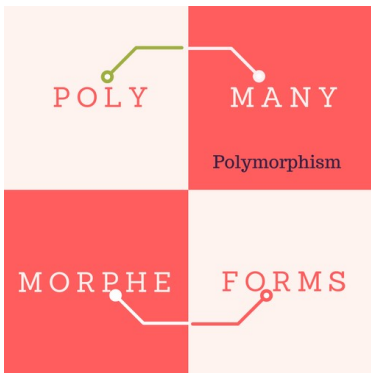
Homework 12 - Searching and sorting analysis

Start Assignment

Due Monday by 11:59pm **Points** 30 **Submitting** a file upload

Module 12 – Polymorphism and Abstract Classes

Homework Exercises



Summary

1. Review Questions #2, 8, 11, and 13 – 18 from chapter 15 of our textbook (9th edition: pages 983 - 984, 10th ed: 997 - 998)
2. Programming Challenges 1 and 3 (9th edition: pages 984 - 985, 10th ed: 999 - 1000)

Please remember to add comments throughout your code, and place each programming project in its own program file.

Review Questions (10 points)

2. A member function of a class that is not implemented is called a(n) _____ function .
8. A base class pointer needs a(n) _____ to be assigned to a derived class pointer.
11. If every **C1** class object can be used as a **C2** class object, the relationship between the two classes should be implemented using _____ .
13. The keyword _____ prevents a virtual member function from being overridden.
14. To have the compiler check that a virtual member function in a child overrides a virtual

member function in the parent, you should use the keyword _____ after the function declaration .

Suppose that classes **Dog** and **Cat** derive from **Animal**, which in turn derives from **Creature**. Suppose further that **pDog**, **pCat**, **pAnimal**, and **pCreature** are pointers to the respective classes. Finally, suppose that **Animal** and **Creature** are both *abstract* classes, but **Dog** and **Cat** are not *abstract*.

15. Will the statement:

```
Animal a ;
```

compile? If not, why not?

16. Will the statement

```
pAnimal = new Cat ;
```

compile? If not, why not?

17. Will the statement

```
pCreature = new Dog ;
```

compile? If not, why not?

18. Will the statement

```
pCat = new Animal ;
```

compile? If not, why not?

Project 1 – Analysis of Sorting Algorithms (10 points)

Programming Challenge #1: Design a class named **AbstractSort** that can be used to analyze the number of comparisons performed by a sorting algorithm. The class should have a member function **compare** that is capable of comparing two array elements, and a means of keeping track of the number of comparisons performed. The heading of the **compare** function should be:

```
int compare(int arr[ ], int k, int m)
```

In addition to tracking the number of times called, it should return:

- a negative number if **arr[k]** should precede **arr[m]** in the sorted array
- zero if **arr[k]** and **arr[m]** can appear in either order in the sorted array

- a positive number if **arr[k]** should follow **arr[m]** in the sorted array.

The **AbstractSort** class should be an abstract class with a pure virtual member function:

```
virtual void sort(int arr[ ], int size ) = 0 ;
```

which, when overridden, will sort the array by calling the **compare** function to determine the relative order of pairs of numbers.

Create a subclass of **AbstractSort** that uses a simple sorting algorithm to implement the sort function. The class should have a member function that can be called after the sorting is done to retrieve the number of comparisons performed.

In **main**, create two arrays, each initialized with an unsorted set of numbers and of different sizes. For each array, your output should show the array before and after sorting, as well as the number of comparisons that were required to sort the array,

Project 2 – Sequence Sum (10 points)

Programming Challenge #3: A sequence of integers such as 1, 3, 5, 7, ... can be represented by a function that takes a non-negative integer as parameter and returns the corresponding term of the sequence. For example, the sequence of odd numbers just cited can be represented by the function

```
int fun(int k) {
    return 2 * k + 1 ;
}
```

Write an abstract class **AbstractSeq** that has a pure virtual member function

```
virtual int fun(int k) = 0 ;
```

as a stand-in for an actual sequence, and two member functions with prototypes:

```
void printSeq(int k, int m} ;
int sumSeq(int k, int m) ;
```

that are passed two integer parameters **k** and **m**, where **k** < **m**. The function **printSeq** will print all the terms **fun(k}** through **fun(m)** of the sequence, and likewise, the function **sumSeq** will return the sum of those terms. Define the **printSeq** and **sumSeq** outside of the **AbstractSeq** class.

Demonstrate your **AbstractSeq** class by creating two child classes that you use to define the **fun** functions of two different sequences:

1. The first child class should return the odd numbers in sequence (fun(0) = 1, fun(1) = 3, fun(2) = 5, fun(3) = 7...). Print the sequence using the **printSeq** function from fun(3) to fun(12).
2. The second child class should return the factorial of the parameter (fun(0) = 1, fun(1) = 1, fun(2) = 2, fun(3) = 6, fun(4) = 24...). Remember that the factorial of a number is the product of all numbers from 1 to that number, for example 5 factorial (written 5!) is 1 * 2 * 3 * 4 * 5 = 120, and 0! = 1. Print the sequence from fun(1) to fun(10).

For each child class above, be sure to show the argument passed to the function and the value returned from the function.

For each child class, also display the sum of the numbers using **sumSeq**. Start and end somewhere *inside* the sequence printed above, rather than summing the entire printed sequence. For example, you might use **sumSeq(7, 9)** to calculate the sum of the functions in the first child class from fun(7) to fun(9), and **sumSeq(4, 6)** to calculate the sum of the factorials from 4 to 6. Make sure to identify the beginning and ending points of the sequences when displaying the sum, instead of simply printing a number.

For example, the output for the first child class might look like the following:

```
Odd numbers fun(3) to fun(12) are:

        n      fun(n)
     -----  ----------
         3           7
         4           9
         5          11
         6          13
         7          15
         8          17
         9          19
        10          21
        11          23
        12          25

The sum of fun(7) to fun(10) is 72
```

Submit all program files, snips of the output of your programs, and a document with the answers to the Review Questions.

Links

Additional Files and Programs

none

Next Assignment

Homework 13
(https://miracosta.instructure.com/courses/31330/assignments/842799)

Homework Assignment

Lab 12 - Polymorphism, Virtual Functions
(https://miracosta.instructure.com/courses/31330/assignments/842810)

Prior Lab

Homework 11
(https://miracosta.instructure.com/courses/31330/assignments/842797)