SAXONY EGYPT UNIVERSITY
FOR APPLIED SCIENCE
AND TECHNOLOGY

ECT 121
Computer Programming I

Dr. Amina Elhawary

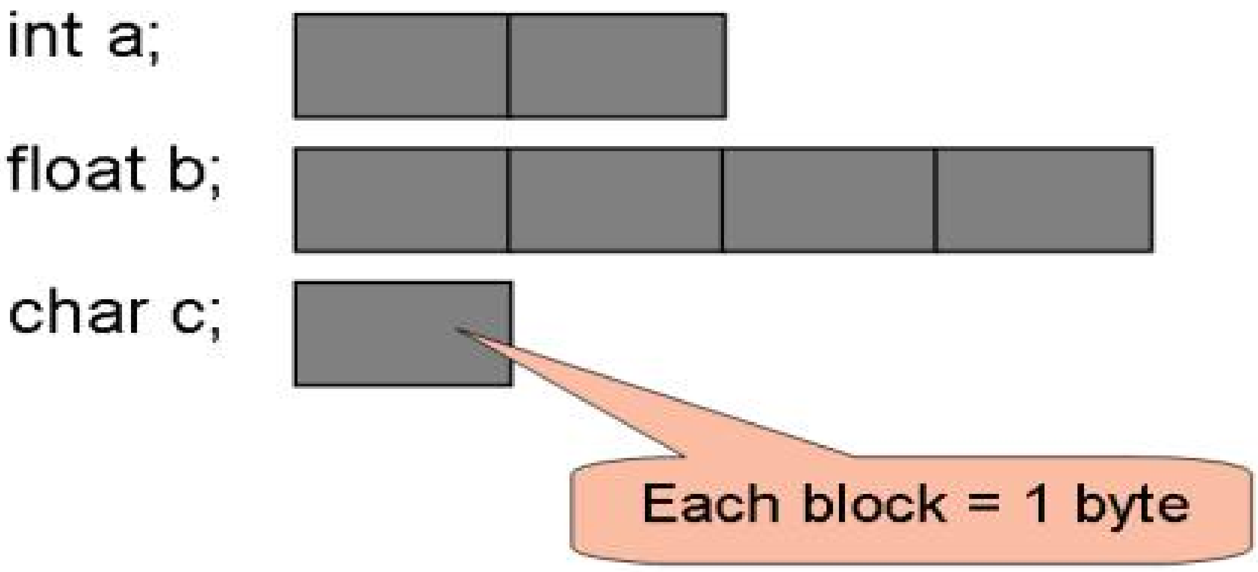
SAXONY EGYPT UNIVERSITY
FOR APPLIED SCIENCE AND TECHNOLOGY

Lecture Two

Data types, Operators and simple functions

What is a Variable ?

- *A **variable** is a location in memory which we can refer to by an identifier, and in which a data value that can be changed is stored.*



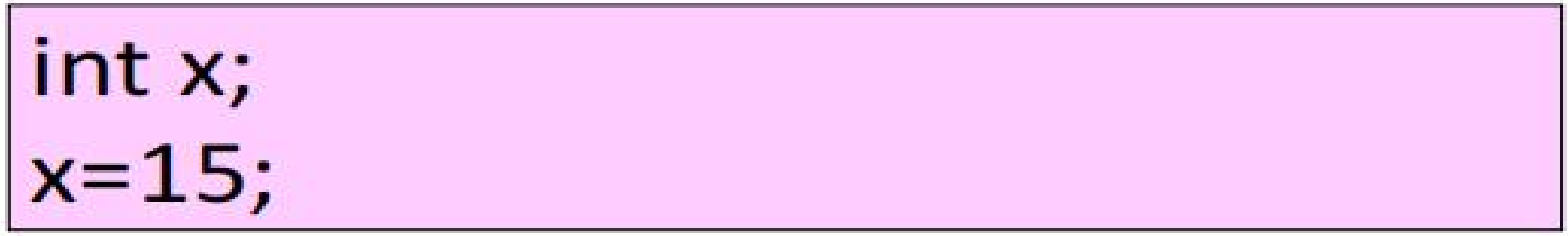
int a;

float b;

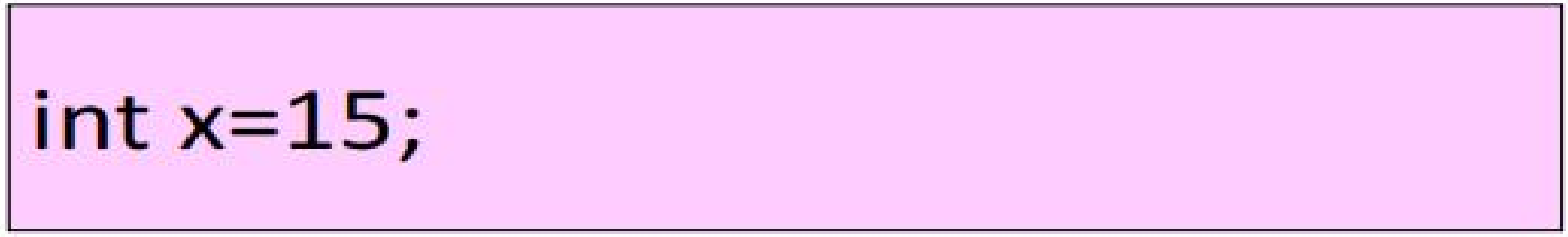
char c;

Each block = 1 byte

Assign a Value to a Variable

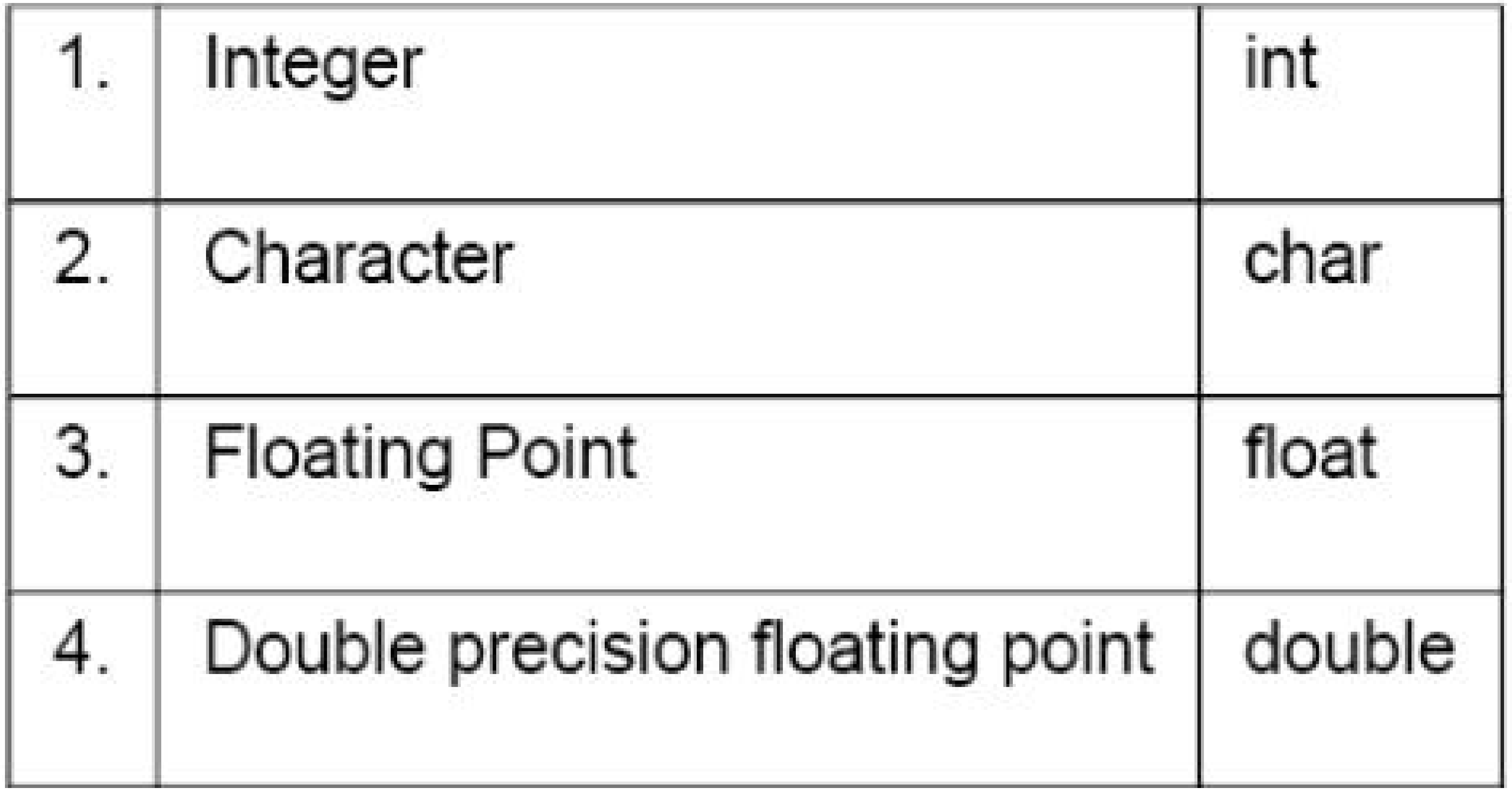


```
int x;
x=15;
```



```
int x=15;
```

Primary Data Types



1.	Integer	int
2.	Character	char
3.	Floating Point	float
4.	Double precision floating point	double

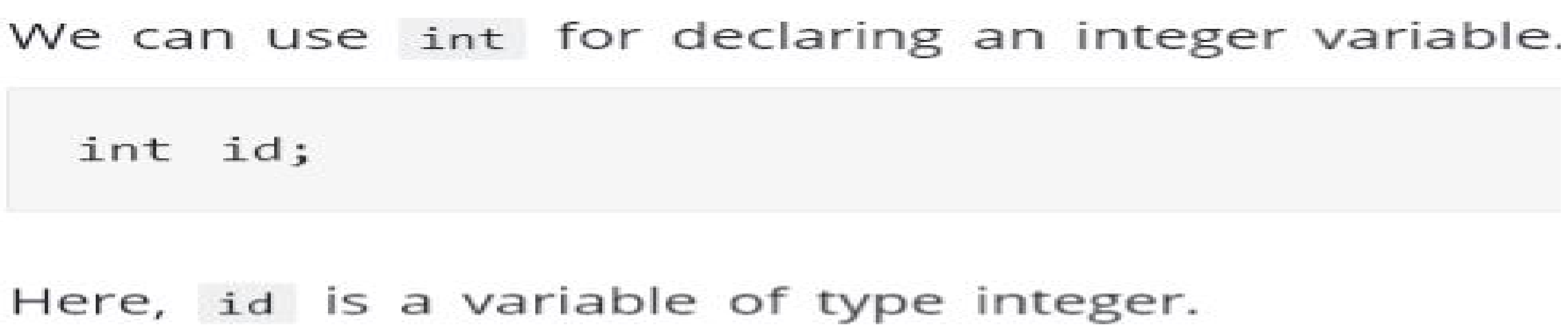
Primary Data Types (Cont.)

1. Integer

Integers are whole numbers that can be zero, positive, or negative but do not have decimal values.

