Comp 11 Lectures

Mike Shah

Tufts University

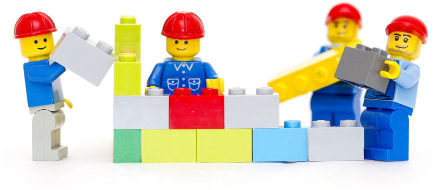
July 17, 2017

Object Oriented Programming 2

Comp 11 - Pre-Class warm up

Object-Oriented Programming is comparable to lego bricks. You have blocks, you build bigger pieces from smaller ones, and generally you can work in teams more easily by separating out tasks.

An important part of building a lego set was reading the blueprint(interface or spec), and then planning out the implementation.



Lecture

Frances Allen

Figure 1: Frances Allen was the first female computer scientist to win the Turing Award in 2006. She is best known for her contributions in optimizing compilers in the programming language FORTRAN. Her seminal paper in 1996 on Program transformation laid many of the foundations down for this work.

Speaking of Compilers

The Compiler

- It has been a while since we have talked about the compiler.
- We previously left off with knowing that it is a tool that turns our program into 1's and 0's.
- Let's uncover another layer of what is going on.

Object Files

- You might have noticed these .o files that occasionally pop up.
- In fact, maybe you have seen something like this on a tutorial.
  clang++ -c myfile.cpp
- This generates what is known as an object file.
- An object file describes at a low level what code is available.
- We can investigate our object file further using some special tools:
  llvm-nm-3.8 myfile.o

Object Files Put to work

- That's neat, but I cannot execute an object file. An object file is not an executable(.exe or a.out) file, it is simply machine code.
- But what we can do, is link multiple object files together!
- So now if we split our code into multiple files, we can link together all of the .o files.
- There is an actual program called a linker that performs this step.
- A simple way to do this in one step if we have multiple .cpp files (which sort of hides the linker from us).
- clang++ myfile.cpp myfile2.cpp myfile3.cpp -o myExecutable

Compilation Picture

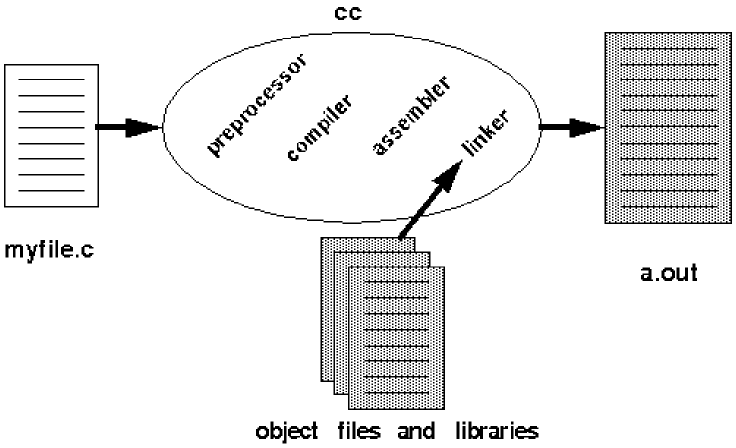


Figure 2: Compilation process

`https://courses.cs.washington.edu/courses/cse378/97au/help/compilation.html`

Object-Oriented Programming Separating the Interface from the Implementation

Code Maintenance

- The reason we are learning about linking object files, is because we are going to be separating code into different parts.
- Now we will be able to write bigger software–many small modules are generally easier to maintain than one large one.
- We will also be able to share reusable components with other projects! (Whether that means the source, or the .object files)
- It is generally easier to debug and test a smaller chunk of code.

Header Files - The Interface

- When we include code(#include <iostream>), we are essentially copying and pasting code from a .h file file into that file.
- A .h file is a "Header" file. We have included vector, stack, and other header files previously.
- Each of these header files generally contains a class (sometimes more than one), with a description of how to use it.
- By description, I mean a specification with function definitions in the class. No actual loops, if-statements, or program logic.

Header Files - Example

```
class myClass{
public:
    // Constructor
  myClass();
    // Destructor
    ~myClass();
    // No code, just the return type and any parameters
   needed
    void printMsg(int times);

};
```

Listing 1: A header file myClass.h

C Preprocessor

- When we include code, we are essentially copying and pasting code from a .h file file into that file, and then that file will get compiled.
- Header files that we reference from C++ have <>'s, header files that we include from our computer contain ""'s
- Additionally, we can include some header guards (using C-preprocessor syntax) to prevent us from including a file multiple times.
- If we do not do this, then we will get 'symbol multiply defined' or other ambiguity errors from the compiler (i.e. it won't know which definitions to use).

