CS378 - Autonomous Vehicles in Traffic  
Spring 2012

Programming Assignment 1 (10 Points)  
Due Date: Wednesday 2/8/12 11:59 PM

Instructions
- It is important that you work on this assignment independently as it will serve as a building block for the rest of the course.
- Ask the teaching staff first if you have any problems. It is OK to discuss the assignment with your friends and classmates as long as you only discuss programming concepts.
- **Do not copy code from them!**
- Section I can be done in C++ or Python. Section II can only be done in C++.
- The standard late policy as detailed in the syllabus will be followed.
- Remember to follow the submission instructions with each section
- There are **3 pages** in this assignment

Section I - Fibonacci (5 points)
The purpose of this section is to test some simple programming skills. Implement a small program to compute the \( n \)th number in the Fibonacci sequence. The number \( n \) has to be provided by the user at the command line. The Fibonacci sequence is defined using the recursive formula \( f(n) = f(n-1) + f(n-2) \)

The first few numbers of the Fibonacci sequence are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,... More information is available on the wikipedia page:
http://en.wikipedia.org/wiki/Fibonacci_number

Below are a number of references you might have to use to complete this section. These might be especially useful if you are unfamiliar with C++
http://cplusplus.com/doc/tutorial/basic_io/  
http://cplusplus.com/doc/tutorial/functions/  
http://cplusplus.com/doc/tutorial/functions2/ (Check out the section on recursive functions)

Section I - Extra Credit (1 points)
Simply coding up the formula above leads to a very inefficient solution. Can you think of why? Can you code up a more efficient solution to this problem?

Section I - Submission Instructions
- Place all your code in a single directory. Lets assume this directory is called sec1
- Compress the directory into a single file by issuing the following command in the parent directory of sec1 using the command `tar czvf sec1.tar.gz sec1`
- We'll use the unix command line `turnin` for submission. Submit this section using the command `turnin --submit todd programming1 sec1.tar.gz`
Section II - Temperature Control (5 points)

A) (3 points) In this section you will need to implement 2 classes, HeatingUnit and BangBangControl. Remember to implement the classes in separate files as discussed in class. A Bang-bang control is one of the simplest controllers out there, and you will probably have one in your home. You can read up on Bang-bang control here: http://en.wikipedia.org/wiki/Bang%E2%80%93bang_control

The HeatingUnit class should expose 4 functions
constructor - Takes in two arguments. The first argument tells whether the heating unit is on or off initially. The second argument provides the initial temperature.
turnOn() - The heating unit is now on. Every time tick() is called the temperature increases by 1. The temperature does not go above 100.
turnOff() - The heating unit is now off. Every time tick() is called the temperature decreases by 1. The temperature does not go below 0.
tick() - increases or decreases the temperature depending on whether the heating unit is on or off initially. tick() also returns the temperature after changing the value.

The BangBangControl should have an object of the HeatingUnit class and should expose the following 4 functions:
constructor - takes 3 initial parameters: the temperature to maintain, whether the heating unit is on or off, and the initial temperature. The last 2 parameters are passed to the constructor of the HeatingUnit class
setTemp() - set the temperature to maintain
getTemp() - get the temperature that is currently being maintained
update() - update first calls the tick() function of the HeatingUnit object. It then uses the following check to figure out whether the heating unit needs to be turned on or off.
if (current_temp > temp_to_maintain + 2) heating_unit.turnOff()
if (current_temp < temp_to_maintain - 2) heating_unit.turnOn()

Also write a simple main() function to test your code

B) (2 points) Write 3 different Makefiles as discussed in class and using the help of examples provided. The 3 Makefiles should implement each of the following:
1. Compile both classes and your main code directly into 1 binary (executable).
2. Compile the 2 classes into separate static libraries and then link it to the binary.
3. Compile the 2 classes into separate dynamic libraries and then link it to the binary.

Section II - Extra Credit (1 Point)
Re-write the first Makefile from above using variables so that none of the source files or objects are in the actual dependency list (as the example in class)

Section II - Submission Instructions
- Once you write your code, you should have 5 files (2 headers, 2 class implementation files and 1 main function files). Place all of them in a single directory.
- Rename your makefiles as Makefile_Executable, Makefile_Static and Makefile_Dynamic and place them in the above directory. Ensure that you use these names as it is, as we
might use an automated script to test your code.

- Compress this directory using the same instructions as Section I. Use the following command in the parent directory tar czvf sec2.tar.gz sec2/
- You can submit this section by using the command turnin --submit todd programming1 sec2.tar.gz
- You can also submit multiple files at the same time using the command turnin --submit todd programming1 sec1.tar.gz sec2.tar.gz

To know more about tar and turnin, look up the manual for these commands. Type in man tar or man turnin for further documentation.