Programming Fundamentals with C++

Lecture 5 – Conditional Statements

Dr. Muhammad Sajjad

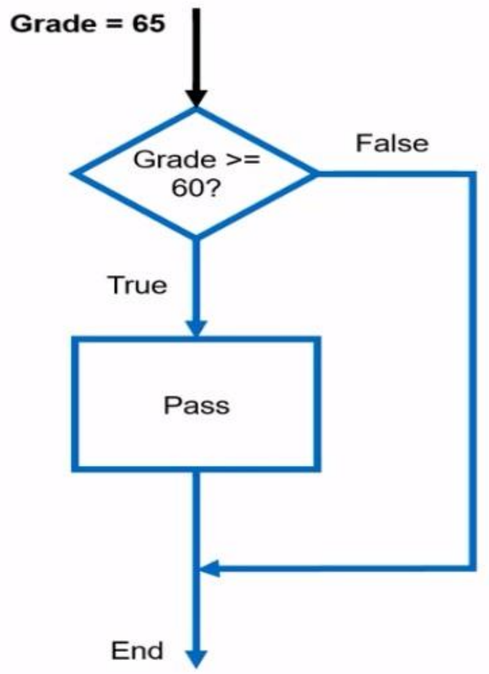
RA: Asad Ullah

Overview

- Conditional Statements
- Relational Expressions
- Relational Operators
- Logical Operators

Conditional Statements

- The statements of computer program are executed one after the other in the order in which they are written.
- This is known as sequential execution of the program.
- This order can be changed by using conditional statements.
- The conditional statements are used to execute (or ignore) a set of statements after testing a condition.
- The conditional statements are also called selection or decision statements.



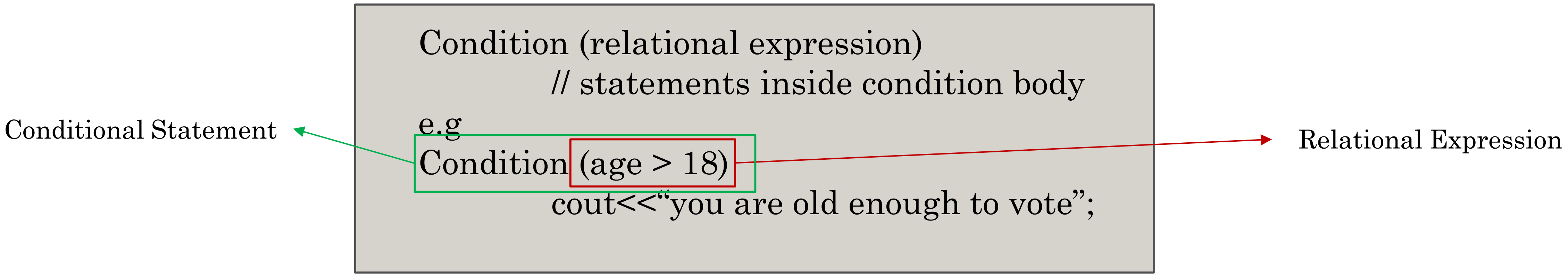
Execute these statements if condition is true

```
int main() {
        // some statmets
        condition
        if true
                // some statements
        if false
        // statements outside condition
```

Relational Expressions

- A relational expression consist of constant, variables or arithmetic expression that are combined by a relational operator.
- A relational expression is written to find a relation between two expression.
- Return a single value which is either **true** or **false**.

- **Example**: 12 > 7 is a relational expression. It indicate a relation between two constant. Since the constant value 12 is greater than 7, it returns a **true** value.
- The operator '>' is a relational operator.
- These expressions can be used as a condition for conditional statements.



```
Condition (relational expression)
            // statements inside condition body
e.g
Condition (age > 18)
            cout<<"you are old enough to vote";
```

Relational Operators

- A relational operator compares two values.
- The values can be any built-in C++ data type, such as char, int, float, or constants.
- These operators specify a relation between two expressions or values such as **equal to**, **less than**, **greater than** etc.
- Some of the relational operators we will discuss in details.

Relational Expressions

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

Int main () {
          int num;