C Preprocessor Language

- Items with a pound symbol in front of them do not get compiled into code.
- Instead, they guide the compiler as what to do while we are compiling our source code.
- More information is here on how to control the preprocessor. `http://www.cplusplus.com/doc/tutorial/preprocessor/`

Header Guards Example

```cpp
// If symbol not defined in preprocessor
// Then define it.
// Now that it is defined, the above check will
// fail, and our code will never get accidentally
// included twice!
#ifndef MYCLASS_H
#define MYCLASS_H

class myClass{
public:
    // Constructor
  myClass();
    // Destructor
    ~myClass();
    // No code, just the return type and any parameters
    needed
    void printMsg(int times);
};
#endif  // End our preprocesor #ifndef block!
```

Listing 2: Notice this time how we added our file

Implementation

- Now that we have the interface, we have to write the implementation.
- This typically means creating a .cpp file with the same name.
- We then implement each function, and we precede it with the class name and two colons (This is C++ grammar).
- void myClass::printMsg(int times) { // some code }
- Now our member functions are neatly organized away.

Implementation Example

```cpp
#include "myClass.h" // First include our interface
#include <iostream> // Any other header files we need for
    implementation

myClass::myClass(){
  // Nothing to do in constructor
}
myClass::~myClass(){
  // Nothing to do in destructor
}
// Make sure return type and parameters match
void myClass::printMsg(int times){
  for(int i =0; i < times; ++i){
        std::cout << "Hello from myClass\n";
      }
}
```

Listing 3: Writing myClass.cpp

Usage of our Header

```
#include <iostream>
#include "myClass.h" // Use double quotes
// The file path to our file is relative to where
// we compile our code.

int main(){

  myClass m;

  m.printMsg(5);

  return 0;
}
```

Listing 4: Usage of myClass.cpp

Compilation of everything together

- clang++ myClass.cpp main.cpp -o myProgram
- Why is the header file not included?
- Remember, it does not need to be. Header files are essentially copy and pasted in

Operating overloading

Power of abstraction

- As we abstract our code into smaller pieces, it becomes easier to think about more abstractions or functionality.
- In this case, we are going to add more power with what is known as operator overloading.
- This means we are going to add meaning to operators like =,+,-,*, ¡¡, [], for our own custom data types(clases and structs).

Revisiting copy constructor

- As we remember, with a copy constructor C++ makes one for us by default.
- If we are working with pointers, we need to perform a deep copy, and write our own.
- We saw how we could do this when we initialize an object, but we may want to just assign anywhere in our code an object to another.
- In order to do this, we will overload the '=' operator.
- (In fact, when we do this, we are also overriding the assignment operator that C++ tries to generate for us too.)
- So in general, you will always write two copy constructors.

this keyword

- We have another new C++ keyword. The **this** keyword, refers to the current instance of an object.
- We use this keyword only within a class. It is implicitly available.

this keyword usage

```cpp
myClass & myClass::myClass = (const myClass &t)
{
   // Check for self assignment(e.g. avoid doing A = A)
   if(this != &t){
     // perform our deep copy in this code block
   }

   // We return a pointer to our current object.
   // Why? Look at our return type, and think about
   // what we are modifying. We are modifying our current
   // instance of an object.
   return *this;
}

// Well, we need to return
// myClassA = myClassB;
```

Listing 5: Overloaded Assignment operator

templating

templating when splitting up files

- In the activity and lab today, you are going to notice the interface and implementation are templated.
- Remember though, templates allow us to use every type, but when we have many files separated, the compiler cannot infer the types easily.
- So we have to manually tell it which types we will use.
- To do so, put at the bottom of your .cpp the instantiations (template defintion) you would like to use.
- Example for using int: template class CompVector<int>;
- Example for using float: template class CompVector<float>;

In-Class Activity

http:
//www.mshah.io/comp/11/activities/activity11/activity.pdf

Activity Discussion

Review of what we learned

- (At least) Two students
- Tell me each 1 thing you learned or found interesting in lecture.

5-10 minute break

To the lab!

Lab: `http://www.mshah.io/comp/11/labs/lab11/lab.pdf` [1]

[1] You should have gotten an e-mail and hopefully setup an account at `https://www.eecs.tufts.edu/~accounts` prior to today. If not–no worries, we'll take care of it during lab!