Classes, Control Flow, Relational Operators

Thomas Merkh
tmerkh55@gmail.com

Section 2, Week 3

April 14, 2020
Overview

1. Classes
2. Relational Operators
3. Control Flow
4. Next Time
Last time we discussed casting, cmath, and strings. These are essential components of most c++ programs. As a refresher:

- Casting is how the compiler deals with performing operations between compatible types (adding int and double, for example)
- #include <cmath> is a way to include many common mathematical functions in your program so you do not have to code them yourself (raising number to a power, for example)
- Strings are objects which are easier to work with than arrays of chars. These objects and their methods are defined by the string class.

What is the string class?
1. Classes

2. Relational Operators

3. Control Flow

4. Next Time
What is a class?

- A class is a user-defined data type, which holds its own data members and member functions.
- These functions can be accessed and used by creating an instance of that class.
- Classes are an expanded concept of data structures: like data structures, they can contain data members, but they can contain functions too.

*Note*: A class in C++ is the primary building block that leads to object-oriented programming (OOP). Originally, this was the difference between the functional programming language C and the OOP language C++. 
Not all of this will be clear yet, but consider the code:

class ExpFun{
    public:
        // Data member
        double power;

        // Default constructor
        ExpFun();

        // Destructor
        ~ExpFun();

        // Member function
        void printPower();
};

This is the header file associated with the class ExpFun. Note that none of the functions in the class have been defined.
Often, the functions in a class will be declared in a .h file, and defined in a .cpp file:

```cpp
#include "ExpFun.h"
#include <iostream>

ExpFun::ExpFun() {power = 2.0;};
ExpFun::~ExpFun() {}
void ExpFun::printPower()
{
    std::cout << "Power = " << power << std::endl;
};
```
Classes: An example

Often, the functions in a class will be declared in a `.h` file, and defined in a `.cpp` file:

```cpp
#include "ExpFun.h"
#include <iostream>

ExpFun::ExpFun() {power = 2.0;};
ExpFun::~ExpFun() {};
void ExpFun::printPower()
{
    std::cout << 'Power = ' << power << std::endl;
};
```

Then, we can use this new data type or class in our main code:

```cpp
#include "ExpFun.h"

int main()
{
    ExpFun Myfirstclass;
    Myfirstclass.printPower();
    return 0;
}
```
There is a lot going on here and a lot to know: namespaces, separation into several files, public member functions vs. private, constructors, destructors, include statements, and more. **Don’t worry about it yet, we will learn this in time.**
There is a lot going on here and a lot to know: namespaces, separation into several files, public member functions vs. private, constructors, destructors, include statements, and more. **Don’t worry about it yet, we will learn this in time.**

Right now, just know that classes are out there, we can create huge classes with many operations defined for them, and that classes are possibly **the most important concept in C++ at this level.** Our best example?

- The string class
- Holds a character array
- Has many functions defined for manipulating that character array (like concatenating two strings, lower-casing all letters, etc.)
1. Classes

2. Relational Operators

3. Control Flow

4. Next Time
Switching speeds away from classes and strings and more toward the infrastructure of most C++ codes, we will be learning about control flow for the rest of this discussion. In order to have the program make decisions, it needs to be able to compare two things.

The relational operators we’ll consider are:

- `==` compares if two expressions are equal
- `!=` compares if two expressions are not equal
- `<=` (or `>=`) compares if first expression is less than (or greater than) or equal to the second
- `<` (or `>`) compares if first expression is less than (or greater than) the second
See if you can guess what this code would do:

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

int main()
{
    int a = 10, b = 20, c = 10;
    if (a > b) {cout << "a is greater" << endl;}
    if (a < b) {cout << "b is greater" << endl;}
    if (a <= c) {cout << "a is less than/equal to c" << endl;}
    if (a != b-c) {cout << "a not equal to b-c" << endl;}
}
```

Which statements will output to the console?
See if you can guess what this code would do:

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

int main()
{
    int a = 10, b = 20, c = 10;
    if (a > b) {cout << 'a is greater' << endl;}
    if (a < b) {cout << 'b is greater' << endl;}
    if (a <= c) {cout << 'a is less than/equal to c' << endl;}
    if (a != b - c) {cout << 'a not equal to b-c' << endl;}
}
```

Which statements will output to the console?

If you guessed statements two and three, you are correct. This provides an example of how \(<, \>, \leq, \geq, !=,\) and \(==\) work.
Relational Operators and Control Flow

Looking again at:

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

int main() {
    int a = 10, b = 20, c = 10;
    if (a > b) {cout << "'a is greater'" << endl;}
    if (a < b) {cout << "'b is greater'" << endl;}
    if (a <= c) {cout << "'a is less than/equal to c'" << endl;}
    if (a != b - c) {cout << "'a not equal to b-c'" << endl;}
}
```

We see four if statements. Such statements are regarded as control flow statements, and these are the backbone of any C++ program. Next, we will discuss the most important types of control flow you’ll be using.
1 Classes

2 Relational Operators

3 Control Flow

4 Next Time
Figure: An visualization of what control flow in a program may look like. Programs are sequential by nature (not considering parallel programming), meaning that they can be thought of as being a flow chart + data manipulations. Photo Credit: Wikipedia “Control Flow”
Our first type of control flow are if, else if, else statements:

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

Although simple, these statements are absolutely essential to be proficient with when coding C++. We will see examples in a few slides.
Logical Operators

To make your control flow statements more sleek, it is useful to utilize *logical operators*. These operators take in two boolean values (true/false values, 0/1 values), and return a single boolean value. For example:

- `||` means *or*, `&&` means *and*
- `true || false` will return `true`
- `true && false` will return `false`
- `false || false` will return `false`
- `false && false` will return `false`
- `true || true` will return `true`
- `true && true` will return `true`
If you have a question at time $t \in [0, 24)$ and

```cpp
if (t > 8 || t < 18) {
    cout << "Within business hours" << endl;
} else if (t < 8) {
    cout << "Before business hours" << endl;
} else {
    cout << "After business hours" << endl;
    if (t > 23 && t < 24) {
        cout << "Probably should be sleeping" << endl;
    }
}
```

If you have a question at $t = 23.25$ i.e. 11:15pm at night, what message would you receive?
If you have a question at time $t \in [0, 24)$ and

```cpp
if (t > 8 || t < 18) {
    cout << "Within business hours" << endl;
} else if (t < 8) {
    cout << "Before business hours" << endl;
} else {
    cout << "After business hours" << endl;
    if (t > 23 && t < 24) {
        cout << "Probably should be sleeping" << endl;
    }
}
```

If you have a question at $t = 23.25$ i.e. 11:15pm at night, what message would you receive? The correct answer would look like

After business hours
Probably should be sleeping
1. Classes

2. Relational Operators

3. Control Flow

4. Next Time
Next Time

Next lecture is crucial, it discusses loops, which are the other half of the control flow statements upon which any C++ program is built. Office hours are from 12-1:30. If you plan on coming to office hours, please let me know!