          cout<<"Enter number: ";
          cin>>num;
          cout<<"num < 10 is "<< (num < 10) << endl;        // "<" is less than operator
          cout<<"num > 10 is "<< (num > 10) << endl;        // ">" is greater than operator
          cout<<"num == 10 is "<< (num == 10) << endl;      // "==" is equal to operator
          return 0;
}
```

Relational Operator

Greater than (>)

It is used if one value is greater than the other.

For example if v1 and v2 are two values then:

v1 > v2

- Returns **true** value if v1 is greater than v2.
- Returns **false** value if v1 is less than v2.
- **Example**: 9 and 3
  - 9 > 3 returns true
  - 3 > 9 returns false

Relational Operator

Greater than or Equal to (>=)

It is used if one value is greater than or equal to the other.

For example if v1 and v2 are two values then:

v1 >= v2

- Returns **true** value if v1 is greater than or equal to v2.
- Returns **false** value if v1 is less than v2.
- **Example**: 10 and 9
  - 10 >= 9 returns true
  - 9 >= 10 returns false
- **Example**: 4 and 4
  - 4 >= 4 returns true

Relational Operator

Less than (<)

It is used if one value is less than the other.

For example if v1 and v2 are two values then:

v1 < v2

- Returns **true** value if v1 is less than v2.
- Returns **false** value if v1 is greater than v2.
- **Example**: 9 and 3
  - 9 < 3 returns false
  - 3 < 9 returns true

Relational Operator

Less than or Equal to (<=)

It is used if one value is less than or equal to the other.

For example if v1 and v2 are two values then:

v1 <= v2

- Returns **true** value if v1 is less than or equal to v2.
- Returns **false** value if v1 is greater than v2.
- **Example**: 10 and 9
  - 10 <= 9 returns false
  - 9 <= 10 returns true
- **Example**: 4 and 4
  - 4 <= 4 returns true

Relational Operator

Equal to (==)

It is used if one value is equal to the other.

For example if v1 and v2 are two values then:

v1 == v2

- Returns **true** value if v1 is equal to v2.
- Returns **false** value if v1 is not equal to v2.
- **Example**: 5 and 5
  - 5 == 5 returns true
- **Example**: 5 and 6
  - 5 == 6 returns false

Relational Operator

Not Equal to (!=)

It is used if one value is not equal to the other.

For example if v1 and v2 are two values then:

v1 != v2

- Returns **true** value if v1 is not equal to v2.
- Returns **false** value if v1 is equal to v2.
- **Example**: 5 and 5
  - 5 != 5 returns false
- **Example**: 5 and 6
  - 5 != 6 returns true

Relational Operator

Example

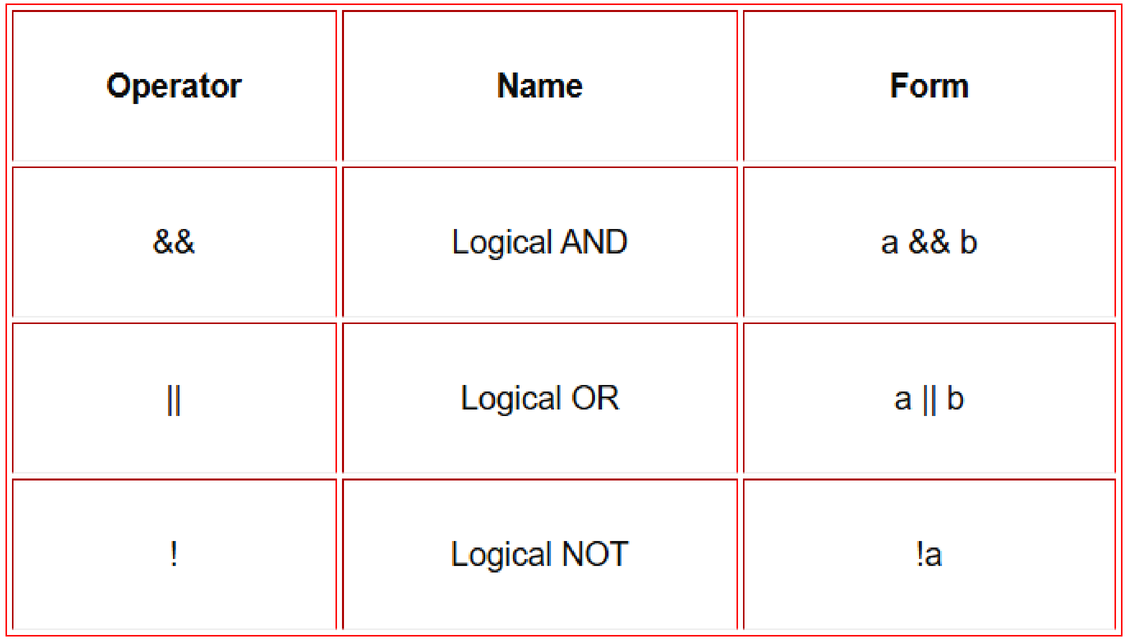
If x = 10, y = 20 and z = 5 then find out the output of the following relational expressions.

Relational Expression	Output Returned
x > y	False
y < z	False
y != x	True
x == z	False
x <= y	True
z >= y	False

Logical Operators

- Logical operators in C++ allow you to combine or modify Boolean expressions (expressions that evaluate to true or false).
- They're primarily used in control flow statements like if, while, and for loops to make more complex logical decisions.
- Here are the three main logical operators in C++:

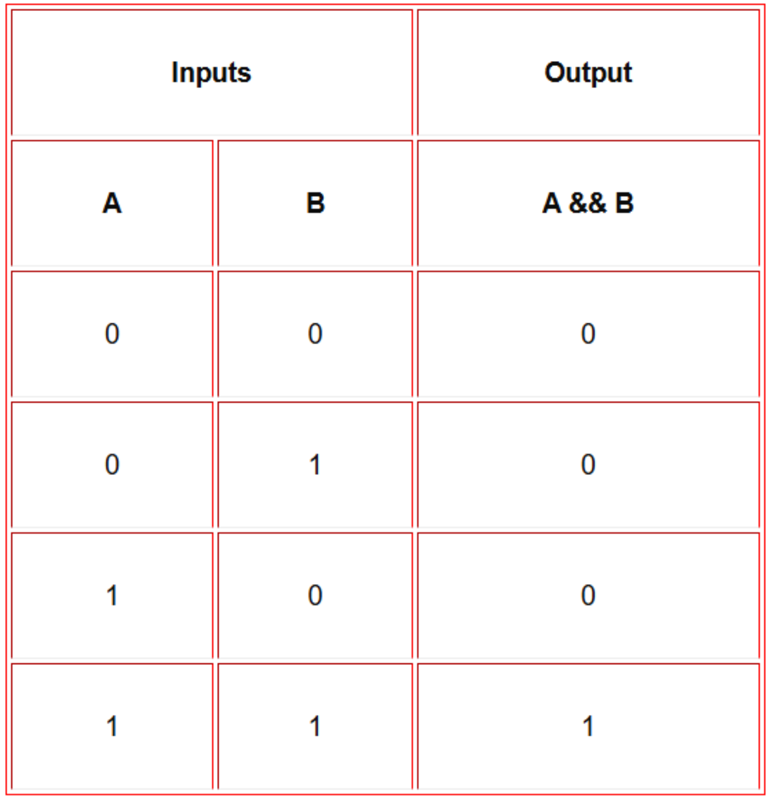
1.	&&	(AND operator)
2.	| |	(OR operator)
3.	!	(NOT operator)



Operator	Name	Form
&&	Logical AND	a && b
||	Logical OR	a || b
!	Logical NOT	!a

Logical Operators

1. **Logical AND Operator (&&)**
   - The Logical AND Operator works on the following Truth Table of AND Logic:



Inputs		Output
A	B	A && B
0	0	0
0	1	0
1	0	0
1	1	1

where, 0 = False, 1 = True in programming language

Logical Operators

1. Logical AND Operator (&&)

- The Logical AND Operator returns the Boolean value True or 1 when both the operands satisfy the conditions and are true in nature.
- If any one of the operands does not satisfy the condition, then it will return a False or 0 value.
- The operand values are by default converted into Boolean values and then the result is computed. Hence, the result returned by the operator is of type Boolean. Logical AND has left-to-right associativity.

