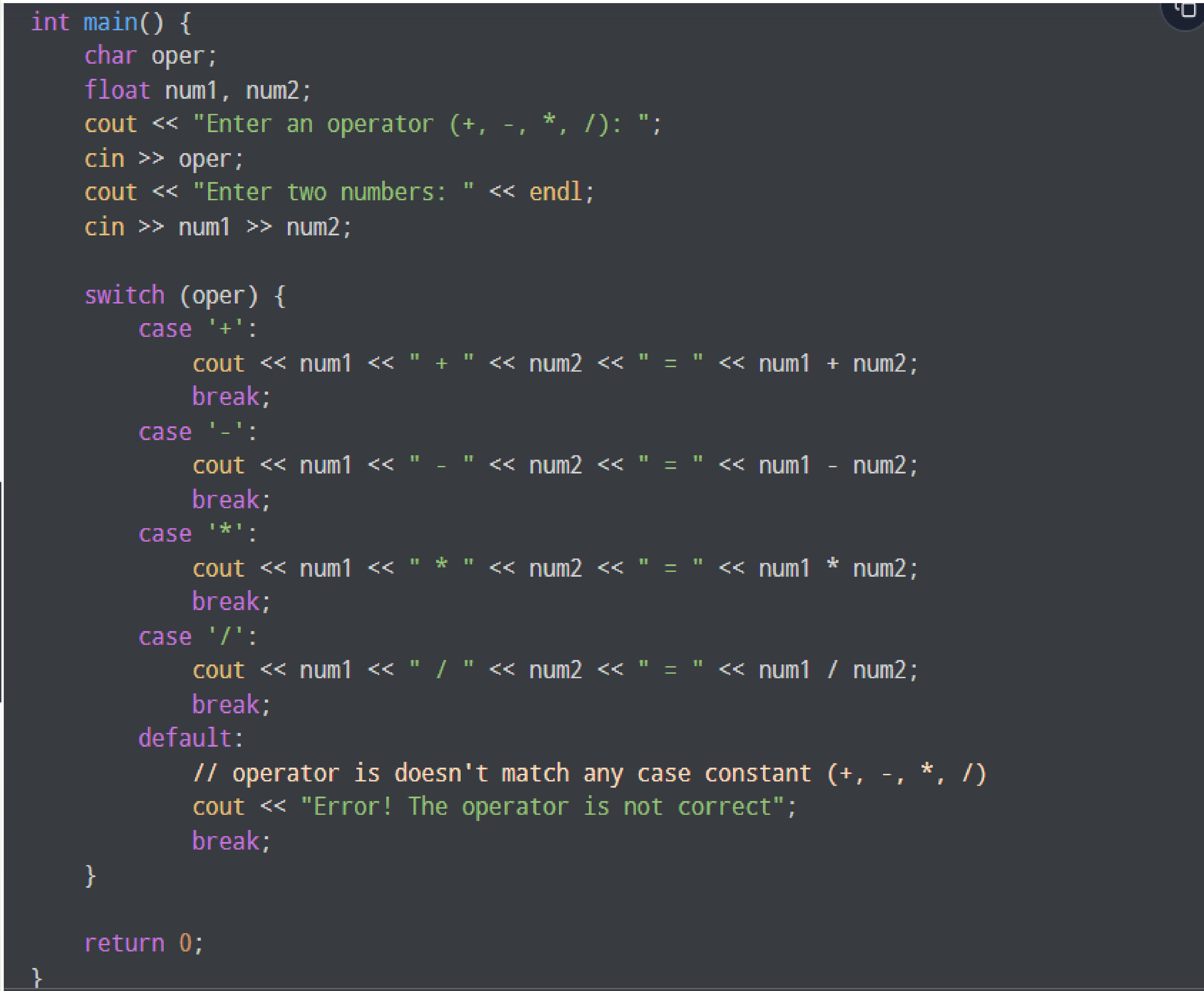
```
Enter a character?
r
r is an Alphabet.

