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 -- LIFO
  - Queue -- FIFO
  - Priority Queue -- Highest Priority out first
Types of Containers

- Sequential: arrange data items into a list [first, next, ..., last]
- Adapters: implemented on top of other classes (abstract data types)
- Associative: very simple databases (set, map)
Iterators

- A construct that allows 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
Thinking with Data Structures

- Ignore the implementation details of Containers and Iterators
- Focus on how Containers and Iterators interact
- What is the ‘shape’ of your container?
- How do you want to **Iterate** through the items it holds?
- What do you want to **access** from the item?
  - Now we care about a member variable!
Practice

1] Build a vector of five items, holding the number item they are

2] Output “ 1 2 3 4 5” by iterating over the vector
#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;

    return 0;
}
```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;

    return 0;
}
```
Commonly Overloaded Operators for Iterators

++ move to next item
-- move to previous item
== equality test
!= not equal test
* dereference operator
Types of Iterators

- Forward: ++ works on 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 program is running
- Not stored contiguously in memory
- Use nodes (struct) to create

```c
struct Node {
    int val;
    node* next;
}
```
Build A Singly Linked List with Commands

1. node* head = new node;
2. head -> next = new node;
3. head -> next -> next = new node
4. head -> next -> next -> next = NULL;
5. head -> val = 1;
6. head -> next -> val = 2;
7. head -> next -> next -> val = 3;
Build a Singly Linked List with a loop

current = new node;

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