- **int sample values**

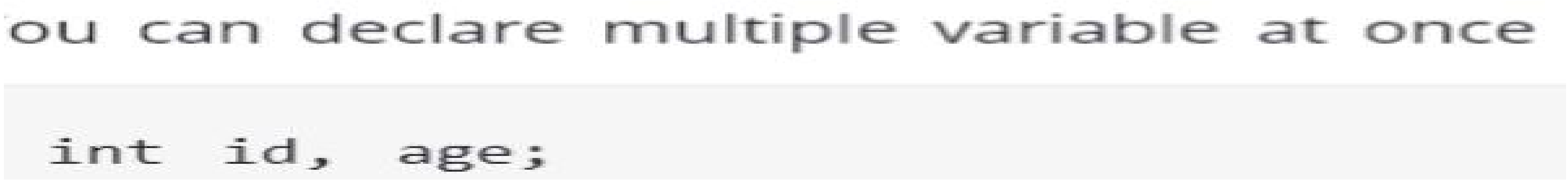
4578 -4578 0



We can use `int` for declaring an integer variable.

```
int id;
```

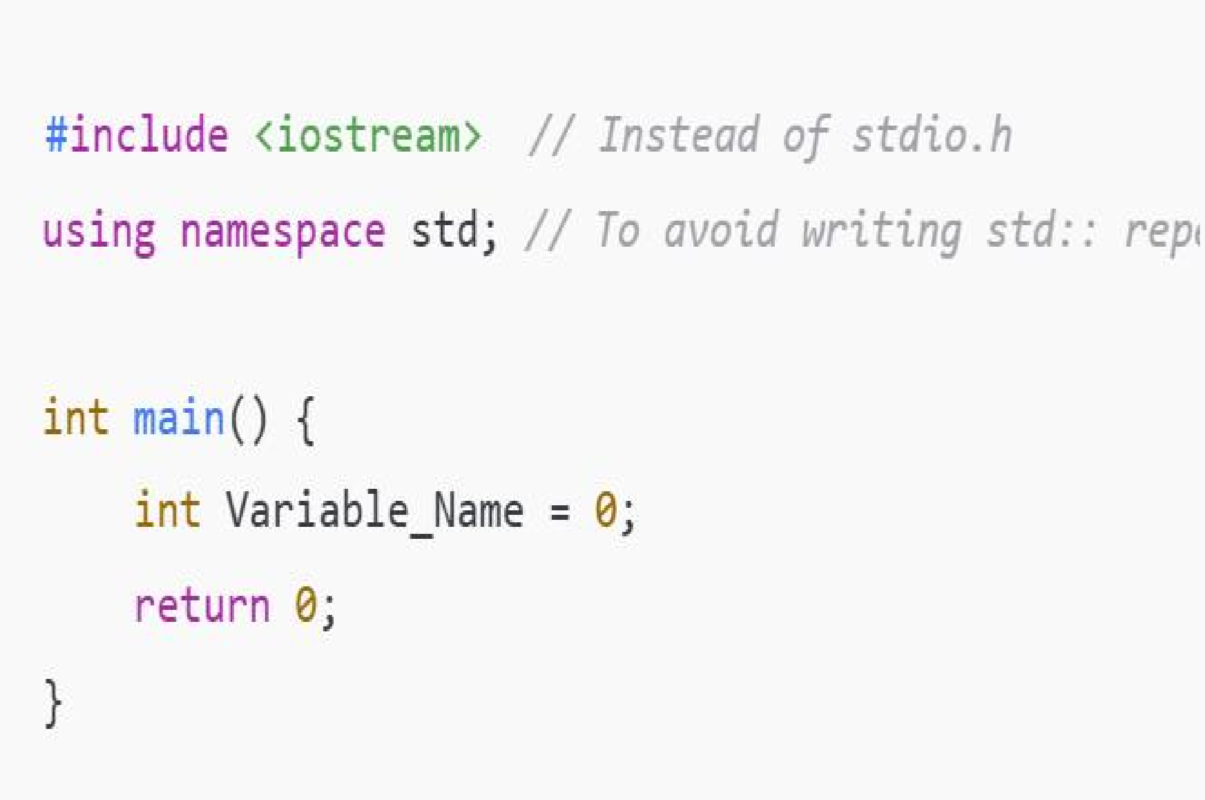
Here, `id` is a variable of type integer.



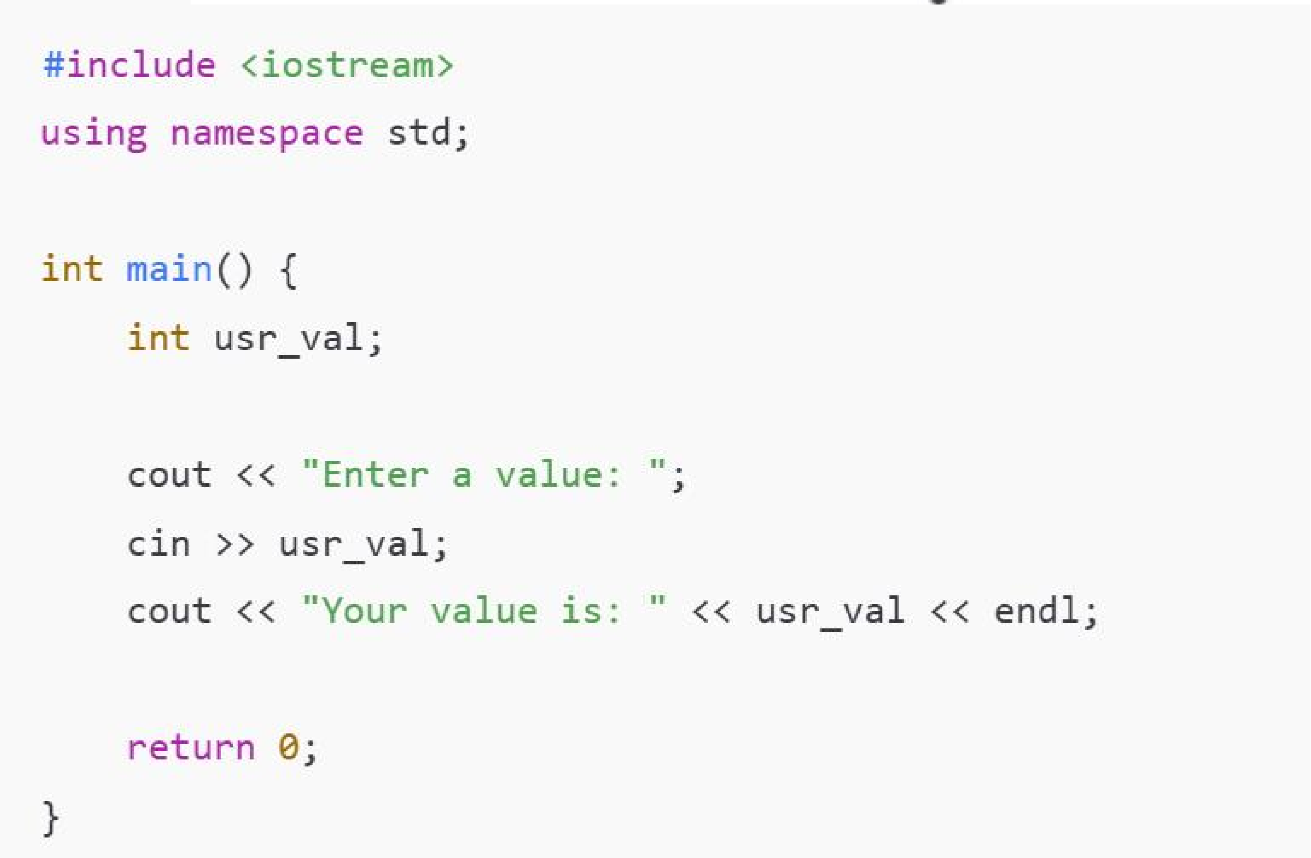
ou can declare multiple variable at once

```
int id, age;
```

Primary Data Types (Cont.)

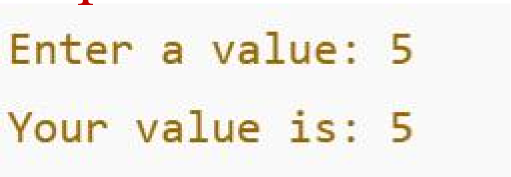


```
#include <iostream>  // Instead of stdio.h
using namespace std; // To avoid writing std:: repe

int main() {
    int Variable_Name = 0;
    return 0;
}
```



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

int main() {
    int usr_val;

    cout << "Enter a value: ";
    cin >> usr_val;
    cout << "Your value is: " << usr_val << endl;

    return 0;
}
```

Output

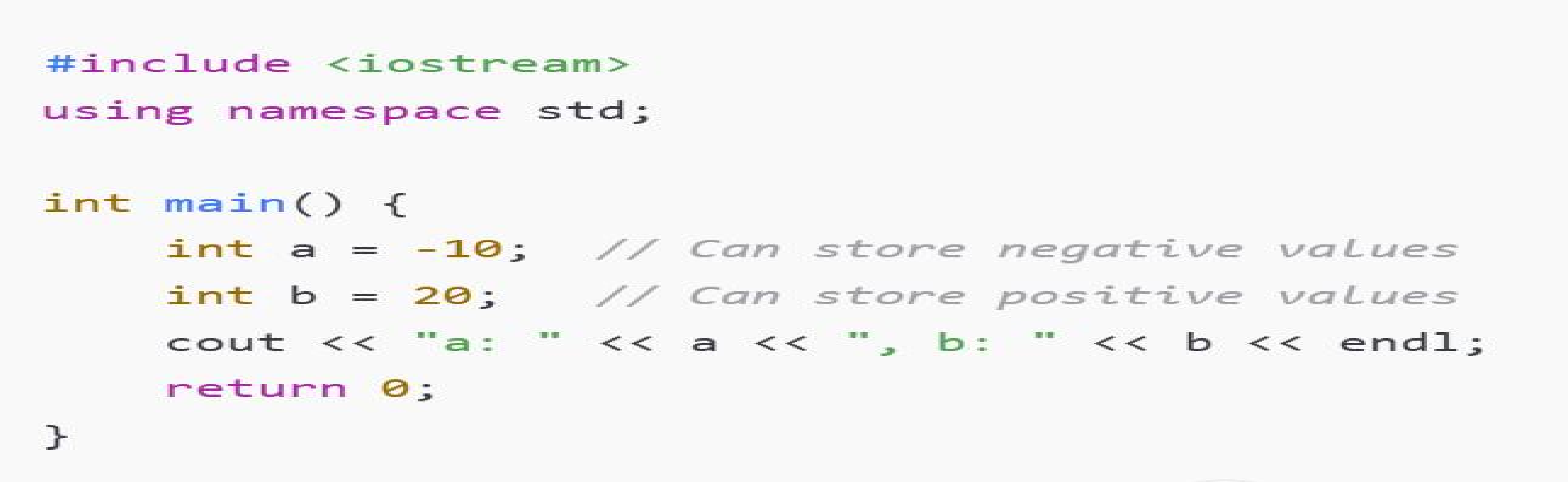


```
Enter a value: 5
Your value is: 5
```

Primary Data Types (Cont.)

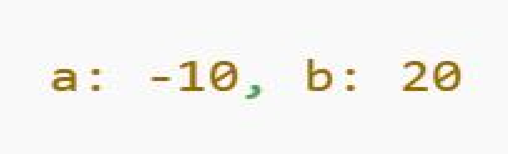
2. Signed int

- Explicitly tells the compiler that the integer is signed and It can store both positive and negative values.



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

int main() {
    int a = -10;  // Can store negative values
    int b = 20;   // Can store positive values
    cout << "a: " << a << ", b: " << b << endl;
    return 0;
}
```

Output



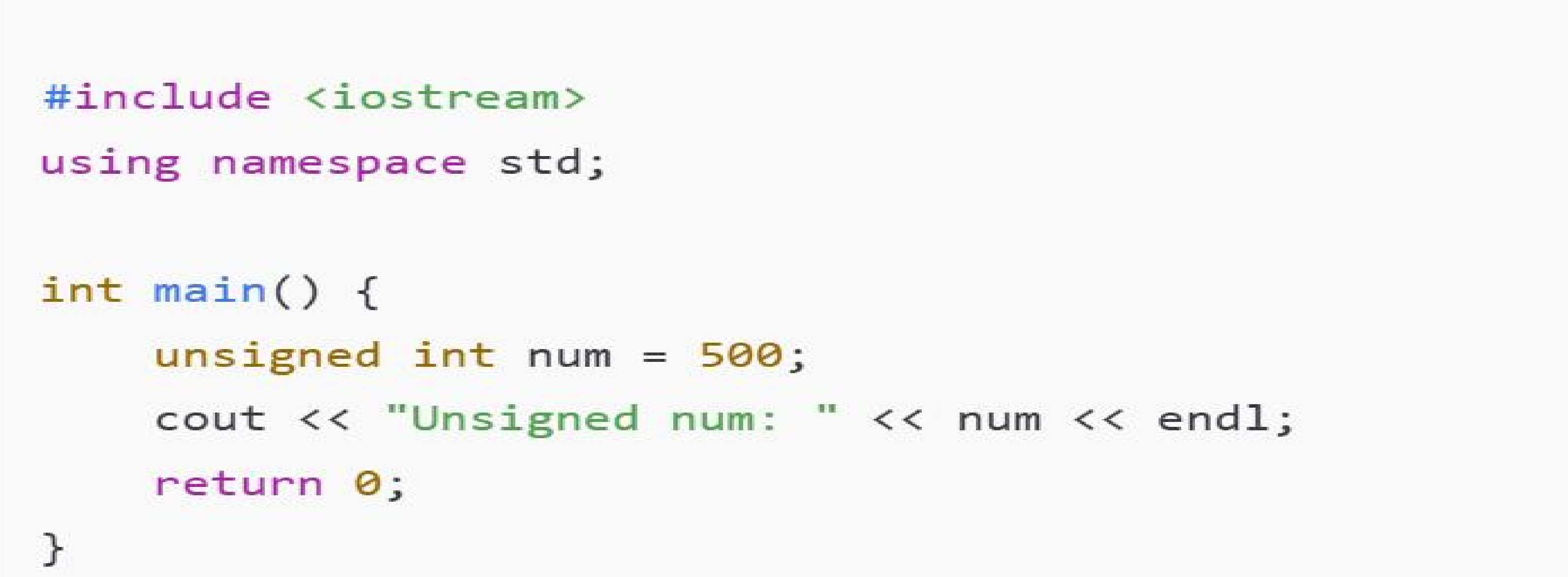
```
a: -10, b: 20
```

Primary Data Types (Cont.)

