Assignment 8

Due Nov 16 by 7pm **Points** 120

Project 8.a

Write a template class called ValSet, which represents a mathematical set of values. For convenience I will refer to the generic type of these values as "T". The ValSet will store the values in an array, which will need to grow if it gets full. ValSet should have the following data members:

- a pointer-to-T that points to a dynamically allocated array (of T) that holds the values that are part of the set
- an int that holds the current size of the array - it will need to be updated whenever the add() method creates a larger array
- an int that holds the number of values that are currently part of the set - it will need to be updated in the add() and remove() methods

The ValSet class should have the following methods:

- a default constructor that initializes the pointer data member to point to an array of size 10, initializes the variable that stores the size of the array to 10, and initializes the variable that stores the number of values in the set to zero
- a copy constructor that initializes the pointer data member to point to an array of the same size as the one being copied, copies over the array values, and also copies the values for the size of the array and the number of values in the set
- an overloaded assignment operator that initializes the pointer data member to point to an array of the same size as the one being copied, copies over the array values, copies the values for the size of the array and the number of values in the set, and returns the object pointed to by the *this* pointer
- a destructor that deallocates the dynamically allocated array
- the size() method should return the number of values currently in the ValSet
- the isEmpty() method should return true if the ValSet contains no values, and return false otherwise
- the contains() method should take a parameter of type T and return true if that value is in the ValSet, and return false otherwise
- the add() method should take a parameter of type T and add that value to the ValSet (if that value is not already in the ValSet) - if the array is currently full and you need to add another value, then you must first increase the size of the array by allocating a new array that is twice as large, copying the contents of the old array into the new array, redirecting the data member pointer to the new array, and deallocating the old array (**avoid memory leaks - order matters**)
- the remove() method should take a parameter of type T and remove it from the ValSet (if that value is in the ValSet) by shifting over all of the subsequent elements of the array - it's okay if values that are no longer part of the set are still in the array, so long as you do the right bookkeeping
- the getAsVector method should return a vector (of type T) that contains all of the values in the ValSet and only those values. Order doesn't matter.

One short example of how the ValSet class could be used:

```
ValSet<char> mySet;
mySet.add('A');
mySet.add('j');
mySet.add('&');
mySet.remove('j');
mySet.add('#');
int size = mySet.size();
ValSet<char> mySet2 = mySet;
bool check1 = mySet2.contains('&');
bool check2 = mySet2.contains('j');
```

The files must be named: **ValSet.hpp** and **ValSet.cpp**

At the end of the ValSet.cpp file, you must put the following lines:

```
template class ValSet<int>;
template class ValSet<char>;
template class ValSet<std::string>;
```

The reason is that since the template definition and the main method that uses it are in separate "compilation units", the compiler doesn't see what types of ValSet classes need to be created, so it doesn't create them. Putting in those three lines forces the compiler to create those three versions of the ValSet class. This way of handling it can only be used if you know ahead of time which types of the template classes will be needed. The other way to handle this is to put all of the template definition into the header file, which is the way the Standard Template Library handles it, but is not what you will do for this project.

Project 8.b

Write a class called CustomerProject. It should have three double fields called *hours* (the number of hours the project took), *materials* (the cost of materials) and *transportation* (transportation costs), in that order. It should have get and set methods for each field. It should have a constructor that takes as parameters the values for each field and calls the set methods to initialize them. It should have a **pure virtual** function called *billAmount* that takes no parameters and returns a double.

Write two classes that inherit from CustomerProject: RegularProject and PreferredProject. Both should have a constructor that takes three doubles and passes them to the base class constructor. RegularProject should override billAmount to return the sum of the materials costs, the transportation costs, and $80 times the number of hours. PreferredProject (for preferred customers) should override billAmount to return the sum of 85% of the materials costs, 90% of the transportation costs, and $80 times the number of hours up to a maximum of 100 hours (any hours beyond 100 are free).

Hint: You can access the member data of the base class either of two ways: 1) make that member data *protected* in the base class, or 2) simply use the get methods, which the derived classes have inherited.

There are no input validation or output formatting requirements.

The files must be called: **CustomerProject.hpp**, **CustomerProject.cpp**, **RegularProject.hpp**, **RegularProject.cpp**, **PreferredProject.hpp**, and **PreferredProject.cpp**.