CS32 - Week 5

Doga Kisa

July 25, 2014

Outline

- STL Containers
  - vector
  - list
  - set
  - map
- Iterator
- Big-O Notation

STL

Standard **T**emplate **L**ibrary

- Library of commonly used data structures.
  - stack
  - queue
  - vector (resizable array)
  - list (linked list)
  - set (collection of unique values)
  - map (collection with one-to-one correspondence)

STL

Common functions for all

- `.empty()` whether a container is empty
- `.size()` number of elements currently hold

All are using templates

- `stack<int> a;`
- `vector<string> b;`
- `list<Coor> c;`

STL

`stack<T>` (LIFO)

- `#include <stack>`
- `.push(T), .pop(), .top()`

`queue<T>` (FIFO)

- `#include <queue>`
- `.push(T), .pop(), .front(), .back()`

STL

`vector<T>` : resizable array

- Constructor
  - `vector<int> a;` // size 0
  - `vector<int> b(100)` // size 100, initialized to 0
  - `vector<int> c(100,999)` // size 100, initialized to 999
- `.push_back(T) // insert element at the end`
- Random access by [ ] // like in arrays
- `.front(), .back() // accessor`
- `.pop_back() // remove last element`
- `.insert(position, val)`

STL

list<T> : linked list

- .push_back(T), .push_front(T)
- .pop_back(), .pop_front()
- .front(), .back()

No random access memory!

STL

set<T> : collection of unique elements

- .insert() // no effect of inserting an element twice
- If you are using your customized class, you need to define **operator<** function.

```
      set<Coor> coor;
```

- Elements are stored in ascending order.

STL

map<K,V> : one-to-one correspondence, unique key

- Constructor

  ```
  map<string,double> gpaMap;
  ```

- Access using [ ]

  ```
  cout « gpaMap["Alice"];
  gpaMap["Bob"] = 2.5;
  ```

STL

What's missing?

How to traverse the container?

vectors?

- Access elements as in arrays.

others?

- Use **Iterator**!

STL

Iterator

- Iterate through all elements in a STL container.
- Works like a pointer.
- Type
  - container_type::iterator it;
  - list<int>::iterator it1;
  - set<string>::iterator it2;

STL

Iterator

- Each STL container has
  - .begin() return an iterator pointing to the first element
  - .end() return an iterator pointing to the position just past the last element
- list<int>::iterator it = lst.begin();
- Access the element pointed by the iterator: *it
- You can use -> for functions/public variables: `it->func_name()`
- You can move your iterator forward by ++ and backward by `--`

STL

Traverse a vector

```
vector<string> vec(10, "abc");
vector<string>::iterator it = vec.begin();
while(it != vec.end()) {
  cout << (*it);
  it++;
}
```

STL

Traverse a list

```
list<string> myList;
myList.push_back("a");
myList.push_back("b");
list<string>::iterator it = myList.begin();
while(it != myList.end()) {
  cout << (*it);
  it++;
}
```

STL

Traverse a map

```
map<string, int> myMap;
myMap["cat"] = 5;
myMap["dog"] = 10;
map<string, int>::iterator it = myMap.begin();
while(it != myMap.end()) {
  cout << it->first << " " << it->second << endl;
  it++;
}
```

STL

Use **const_iterator** if the container is constant!

```
void func(const list<string> & myList){
  list<string>::const_iterator it = myList.begin();
  while(it != myList.end()) {
    cout << (*it);
    it++;
  }
}
```

STL

Iterators can be used with functions like insert() and erase():

```
list<int> myList;
myList.push_back(0);  // 0
myList.push_back(1);  // 0 1

list<int>::iterator it = myList.begin();
it++;
myList.insert(it,30);
// 0 30 1, it points to 1
myList.erase(it);
// 0 30
```

STL

.erase(iterator) function for **list** actually returns an iterator (pointing to the next element).

```
void eraseAll(list<int> & myList){
  list<int>::iterator it = myList.begin();
  while (it != myList.end()) {
      it = myList.erase(it);
  }
}
```

STL

For set `.find(x)` function returns an iterator pointing to the element with value x.

For map `.find(x)` function returns an iterator pointing to the key x.

If not exists, it returns `.end()` iterator.

```
set<int> mySet;
mySet.insert(5);
mySet.insert(10);
mySet.insert(2);
set<int>::iterator it = mySet.find(5);
if (it != mySet.end())
  cout << *it << endl;
else
  cout << "sorry" << endl;
```

STL

You don’t have to memorize name of member functions for each, just look things up when you need to.
e.g. http://www.cplusplus.com/reference/stl/

STL Example 1

Given a vector of strings, print how many times each string appears.

Input: "x", "y", "z", "x", "y", "x"

Output:

- x:3
- y:2
- z:1

```
void printCount(const vector<string> vec);
```

STL Example 1

```
void printCount(const vector<string> vec){
  map<string,int> m;
  vector<string>::const_iterator it;
  map<string,int>::iterator itMap;
  for(it = vec.begin(); it != vec.end(); it++){
    itMap = m.find(*it);
    if(itMap == m.end())
      m[*it] = 1;
    else
      m[*it]++;
  }
  itMap = m.begin();
  while(itMap != m.end()){
    cout<< itMap->first <<":"<< itMap->second <<endl;
    itMap++;
  }
}
```

STL Example 2

Given a vector of strings, print out strings which appear odd number of times.

Constraint: You can not use map stl!

