Test 2: CPS 006X

Owen Astrachan       Susan Rodger

November 13, 2003

Name: ________________________________ (1 point)

Login: ________________

Honor code acknowledgment (signature) __________________________________

<table>
<thead>
<tr>
<th></th>
<th>value</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15 pts.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>30 pts.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>15 pts.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TOTAL:</td>
<td>72 pts.</td>
</tr>
</tbody>
</table>

This test has 11 pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes.

In writing code you do not need to worry about specifying the proper #include header files. Assume that all the header files we’ve discussed are included in any code you write.
PROBLEM 1:  *(You don’t need to have a point to have a point (15 points))*

The struct/class Point is declared in point.h and is reproduced at the end of the test.  
As an example of using this class, the code below prints 5, the distance from the origin (0,0) to point (3,4).

```cpp
Point origin(0,0);
Point p(3,4);

cout << p.distanceFrom(origin) << endl;
```

**Part A (5 points)**

Write the function `createPoints` whose header is given below. The function creates a vector of points from vectors of x- and y-coordinates. For example, the call `createPoints(x,y,v)` where x is (-4,-2,5,8) and y is (-2,3,8,4), should create in vector v the points (-4,-2), (-2,3), (5,8), (8,4) — see the diagram on the next page for the points.  
Complete the function `createPoints` below.

```cpp
void createPoints(const tvector<double>& x, 
                  const tvector<double>& y, 
                  tvector<Point>& points)
// pre: x.size() == y.size()
// post: points.size() == x.size() and points[k] represents
//       a point with x-coordinate x[k] and y-coordinate y[k]
{ 
    
}
```
Part B (10 points)

Write the function closest that returns the distance between the two closest points in a vector of points. For example, if the vector contains the points in the diagram above the function should return 5.0 since the closest points are those with coordinates (5,8) and (8,4) and the distance between these points is 5.0.

double closest(vector<Point> & p)
// pre: 1 < p.size()
// post: returns distance between closest points in p
{
}
PROBLEM 2: *Lottery (30 points)*

You’re writing code to process numbers chosen in a lottery. A lottery player chooses six different numbers between 1 and 52, these six numbers together are called a *ticket*. The total Lottery winnings/payout are based on how many tickets share numbers with the winning ticket, and on how many numbers the tickets share.

Tickets (6 different numbers between 1 and 52) are sent over secure Internet channels as *ticket strings*. Player winnings are based on how many numbers a player’s ticket has in common with the winning ticket. The int values in a ticket string are separated by one space, there are no leading or trailing spaces in a ticket string.

For example, if the winning ticket is

"3 22 17 49 8 35"

Then the table below shows how many numbers in common several tickets have with the winning ticket.

<table>
<thead>
<tr>
<th>ticket</th>
<th>count</th>
<th>common numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;22 8 19 51 49 7&quot;</td>
<td>3</td>
<td>8, 22, 49</td>
</tr>
<tr>
<td>&quot;15 20 25 35 40 45&quot;</td>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>&quot;35 8 17 22 47 48&quot;</td>
<td>4</td>
<td>8, 17, 22, 35</td>
</tr>
</tbody>
</table>

Part A (5 points)

Write the function `stringToVector` that returns a vector of the int values represented in a lottery ticket string. For example, the call `stringToVector("3 22 17 49 8 35")` should return a vector of six ints, (3, 22, 17, 49, 8, 35).

```cpp
tvector<int> stringToVector(const string& s)
// pre: s contains int values separated by whitespace (one space)
// post: returns vector of int values in s
{
```

}
**Part B (8 points)**

In writing the function `commonCount` in this problem you may call the function `contains` below. You may also call the function `stringToVector` you wrote in Part A – assume `stringToVector` works as specified regardless of what you wrote in Part A.

```cpp
def contains(const tvector<int>& v, int value)
    // post: returns true iff value contained in v
    for(int k=0; k < v.size(); k++){
        if (v[k] == value) return true;
    }
    return false;
```

Write the function `commonCount` that returns a count of how many numbers two lottery tickets have in common. The table on the previous page provides examples of values returned for several tickets when compared to a winning ticket. For example, the call

```cpp
commonCount(“22 8 19 51 49 7”, ”3 22 17 49 8 35”)
```

should evaluate to/return 3 as shown in that table.

```cpp
int commonCount(const string& a, const string & b)
    // pre: a and b both represent lottery tickets (contain ints separated by one space)
    // post: return count of how many numbers a and b have in common
{
```
**Part C (7 points)**

The lottery payout is determined by how many lottery tickets share 4, 5, and 6 numbers in common with the winning ticket. A ticket that shares 6 numbers in common pays $100,000; a ticket that shares 5 numbers in common pays $20,000; and a ticket that shares 4 numbers in common pays $1,000. A ticket that has six numbers in the same order as the winning ticket pays $2,000,000 (two million) dollars (but not the $100,000 for a ticket with 6 in common that aren’t in the same order).