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

Example: Create a Calculator using the switch Statement



Output 1

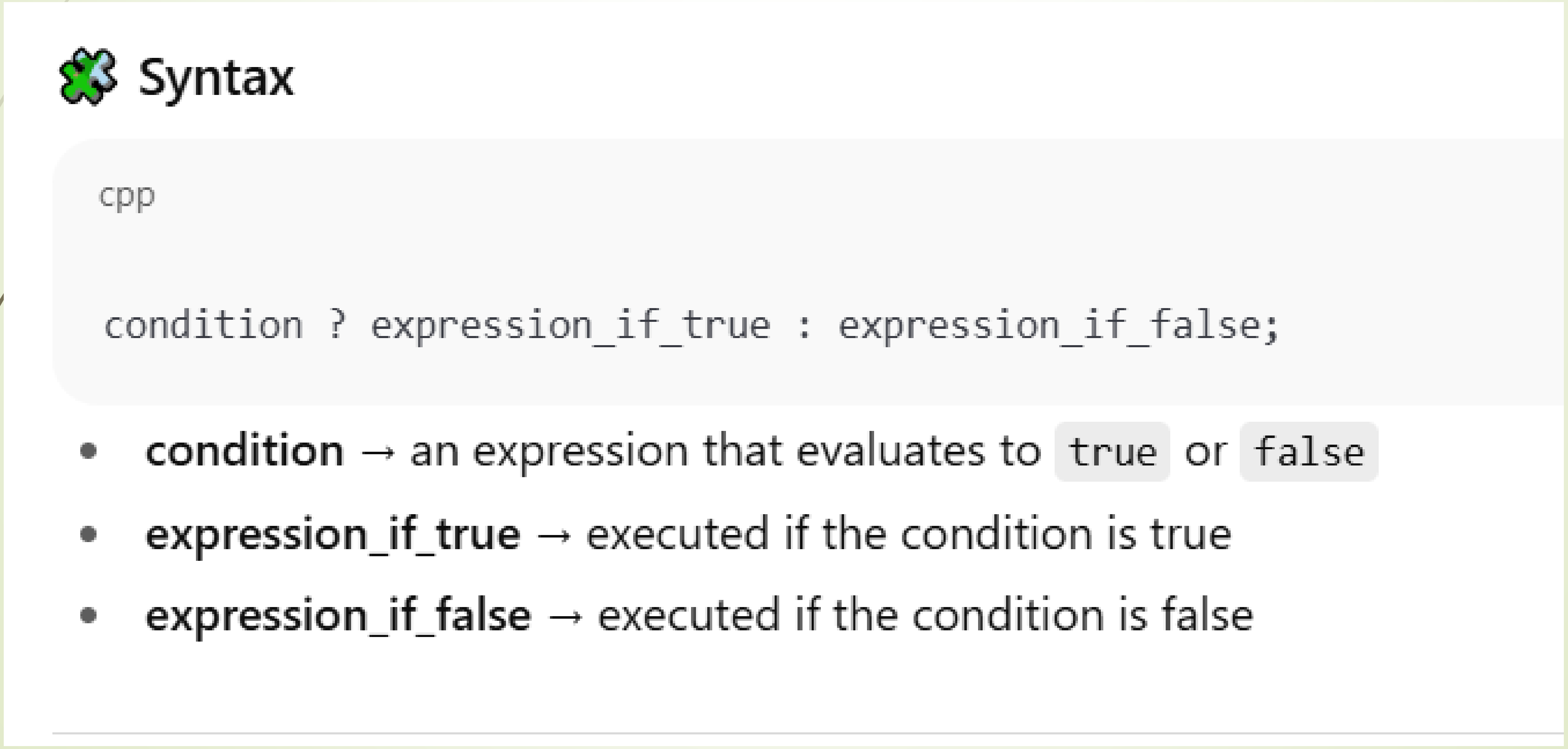
```
Enter an operator (+, -, *, /): +
Enter two numbers:
2.3
4.5
2.3 + 4.5 = 6.8
```



```cpp
int main() {
    char oper;
    float num1, num2;
    cout << "Enter an operator (+, -, *, /): ";
    cin >> oper;
    cout << "Enter two numbers: " << endl;
    cin >> num1 >> num2;

    switch (oper) {
        case '+':
            cout << num1 << " + " << num2 << " = " << num1 + num2;
            break;
        case '-':
            cout << num1 << " - " << num2 << " = " << num1 - num2;
            break;
        case '*':
            cout << num1 << " * " << num2 << " = " << num1 * num2;
            break;
        case '/':
            cout << num1 << " / " << num2 << " = " << num1 / num2;
            break;
        default:
            // operator is doesn't match any case constant (+, -, *, /)
            cout << "Error! The operator is not correct";
            break;
    }

    return 0;
}
```

Ternary operator

- The **ternary operator** is a **short form of the if...else statement**. It is called *ternary* because it takes **three operands**.

