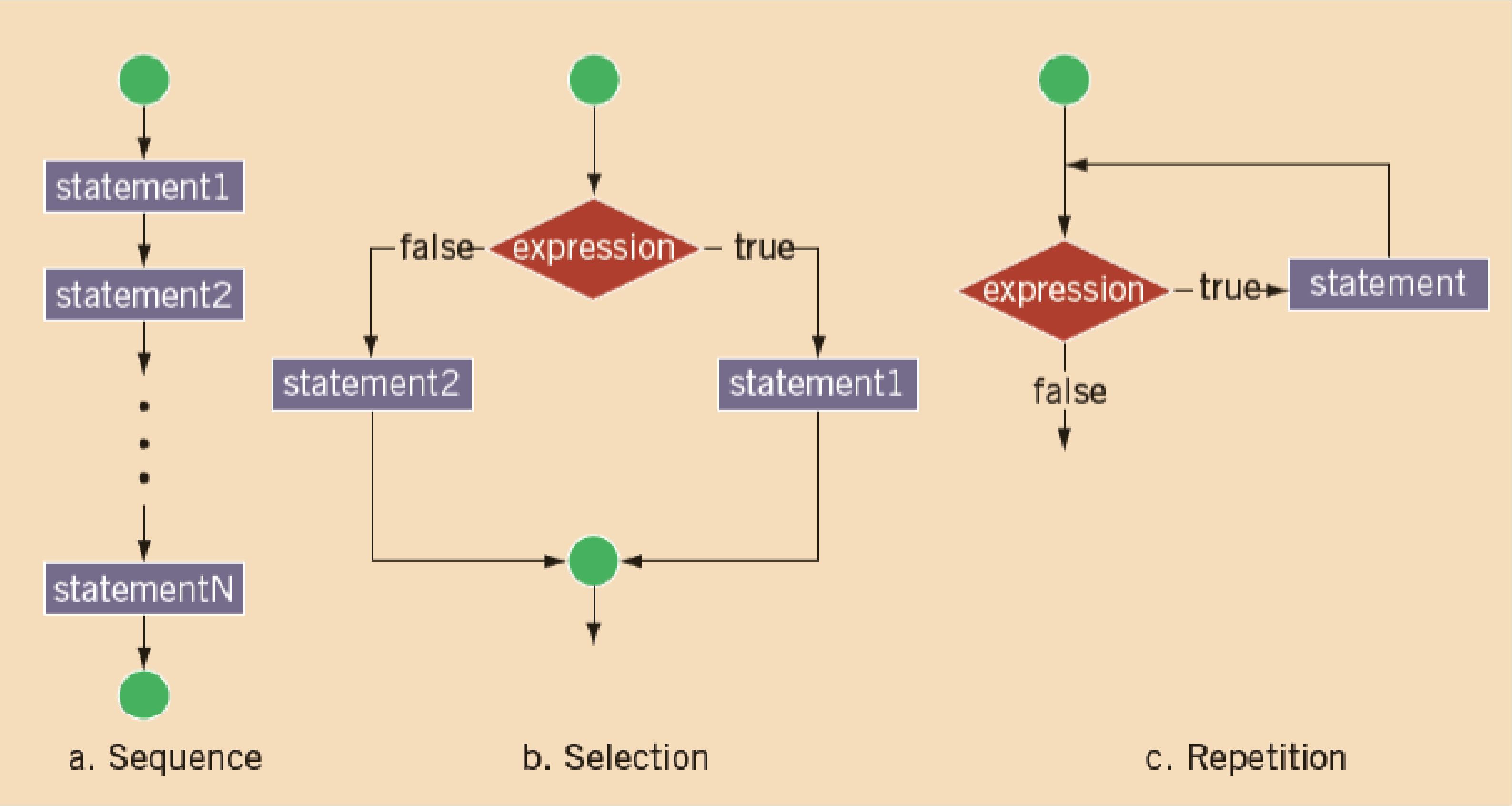
Lecture 06

M M Imran

Control Structures

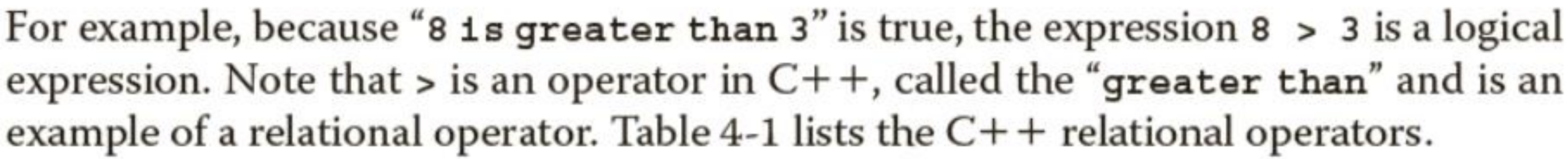
Any task solvable by a computer is accomplished by some combination of these operations :

- Sequence
- Selection
- Iteration



Logical Expression

An expression that evaluates to **true** or **false** is called a logical expression.

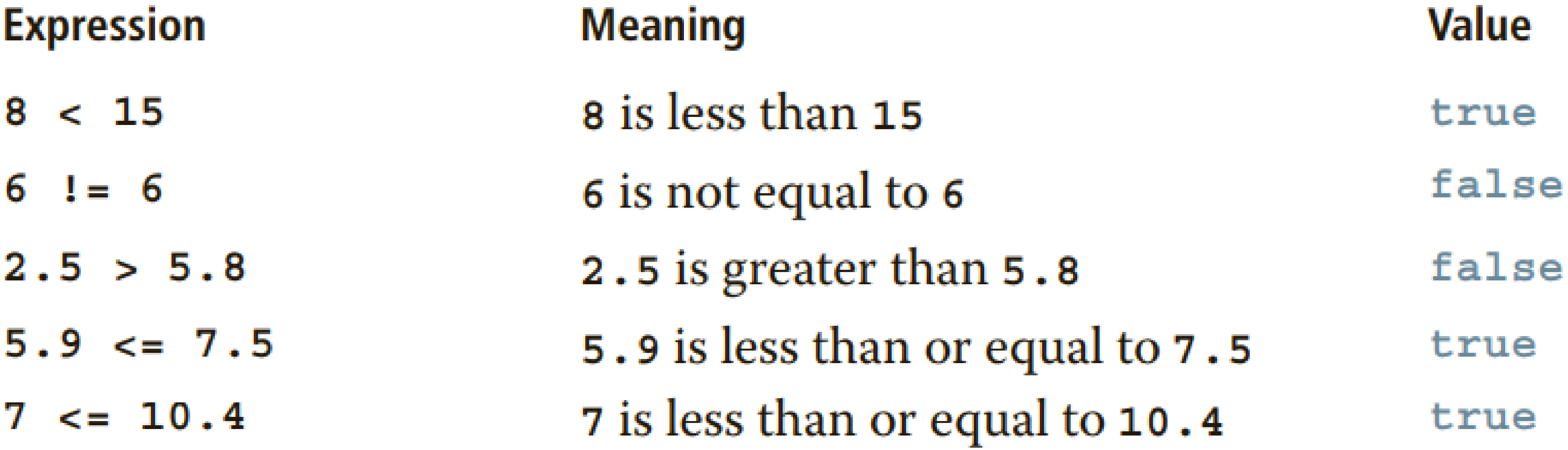


For example, because "`8 is greater than 3`" is true, the expression `8 > 3` is a logical expression. Note that `>` is an operator in C++, called the "`greater than`" and is an example of a relational operator. Table 4-1 lists the C++ relational operators.

Relational Operators

- A relational operator compares two expressions of the same data type.
- Note that the equality operator is different than the assignment operator.
- A relational operator is used to form a simple condition.
- A simple condition is either TRUE or FALSE.
- To determine the value of a simple condition, the expressions on either side of the operator are evaluated and the values compared.