SAXONY EGYPT UNIVERSITY
FOR APPLIED SCIENCE AND TECHNOLOGY

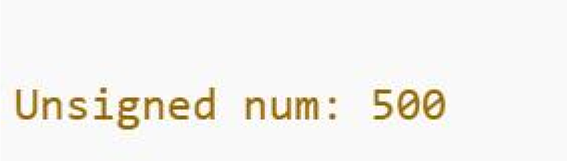
2. Unsigned int

- Stores only non-negative values (no negative numbers).
- Expands the range of positive numbers by using all bits for positive values.



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

int main() {
    unsigned int num = 500;
    cout << "Unsigned num: " << num << endl;
    return 0;
}
```

Output



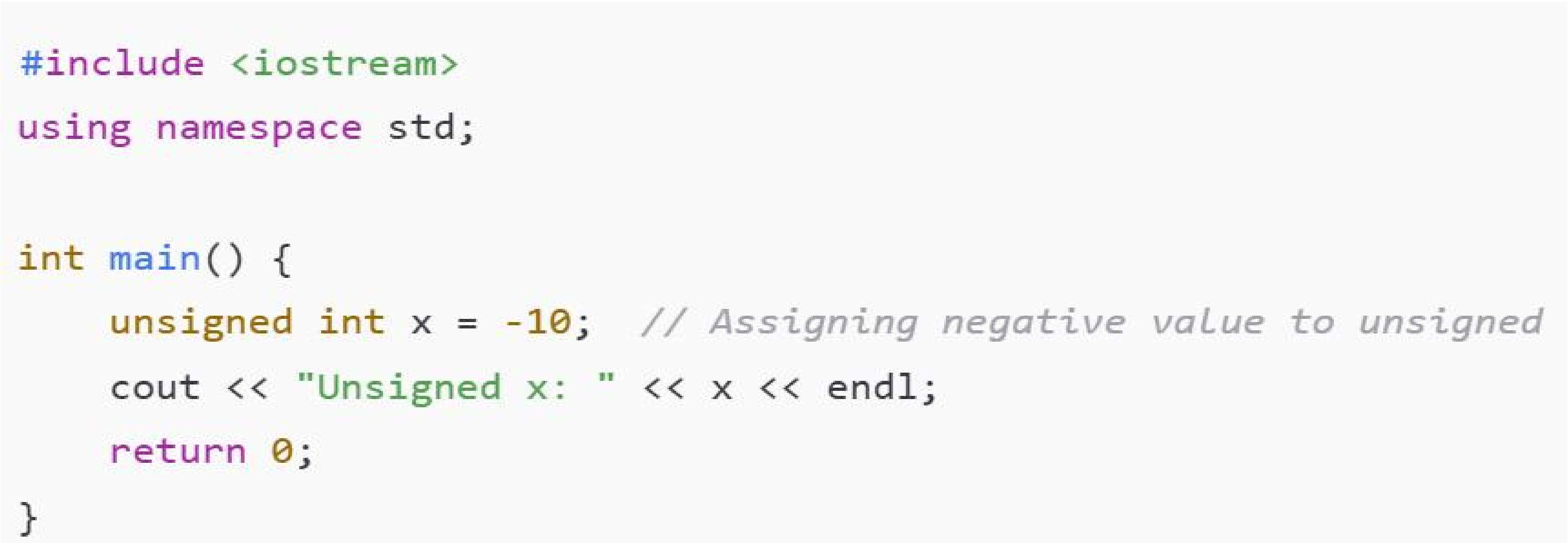
```
Unsigned num: 500
```

Primary Data Types (Cont.)

SAXONY EGYPT UNIVERSITY
FOR APPLIED SCIENCE AND TECHNOLOGY

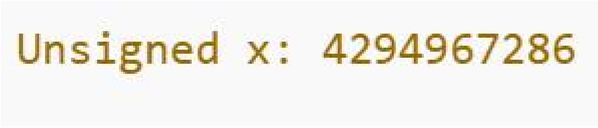
2. Unsigned int

- If you assign a negative number to an unsigned int, it wraps around due to binary representation.

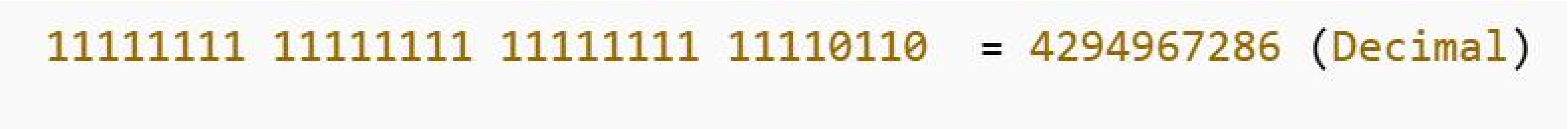


```
#include <iostream>
using namespace std;

int main() {
    unsigned int x = -10;  // Assigning negative value to unsigned
    cout << "Unsigned x: " << x << endl;
    return 0;
}
```

Output



```
Unsigned x: 4294967286
```



```
11111111 11111111 11111111 11110110  = 4294967286 (Decimal)
```

Primary Data Types (Cont.)

3. Float & Double

- In C++, float and double are data types used to store decimal numbers (floating-point numbers). The main difference between them is precision and size.

float

•Uses 4 bytes (32 bits)

•Can store ~7 decimal digits accurately

•Less precise but faster

double

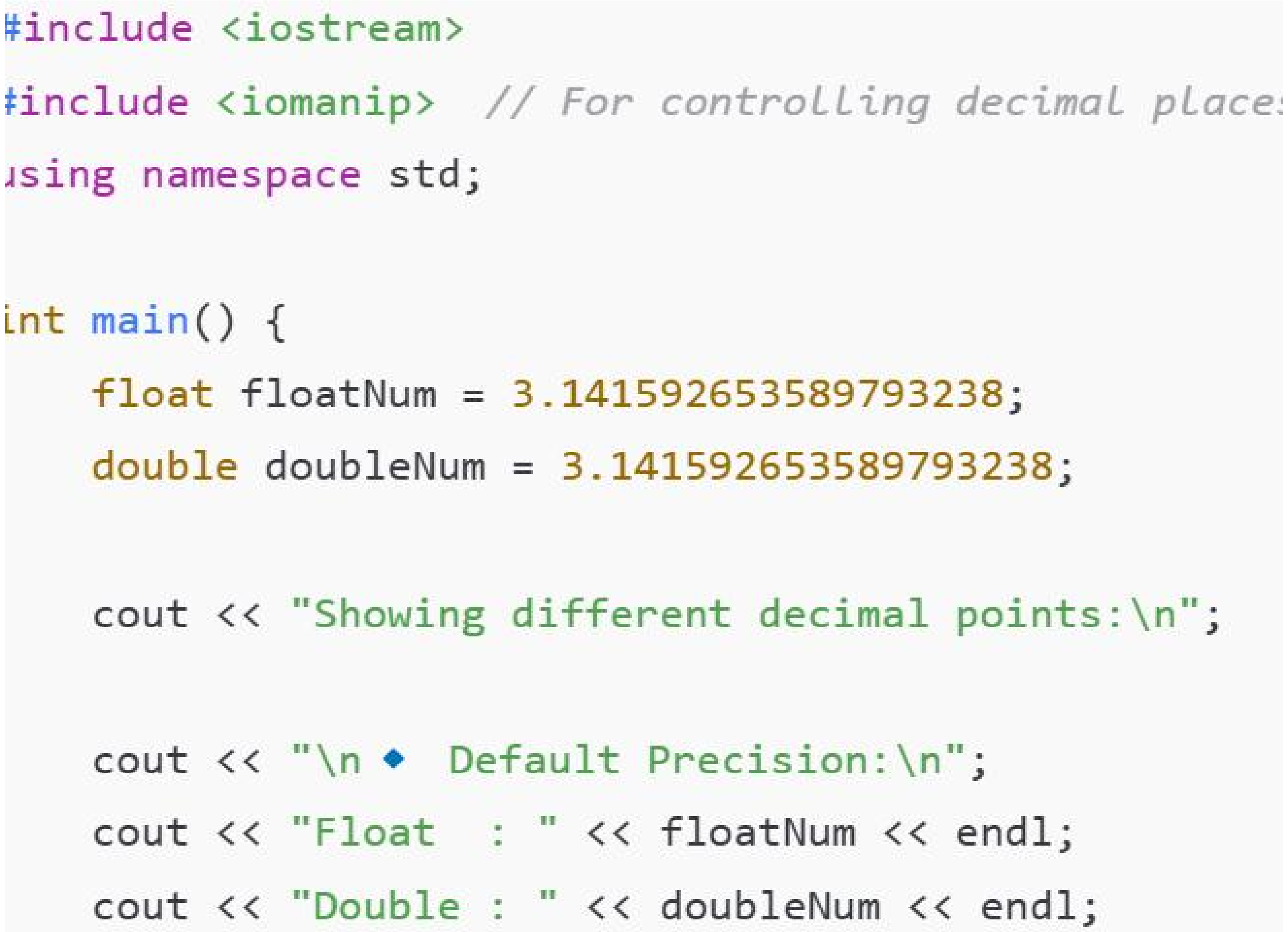
•Uses 8 bytes (64 bits)

•Can store ~15 decimal digits accurately

•More precise but slightly slower

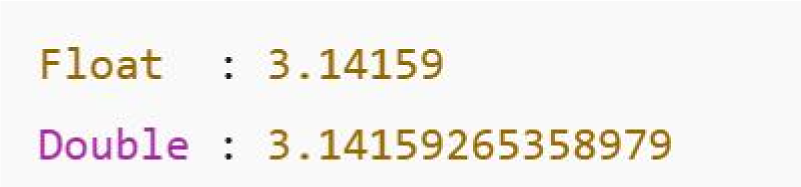
Primary Data Types (Cont.)

3. Float & Double



```cpp
#include <iostream>
#include <iomanip>  // For controlling decimal places
using namespace std;

int main() {
    float floatNum = 3.141592653589793238;
    double doubleNum = 3.141592653589793238;

    cout << "Showing different decimal points:\n";

    cout << "\n ◆  Default Precision:\n";
    cout << "Float  : " << floatNum << endl;
    cout << "Double : " << doubleNum << endl;
```

Output



