CS515 Assignment 6

The purpose of this assignment is to give you an introduction to creating container iterators. You will write an improved version of the Set ADT based on Lab 5, and you will also write an iterator for this class. Download all starter files from `~cs515/public/6P`.

In Lab 5, you implemented a simple set ADT using a sorted doubly-linked list with head and tail sentinels. Our set container is used for the storage and retrieval of data from a collection in which the values of the elements contained are unique and these values are ordered.

Now you are asked to implement a set that supports the following new functionalities:

- Iterators for accessing set elements.
- Overloaded operators for set equality, set difference, set join and set intersect operations.

What is an iterator?

Each C++ standard STL container has an iterator associated with it. An iterator is sort of like a pointer or an index into an array that lets you access an item in the array, then move to the next (or previous) item). The iterator helps users to manipulate the data collection in a secure and easy way, without worrying about the implementation details of the collection itself. Like STL containers, our Set container should be associated with an iterator. The user of a Set object should not directly access the underlying data structure that implements the Set, but should do so only through the container's public iterator.

Refer to STL documentation to familiarize yourself with STL container iterators. The code segments below show examples of using a Set iterator and reverse iterator.

Example 1: Usage of an iterator. This code segment prints out all the elements from the set in sorted order.

```
Set s1;
// creates an iterator set to the beginning of the collection
Set::Iterator it = s1.begin();

// use iterator to traverse the collection of elements
// operators are overloaded to support pointer-like behaviors.
while( it != s1.end() ){
    cout << *it;
    it++;
}
```

Example 2: Usage of a reverse iterator. This code segment prints out all the elements from the set in the reverse-sorted order.

```
Set s1;
// creates an iterator set to the "reverse-beginning" of the collection
Set::Reverse_Iterator rit = s1.rbegin();

// use iterator to traverse set elements in reverse order
while( rit != s1.rend() ){
    cout << *rit << " ";
    rit++;
}
cout << endl;
```

Important: the value of an element in a set may not be changed directly by the user. Instead, you must delete old values and insert elements with new values. However, this is not enforced by the implementation of the Set container. For example, a user can technically dereference an iterator and use it as an *lhs* value, e.g. use `*it = 100` to modify the element and break the sorting order of the collection. It is indeed the user's responsibility not to modify set elements directly through dereferencing the iterator.

How do I implement an iterator?

An iterator is typically implemented as an inner class of its associated container. It represents a pointer abstraction but without exposing the actual structure of the container.

The dereference (*) and pointer-increment and -decrement operators (++ and --) are overloaded so an iterator behaves like a pointer:

- The dereference operator will return the element in the container. Suppose the set were a string set, then dereference its associated iterator would return us the current string object.
- Both increment and decrement operators (++ and --) have an iterator return type. The prefix and postfix forms return the iterator before and after the operation, respectively.

You also need to include the logic to compare two iterators. Iterator comparison is conceptually similar to pointer comparison: the "pointer" values are compared, rather than the "pointee" values.

How does an iterator bind with its associated container?

A Set iterator is associated with the Set container through two public methods of the container. The methods `begin()` and `end()` of the container will return the beginning and the end of the collection. Based on a simplified version of STL iterator design, we will have

- **begin()** Returns an iterator referring to the first element in the set container.
- **end()** Returns an iterator referring to the "past-the-end" element in the set container. The *past-the-end* element is the theoretical element that would follow the last element in the set container. In our case, it points to one step beyond the last element in the list.

Similarly, a Set reverse iterator is associated with the Set container through two public methods of the container.

- **rbegin()** Returns a reverse iterator pointing to the last element in the container (i.e., its reverse beginning).
- **rend()** Returns a reverse iterator pointing to the theoretical element right before the first element in the set container (which is considered its reverse end)

These methods are used to specify the range of the iterator's traversal in the collection. The range between rbegin() and rend() contains all the elements of the container (in reverse order).

More set operations

Since the Set now supports iterators, you can use the iterators to implement some set operators:

|	for set union
&	for set intersect.
–	for set difference (shown in Figure 1)
==	for logical equality of sets

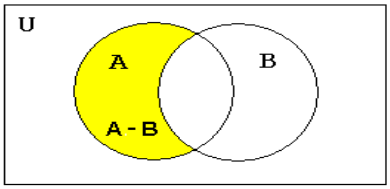

Figure 1. Set difference.

To ensure information hiding, these operator functions are not declared as friends of the Set class. Instead, they access the private set elements indirectly through an iterator.

You should not modify the header file in `set.h`. Your implementation should be written in a single source file named **`set.cpp`**. A sample test file is provided in the starter code. However, write your own tests and test your program thoroughly.

Note: The Set range constructor will take a range of ELEMENT_TYPE elements passed as an array and construct a set from the elements within the range specified by the first and last pointers.

```
Set (ELEMENT_TYPE* first, ELEMENT_TYPE* last);
```

Submission:

- Submit **`set.cpp`**.
- You should fill out the README file and submit it as well. Submit the following files to the appropriate assignment on Mimir:

  `set.cpp README`

- Do not turn in executables or object code. Programs that produce compile time errors or warnings will receive a zero mark (even if it might work perfectly on your home computer).
- Be sure to provide comments in your program. You must include the information as the section of comments below:

```
/**     CS515 Assignment X
File: XXX.cpp
Name: XXX
Section: X
Date: XXX
Collaboration Declaration: assistance received from TA, PAC etc.
*/
```

Some notes on grading:

- Programs are graded for correctness (output results and code details), following directions and using specified features, documentation and style.
- To successfully pass the provided sample tests is not an indication of a potential good grade; to fail one or more of these tests is an indication of a potential bad grade.
- You must test thoroughly your program with your own test data/cases to ensure all the requirements are fulfilled. We will use additional test data/cases other than the sample tests to grade your program.
- Here is a tentative grading scheme.

set test run 1	10
set test run 2	20
set test run 3	20
set test run 4	20
set test run 5	15
Valgrind check	15
others	
Use STL container	-100