**STANDARD TEMPLATE LIBRARY**

A container is a holder object that stores a collection of other objects (its elements). They are implemented as class templates, which allows a great flexibility in the types supported as elements.

**Sequence Containers**

Sequence containerss are container classes that maintain the ordering of elements in the container. A defining characteristic of sequence containers is that you can choose where to insert your element by position. The most common example of a sequence container is the array: if you insert four elements into an array, the elements will be in the exact order you inserted them.

The STL contains 3 sequence containers: vector, deque, and list.

The **deque** class (pronounced “deck”) is a double-ended queue class, implemented as a dynamic array that can grow from both ends.

**list** is a special type of sequence container called a doubly linked list where each element in the container contains pointers that point at the next and previous elements in the list. Lists only provide access to the start and end of the list -- there is no random access provided. If you want to find a value in the middle, you have to start at one end and “walk the list” until you reach the element you want to find. The advantage of lists is that inserting elements into a list is very fast if you already know where you want to insert them. Generally iterators are used to walk through the list.

**Associative Containers**

Associative contains are containers that automatically sort their inputs when those inputs are inserted into the container. By default, associative containers compare elements using operator<.

* A **set** is a container that stores unique elements, with duplicate elements disallowed. The elements are sorted according to their values.
* A **multiset** is a set where duplicate elements are allowed.
* A **map** (also called an associative array) is a set where each element is a pair, called a key/value pair. The key is used for sorting and indexing the data, and must be unique. The value is the actual data.
* A **multimap** (also called a dictionary) is a map that allows duplicate keys. Real-life dictionaries are multimaps: the key is the word, and the value is the meaning of the word. All the keys are sorted in ascending order, and you can look up the value by key. Some words can have multiple meanings, which is why the dictionary is a multimap rather than a map.

**Container Adapters**

Container adapters are special predefined containers that are adapted to specific uses. The interesting part about container adapters is that you can choose which sequence container you want them to use.

* A **stack** is a container where elements operate in a LIFO (Last In, First Out) context, where elements are inserted (pushed) and removed (popped) from the end of the container. Stacks default to using deque as their default sequence container (which seems odd, since vector seems like a more natural fit), but can use vector or list as well.
* A **queue** is a container where elements operate in a FIFO (First In, First Out) context, where elements are inserted (pushed) to the back of the container and removed (popped) from the front. Queues default to using deque, but can also use list.
* A **priority queue** is a type of queue where the elements are kept sorted (via operator<). When elements are pushed, the element is sorted in the queue. Removing an element from the front returns the highest priority item in the priority queue.

**The find() algorithm**

// find.cpp

// finds the first object with a specified value

#include <iostream>

#include <algorithm> //for find()

using namespace std;

int arr[] = { 11, 22, 33, 44, 55, 66, 77, 88 };

int main()

{

int\* ptr;

ptr = find(arr, arr+8, 33); //find first 33

cout << "First object with value 33 found at offset "

<< (ptr-arr) << endl;

return 0;

}

First object with value 33 found at offset 2

**The count() algorithm**

// count.cpp

// counts the number of objects with a specified value

#include <iostream>

#include <algorithm> //for count()

using namespace std;

int arr[] = { 33, 22, 33, 44, 33, 55, 66, 77 };

int main()

{

int n = count(arr, arr+8, 33); //count number of 33's

cout << "There are " << n << " 33's in arr." << endl;

return 0;

}

There are 3 33's in arr.

