Sets

By Varick Erickson

Introduction

For this project you will implement a Set using a hash based chain implementation (each bucket can contain multiple elements).

```
template<class T>
class SetT
{
public:
	SetT();
	SetT(int numBucks);
	~SetT();
	void Add(T elem);
	void Remove(T elem);
	bool Contains(T elem);
	void Union(SetT otherSet);
	void Difference(SetT otherSet);
	void Intersection(SetT otherSet);
	void Filter(bool fnc(T elem));
	void Traverse(void visit(T& elem));
	int Size() { return numElems };

	SetT operator+(T elem);		// Add
	SetT operator+(SetT& other);	// Union
	SetT operator*(SetT& other);	// Intersection
	SetT operator-(T elem);		// Remove
	SetT operator-(SetT& other);	// Difference

private:
	forward_list<T>** buckets;  // An array of forward_list's (ie, each index is a
forward_list pointer)
	int numBuckets;
	int getHashIndex(const T& elem);
	int numElems;
	// Any other private functions and variables you need
};
```

Your job is to implement each of the methods in the header class. You are also responsible for creating a test driver (similar to Chunklist) that takes input files and generates output files. You only need to test using SetT<int> sets.

forward_list

Each bucket contains should contain a linked list of elements of type T. For this assignment, you will be using a Standard Template Library (STL) implementation of a singly linked list. Here is a code snippet that should be useful:

```
#include <iostream>
#include <forward_list>

int main ()
{
  std::forward_list<int> mylist;

  mylist.push_front(42);
  mylist.push_front(55);
  mylist.push_front(32);
  mylist.push_front(5);

  std::cout << "mylist contains:";
  for ( auto it = mylist.begin(); it != mylist.end(); ++it )
    std::cout << ' ' << *it;

  std::cout << '\n';

  return 0;
}
```

Operator Overloading

Operator overloading allows the programmer to override the behavior of operators such as +, -, >, and <. While this may sound complicated, it actually straight forward. The following shows how we overload the + operator to add and element to the set:

```
template<class T>
SetT<T> SetT<T>::operator+(T elem)
{
	this->Add(elem);
	return result;
}
```

Note that the operator behavior is based on the type of the argument on the right side. In this case, the argument is of type T. If the argument was of type SetT, then the + would indicate Union.

Test Driver

You should have a test driver similar to the list driver from the previous program. In particular, it should support the following commands:

- Add (this should test add and the + operator)
- Remove (this should test remove and the - operator)
- Intersection (this should test intersection and the * operator)
- Union (this should test union and the + operator)
- Difference (this should test union and the + operator)
- Size
- Print

Part 3: Questions

1. What is the advantages/disadvantages of a tree approach rather than a hash based approach?
2. What is the average big O of:

    - add
    - remove
    - contains
    - intersection
    - union
    - difference
    - size

Deliverables

- Your template class implementations for SetT
- Your test driver
- Input to the test driver demonstrating functionality
- The output file generated from the test driver
- Answers to the questions for part 3