Lecture 11: STL Vectors
Text Reference: N/A
Standard Template Library (STL)

- Collection of data types and algorithms
- Not native or primitive, programmer defined
- Data structures in STL are
  - Containers: classes to store and organize data
  - Iterators: objects that behave like a pointer and that are associated with containers
Containers in STL

- **Sequence**: data is in a sequential fashion
  - Vector
  - Deque
  - List
- **Associative**: organize data with keys
  - Set
  - Multiset
  - Map
  - Multimap
Vector Container

• A vector:
  • Holds a sequence of elements
  • Stores elements in contiguous memory locations
  • Can use the array subscript operator [ ]

• Advantages to an array
  • Do not need to declare the size
  • Can add an element to a full vector
  • Vectors can report their size
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    vector<int> numbers (5);

    for (int i = 0 ; i < 5; i++)
        cin >> numbers[i];

    for (int i = 0 ; i < 5; i++)
        cout << numbers[i] << endl;

    numbers.push_back(25);

    for (int i = 0; i < numbers.size(); i++)
        cout << numbers[i] << endl;

    return 0;
}
Vector Functions

- `.at(i)` – returns the value of element at index `i`
- `.capacity()` – returns the maximum number of elements that can be stored without additional memory
- `.clear()` – removes all the elements from the vector
- `.empty()` – returns true if the vector is empty
Vector Functions

• .pop_back() – removes the last element from the vector
• .push_back(value) – stores value in the last element of the vector
• .reverse() – reverses the order of the elements in the vector
Iterators

- Iterators are objects that behave like cursors
- Used to access items stored in containers
- Each STL container object provides two member functions:
  - `begin()` – returns an iterator pointing to the vector’s first element
  - `end()` – returns an iterator pointing to the vector’s last element
Using Iterators

- To define an iterator object for vectors:

```cpp
vector<int>::iterator iter;
```
Vector Example

```cpp
#include <iostream>
#include <vector>  // needed to use vectors

int main ()
{
    std::vector<int> vect;  // Create a vector of int
    for (int x = 0; x < 10; x++) {
        vect.push_back (x*x);
    }

    // print everything using iterators.
    std::vector<int>::iterator iter = vect.begin ();
    while (iter != vect.end ())
    {
        std::cout << *iter << " ";
        iter++;
    }
    return 0;
}
```
Algorithms in STL

- Many algorithms including:
  - binary_search
  - count
  - for_each
  - find
  - max_element
  - min_element
  - random_shuffle
  - sort
4.3 Program Design Example: Rating the Field

Pretty Polly has no shortage of gentlemen suitors who come a’ courting. Indeed, her biggest problem is keeping track of who the best ones are. She is smart enough to realize that a program which ranks the men from most to least desirable would simplify her life. She is also persuasive enough to have talked you into writing the program.

Polly really likes to dance, and has determined the optimal partner height is 180 centimeters tall. Her first criteria is finding someone who is as close as possible to this height; whether they are a little taller or shorter doesn’t matter. Among all candidates of the same height, she wants someone as close as possible to 75 kilograms without going over. If all equal-height candidates are over this limit, she will take the lightest of the bunch. If two or more people are identical by all these characteristics, sort them by last name, then by first name if it is necessary to break the tie.

Polly is only interested in seeing the candidates ranked by name, so the input file:

<table>
<thead>
<tr>
<th>Name</th>
<th>Height</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>George Bush</td>
<td>195</td>
<td>110</td>
</tr>
<tr>
<td>Harry Truman</td>
<td>180</td>
<td>75</td>
</tr>
<tr>
<td>Bill Clinton</td>
<td>180</td>
<td>75</td>
</tr>
<tr>
<td>John Kennedy</td>
<td>180</td>
<td>65</td>
</tr>
<tr>
<td>Ronald Reagan</td>
<td>165</td>
<td>110</td>
</tr>
<tr>
<td>Richard Nixon</td>
<td>170</td>
<td>70</td>
</tr>
<tr>
<td>Jimmy Carter</td>
<td>180</td>
<td>77</td>
</tr>
</tbody>
</table>

yields the following output:

Clinton, Bill
Truman, Harry
Kennedy, John
Carter, Jimmy
Nixon, Richard
Bush, George
Reagan, Ronald