CS 162, Lecture 23: Standard Template Library

23 May 2018
Standard Template Library

• Collection of common abstract data types
• When found in the STL, abstract data type == container classes
• Data Structure: a particular way of storing and organizing data
  • Array, Linked List, Trees
• Abstract Data Type: the rules that govern how data is accessed
  • Stack – Last in, First Out
  • Queue – First in, First Out
  • Priority Queue – Highest priority out first
Types of Containers

• Sequential: arrange data items into a list such that there is a first, next, and so on up to last
• Adapters: implemented on top of other classes (abstract data types)
• Associative: very simple databases (set, map)
Iterators

• A construct that allow you to cycle through the data items stored in a data structure

• Generalization of a pointer: typically implemented as a pointer but is abstracted so we don’t have to deal with it

• Every container class in STL has its own iterator
```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {

    vector<int> v;

    for(int i=0; i<5; i++) {
        v.push_back(i);
    }
    cout << "Print v" << endl;

    vector<int>::iterator itr;
    for(itr = v.begin(); itr != v.end(); itr++) {
        cout << *itr << " ";
    }
    cout << endl;
    cout << "sentinel";

    return 0;
}
```
Overload Operators

• ++ move to next item
• -- move to previous item
• == equality test
• != not equal test

* dereference operator
Types of Iterators

• Forward: ++ works on the iterator
• Bidirectional: both ++ and -- work on the iterator
• Random Access: ++, --, and random access all work
• Constant: doesn’t allow changes to be made to element at its location
• Mutable: can change the element at its location
• Reverse: can be used to cycle through all elements of a container with bidirectional iterators
• Input: forward iterator that can be used with input stream
• Output: forward iterator to be used with output stream
Linked Lists

- A list constructed using pointers
- Can grow and shrink while the programming is running
- Not stored contiguously in memory
- Use structs to create

```c
struct Node {
    int val;
    node* next;
};
```
Singly Linked List

1. node * head = new node;
2. head -> next = new node;
3. head = head -> next -> next = NULL;
4. head = head -> next -> next;
5. head = head -> next;
6. head = head -> next;
7. head = head -> next -> next;

For (int i = 0; i < 3; i++)
{
    struct node * current;
    current = head;
    current -> val = i + 1;
    current -> next = new node;
    current = current -> next;
}