CS 112 – Computer Science Fundamentals II
Assignment #4

Deadlines

This assignment is due on **Tuesday, November 10, 2020 @ 11:00 AM (beginning of class!)**

How to submit your work on the assignment:

Assignments will be accepted via Canvas as CPP and H files. Use the Multiple File submission tool in Canvas to submit all files. Make sure to include the CPP file containing your main() function that contains the tests of your code. Submit all the files in a single submission, and do NOT use ZIP.

It is expected AND REQUIRED that all the code you submit is your work and your work alone. Copying of any code between students is strictly prohibited! Also, all functions you write MUST have the correct name and headers that match the problems specifications. If the instructor's testing statements can't run your functions because of errors in the headers, that will result in major deductions in your assignment grade!

Defining a PlayingCard class

This assignment is designed to be a short review of creating C++ class definitions. Instead of applying new concepts, the focus should be in creating well-written code that meets all relevant style requirements – make this as clean and polished as you can! It will also be used in future assignments in this class, so be sure to meet this assignment's requirements as listed below.

You will be modeling a playing card, as found in a standard 52-card deck with the following identifying characteristics:

SUIT – Hearts, Diamonds, Clubs, or Spades
RANK – Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King

Create files **PlayingCard.cpp** and **PlayingCard.h** to hold the class definition and methods.

You must set up your **PlayingCard** class to have its data members all private.

The data types you use for the data members in your class definition are entirely up to you. Anyone who uses your **PlayingCard** class doesn't need to know how you store that information, since those are private data members. All the use needs to know is how to call the methods you define below.

Minimum requirements for class PlayingCard class – 40 points

- It needs to include at least two suitable **private** data fields.
- It needs to include **at least two constructor methods**:
  - A no-argument **constructor** – What you choose for the default values of the **PlayingCard** data members when this constructor is used in entirely up to you, but the data members must be initialized to some valid value.
  - A **constructor** that allows the user to initialize all the data fields for a newly-created **PlayingCard** object. It should expect a **int** containing the rank (in the range **[1,13]**, where **1** is an **Ace** all the way up to **13** for a **King)** and a **char** containing the suit (one of **'H'**, **'D'**, **'C'**, **'S'**, upper or lower case, for Hearts, Diamonds, Clubs, or Spades, respectively. This **constructor** should initialize all the data fields for the new **PlayingCard** object, with error checking so the object's data members are set to valid values.

- It needs an **accessor method** for each data member, named **getSuit** and **getRank**. The **getSuit** method should return a **char** of one of **'H', 'D', 'C',** or **'S'**. The **getRank** should return a value of type **int** in the range **[1,13]**, where **1** is an **Ace** all the way up to **13** for a **King**.
- It needs a **mutator method** for each data member, named **setSuit** and **setRank**. Each mutator should perform error-checking to ensure that the object's data members are set to valid values. It should expect a value for the data member that exactly matches the description above for what the accessor method returns.
- It needs a public method **printCard** that expects nothing and returns nothing, and has the side effect of printing to the screen the values stored in the data members for the calling **PlayingCard** object. The format of the output printed to the screen needs to conform to the style given below. Note the use of spaces to align the suit values (in other words, the rank values should print in exactly 5 characters, including trailing blanks, and the suit values should print in exactly 8 characters, including trailing blanks!).

```
Three of Diamonds
Jack  of Clubs
Ace   of Spades
```

- It needs a public method **cardToString** that expects nothing and returns a **string** version of the **PlayingCard** object (a **string** that includes the data member values for the calling **PlayingCard** object). The content of the **string** value should match exactly the style specified above for the output to the screen.
- You may include any additional public and private methods that you would like to include.

In the code you write to test your **PlayingCard** class definition and methods:

- Declare at least two **PlayingCard** instances, with at least one created using your no-argument constructor, and at least one created using the multiple-argument constructor.
- Use the method **printCard** to display your **PlayingCard** instances.
- Demonstrate the correct execution of the other methods you write by writing appropriate tests.

Define a derived BridgeCard class based on the PlayingCard class – 60 points

In many playing card games like Spades and Bridge, there is the concept of a "trump suit" that is either fixed (as in the case of Spades, where "Spade" is always the suit) or can vary (as in the case of Bridge, where players make bids for the right to set the trump suit for that hand). This means that in these games, having a **PlayingCard** object with an additional data member to indicate whether the card is in the trump suit for that hand can be useful.

We'll implement this as a derived class (or subclass) of the **PlayingCard** class. This derived class will be called **BridgeCard** and will have an additional **private** data member to indicate whether the card is in the trump suit. How you implement this additional data member is up to you.

Create files **BridgeCard.cpp** and **BridgeCard.h** to hold the derived class definition and methods. The methods your derived classes must include:

- A no-argument constructor
- A constructor that expects two arguments:

- An **int** for the rank of the card, as explained in the PlayingCard instructions above
- A **char** for the suit of the card, as explained in the PlayingCard instructions above

The value of the additional data field for the trump suit should be initialized to indicate that the card is not in the trump suit

- An accessor method **getTrump** that expects nothing and returns **true** if the **BridgeCard** field says the card is in the trump suit and **false** if it does not.
- A mutator method **setTrump** that expects a **char** value of **'H', 'D', 'C', 'S',** or **'N'** (for "no trump"). It will set the **BridgeCard** field as appropriate – depending on whether the **BridgeCard's** suit matches the suit of the passed **char** value.
- A **REDEFINED** version of the public method **printCard**, which will also print the additional data field(s) of this derived class instance as an asterisk. For example, if the **BridgeCard** is the Queen of Diamonds and has the **BridgeCard** field is set to indicate that it's in the trump suit, then the method will print:

  `Queen of Diamonds*`
- A **REDEFINED** version of the public method **cardToString**, that will return a string that matches exactly what the printCard writes to the screen, including the additional character to indicate trump suit – for example, the Queen of Diamonds will return either **`"Queen of Diamonds*"`** or **`"Queen of Diamonds "`** (← note the space here!).

In the code you write to test your **BridgeCard** class definition and methods:

- Declare at least two **BridgeCard** instances, with at least one created using your no-argument constructor, and at least one created using the multiple-argument constructor.
- Write statements to call all the methods as needed to test cases with the **BridgeCard** data member.
- Demonstrate the correct execution of the other methods you write by writing appropriate tests.

Use the Multiple File submission tool in Canvas to submit your **PlayingCard** and **BridgeCard** files, and your file with the testing code. Submit all the files in a single submission, and do NOT use ZIP.