You are to implement a class that stores a set of strings. The class should use a *dynamic array* as its underlying representation. This assignment is worth 5% of your grade.

Please read the requirements carefully, paying particular attention to the names and input and output requirements of the class and its methods. We will be testing your class in our test program, so if you do not follow the requirements the program may not compile. *Assignment submissions that do not compile will not be marked.*

We will be compiling and running our test program and your class in Linux using the *g++* compiler with the `-std=c++11` option. If you complete your assignment in some other environment (such as Visual Studio) I would strongly suggest checking that it compiles in *g++* before submitting it.

### String Set Class

#### Class Description
Your class should be named *StringSet* and should support these operations:
- Creating an empty set
- Inserting a string
- Removing a string
- Finding a string
- Returning the size of a set
- Returning the union of the calling object and another *StringSet*
- Returning the intersection of the calling object and another *StringSet*
- Returning the set difference of the calling object and another *StringSet*

#### Class Attributes
Your class should be implemented using a dynamic array and should have at least the following private member variables
- A *pointer to a string* that will point to an array of strings created in dynamic memory
- An *int* that records the current size of the array (i.e. the number of strings stored in the array)
- An *int* that records the maximum size of the array

#### String Support
Your header file should include the string class (*#include <string>*). The string class is part of the standard namespace (*std)*.

#### Class Files
Your class should consist of a header file called *StringSet.h* that contains the class definition and an implementation file called *StringSet.cpp* that contains the method definitions
**StringSet Methods**

You should implement each of the public methods described below. In each case you are given the method header which specifies the method’s name, return type and parameter list. A brief description follows each method, including any exceptions that should be thrown by the method. You may implement other private methods as you deem necessary.

**Methods**

- **StringSet()** – constructor that creates an array of strings of size 2 in dynamic memory, sets maximum size to 2, and current size to 0
- **StringSet(const StringSet &)** – copy constructor that creates a deep copy of its parameter
- **~StringSet()** – destructor that frees dynamic memory associated with the object’s string (array) pointer
- **StringSet & operator=(const StringSet &)** – overloaded assignment operator that creates a deep copy of its parameter, assigns it to the calling object, deallocates any no longer required dynamic memory associated with the calling object; this method should check for self assignment as discussed in class
- **bool insert(string)** – returns false without inserting value if a string matching the parameter is already in the array, otherwise: inserts the string parameter at the next available index; if the underlying array is full, doubles the maximum size attribute, creates an array of twice the size of the current array, copies the contents of the old array to the new array, frees memory associated with the old array, and assigns new array’s address to object’s array pointer, then inserts parameter; increments current size and returns true
- **bool remove(string)** – returns false if a string matching the parameter is not in the array, otherwise: replaces matching string with the last string in the array (if there is one); decrements current size and returns true
- **int find(string) const** – if a string matching the parameter is in the array returns the index of that string, otherwise returns -1
- **int size() const** – returns the current size (the number of strings in the array)
- **StringSet unions(const StringSet &) const** – returns a StringSet object that contains the union of the calling object and the parameter (if the result is the empty set the returned StringSet object’s current size should equal zero); in case you were wondering, this method is called unions because union is a reserved word
- **StringSet intersection(const StringSet &) const** – returns a StringSet object that contains the intersection of the calling object and the parameter
- **StringSet difference(const StringSet &) const** – returns a StringSet object that contains the set difference of the calling object and the parameter

**Notes**

**Dynamic Arrays**

A dynamic array is an array that increases in size when necessary – see the [first lab](#) for more details
**String Objects**

Your array should contain string objects. You can find information about the standard string class [here](#). The main things you need to know about strings for this assignment are:

- You can assign an existing string variable or a string literal to a string variable
  - e.g. `string str1 = "bob";` or `str1 = str2;` (where `str1` and `str2` are strings)
- You can use the normal comparison operators to compare strings
  - e.g. if (str1 < "abc") cout << str1 << " comes before abc in a dictionary"; (where `str1` is a string)
- You do not need to know anything about the underlying representation of strings; in fact, assignment and comparisons are pretty much all you need to do with strings in this assignment. The set operations should all be acting on strings, not on the characters that make up those strings.

**Set Operations**

The set operations required by the methods should be familiar from MACM 101. Recall that:

- A set cannot contain duplicate values
- The union of two sets, \( R \cup S \), is the set of values that appear in either \( R \) or \( S \) (or in both)
  - e.g. \( \{1,3,4,5\} \cup \{2,3,4\} = \{1,2,3,4,5\} \)
  - e.g. \( \{\text{cat, bat, rat, badger}\} \cup \{\text{elephant, bat, hamster}\} = \{\text{cat, bat, rat, badger, elephant, hamster}\} \)
- The intersection of two sets, \( R \cap S \), is the set of values that are common to both \( R \) and \( S \)
  - e.g. \( \{1,3,4,5\} \cap \{2,3,4\} = \{3,4\} \)
  - e.g. \( \{\text{cat, bat, rat, badger}\} \cap \{\text{elephant, bat, hamster}\} = \{\text{bat}\} \)
- The set difference of two sets, \( R - S \), is the set of values that appear in \( R \) but not in \( S \)
  - e.g. \( \{1,3,4,5\} - \{2,3,4\} = \{1,5\} \)
  - e.g. \( \{\text{cat, bat, rat, badger}\} - \{\text{elephant, bat, hamster}\} = \{\text{cat, rat, badger}\} \)

**Set Operations Discussion**

The set operations all involve three `StringSet` objects. Two of them are to be combined in some way (union, intersection or set difference) to create a third `StringSet` object which is to be returned. If you glance at the method descriptions, you may be wondering where these two objects to be combined are since the methods only have one parameter. But they are methods of the `StringSet` class so belong to a `StringSet` object – the other object is the calling object.

Here is some sample code showing how you might call the difference method.

```plaintext
StringSet sset1;
sset1.insert("cat");
sset1.insert("bat");
sset1.insert("rat");
sset1.insert("badger");
StringSet sset2;
sset2.insert("elephant");
sset2.insert("bat");
```
sset2.insert("hamster");
// Use the copy constructor to build a StringSet with sset1 – sset2
StringSet sset3(sset1.difference(sset2));
// Use the overloaded assignment operator to make a StringSet
// contain sset2 – sset1
StringSet sset4;
StringSet sset4 = sset2.difference(sset1);

**Main Function**

You should write a main function in a separate file from your `StringSet` class; you should use your main function (and other test functions) to test your class methods. You should not submit the file containing the main function.

I would strongly suggest compiling and running your program after you complete each method (or couple of methods), do not try to complete the entire class before testing it as this will almost certainly end up being considerably more work.

**Assessment**

The assignment is out of 61.

- Class definition – 4 marks
- Methods: 4 marks each – 44
- Comments – 5 marks
- Naming – 5 marks
- Indentation – 3 marks

**Submission**

You should submit your assignment online to the [CoursSys submission server](#). You must submit your `StringSet.h` and `StringSet.cpp` files separately as indicated on the submission page, please read the documentation on the site for further information. The assignment is due by 11:59pm on Monday the 27th of January.

If you are unable to complete one of the methods, make sure that any calls to that method will still compile and run. For example, let's assume that you couldn't complete the `intersection` method. The definition shown below returns a value of the appropriate type (a `StringSet`) and will allow our tests to complete, even though the method does not return the correct value.

```cpp
// Stub for Intersection method
StringSet intersection(const StringSet &) const {
    StringSet result;
    return result;
}
```
CMPT 225 Home

John Edgar (johnwill@sfu.ca)