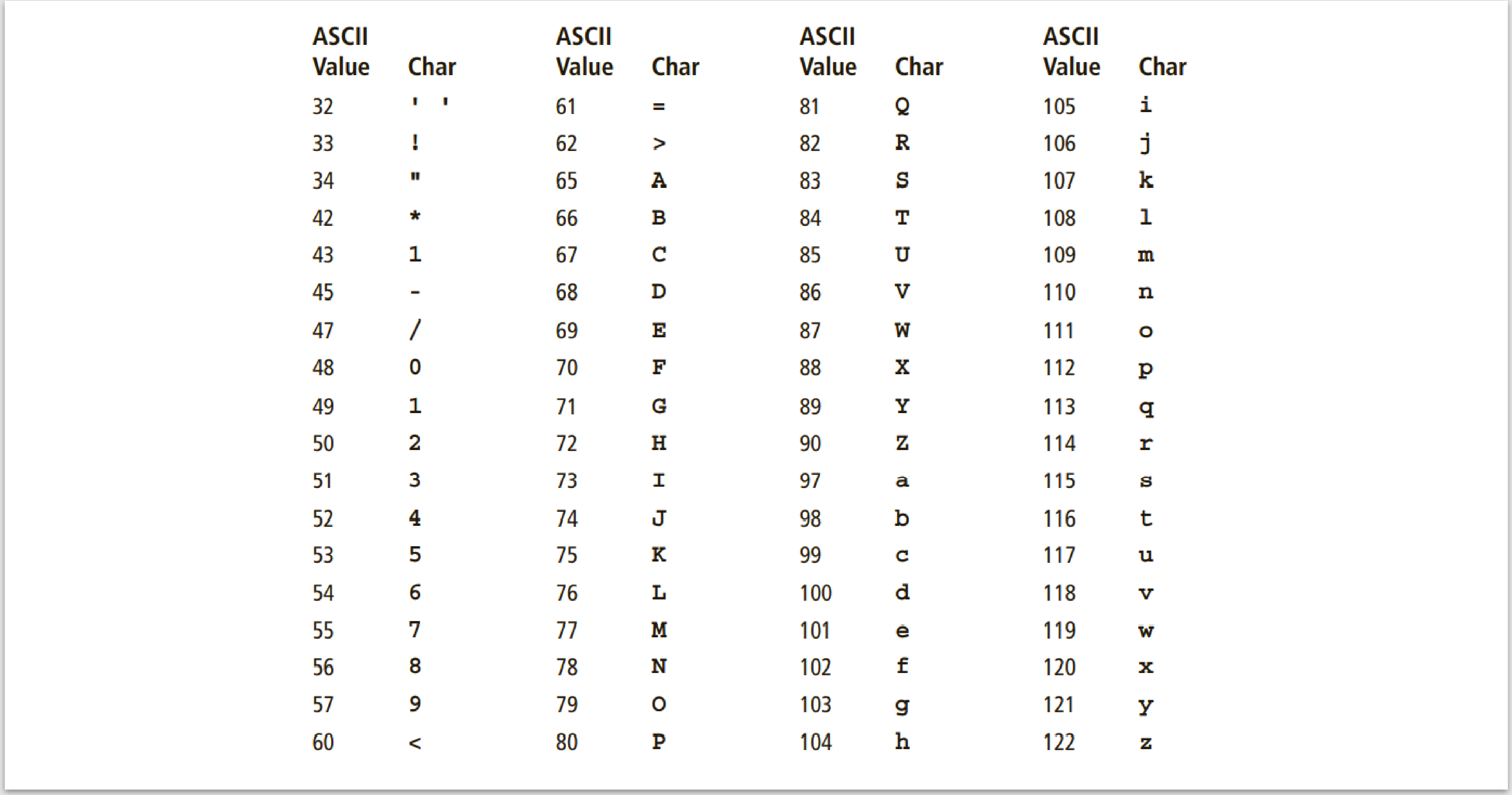
Operator	Purpose
==	Equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to



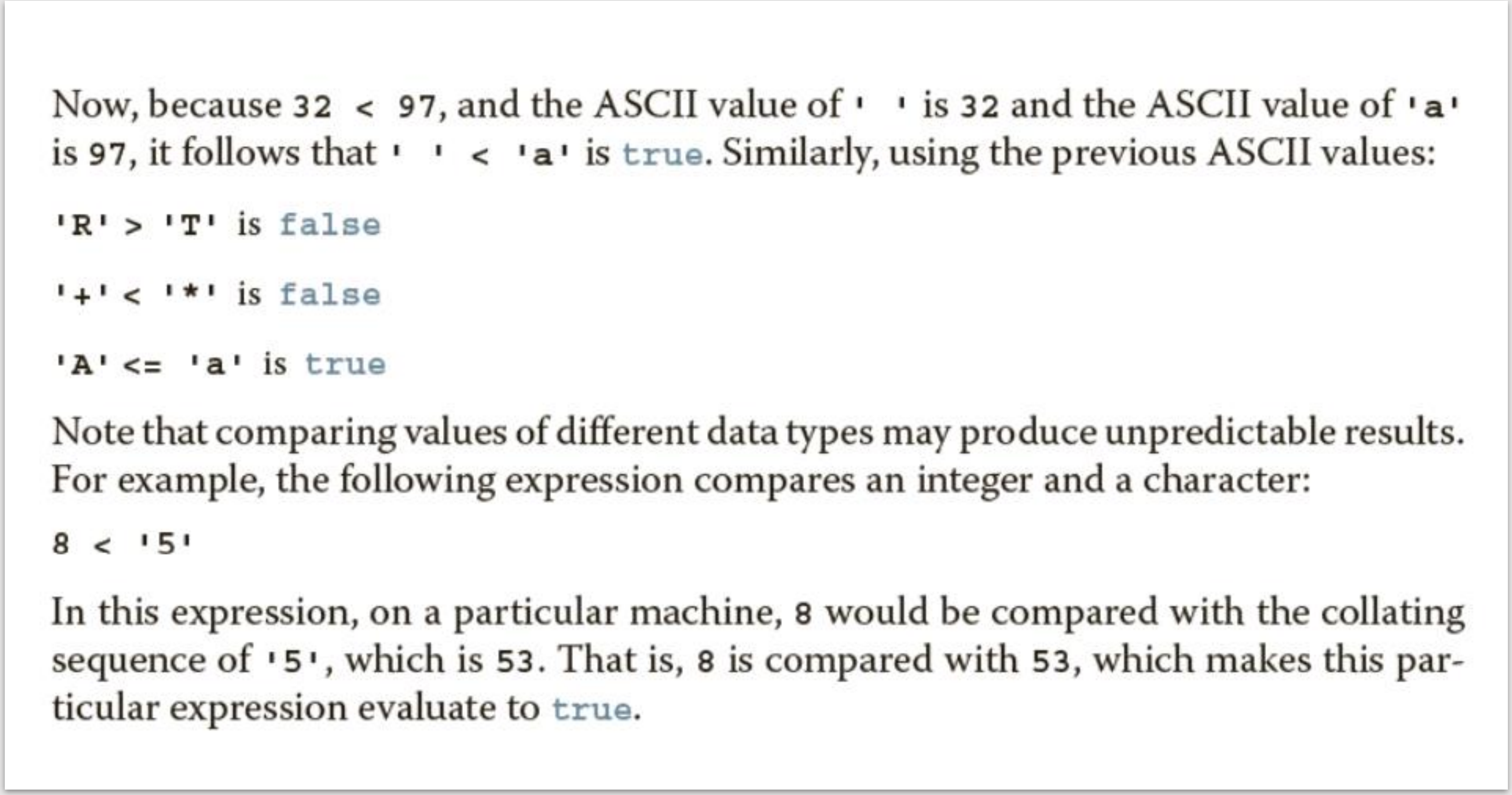
Expression	Meaning	Value
`8 < 15`	`8` is less than `15`	`true`
`6 != 6`	`6` is not equal to `6`	`false`
`2.5 > 5.8`	`2.5` is greater than `5.8`	`false`
`5.9 <= 7.5`	`5.9` is less than or equal to `7.5`	`true`
`7 <= 10.4`	`7` is less than or equal to `10.4`	`true`

Comparing chars and strings

- When two characters are compared:
  - Each character is converted to its ASCII/UNICODE code and the corresponding codes are compared.
  - When one character is less than another, it means that it comes before the other in the ASCII/UNICODE sequence.
- When two string literals are compared:
  - Since they are stored as character arrays, the memory address of each is compared. This is usually unnecessary.
- When two string variables, or a string variable and a string literal, are compared:
  - Corresponding characters from each string are compared from left to right.
  - Each character is converted to its ASCII/UNICODE code and the corresponding codes are compared.
  - When one character is less than another, it means that it comes before the other in the ASCII/UNICODE sequence.



ASCII Value	Char	ASCII Value	Char	ASCII Value	Char	ASCII Value	Char
32	' '	61	=	81	Q	105	i
33	!	62	>	82	R	106	j
34	"	65	A	83	S	107	k
42	*	66	B	84	T	108	l
43	1	67	C	85	U	109	m
45	-	68	D	86	V	110	n
47	/	69	E	87	W	111	o
48	0	70	F	88	X	112	p
49	1	71	G	89	Y	113	q
50	2	72	H	90	Z	114	r
51	3	73	I	97	a	115	s
52	4	74	J	98	b	116	t
53	5	75	K	99	c	117	u
54	6	76	L	100	d	118	v
55	7	77	M	101	e	119	w
56	8	78	N	102	f	120	x
57	9	79	O	103	g	121	y
60	<	80	P	104	h	122	z



Now, because `32 < 97`, and the ASCII value of `' '` is `32` and the ASCII value of `'a'` is `97`, it follows that `' ' < 'a'` is `true`. Similarly, using the previous ASCII values:

`'R' > 'T'` is `false`

`'+' < '*'` is `false`

`'A' <= 'a'` is `true`

Note that comparing values of different data types may produce unpredictable results. For example, the following expression compares an integer and a character:

```
8 < '5'
```

In this expression, on a particular machine, `8` would be compared with the collating sequence of `'5'`, which is `53`. That is, `8` is compared with `53`, which makes this particular expression evaluate to `true`.

