Problems for Op 2010

By Eric Durant, PhD <durant@msoe.edu>
Friday 19 November 2010
Copyright © 2010 MSOE

1. Triangle side length (10 Points)
Given a right triangle with hypotenuse \( c \) and remaining sides \( a \) and \( b \), recall that \( a = \sqrt{c^2 - b^2} \). Write a program that asks the user for \( b \) and \( c \), checks that they describe a valid triangle (\( c > b > 0 \)) and reports the value of \( a \) if they do or an error message if they do not.

2. Triple character detector (10 Points)
Write a function that accepts a string and returns a boolean (bool in C++). The value should be true if the string contains a triple character (e.g., “ooo” is a triple character, but “qqxq” is not and “XxX” is not because the letters differ in case) and false otherwise. Include a main() that asks the user for a string, tests it with the function, and outputs the returned value.

3. Fibonacci series (10 Points)
The Fibonacci series is defined such that \( F_0 = 0, F_1 = 1 \), and \( F_n = F_{n-1} + F_{n-2} \). Write a program that asks the user for \( n \) and then outputs \( F_0 \) through \( F_n \). You may assume that the user enters \( n \geq 2 \).
4. Thanksgiving (20 Points)

Write a program that asks the user to enter the 4-digit year and then calculates the day on which Thanksgiving falls in the US, which is the 4th Thursday in November. Your program must output correct values for the years 1942 through 2037.

Hint: you can calculate the weekday as a function of the year as follows:

Java

```java
import java.util.Calendar;
import java.util.GregorianCalendar;
...
Calendar cal = new GregorianCalendar(nYear, 10, 1); // year, month (January is 0),
day
int dow = cal.get(Calendar.DAY_OF_WEEK); // Sunday is 1, Saturday is 7
```

C++

```c++
#include <time.h> // localtime

// Generate a valid time struct
time_t tret;
time(&tret);
struct tm tm;
struct tm _localtime64_s(&tm, &tret);

// Calculate the day of the week for November 1st in the given year...
tm.tm_mon = 10; // November - 0-based
tm.tm_mday = 1; // 1st of month
int nYear = nYear - 1900; // 1995 becomes 95, 2010 becomes 110, etc.
mktime(&tm);
unsigned int nWeekday = tm.tm_wday;
// Now nWeekday contains a number 0-6, where 0 is Sun, 1 is Mon, etc.
```

5. Cab dispatching (20 Points)

You are given two data files (samples are available on the contest website). Cabs.txt contains the location of available cabs at the beginning of the day. Each cab has a sequential ID number beginning with 0. Passengers.txt contains the location of passengers in the order that they call for a cab. The 1st line of each file contains the number of cabs or passengers defined in that file. Each passenger has a sequential name beginning with ‘A’; there are at most 26 passengers. For the purposes of this problem, people take very long cab rides; once a cab is used, it is never available again. Write a program that, given files containing cab locations and passengers, assigns the closest cab to each passenger and dispatches it as the calls come in; your program will output a list of cabs and passengers (for example, “Cab 3 dispatched to passenger N.”). If there are more passengers than cabs, stop after all cabs are dispatched. Since the cabs need to travel streets that form a grid pattern with only right angles, “distance” between two points is defined as the distance in the x direction plus the distance in the y direction (no square roots or trigonometry are involved).
6. Area of polygon (20 Points)
Calculate and display the area of a user-specified polygon. Your program should first prompt the user for \( n \), the number of vertices, and then for the \( x \) and \( y \) coordinate of each vertex from 0 through \( n-1 \). The "surveyor's formula" gives a surprisingly simple way to calculate the area of a polygon by summing up the differences of the products of adjacent \( x \) and \( y \) coordinates: \( A = \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i) \). Note that the 0\(^{\text{th}} \) vertex must be reused as the \( n^{\text{th}} \) vertex: \((x_n, y_n) = (x_0, y_0)\).

7. Tetris calculator (40 Points)
Write a function that determines whether a piece in the classic game of Tetris can be moved down (true) or is blocked from below (false). Also write a program to test your function by reading in a 2-D array (the exact data structures are left up to you) representing a Tetris board from a file, passing it to your function, and then displaying the result. The board data is always the same size, 22 rows high by 10 columns wide; see the website for an example data file of the format that your program must support. Each line of the file contains 10 numbers representing the contents of a row, starting with the top row. '0' represents an empty spot, '1' represents a previously filled spot, and '2' represents a spot filled by the piece in motion. If a piece is not at the bottom of the board and every spot immediately below it contains a 0, it can move down; otherwise it cannot.
8. Trapezoidal integration (40 Points)

The area under a curve can be estimated by dividing the region under the curve into a sufficiently large number of narrow trapezoids and adding up the area of each of the trapezoids. This technique is shown in the figure¹.

The area of the first trapezoid is: \( \frac{h}{2}[f(a) + f(a+h)] \)

This formula generalizes to the following:

\[
\text{Area} = \frac{h}{2}[f(a) + 2f(a+h) + 2f(a+2h) + \ldots + 2f(a+(n-1)h) + f(a+nh)] ; \text{where } h = \frac{b-a}{n}
\]

Write a program for estimating the area under a curve. Prompt the user for \( a, b, \) and \( n \) (the number of trapezoids to use).

Have your solution use the following function whenever it needs a value for \( f(x) \).²

**Java**

```java
import java.lang.Math;
public static double f(double x) {
    return Math.exp(-x*x/2)/Math.sqrt(2*Math.PI);
}
```

**C++**

```cpp
double f(double x) {
    const double invrt2pi = pow(atan(1.0)*8.0, -2);
    return exp(-x*x/2) * invrt2pi;
}
```

Hint: the area under this curve from -1 to 2 is approximately 0.8185. You can test your program by adding more trapezoids and noting that your answers converge to the indicated area.

---

¹ Image courtesy of http://metric.ma.ic.ac.uk/integration/techniques/definite/numerical-methods/trapezoidal-rule/

² This is the probability density function for the normal distribution with mean 0 and standard deviation 1.
9. US congressional apportionment (40 Points)

Write a program that calculates and displays an apportionment of a fixed number of seats among groups of people competing for them using the same method that is used in the US House of Representatives:

\[
\text{Begin by allocating one seat to each competing group}
\]
\[
\text{While there are seats remaining to be assigned}
\]
\[
\quad \text{Assign the next seat to the "neediest" group (assume there are no ties)}
\]

The "need," \(n_i\) of group \(i\) for a seat is defined as the population in the group, \(p_i\), divided by the geometric mean of how many seats they have, \(s_i\), and how many they would have if they got the next seat. Mathematically, \(n_i = \frac{p_i}{\sqrt{\frac{1}{s_i} + \frac{1}{s_i}}}\). Prompt the user to type in the number of seats, followed by the number of groups, followed by the size of each group.