Chapter 25
Standard Template Library

STL Components

- Containers – A collection of data
- Iterators – Things that go through the data
- Algorithms – Data manipulation functions

Containers

- vector – Looks like an array
- deque – Like vector but allows faster insert and delete to the middle of the container.
- list – Double linked list (no random access)
- set – Unique, ordered, set of items
- multiset – Set with no unique restriction
- map – Associative array with unique keys
- multimap – Like map, but no unique restriction

Iterators

Iterators are used to go through a container

- Forward iterator
- Reverse iterator
- Random access iterator

Not all iterators work on all containers

Algorithms

- find – Locate an item in a container
- count – Count a number of matching items
- equal – Check to see if containers are the same
- copy – Copy container contents
- reverse – Reverse the elements in a container

Class List Design

- Container to use: set
  - Each student in uniquely identified

Defining the `class_set` variable:

```
#include <set>
#include <string>
std::set<std::string> class_set;
```

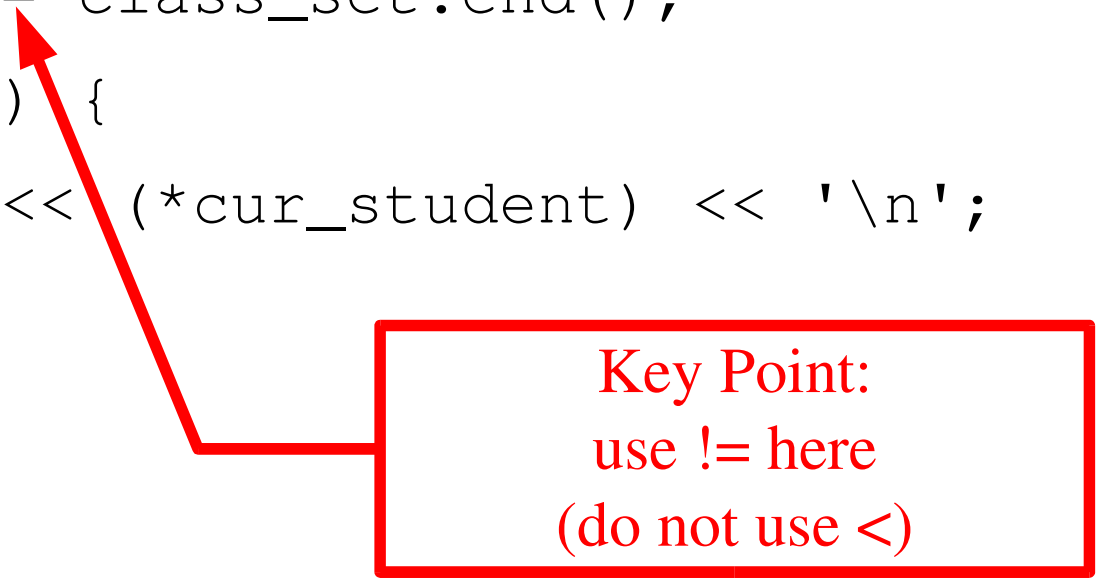
Adding students to the class_set

```
while (! in_file.eof()) {
    std::string student;

    in_file >> student;
    class_set.insert(student);
}
```

Iterating through the student list

```
// Print the list of students
std::set<std::string>::const_iterator
      cur_student;
for (cur_student = class_set.begin();
      cur_student != class_set.end();
      ++cur_student) {
         std::cout << (*cur_student) << '\n';
}
```



Using an algorithm

- Using the `foreach` algorithm for listing the class

```
#include <algorithem>
// ....
static void write_student(
    std::set<std::string>::const_iterator&
    cur_student) {
    std::cout << (*cur_student) << '\n';
}
// ....
   foreach(class.set.begin(), class_set.end(),
           write_student);
```

Allowing multiple students with the same name

```
std::multiset<std::string>
    class_set;
```

Note: **typedef** would be very useful when dealing with the STL.

```
typedef std::multiset<std::string>
        class_set_type;
```

For simplicity **typedef** is good. For learning (in this chapter) it's not, so it's not used. Use it in real life.

Waiting List

- Waiting list is a first come, first served list.
- Can be implemented with a STL `list`.

```
#include <list>
std::list<std::string>
     waiting_list;

// Adding a student
waiting_list.push_back(student);

// Removing a student
student = waiting_list.top();
waiting_list.pop_front();
```

Storing names and grades

- Solution: Create a map where the key=student and value=grade

```
#include <map>
template
    std::map<std::string, char>
    student_roster;
```

Adding a student

- The `pair` function creates a key/value item to be inserted into the list.

```
student_roster.insert(
    std::pair(
        std::string("John Smith"),
        'A')
);
```

Finding a student

```
std::map<std::string, char>::
    const_iterator record_loc;
record_loc = std::find(
    student_roster.begin(),
    student_roster.end(),
    std::string("John Smith"));
```

Now that we've found the student

```
if (record_loc == student_roster.end())
    std::cerr << "No such student\n";

std::cout << "Student: " <<
    record_loc->first <<
    " Grade: " <<
    record_loc->second << '\n';
```

