STL’s map class

```cpp
#include <map>
#include "student.h"

int main() {
    map<string, Student> slist1;
    Student s1("Tommy", 86328);
    Student s2("Jill", 54982);
    ... 
    slist1["Tommy"] = s1; //associate the string Tommy with his student record.
    string myname = "Jill";
    slist1[myname] = s2;
    ... 
    Student s3 = slist1["Tommy"]; 
    slist1.erase("Jill");
    return 0;
}
```

STL’s pair class

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

STL’s iterator class

```cpp
map<int, string> m;
... 
map<int, string>::iterator it;
for (it = m.begin(); it != m.end(); ++it) {
    cout << 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.
- The iterator is a custom class defined within the scope of the data structure it iterates over. It overloads operator== and operator!= (so you can check if your iterator is at the end), operator++ (so you can get to the next piece of data), and operator* (so you can look at the value the iterator is currently sitting at).
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.

Exceptions

What would be the correct way of handling the following request?

```cpp
LinkedList<int> *LL = new LinkedList;
for (int i = 0; i < 10; i++) LL->prepend(i);
cout << LL->get(15) << endl;
```

Put this in LinkedList.cpp:

```cpp
#include <exception>
#include <stdexcept>

... if (position >= this->size()) throw logic_error("position was too large!");
```

A thrown exception will propagate up through the program stack until it reaches a piece of code designed to handle it. If no such code is found, the program terminates.

The user should do this:

```cpp
try {
    cout << LL->get(15) << endl;
    cout << "Printed successfully!" << endl;
} catch (logic_error &e) {
    cout << "A logic_error occurred!" << endl;
    cout << e.what();
} catch (exception &e) {
    cout << "General exception" << endl;
}
```
Keeping a List Sorted

Suppose we want to have a list class where the data is stored in sorted order. Our list previously had the following functions:

```cpp
void set(int pos, const T& data);
const T& get(int pos) const;
void insert(int pos, const T& data);
void remove(int pos);
```

Note the following:

- Nothing need change for get or remove.
- Insert needs to be modified so that it doesn’t accept a position.
- There is no point in having a Set function; accordingly, our interface can be:

```cpp
const T& get(int pos) const;
void insert(const T& data);
void remove(int pos);
```

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

**Question 2.** On a sorted linked list?

### Interpolation Search

```cpp
T *a;
int InterpolationSearch(T x, int l, int r) {
    if (r < l) return -1;
    if (a[r] == a[l]) {
        if (x == a[r]) return r;
        return -1;
    }
    int m = (x-a[l])/(a[r]-a[l])*(r-l)+l;
    if (a[m] == x) return m;
    if (a[m] < x) return InterpolationSearch(x, m+1, r);
    return InterpolationSearch(x, l, m-1);
}
```

To get a good result, we are assuming that the elements of the array are “roughly evenly spread.” If you assume this is true (for a precise mathematical definition of this statement, which we are not providing) then you would find an average case runtime of \(O(\log \log n)\).