Lab 7: Inheritance & OOP Practice

CPSC 1021

School of COMPUTING Clemson University

Lab Objective

- Use inheritance to create a class from an existing class
- Practice solving programming problems using OOP concepts and techniques

Introduction

The purpose of this lab is to practice using OOP principles to solve problems. There are four programming challenges. The first two challenges require using inheritance to create a new class derived from the C++ string class. The purpose of the other challenges is to practice concepts and techniques we have covered up to this point, such as writing classes and creating objects, working with static member variables, and operator overloading.

Collaboration Policy

Work with your assigned team members from Lab 0. ***Please make sure to integrate your team members who – because of COVID requirements – have to participate remotely.*** You can use the MS Teams video conferencing feature to collaborate with a remote team member. Use the search bar on the top of the MS Teams app to search for your team member with their CU user name, then click the video icon in the top-right corner of the screen to start a video call.

You and your partners can work on the assignments together.

Submission Policy

Code solutions to all challenges **can be submitted as teams** this week. This means only one member from each team needs to submit the code files on Gradescope (this person must then add all their other team members to the submission). However, **you are required to work on all problems together**.

Challenge 1 – Palindrome Testing

A palindrome is a string that reads the same backward as forward. For example, the words *mom*, *dad*, *madam*, and *radar* are all palindromes. Write a class `Pstring` that is derived from the STL string class. That is, use inheritance to create `Pstring` from the existing string class. The Pstring class adds a member function

```
bool isPalindrome( )
```

that determines whether the string is a palindrome. Include a constructor that takes an STL string object as parameter and passes it to the string base class constructor.

Demonstrate your class by having a main program that asks the user to enter a string. The program uses the string to initialize a `Pstring` object and then calls `isPalindrome()` to determine whether the string entered is a palindrome. You may find it useful to use the subscript operator `[]` of the string class: If str is a string object and k is an integer, then `str[k]` returns the character at position k in the string. Format your program output exactly as demonstrated in the following sample runs.

Sample Run

Example 1:

```
This is a palindrome-testing program. Enter a string to test: radar
radar is a palindrome
```

Example 2:

```
This is a palindrome-testing program. Enter a string to test: computer
computer is not a palindrome
```

Submission Instructions

- Submit your complete and correct program to Gradescope. Your submission must include the following four files:
  1. main.cpp
  2. Pstring.h
  3. Pstring.cpp
  4. Makefile (this file should create main.out)
- Make sure your program compiles and runs on the SoC Linux machines; the autograder automatically assigns 0 points to programs that do not compile.
- Code style (proper indentation, consistency, readability, use of comments, etc.) will be considered when grading your submission.

Challenge 2 – String Encryption

Write a class `EncryptableString` that is derived from the STL string class. That is, use inheritance to create `EncryptableString` from the existing string class. The Encryptable string class adds a member function

```
void encrypt( )
```

that encrypts the string contained in the object by replacing each letter with its successor in the ASCII ordering. For example, the string `baa` would be encrypted to `cbb`. Assume that all characters that are part of an `EncryptableString` object are letters a, .., z and A, .., Z, and that the successor of z is a and the successor of Z is A. Demonstrate your class with a program that asks the user to enter strings that are then encrypted and printed. Format your program output exactly as demonstrated in the following sample runs.

Sample Run

Example 1:

```
This is an Encryption program. Enter a string to encrypt: Clemson
Here is the encrypted string: Dmfntpo
```

Example 2:

```
This is an Encryption program. Enter a string to encrypt: Zebra
Here is the encrypted string: Afcsb
```

Example 3:

```
This is an Encryption program. Enter a string to encrypt: Fizz
Here is the encrypted string: Gjaa
```

Submission Instructions

- Submit your complete and correct program to Gradescope. Your submission must include the following four files:
  1. main.cpp
  2. EncryptableString.h
  3. EncryptableString.cpp
  4. Makefile (this file should create main.out)
- Make sure your program compiles and runs on the SoC Linux machines; the autograder automatically assigns 0 points to programs that do not compile.
- Code style (proper indentation, consistency, readability, use of comments, etc.) will be considered when grading your submission.