Logical Operators

- A logical operator combines two simple conditions into a compound condition, or reverses the value of a simple condition.
- Each simple condition is TRUE or FALSE, and after applying the logical operator, the compound condition is TRUE or FALSE.
- There are three logical operators:

Operator	Purpose
!	Not
&&	And
||	Or

Truth Tables

- A truth table shows the result when two simple conditions are connected with a logical operator.
- Truth tables summarize how we combine two logical conditions based on AND, OR, and NOT.

&& (and) operator

The simple conditions on each side of the && must be true for the compound condition to be true. Otherwise the compound condition is false.

“and” operator truth table		
Left side	Right side	Result
True	True	True
True	False	False
False	True	False
False	False	False

|| (or) operator

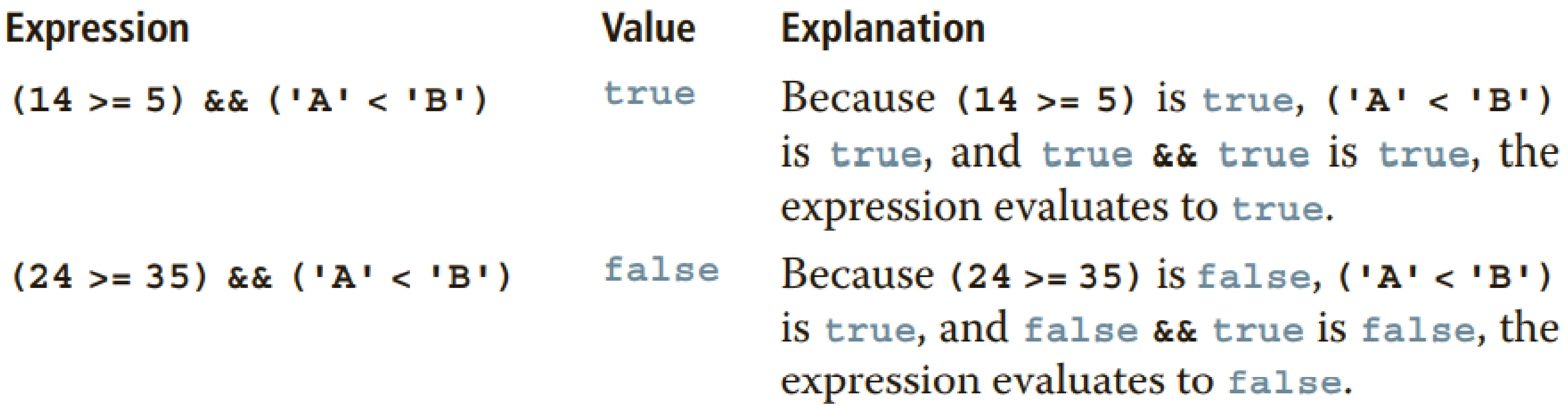
The simple conditions on each side of the || must be false for the compound condition to be false. Otherwise the compound condition is true.

“or” operator truth table		
Left side	Right side	Result
True	True	True
True	False	True
False	True	True
False	False	False

! (not) operator

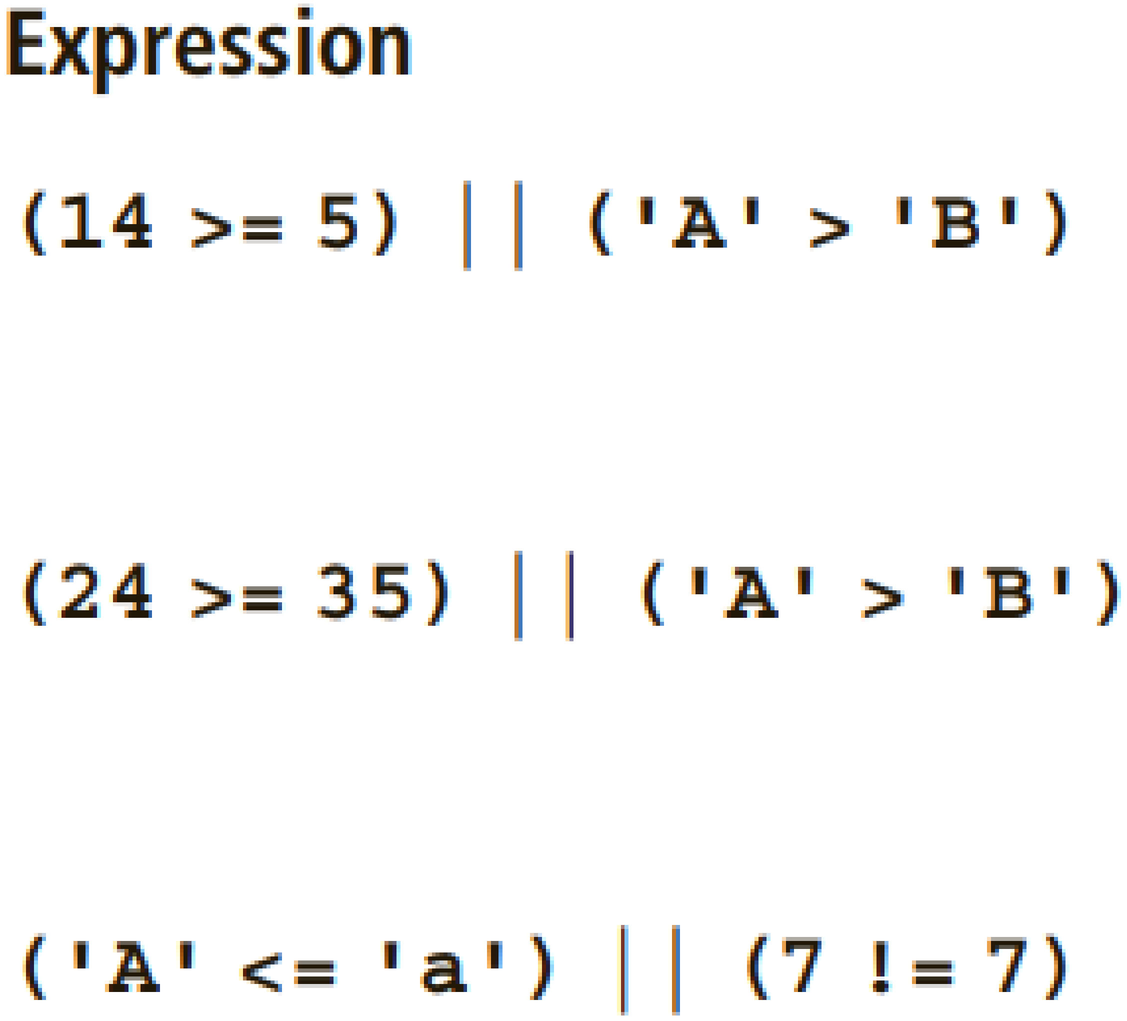
The value of the condition after the ! is reversed.

NOT operator truth table	
Condition	Result
True	False
False	True



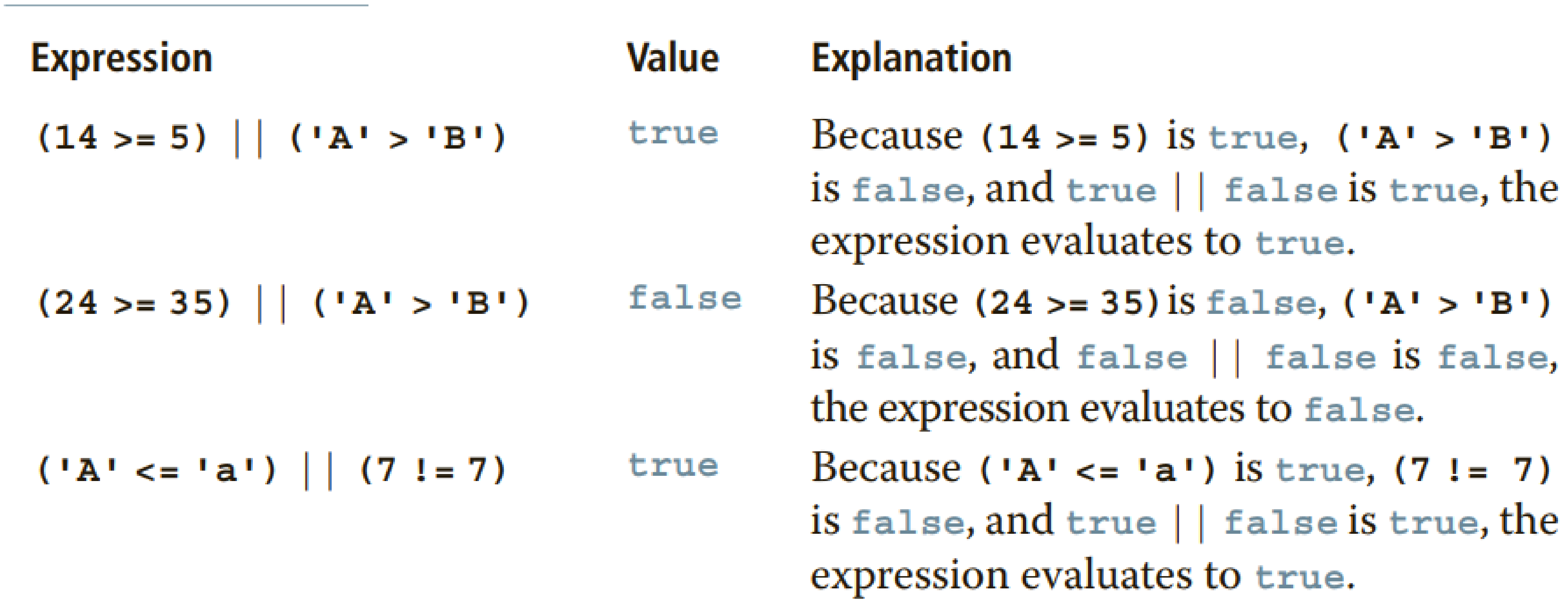
Expression	Value	Explanation
`(14 >= 5) && ('A' < 'B')`	`true`	Because `(14 >= 5)` is `true`, `('A' < 'B')` is `true`, and `true && true` is `true`, the expression evaluates to `true`.
`(24 >= 35) && ('A' < 'B')`	`false`	Because `(24 >= 35)` is `false`, `('A' < 'B')` is `true`, and `false && true` is `false`, the expression evaluates to `false`.

