CS 378 - Autonomous Vehicles in Traffic I
Week 3a - Working with C++ I

Logistics

• PRC visit on Friday
  ◦ http://z.cs.utexas.edu/users/piyushk/marvin/index.php/PRC

• Tentative meeting with regional recruiter for the CIA next Monday at 2PM
  ◦ I believe they want to interact with students and talk about internships and co-op programs at the CIA
  ◦ It might be interesting. If you wish to attend let me know. The meeting time has yet to be finalized

Today

• Basics of writing and executing a program
• A small example to demonstrate the differences between C++ and Java
• Compilation and linking
• Libraries

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

Learning C++

• For this course you will need to get comfortable with enough C++ programming to
  ◦ be able to write ROS nodes with ease on your own
  ◦ integrate your code with existing libraries and code
  ◦ be able to read online documentation on existing code
  ◦ be able to look into code when this documentation is not available

• We will help you get a grasp on these with suitable problems in the programming assignments
The basic idea

1. Write the code for your program
2. Compilation - Generate machine understandable code
   - Different languages have different compilers. We'll be using GNU C++. For Java you typically use javac.
   - If there are errors during compilation, fix your code and recompile
3. Execution - Execute your code in an environment
   - If there are any problems, fix your code, recompile and rerun

Hello World! in Java and C++

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

```cpp
#include <iostream.h>
int main(int argc, char** argv) {
    std::cout << "Hello World!\n";
    return 0;
}
```

Compile and Run:
g++ HelloWorld.cpp -o HelloWorld
java HelloWorld

Allocate memory and execute the code
```
#include <iostream.h>
int main(int argc, char** argv) {
    std::cout << "Hello World!\n";
    return 0;
}
```

Class example in Java (example1)

**Bicycle.java**

```java
public class Bicycle {
    // the Bicycle class has two fields
    private int gear;
    private int speed;
    // the Bicycle class has one constructor
    public Bicycle(int startSpeed, int startGear) {
        gear = startGear;
        speed = startSpeed;
    }
    public void setGear(int newValue) {
        gear = newValue;
    }
    public void applyBrake(int decrement) {
        speed -= decrement;
    }
    public void speedUp(int increment) {
        speed += increment;
    }
    public void printStatus() {
        System.out.println("Speed: ", speed + ", Gear: ", gear);
    }
}
```

**BicycleTest.java**

```java
public class BicycleTest {
    public static void main(String[] args) {
        Bicycle bicycle = new Bicycle(10, 4);
        bicycle.printStatus();
        bicycle.speedUp(5);
        bicycle.setGear(7);
        bicycle.printStatus();
        bicycle.applyBrake(10);
        bicycle.setGear(3);
        bicycle.printStatus();
    }
}
```

Output:
```
Speed: 10, Gear: 4
Speed: 15, Gear: 6
Speed: 5, Gear: 3
```
Now in C++ (example2)

Bicycle.h
#include <iostream>
class Bicycle {
private:
  int gear;
  int speed;
public:
  Bicycle(int startSpeed, int startGear) {
    gear = startGear; speed = startSpeed;
  }
  void setGear(int newValue) {
    gear = newValue;
  }
  void applyBrake(int decrement) {
    speed -= decrement;
  }
  void speedUp(int increment) {
    speed += increment;
  }
  void printStatus() {
    std::cout << "Speed: " << speed << ", Gear: " << gear << std::endl;
  }
};

BicycleTest.cpp
#include "Bicycle.h"
int main(int argc, char** argv) {
  Bicycle bicycle(10, 4);
  bicycle.printStatus();
  bicycle.speedUp(5);
  bicycle.setGear(6);
  bicycle.printStatus();
  bicycle.applyBrake(10);
  bicycle.printStatus();
  return 0;
}

Output:
Speed: 10, Gear: 4
Speed: 15, Gear: 6
Speed: 5, Gear: 3

Main differences in bicycle example

- The *semicolon* at the end of the class is very important
- Define public or private by section and not at each method or variable
- In C++ it is not necessary to name a file based on the class it contains, but it's good coding practice to do so anyway.
- Including header files is similar to importing packages and classes in Java
- It is necessary in C++ to include all the files yourself
  o `<...>` are headers which are defined globally or from a known include directory
  o "..." are headers defined from the current path
- For more 'Moving from Java to C++’ stuff, check out: http://horstmann.com/ccj2/ccjapp3.html

Some C++ Tutorials

- If you wish to learn more C++, there are some excellent tutorials at:
  o http://cplusplus.com/doc/tutorial/
  o I will recommend going through at least the tutorials from "Structure of a program" to Classes (I)
  o Working on these tutorials may prove necessary for programming assignment 1 if you are not familiar with C++
  o We can assist you during OH if you face any problems with these tutorials
- Jack also suggested these tutorials
  o http://www.learncpp.com/

The need for libraries

- In the example we just saw, we had 2 separate files in the C++ example
- There are many advantages of splitting your code in multiple files.
  o You typically do not want to have 1000s of lines of code in 1 file - *its a mess!*
  o **Splitting up your code can help in code reuse.** If someone else wants to write a program using the Bicycle class, they can by including Bicycle.h.
- There is a better way of splitting up code. Enter Libraries!
  o Libraries are pieces of code that cannot be executed and can be used by many executables and other libraries
Compilation and Linking

Whenever you compile code, you create machine readable files. In C++ while compiling a section of your code a number of such intermediate object files can be created. These files are similar to the .class files created in Java.

The code (example3)

Bicycle.cpp

```cpp
#include "Bicycle.h"

class Bicycle {

private:
  int gear;
  int speed;

public:
  Bicycle(int startSpeed, int startGear) {
    gear = startGear;
    speed = startSpeed;
  }

  void setGear(int newValue) {
    gear = newValue;
  }

  void applyBrake(int decrement) {
    speed -= decrement;
  }

  void speedUp(int increment) {
    speed += increment;
  }

  void printStatus() {
    std::cout << "Speed: " << speed << "", Gear: " << gear << std::endl;
  }
};
```

Building an executable directly (example3)

Without intermediate object files:
g++ -o BicycleTest Bicycle.cpp \
BicycleTest.cpp

Cleanup:
rm -f BicycleTest

With intermediate object files:
g++ -o BicycleTest Bicycle.o BicycleTest.cpp

Cleanup:
rm BicycleTest Bicycle.o BicycleTest.o
Building through a library (example3)

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

Cleanup:
rm -f 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

Library example - cmath

- *cmath* is a c++ library that provides a set of functions to compute common mathematical operations and transformations
- The library provides functions such as *sin*, *cos*, *sqrt*, *log* and *floor* among others.
- You can use it in your program by using the directive
#include `<cmath>` in your program
- The library header typically only includes the function definitions but not the implementations.
- On cplusplus.com, you can find documentation on most major system libraries. For instance:

Review

- 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

On Wednesday

- *Static vs dynamic* linking
- Makefiles
- Reading C++ documentation online
- Programming assignment 1 discussion