Data Structures and Algorithms

Section: Modular Design and Make

Devere Anthony Weaver

Preprocessor and include

The **preprocessor** scans a file before compilation to look for preprocessor directives. The most common preprocessor directive is the **#include** directive which directs the compiler to replace the line by the entire contents of a given file.

Header file guards are used to tell the compiler to only include the contents of a header file once. Good practice tends to simply go ahead and provide header guards for every header file.

Separate Files for Classes

C++ programmers typically will use two files when defining a class.

- ClassName.h - contains the class definition, including data members and member function declarations
- ClassName.cpp - contains member function definitions

It is necessary to include the class's header file for any files that want to use this class. It is also "best practice" to create separate header and implementation files for each separate class.

One of the advantages to using the separate files approach is that it shortens compilation time. If a change is made in only one file, then only that file has to be recompiled.

Unit Testing (classes)

A **testbench** is a program whose only job is to thoroughly test another program or portion of a program via a series of input/output checks known as **test cases**. **Unit testing** means to create and run a testbench for a specific item (or unit) like a function or a class.

Some features of a good testbench include:

- automatic checks
- independent test cases (e.g. assign new values instead of relying on other ones
- 100% code coverage (make sure every line of code is executed
- include not just typical values but also border cases
- only output any FAILED tests

Regression testing means to retest an item like a class anytime that item is changed. The name means that if a previously-passed test case fails, then the item has "regressed".

Note that for commercial software, testing can consume a majority of the development time to ensure that a product as error-free as possible.

Modular Compilation

Modular compilation is an approach to compilation that separates compiling and linking steps within the compilation process. Each source file is independently compiled into an object file which contains machine instructions for the compiled code and references for calls to functions and access to object defined in other source files or libs.

Once each source file has been compiled into an object file, the **linker** will create the final executable by linking together the object files and libraries. This type of approach allows for changes to be made to specific files without having to recompile the files that haven't changed, speeding up compile time after changes.