```
Float  : 3.14159
Double : 3.14159265358979
```

Primary Data Types (Cont.)

4. Character

The char data type is used for storing single characters.

Key Points:

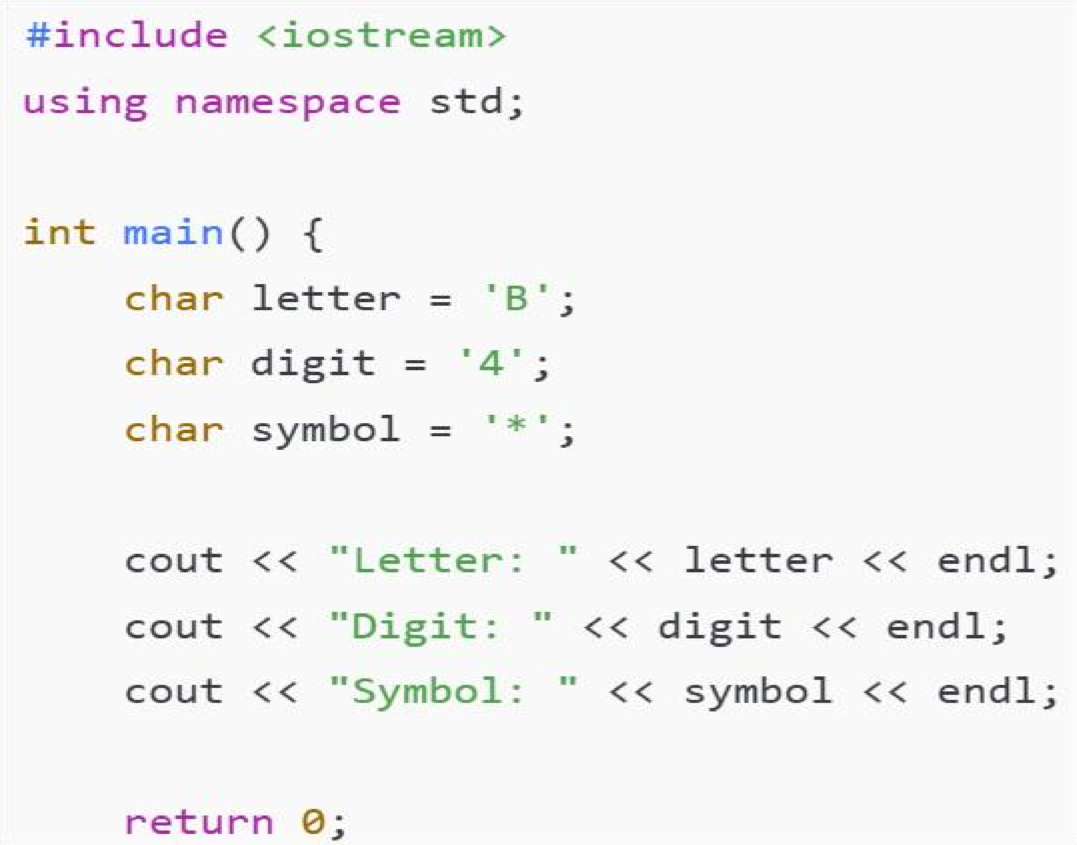
- The size of a character variable is 1 byte.

- char variables store a single character enclosed in single quotes ' '.

Character	Example
'B'	Letter
'd'	Lowercase Letter
'4'	Digit
'?'	Symbol
'*'	Special Character

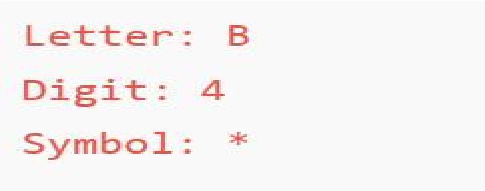
Primary Data Types (Cont.)

4. Character

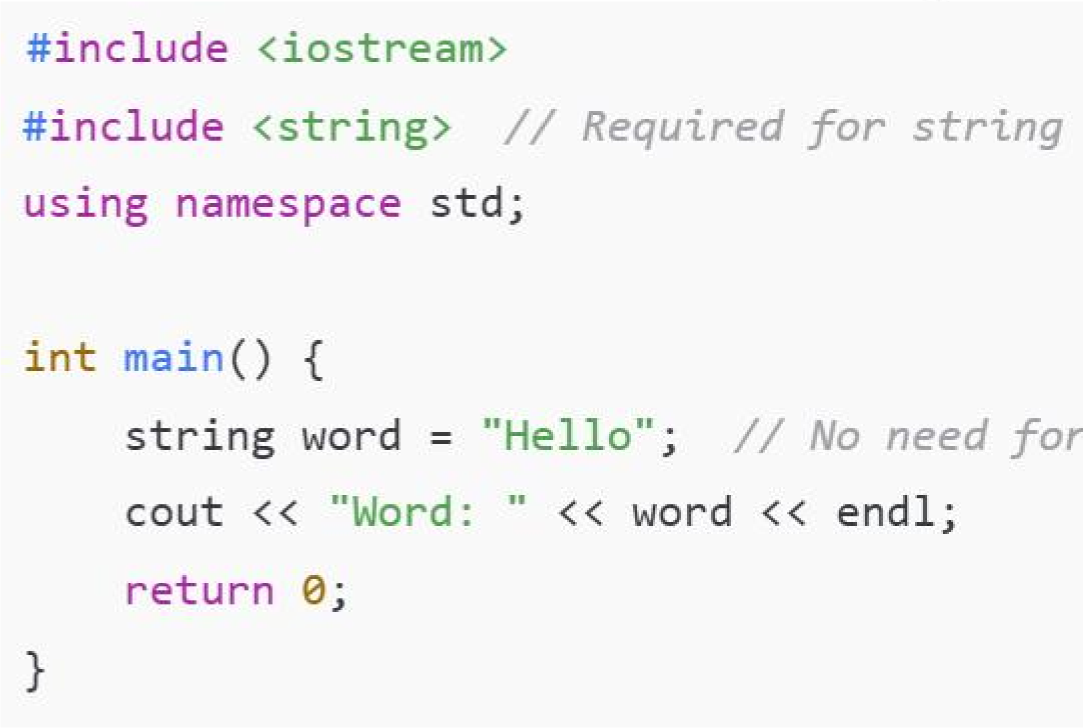


```
#include <iostream>
using namespace std;

int main() {
    char letter = 'B';
    char digit = '4';
    char symbol = '*';

    cout << "Letter: " << letter << endl;
    cout << "Digit: " << digit << endl;
    cout << "Symbol: " << symbol << endl;

    return 0;
```

Output

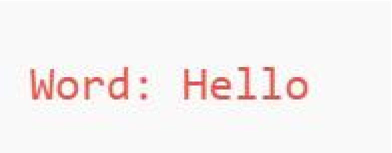


```
Letter: B
Digit: 4
Symbol: *
```



```
#include <iostream>
#include <string>  // Required for string
using namespace std;

int main() {
    string word = "Hello";  // No need for
    cout << "Word: " << word << endl;
    return 0;
}
```

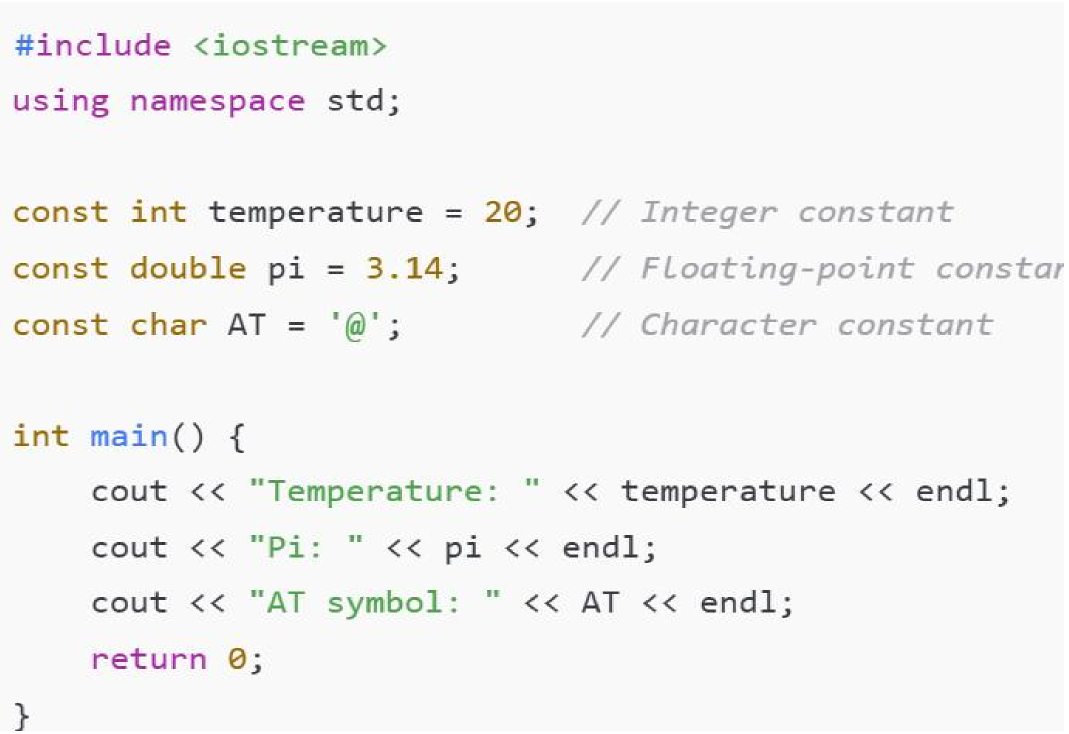
Output



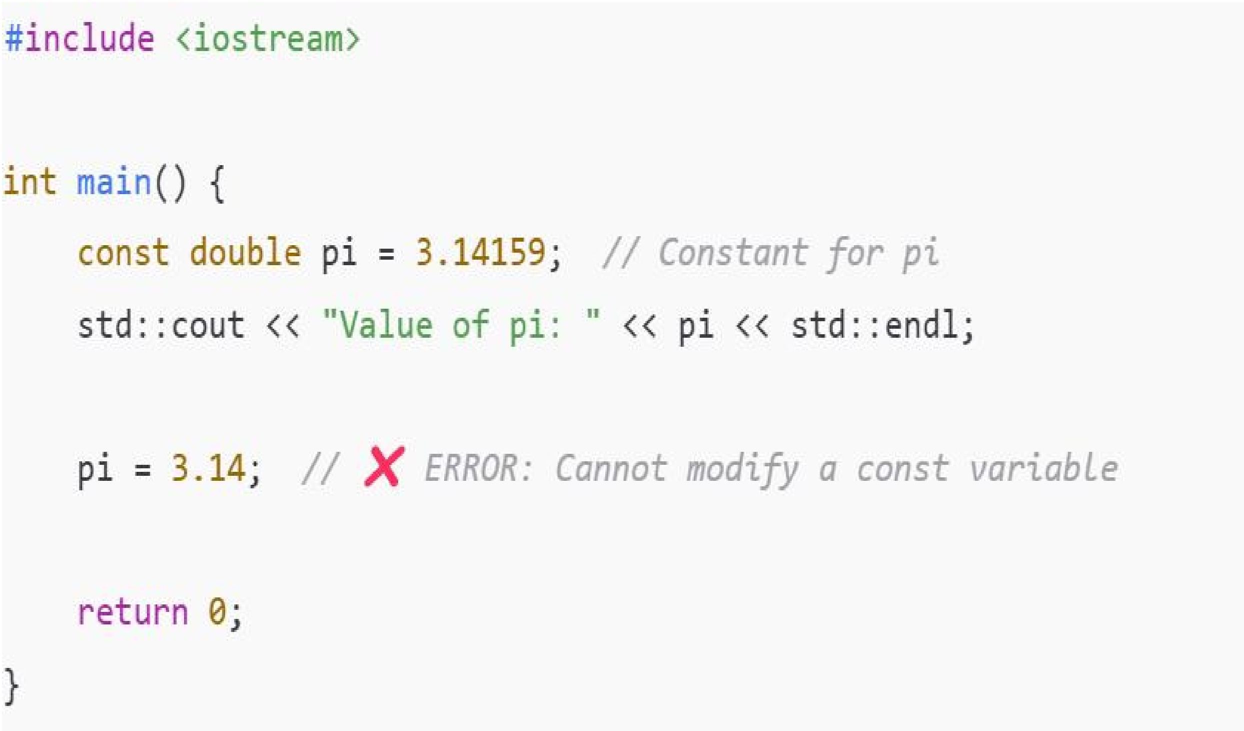
```
Word: Hello
```

What is a Constant ?

- A **named constant** is a location in memory that we can refer to by a name, and in which a **data value that cannot be changed** is stored.

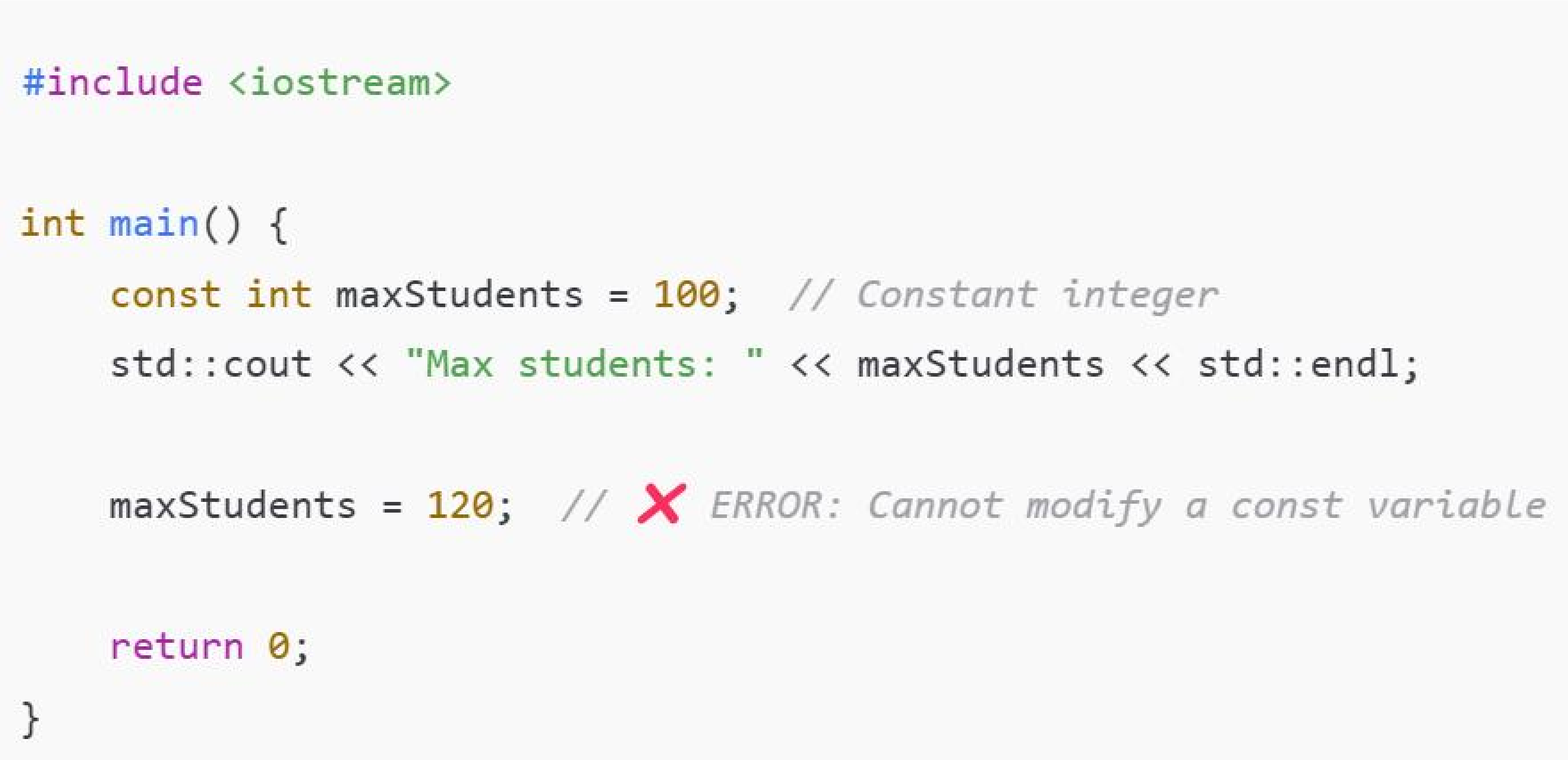


