CPSC 427: Object-Oriented Programming

Michael J. Fischer

Lecture 2
January 21, 2016

Tasks for this week

- Sign up for a Zoo account and a CPSC 427 course account.
- Read Chapters 1–3 of *Exploring C++*.
- Start work on problem set 1 when available (probably tomorrow).

C++ Overview

Why did C need a ++?

Chapter 2 of Exploring C++

1. C was designed and constructed a long time ago (1971), as a language for writing Unix.
2. The importance of data modeling was very poorly understood at that time.
3. Data types were real, integer, character, and array, of various sizes and precisions.
4. It was important for C to be powerful and flexible, but not to have clean semantics.
5. Nobody talked about portability and code re-use.

Today, we demand much more from a language.

C++ was Designed for Modeling

Design goals for C++ (Bjarne Stroustrup)

1. Provide classes (replacing structs) as a means to model data.
2. Let a class encapsulate data, so that its implementation is hidden from a client program.
3. Permit a C++ program to link to libraries from other languages, especially FORTRAN.
4. Produce executable code that is as fast as C, unless run-time binding is necessary.
5. Be fully compatible with C, so that C programs could be compiled under a C++ compiler and still work properly.

General properties of C++

- Widely used in the real world.
- Close to the machine and capable of producing efficient code.
- Gives a programmer fine control over the use of resources.
- Supports the object-oriented programming paradigm.
- Supports modularity and component isolation.
- Supports correctness through privacy, modularity, and use of exceptions.
- Supports reusabale code through derivation and templates.

C++ Extends C

- C++ grew out of C.
- Goals were to improve support for modularity, portability, and code reusability.
- Most C programs will compile and run under C++.
- C++ replaces several problematic C constructs with safer versions.
- Although most old C constructs will still work in C++, several should *not* be used in new code where better alternatives exist.

  Example: Use Boolean constants `true` and `false` instead of 1 and 0.

Some Extensions in C++

- Comments // (now in C11)
- Executable declarations (now in C11)
- Type bool (now in C11)
- Enumeration constants are not synonyms for integers
- Reference types
- Definable type conversions and operator extensions
- Functions with multiple methods
- Classes with private parts; class derivation.
- Class templates
- An exception handler.

Building a Project

Compilation modules

An **application** (or **executable** or **command** file) is built from a number **compilation modules**, also called **object files** or `.o` files. Often, `.o` files are packed together into **library** files, which have extensions `.a` or `.so`. Think of these files as components of the finished application.

Modules are joined together during final assembly of the application. This step of the process is called **linking**.

Building compilation modules

Modules are built from **implementation files**, also called **code files**, or `.cpp` files. These are the files that contain executable C++ code. A `.cpp` source file can be **compiled** to produce a corresponding `.o` object file.

Object files can be produced by different programmers at different times. Many of the modules you will be using come pre-compiled and pre-installed on your machine. Only during linking do all of the required object files and libraries need to be collected together.

System libraries are often found in directories `/lib`, `/usr/lib`, or `/usr/lib64`, but they can be placed anywhere as long as the *linker* is informed about where to find them.

Header files

Header files contain **declarations** about the functions and objects contained in the module, but they are not compiled alone and they do not produce object code.

Modules will generally need to refer to data and functions provided by other modules. In order to do this, they need a **blueprint** of those entities which describes their properties.

The blueprint takes the form of a **header file**, also called a `.h` or a `.hpp` file. In this course, we will use the `.hpp` extension to denote C++ header files, reserving the older `.h` extension for C header files.

Header files for system modules are often found in the `/usr/include` directory, but they can be placed anywhere as long as the *compiler* is informed about where to find them.

Compiling and linking in linux

The command for compiling and linking in linux is `g++`, the GNU implementation of C++. `g++` is a very powerful tool and requires many `man` pages to describe.

When used with the `-c` switch, it build an object file from a source code file.

Otherwise, it builds an executable from one or more object modules. It invokes the GNU linker `ld` to accomplish this task.

If called with one or more source code files and object files but no `-c` switch, it first compiles all of the source code files and then links the resulting object files with any object files from the command line and the libraries.

One-line compilation

Often all that is required to compile your code is the single command

```
g++ -o mycommand <switches> *.cpp
```

We will generally be using the following switches:

`-g -O1 -Wall -std=c++11`.

The job of the project manager

As we've seen, a project consists of many different files. Keeping track of them and remembering which files and switches to put on the command line is a major chore.

To aid in this task, one uses one of a number of project development tools such as `make` or **Integrated Development Environments**.

Command line development tools

- A text editor such as `emacs` or `vi`.
- The compiler suite: `g++`.
- Project management: `make`

The default `g++` compiler installed on the Zoo is version 4.8.6-4. The newer version 5.3.0 and associated files is installed in `/usr/local/gcc-5.3.0`.

The principal difference is that the newer version provides fuller support for the C++11 and C++14 language standards.

Both versions will work perfectly well for many of the assignments, but for some of the later assignments, you will need to configure your environment to use the newer compiler.