CSCI 104L Lecture 6: ADTs and the STL

Abstract Data Types

- If we are precise about what we want to do (the operations we want to implement), then we have specified an **Abstract Data Type** or ADT.
- A **List** is defined by the following operations, where T denotes any one type (such as int, string, etc).
  1. void insert (int position, T value): inserts value at the specified position, moving all later elements one position to the right.
  2. void remove(int position): removes the value at the specified position, moving all later elements one position to the left.
  3. void set(int position, T value): overwrites the specified position with the given value.
  4. T get (int position): returns the value at the specified position.
- A **Set** (sometimes referred to as a Bag) supports the following:
  1. void add (T item): adds item to the set.
  2. void remove (T item): removes item from the set.
  3. bool contains (T item): determines whether the set contains item.
- A **Map** (sometimes referred to as a Dictionary) associates values with keys. keyType can be any individual data type, as can valueType.
  1. void add (keyType key, valueType value): adds a mapping from key to value.
  2. void remove (keyType key): removes the mapping for key.
  3. valueType get (keyType key): returns the value that key maps to.
- A List cares about order, a map associates keys and values, and a set only determines whether a thing is contained inside or not.

Array Lists

Analyze the runtime analysis for each of the operations of a List, when implemented with a Linked List.

Now instead consider implementing a List with an Array.

Question 1. What is the runtime for insert/remove/get on a sorted array? On a sorted linked list?

STL's map class

```
#include<map>
#include "student.h"
int main() {
  map<string, Student> slist1;
  Student s1("Tommy", 86328);
  slist1["Tommy"] = s1; //associate the string Tommy with his student record.
  slist1.erase("Tommy");
  return 0;
}
```

STL's pair struct

```
std::pair<string, int> mypair("Tina", 1);
cout << mypair.first << " " << mypair.second << endl;
std::pair<char, double> p2('c', 2.3);
```

STL's iterator class

```
map<int, string> m;
map<int, string>::iterator it;
for (it = m.begin(); it != m.end(); ++it) {
  cout << it->second << endl;
}
it = m.find(42);
if (it != m.end()) cout << "meaning of life found: " << it->second << endl;
```

- The data structure has two public functions: begin(), which returns an iterator at the start of the data, and end() which returns an iterator at the end of the data.
- Every iterator in the STL is implemented in the same manner, so that you can always use an iterator for a data structure, even though you may not understand how the data structure works.
- Think of it like a pointer (it is not a pointer, but it has overloaded operator* to act like one).
- Also, think of the end() function as returning one PAST the end of the data structure, so the above for-loop works properly.