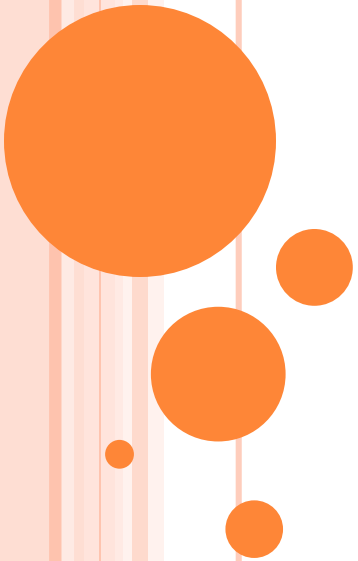

The Standard Template Library (STL – Part 2)

CS 250 – C++ Programming 2

Last Lecture

- A **sequence container** stores and manages objects in a ***sequential*** order
- STL **sequence containers**:
  - `vector`
    - Implemented as a **dynamic array**
  - `list`
    - Implemented as **doubly-linked list**
  - `deque`
    - Implemented as a **dynamic array**

Review

- Given the following code:

```
vector<int> v1 = { 10, 11, 12, 13, 14, 15 };
vector<int>::iterator iter = v1.begin() + 1;
vector<int> v2(iter, iter + 3);
```

- What are the elements in v2 after the code is executed?

Review

- Given the following code:

```
vector<int> v1 = { 10, 11, 12, 13, 14, 15 };
vector<int>::iterator iter = v1.begin() + 1;
vector<int> v2(iter, iter + 3);
```

- What are the elements in v2 after the code is executed?
  - 11 12 13

Review

- Given the following code:

```
vector<int> v1 = { 10, 11, 12, 13, 14, 15 };
vector<int>::iterator iter = v1.begin() + 1;
vector<int> v2(iter, iter + 3);
```

- What are the elements in v2 after the code is executed?
  - 11 12 13

- What was used to execute the third statement?

Review

- Given the following code:

```
vector<int> v1 = { 10, 11, 12, 13, 14, 15 };
vector<int>::iterator iter = v1.begin() + 1;
vector<int> v2(iter, iter + 3);
```

- What are the elements in v2 after the code is executed?
  - 11 12 13
- What was used to execute the third statement?
  - Overloaded constructor

Review

- Given the following code:

```
vector<int> v1 = { 10, 11, 12, 13, 14, 15 };
vector<int>::iterator iter = v1.begin() + 1;
vector<int> v2(iter, iter + 3);
```

- What are the elements in v2 after the code is executed?
  - 11 12 13

- After the code is executed, to which element is iter pointing to?

Review

- Given the following code:

```
vector<int> v1 = { 10, 11, 12, 13, 14, 15 };
vector<int>::iterator iter = v1.begin() + 1;
vector<int> v2(iter, iter + 3);
```

- What are the elements in v2 after the code is executed?
  - 11 12 13
- After the code is executed, to which element is iter pointing?
  - 11

Review (cont.)

- Given this other code segment:

```
int a[] = { 10, 11, 12, 13, 14, 15 };
vector<int> v (a, a + 3);
```

- What are the elements in v after the code is executed?

Review (cont.)

- Given this other code segment:

```
int a[] = { 10, 11, 12, 13, 14, 15 };
vector<int> v (a, a + 3);
```

- What are the elements in v after the code is executed?
  - 10 11 12

More Containers

- This time we will look at:
  - Class **pair**
    - Used by **map** and **multimap**
  - **Associative containers**
    - **set**, **multiset**, **map**, and **multimap**
  - **Container adaptors**
    - Layered on top of ***sequential containers***
    - **stack**, **queue**, and **priority_queue**

STL Class Pair

CLASS pair

- The class **pair** combines **two values** in a single unit
- Every object of type pair has **two data members**:
  - **first** and **second**
  - Can be **different types**
  - Both member variables are **public**

> - This means that **first** and **second** can be accessed **without** using an accessor function.

- Need to include `<utility>`

CLASS pair (cont.)

