6.S096: Introduction to C/C++

Frank Li, Tom Lieber, Kyle Murray

Lecture 8:

Last Lecture

Helter Skelter Fun!

January 31, 2012

Today…

- Standard Template Library (STL)
- Crazay Const
- Exceptions
- Function pointers
- Brief intro to C++11

Today…

- **Standard Template Library (STL)**
- Crazay Const
- Exceptions
- Function pointers
- Brief intro to C++11

Standard Template Library (STL)

- “Included” w/ compiler
- Contains containers/data structures, iterators, and algorithms
- http://www.cplusplus.com/reference

Vectors

- Equivalent to array lists in other languages (dynamic size)

```
#include <vector>

...

std::vector<int> int_list;

int_list.push_back(1);

int tmp = int_list[0]; // tmp = 1

int_list.pop_back(); // int_list now empty
```

Map

- Like a dictionary in Python (tree implementation).

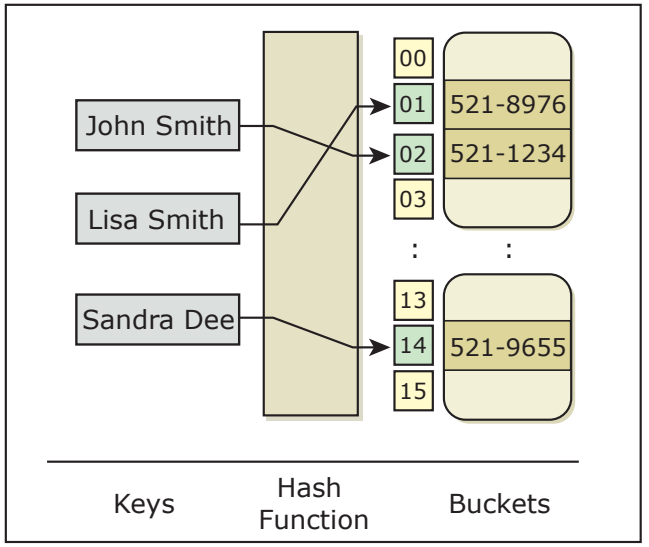

Image by MIT OpenCourseWare.

```
#include <map>

...

std::map<char, int> letter_to_int;

letter_to_int['a'] = 1;

letter_to_int['b'] = 2;

int pos = letter_to_int['a'] // pos = 1;
```

Others Containers

- Other useful containers:
  - **<array>:** array w/ functions
  - **<list>:** doubly linked list
  - **<set>:** stores only unique elements, tree implementation
  - **<unordered_map>:** actual hash table
  - **<unordered_set>:** stores only unique elements, hash table implementation
  - *Look online for more!*

Iterators

- Object that points to elements in a data structure, and can *iterate* through. Like a pointer.

- Vector iterator: std::vector<int>::iterator it;

- Access element at iterator: *it;

- Can do add/subtract to iterators: it++, it--

Vector Iterators

```
#include <vector>
#include <iostream>
...
std::vector<int> vec;
for (int i=1; i<=5; i++)
    vec.push_back(i);
for (std::vector<int>::iterator it = vec.begin();
    it != vec.end(); ++it){
    std::cout << '  ' << *it;
}
//Will print: 1  2  3  4  5
```

Cool/Useful Algorithm

- #include <algorithm>
- Sort a vector:
  - std::sort(vec.begin(), vec.end());
- Reverse a vector:
  - std::reverse(vec.begin(), vec.end());
- Min/Max:
  - std::min(3,1) == 1 ; std::max(3,1) == 3
- So many more online!

Today…

- Standard Template Library (STL)
- **Crazay Const**
- Exceptions
- Function pointers
- Brief intro to C++11

Const Madness!

- const int x == int const x;

  // Const int

- const int * c1;

  // Pointer to const int

- int const * c2; // Same as c1

- int * const c3; // Const pointer to variable int

More Const Madness!

- Const in function parameter

```
bool cant_touch_this(const myobj & obj);
//cant_touch_this can't modify obj
```

- Const functions are safe for const objects, but can be called by all objects. Non-const functions can only be called by non-const objects.

```
bool catch_change_this() const {
    ...
}
```

When will the madness end!?!?

const int* const

crazay(const int* const & madness) const;

Today…

- Standard Template Library (STL)
- Crazay Const
- **Exceptions**
- Function pointers
- Brief intro to C++11

Exceptions

- Exceptions are “error” objects that are “thrown” when things go bad.

