COSC 1337 Programming Lab Exam 3 Fall 2018

This lab covers several topics from the course in 4 parts:

1. Chapters 1-9 (basic C++ with: functions, classes, arrays, search & sort) 25 points
2. Chapter 10 (pointers, dynamic memory allocation, de-allocation) 20 points
3. Chapter 11 (object oriented features, operator overloading, inheritance) 20 points
4. Chapter 14 (recursion) 20 points

Total: 85 points

Extra credit: 5 points

To save time, a startup file with startup and testing code is provided. From Blackboard, download the file: DDHH\_X3\_Lastname.cpp. It compiles and runs, but needs work. The file contains sections for each step where your new code is to be placed; see the comments.

**Part 1. [25 points]** In this part, you program for the “Dollar or More” discount store where each item sells for $1 or above. Create a class, make an array of the class, load it and display it.

**Part 1 steps:**

1. **[5 points]** Create a class called **Item** that contains information about a single item for sale at a store. Provide private data members for the **name** of the item and the **price**.
2. **[5 points]** Provide a display method that displays the name and price of the item. **Use $xx.xx format for price.** (Do not waste time creating unnecessary getters and setters.)
3. **[5 points]** Provide a constructor which has parameters that specify the name and price. In the constructor, validate the inputs – the name must not be empty (must have at least 1 letter), and the price must be $1 or more. If either the name and/or price provided are missing or invalid, set the name to “generic” and the price to 1.00. **Inside the constructor, do not prompt** the user for values or corrections to values – just use the parameters as provided.
4. **[5 points]** Create an array of Items called: **store** and load array **store** with items constructed from the data in the vector **warehouse.** The vector **warehouse** is provided. Elements in a vector can be accessed using regular array notation: “warehouse[i]”; individual fields can be recovered using dot notation: .name, .price.
5. **[5 points]** Provide a loop that displays the contents of array **store** to the console. To display individual items, use the display method you wrote in Step 1B above. This will display the items in array **store** that were originally in vector **warehouse.** Generic items will be marked as such. See below.

The left table shows the vector **warehouse** (provided). The right table is the contents of array **store** after it is loaded from the data in vector stuff. Two items (in **bold**) were discovered to be generic.

|  |  |  |  |
| --- | --- | --- | --- |
| **vector: warehouse** | | | |
| **name** | **Price** | **made\_in** | **weight** |
| hat | $8.00 | china | 6 |
| t-shirt | $15.00 | usa | 14 |
| coffee mug | $6.50 | canada | 22 |
| candy bar | $0.79 | mexico | 3 |
|  | $34.59 | japan | 122 |
| mixing bowl | $4.99 | brazil | 20 |

|  |  |
| --- | --- |
| **array: store** | |
| **name** | **price** |
| hat | $8.00 |
| t-shirt | $15.00 |
| coffee mug | $6.50 |
| **generic** | **$1.00** |
| **generic** | **$1.00** |
| mixing bowl | $4.99 |

**Part 2. [20 points]** For this part you will use pointers and dynamic memory allocation.

**Part 2 steps:**

1. **[5 points]** Dynamically allocate one instance of class Item. To construct this, use name “black umbrella” and price 19.95. Provide name and price to the constructor, because there are no setters.
2. **[5 points]** Display the address in memory where this Item was allocated. Provide output such as: **black umbrella is at address: 0x356480**
3. **[5 points]** Display the contents of the umbrella instance (its name and price) allocated in Step 2A using the display method provided for the item class in Step 1B.
4. **[5 points]** De-allocate the instance of class Item that you allocated in Step 2A above. After de-allocation, set the Item pointer to a safe value, so it can be determined later that it does not point to anything.

**Part 3. [20 points]** For this part you will overload operators. Provided is a class **Fraction**. You will add some new overloaded operator methods to class Fraction.