What should be the result of these?



Expression
`(14 >= 5) || ('A' > 'B')`
`(24 >= 35) || ('A' > 'B')`
`('A' <= 'a') || (7 != 7)`

Solution



Expression	Value	Explanation
`(14 >= 5) || ('A' > 'B')`	`true`	Because `(14 >= 5)` is `true`, `('A' > 'B')` is `false`, and `true || false` is `true`, the expression evaluates to `true`.
`(24 >= 35) || ('A' > 'B')`	`false`	Because `(24 >= 35)`is `false`, `('A' > 'B')` is `false`, and `false || false` is `false`, the expression evaluates to `false`.
`('A' <= 'a') || (7 != 7)`	`true`	Because `('A' <= 'a')` is `true`, `(7 != 7)` is `false`, and `true || false` is `true`, the expression evaluates to `true`.

Order of Precedence

Precedence	Operator type	Operators
Highest	Postfix	a++, a--
	Unary, prefix	!, +a, –a, ++a, --a
	Arithmetic	*, /, %
	Arithmetic	+, –
	Relational	<, <=, >, >=
	Relational	==, !=
	Logical	&&
	Logical	||
Lowest	Assignment, compound	=, +=, -=, *=, /=, %=

```
bool found = true;
int age = 20;
double hours = 45.30;
```

Expression	Value / Explanation
`!found`	`false` Because `found` is `true`, `!found` is `false`.
`hours > 40.00`	`true` Because `hours` is `45.30` and `45.30 > 40.00` is `true`, the expression `hours > 40.00` evaluates to `true`.
`!age`	`false` `age` is `20`, which is nonzero, so `age` evaluates to `true`. Therefore `!age` is `false`.
`!found && (age >= 18)`	`false` `!found` is `false`; `age > 18` is `20 > 18` is `true`. Therefore, `!found && (age >= 18)` is `false && true`, which evaluates to `false`.
`!(found && (age >= 18))`	`false` Now, `found && (age >= 18)` is `true && true`, which evaluates to `true`. Therefore, `!(found && (age >= 18))` is `!true`, which evaluates to `false`.

Block

- A block is a set of one or more statements.
- If a block contains:
    - One statement, curly braces are NOT required around the block.
    - More than one statement, curly braces are required around the block.
- A block has syntax:

```
<statement>;
   OR
<statement-1>;
<statement-2>;
…
<statement-n>;
```

if statement

- An if statement has one or more conditions and one or more execution paths.
- If a condition is true, the block following it is executed.
- An if statement has several variations

if statement

There are three forms of if...else statements in C++.

1. **if** statement
2. **if...else** statement
3. **if...else if...else** statement

if Statement

The syntax of the if statement is:

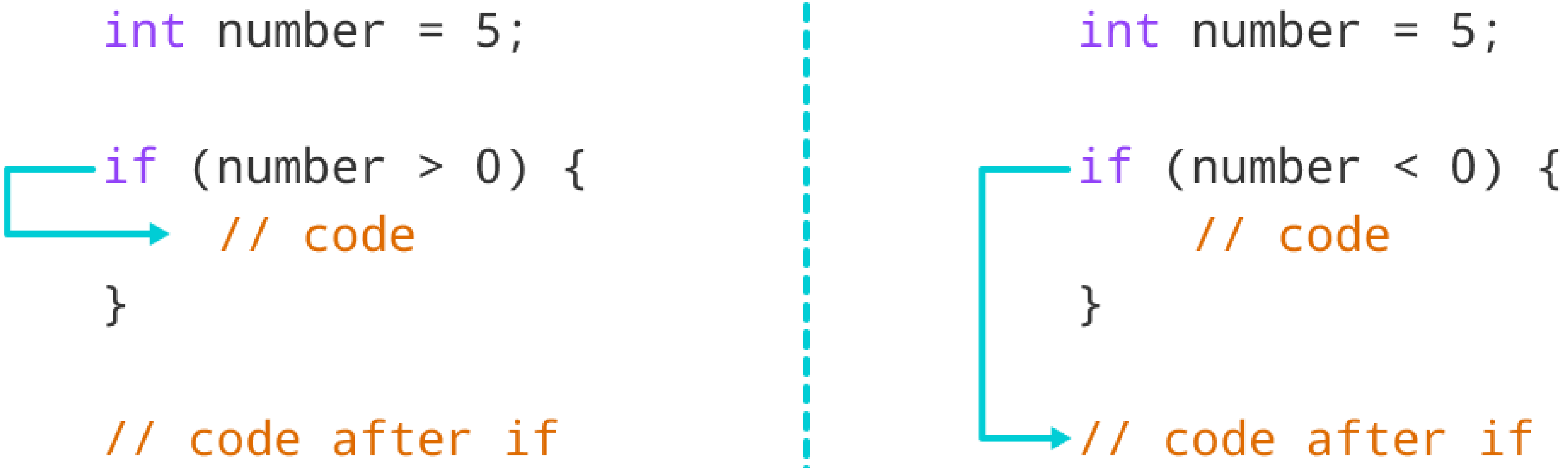
```
if (condition) {
  // body of if statement
}
```

The if statement evaluates the condition inside the parentheses ( )

- If the condition evaluates to true, the code inside the body of if is executed.
- If the condition evaluates to false, the code inside the body of if is skipped.

