MATH CSI 32 - Programming II Professor Luis Fernández

TEST 2. Time allowed: From 1pm on Thursday 04/23/2020to 11:59pm on Friday 04/24/2020.

Instructions

- Do the following questions. The value of each question is next to the question number. If you do more than 100 points, the additional will be extra credit.
- All the files required in the exam are located at the usual webpage, under "Test 2 files": https://fsw01.bcc.cuny.edu/luis.fernandez01/web/teaching/classes/csi32/csi322020-1.html. In addition, the links in this document take you directly to the files you need to download.
- Make sure your programs compile and run, although partial credit may be given even if it does not.
- Solve each exercise in a single file, and name each file using the naming convention. For example, for exercise 2, I would upload "LFernandez.ex3.cpp". If you really prefer, for the exercises involving classes you can use one file for the header, one for the class, and one for the `main` part. But make sure to make this clear.
- When you are finished, **upload your files to the following Dropbox link: https://www.dropbox.com/request/tK6G3R0XiOLp0iqsa5zY.**

Rules

- **You must use the names for the variables, and follow closely the instructions in the exercises**. You may use a different way of doing it, but the general structure **has to be** the one specified in the exercise.
- **You may not copy/paste from the internet**. However, you can use the book, and you can look for code in the internet and adapt it to your needs.
- **You may not get outside help** except from me (I may give you a hint if you ask).
- By uploading the solution of the exercises in this test, you implicitly commit to the following: **any violation of the previous two rules above will result on the total invalidation of the test**.

---

[20] **1.** Download the file timeclassexam2.cpp that contains the definition of class `time` (this is the one that we have been doing in class) and do the following modifications:

a) Write a `main` part of the program where you define an object `time1` of class `Time` initialized to 11:35:34pm.

b) In the same `main` part, define a variable `refTime1` that is a reference to the object `time1` and print its value using the member function `showTime24()`.

c) In the same `main` part, define a pointer `time1Ptr` that is a pointer to the object `time1` and print its value using the member function `showTime24()`.

d) Modify the destructor of class `Time` so that it prints "I am being destroyed!!" in a new line when an object is destroyed.

e) Define the `friend` external function `subTime` that takes two `Time` objects as input and returns another `Time` object that is the difference between the two input times. If the difference is less than 0, add 24 hours to it (this way, when t_2 is greater than t_1, $t_1 - t_2$ is interpreted as the amount of time from time t_2 to time t_1 *tomorrow*). Test it in the `main` part of the program by subtracting 14:56:52 from 13:34:45.

[Hint: convert both times to seconds as we did with the external `addTime` function, but subtract them instead of adding them.]

[20] **2.** Write a program whose `main` part contains two variables: `var1` of type `string` and `var2` of type `long double`. Then in the same `main` part, write statements (in sequence) of the following.

a) Define a pointer `varPtr` that points a `string`.

b) Make `varPtr` point to `var1`.

c) Use `varPtr` to assign the value "Good news!" to `var1`.

d) Print out the address (in the computer) of `var2`, followed by an `endl`.

e) Use the pointer `varPtr` to display the value of the variable it points to.

f) Can you make `varPtr` point to `var2`? Explain why or why not (write the answer as a comment in the `main` part of the program).

---

[20] **3.** Write a program with the definition of `class Rectangle` that has:

- Data members `length` and `width`, of type `double`, each of which defaults to 1.
- The constructor, that takes two `double`s (the length and the width)
- Function members `setLength`, `setWidth`, that set the length and width of the rectangle.
- Function member `perimeter` that finds the perimeter of the rectangle.
- Function member `area` that finds the area of the rectangle.

Provide a `main` part of the program where you define an object of class `Rectangle` and you use the functions `perimeter` and `area` to find the perimeter and area of the object, printing out the length, the width, the perimeter, and the area in a single line.

---

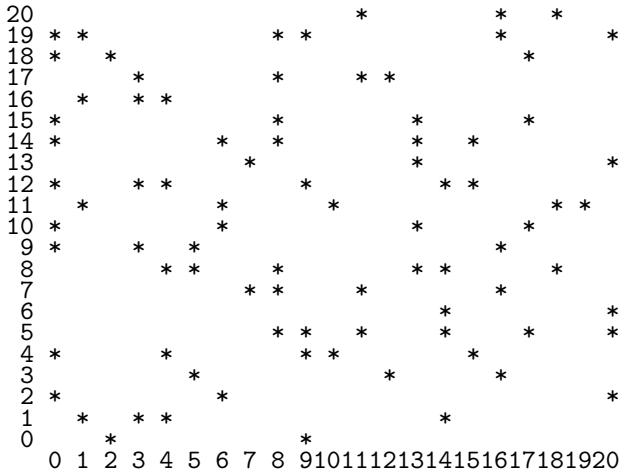
[20] **4.** Write a program where you define `class Card`, with the cards of a poker deck (excluding jokers). Your class must have:

- `int` private members `suit` and `value`, which are the suit (with values 0, 1, 2, 3, corresponding to Spades, Diamonds, Clubs, Hearts) and the value (with values 0 through 12, corresponding to Ace, 2, 3, ..., 10, Jack, Queen, King) (thus, if `suit` is 2 and `value` is 10, the card is the Jack of Clubs).
- Constructor and function `setCard` that take two integers as arguments, the first between 0 and 12 (to set the value) and the second between 0 and 3 (to set the suit). They should check that the values entered are correct.
- Functions `getValue` and `getSuit` that return the value and suit of the card, respectively.
- Function `printCard` that prints out the card in the format "Ace of Spades", or "2 of Hearts", or "Queen of Diamonds".