The search () algorithm

// search.cpp

// searches one container for a sequence in another container

#include <iostream>

#include <algorithm>

using namespace std;

int source[] = { 11, 44, 33, 11, 22, 33, 11, 22, 44 };

int pattern[] = { 11, 22, 33 };

int main()

{

int\* ptr;

ptr = search(source, source+9, pattern, pattern+3);

if(ptr == source+9) // if past-the-end

cout << "No match found\n";

else

cout << "Match at " << (ptr - source) << endl;

return 0;

}

Match at 3

**The sort() algorithm**  
// sort.cpp

// sorts an array of integers

#include <iostream>

#include <algorithm>

using namespace std;

// array of numbers

int arr[] = {45, 2, 22, -17, 0, -30, 25, 55};

int main()

{

sort(arr, arr+8); // sort the numbers

for(int j=0; j<8; j++) // display sorted array

cout << arr[j] << ' ';

cout << endl;

return 0;

}

**Sequence containers**

**Vectors**

// vector.cpp

// demonstrates push\_back(), operator[], size()

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<int> v; // create a vector of ints

v.push\_back(10); // put values at end of array

v.push\_back(11);

v.push\_back(12);

v.push\_back(13);

v[0] = 20; // replace with new values

v[3] = 23;

for(int j=0; j<v.size(); j++) // display vector contents

cout << v[j] << ' '; // 20 11 12 23

cout << endl;

return 0;

}

20 11 12 23

// vectcon.cpp

// demonstrates constructors, swap(), empty(), back(), pop\_back()

#include <iostream>

#include <vector>

using namespace std;

int main()

{ // an array of doubles

double arr[] = { 1.1, 2.2, 3.3, 4.4 };

vector<double> v1(arr, arr+4); // initialize vector to array

vector<double> v2(4); // empty vector of size 4

v1.swap(v2); // swap contents of v1 and v2

while( !v2.empty() ) // until vector is empty,

{

cout << v2.back() << ' '; // display the last element

v2.pop\_back(); // remove the last element

} // output: 4.4 3.3 2.2 1.1

cout << endl;

return 0;

}

4.4 3.3 2.2 1.1

// vectins.cpp

// demonstrates insert(), erase()

#include <iostream>

#include <vector>

using namespace std;

int main()

{

int arr[] = { 100, 110, 120, 130 }; //an array of ints

vector<int> v(arr, arr+4); // initialize vector to array

cout << "\nBefore insertion: ";

for(int j=0; j<v.size(); j++) // display all elements

cout << v[j] << ' ';

v.insert( v.begin()+2, 115); // insert 115 at element 2

cout << "\nAfter insertion: ";

for(int j=0; j<v.size(); j++) // display all elements

cout << v[j] << ' ';

v.erase( v.begin()+2 ); // erase element 2

cout << "\nAfter erasure: ";

for(int j=0; j<v.size(); j++) // display all elements

cout << v[j] << ' ';

cout << endl;

return 0;

}

Before insertion: 100 110 120 130

After insertion: 100 110 115 120 130

After erasure: 100 110 120 130

**Lists**

// list.cpp

// demonstrates push\_front(), front(), pop\_front()

#include <iostream>

#include <list>

using namespace std;

int main()

{

list<int> ilist;

ilist.push\_back(30); // push items on back

ilist.push\_back(40);

ilist.push\_front(20); // push items on front

ilist.push\_front(10);

int size = ilist.size(); // number of items

for(int j=0; j<size; j++)

{

cout << ilist.front() << ' '; // read item from front

ilist.pop\_front(); // pop item off front

}

cout << endl;

return 0;

}

10 20 30 40

// listplus.cpp

// demonstrates reverse(), merge(), and unique()

#include <iostream>

#include <list>

using namespace std;

int main()

{

int j;

list<int> list1, list2;

int arr1[] = { 40, 30, 20, 10 };

int arr2[] = { 15, 20, 25, 30, 35 };

for(j=0; j<4; j++)

list1.push\_back( arr1[j] ); // list1: 40, 30, 20, 10

for(j=0; j<5; j++)

list2.push\_back( arr2[j] ); // list2: 15, 20, 25, 30, 35

list1.reverse(); // reverse list1: 10 20 30 40

list1.merge(list2); // merge list2 into list1

list1.unique(); // remove duplicate 20 and 30

int size = list1.size();

while( !list1.empty() )

{

cout << list1.front() << ' '; // read item from front

list1.pop\_front(); // pop item off front

}

cout << endl;

return 0;