Challenge 3 – Corporate Sales

A corporation has four divisions, each responsible for sales to different geographic locations. Design a `DivSales` class that keeps sales data for a division, with the following members:

- A vector `sales` with four elements for holding four quarters of sales figures for the division.
- A private variable `double divSales` for holding the total sales for a division for the entire year.
- A private static variable `double totalSales` for holding the total corporate sales for all divisions for the entire year.
- A member function `setSales` that takes four arguments, each assumed to be the sales for a quarter. The value of the arguments should be copied into the vector that holds the sales data. The total of the four arguments should be added to `divSales` and to the static variable that holds the total yearly corporate sales. Use this member function when entering the a division's quarterly sales.
- A getter function `getDivSales`.
- A getter function `getCorpSales`.

Write a program that creates a vector of four `DivSales` objects. The program should ask the user to enter the sales for four quarters for each division. If the user enters a value <0, the program should prompt the user to enter a valid value (implement this input validation in your main program). After the data is entered, the program should display a table showing the division sales for each quarter. The program should then display the total corporate sales for the year. Format your program output exactly as demonstrated in the following sample run.

Sample Run

```
Enter sales data for Division 1
Quarter 1: 10.0
Quarter 2: 20.0
Quarter 3: 30.0
Quarter 4: 40.0
Enter sales data for Division 2
Quarter 1: -100.0
Please enter 0 or greater: -100.0
Please enter 0 or greater: 100.0
Quarter 2: 200.0
Quarter 3: 300.0
Quarter 4: 400.0
Enter sales data for Division 3
Quarter 1: 10.50
Quarter 2: 20.50
Quarter 3: 30.50
Quarter 4: 40.50
Enter sales data for Division 4
Quarter 1: 1000.0
Quarter 2: 2000.0
Quarter 3: 3000.0
Quarter 4: 4000.0
Total Sales for Division 1: $100.00
Total Sales for Division 2: $1000.00
Total Sales for Division 3: $102.00
Total Sales for Division 4: $10000.00
Total Corporate Sales: $11202.00
```

Submission Instructions

- Submit your complete and correct program to Gradescope. Your submission must include the following four files:
  1. main.cpp
  2. DivSales.h
  3. DivSales.cpp
  4. Makefile (this file should create main.out)
- Make sure your program compiles and runs on the SoC Linux machines; the autograder automatically assigns 0 points to programs that do not compile.
- Code style (proper indentation, consistency, readability, use of comments, etc.) will be considered when grading your submission.

Challenge 4 – Rational Arithmetic

A *rational number* is a quotient of two integers. For example, 12/5, 12/–4, –3/4, and 4/6 are all rational numbers. A rational number is said to be in *reduced form* if its denominator is positive and its numerator and denominator have no common divisor other than 1. For example, the reduced forms of the rational numbers given above are 12/5, –3/1, –3/4, and 2/3.

Write a class called `Rational` with a constructor `Rational(int, int)` that takes two integers, a numerator and a denominator, and stores those two values in reduced form in corresponding private members. The class should have a private member function `void reduce()` that is used to accomplish the transformation to reduced form. The class should have an overloaded insertion operator << that will be used for output of objects of the class.

Demonstrate your class by having a main program that asks the user to enter two integres. The program uses the integers to initialize a Rational object and then calls the overloaded operator << to print the rational number in reduced form. Format your program output exactly as demonstrated in the following sample runs.

Sample Run

Example 1:

```
Enter the numerator and denominator separated by a space: 12 4
Reduced form: 3/1
```

Example 2:

```
Enter the numerator and denominator separated by a space: 6 -24
Reduced form: -1/4
```

Submission Instructions

- Submit your complete and correct program to Gradescope. Your submission must include the following four files:
  1. main.cpp
  2. Rational.h
  3. Rational.cpp
  4. Makefile (this file should create main.out)
- Make sure your program compiles and runs on the SoC Linux machines; the autograder automatically assigns 0 points to programs that do not compile.
- Code style (proper indentation, consistency, readability, use of comments, etc.) will be considered when grading your submission.