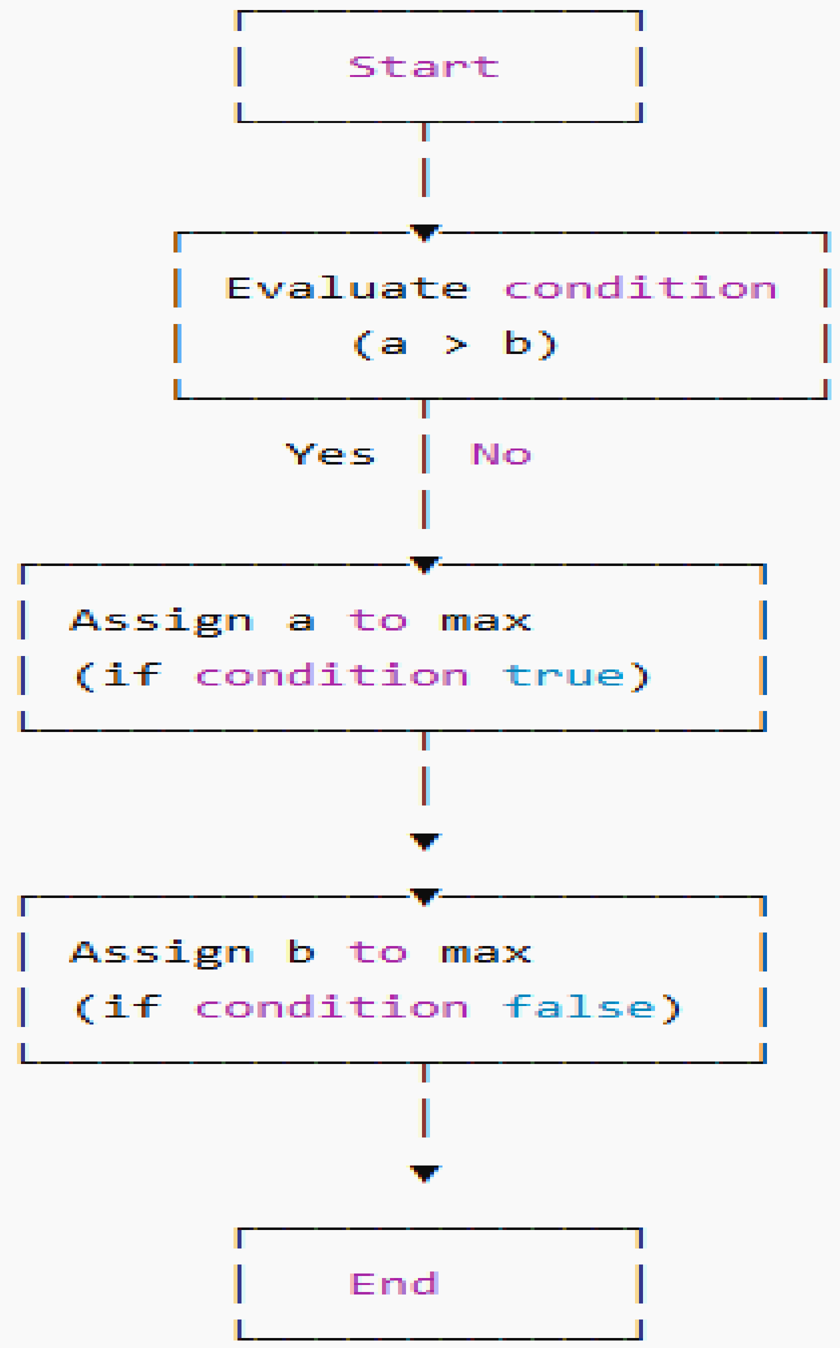


Syntax

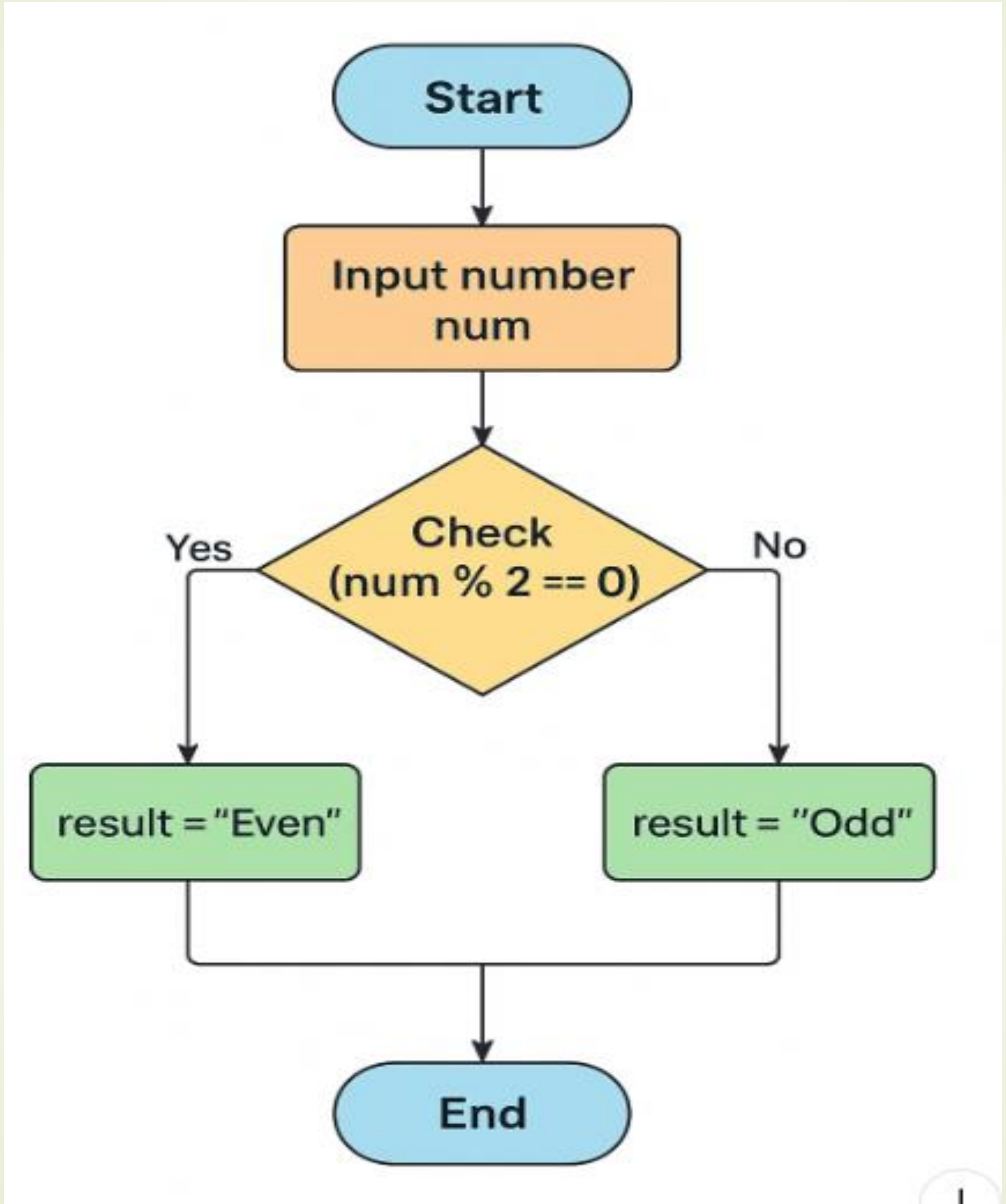
```cpp
condition ? expression_if_true : expression_if_false;
```

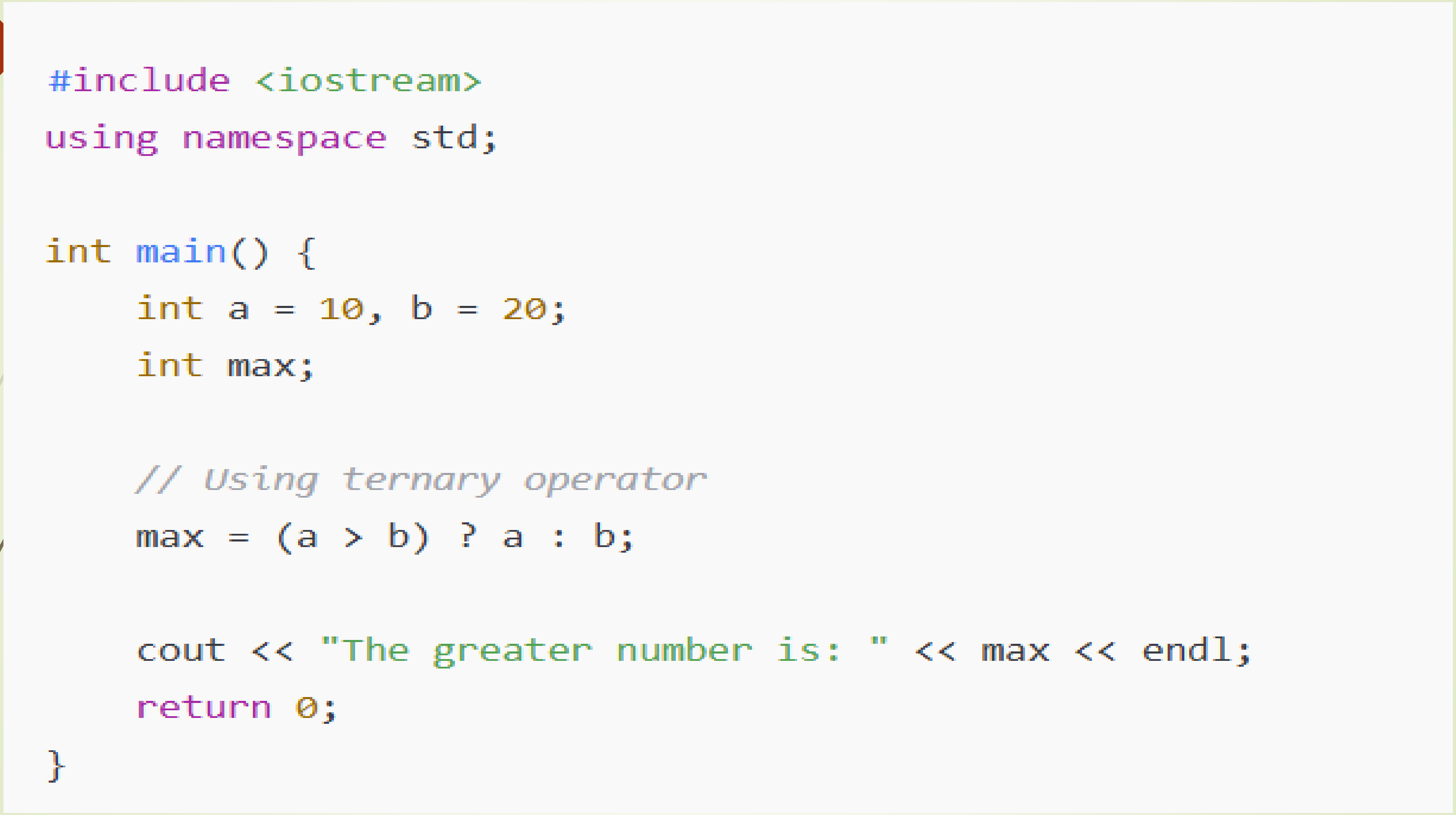
- **condition** → an expression that evaluates to `true` or `false`
- **expression_if_true** → executed if the condition is true
- **expression_if_false** → executed if the condition is false

A Flowchart (Step-by-Step Visual)

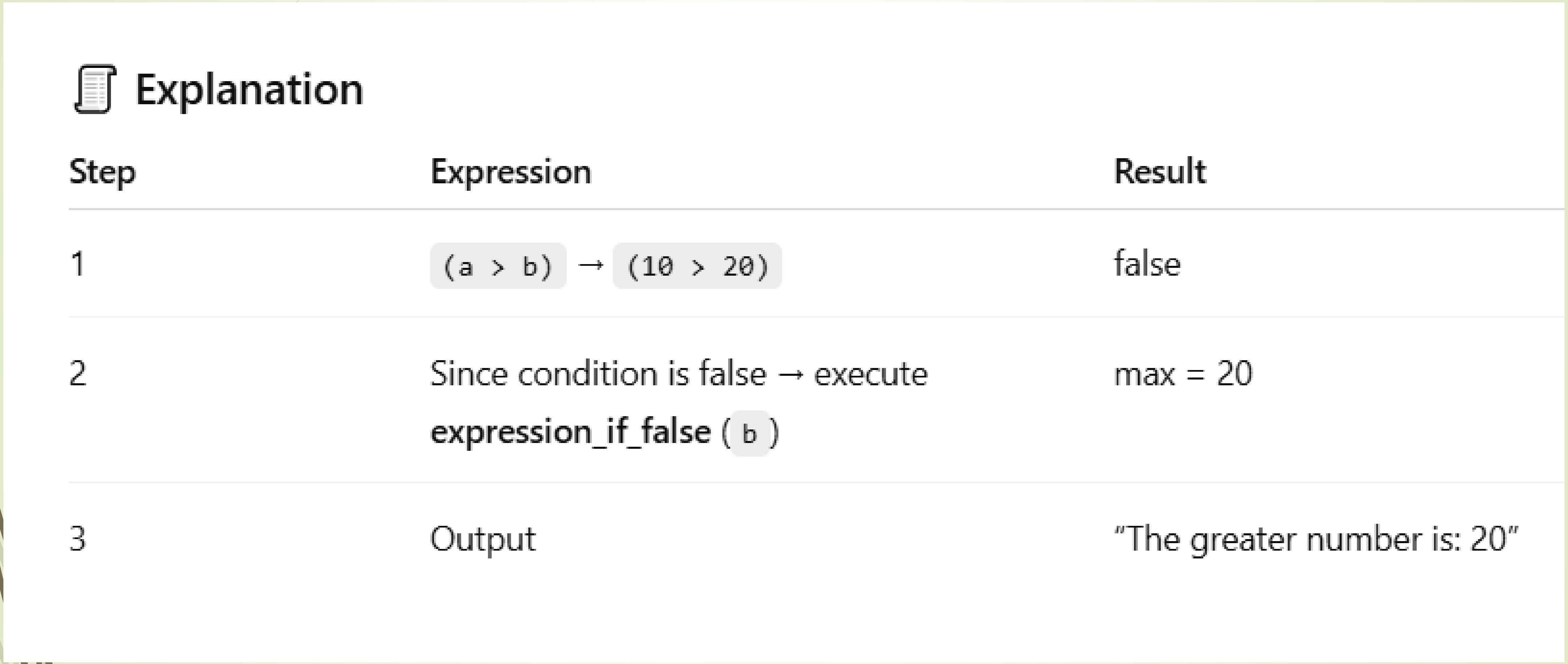


```
┌──────────────────────┐
│        Start         │
└──────────┬───────────┘