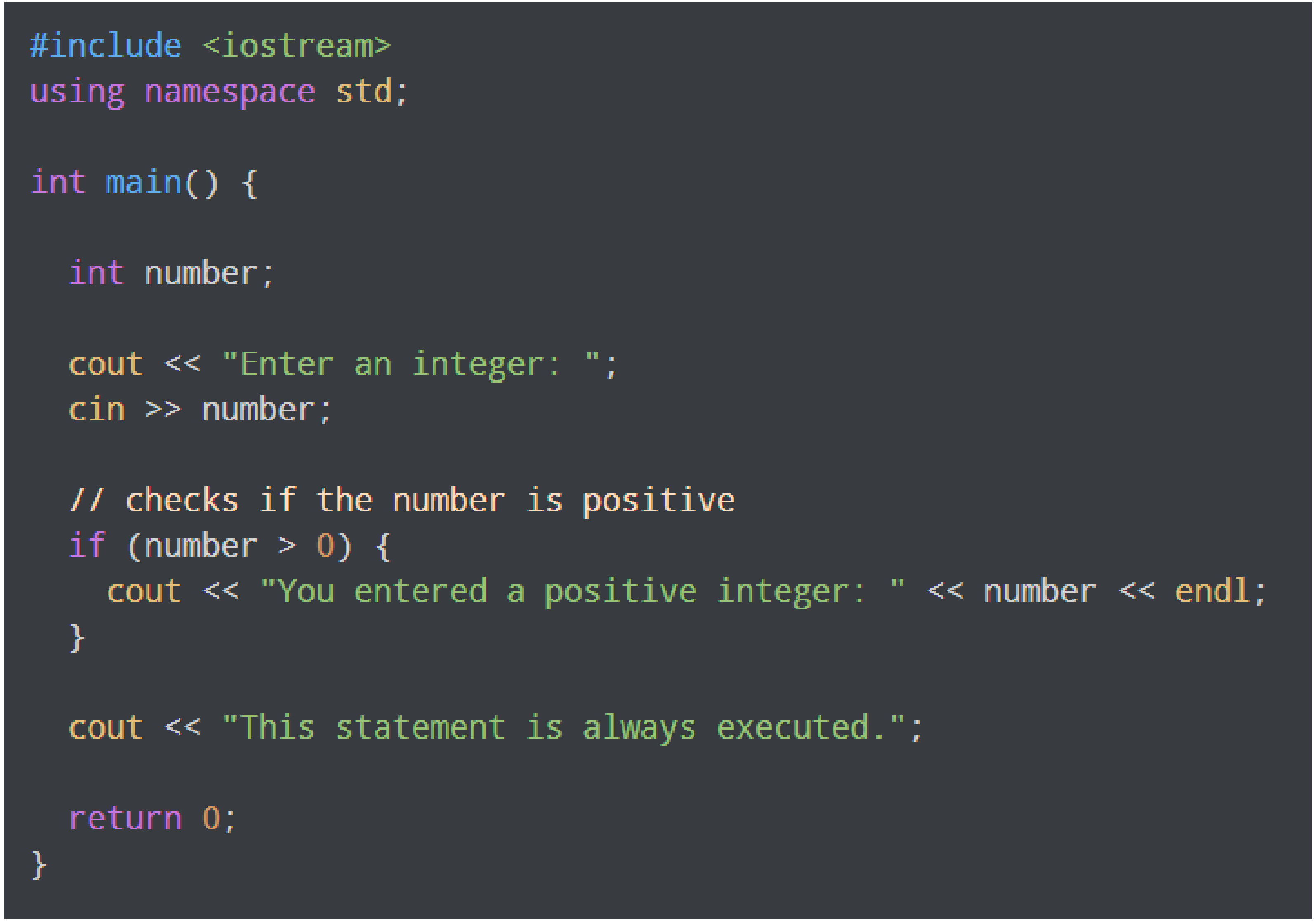
Condition is true

Condition is false

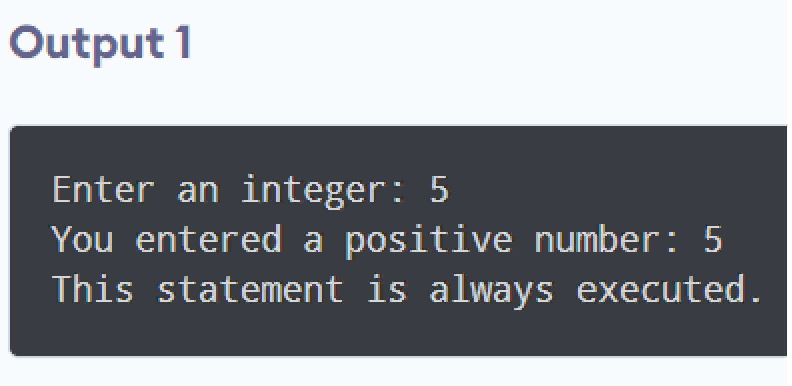


```
int number = 5;

if (number > 0) {
    // code
}

// code after if
```

```
int number = 5;

if (number < 0) {
    // code
}

// code after if
```

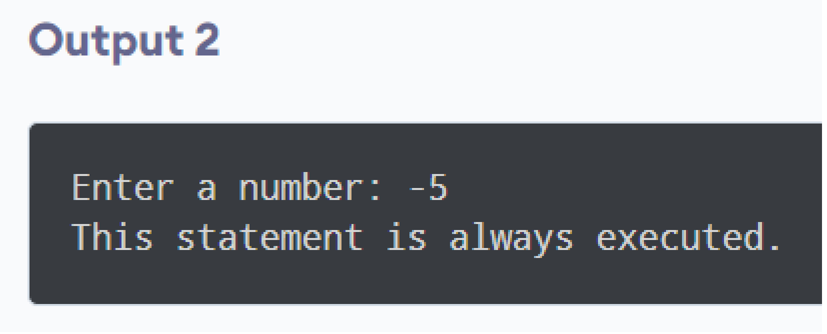


```cpp
#include <iostream>
using namespace std;

int main() {

  int number;

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

  // checks if the number is positive
  if (number > 0) {
    cout << "You entered a positive integer: " << number << endl;
  }

  cout << "This statement is always executed.";

  return 0;
}
```



Output 1

```
Enter an integer: 5
You entered a positive number: 5
This statement is always executed.
```



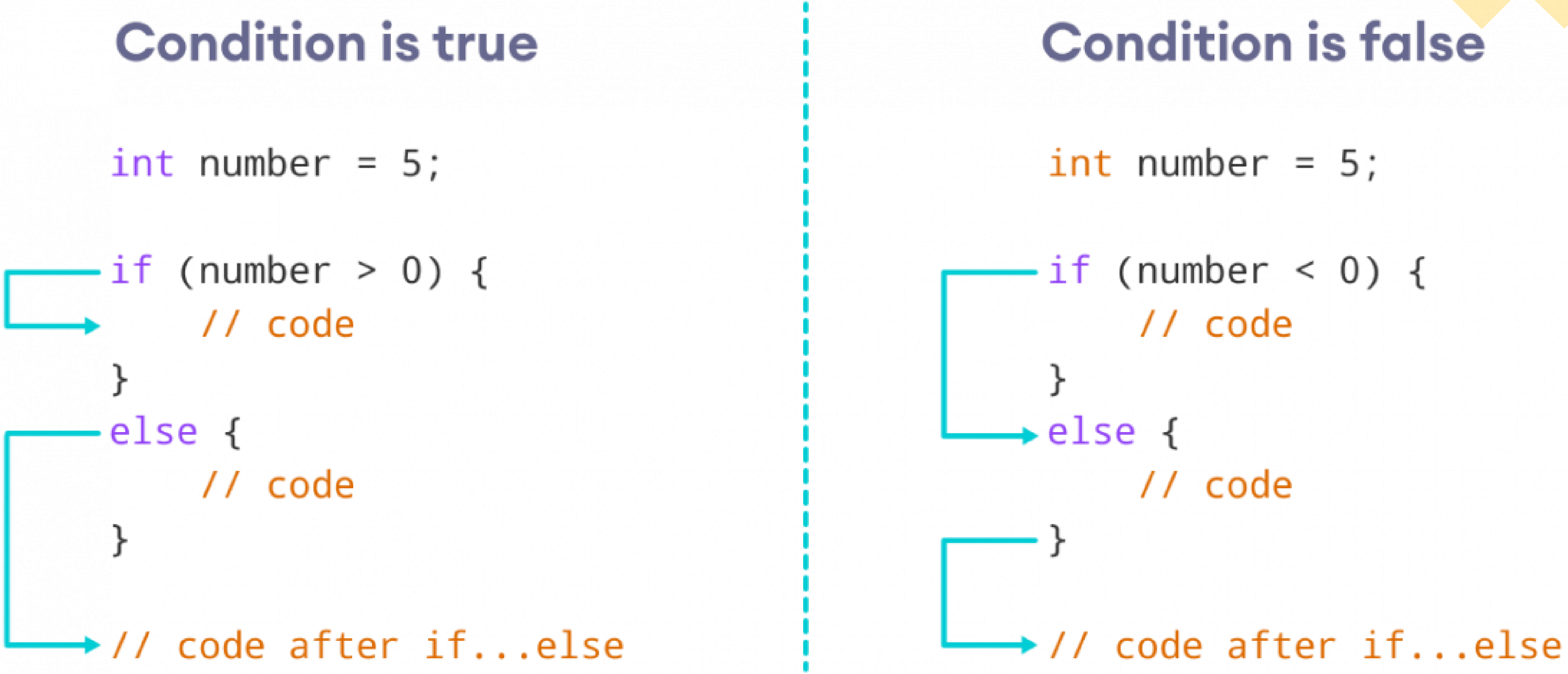
Output 2

```
Enter a number: -5
This statement is always executed.
```

if…else statement

The if statement can have an optional else clause. Its syntax is:

```
if (condition) {
  // block of code if condition is true
}
else {
  // block of code if condition is false
}
```

