EECE 230 — Introduction to Programming Using C++, Sections 10,11 and 12 — Quiz 2

May 3, 2018

- The duration of this exam is 2 hours and 45 minutes. Keep in mind that you need around 10 minutes at the end of the duration of the exam to submit your answers. It is your responsibility to make sure your files are correctly submitted.
- The exam consists of 4 problems for a total possible of 155 points.
- You can use all the material available on "Volume D:" in the EECE230Files folder. In particular, the folder contains lecture slides, source code from the lectures, programming assignments, and their solutions.
- A directory "D:\EECE230Files\SubmitQuiz2" will be created on your machine and will be dedicated to collect your solutions. Make sure you copy your **cpp files** (files ending with the .cpp extension) to that directory. Failure to do so may lead to a failing grade on the exam.
- You are **NOT** allowed to use the **web**. You are not allowed to use **USB's** or files previously stored on your machine other than files in the EECE230Files directory.
- If you get caught violating the above rules or if you communicate with a person other than the exam proctors during the exam, you will immediately get zero and you will be referred to the appropriate disciplinary committee.
- Cell phones and any other unauthorized electronic devices are absolutely not allowed in the exam rooms. They should be turned off and put away.
- The problems are of varying difficulty. Below is rough ordering estimate of the problems in order of increasing difficulty.
  - Level 1 (42 points): Parts 1a-d, and Parts 4a-b
  - Level 2 (26 points): Parts 1e, 3a and 4.c
  - Level 3 (37 points): Problem 2, and Part 3b
  - Level 4 (30 points): Part 1f, Part 3.c
  - Level 5 (20 points): Part 4d.
- Detailed comments are worth partial credit.
- Plan your time wisely. Do not spend too much time on any one problem. Read through all of them first and attack them in the order that allows you to make the most progress.
- Submit your solutions each part in a separate file as indicated in the booklet. Include your name and ID number in each file.
- Good luck!

Problem 1 (45 points). Bookstore: class, constructor and arrays

This problem provides basic support for a bookstore inventory system.

a. **(5 points).** A book has several attributes such as a title, an author name, a publisher name, a version number, an ISBN number, and a year of publication. For brevity, develop a class `Book` which only supports title and author name.

b. **(5 points).** Provide a constructor that takes a booktitle and an author name and initializes an object of type `Book`.

c. **(5 points).** The bookstore keeps the price and the number of available copies for each book. For that you need to develop a class `BookItem` that supports price and number of copies. Implement `BookItem` using one of the following choices.

   - Class `BookItem` inherits from class `Book` and has two additional data members to express the price and the number of available copies of the book.
   - Class `BookItem` has a data member object of type `Book` and has two additional data members to express the price and the number of available copies of the book.

   Provide a constructor that takes a title, an author name, the price and the number of availble copies of the book and initializes an object of type `BookItem` accordingly.

d. **(7 points).** Class `BookStore` has an array (or a vector) of available book items. It also has another array (or a vector) that indicates the number of sold copies of each book. Develop class `BookStore` with both arrays (or vectors). Provide a constructor for the `BookStore` class that starts with no elements in both arrays (or vectors). Provide a method `void initialize()` that fills the arrays (or vector) of available books and sold books with your choice of books.

e. **(8 points).** Write method `print` in classes `BookStore` and `BookItem` to print the corresponding objects. The `print` in class `BookStore` should call the `print` in `BookItem` to print each book. The method should also print the total number of books and the total revenue so far.

f. **(15 points).** Provide a method `void sell(string title, string author)` in class `BookStore`. This method will be called frequently often as this is the main functionality of the book store. The method takes the title and the author information of a book, finds the book in the bookstore arrays (or vectors), and updates the bookstore arrays (vectors) accordingly. In particular, the count of available copies of the book should be decremented, the count of sold copies of the book should be incremented. If the book does not exist or no copies are available, an error should be reported. If the array (or vector) of sold books does not have a corresponding book item, a book item must be constructed and added and then the count of sold copies of that book should be updated. A correct solution is worth **10 points.** More efficient implementations are worth up to **5 additional points.**

Submit your solution is a file called `Prob1.cpp` including your name and ID number. You may also use Book.h, BookItem.h and BookStore.h to define your classes.

Problem 2 (25 points). Cars: virtual functions

The following code instantiates objects of type `BMW`, `VW`, and `Audi` and inserts them into a vector `cars` of pointers to objects of type `Car`. Then a loop goes over the `cars` vector and calls the `beep` method of each object in `cars` where method `beep` prints a different word for each car type.

```
BMW b1, b2, b3;
VW v1,v2;
Audi a1,a2;
vector<Car*> cars;
```

```
cars.push_back(&b1);
cars.push_back(&b2);
cars.push_back(&b3);
cars.push_back(&v1);
cars.push_back(&v2);
cars.push_back(&a1);
cars.push_back(&a2);
```

```
int i = 0;
while (i < cars.size()){
  cars[i]->beep();
  i = i+1;
}
```

Implement the classes `Car`, `BMW`, `VW`, and `Audi` and the `beep` method inside each of them using inheritance and virtual functions such that you get the following output.

```
Paap-paap Paap-paap Paap-paap Toot-toot Toot-toot Peep-peep Peep-peep
```

Submit your solution is a file called `Prob2.cpp` including your name and ID number. You may also use car.h, bmw.h, vw.h and audi.h to define your classes.

