159.234 Lecture 23

Today

- Standard Template Library

Programs in: programs/p19

Bibliography:

- Textbook p.252,
- http://www.research.att.com/~bs/3rd_tour2.pdf

Bjarne Stroustrup, *The C++ Programming Language* (3rd Ed.),

A Tour of the Standard Library

Standard Template Library

What is it?

STL is a set of general-purpose template classes, built using the template mechanism.

What are the main parts?

- Containers,
- Iterators,
- Algorithms.

Containers

What are they?

Containers are objects that hold other objects.

There are different types of containers:

•**sequence** containers: `vector`, `deque`, `list`

•**associative** containers for allowing efficient retrieval of values based on keys: `map`, `sets`.

```
#include <iostream>
using namespace std;

template< class T>  // a generic func to print an array
void printArray( T * a, int len ){
  for(int j=0;j<len;j++) cout << a[j] << " ";
  cout << endl;
}

int main(){

  //typedef  long int MyType;
  typedef  long long int MyType;

  const int Length = 10;
  MyType  array[Length];

  cout << "Size of  MyType is: " << sizeof(MyType ) << endl;

  for( int i=0;i<Length;i++){
    array[i] = i + 1;
  }

  printArray( array, Length );

  MyType *ptr;
  MyType *start = array;
  MyType *end = array + Length;

  for( ptr=start;ptr!=end;ptr++){
    *ptr *= -1;

    cout << ptr << endl;
  }

  printArray( array, Length );

  return 0;
}
```

- Preamble to iterators:
- Pointer arithmetic
- Example Output:

```
Size of  MyType is: 8
1 2 3 4 5 6 7 8 9 10
0xbffffc10
0xbffffc18
0xbffffc20
0xbffffc28
0xbffffc30
0xbffffc38
0xbffffc40
0xbffffc48
0xbffffc50
0xbffffc58
-1 -2 -3 -4 -5 -6 -7 -8 -9 -10
```

- This code will work regardless of what MyType actually is.

- The operators + etc are all overloaded for pointers so that pointer arithmetic works as it should.

Vector class

Vectors are dynamic arrays: arrays that can grow as needed.

The template specification for the vector class is:

```
template <class T>
class vector{
  //...
};
```

When using the vector type the required header to be included is `<vector>`

Vector class

Declaring/ defining a vector:

```
vector<int> iv;

vector<char> cv(5);

vector<char> cv(5,  'x');

vector<int> iv2(iv);
```

Vector class

Using a vector:

// display original size of v
```
 cout<<"Size = "<<v.size()<<endl;
```

 /* put values onto end of a vector; the vector will grow as needed */
```
   for(i=0; i<10; ++i)
       v.push_back(i);
```

// change contents of a vector
```
 for(i=0; i<v.size(); ++i)
      v[i] = v[i] + v[i];
```

`Vector` class

// access using subscripting

```
for(i=0; i<10; ++i) cout <<v[i]<<" ";
cout << endl;
```

// access via iterator

```
vector<char>::iterator p = v.begin();
while(p != v.end()) {
  cout << *p << " ";
  ++p;
}
```

Declaring an iterator:

container_name :: iterator *iterator_name*

Iterator

- An Iterator is just a convenient pseudo-pointer that is set up as a type to use associated with each container class.

- Various operators will have been set up to use with them eg =, ==, != and +=

- Standard idiomatic form:

```
int array[10];
for(inti=0;I<10;I++){
}
```

- Becomes:

```
vector<int> v(10);
vector<int>::iterator it;
for(it=v.begin();it!=v.end();it++){
}
```

Iterators

Iterators are **objects** similar to **pointers**.

They are design to give us the ability to cycle through the content of a container.

There are five iterator types:

- input
- output
- forward
- bidirectional
- random access

Iterators

Container classes and algorithms dictate the category of iterator available or needed.

- `vector` containers allow random-access iterators
- `lists` do not;

- sorting requires a random-access iterators
- find requires an input iterator.

Iterators

Iterators can be adapted to provide backward traversal.

Example that uses a reverse iterator to traverse a sequence.

```
template <class ForwIter>
void print(ForwIter first, ForwIter last,
           const char* title){
   cout << title << endl;
   while ( first != last)
      cout << *first++ << '\t';
   cout << endl;
}
```

Iterators

```
int main(){

   int data[3] = { 9, 10, 11};

   vector<int> d(data, data + 3);

   vector<int>::reverse_iterator
            p = d.rbegin();

   print(p, d.rend(), "Reverse");
   //...
}
```

Other standard components

STL relies upon several other standard components for support:

allocators: to manage memory allocation for a container

predicates: special functions returning `true/false` results

comparison functions, etc.

Summary

The standard template library (STL) is the C++ library that provides generic programming for many standard data structures and algorithms.

Containers come in two major families: sequence (are ordered) and associative (have keys for looking up elements).

Iterators can be thought of as an enhanced pointer type.

The algorithms use iterators to access containers.

C++ Standard

C++Standard: http://www.cygnus.com/misc/wp/

International standard for the C++ programming language approved!

Morristown, New Jersey, USA, Friday, November 14, 1997

FOR IMMEDIATE RELEASE

This week, technical experts representing eight countries and about 40 companies involved with software technologies met in Morristown, New Jersey and completed the content of an international standard for the C++ programming language.

Scope of the Standard:

The C++ standard covers both the C++ language itself and its standard library.

The standard library provides

- standard input/output,
- strings,
- containers (such as vectors, lists, and strings),
- non-numerical algorithms (such as sort, search, and merge), and support for numeric computation.

The ISO/ANSI Standard for C++ was unanimously approved by the C++ Standardization Committee on June 23, 1998.

From:
http://www.awl.com/cseng/meyerscd/cstandard.htm

The current status (June'99) is that C++ has an International Standard and the committee are in the process of handling Defect Reports. We shall continue to do this until 2003 when ISO rules require us to revise the Standard.

From: http://www.inquiry.com/techtips/cpp_pro/