# The Standard Template Library Classes

## Lecture 35
## Sections 9.7, 9.8

Robb T. Koether

Hampden-Sydney College

Mon, Apr 23, 2018

# Outline

# The Standard Template Library

- The Standard Template Library (STL) was added to the C++ standard in 1994.
- It contains
  - Container classes - Objects that hold collections of objects.
  - Iterators - Objects that iterate through containers.
  - Generic algorithms - Objects that perform standard procedures on objects.

# Outline

# The Container Classes

- The basic container classes are
  - `vector`
  - `deque` (double-ended queue)
  - `list`
- The adaptor classes use container classes.
- The adaptor classes are
  - `stack`
  - `queue`
  - `priority_queue`

# The Container Classes

- Other container classes
  - `set`
  - `multiset`
  - `map`
  - `multimap`
  - `bitset`

# The Container Classes

- Visit the website

  `http://www.cplusplus.com/reference/`

  for a full description of the STL.

# Outline

# The `vector` Container Class

## `vector` Constructors

```
vector();
vector(vector& v);
vector(size_type sz, T& value);
vector(iterator first, iterator last);
```

- `vector()` – Constructs the default vector.
- `vector(vector)` – Constructs the copy vector.
- `vector(sz, value)` – Constructs a `vector` of size `sz`, filled with `value`.
- `vector(first, last)` – Constructs a `vector` with range of values given by the iterators `first` and `last`.
- We will use the `vector` class as an example.

# The `vector` Container Class

## The Assignment Operator

```
vector& operator=(const vector& v);
```

- **`operator=()`** – Assigns one `vector` to another.

# The `vector` Container Class

## The Capacity Functions

```
size_type size() const;
size_type max_size() const;
void resize(size_type sz, T value);
size_type capacity() const;
bool empty() const;
void reserve(size_type cap);
```

- `size()` – Returns the number of element in the vector.
- `max_size()` – Returns the maximum size possible.
- `resize()` – Changes the size to `sz`, filling in with `value`.
- `capacity()` – Returns current capacity.
- `empty()` – Determines whether the vector is empty.
- `reserve()` – Sets the capacity to `cap`.

# The `vector` Container Class

## Element Access Functions

```
reference operator[](size_type i);
reference at(size_type i);
reference front();
reference back();
```

- **`operator[]()`** – Returns the element in position `i`.
- `at()` – Same as **`operator`**`[](i)`, but with range checking.
- `front()` – Returns the element in the first position.
- `back()` – Returns the element in the last position.

# The `vector` Container Class

## Mutator Functions

```
void assign(size_type n, const T& value);
void assign(iterator first, iterator last);
void push_back(const T& value);
void pop_back();
```

- `assign(n, value)` – Replaces the contents with `n` copies of value.
- `assign(first, last)` – Replaces the contents with the range of values from `first` to `last`.
- `push_back(value)` – Appends `value` to the end of the vector.
- `pop_back()` – Removes the last element.

# The `vector` Container Class

## Mutator Functions

```
iterator insert(iterator it, const T& value);
void insert(iterator it, size_type n,
    const T& value);
void insert(iterator it, iterator first,
    iterator last);
```

- `insert(it, value)` – Inserts `value` in position given by the iterator `it`.
- `insert(it, n, value)` – Inserts `n` copies of `value` starting in position given by the iterator `it`.
- `insert(it, first, last)` – Starting in position given by iterator `it`, inserts values in range given by iterators `first` and `last`.

# The `vector` Container Class

## Mutator Functions

```
iterator erase(iterator it);
iterator erase(iterator first, iterator last);
void swap(vector& v);
void clear();
```

- `erase(it)` – Removes element in position given by the iterator `it`.
- `erase(first, last)` – Removes range of elements given by the iterators `first` and `last`.
- `swap()` – Swaps this vector with the given `vector`.
- `clear()` – Removes all elements.

# The `vector` Container Class

## Iterator Functions

```
iterator begin();
iterator end();
reverse_iterator rbegin();
reverse_iterator rend();
```

- `begin()` – Returns iterator set to beginning.
- `end()` – Returns iterator set to end.
- `rbegin()` – Returns reverse iterator set "reverse beginning."
- `rend()` – Returns reverse iterator set to "reverse end."

# The `vector` Container Class

## Programming with `vector`s

- Write a program that creates a vector of **`int`**s, adds some **`int`**s to it, and and then prints the list.

# The `vector` Container Class

## Programming with `vector`s

```
#include <vector>
int main()
{
    vector<int> vec;
    vec.push_back(10);
    vec.push_back(20);
    vec.push_back(30);
    vector<int>::iterator it;
    for (it = vec.begin(); it != vec.end(); it++)
        cout << *it << endl;
}
```

# Outline

# The stack Adaptor Class

- An adaptor class uses a container class.
- We may construct a stack in any of the following ways.

## Ways to Construct a Stack

```
#include <stack>
int main()
{
    stack<int> s1;
    stack<int, vector<int> > s2;
    stack<int, deque<int> > s3;
    stack<int, list<int> > s4;
}
```

# The stack Adaptor Class

- The `stack` class has the following member functions (besides the fundamental four).

## `stack` Member Functions

```
bool empty() const;
int size() const;
T& top();
void push(const T& value);
void pop();
```

# Outline

# The map Container Class

- A map is an *associative* list.
- Each member has
  - A key.
  - A value.
- The key must be unique for that member.
- The value is accessed through the key, by matching the key.
- This sounds like a hash table.

# The map Container Class

- Suppose we want to store a list of students and their declared majors.

| Name | Major |
|---|---|
| John | Mathematics |
| Tim | Computer Science |
| Betty | Chemistry |
| Ann | Mathematics |

# The map Container Class

- If we intend to locate members by name, then
  - The name is the key
  - The major is the value.
- We construct the (empty) map:

## Construct a `map`

```
#include <map>
map<string, string> major;
```

# The map Container Class

- To add the data, we may use the subscript operator:

## Initialize the `map`

```
major["John"] = "Mathematics";
major["Tim"] = "Computer Science";
major["Betty"] = "Chemistry";
major["Ann"] = "Mathematics";
```

# The map Container Class

- To find `"John"`, we use the `find()` function.
- It returns an iterator to John's location in the map.

## Search the `map`

```
map<string, string>::iterator it;
it = major.find("John");
```

# The map Container Class

- The data members first and second store the key and the value.

## Print the `map`

```
map<string, string>::iterator it;
for (it = major.begin(); it != major.end(); it++)
    cout << it.first << " is majoring in "
         << it.second << endl;
```

# Outline

# Assignment

## Assignment

- Read Sections 9.7 - 9.8.