Exceptions

- Exception parent object in std::exception
  - std::runtime_error
  - std::bad_alloc
  - std::bad_cast
- Can create your own custom exceptions

```
class MyException : public exception{
  const char * what() const {
      return "MyException"; // human-readable
  }
};
```

Try Catch

```
try {
    ...  // Protected code
    throw MyError();
}catch( YourError e1 ){
    cout << e1.what()  << endl;
}catch( MyError e2 ){
    cout << e2.what() << endl;
}catch(...) {
    ... // handle all other exceptions
}
```

Throwing that Exceptions

- Can throw a primitive as an exception

```
try{
    …
    throw 20;
}catch(int e) { cout << “Error: ” << e << endl; }
```

- Best way to catch your own exception:

```
try{
    throw MyException();
}catch(MyException & e) {
    cout << e.what() << endl;
}
```

Functions Throwing

- Functions add throw(*exceptions-thrown*) at end of declaration.

```
void ahh(int i) throw() {
    //This function assumed to not throw anything
}

void blahh(int i) throw(int) {
    //This function may throw int if there's an error
}
```

Today…

- Standard Template Library (STL)
- Crazay Const
- Exceptions
- **Function pointers**
- Brief intro to C++11

Functions Pointers

- void (*foo) (int); // Foo is pointer to void function that takes 1 int argument.

- void *(*foo)(int *, char); //Any guesses?

Functions Pointers

- void (*foo) (int); // Foo is pointer to void function that takes 1 int argument.

- void *(*foo)(int *, char); //Foo is a pointer to a function that takes 1 int* argument and 1 char argument, that returns void*

Using Functions Pointers

```
#include <iostream>
void my_int_func(int x){
    std::cout << x << std::endl;
}
int main(){
    void (*foo)(int);
    foo = &my_int_func;  // The ampersand is actually optional
    (*foo)(2); // Calls my_int_func
    foo(2); // Same as above line
    return 0;
}
```

Functions Pointers Arguments

```
#include <iostream>
void call(int x){
   std::cout << x << std::endl;
}
void call_me_maybe(int number, void (*call_func)(int)) {
    call_func(number);
}
int main(){
   void (*foo)(int);
   foo = call;
   call_me_maybe(911, foo);
   return 0;
}
```

Functions Pointers Arguments

```
#include <iostream>
void call(int x){
    std::cout << x << std::endl;
}
void call_me_maybe(int number, void (*call_func)(int)) {
     call_func(number);
}
int main(){
    call_me_maybe(911, call);
    return 0;
}
```

For_each

```
#include <algorithm> // Header file need for for_each()
#include <vector>
#include <iostream>
using namespace std;
void print (int i) {  cout << ' ' << i;  }
int main(){
    vector<int> myvector;
    myvector.push_back(10);
    myvector.push_back(20);
    for_each (myvector.begin(), myvector.end(), print);
    ...
} // Prints out 10 20
```

Today…

- Standard Template Library (STL)
- Crazay Const
- Exceptions
- Function pointers
- **Brief intro to C++11**

C++11

- Latest and “greatest” standard of C++ w/ fancy features

- Fancy feature 1: Auto typing

```
int x = 1;
auto y = x;
vector<int> vec;
auto itr = vec.begin(); //versus vector<int>::iterator
```

C++11

- Latest and “greatest” standard of C++ w/ fancy features

- Fancy feature 2: Declare typing

```
int x = 1;
decltype(x) y = x; //Makes y the same type as x
```

C++11

- Latest and “greatest” standard of C++ w/ fancy features

- Fancy feature 3: Right angle brackets
  - Before C++11:

    vector<vector<int> > vector_of_int_vectors;
  - C++11:

    vector<vector<int>> vector_of_int_vectors;

C++11

- Latest and “greatest” standard of C++ w/ fancy features

- Fancy feature 4: Range for loops

```
vector<int> vec;
vec.push_back( 1 );
vec.push_back( 2 );
for (int& i : vec ) {
    i++; // increments the value in the vector
}
```

C++11

- Latest and “greatest” standard of C++ w/ fancy features

- Fancy feature 5: Lambda functions!!!
  - Syntax: [] (arg_list) { func_definition }
  - Ex:

```
#include <iostream>
using namespace std;
int main(){
   auto func = [] (int i) { cout << "Hi " << i << endl; };
   func(1); // now call the function
}
```

C++11

- Many more fancy features (strongly typed enums, null pointer constant, etc)!

\(^_^) /

Yay!!!

- Reference:

http://www.cprogramming.com/tutorial.html#c++11

http://en.wikipedia.org/wiki/C%2B%2B11

It's about that time...

- Introduced you to C and C++ syntax, features, and idoms.
- Gave you some experience w/ writing C/C++ code
- Much more to learn, best way is to simply code more!

6.S096 Introduction to C and C++

IAP 2013