Input: "x", "y", "z", "x", "y", "x"

Output:

- x
- z

void printOdds(const vector<string> vec);

STL Example 2

```
void printOdds(const vector<string> vec){
  set<string> s;
  vector<string>::const_iterator it;
  set<string>::iterator itSet;
  for(it = vec.begin(); it != vec.end(); it++){
    itSet = s.find(*it);
    if(itSet == m.end())
      s.insert(*it);
    else
      s.erase(itSet);
  }
  itSet = s.begin();
  while(itSet != s.end()){
    cout << *itSet << endl;
    itSet++;
  }
}
```

Big-O Analysis

- We should implement **efficient** programs!
    - Economic use of time and memory space.
- How to measure efficiency?
    - Big-O analysis / asymptotic analysis
    - Given an input of size **n**, approximately how long does it take the algorithm to finish the task?
        - Linear, polynomial, exponential in **n**

Big-O Analysis

Let f(x) and g(x) be two functions of real numbers. Then:

$$\boxed{\begin{array}{l} f(x) \text{ is } O(g(x)) \text{ as } x \to \infty \text{ if and only if} \\ \exists x_0, c > 0 \text{ such that } f(x) \leq c \cdot g(x) \text{ for } x > x_0. \end{array}}$$

- We read O(g(x)) "Big-O of g(x)".
- Big-O is supposed to give you an upper bound on (time or size) of the function.

Examples

Examples:

- If $f(x) = 100x + 1000$, then $f(x) = O(x)$.
- If $f(x) = x^2 + 5x$, then $f(x) = O(x^2)$.
- If $f(x) = x^2 + 5x$, then $f(x) = O(x^3)$.
- If $f(x) = 5x^{100} - 5x^{99}$, then $f(x) = O(x^{100})$.
- If $f(x) = 2^x$, then $f(x) = 2^x$.
- If $f(x) = 5$, then $f(x) = O(1)$.

We are mostly interested in the tightest bound we can find.

Big-O Analysis

Big O	Name	$n = 128$
$O(1)$	constant	1
$O(\log n)$	logarithmic	7
$O(n)$	linear	128
$O(n \log n)$	"n log n"	896
$O(n^2)$	quadratic	16192
$O(n^k), k \geq 1$	polynomial	
$O(2^n)$	exponential	10^{40}
$O(n!)$	factorial	10^{214}

Big-O Analysis

Unit operations take O(1) time

- Addition, subtraction, multiplication, division
- Comparison, assignment

Repetition upgrades complexity.

- One loop -> $O(n)$
- Nested loop (double) -> $O(n^2)$

Count the most complicated loop!

Big-O Analysis

If things happen sequentially, we add Big-Os

- int x = 1;
  x ++;
  O(1) + O(1) = O(1)
- foo1(); // $O(n)$
  foo2(); // $O(n^2)$
  $O(n) + O(n^2) = O(n^2)$

If one thing happens within another (like loops), we multiply Big-Os

- for(int i =0; i<n: i++)
  { foo(i); }
  Assume foo(i) takes O(log n) time.
  Total: O(n log n)

Big-O Analysis

```
search(int arr[], int size, int val) {
  for (int i = 0; i<size; i++){
    if (arr[i] == val)
      return i;
  }
  return -1;
}
```

What is the running time of the algorithm?

Best case:	O(1)
Worst case:	O(n)
Average case:	O(n)

Big-O Analysis

Problem: Given an array of n unique elements, print out all pairs in the set?
For example, if the array is $\{123\}$, you should print $\{12\}\{13\}\{21\}\{23\}\{31\}\{32\}$.

```
  all_pairs(int arr[], int size) {
    for(int i = 0; i < size; i++){
      for(int j = 0; j < size; j++){
        if (i != j)
          cout << "{arr[i] arr[j]}" ;
      }}}
```

What is the running time of the algorithm?

- Best case:
- Worst case: All $O(n^2)$
- Average case:

Big-O for STLs

Vector	
size	O(1)
push_back	O(1)
pop_back	O(1)
erase	O(n)
insert	O(n)
clear	O(n)
find	O(n)
random access	O(1)

List	
size	O(1)
push_front	O(1)
push_back	O(1)
pop_front	O(1)
pop_back	O(1)
erase	O(1)
insert	O(1)
clear	O(n)
find	O(n)

Big-O for STLs

Set	
`size`	O(1)
`insert`	O(log n)
`erase`	O(log n)
`clear`	O(n)
`find`	O(log n)

Map	
`size`	O(1)
`insert`	O(log n)
`erase`	O(log n)
`clear`	O(n)
`find`	O(log n)

STL Example 1

Given a vector of strings, print how many times each string appears.

Input: "x", "y", "z", "x", "y", "x"

Output:

- x:3
- y:2
- z:1

```
void printCount(const vector<string> vec);
```

STL Example 1

```
void printCount(const vector<string> vec){
  map<string,int> m;
  vector<string>::const_iterator it;
  map<string,int>::iterator itMap;
  for(it = vec.begin(); it != vec.end(); it++){
    itMap = m.find(*it);
    if(itMap == m.end())
      m[*it] = 1;
    else
      m[*it]++;
  }
  itMap = m.begin();
  while(itMap != m.end()){
    cout<< itMap->first <<":"<< itMap->second <<endl;
    itMap++;
  }
}
```

STL Example 2

Given a vector of strings, print out strings which appear odd number of times.

Constraint: You can not use map stl!

Input: "x", "y", "z", "x", "y", "x"

Output:

- x
- z

void printOdds(const vector<string> vec);

STL Example 2

```
void printOdds(const vector<string> vec){
  set<string> s;
  vector<string>::const_iterator it;
  set<string>::iterator itSet;
  for(it = vec.begin(); it != vec.end(); it++){
    itSet = s.find(*it);
    if(itSet == m.end())
      s.insert(*it);
    else
      s.erase(itSet);
  }
  itSet = s.begin();
  while(itSet != s.end()){
    cout << *itSet << endl;
    itSet++;
  }
}
```

Slides

Slides will be available at

`http://www.cs.ucla.edu/~doga/cs32`