Introduction to C, C++, and Unix/Linux

CS 60

Lecture 20: Templates & STL

Today

→ Templates, STL

→ Reading [KR] Chapters 1-7

→ Read [So] chapters 1, 3, 4 (Boolean), 9, 13, 14 & 18.

Notes

- Questions?

C++ Templates

- Templates allow you to efficiently write generic functions and classes that work for several different data types
  - Makes overloaded functions
  - Better than writing a function/class for each type
  - Better than writing macros
  - Can be tricky, though
- The Standard Template Library (STL) provides C++ with a number of useful templates and efficient ways to access their data

Templetes for Class stack

We looked at several examples through the progs web page.

The Standard Template Library (STL)

- The STL is a general purpose library of data structures and algorithms, using the C++ template mechanism
- Its components are heavily parameterized – almost every component in the STL is a template

Main STL components

- At its core are *container classes* – classes whose purpose is to contain other objects
  - These are templates that can be instantiated to contain any type of object
- The STL also includes a large collection of *algorithms* that manipulate the data stored in containers
  - These are global functions, decoupled from the classes
- Function Objects: Objects that act like functions.
- The STL defines *iterators* – a generalization of pointers – that allow access to the data inside containers
  - Forward, reverse, random

STL containers

- The basic STL containers:
  - vector : like an array and dynamic memory
  - deque : double ended queue
  - priority-queue
  - stack, queue, bit set
  - list : doubly-linked list
  - set : an unordered set of unique items
  - multiset : nonunique set
  - map : an associative array: key/value lookup
  - multimap : a map that allows multiple values per key

```
vector<int> r(5);  // 5 ints
vector<double> scores(n); //n doubles
//You may use scores[i], [] is overloaded
size():  # of elements in container
swap(): Exchanges contents
begin(): Iterator (ref to 1st element)
end(): Iterator (ref to one after last)
vector<double>:: iterator pd;
vector<double scores(n);
pd=scores.begin();
*pd=22.3; ++pd;
for(pd=scores.begin();pd!=scores.end();pd++)
```

Other Functions

erase();
insert();
find();
for_each()
random_shuffle()
sort();

Examples

```
vector<int> V;
V.insert(V.begin(), 3);

list<int> L;
L.push_back(0);
L.push_front(1);
L.insert(++L.begin(), 2);
copy(L.begin(), L.end(), ostream_iterator<int>(cout,
" "));

template <class InputIterator, class T>
InputIterator find(InputIterator first,
                   InputIterator last, const T&
value) {
    while (first != last && *first != value) ++first;
    return first;
}
```

In conclusion...

What we've learned

- Basics of Unix/Linux
  - commands, editing, compiling, debugging, shell scripting, ...
- The C language
  - Types, operators, memory and pointers...
- The C++ language
  - Classes, function overloading, exceptions, templates

And there is all there is and there isn’t any more…