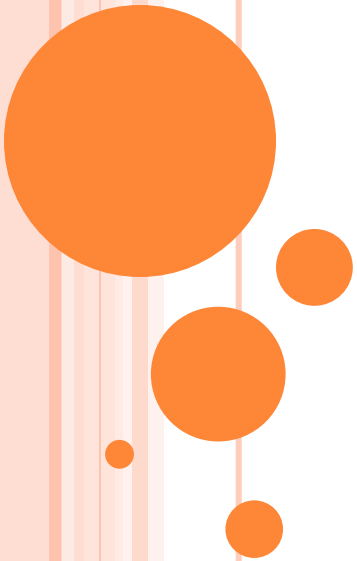

The Standard Template Library (STL – Part 3)

CS 250 – C++ Programming 2

Types of Iterators

- **NOTE:** The classification of iterators in this set of slides is **not** complete.
  - We will only discuss types of iterators we are interested in at this point.

Types of Iterators (cont.)

- **Forward iterators**
  - Can access the sequence of elements in a range in the direction that goes from its beginning towards its end.
- **Bidirectional iterators**
  - Can access the sequence of elements in a range in both directions (towards the end and towards the beginning).
- **Random access iterators**
  - Can access elements at an arbitrary offset position relative to element they point to, offering the same functionality as pointers.

Containers and Iterators

Container	Type of iterator supported
Sequence containers	
`vector`	random access
`deque`	random access
`list`	bidirectional
Associative containers	
`set`	bidirectional
`multiset`	bidirectional
`map`	bidirectional
`multimap`	bidirectional
Container adaptors	
`stack`	no iterators supported
`queue`	no iterators supported
`priority_queue`	no iterators supported

Efficiency

- The **STL** is designed to be **highly** **efficient**.
  - **Sets** and **maps** store elements in sorted order for fast searches.
  - **Stacks** and **queues** are designed to work in a very specific way.
  - And so on…

Efficiency (cont.)

- When choosing **containers**, you need to consider **how often** and in **which manner** the container needs to be accessed, and what is the **general purpose** of the container.

  - For example, there is **no** need to create a **set** if it is not necessary to have elements in order.
    - **Why?**
      - Every time you insert a new element, the algorithm will re-structure the set to place the new element in its proper set

        → Unnecessary step if elements need not to be in order.

Container Functions

- You have seen **functions** that belong to specific **containers**
  - **clear**
  - **sort**
  - **merge**
  - These are **member functions**
  - They **cannot** be called without creating an **object** of the **container** class to which they belong.

STL Algorithms

- The **STL** has functions that are available in a more general form, called **generic algorithms**
  - You have already seen a **generic algorithm**: **copy**

- We will look at a few **algorithms**
  - **What to keep in mind:**
    - Not all algorithms will/should work with every container
    - The type of iterators used can determine which containers will work with that specific algorithm
    - Think logically…

Important Details

- A function **sort** can be found in both the STL list and the STL algorithm.

- Member function of STL list:

```
void sort( );
```

- Non-member function of STL algorithm:

```
void sort (RandomAccessIterator first,
           RandomAccessIterator last);
```

Important Details (cont.)

- Member function of STL list:

```
void sort( );
```

Will this work?

```
list<int> myList;
// insert elements

myList.sort( );      // member function
                     // requires object: myList
```

Important Details (cont.)

- Member function of STL list:

```
void sort( );
```

Yes. Function sort is designed to work with the STL list.

```
list<int> myList;
// insert elements

myList.sort( );      // member function
                     // requires object: myList
```

Important Details (cont.)

- Non-member function of STL algorithm:

```
void sort (RandomAccessIterator first,
           RandomAccessIterator last);
```

Will this work?

```
vector<int> yourVector;
// insert elements

sort(yourVector.begin(), yourVector.end());
```

Important Details (cont.)

- Non-member function of STL algorithm:

```
void sort (RandomAccessIterator first,
           RandomAccessIterator last);
```

Yes. The non-member function sort will work with an STL vector.

```
vector<int> yourVector;
// insert elements

sort(yourVector.begin(), yourVector.end());
```

Important Details (cont.)

- Non-member function of STL algorithm:

```
void sort (RandomAccessIterator first,
           RandomAccessIterator last);
```

Will this work?

```
list<int> yourList;
// insert elements

sort(yourList.begin(), yourList.end());
```

Important Details (cont.)

- Non-member function of STL algorithm:

```
void sort (RandomAccessIterator first,
           RandomAccessIterator last);
```

No. Why not?

```
list<int> yourList;
// insert elements

sort(yourList.begin(), yourList.end());
```

Important Details (cont.)

- Non-member function of STL algorithm:

```
void sort (RandomAccessIterator first,
           RandomAccessIterator last);
```

It does NOT work, because an STL list does NOT have random-access iterators.

```
list<int> yourList;
// insert elements

sort(yourList.begin(), yourList.end());
```

Algorithms

- **Project algorithms** shows how some of the most common algorithms work.

- You can certainly (and you *should*) try other **algorithms**, but you will be tested only for those **algorithm functions** used on the files that accompany this set of slides.

STL 3 (END)