CSCI-1200 Computer Science II — Spring 2006
Homework 1 — Getting Started

This two-part homework explores the background and review material covered in Lectures 1 and 2. Part 1 is a set of short problems, similar to problems that might appear on a test. Part 2 is a short programming problem. These problems are not particularly difficult. Students who feel uncomfortable about their programming background should view this as an opportunity to gauge their current skills. Programming assignments will become more challenging soon, so if you struggle with this one, you should work hard to practice and catch up. *Before starting this assignment, make sure you have read and understood the statement of Academic Integrity for Programming Assignments.*

Part 1: Short Answer

While it is clearly possible for you to create a program file containing the code below and then compile and execute the program, you will not have the luxury of doing this on a test. You should practice by studying the code carefully yourself first and trying to determine the answers. This is a good skill to develop.

In your main CSII folder on your laptop, create a subfolder for homeworks and within that folder create a subfolder `hw1` for this assignment. Create and open a *plain text* file named `part1.txt` and write your solutions for Part 1 in this file.

1. What is the output of the following program, which is supposed to compute the average, max and min of an array of numbers? Explain why this output occurs and give a small set of changes to the code to fix it. Do not fix the code by changing its structure; in other words there must be a `for` loop and a `while` loop in the `stats_calculation` function, and the `while` loop must count backwards.

```
#include <iostream>
using namespace std;

void stats_calculation(float values[], int &n, float &max_value,
                       float min_value, float &avg) {
  //  Find the average value
  float sum = values[0];
  for (int i=0; i<n; ++i)
    sum += values[n];
  avg /= n;

  // Find the maximum and minimum values.
  min_value = values[n-1], max_value = values[n-1];
  while (n > 0) {
    --n;
    if (values[n] > max_value) max_value = values[n];
    else if (values[n] < min_value) min_value = values[n];
  }
}

int main() {
  // Initialize an array in order to test the function. The size of
  // the array (6) is automatically determined from the number of
  // values in the list.
  float a[] = { 12.3, 15.4, 1.5, 7.8, 2.3, 8.9 };
  int size = 6;
  float min_value = 0, max_value = 0, average = 0;

  stats_calculation(a, size, max_value, min_value, average);
  cout << "Here are the values: ";
  for (int i=0; i<size; ++i) cout << a[i] << " ";
  cout << '\n';
  cout << "average = " << average << '\n'
       << "max = " << max_value << '\n'
       << "min = " << min_value << '\n';
  return 0;
}
```

2. What is the output of the following program?

```
#include <iostream>
using namespace std;

int main() {
  int x=5, y=4;
  float a[3] = { 1.0, 2.0, 3.0 };

  if (x > y) {
      float y = 1.0;
      int a = 6;
      cout << "x = " << x << ", y = " << y << ", a = " << a << endl;
      x = 2;
    }

  cout << "x = " << x << ", y = " << y << ", a[0] = " << a[0] << endl;
  return 0;
}
```

3. For each of the two functions in the following code, give an order notation count of the number of operations it requires. Justify your answer briefly. For extra credit, rewrite the second function so that it has a better order notation estimate, and give this estimate.

```
// Count the number of entries in an array that have both an even
// index (subscript) and an even value.
int count_evens( int arr[], int n ) {
  int count = 0;
  for (unsigned int i=0; i<n; i+=2)
    if (arr[i] % 2 == 0) ++count;
  return count;
}

// Given an array of floats, form an array of sums.  Entry 0
// in the array of sums is the addition of all values in the
// array.  Entry 1 is the addition of values in locations
// 1..n-1.  In general, entry i is the addition of the values
// in locations i..n-1.
void subsequence_sum( float arr[], int n, float sums[] ) {
  for (int i = 0; i < n; ++i)  {
      sums[i] = 0.0;
      for ( int j=i; j<n; ++j )
        sums[i] += arr[j];
    }
}
```

Part 2: Circle Intersections

Write a program that reads in a set of circles and determines if any of the circles intersect. This is an example of the kinds of problems that must be solved in graphics and game software. Put the code in a file named `circles.cpp`. The input to the program, via `std::cin`, will be an integer giving the number of circles (at most 100) and then a list of circles, one per line of input. Each circle will be specified by 3 floats giving the x and y values of the center of the circle and the radius of the circle. The output of the program, via `std::cout`, will be all pairs of intersecting circles, with the count of the number of intersections at the end. When you output an intersection, you need only specify the indices of the two intersecting circles. The first circle has index 0.

Here is an example of the input:

```
5
1 2 5
-2.5 3 8
7 9.1 3
15.4 4 2
0.5 0.5 13
```

And here is the correct output for this test case:

```
0 intersects 1
0 intersects 4
1 intersects 4
2 intersects 4
Number of intersections = 4
```

You should make the format of your output **exactly** this way to aid the TAs in grading and ensure you receive full credit. We have provided you sample input and output files so you may check your work. See the Resources page on the course website to learn how to redirect input & output streams to files in your development environment. Make up your own test cases too!

You will need three arrays to represent the circles, one for the x coordinates, one for the y coordinates and one for the radii. Soon we will see a much better way to represent this data. You should start by writing and testing a simple function that determines if two circles intersect; i.e., if any part of the region enclosed in one circle is contained in the other. We do not give you the method for doing this — you should figure it out for yourself. This function should take 6 parameters: the center x and y and the radius of each circle. You may (or may not) find the function `sqrt` useful. You get access to this function by including `cmath` at the start of your program, just as we did in Lab 1:

```
#include <cmath>
```

When you've finished writing, testing, debugging, and commenting your code, prepare your assignment for submission. If you're using cygwin, linux, or freebsd, go to the top level directory for your homeworks and type:

```
tar -cvzf hw1_submit.zip hw1
```

This will pack together and compress your `hw1` directory and all of the files it contains. On Windows, use WinZip to create a new archive named `hw1_submit.zip` and then drag & drop your entire `hw1` folder into the archive. Then, go to the course website, click on the link to "Submit Homework" and follow the instructions.

Please ask a TA if you need help preparing your assignment for submission. Do not submit any other type of compressed file format or you will not receive full credit.