}

10 15 20 25 30 35 40

**Iterators**

// listout.cpp

// iterator and for loop for output

#include <iostream>

#include <list>

#include <algorithm>

using namespace std;

int main()

{

int arr[] = { 2, 4, 6, 8 };

list<int> theList;

for(int k=0; k<4; k++) //fill list with array elements

theList.push\_back( arr[k] );

list<int>::iterator iter; //iterator to list-of-ints

for(iter = theList.begin(); iter != theList.end(); iter++)

cout << \*iter << ' '; //display the list

cout << endl;

return 0;

}

2 4 6 8

// listfill.cpp

// uses iterator to fill list with data

#include <iostream>

#include <list>

using namespace std;

int main()

{

list<int> iList(5); // empty list holds 5 ints

list<int>::iterator it; // iterator

int data = 0;

// fill list with data

for(it = iList.begin(); it != iList.end(); it++)

\*it = data += 2;

// display list

for(it = iList.begin(); it != iList.end(); it++)

cout << \*it << ' ';

cout << endl;

return 0;

}

2 4 6 8 10

// iterfind.cpp

// find() returns a list iterator

#include <iostream>

#include <algorithm>

#include <list>

using namespace std;

int main()

{

list<int> theList(5); // empty list holds 5 ints

list<int>::iterator iter; // iterator

int data = 0;

// fill list with data

for(iter = theList.begin(); iter != theList.end(); iter++)

\*iter = data += 2; //2, 4, 6, 8, 10

// look for number 8

iter = find(theList.begin(), theList.end(), 8);

if( iter != theList.end() )

cout << "\nFound 8.\n";

else

cout << "\nDid not find 8.\n";

return 0;

}

Found 8.

**Associative Containers**

**Set**

// set.cpp

// set stores string objects

#pragma warning (disable:4786) //for set (Microsoft compilers only)

#include <iostream>

#include <set>

#include <string>

using namespace std;

int main()

{ // array of string objects

string names[] = {"Juanita", "Robert",

"Mary", "Amanda", "Marie"};

// initialize set to array

set<string, less<string> > nameSet(names, names+5);

// iterator to set

set<string, less<string> >::iterator iter;

nameSet.insert("Yvette"); // insert more names

nameSet.insert("Larry");

nameSet.insert("Robert"); // no effect; already in set

nameSet.insert("Barry");

nameSet.erase("Mary"); // erase a name

// display size of set

cout << "\nSize=" << nameSet.size() << endl;

iter = nameSet.begin(); // display members of set

while( iter != nameSet.end() )

cout << \*iter++ << '\n';

string searchName; // get name from user

cout << "\nEnter name to search for: ";

cin >> searchName;

// find matching name in set

iter = nameSet.find(searchName);

if( iter == nameSet.end() )

cout << "The name " << searchName << " is NOT in the set.";

else

cout << "The name " << \*iter << " IS in the set.";

cout << endl;

return 0;

}

Size=7

Amanda

Barry

Juanita

Larry

Marie

Robert

Yvette

**Maps and Multimaps**

// asso\_arr.cpp

// demonstrates map used as associative array

#pragma warning (disable:4786) //for map (Microsoft only)

#include <iostream>

#include <string>

#include <map>

using namespace std;

int main()

{

string name;

int pop;

string states[] = { "Wyoming", "Colorado", "Nevada",

"Montana", "Arizona", "Idaho"};

int pops[] = { 470, 2890, 800, 787, 2718, 944 };

map<string, int, less<string> > mapStates; //map

map<string, int, less<string> >::iterator iter; //iterator

for(int j=0; j<6; j++)

{

name = states[j]; //get data from arrays

pop = pops[j];

mapStates[name] = pop; //put it in map

}

cout << "Enter state: "; //get state from user

cin >> name;

pop = mapStates[name]; //find population

cout << "Population: " << pop << ",000\n";

cout << endl; //display entire map

for(iter = mapStates.begin(); iter != mapStates.end(); iter++)

cout << (\*iter).first << ' ' << (\*iter).second << ",000\n";

return 0;