Write the function `payout` that computes a lottery payout from a vector of all chosen lottery tickets and the winning ticket. In writing `payout` you may call call functions written or described in Parts A and B. Assume they work as specified regardless of what you wrote.

```cpp
double payout(const tvector<string>& tickets,
              const string& winner)
// pre: all strings in tickets represent lottery picks (each string is
//       six white-space separated int values)
// post: returns the lottery payout
{
```
**Part D (10 points)**
Lottery clients/players can pick their own numbers for a ticket or have the computer generate a random ticket. Write the function `randomTicket` that returns a string that is a randomly-generated lottery ticket. The string should contain six int values separated by one space with no duplicate values; each int value (between 1 and 52, inclusive) in the string should have an equal likelihood of being chosen.

In writing `randomTicket` you may call call functions written or described in Parts A, B, and C. Assume they work as specified regardless of what you wrote.

```
string randomTicket()
// post: returns string representing a lottery ticket
{
```
PROBLEM 3:  (*X-country Scoring (15 points)*)

In cross country running meets a team score is based on where its runners place (first, second, last), but the a runner’s time can be used to break ties.

For example, consider running data stored in the format shown below, where each line stores the time a runner crossed the finish line, and the school of the runner. The first line of the file is for the first-place finisher, the second line of the file for the second-place finisher, and so on so that in general the \( n^{th} \) line of the file is for the \( n^{th} \) finisher. In the example below a UNC runner finishes first, but Duke runners finish second, third, fifth, and ninth.

```
16:58  UNC
17:52  Duke
17:57  Duke
18:03  Wake Forest
18:07  Duke
18:10  Wake Forest
18:12  UNC
18:25  William and Mary
18:27  Duke
18:37  William and Mary
18:39  Wake Forest
18:45  Wake Forest
18:59  UNC
19:01  UNC
19:02  William and Mary
19:05  William and Mary
```

Write the function `scoreTeam` whose input parameters specify a file of xcountry-running data and a school and that returns a `XCinfo` object with the school’s place-total and time-total (the `XCinfo` declaration is on the next page).

For example, in the file above the call shown below would store in `info.places` the value 19, in `info.times` the value 1:12:23, and would store in `info.school` the string "Duke".

```
XCinfo info("""
info = scoreTeam("xe.dat", "Duke")
```

Duke runners place 2,3,5,9, and \( 2 + 3 + 5 + 9 = 19 \); note that 17:52 + 17:57 + 18:07 + 18:27 = 1:12:23. Similarly, the call below will return 35 in `info.places` and 1:13:10 in `info.times` and "UNC" in `info.school`.

```
XCinfo info("""
info = scoreTeam("xe.dat", "UNC"
```

(function started on next page)
Complete scoreTeam below.

```cpp
struct XCinfo
{
    string school;   // school name
    int places;      // total of all places
    ClockTime times;  // total of all times

    XCinfo(const string& sch) 
        : school(sch), places(0), times(0,0,0)
    {
    }
}

XCinfo scoreTeam(const string& filename, 
                   const string& college, 
               // pre: filename can be opened for reading
               // post: returns scoring info for college (both place and time totals)
{
    ifstream input(filename.c_str());
    string line;

    while (getline(input,line))    // reading a line succeeded
    {
    }
}
```
PROBLEM 4:  (*StarLogo (12 points)*)

The following StarLogo code draws the outline of a square that is size 6 by 6. The outline drawn is blue with red corners as shown in the figure below.

to draw-square-outline
  setc blue  // turtle color is set to blue
  repeat 4
    [stamp red  // stamp patch to red
     fd 1
     pd  // pen down
     fd 5  // forward 5 and draw blue
     pu  // pen up
     rt 90]  // right turn 90
  end
  stamp red
end

For the following problems, assume a turtle is heading north (seth 0), the background color is black, and there are rectangles of other colors (but not blue) drawn in the window.

**Part A (6 points)**

Write the StarLogo procedure move-until-rectangle that moves a turtle along the black background until it reaches a patch of another color, thus stopping in front of a rectangle (not on the rectangle) as shown in the picture.

to move-until-rectangle
end
Part B (6 points)
Write the StarLogo procedure `draw-blue-stripe` that draws a blue stripe on a rectangle. Assume the turtle is facing north and is right below the rectangle (not on the rectangle). The turtle should move onto the rectangle in the same direction and draw a blue stripe on the rectangle as shown in the figure below. When `draw-blue-stripe` finishes the turtle should be one step past the rectangle as shown in the picture.