Problem 3 (35 bonus points). Recursion

a. **(8 points). Number of moves for towers of Hanoi.** The towers of Hanoi recursive function moves n disks from tower 1 to tower 3 using tower 2 as intermediate. It makes one disk move, and calls itself twice recursively for $n-1$ disks.

One can use that formulation to compute the number of single disk moves needed to move n disks using the towers of Hanoi recursion. The formula follows.

$$f(n) = 2 * f(n-1) + 1 \; for \; n > 0; \; f(0) = 0$$

Write a recursive function `numMoves` that takes the number of disks n and returns the number of moves required by the towers of Hanoi recursion to move the n disks.

Submit your solution is a file called `Prob3a.cpp` including your name and ID nunber.

b. **(12 points). Derangement.** A *derangement* of n integers from 0 to $n-1$ is a permutation p of the n integers such that $p[i] \neq i, 0 \leq i < n$. For example, when $n = 0$ or $n = 1$ we have zero derangements, when $n = 2$ we have only one derangement: 10, when $n = 3$ we have only two deragements: 201 and 210, when $n = 4$ we have nine derandements: $1032, 1230, 1302, 2031, 2301, 2310, 3012, 3201$, and 3210.

Write a recursive function to compute the number of derangements of size n using the following recurrence.

$$d(n) = (n-1)(d(n-1) + d(n-2)) \; for \; n > 2; \; d(2) = 1; \; d(1) = 0$$

The first few results should be $0, 1, 2, 9, 44, 265, 1854, 14833$, and 133496.

Submit your solution is a file called `Prob3b.cpp` including your name and ID nunber.

c. **(15 points). Interleaving two strings.** Given two strings s_1 and s_2 of size m and n, respectively, develop a *recursive* function `interleave(const string & s1, const string & s2, string & s, int m, int n, int idx1, int idx2, int idx)` that prints all the $\frac{(m+n)!}{m!n!}$ interleavings of s_1 and s_2. *Hint: figure out how large string s needs to be to hold one interleaving result before you start the recursion.* The function declaration above is given to help you and you may change it if you think you can do it in another way. Examples follow.

For s_1 =“ab” and s_2 =“1” the function prints the following.

```
ab1  1ab  a1b
```

For s_1 =“ab” and s_2 =“12” the function prints the following.

```
ab12  1ab2  a1b2  1a2b  a12b  12ab
```

Submit your solution is a file called `Prob3c.cpp` including your name and ID nunber.

Problem 4 (50 points). Minimum distance point

You are given a set A with n points where each point is given with its (x, y) coordinates. Your are also given a line L with its parameters a, b and c such that L satisfies formula $ax + by + c = 0$. The problem is to find a point p on L with coordinates (x_p, y_p) such that the sum of the distances between p and points in A is minimum. Proceed as follows.

a. (**10 points.**) Define a class `Point` with an adequate constructor to represent a point. The class should also support method `double dist(const Point & p2)` that computes and returns the distance between points $p1$ and $p2$ for the call `p1.dist(p2)`. Recall that the distance between two points $p1$ and $p2$ is given by the formula $\sqrt{(p1.x - p2.x)^2 + (p1.y - p2.y)^2}$.

b. (**10 points.**) Define a class `Line` with an adequate constructor to represent a line. The class should support a method `double getY(double x)` that returns the correponding y-coordinate for a given x such that point $(x, getY(x)) \in L$. Recall to use the formula of the line to compute y from x.

c. (**10 points.**) Write a function `computeSum(Point A[], int n, Point pl)` that computes the sum of distances from point pl to all n points in array A.

d. (**20 points.**) Write a function `computePointWithMinDist(Point A[], int n, Line L)` that computes the point $p \in L$ such that $-1,000,000.00 \leq p.x \leq 1,000,000.00$ and the sum of distances from p to all points in A is minimal. For simplicity, a solution within $\epsilon = 10^{-6}$ from p on the $x - axis$ is fine. *Hint: distance from p to a point in A decreases as p approaches the point, reaches a minimum, and then keeps increasing as p moves away from the point. So the sum of distances is not necessarily strictly increasing, however, you want to make use of the fact that it has a unique minimum in the range.*

Submit your solution in a file called Prob4.cpp including your name and ID nunber. You may also submit file `Point.h` and `Line.h` to define your classes.