

STL I

- STL: Standard Template Library provides lots of template based tools to make programming easier
- do not write tools already there in STL unless you consider it as a programming exercise, because otherwise you'll lose efficiency

STL I

- lets learn by an example: dealing with vector of strings

- to use the vector (and the string) class use this:

```
#include <iostream>
#include <vector>
#include <string>
using std::string;
using std::vector;
```

- create a vector of strings (here #define N_SMALL 3):

```
vector<string> vec_strings(N_SMALL);
```

- fill in the values of the vector:

```
string tmp1 = "Method 1 ";
// tmp2 has enough length to hold 1, 2, ... in string form
char tmp2[10];
// using the overloaded operator + for strings
for (int ii = 0; ii < N_SMALL; ++ii) {
    sprintf(tmp2, "%d", ii);
    vec_strings.at(ii) = tmp1 + tmp2;
}
```

- note the use of member function at()

STL I

- print the values:

```
cout << "Printing with for loop" << endl;
for (int ii = 0; ii < N_SMALL; ++ii)
    cout << ii << ": " << vec_strings[ii] << endl;
```

- note the overloaded operator []
- use the overloaded operators [] and =: fill in the values again!

```
tmp1 = "Method 2 ";
// using the overloaded operators [ ] and =
for (int ii = 0; ii < N_SMALL; ++ii) {
    sprintf(tmp2, "%d", ii);
    vec_strings[ii] = tmp1 + tmp2;
}
```

STL I

- use of iterators:
- print the values, this time using iterators:

```
cout << "Printing with iterators" << endl;
int ii = -1;
for (vector<string>::iterator iter = vec_strings.begin( );
     iter != vec_strings.end( ); ++iter)
    cout << ++ii << ": " << *iter << endl;
```

- iterators behave like pointers: you can do “++”, “*” and “->” on them as if they were pointers
- one could also use `const_iterator`, here it makes sense to use these, since we are just printing:

```
cout << "Printing with iterators" << endl;
int ii = -1;
for (vector<string>::const_iterator iter = vec_strings.begin( );
     iter != vec_strings.end( ); ++iter)
    cout << ++ii << ": " << *iter << endl;
```

STL I

- use of built-in algorithms for sorting:
- suppose you have the following class:

```
class Student {
    friend ostream & operator<< (ostream &ostream, Student const &st);
    friend bool compareStudentByID (Student const &st1, Student const &st2);

private:
    string firstName;
    string lastName;
    double GPA;
    int ID;

public:
    Student (void);
    Student (string firstName, string lastName, int GPA, int ID);
    int operator< (Student const &st) const;
};
```

STL I

- some things to note:

- the top of the file having the Student class should look like:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>

using std::ostream;
using std::cout;
using std::endl;
using std::string;
using std::vector;
using std::operator<<;
using std::operator<;
```

- we have two constructors
- the functions `operator<()` and `compareStudentByID()` sort by GPA and ID respectively

STL I

- the implementations of

```
Student (string firstName, string lastName, int GPA, int ID);
```

and of

```
friend ostream & operator<< (ostream &ostream, Student const &st);
```

are easy and we have seen similar stuff before

- lets look at:

```
int
Student::operator< (Student const &st) const
{
    int retVal;
    if (GPA < st.GPA)
        retVal = -1;
    else
        retVal = 1;
    return retVal;
}
Student::Student (void) { }
```

- we need to write Student() because we will use it later for creating a vector with “default” values

STL I

- the comparator function:

```
bool
compareStudentByID (Student const &st1, Student const &st2)
{
    int retVal;
    if (st1.ID < st2.ID)
        retVal = true;
    else
        retVal = false;
    return retVal;
}
```

STL I

- use of the `Student::operator<()` function to sort vector of Students:
 - create the vector and print it:

```
vector<Student> vec_students(N_SMALL);
string tmpsf = "FirstNameOfStudent";
string tmpsl = "LastNameOfStudent";
char tmpc[10];
for (int ii = 0; ii < N_SMALL; ++ii) {
    sprintf(tmpc, "%d", ii);
    vec_students[ii] = Student(tmpsf + tmpc,
                               tmpsl + tmpc,
                               4 - ii,
                               100 + ii);

    ii = -1;
    cout << "Printing vector of Students" << endl;
    for (vector<Student>::const_iterator iter = vec_students.begin( );
         iter != vec_students.end( ); ++iter)
        cout << ++ii << ": " << *iter << endl;
}
```

STL I

- note one use of `sort()` (its part of the `#include <algorithm>` header):
- there are other “generic” algorithms like this
- sort it via the overloaded operator `operator<()` and print it:

```
sort(vec_students.begin( ), vec_students.end( ));  
ii = -1;  
cout << "Printing vector of Students after sorting by GPA" << endl;  
for (vector<Student>::const_iterator iter = vec_students.begin( );  
     iter != vec_students.end( ); ++iter)  
    cout << ++ii << ": " << *iter << endl;
```

STL I

- note another use of `sort()`:
- sort it via the `compareStudentByID()` function and print it:

```
sort(vec_students.begin( ), vec_students.end( ), compareStudentByID);  
ii = -1;  
cout << "Printing vector of Students after sorting by ID" << endl;  
for (vector<Student>::const_iterator iter = vec_students.begin( );  
     iter != vec_students.end( ); ++iter)  
    cout << ++ii << ": " << *iter << endl;
```

Code Files

prog10.C