Putting it all together (class program)

```
/*****************************************
 * class_stuff -- A simple class to handle *
 *   students and grades.                  *
 *****************************************/
#include <iostream>

#include <string>
#include <vector>
#include <map>
#include <list>

#include <algorithm>

const unsigned int MAX_STUDENTS = 5;
// Max number of students per class
// Set low for testing
```

class (continued)

```
class class_stuff {
    public:
        // A set of grades
        typedef std::vector<int> grades;

        // Roster of current class
        std::map<std::string, grades> roster;

        // People waiting on the list
        std::list<std::string> waiting_list;
    public:
        // Constructor defaults
        // Destructor defaults
        // Copy constructor defaults
        // Assignment operator
```

class (continued)

```
    public:
        void add_student(const std::string& name);
        void drop_student(const std::string& name);
        void record_grade(const std::string& name,
            const int grade,
            const unsigned int assignment_number
        );
        void print_grades();
    private:
        // Insert a student into the class
        void new_student(
            // Student to add to the class
            const std::string& name
        )
        {
            grades no_grades;// Empty grade vector
            roster.insert(
                std::pair<std::string, grades>(
                    name, no_grades));
    }
};
```

class (continued)

```
void class_stuff::add_student(
    // Name of the student to add
    const std::string& name
)
{
    if (roster.find(name) != roster.end())
        // Already in the class, don't reuse
        return;

    if (roster.size() < MAX_STUDENTS) {
        // Class has room, add to class
        new_student(name);
    } else {
        // No room, put on waiting list
        waiting_list.push_back(name);
    }
}
```

class (continued)

```
void class_stuff::drop_student(
    const std::string& name  // Name of the student to drop
)
{
    // The student we are probably going to drop
    std::map<std::string, grades>::iterator
        the_student =  roster.find(name);

    if (the_student == roster.end())
        return; // Student is not in the class

    roster.erase(name);
    // Add a person from the waiting_list if
    // there's anyone waiting
    if (waiting_list.size() > 0) {
        std::string wait_name = waiting_list.front();
        waiting_list.pop_front();
        new_student(wait_name);
    }
}
```

class (continued)

```
void class_stuff::record_grade(
    const std::string& name,  // Name of the student
    const int grade,      // Grade of this assignment
    // Assignment number
    const unsigned int assignment_number
)
{
    std::map<std::string, grades>::iterator
        the_student = roster.find(name);

    if (the_student == roster.end())
    {
        std::cerr << "ERROR: No such student " <<
                     name << '\n';
        return;
    }
    // Resize the grade list if there's not enough room
    if (the_student->second.size() <= assignment_number)
       the_student->second.resize(assignment_number+1);

    the_student->second[assignment_number] = grade;
}
```

class (continued)

```
void class_stuff::print_grades(){
    // Student names sorted
    std::vector<std::string> sorted_names;

    // The student we are inserting into the
    // sorted_names list
    std::map<std::string, grades>::iterator
        cur_student;

    for (cur_student = roster.begin();
         cur_student != roster.end();
         ++cur_student) {
       sorted_names.push_back(cur_student->first);
    }
    std::sort(sorted_names.begin(),
              sorted_names.end());
```

class (continued)

```
    // The current student to print
    std::vector<std::string>::const_iterator cur_print;

    for (cur_print = sorted_names.begin();
         cur_print != sorted_names.end();
         ++cur_print)
    {
        std::cout << *cur_print << '\t';

        // The grade we are printing now
        grades::const_iterator cur_grade;

        for (cur_grade = roster[*cur_print].begin();
             cur_grade != roster[*cur_print].end();
             ++cur_grade)
        {
            std::cout << *cur_grade << ' ';
        }
        std::cout << '\n';
    }
}
```

class (continued)

```
int main()
{
    // A class for testing
    class_stuff test_class;

    test_class.add_student("Able, Sam");
    test_class.add_student("Baker, Mary");
    test_class.add_student("Johnson, Robin");
    test_class.add_student("Smith, Joe");
    test_class.add_student("Mouse, Micky");

    test_class.add_student("Gadot, Waiting");
    test_class.add_student("Congreve, William");
```

class (continued)

```
std::cout << "Before drop " << std::endl;
test_class.print_grades();
std::cout << "\n";

test_class.drop_student("Johnson, Robin");

std::cout << "After drop " << std::endl;
test_class.print_grades();
std::cout << "\n";
```

class (continued)

```
    int i;

    for (i = 0; i < 5; ++i)
    {
        test_class.record_grade("Able, Sam",       i*10+50, i);
        test_class.record_grade("Baker, Mary",     i*10+50, i);
        test_class.record_grade("Smith, Joe",      i*10+50, i);
        test_class.record_grade("Mouse, Micky",    i*10+50, i);
        test_class.record_grade("Gadot, Waiting", i*10+50, i);
    }

    std::cout << "Final " << std::endl;
    test_class.print_grades();
    std::cout << "\n";

    return (0);
}
```

Practical Information

- Getting the types right can be tricky
- Error messages extremely verbose
  - They include information on the internal structure of the STL.
  - The do not really tell you what's wrong
- For more information:

  http://www.sgi.com/tech/stl/index.html.