}

Enter state: Wyoming

Population: 470,000

Arizona 2718,000

Colorado 2890,000

Idaho 944,000

Montana 787,000

Nevada 800,000

Wyoming 470,000

**Storing user-defined objects**

// setpers.cpp

// uses a multiset to hold person objects

#pragma warning (disable:4786) // for set (Microsoft only)

#include <iostream>

#include <set>

#include <string>

using namespace std;

class person

{

private:

string lastName;

string firstName;

long phoneNumber;

public: // default constructor

person() : lastName("blank"),

firstName("blank"), phoneNumber(0)

{ }

// 3-arg constructor

person(string lana, string fina, long pho) :

lastName(lana), firstName(fina), phoneNumber(pho)

{ }

friend bool operator<(const person&, const person&);

friend bool operator==(const person&, const person&);

void display() const // display person's data

{

cout << endl << lastName << ",\t" << firstName

<< "\t\tPhone: " << phoneNumber;

}

};

// operator < for person class

bool operator<(const person& p1, const person& p2)

{

if(p1.lastName == p2.lastName)

return (p1.firstName < p2.firstName) ? true : false;

return (p1.lastName < p2.lastName) ? true : false;

}

// operator == for person class

bool operator==(const person& p1, const person& p2)

{

return (p1.lastName == p2.lastName &&

p1.firstName == p2.firstName ) ? true : false;

}

////////////////////////////////////////////////////////////////

int main()

{ // create person objects

person pers1("Deauville", "William", 8435150);

person pers2("McDonald", "Stacey", 3327563);

person pers3("Bartoski", "Peter", 6946473);

person pers4("KuangThu", "Bruce", 4157300);

person pers5("Wellington", "John", 9207404);

person pers6("McDonald", "Amanda", 8435150);

person pers7("Fredericks", "Roger", 7049982);

person pers8("McDonald", "Stacey", 7764987);

// multiset of persons

multiset< person, less<person> > persSet;

// iterator to a multiset of persons

multiset<person, less<person> >::iterator iter;

persSet.insert(pers1); // put persons in multiset

persSet.insert(pers2);

persSet.insert(pers3);

persSet.insert(pers4);

persSet.insert(pers5);

persSet.insert(pers6);

persSet.insert(pers7);

persSet.insert(pers8);

cout << "\nNumber of entries = " << persSet.size();

iter = persSet.begin(); // display contents of multiset

while( iter != persSet.end() )

(\*iter++).display();

// get last and first name

string searchLastName, searchFirstName;

cout << "\n\nEnter last name of person to search for: ";

cin >> searchLastName;

cout << "Enter first name: ";

cin >> searchFirstName;

// create person with this name

person searchPerson(searchLastName, searchFirstName, 0);

// get count of such persons

int cntPersons = persSet.count(searchPerson);

cout << "Number of persons with this name = " << cntPersons;

// display all matches

iter = persSet.lower\_bound(searchPerson);

while( iter != persSet.upper\_bound(searchPerson) )

(\*iter++).display();

cout << endl;

return 0;

} // end main()

Number of entries = 8

Bartoski, Peter Phone: 6946473

Deauville, William Phone: 8435150

Fredericks, Roger Phone: 7049982

KuangThu, Bruce Phone: 4157300

McDonald, Amanda Phone: 8435150

McDonald, Stacey Phone: 3327563

McDonald, Stacey Phone: 7764987

Wellington, John Phone: 9207404

Enter last name of person to search for: KuangThu

Enter first name: Bruce

Number of persons with this name = 1

KuangThu, Bruce Phone: 4157300

// listpers.cpp

// uses a list to hold person objects

#include <iostream>

#include <list>

#include <algorithm>

#include <string>

using namespace std;

class person

{

private:

string lastName;

string firstName;

long phoneNumber;

public:

person() : // no-arg constructor

lastName("blank"), firstName("blank"), phoneNumber(0L)

{ }

// 3-arg constructor

person(string lana, string fina, long pho) :

lastName(lana), firstName(fina), phoneNumber(pho)

