Lab # 4: Relational Operators, Logical Operators and Decisions

EC-102 – Computer Systems and Programming

Usman Ayub Sheikh

School of Mechanical and Manufacturing Engineering (SMME),
National University of Sciences and Technology (NUST)

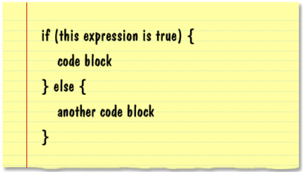
September 29, 2015

Outline

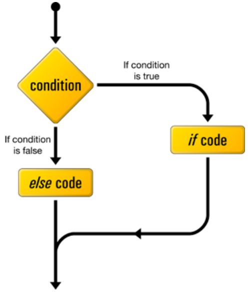
Should I remove this code...
or just comment it out until management changes their mind

Relational Operators – Why do we need them?

- Most programs decide what to do in response to changing circumstances
- The flow of control jumps from one part of the program to another in response to such events
- Program statements that cause such jumps are called **control statements** e.g. decisions and loops
- How many times a loop is executed or whether a decision results in the execution of a section of code depends on whether certain expressions turn out to be *true* or *false*



```
if (this expression is true) {
    code block
} else {
    another code block
}
```



Relational Operators – What are they?

- A relational operator compares two values
- The comparison involves such relationships as equal-to, lesser-than, and greater-than
- The result of the comparison is true or false

Relational Operators – Examples

Example # 1

```cpp
// this program demonstrates relational operators in a
    comparison of int, float and char constants
#include <iostream>
using namespace std;
int main()
{
    cout << (10 > 20) << endl; // false
    cout << (10 < 20) << endl; // true
    cout << (20 == 20) << endl; // true

    cout << (20.5 > 20.0) << endl; // true
    cout << (20.5 == 2.5) << endl; // false

    cout << ('a' == 'a') << endl; // true
    cout << ('a' > 'b') << endl; // false
    return 0;
}
```

Relational Operators – Examples

Example # 2

```
// relational operators in a comparison of int variables
#include <iostream>
using namespace std;
int main()
{
    int jane = 44; // assignment statement
    int harry = 12;

    cout << (jane == harry) << endl;
    cout << (harry <= 12) << endl;
    cout << (jane > harry) << endl;
    cout << (jane >= 44) << endl;
    cout << (harry != 12) << endl;
    cout << (7 < harry) << endl;
    cout << (0) << endl;
    cout << (44) << endl;
    return 0;
}
```

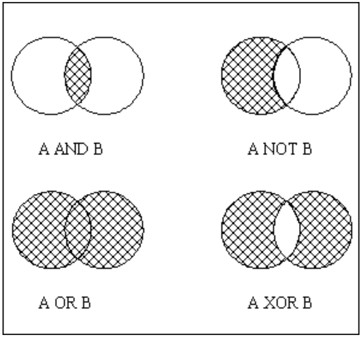
Relational Operators in C++

Here's a complete list of C++ relational operators,

Operator	Meaning
>	Greater than
<	Lesser than
==	Equal to
! =	Not equal to
>=	Greater than or equal to
<=	Lesser than or equal to

Logical Operators – Why do we need them?

- While relational operators can be used to test whether a particular condition is true or false, they can only test one condition at a time
- Often we need to know whether multiple conditions are true at once
- Other times, we need to know whether any one of the multiple conditions is true



Logical Operators – What are they?

- A relational operator is used to combine two Boolean expressions
- For example, to check whether a number x entered by the user satisfies the expression $20 < x < 30$, we would need to logically connect both the expressions `(x > 20)` and `(x < 30)` and see if there combination yields true or false
- The logical connection in this case is the word AND
- The result of logical operation is either true or false

Logical Operators in C++

Here's a complete list of C++ logical operators,

Operator	Meaning
&&	AND
||	OR
!	NOT

Logical AND Operator (&&)

Expression 1	Expression 2	Expression 1 && Expression 2
false	false	false
false	true	false
true	false	false
true	true	true

Logical OR Operator (||)

Expression 1	Expression 2	Expression 1 || Expression 2
false	false	false
false	true	true
true	false	true
true	true	true

Logical NOT Operator (!)