- The class pair has three constructors:
  - The **default constructor**:

```
pair<T1,T2> pairObj;
```

  - An **overloaded constructor** with two parameters:

```
pair<T1,T2> pairObj(T1,T2);
```

  - A **copy constructor**:

```
pair<T1,T2> pairObj(otherPairObj);
```

CLASS pair (CONT.)

```
#include <utility>
...
pair<int,double> x;
x.first = 3;
x.second = 4.0;

pair<int,double> y(13, 45.9);
pair<string,int> student("Bob", 1234);

cout << student.first;     //Output: Bob
cout << student.second; //Output: 1234
```

member variables are **public**

Associative Containers

Associative Containers

- **Associative containers** are ***automatically*** **sorted** according to some ordering criteria
  - Default **ordering criterion**
    - The relational operator **<** (less than)
    - **Ascending** order
    - Can be changed to other criteria

- STL **associative containers**:
  - **Sets** and **multisets**
  - **Maps** and **multimaps**

Sets and Multisets

- The STL **set** and **multiset** classes ***automatically*** **sort** their elements according to some criteria
  - By **default**, the sorting is done in **ascending order**
    - But it can also be specified according to a different sorting criterion

- The only difference between **sets** and **multisets** is a **multiset** allows **duplicates**

Maps and Multimaps

- The STL **map** and **multimap** classes manage their elements in the form **key/value** (a given ordered pair)
  - The elements are ***automatically*** **sorted** according to some sort criteria applied on the **key**
  - By **default**, the sorting is done in **ascending order**
    - But it can also be specified according to a different sorting criterion

- The only difference between **maps** and **multimaps** is that a **multimap** allows **duplicates**

FUNCTION make_pair

- The **header file `utility`** contains also the definition of the **function template `make_pair`**

- The **function** creates pairs **without** explicitly specifying the type **pair**
  - Useful when using **maps**

    (see example on next slide)

FUNCTION make_pair (CONT.)

```
map<int,int> intMap;
map<int,int>::iterator it;

for (int i = 1; i < 10; ++i)  // insert integers
      intMap.insert( make_pair(i, (i * 10)) );

for (it = intMap.begin(); it != intMap.end(); ++ it)
      cout << it->first << " " << it->second << endl;
```

The **map** will contain the following elements:

{ (1, 10), (2, 20), … (8, 80), (9, 90) }

Container Adaptors

Container Adaptors

- **Container adaptors** are **template classes** that are implemented on top of other classes
- STL **container adaptors**:
  - **stack**
    - Implemented on top of the ***deque*** template class
  - **queue**
    - Implemented on top of the ***deque*** template class
  - **priority_queue**
    - Implemented on top of the ***vector*** template class

Note: "adaptor" or "adapter" ➔ *correct spelling either way*

Container Adaptors (cont.)

- We have already looked at **stacks** and **queues**…
- In the **priority_queue** class, elements with **higher priority** are popped from queue
- **Default underlying** container:
  - The ***vector*** template class

THE priority_queue CLASS

- The **priority_queue** class is a **queue** with the additional property that each entry is given a priority when it is added to the queue
  - If all **entries** have the **same priority**, then entries are removed from a priority queue in the same manner as they are removed from a queue.
  - If **entries** have **different priorities**, by default, the **higher-priority items** are removed **before** lower-priority items.

THE priority_queue CLASS (CONT.)

- There are certain situations where the **first-in first-out** rule needs to be relaxed somewhat
  - Examples of a **priority queue**
    - In a hospital environment, patients are, usually, seen in the order they arrive. If a patient, however, arrives with severe or life-threatening symptoms, s/he is treated first. In other words, these patients take priority over the patients who can wait to be seen.
    - In a shared environment, when print requests are sent to the printer, interactive programs take over batch-processing programs.

Practice

- The best way for you to learn about the **STL** is to try and manipulate the functions listed on the tables.

- Make sure you also check **cplusplus.com** and read the information provided for the functions presented earlier.

STL 2 (END)