```cpp
#include <iostream>
using namespace std;

const int temperature = 20;   // Integer constant
const double pi = 3.14;       // Floating-point constar
const char AT = '@';          // Character constant

int main() {
    cout << "Temperature: " << temperature << endl;
    cout << "Pi: " << pi << endl;
    cout << "AT symbol: " << AT << endl;
    return 0;
}
```



```cpp
#include <iostream>

int main() {
    const double pi = 3.14159;  // Constant for pi
    std::cout << "Value of pi: " << pi << std::endl;

    pi = 3.14;  // ❌ ERROR: Cannot modify a const variable

    return 0;
}
```

Is it correct for Constant ?

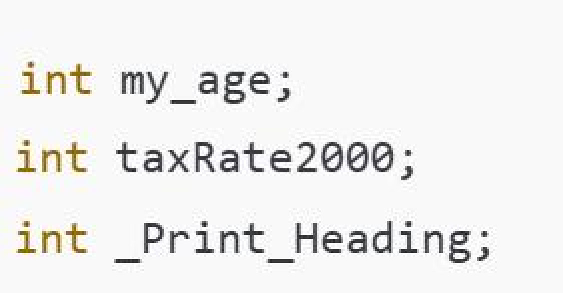


```
#include <iostream>

int main() {
    const int maxStudents = 100;  // Constant integer
    std::cout << "Max students: " << maxStudents << std::endl;

    maxStudents = 120;  // ❌ ERROR: Cannot modify a const variable

    return 0;
}
```

What is an Identifier?

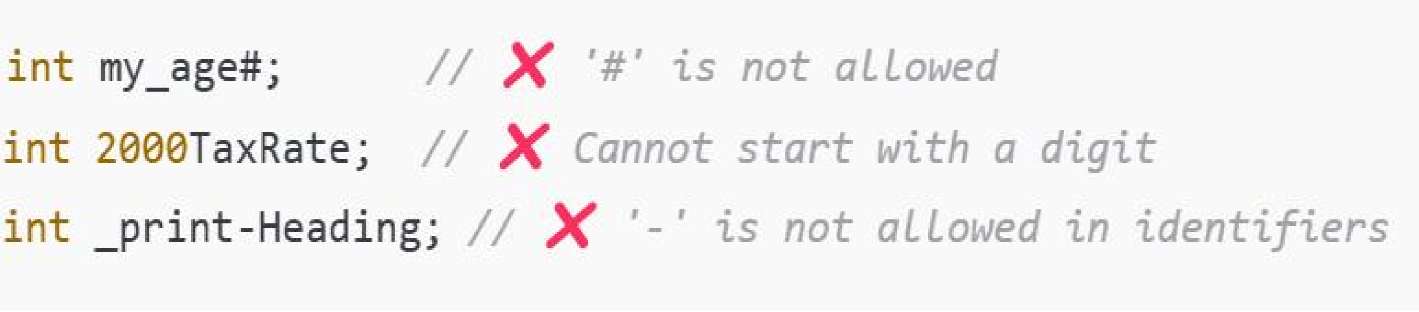
- An ***Identifier*** is the name used for a variable, a constant, or for a function, in a C++ program.
- C++ is a case-sensitive language (AB is not Ab).
- using meaningful identifiers is a good programming practice.
- an identifier must start with a letter or underscore, and be followed by letters (A-Z, a-z), digits (0-9), or underscores.

VALID



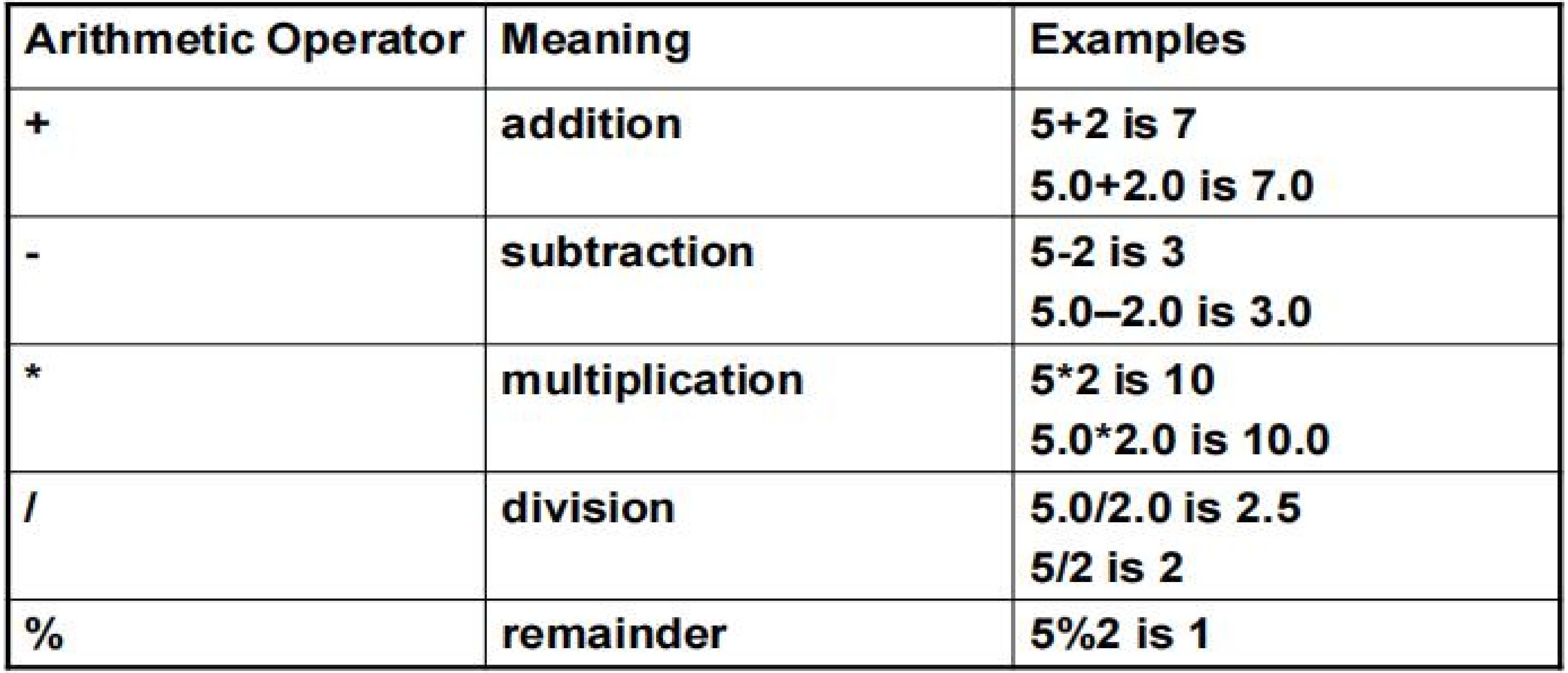
```
int my_age;
int taxRate2000;
int _Print_Heading;
```

NOT VALID



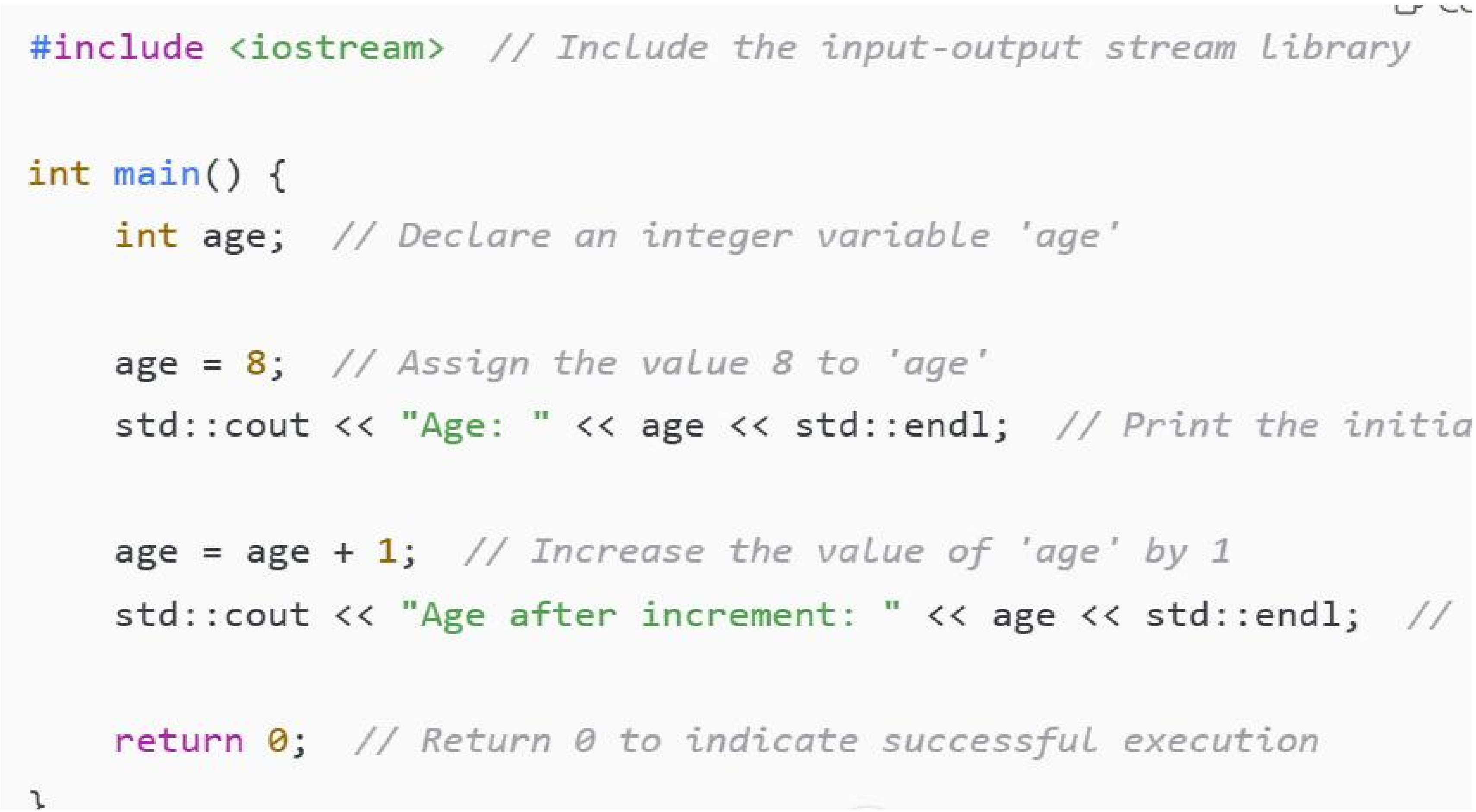
```
int my_age#;        // ❌ '#' is not allowed
int 2000TaxRate;    // ❌ Cannot start with a digit
int _print-Heading; // ❌ '-' is not allowed in identifiers
```

Arithemtic Operators



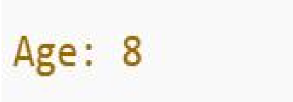
Arithmetic Operator	Meaning	Examples
+	addition	5+2 is 7 5.0+2.0 is 7.0
-	subtraction	5-2 is 3 5.0–2.0 is 3.0
*	multiplication	5*2 is 10 5.0*2.0 is 10.0
/	division	5.0/2.0 is 2.5 5/2 is 2
%	remainder	5%2 is 1

More C++ Operators

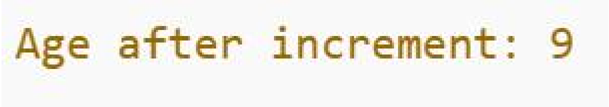


