README

CSC 254
Assignment 5
Shuyang Liu, Nina Bose

Included Files:
README.pdf (this file)
sets.h
sets.cc
tests.cc

Running Instructions
<*enter directory in which code is contained*>
$ g++ -o sets sets.h tests.c std=c++11
$ ./sets

Problem Description:
For this assignment, we were to implement templates for various types of sets in C++ without using the standard library collections.

Solution Description:
carray_simple_set
For carray_simple_set, we created a boolean array representing the values contained in [l, h). Thus, the array was of the size, h-l, with each element representing a value in [l, h). For our += operator, we make sure the parameter is within the bounds using the given comparitor. If it is, we changed the value of the boolean located at that position to true, indicating that the element at that spot is present. Similarly, in -=, we simply set the value of the boolean to false. In the contains method, we check to see whether the boolean at the specified position is true, in which case, we also return true. Otherwise, the element is not contained within the set, and we return false.

hashed_simple_set
We used the hash function given in order to insert elements into an array that was of size p, which was the smallest prime number larger than n, the user given max. For contains, we first use the hash function in order to look at the spot in the array at which the element should be located, but if we don't find it there, we iterate over the entire array to make sure it wasn't placed in a different spot due to collisions.

bin_search_simple_set

Although we originally created a tree to store the values we received, we we having some trouble with pointers, and thus decided to revert back to an array. However, in order to meet the runtime requirements given in the specifications, we made sure to use a sorted array at each stage of the process. That is, in both += and -=, we assume we have a sorted array, and then look through that element by element to see where the value we are trying to insert should fit / be removed from. Once we find the position, we shift the existing array by one to the left, or the right, respectively. This provides an O(n) runtime. However, we always check whether an element is larger than the largest element first, in which case, we simply append it to the end. This provides an amortized runtime of O(1) when elements are inserted in sorted order. For contains, we conduct a binary search on the array in order to find the element, which takes O(log(n)) time. We could have improved the time of += and -= by either using a tree, or performing binary search, as well.

carray_range_set and hashed_range_set
For both of these cases, we used a relatively simple implementation in which we inserted/removed all of the elements that were included in the given range into the simple_set implementation of each respective range set one at a time. We did not change the implementation of contains between the range_set and simple_set for these two templates.

bin_search_range_set
We struggled with the implementation of this structure originally as we tried to insert ranges into the bin_search_simple_set. However, we ran into a lot of type issues, and thus decided that although we would inherit from bin_search_simple_set as per the instructions, we would reimplement the += and -= operators without using the simple_set implementations. Instead, we created an array of discrete ranges to which we added discrete ranges either by simply adding the range if it was already discrete, or by merging several ranges to make them discrete. We used a similar idea to implement delete. We override contains in order to check each range to see whether the range contains the given element.

<u>Limitations:</u>

1. bin_search_range_set
    a. This structure currently does not accept ranges of string (however, ranges of chars are accepted).
    b. Although we inherit from bin_search_simple_set, we do not use the += or -= operators from bin_search_simple_set in our implementation of bin_search_range_set += and -= operators.

2. We commented out the static_assertion in incrementer class which checked whether the incoming type was of integral type. This is because the assertion failed when we tried using enums.

3. We also commented out the check invalid ranges in the ranges constructor. That is, the programmer may now create a range with a lower bound that is greater than the upper bound

(however, such ranges would not be accepted into our bin_search_range_set. It will be a soft rejection - no error will be thrown, but the range will also not exist within our set).