{ }

friend bool operator<(const person&, const person&);

friend bool operator==(const person&, const person&);

friend bool operator!=(const person&, const person&);

friend bool operator>(const person&, const person&);

void display() const // display all data

{

cout << endl << lastName << ",\t" << firstName

<< "\t\tPhone: " << phoneNumber;

}

long get\_phone() const // return phone number

{ return phoneNumber; }

};

// overloaded == for person class

bool operator==(const person& p1, const person& p2)

{

return (p1.lastName == p2.lastName &&

p1.firstName == p2.firstName ) ? true : false;

}

// overloaded < for person class

bool operator<(const person& p1, const person& p2)

{

if(p1.lastName == p2.lastName)

return (p1.firstName < p2.firstName) ? true : false;

return (p1.lastName < p2.lastName) ? true : false;

}

// overloaded != for person class

bool operator!=(const person& p1, const person& p2)

{ return !(p1==p2); }

// overloaded > for person class

bool operator>(const person& p1, const person& p2)

{ return !(p1<p2) && !(p1==p2); }

////////////////////////////////////////////////////////////////

int main()

{

list<person> persList; // list of persons

// iterator to a list of persons

list<person>::iterator iter1;

// put persons in list

persList.push\_back( person("Deauville", "William", 8435150) );

persList.push\_back( person("McDonald", "Stacey", 3327563) );

persList.push\_back( person("Bartoski", "Peter", 6946473) );

persList.push\_back( person("KuangThu", "Bruce", 4157300) );

persList.push\_back( person("Wellington", "John", 9207404) );

persList.push\_back( person("McDonald", "Amanda", 8435150) );

persList.push\_back( person("Fredericks", "Roger", 7049982) );

persList.push\_back( person("McDonald", "Stacey", 7764987) );

cout << "\nNumber of entries = " << persList.size();

iter1 = persList.begin(); // display contents of list

while( iter1 != persList.end() )

(\*iter1++).display();

// find person or persons with specified name (last and first)

string searchLastName, searchFirstName;

cout << "\n\nEnter last name of person to search for: ";

cin >> searchLastName;

cout << "Enter first name: ";

cin >> searchFirstName;

//make a person with that name

person searchPerson(searchLastName, searchFirstName, 0L);

//search for first match of names

iter1 = find(persList.begin(), persList.end(), searchPerson);

if( iter1 != persList.end() ) //find additional matches

{

cout << "Person(s) with that name is(are)";

do

{

(\*iter1).display(); //display match

++iter1; //search again, one past match

iter1 = find(iter1, persList.end(), searchPerson);

} while( iter1 != persList.end() );

}

else

cout << "There is no person with that name.";

// find person or persons with specified phone number

cout << "\n\nEnter phone number (format 1234567): ";

long sNumber; //get search number

cin >> sNumber;

//iterate through list

bool found\_one = false;

for(iter1=persList.begin(); iter1 != persList.end(); ++iter1)

{

if( sNumber == (\*iter1).get\_phone() ) //compare numbers

{

if( !found\_one )

{

cout << "Person(s) with that phone number is(are)";

found\_one = true;

}

(\*iter1).display(); //display the match

}

} //end for

if( !found\_one )

cout << "There is no person with that phone number";

cout << endl;

return 0;

} //end main()

Number of entries = 8

Deauville, William Phone: 8435150

McDonald, Stacey Phone: 3327563

Bartoski, Peter Phone: 6946473

KuangThu, Bruce Phone: 4157300

Wellington, John Phone: 9207404

McDonald, Amanda Phone: 8435150

Fredericks, Roger Phone: 7049982

McDonald, Stacey Phone: 7764987

Enter last name of person to search for: Bartoski

Enter first name: Peter

Person(s) with that name is(are)

Bartoski, Peter Phone: 6946473

Enter phone number (format 1234567): 8435150

Person(s) with that phone number is(are)

Deauville, William Phone: 8435150

McDonald, Amanda Phone: 8435150