Today

- Libraries review
- Static vs dynamic linking
- Compilation
- Looking at C++ documentation online
- A brief discussion of programming assignment 1

The code for this week is available at
www.cs.utexas.edu/~piyush/kourses/spr12/code/intro_to_cpp.tar.gz

Review - Libraries

- Before running a program you need to compile and link it to form an executable
- Libraries are compiled object files which do not contain a main function - they are instead used by other libraries or executables
- Libraries help in code reuse and can separate code into different components in an intelligent manner
- When using libraries, you typically include just the header into your code. The header only contains the declarations but not the definitions (implementations). The library or implementation gets linked to your executable after compilation
- There are a number of system libraries that you can reuse

Static vs dynamic linking

- What we have seen till now is static linking. In static linking, the library becomes part of your executable.
  - For static linking you generate static libraries at compile time - these libraries typically have the .a extension as we have seen
- Alternatively, you can compile your libraries as shared libraries.
  - These libraries are compiled slightly differently and have the .so extension
  - The library gets linked dynamically at run-time
Advantages of dynamic linking

- Only one copy of a shared library is in memory
  - This reduces memory swaps and helps speed up processing on your machine - especially if all your processes are using the same common libraries

- You can change the implementation of a shared library without needing to recompile your code
  - This helps provide after market support to dlls once they are released


Advantages of static linking

- A statically linked executable is self contained - does not cause a problem if the dynamic library is not present

- Statically linked executables are expected to have slightly faster loading times

- An excellent discussion on static vs dynamic linking is available at

Review - Building a static library (example3)

```
Bicycle.cpp
BicycleTest.cpp

On the command line:
g++ -fPIC -o Bicycle.o -c Bicycle.cpp
ar rcs libbicycle.a Bicycle.o

g++ -o BicycleTest BicycleTest.cpp -L. -lbicycle

To execute (inside the example3 directory):
export LD_LIBRARY_PATH=.

Cleanup:
r/ BicycleTest libbicycle.a Bicycle.o
```

A good discussion of creating static libraries is available at:
[http://stackoverflow.com/questions/5947067/how-to-create-a-static-library-with-g]

Building a dynamic library (example3)

```
On the command line:
g++ -o Bicycle.o -c Bicycle.cpp
ar rcs libbicycle.a Bicycle.o
g++ -o BicycleTest BicycleTest.cpp -L. -lbicycle

To execute (inside the example3 directory):
export LD_LIBRARY_PATH=.

Cleanup:
r/ BicycleTest libbicycle.a Bicycle.o
```

- The fPIC flag is needed to create position independent code (not needed for x86?)
- LD_LIBRARY_PATH is an environment variable which tells where to look for dynamic libraries. In practice, never use relative paths as we have in the example above

A good discussion of creating both static and shared libraries is
[http://www.adp-gmbh.ch/cpp/gcc/create_lib.html]
Compilation (vs Java)

- In Java:
  - javac blah.java creates blah.class
  - javac recursively finds all the required classes for compiling
  - blah.java and compiles them as well

- In C++:
  - g++ blah.cc creates blah.o
  - g++ will not find all required files recursively! It is necessary to
  - specify all the files during compilation
  - For this reason, the C++ commands in the previous example
  - were highly specific.
  - To save time C++ makes heavy use of build systems, such as
  - Makefiles or CMake

A simple Makefile (example 4)

This Makefile builds a static library and links the executable against it:

```
all: BicycleTest
BicycleTest: libbicycle.a Bicycle.h BicycleTest.cpp
  g++ -o BicycleTest BicycleTest.cpp -L. -libicycle
libbicycle.a: Bicycle.o
  ar rcs libbicycle.a Bicycle.o
Bicycle2.o: Bicycle.h Bicycle.cpp
  g++ -o Bicycle.o -c Bicycle.cpp
clean:
  rm -f BicycleTest libbicycle.a Bicycle.o
```

Makefiles

- Makefiles typically comprise of multiple statements of the
  form:

  - `target: dependencies`
  - `<tab> system command`

- Makefiles are useful because you do not need to re-compile
  your entire project when you change a particular file
- You can also use variables in makefiles which makes editing
  them a lot simpler
- There are configuration tools such as `autoconf` that
  automatically generate makefiles. Instead of make, you can
  also use alternative tools such as `cmake` (used in ROS)

Now with variables (example 5)

This Makefile builds an executable directly without building a library:

```
CC=g++
CFLAGS=-c -Wall
LDFLAGS=
SOURCES=Bicycle.cpp BicycleTest.cpp
OBJECTS=$(SOURCES:.cpp=.o)
EXECUTABLE=BicycleTest

all: $(SOURCES) $(EXECUTABLE)
$(EXECUTABLE): $(OBJECTS)
  $(CC) $(LDFLAGS) $(OBJECTS) -o $@

.cpp.o:
  $(CC) $(CFLAGS) $(SOURCES:.cpp=.o) -c -o $@

clean:
  rm -f $(OBJECTS) $(EXECUTABLE)
```

[http://mrbook.org/tutorials/make/](http://mrbook.org/tutorials/make/)
Looking up documentation online

There is a wealth of information available online. There are a number of ways you can approach this:

- If you know the name of the function you wish to search -
  - You can search for it on cplusplus.com
  - Lets quickly take a look at the function *floor*

- A number of people have probably asked the same question you might have.
  - For instance "how to compile static library with g++"
  - I have found stackoverflow to typically be a good site for answers

---

Review

- Today we saw how to create a dynamic library and saw some of the differences with a static library
- We saw more about the compilation process and how to write simple makefiles to speed up the process of compiling your program
- We saw a couple of examples of looking up documentation online

---

Programming Assignment 1

- It is now time to work on Programming Assignment 1. Lets take another look at it.

- Links to C++ Tutorials
  - [http://www.learncpp.com](http://www.learncpp.com)