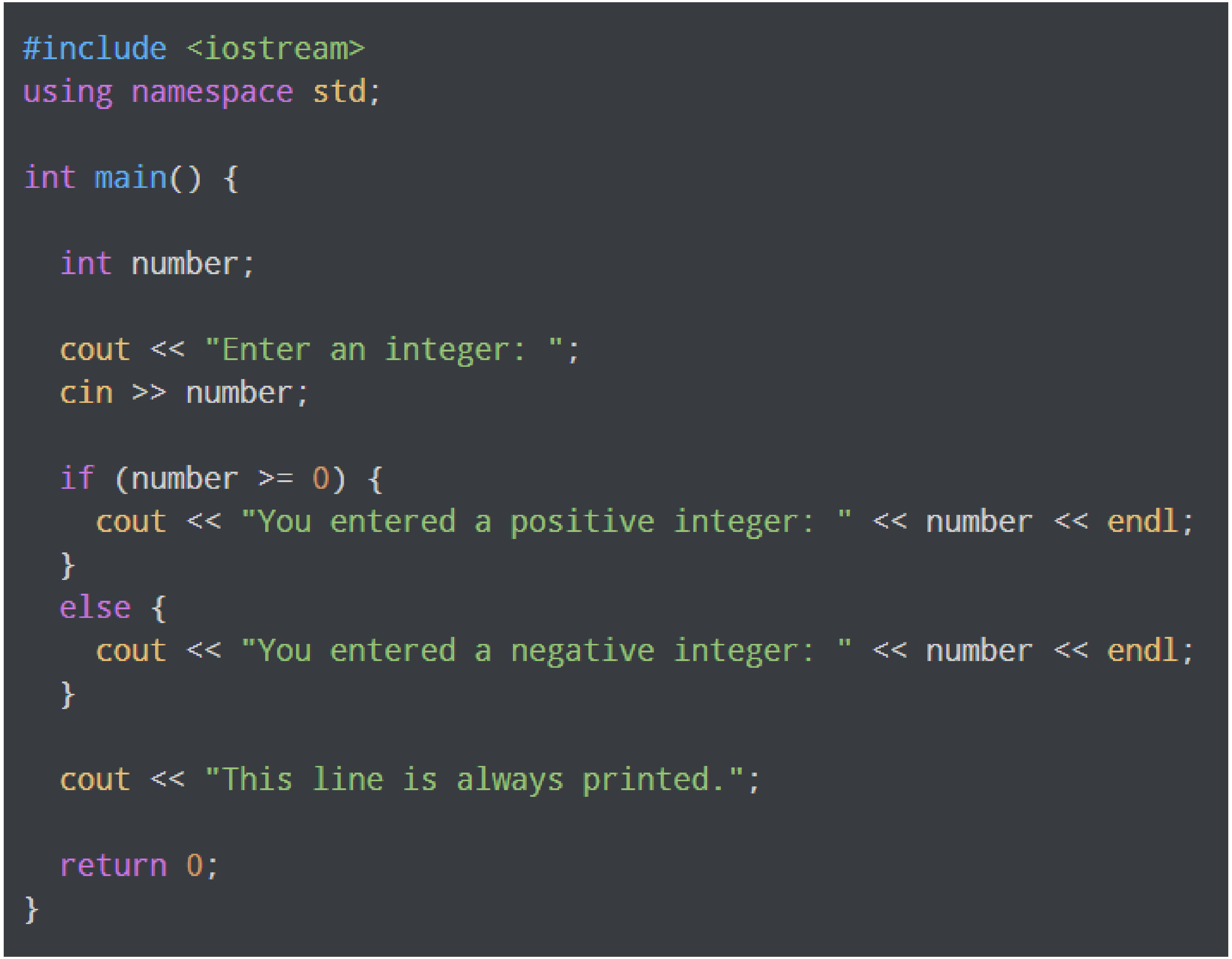


Condition is true

```
int number = 5;

if (number > 0) {
    // code
}
else {
    // code
}

// code after if...else
```

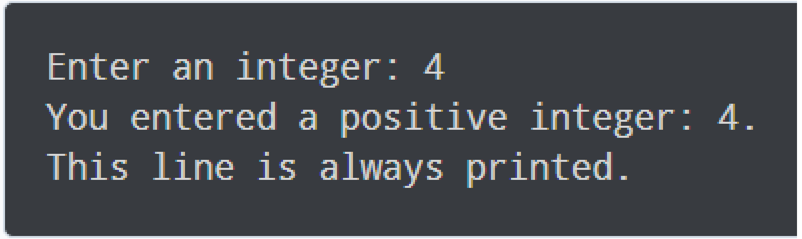
Condition is false

```
int number = 5;

if (number < 0) {
    // code
}
else {
    // code
}

// code after if...else
```



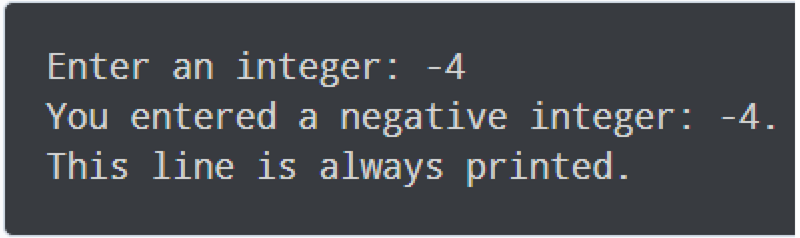
```cpp
#include <iostream>
using namespace std;

int main() {

  int number;

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

  if (number >= 0) {
    cout << "You entered a positive integer: " << number << endl;
  }
  else {
    cout << "You entered a negative integer: " << number << endl;
  }

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

  return 0;
}
```

Output 1



```
Enter an integer: 4
You entered a positive integer: 4.
This line is always printed.
```

Output 2



```
Enter an integer: -4
You entered a negative integer: -4.
This line is always printed.
```

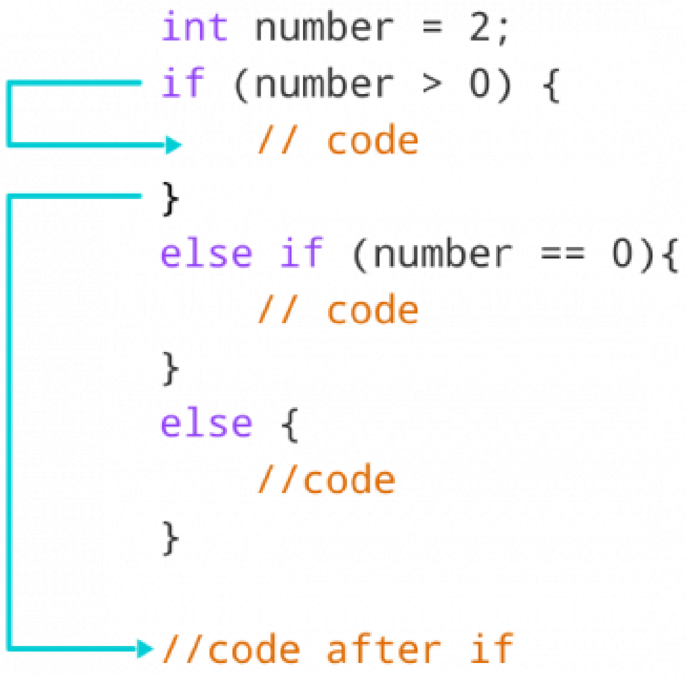
if…else…else if statement

The if...else statement is used to execute a block of code among two alternatives. However, if we need to make a choice between more than two alternatives, we use the if...else if...else statement.

The syntax of the if...else if...else statement is:

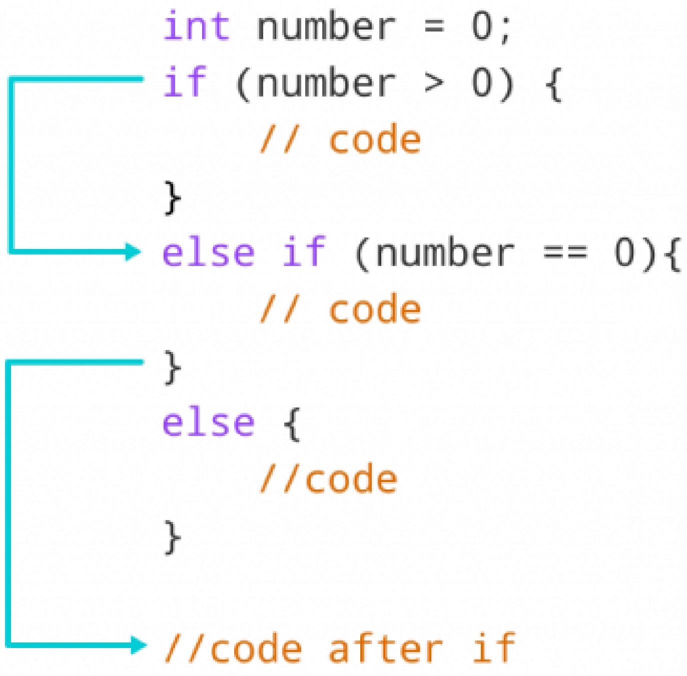
```
if (condition1) {
  // code block 1
}
else if (condition2){
  // code block 2
}
else {
  // code block 3
}
```