```
int age = 20;
int grade = 80;
if (age > 18 && grade > 70) {
    cout << "Eligible for the program." << endl;
}
```

Logical Operators

2. Logical OR Operator (| |)

- The Logical OR Operator works on the following Truth Table of OR Logic:

Inputs		Output
A	**B**	**A || B**
0	0	0
0	1	1
1	0	1
1	1	1

where, 0 = False, 1 = True in programming language

Logical Operators

1. Logical AND Operator (&&)

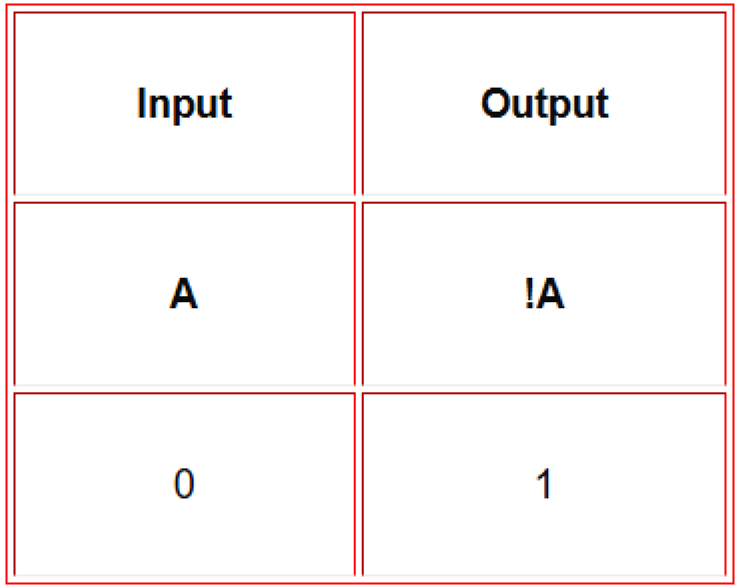
- The Logical OR Operator returns the Boolean value True or 1 when any one of the operands satisfies the conditions and are true in nature.
- If both the operands do not satisfy the condition, then it will return a False or 0 value.
- Logical OR has left-to-right associativity.

```
bool hasPermit = true;
bool hasExperience = false;
if (hasLicense || hasPermit) {
    cout << "Eligible for a provisional license." << endl;
}
```

Logical Operators

2. Logical NOT Operator (!)

- Logical NOT Operator is a Unary Operator i.e., it requires only one operand to work on. Following is the Truth Table of NOT Logic:



Input	Output
A	!A
0	1

where, 0 = False, 1 = True in programming language

Logical Operators

1. Logical AND Operator (&&)

- The Logical NOT Operator returns a negate value i.e. if the condition is not satisfied then it will return a True value.
- If the condition is satisfied, then it returns a False value.

```
bool isRaining = false;
if (!isRaining) {
    cout << "You can go outside." << endl;
}
```

Thank You