**Stacks and Queues**

Stacks: Last in First Out

#include <stack>

using namespace std;

stack<int> myStack;

myStack.push(1);

Functions:

empty() – is empty?

size() – size of stack

top() – get the top element

push(element) – put element on top

pop() – take off top element

Use top() then pop() otherwise you lose the thing on top

Queues: First in First out

instead of top(), it has:

front()

back()

Stacks: depth first search

Queues: breadth first search

**Inheritance and polymorphism**

class A {

};

class B : public A

{

};

protected: methods only subclass can see, no public calling

-don’t use protected vars

virtual: goes down the chain

Construction:

1.call base class’s constructor

2.initialize base class’s member variables

3.run base class’s constructor

4.initialize derived class’s member variables

5.run derived class’s constructor

Destruction:

1.destructor derived runs

2.derived member vars destructed

3.destructor base runs

4.base member vars destructed

abstract base classes:

have at least one pure virtual function, can’t be constructed. Must construct derived class

virtual void myFunc() = 0;

When to use virtual?

-redefinition of functions in derived classes

-any time you redefine a function

-virtual for destructor

-no virtual constructors

STL Classes

Vector:

#include <vector>

using namespace std;

vector<int> myVec;

push\_back(int)

access existing items with myVec[i]

pop\_back()

size()

empty()

don’t use an iterator, use [] to access

List

#include <list>

using namespace std;

push\_back

pop\_back

front

back

insert

erase

push\_front, pop\_front (vector doesn’t have)

iterating:

list<int> l1(5, MAGIC);

for (list<int>::iterator it = l1.begin(); it != l1.end(); it++)

{

cout << (\*it) << endl;

}

Map

#include <map>

#include <string>

using namespace std;

map<string, int> peeps;

peeps [“Joe”] = 22;

peeps[“Chris”] = 19;

to go through, use map<string, int>::iterator it and access with (\*it).first, (\*it).second

can also do cout << peeps[“Joe”] << endl;

maintains items alphabetically

Set

#include <set>

using namespace std;

set<int> a;

a.insert(1);

a.insert(3);

a.insert(2);

a.insert(1); //duplicate

a.erase(1);

sets are in alphabetical order