1st Condition is true



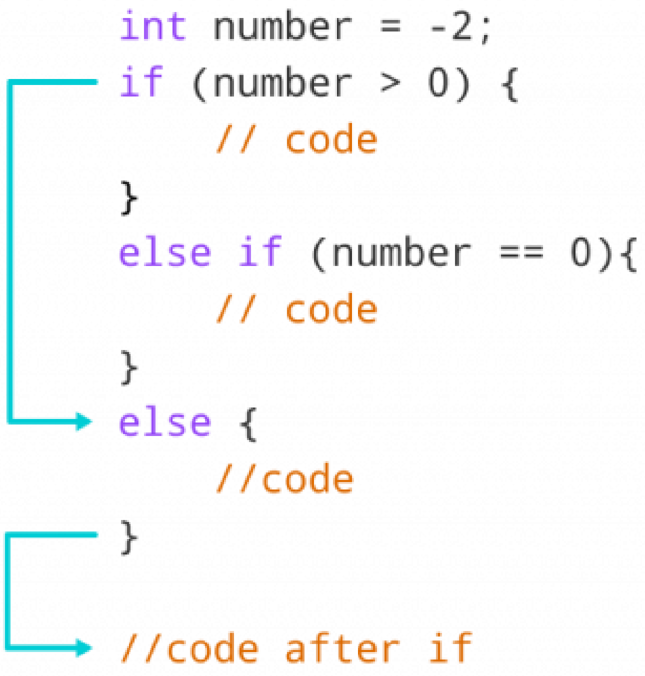
```
int number = 2;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

2nd Condition is true

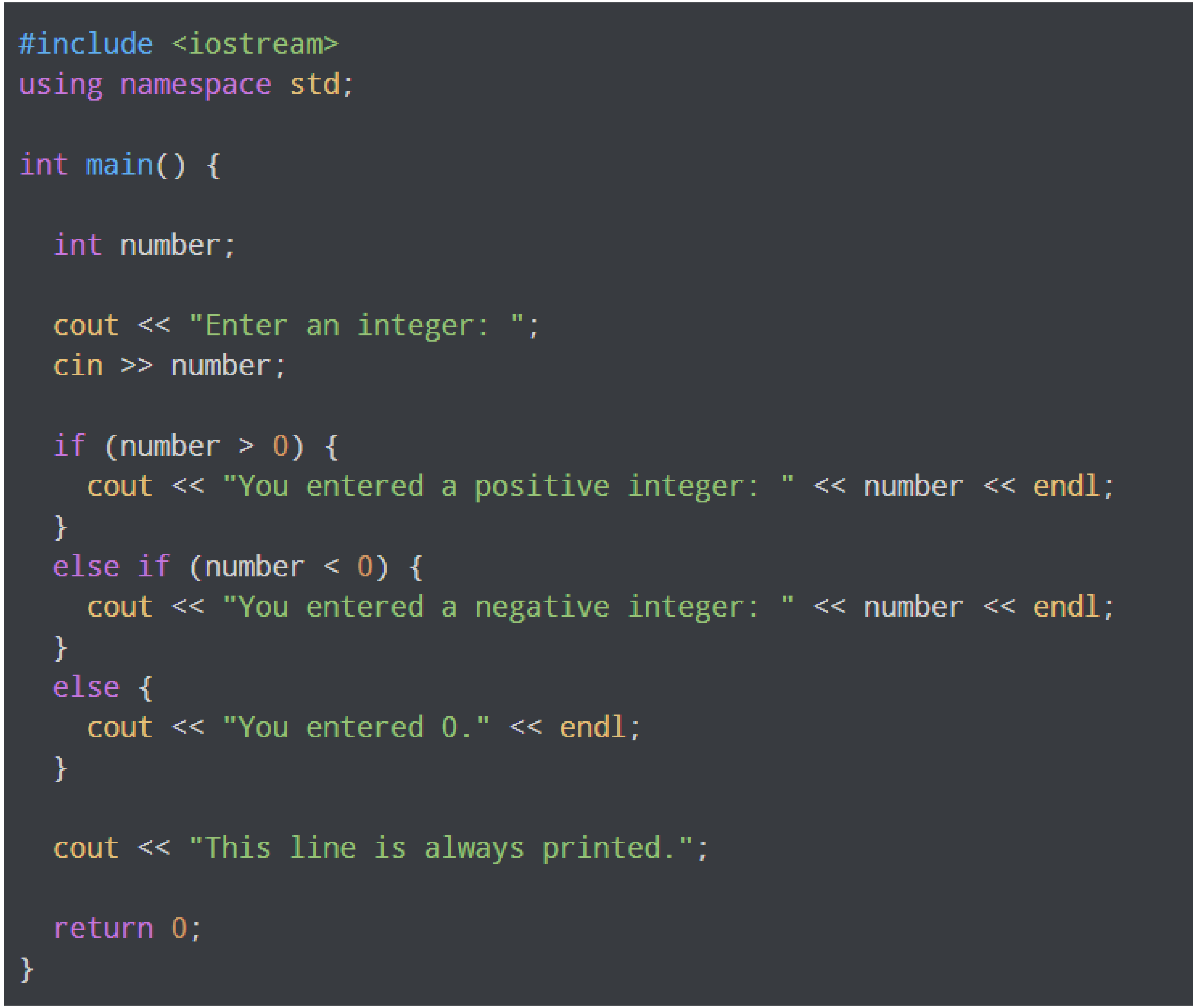


```
int number = 0;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

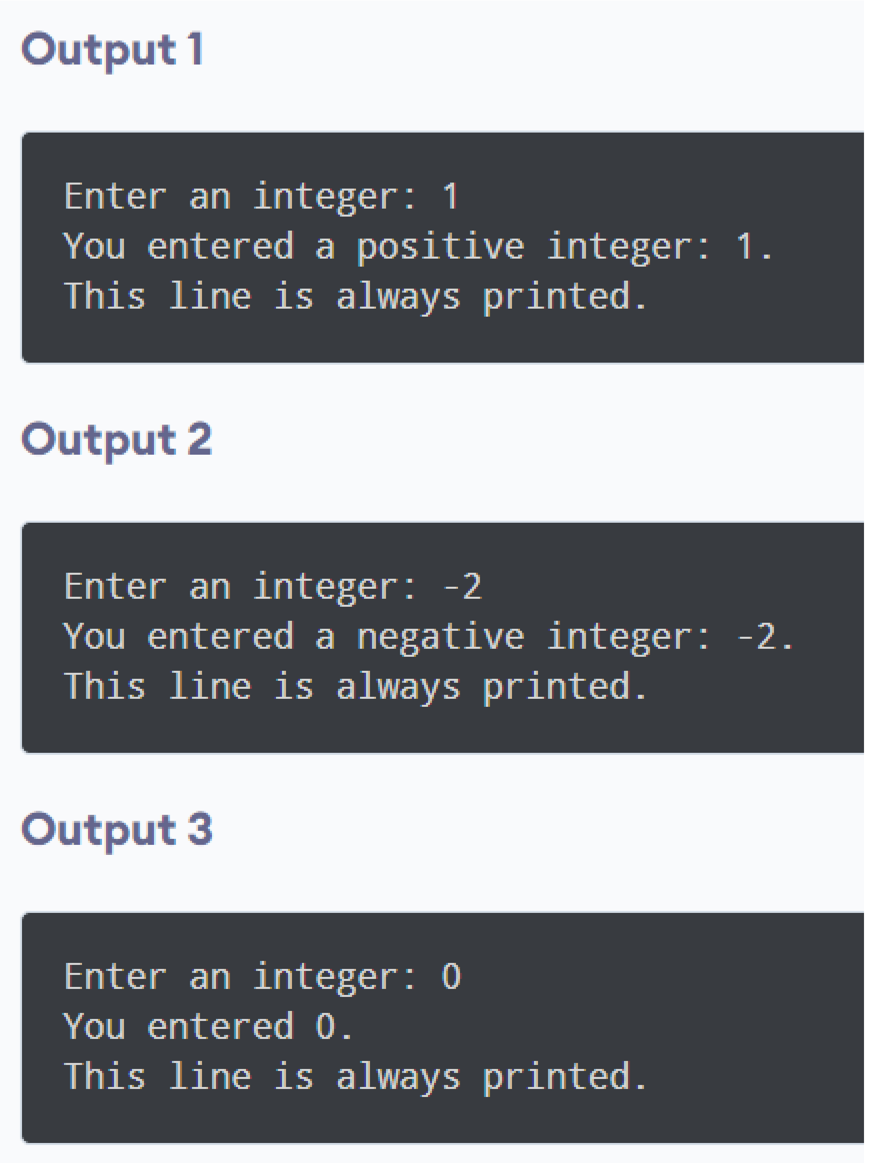
All Conditions are false



```
int number = -2;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```



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

int main() {

  int number;

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

  if (number > 0) {
    cout << "You entered a positive integer: " << number << endl;
  }
  else if (number < 0) {
    cout << "You entered a negative integer: " << number << endl;
  }
  else {
    cout << "You entered 0." << endl;
  }

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

  return 0;
}
```



Output 1

```
Enter an integer: 1
You entered a positive integer: 1.
This line is always printed.
```

Output 2

```
Enter an integer: -2
You entered a negative integer: -2.
This line is always printed.
```

Output 3

```
Enter an integer: 0
You entered 0.
This line is always printed.
```