Write a `main` part of the program where the user is asked to enter 'd' to draw a card at random or 'q' to quit. When 'd' is entered, use appropriate `rand()` statements to pick a random value and suit, generate a card object with that value and suit and print it out using the function `printCard()`.

[20] **5.** Download the point.cpp class (or you can use the one you uploaded for HW 9). (Note that there is one small difference with the HW 9 exercise: the size of the board, which was 40 by 40 in the exercise, is set at the top using `xSize` and `ySize`; this way you can try different board sizes easily.) In the same file, define a class `PointArray` with the following members:

- One data member `ptArray` which is a vector of points (that is, `vector <Point> ptArray`).
- The constructor and set function `setArray` (which receive a `vector` of `Points` as argument).
- A plot function `void plotArray()` that plots the points in a similar way as you did in the `Point` class. Note that all the points should be plotted *at the same time*. Use this class to generate a random vector of 100 `Points` and plot it. The output may look like:



[Hint: for the `plotArray` function, the pseudocode can be something like

for j from `ySize` to 0 (y coordinate going down)
  for i from 0 to `xSize` (x coordinate going right)
    for each element in arrPt
    {
      if x and y coordinates of element are i and j, cout "*" and increase i by 1;
    }
  cout " " after the end of the innermost loop but inside the second loop.
then cout an endl after the end of the inner loop.

This assumes that you are using `xSize` and `ySize` as the dimensions of your drawing board.]

[15] **6.** (You need to do exercise 4 before doing this one. Do it as an addition to the file in exercise 4 since you have to use the class `Card`; but make sure you upload it as a different file).

Define a class `DeckOfCards` as follows:

- One data member `deck` that is a `vector` of `Card`s.
- The constructor should receive no arguments and produce a vector of 52 **different** `Card`s ordered by suit (Spades, Diamonds, Clubs, Hearts) and by value (1, 2, ..., Queen, King).
- A member function `showDeck` that prints out the cards in the deck, ordered in rows of 13 cards each, ordered left to right. Each card should be printed using value and suit with the format " 2S" (2 of Spades), or " KH" (King of Hearts) — you may want to redefine the function `showCard` from exercise 4 to do this.
- A member function `shuffleDeck` that changes the order of the elements in the vector `deck`. To do this, pick two random cards in the deck and swap them, and repeat this 500 times.
- An external `friend` function `mergeDecks` that takes two `DeckOfCards` objects and returns a new `DeckOfCards` object made by merging the two decks, one after the others.

In the `main` part, define two `DeckOfCards` object, show one of them, then shuffle one of them and show it, then merge them, and finally show the resulting (double) deck. The output may look like:

```
New deck:

 AS    2S    3S    4S    5S    6S    7S    8S    9S   10S    JS    QS    KS
 AD    2D    3D    4D    5D    6D    7D    8D    9D   10D    JD    QD    KD
 AC    2C    3C    4C    5C    6C    7C    8C    9C   10C    JC    QC    KC
 AH    2H    3H    4H    5H    6H    7H    8H    9H   10H    JH    QH    KH

Shuffled deck 1:

 4H    8H    4S    4C    3D    6S    2C    7D    KD    3H    AS    JS    7H
 QH    9S    JD    8C    AH    QD    2D    AC    KH    JH    9H    JC    3C
 9D    6C    5H   10S    QS    KC    5C    9C    QC   10D    KS    5S    5D
 8D    6H    3S    2S    4D   10H    7S    AD    8S    2H   10C    6D    7C

Shuffled merged decks:

 AS    2S    3S    4S    5S    6S    7S    8S    9S   10S    JS    QS    KS
 AD    2D    3D    4D    5D    6D    7D    8D    9D   10D    JD    QD    KD
 AC    2C    3C    4C    5C    6C    7C    8C    9C   10C    JC    QC    KC
 AH    2H    3H    4H    5H    6H    7H    8H    9H   10H    JH    QH    KH
 4H    8H    4S    4C    3D    6S    2C    7D    KD    3H    AS    JS    7H
 QH    9S    JD    8C    AH    QD    2D    AC    KH    JH    9H    JC    3C
 9D    6C    5H   10S    QS    KC    5C    9C    QC   10D    KS    5S    5D
 8D    6H    3S    2S    4D   10H    7S    AD    8S    2H   10C    6D    7C
```

[15] **7.** (You need to do exercise 5 before doing this one. Do it as an addition to the file in exercise 5 since you have to use classes `Point` and `Pointarray`; but make sure you upload it as a different file).

Define a class `Rectangle` with:

- Data members: two `Point` objects corresponding to the lower left corner and the upper right corner of the rectangle.
- Two constructors and two member functions `setRect`, one that takes two `Point` objects as input and one that takes 4 integers as inputs (the x and y coordinates of the lower left and upper right corners of the rectangle). Validate the input (points should be within the board defined in `class Point`, and the coordinates of the lower left corner should be less than the corresponding coordinates of the upper right corner) and default to the rectangle with lower left corner $(0, 0)$ and upper right corner $(1, 1)$.
- Get functions that return the x and y coordinates of the lower left and upper right corners: `getXll` (get x coordinate of lower left corner), get `getYll`, `getXur`, and `getYur`.
- Member function `plotRectangle` that plots the rectangle object with '*' characters (to do this, define a vector of `Point`s and plot it using `plotArray` from exercise 5).

In the `main` part, define a `Rectangle` object and plot it using `plotRectangle`. For example, the plot of the object `Rectangle rect{5,8,14,12}` may look like