CISP 400 – Lab Assignment #11
Due December 20, 2018 at 11:59 PM
NO EXTENSIONS ARE AVAILABLE FOR THIS ASSIGNMENT UNDER ANY CIRCUMSTANCE

This exercise is an attempt to work with a relatively simple template, as well as using the Standard Template Library (STL) from C++. Although this is only intended as a primer, it should give you the basics and give you an idea of the power and functionality available with these features.

This lab involves:

- Templates.
- The Standard Template Library (STL)

CLASS: DataList

The main class that we will write, we will call DataList. It accepts one type parameter (template parameter, declared using the template keyword, which we shall refer to as TD) so that it may be customized. Strictly speaking, the only requirement for this class or data type is that it must be compatible with the << operator for cout, but we will only be using it with data types that already have this functionality (string and int).

It will contain, as a private member, one vector, that we will name data, that uses type TD.

It will have a virtual destructor that is empty. It does not need a constructor, but you can provide a blank one if you prefer.

It will have several public member functions.

Function: append – this function will take an element of type TD and will add it onto the data vector. It must use a method that increases the memory available to the vector if there is insufficient room currently there. It should be able to function indefinitely, e.g. it does not have some specific limit set by the programmer, such as 5 or 6. Technically it does have a limit, but you do not need to worry about that for our purposes.

Function: printContents – this function will use an appropriate iterator to progress through the vector. At each element, it will use cout to print each element, on its own separate line. It will continue until it runs out of elements to print, and then it will exit. **NOTE:** If you are having problems with declaring the iterator, see the TIPS section.

Function: sortContents – this function will take the data vector and sort it using the STL sort function.

Function: scrambleContents – this function will take the data vector and shuffle it using the STL shuffle function.

Function: isPresent – this function takes one parameter of type TD, and uses the STL to search the vector for the appropriate value. It returns a bool that is true if the parameter has been found, or false if it has not been found.

In **addition** to the class member functions, we will have one major utility function, which will be an independent template function, which is not part of our template class.

FUNCTION: processList

This function will be a template function that takes one type parameter that we will refer to as type T. The function itself takes two parameters of whatever type is specified by T. These parameters must not be passed by reference, because we will be passing constants to them, although in a more complicated situation involving classes being used as the data type, passing by reference might be much more desirable. One will be named 'stop,' which will be used to determine when we stop taking input. The other will be named 'findit.' Remember, both are of type T, which the template system will compile into whatever data type the user desires.

The function will have a local variable, a DataList instance, which takes the template type specified by T.

This function will have an input loop. It will take input, of type T, from the user. It will check to see if it is equal to the 'stop' variable, e.g. if we are working with int, and the stop value is -999, then the program will compare user input to stop, and check if the user input -999. If it is, then the input loop terminates. If not, it will append whatever data it has to the DataList object using the appropriate public member function.

Once it is done, it will print out a copy of the contents of the DataList.

It will then scramble (randomly shuffle) the DataList, and print out the contents.

It will then sort the DataList and print out the contents.

Finally it will search the DataList, and report to the user whether or not the 'findit' value has been found in the data list.

Main function:

This function will have several constants defined (or you can define them globally if you would like):

```
const string SEARCH_VALUE_STR = "banana";
const string STOP_VALUE_STR = "END";
const int SEARCH_VALUE_INT = 5;
const int STOP_VALUE_INT = -999;
```

This function will have a brief menu that asks the user what type of data they want to input – int or string. It then passes the appropriate const values to the processList function (see the specific name for what is appropriate, and match it to the name of the parameter), and exits.

TIPS:

- I have provided many examples and a lot of example code. A great deal more is available on the Internet. Make use of it.

- Do not try to make this program particularly fancy. Be straightforward. There are a lot of ways to go off into nowhere with templates. Considering this is probably your first time using them it's best just to get it done and make sure it's right.
- Less is often more.
- The "type" parameter is the parameter in angular brackets that you declare with the template keyword. Usually it is referred to as 'T' in simple situations involving templates. I have referred to it as type 'TD' for the DataList class, and type 'T' for the processList() template function. Remember to pass the type parameter to the appropriate places, e.g. make sure you use the type parameter in the code to processList to declare your instance of DataList.
- Put all of this code in one file.
- The compiler does an extremely bad job of providing feedback when there are template problems, especially with the STL. Train yourself to ignore the error messages that do not specifically mention your code. You are seeing a lot of detail that is of no practical use to you.
- The only time you should write code to differentiate between int and string should be the main() function. As <u>everything else</u> is either a template function or a template class, you should be able to write code for both int and string only once.
- Do NOT write separate code for each data type. This will receive little or no credit. It would be better to provide partially functioning template code than perfect code that does not properly use templates.
- Remember to declare all iterator objects for STL containers (vector, etc.) using the scope operator, passing along appropriate type parameters, and to use the typename keyword in front of the declaration as needed.
- See the slides on how to use the 'shuffle' STL function. You will probably need a seed for the random number generator. In order to get this, add #include <time.h> to the top of the program, and when you need the value, use a call to the 'time()' function. Specifically, use 'time(NULL)', since we do not want to pass it a specific parameter for our purposes. This is not perfect C++, as it uses a C library function, but it is not worth the time digging around to find the best C++ way to do this. If all else fails, just give a seed of value '1', as in, the literal integer value 1 (this will mean that the randomization order will always come out exactly the same for a given input, but that's OK for this).
- Remember: for the searching, sorting, and scrambling STL functions, the .begin() and .end() member functions are your friends. Also look at the examples I have given for a for-loop involving iterator objects and these functions.
- Use the 'using namespace std' declaration at the beginning of the program, *even if you do not usually use it*. STL can be very unforgiving if you make errors during compilation and otherwise you will be chasing down scope operators left and right. You will not be marked down if you do not do this, but I do not want to spend an hour untangling the error messages because of a typo resulting from something that simple, and I do not want you to take the time to do that, either.
- If you are having problems with declaring the appropriate vector iterator in the printContents() function, consider using the 'typename' keyword in front of the iterator declaration. C++ can sometimes get confused, when using templates within templates, in terms of what it should treat as a data type, and the typename keyword can sometimes clarify the issue.