           │
┌──────────▼──────────────────┐
│ Evaluate condition          │
│      (a > b)                │
└──────────┬──────────────────┘
      Yes  │  No
           │
┌──────────▼──────────────────────┐
│ Assign a to max                 │
│ (if condition true)             │
└──────────┬──────────────────────┘
           │
           ▼
┌──────────▼──────────────────────┐
│ Assign b to max                 │
│ (if condition false)            │
└──────────┬──────────────────────┘
           │
           ▼
┌──────────────────────┐
│        End           │
└──────────────────────┘
```


Start
Input number num
Check (num % 2 == 0)
Yes
No
result = "Even"
result = "Odd"
End



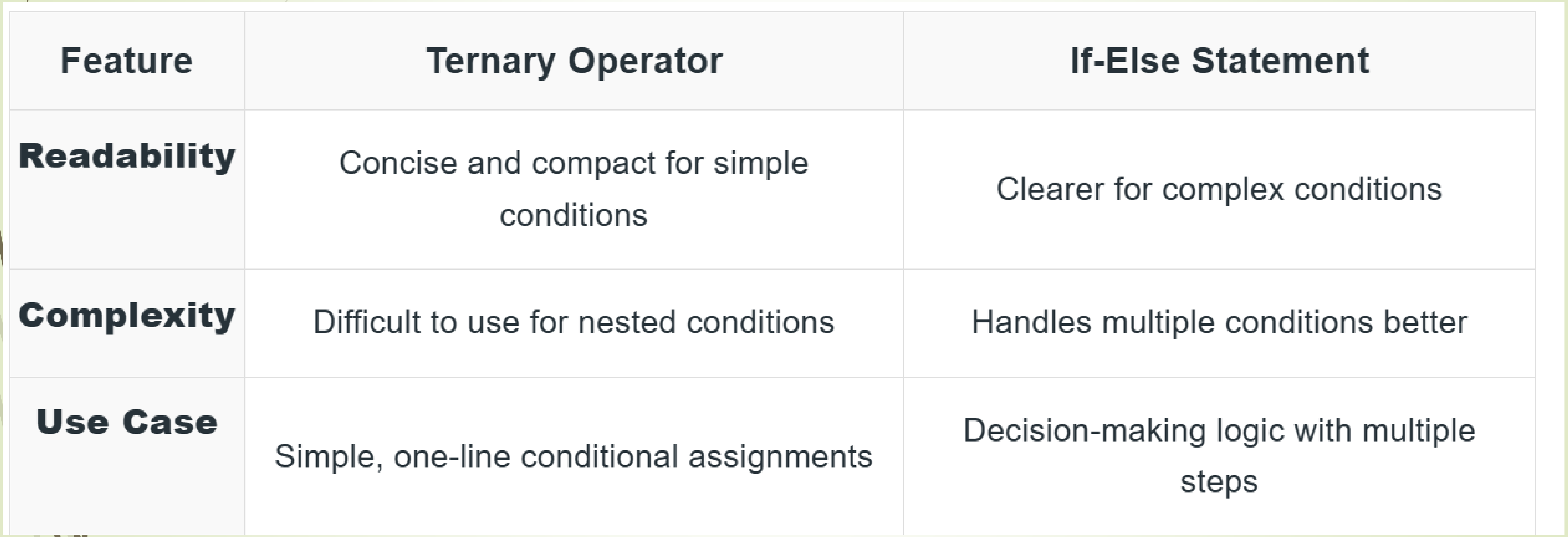
```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 10, b = 20;
    int max;

    // Using ternary operator
    max = (a > b) ? a : b;

    cout << "The greater number is: " << max << endl;
    return 0;
}
```



📜 Explanation

Step	Expression	Result
1	`(a > b)` → `(10 > 20)`	false
2	Since condition is false → execute **expression_if_false** (`b`)	max = 20
3	Output	"The greater number is: 20"

Ternary Operator vs If-Else Statements



Feature	Ternary Operator	If-Else Statement
Readability	Concise and compact for simple conditions	Clearer for complex conditions
Complexity	Difficult to use for nested conditions	Handles multiple conditions better
Use Case	Simple, one-line conditional assignments	Decision-making logic with multiple steps

Exercises

- Write an if-else statement that outputs the word High if the value of the variable score is greater than 100 and Low if the value of score is at most 100? The variables are of type int.

- Write an if-else statement that outputs the word Warning provided that either the value of the variable temperature is greater than or equal to 100, or the
of the variable pressure is greater than or equal to 200, or both. Otherwise, the if else statement outputs the word OK. The variables are of type int.

References

- https://www.programiz.com/cpp-programming/switch-case
- https://www.w3schools.com/cpp/cpp_conditions.asp
- https://www.geeksforgeeks.org/cpp/cpp-ternary-or-conditional-operator/