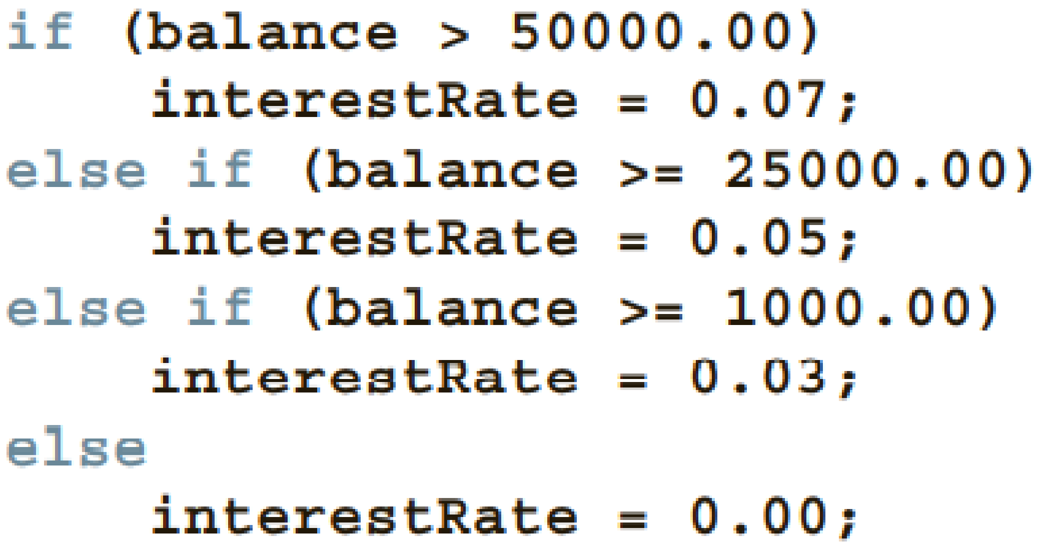
Practice

Suppose that you are in charge of tax. You have made several considarations:

- if the account balance is more than $50,000, the interest rate is 7%;
- if the balance is between $25,000 and $49,999.99, the interest rate is 5%;
- if the balance is between $1,000 and $24,999.99, the inter est rate is 3%;
- otherwise, the interest rate is 0%

Write a pseudocode for the problem

Solution

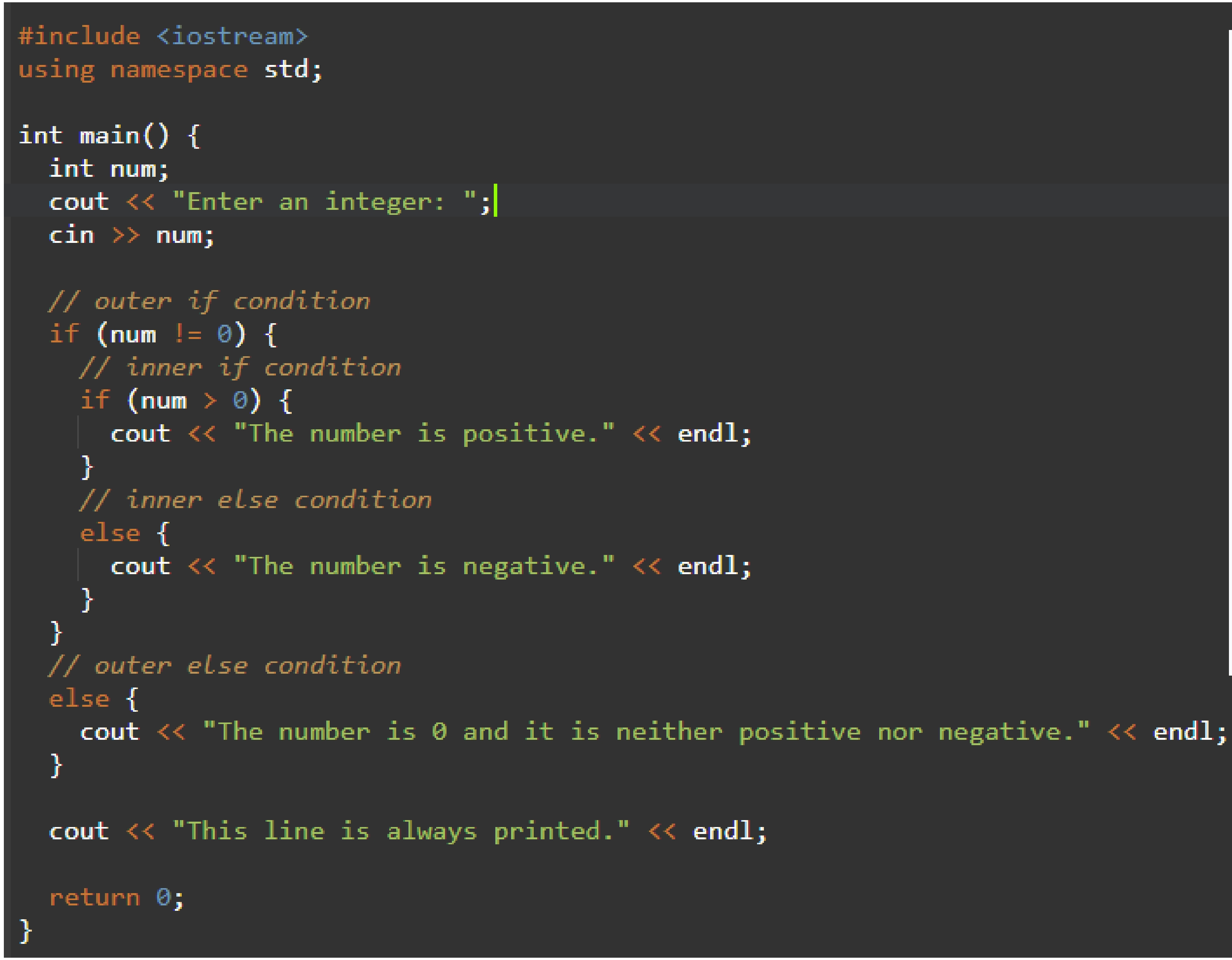


```
if (balance > 50000.00)
    interestRate = 0.07;
else if (balance >= 25000.00)
    interestRate = 0.05;
else if (balance >= 1000.00)
    interestRate = 0.03;
else
    interestRate = 0.00;
```

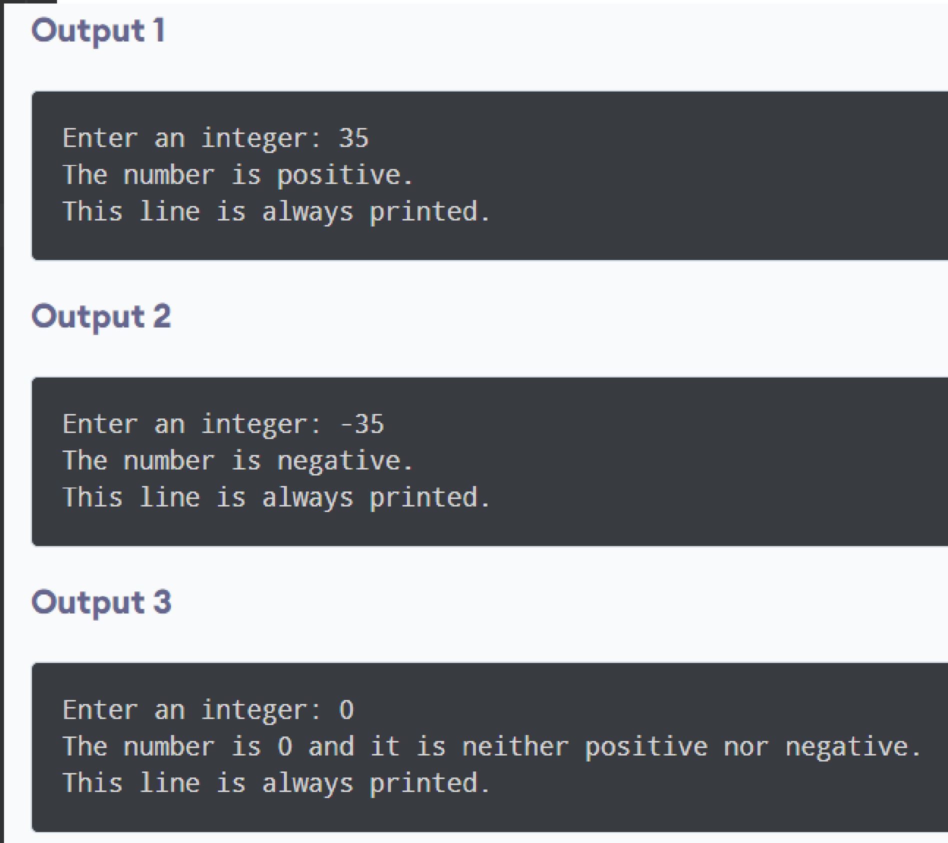
Nested if...else

- Sometimes, we need to use an if statement inside another if statement. This is known as nested if statement.
- Think of it as multiple layers of if statements. There is a first, outer if statement, and inside it is another, inner if statement. Its syntax is:

```
// outer if statement
if (condition1) {
  // statements
  // inner if statement
  if (condition2) {
    // statements
  }
}
```



```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;
  }

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

  return 0;
}
```



Output 1

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

Output 2

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

Output 3

```
Enter an integer: 0
The number is 0 and it is neither positive nor negative.
This line is always printed.
```

As you can see, nested if...else makes your logic complicated.

If possible, you should always try to avoid nested if...else.