```
#include <iostream>  // Include the input-output stream library

int main() {
    int age;  // Declare an integer variable 'age'

    age = 8;  // Assign the value 8 to 'age'
    std::cout << "Age: " << age << std::endl;  // Print the initia

    age = age + 1;  // Increase the value of 'age' by 1
    std::cout << "Age after increment: " << age << std::endl;  //

    return 0;  // Return 0 to indicate successful execution
}
```

Output

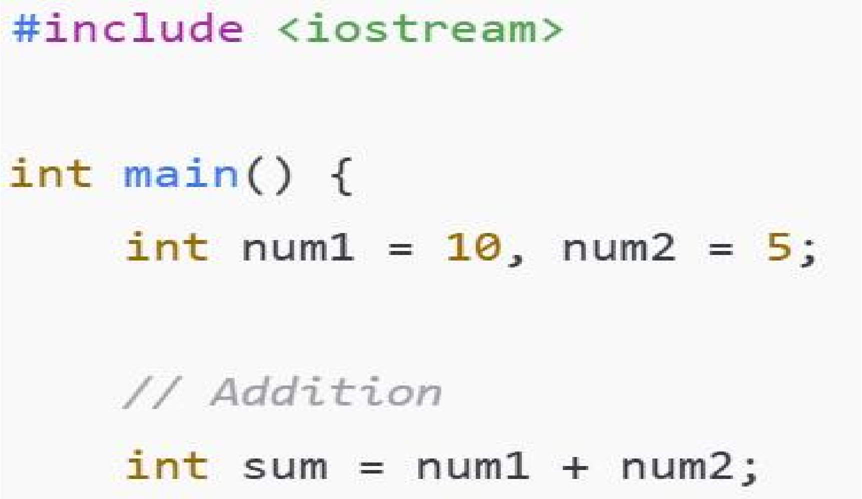


```
Age: 8
```

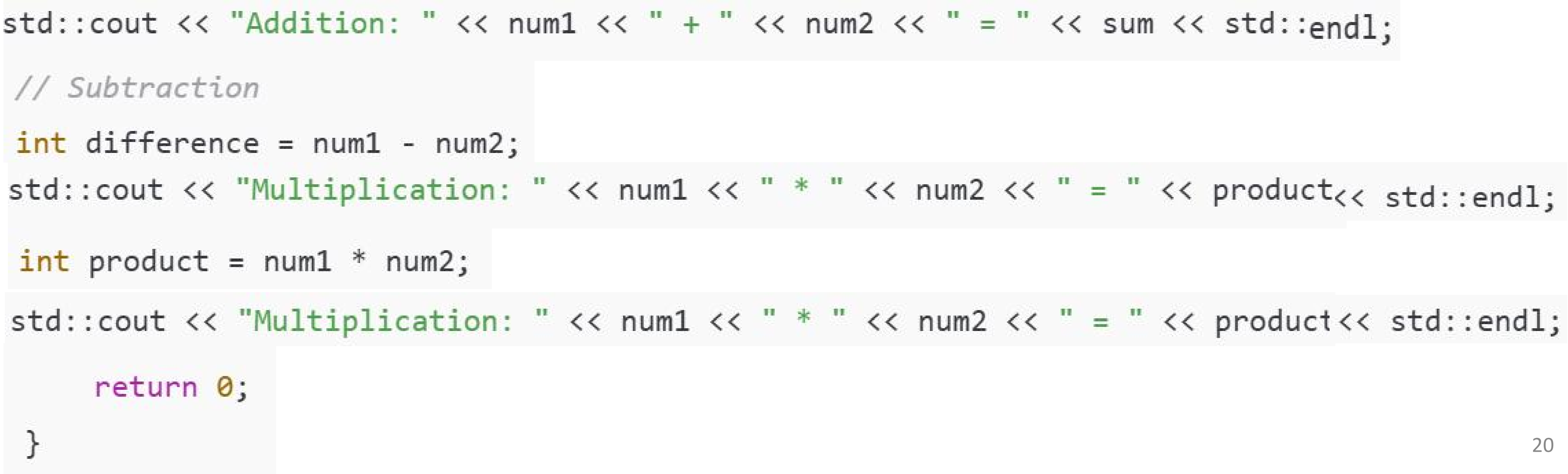


```
Age after increment: 9
```

More C++ Operators

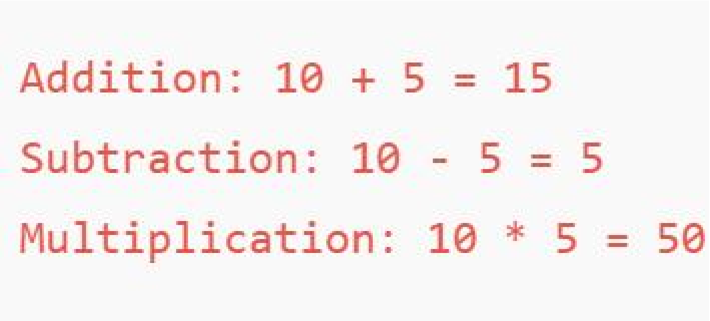


```
#include <iostream>

int main() {
    int num1 = 10, num2 = 5;

    // Addition
    int sum = num1 + num2;
```



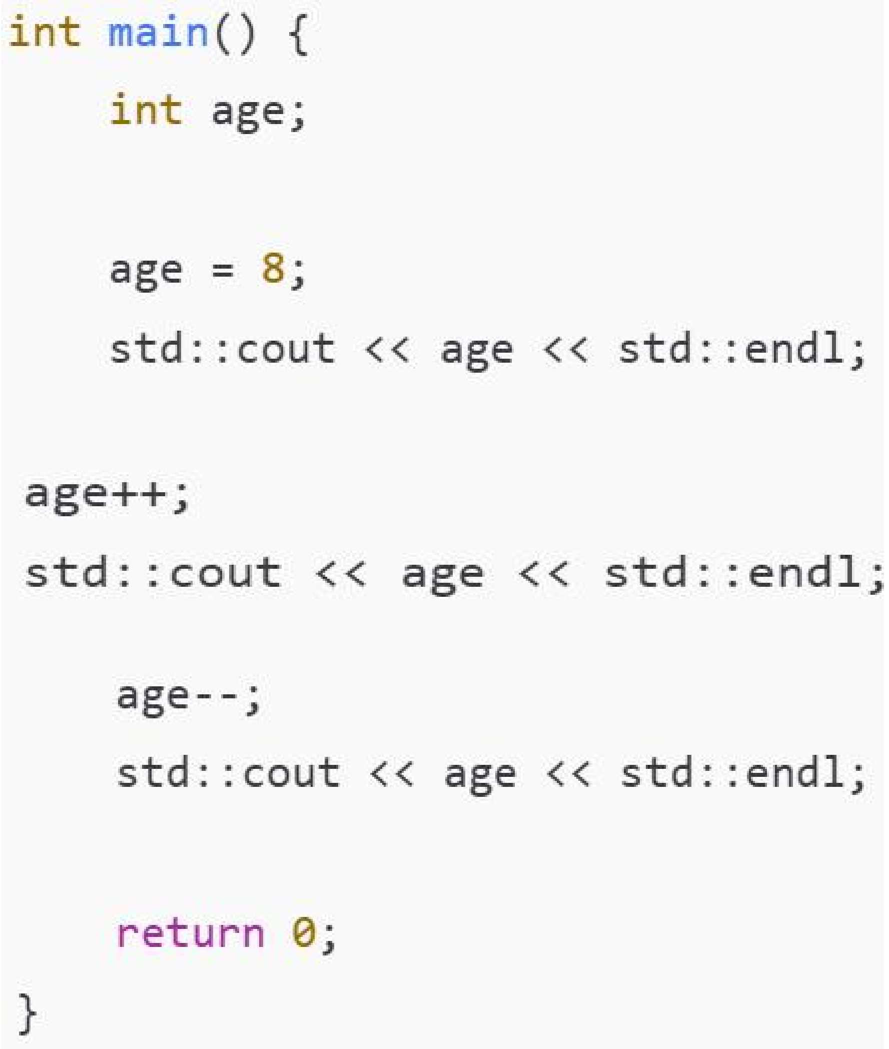
```
std::cout << "Addition: " << num1 << " + " << num2 << " = " << sum << std::endl;

// Subtraction
int difference = num1 - num2;
std::cout << "Multiplication: " << num1 << " * " << num2 << " = " << product<< std::endl;

int product = num1 * num2;
std::cout << "Multiplication: " << num1 << " * " << num2 << " = " << product<< std::endl;

    return 0;
}
```

Output



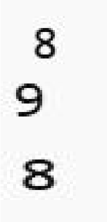
```
Addition: 10 + 5 = 15

Subtraction: 10 - 5 = 5

Multiplication: 10 * 5 = 50
```

PREFIX Form Increment/decrement Operator



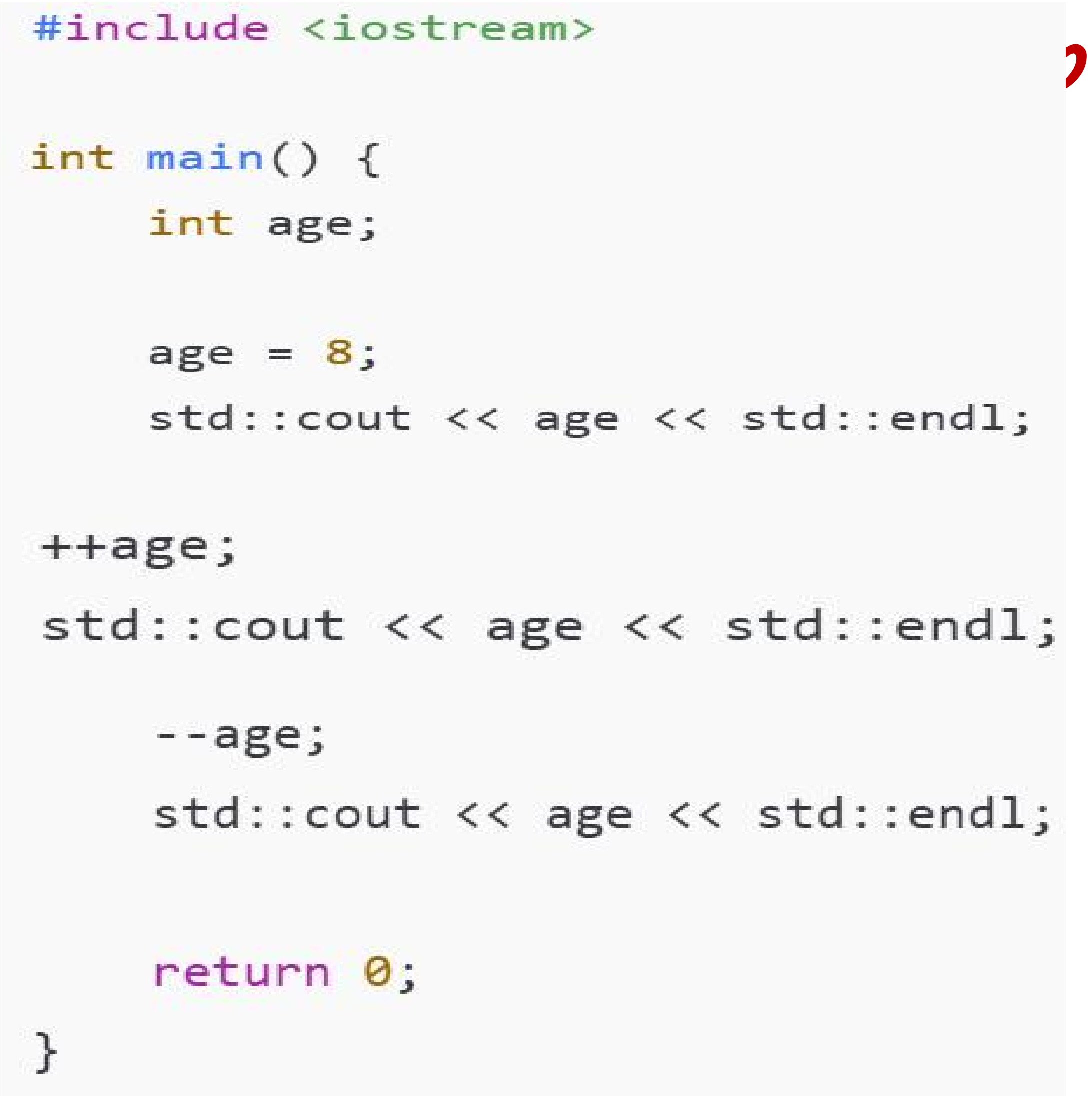
```
int main() {
    int age;