- Logical NOT operator is a unary operator
- It can be used to reverse the meaning of a Boolean expression

Expression	!Expression
false	true
true	false

Logical Operators – Example

```cpp
// this program demonstrates logical operators
#include <iostream>
using namespace std;
int main()
{
    int jane = 44;
    int harry = 12;

    cout << (jane == harry && harry <= 12) << endl;
    cout << (jane == harry || harry <= 12) << endl;
    cout << !(jane == harry) << endl;

    cout << (jane > harry && jane >= 44) << endl;
    cout << (jane > harry || jane >= 44) << endl;
    cout << !(jane > harry || jane >= 44) << endl;
    return 0;
}
```

Decision Making

- Decision making is about deciding the order of execution of statements based on certain conditions
- These statements require the programmer to specify:
    - One or more expressions to be evaluated or tested by the program along with
    - One or more statements to be executed if the condition turns out to be true, and optionally
    - One or more statements to be executed if the expression turns out to be false

Decision Making in C++

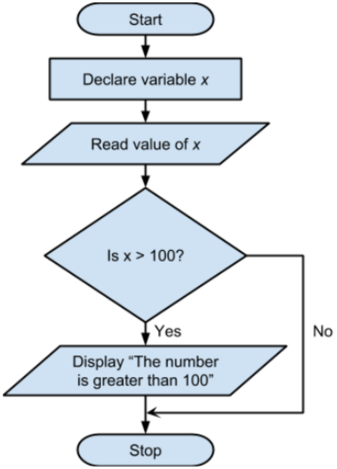
- Decisions can be made in C++ in many ways. The most important is with the `if...else` statement. This statement can also be used without the `else`, as a simple `if` statement.
- Another decision statement, `switch`, creates branches for the multiple alternative sections of code, depending on the value of a single variable

The `if` Statement

Algorithm

1. Start
2. Declare variable x
3. Read value of x
4. If x is greater than 100 then display "The number is greater than 100"
5. Stop

Flowchart



The if Statement

Code

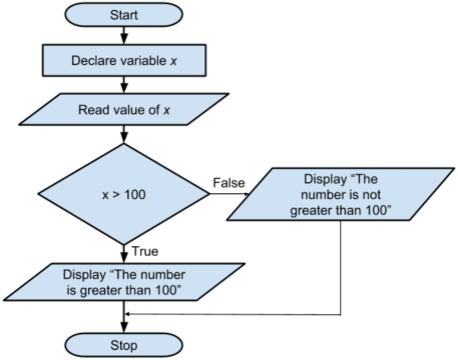
```cpp
// this program demonstrates IF statement
#include <iostream>
using namespace std;
int main()
{
    int x;

    cout << "Enter a number: ";
    cin >> x;

    if(x > 100)
    {
        cout << "That number is greater than 100\n";
    }

    return 0;
}
```

The `if...else` Statement

Algorithm

1. Start
2. Declare variable x
3. Read value of x
4. If x is greater than 100 then display “The number is greater than 100”
5. Else, display “The number is not greater than 100”
6. Stop

Flowchart


Start
Declare variable x
Read value of x
x > 100
False
Display “The number is not greater than 100”
True
Display “The number is greater than 100”
Stop

The if...else Statement

Code

```
// this program demonstrates IF statement
#include <iostream>
using namespace std;
int main()
{
    int x;
    cout << "Enter a number: ";
    cin >> x;

    if(x > 100)
    {
        cout << "That number is greater than 100\n";
    }
    else
    {
        cout << "The number is not greater than 100\n";
    }
    return 0;
}
```