**Part 3 steps:**

1. **[5 points]** Implement operator++ (pre-increment) for class Fraction such that it **increments** the numerator (top) by one and leaves the denominator alone. Testing code is provided: Fraction a=4; ++a; ++a; cout<<a.get()<<endl; // displays 6
2. **[5 points]** Implement operator-- (pre-decrement) for class Fraction so it **decrements** the numerator (top) by one and leaves the denominator alone. Testing code is provided:  
   Fraction b=5; --b; --b; cout<<b.get()<<endl; // displays 3
3. **[5 points]** Implement operator\* for class Fraction. It multiplies two fractions together. It multiplies the numerator times the numerator and the denominator times the denominator. For example: 6/1 \* 7/1 = (6\*7)/(1\*1) = 42. Testing code is provided:  
   Fraction c=6, d=7, e=c\*d; cout<<e.get()<<endl; // displays 42.
4. **[5 points]** Create an array called **evens** of class Fraction. Initialize the fractions in array **evens** to: 2, 4, 6, 8, 10. Use an initialization list. Display the contents of the array on one line: 2 4 6 8 10. **Use for or range loop, not copy/pasting of redundant lines of code.**

**Part 4.** For each step, **write a recursive version** of a provided iterative function. Provided is a recursive function body and a testing loop. Each recursive function should only call itself (not some other function). Avoid global and static local variables.

**Part 4 steps:**

1. **[5 points]** Write a recursive function **line(n)** that draws a line (of dashes) of length n.  
   Example: line(5) displays: -----
2. **[5 points]** Write a recursive function **left\_arrow(n)** that draws an arrow that points to the left (left arrow followed by a n dashes).  
   Example: right\_arrow(5) displays: <-----
3. **[5 points]** Write a recursive function **right\_arrow(n)** that draws an arrow that points to the right (n dashes followed by a right arrow).  
   Example: right\_arrow(5) displays: ----->
4. **[5 points]** Write a recursive function **double\_arrow(n)** that draws an arrow that points to the left and right (left arrow, n dashes, followed by a right arrow).  
   Example: double\_arrow(5) displays: <----->
5. None of your functions should allow “run-away” recursion, regardless of the input.

**Part 5. [5 points]** Extra credit, up to 5 points. Write 5 different examples of C++ code that leads to “undefined behavior” and could **crash** a program. Each example must have a different fault. ***In a comment, explain the problem.*** An infinite loop is a **hang**, **not a crash**.

Example of crash:  
 // float sum=123.45; int count=0; float avg=dbl/count; // divide by zero, undefined behavior

You cannot use divide by zero as one of your crash behaviors, because it was provided.

When testing this code, your program will crash (of course!) Save before testing and comment out before submitting. **As usual for labs, paste your test output at the bottom.**

When you ***start*** exam lab 3, the code provided does this:

Step 1:

Contents of vector warehouse, already loaded:

hat 8 china 6

t-shirt 15 usa 14

coffee mug 6.5 canada 22

candy bar 0.79 mexico 3

34.59 japan 122

mixing bowl 4.99 brazil 20

Step 2:

Step 3:

Step 4:

Test provided iterative versions of line, left\_arrow, right\_arrow, double\_arrow:

n= 1:- <- -> <->

n= 2:-- <-- --> <-->

n= 3:--- <--- ---> <--->

n= 4:---- <---- ----> <---->

n= 5:----- <----- -----> <----->

Test student's new recursive versions of line, left\_arrow, right\_arrow, double\_arrow:

n= 1:

n= 2:

n= 3:

n= 4:

n= 5:

When you are ***all done***, the output will look similar to this:

Start...

Step 1:

Contents of warehouse, already loaded:

hat 8 china 6

t-shirt 15 usa 14

coffee mug 6.5 canada 22

candy bar 0.79 mexico 3

34.59 japan 122

mixing bowl 4.99 brazil 20

Contents of store, as loaded by student's new code:

hat $ 8.00

t-shirt $15.00

coffee mug $ 6.50

generic $ 1.00

generic $ 1.00

mixing bowl $ 4.99

Step 2:

Black umbrella is at address: 0x6e6480

black umbrella $19.95

Step 3:

a=6

b=3

e=42

2 4 6 8 10

Step 4:

Test provided iterative versions of line, left\_arrow, right\_arrow, double\_arrow:

n= 1:- <- -> <->

n= 2:-- <-- --> <-->

n= 3:--- <--- ---> <--->

n= 4:---- <---- ----> <---->

n= 5:----- <----- -----> <----->

Test student's new recursive versions of line, left\_arrow, right\_arrow, double\_arrow:

n= 1:- -> <->

n= 1:- <- -> <->

n= 2:-- <-- --> <-->

n= 3:--- <--- ---> <--->

n= 4:---- <---- ----> <---->

n= 5:----- <----- -----> <----->

...end.