UNIVERSITY OF CALIFORNIA, MERCED

BERKELEY · DAVIS · IRVINE · LOS ANGELES · MERCED · RIVERSIDE · SAN DIEGO · SAN FRANCISCO

SANTA BARBARA · SANTA CRUZ

CSE-165: Object-Oriented Programming
Lab 6

Spring 2024

Preliminary Notes

- Write separate file(s) for each exercise. Zip all your files together and submit the zip file to CatCourses.
- Your solution must be exclusively submitted via CatCourses. Email submissions will not be accepted. Pay attention to the posted deadline!

1 Integer Sorting (20 points)

Write a class called Data. Your class should store a vector of (for now) integers.

Your class should have the following methods:

void add(int number); // Adds a number to the data set

void sort(); // Sorts the data set in ascending order. You may implement any sorting algorithm here, such as bubble sort, insertion sort, merge sort, quick sort, etc... There is no need to try to implement the most efficient one, any one will do.

void print(); // Prints out the entire data set on a single line, with elements separated by spaces

Make sure your class works as expected with the file **intSort.cpp**.

Example Output:
4 5 2 3 1
1 2 3 4 5

2 Sortable Objects (20 points)

Study the file Sortable.h. It contains a simple abstract class called Sortable, which will act as a base class for any object a collection of which can be sorted and printed.

Sortable has two pure virtual methods, compare and print. The compare method tells the object how to compare itself with another object. It returns true if the object it is being called from is smaller than the other object, and false otherwise. The print method allows the object to display itself.

Modify your Data class so that it operates on (pointers to) Sortable objects instead of ints. You will first have to change int to Sortable*. You will also have to change your sort() method to use the compare() method of Sortable, and the print() method to use the Sortable's print() method. (Remember, you are not making Data inherit from Sortable! We are doing composition here, not inheritance.)

For this exercise, you are not provided any .cpp files in order to test your code yet. Testing your code will involve creating a class that extends Sortable and pushes some objects of that class into your Data object and trying to sort and print them. This is the goal of the next exercise.

Submit your modified Data class.

3 Sortable Objects II (20 points)

Reusing Data.h and Sortable.h from the previous exercise, create a class called Circle which inherits from Sortable.

A Circle object will only have a float radius. Circle objects are compared according to radius (one Circle is smaller than another Circle if its radius is smaller). Circle objects should be printed one per line as follows: "Circle with radius: ¡x¿", where ¡x¿ is the radius of the circle.

You can test your code with the file **sortingCircles.cpp**. If you correctly implemented the changes to Data in the previous exercise, the Circle class you create here should work with sortingCircles.cpp without further modifications to Data.

Example Output:
Circle with radius: 0
Circle with radius: 3
Circle with radius: 2
Circle with radius: 4
Circle with radius: 1

Circle with radius: 0
Circle with radius: 1
Circle with radius: 2
Circle with radius: 3
Circle with radius: 4

4 Sortable Objects III (20 points)

This question is almost the same as the last one. You can reuse the files Data.h and Sortable.h. This time we will be storing objects of the Participant class. Each Participant has a name, an age, and a score. These are string, int, and double, respectively. To print a Participant simply print the name, age, and score separated by tab characters.

Participant objects should be ordered by score from highest to lowest. If two Participants have the same score then the younger of them takes priority and if they have the same score and the same age, simply order them alphabetically. You can test your code with the file **sortingParticipants.cpp**.

Example Output:

```
Waymond 24      100
Mary    27      96
John    32      100
Eliza   21      105
Ezekiel 27      96
Alex    20      101

Eliza   21      105
Alex    20      101
Waymond 24      100
John    32      100
Ezekiel 27      96
Mary    27      96
```

5 Sortable Objects IV (20 points)

Suppose we wish to be able to choose specifically which attribute of a Participant to sort by (name, age, or score) as opposed to the default behavior in the previous exercise. We will do this by first declaring three separate comparison functions in the .cpp file. (These functions have been defined for you in the sortingByAttribute.cpp file.)

We will then add to the Participant class a static pointer to a function. (Static members of a class are shared between all instances of the class.) This pointer should be called comp_cb and it will point to one of the comparison functions defined in the source file:

static bool (*comp_cb)(const Participant*, const Participant*);

Your task for this exercise is to make the necessary modifications to your Participant class to use this comparison function in the compare() method. Your class should work with **sortingByAttribute.cpp**. The modifications you will actually have to do to the class are minimal, but focus on understanding what is going on in the code and how the functionality is being added.

Example Output:

```
Waymond 24      100
Mary    27      96
John    32      100
Eliza   21      105
Ezekiel 27      96
Alex    20      101

Alex    20      101
Eliza   21      105
Ezekiel 27      96
John    32      100
Mary    27      96
Waymond 24      100
```

```
Alex    20      101
Eliza   21      105
Waymond 24      100
Mary    27      96
Ezekiel 27      96
John    32      100

Eliza   21      105
Alex    20      101
Waymond 24      100
John    32      100
Ezekiel 27      96
Mary    27      96
```