C++

STL: Introduction to algorithms

Introduction

- STL provides containers (collection of objects) and iterators (to iterate over range of objects in containers)
- Iterators are key to generic programming because they allow us to write generic algorithms operating on many kinds of containers
- STL provide several algorithms
- To use them we need to include the header <algorithm>

Introduction - 2

- Algorithms in STL are organized in different categories:
  - Non-mutating: find, for_each, count_if, etc
  - Mutating: copy, transform, replace, etc
  - Sorting: sort, nth_element, set operations, etc
  - Numeric algorithms: inner_product, partial_sum, etc

Example of STL algorithm: sort()

- In its simplest form **sort()** takes two iterators and sort the container between these two iterators
- Example: sort a vector

```
vector<T> A;
sort(A.begin(), A.end());
```

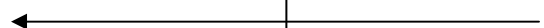
- Note: it implies that there exists an order on T and that **operator<** is defined for element of type T

Example of STL algorithm: sort()

```
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  vector<string> v;
  v.push_back("a"); v.push_back("z");
  v.push_back("e"); v.push_back("w");
  sort(v.begin(), v.end());
  vector<string>::iterator it;
  for (it = v.begin(); it != v.end(); it++)
     cout << *it << " ";
}
```



It works fine because the following function is defined:

```
bool
operator<(const string& s1,
          const string& s2);
```

Example of STL algorithm: sort()

- sort() works with random access container such as vector or deque
- List is not a random access container so calling sort() with list iterators will fail
- sort() has a guaranteed worst case running time complexity of O(n log n)
- Note: list has a member method sort()

find()

- find(first, last, value) returns an iterator in [first, last) if it finds an element equal to value, otherwise it returns last

```
int main ()
{
 vector<int> v;
 v.push_back(1); v.push_back(2);
 v.push_back(5); v.push_back(9);

 vector<int>::iterator it = find(v.begin(), v.end(), 2);
 if (it != v.end()) cout << “Found” << endl;
 else cout << “Not found” << endl;
}
```

copy()

- copy(first, last, out) copies elements in range [first, last) to the range [out, out + (last – first)). It returns result + (last – first).
- Example:

```
vector<int> v;
v.push_back(1); v.push_back(5);

vector<int> v2(v.size());
copy(v.begin(), v.end(), v2.begin());
```

Note: copy is not inserting elements in an empty container but it is overwriting elements of an existing container.
v2 has been initialized with the size of v.