    age = 8;
    std::cout << age << std::endl;

 age++;
 std::cout << age << std::endl;

    age--;
    std::cout << age << std::endl;


    return 0;
}
```

Output



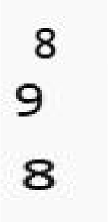
```
 8
9
8
```

POSTFIX Form Increment/decrement erator



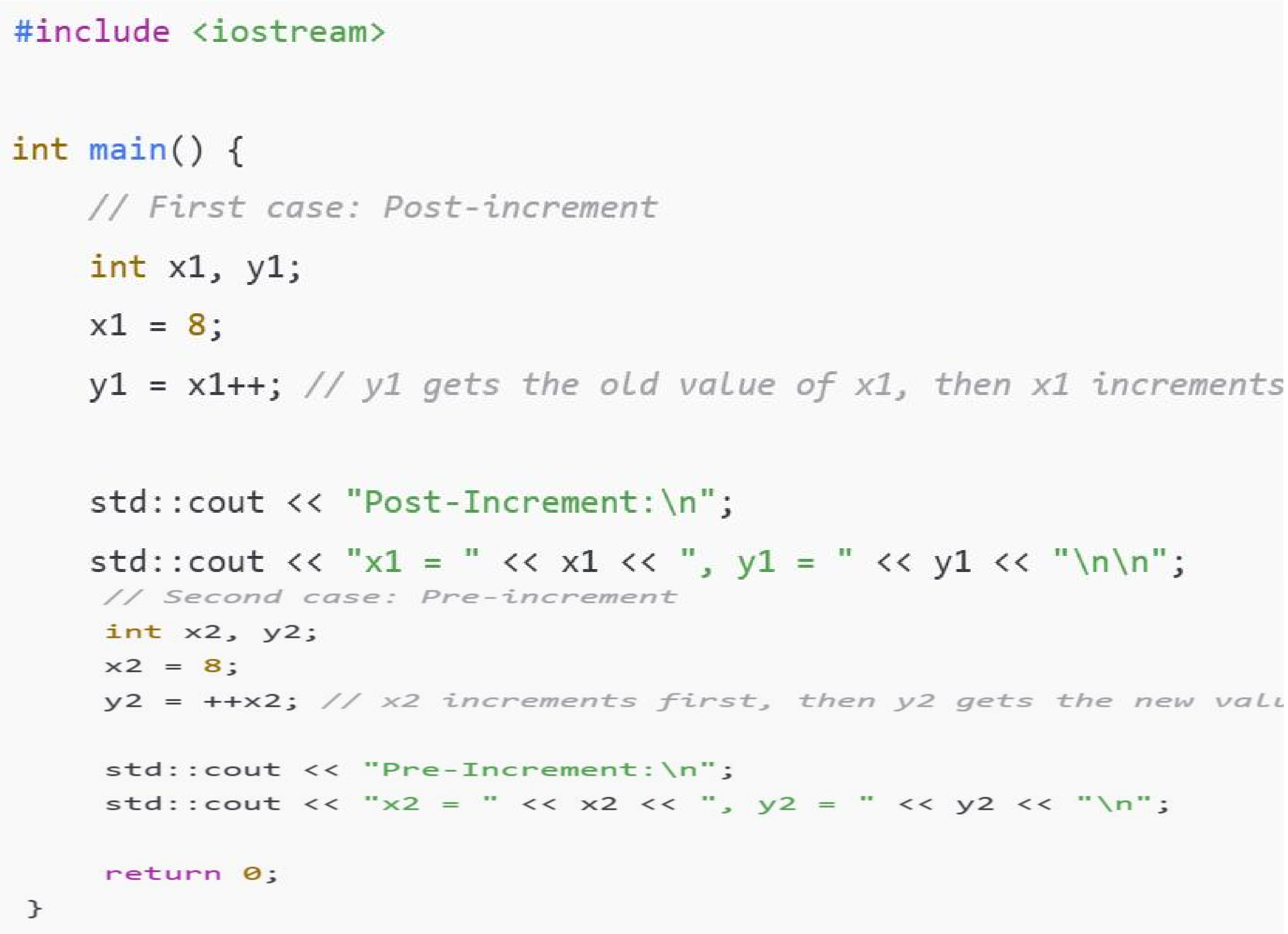
```
#include <iostream>

int main() {
     int age;

     age = 8;
     std::cout << age << std::endl;

 ++age;
 std::cout << age << std::endl;

     --age;
     std::cout << age << std::endl;


     return 0;
}
```

Output



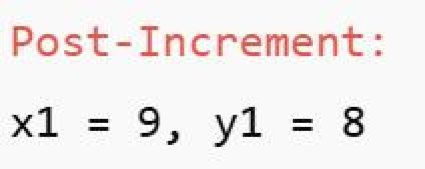
```
 8
9
8
```

Check the difference

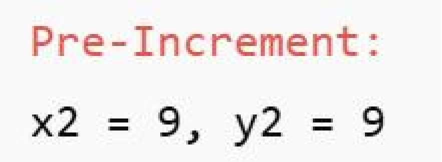


```
#include <iostream>


int main() {
    // First case: Post-increment
    int x1, y1;
    x1 = 8;
    y1 = x1++; // y1 gets the old value of x1, then x1 increments

    std::cout << "Post-Increment:\n";
    std::cout << "x1 = " << x1 << ", y1 = " << y1 << "\n\n";
    // Second case: Pre-increment
    int x2, y2;
    x2 = 8;
    y2 = ++x2; // x2 increments first, then y2 gets the new valu

    std::cout << "Pre-Increment:\n";
    std::cout << "x2 = " << x2 << ", y2 = " << y2 << "\n";

    return 0;
}
```

Output

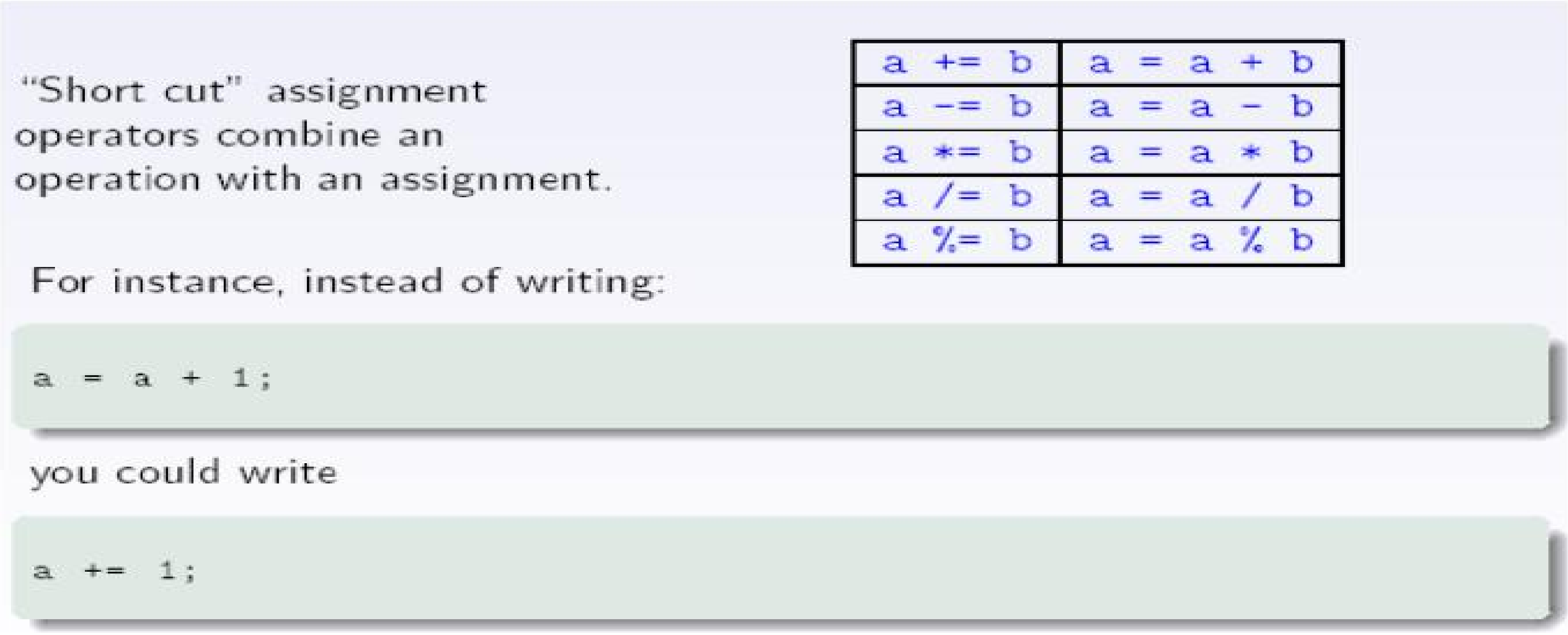


```
Post-Increment:
x1 = 9, y1 = 8
```



```
Pre-Increment:
x2 = 9, y2 = 9
```

Arithemetic Operators shortcut assignment



"Short cut" assignment operators combine an operation with an assignment.

a += b	a = a + b
a -= b	a = a - b
a *= b	a = a * b
a /= b	a = a / b
a %= b	a = a % b

For instance, instead of writing:

```
a = a + 1;
```

you could write

```
a += 1;
```

Control Structures

- C++ allows a program to make a decision based on the value of a condition. Such a condition must evaluate to true or false.

- Use logical expressions which may include:

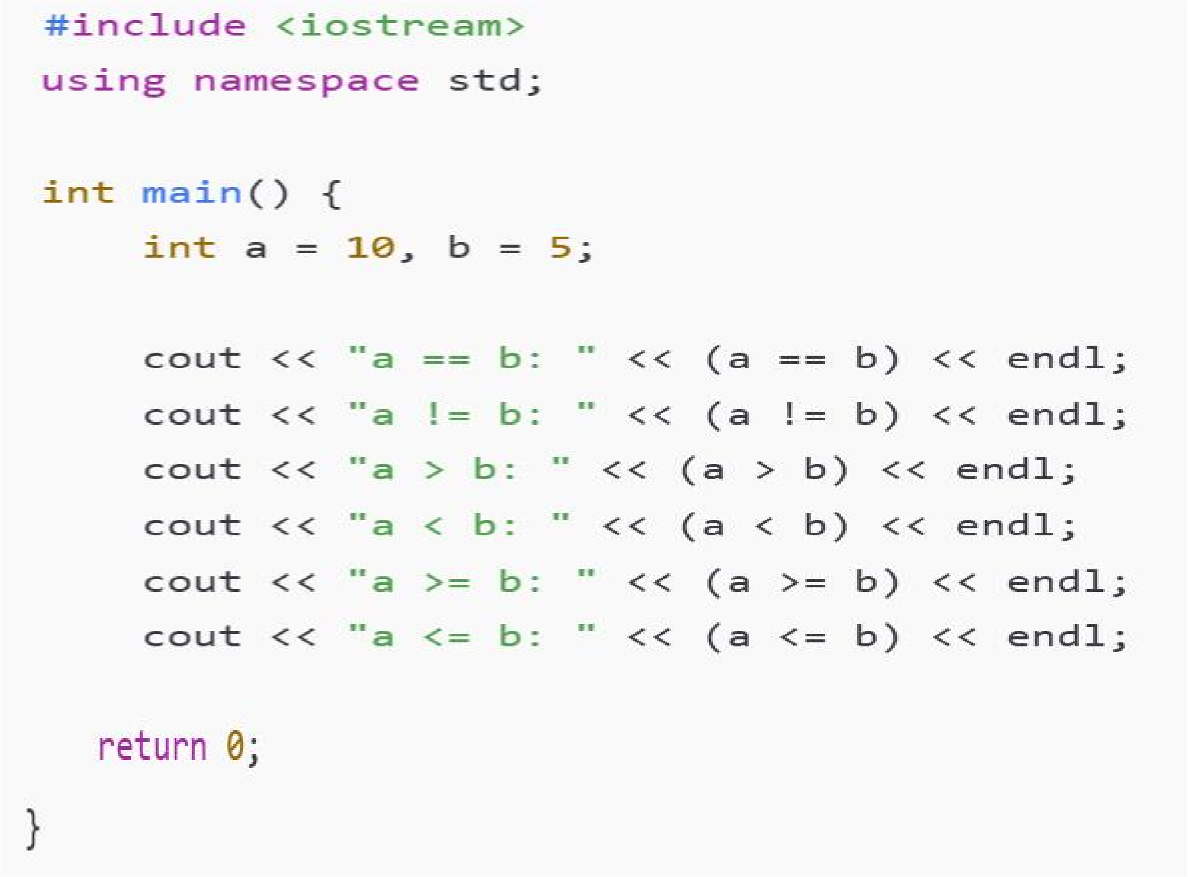
➢ Relational Operators

<   <=   >   >=   ==   !=

➢ Logical Operators

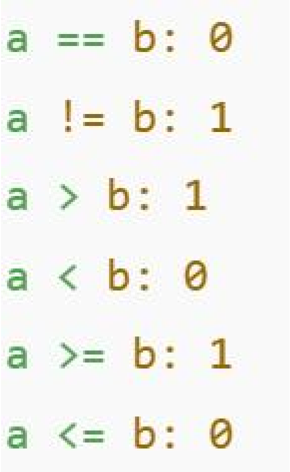
!   &&   II

Control Structures



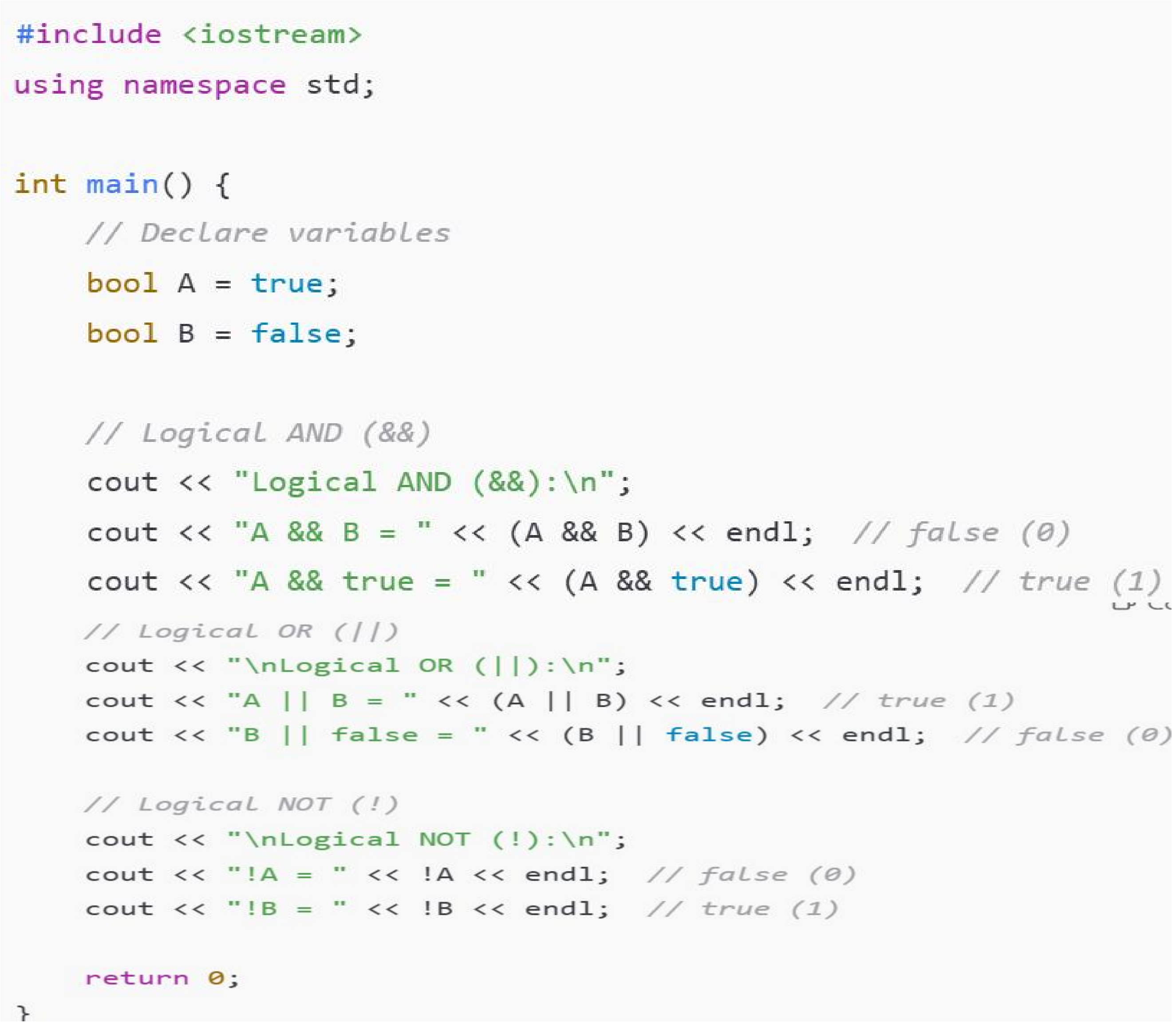
```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 10, b = 5;

    cout << "a == b: " << (a == b) << endl;
    cout << "a != b: " << (a != b) << endl;
    cout << "a > b: " << (a > b) << endl;
    cout << "a < b: " << (a < b) << endl;
    cout << "a >= b: " << (a >= b) << endl;
    cout << "a <= b: " << (a <= b) << endl;

  return 0;
}
```

Output



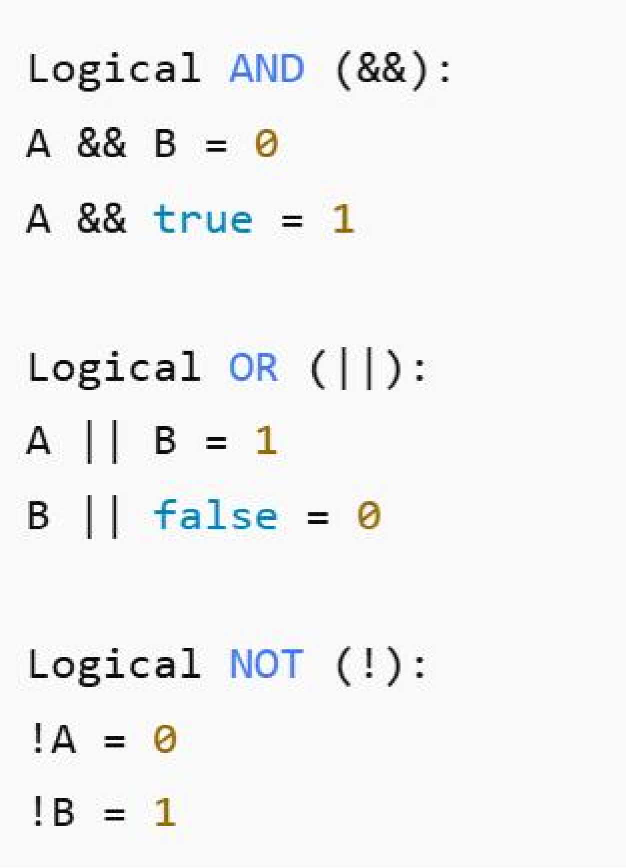
```
a == b: 0
a != b: 1
a > b: 1
a < b: 0
a >= b: 1
a <= b: 0
```

Control Structures



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

int main() {
    // Declare variables
    bool A = true;
    bool B = false;

    // Logical AND (&&)
    cout << "Logical AND (&&):\n";
    cout << "A && B = " << (A && B) << endl;  // false (0)
    cout << "A && true = " << (A && true) << endl;  // true (1)
    // Logical OR (||)
    cout << "\nLogical OR (||):\n";
    cout << "A || B = " << (A || B) << endl;  // true (1)
    cout << "B || false = " << (B || false) << endl;  // false (0)

    // Logical NOT (!)
    cout << "\nLogical NOT (!):\n";
    cout << "!A = " << !A << endl;  // false (0)
    cout << "!B = " << !B << endl;  // true (1)

    return 0;
}
```

Output



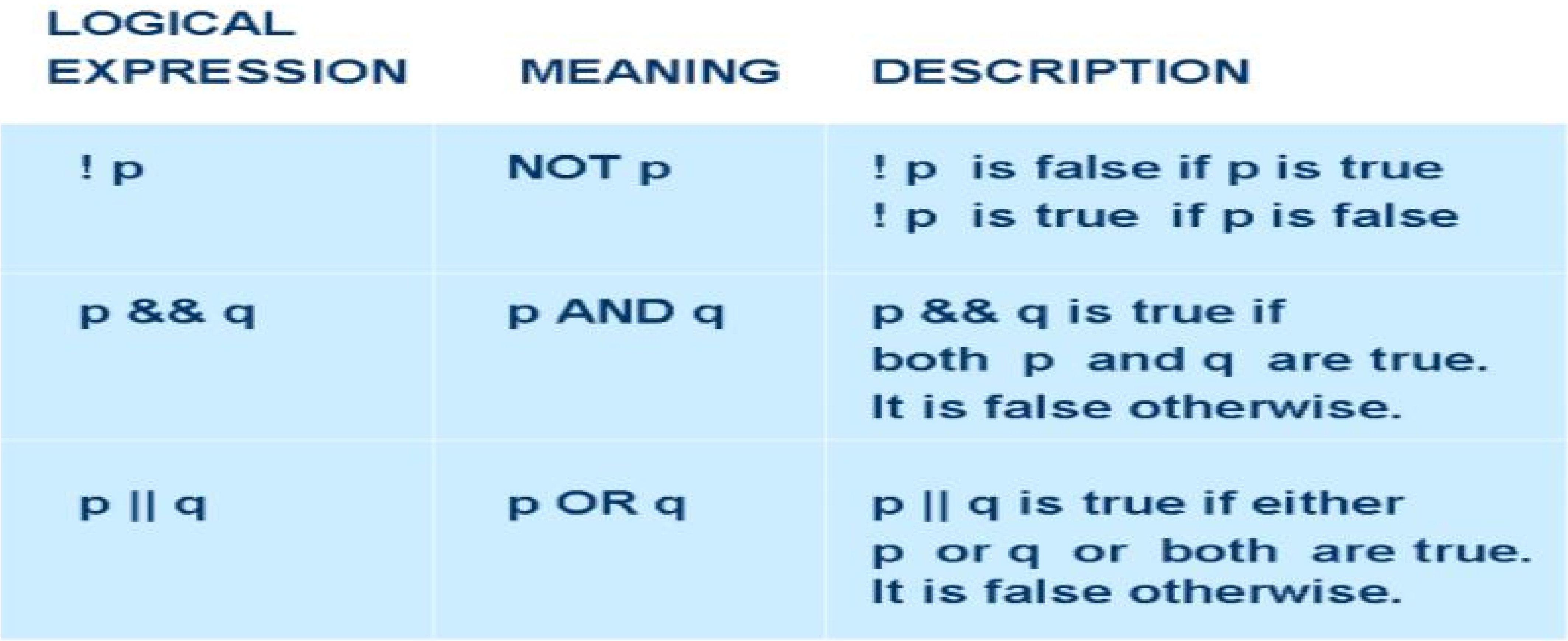
```
Logical AND (&&):
A && B = 0
A && true = 1

Logical OR (||):
A || B = 1
B || false = 0

Logical NOT (!):
!A = 0
!B = 1
```

Relational Operators

- int x, y;
  x = 4;
  y = 6;

Expression	**Value**
• x < y	true
• x != y	true
• y == x	false

Logical Operators



LOGICAL EXPRESSION	MEANING	DESCRIPTION
! p	NOT p	! p is false if p is true ! p is true if p is false
p && q	p AND q	p && q is true if both p and q are true. It is false otherwise.
p || q	p OR q	p || q is true if either p or q or both are true. It is false otherwise.

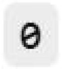
Example (1)

- int age, height;
  age = 25;
  height = 70;

Expression	Value
• !(age < 10)	1
• !(height > 60)	0

Example (2)

-
  ```
  int age, height;
  age = 25;
  height = 70;
  ```

Expression	Value
• (age > 50) && (height >60)	 0

Example (3)

- int age, height;
age = 25;
height = 70;

Expression	Value
• (height > 60) II (age >40)	1

Example (4)

- int age, height;
  age = 25;
  height = 70;

Expression	Value
! (height > 60) II (age >50)	0

Example (5)

- **taxRate is over 25% and income is less than 20000.**

  (taxRate > 0.25) && (income <20000)

- **temperature is less than or equal to 25 or humidity is less than 70%.**

  (temperature <= 25) II (humidity <0.70)

- **age is over 21 and age is less than 60.**

  (age > 21) && (age <60)

- **age is 21 or 22.**

  (age == 21) II (age == 22)

SAXONY EGYPT UNIVERSITY
FOR APPLIED SCIENCE AND TECHNOLOGY

THANK YOU