

# Introduction to the STL

Eric Tune

# What is STL

- STL = Standard Template Library
- Part of the ISO Standard C++ Library
- Data Structures and algorithms for C++.

# Why should I use STL?

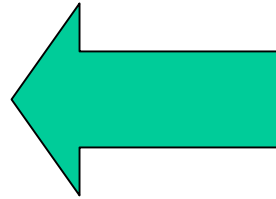
- Reduce development time.
  - Data-structures already written and debugged.
- Code readability
  - Fit more meaningful stuff on one page.
- Robustness
  - STL data structures grow automatically.
- Portable code.
- Maintainable code
- Easy

# Keywords you DONT need

- `class`
- `virtual`
- `public`
- `private`
- `template`
- `mutable`

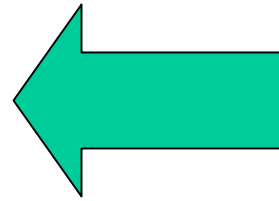
# Example Program

```
#include <map>
#include <string>
map<string, float> price;
price["snapple"] = 0.75;
price["coke"] = 0.50;
string item;
double total=0;
while ( cin >> item )
    total += price[item];
```



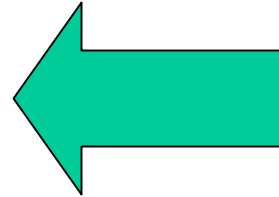
# Example Program

```
#include <map>
#include <string>
map<string, float> price;
price["snapple"] = 0.75;
price["coke"] = 0.50;
string item;
double total=0;
while ( cin >> item )
    total += price[item];
```



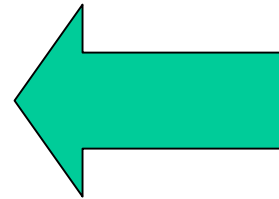
# Example Program

```
#include <map>
#include <string>
map<string, float> price;
price["snapple"] = 0.75;
price["coke"] = 0.50;
string item;
double total=0;
while ( cin >> item )
    total += price[item];
```



# Example Program

```
#include <map>
#include <string>
map<string, float> price;
price["snapple"] = 0.75;
price["coke"] = 0.50;
string item;
double total=0;
while ( cin >> item )
    total += price[item];
```



# The 'Top 3' data structures

- **map**
  - Any key type, any value type.
  - Sorted.
- **vector**
  - Like c array, but auto-extending.
- **list**
  - doubly-linked list

# Simple Example of Map

```
map<long, int> root;  
root[4] = 2;  
root[1000000] = 1000;  
long l;  
cin >> l;  
if (root.count(l)) cout<<root[l]  
else cout<<"Not perfect square";
```

# Two ways to use Vector

- Preallocate

```
vector<int> v(100);  
v[80]=1; // okay  
v[200]=1; // bad
```

- Grow tail

```
vector<int> v2;  
int i;  
while (cin >> i)  
    v.push_back(i);
```

# Example of List

```
list<int> L;  
for(int i=1; i<=5; ++i)  
    L.push_back(i);  
//delete second item.  
L.erase( ++L.begin() );  
copy( L.begin(), L.end(),  
    ostream_iterator<int>(cout, ", "));  
// Prints: 1,2,3,5
```

# The three parts of STL

- Containers
- Algorithms
- Iterators

# Iterators

- Declaring

```
list<int>::iterator li;
```

- Front of container

```
list<int> L;
```

```
li = L.begin();
```

- Past the end

```
li = L.end();
```

# Iterators

- Can increment

```
list<int>::iterator li;
```

```
list<int> L;
```

```
li=L.begin();
```

```
++li; // Second thing;
```

- Can be dereferenced

```
*li = 10;
```

# Algorithms

- Take iterators as arguments

```
list<int> L;
```

```
vector<int> v;
```

```
// put list in vector
```

```
copy(      L.begin(),  
      L.end(),  
      v.begin() );
```

# List Example Again

```
list<int> L;  
for(int i=1; i<=5; ++i)  
    L.push_back(i);  
//delete second item.  
L.erase( ++L.begin() );  
copy( L.begin(), L.end(),  
    ostream_iterator<int>(cout, ", "));  
// Prints: 1,2,3,5
```

# Typdefs

- Annoying to type long names
  - `map<Name, list<PhoneNum> > phonebook;`
  - `map<Name, list<PhoneNum> >::iterator  
finger;`
- Simplify with typedef
  - `typedef PB map<Name, list<PhoneNum> >;`
  - `PB phonebook;`
  - `PB::iterator finger;`
- Easy to change implementation.

# Using your own classes in STL Containers

- Might need:
  - Assignment Operator, `operator=()`
  - Default Constructor
- For sorted types, like `map<>`
  - Need less-than operator: `operator<()`
    - Some types have this by default:
      - `int`, `char`, `string`
    - Some do not:
      - `char *`

# Example of User-Defined Type

```
struct point
```

```
{
```

```
float x;
```

```
float y;
```

```
}
```

```
vector<point> points;
```

```
point p; p.x=1; p.y=1;
```

```
points.push_back(1);
```

# Example of User-Defined Type

- Sorted container needs sort function.

```
struct full_name {  
    char * first;  
    char * last;  
    bool operator<(full_name & a)  
        {return strcmp(first, a.first) < 0;}  
}  
map<full_name, int> phonebook;
```

# What do I need?

- g++ 2.96
  - Fine for all examples in this talk
  - 3.0.x is even better
    - using namespace std;
- Mostly works with MSVC++
  - So i am told.

# Performance

- Personal experience 1:
  - STL implementation was 40% slower than hand-optimized version.
    - STL: used deque
    - Hand Coded: Used “circular buffer” array;
  - Spent several days debugging the hand-coded version.
  - In my case, not worth it.
  - Still have prototype: way to debug fast version.

# Performance

- Personal experience 2
- Application with STL list ~5% slower than custom list.
- Custom list “intrusive”
  - `struct foo {`
  - `int a;`
  - `foo * next;`
  - `};`
- Can only put `foo` in one list at a time ☹️

# Pitfalls

- Accessing an invalid `vector<>` element.

```
vector<int> v;  
v[100]=1; // whoops!
```

## Solutions:

- use `push_back()`
- Preallocate with constructor.
- Reallocate with `reserve()`
- Check `capacity()`

# Pitfalls

- Inadvertently inserting into `map<>` .  
    `if (foo["bob"]==1)`  
    `//silently created entry "bob"`

Use `count()` to check for a key without creating a new entry.

```
if ( foo.count("bob") )
```

# Pitfalls

- Not using `empty()` on `list<>` .
  - Slow
    - `if ( my_list.count() == 0 ) { ... }`
  - Fast
    - `if ( my_list.empty() ) { ... }`

# Pitfalls

- Using invalid iterator

```
list<int> L;  
list<int>::iterator li;  
li = L.begin();  
L.erase(li);  
++li;           // WRONG
```

- Use return value of erase to advance

```
li = L.erase(li); // RIGHT
```

# Common Compiler Errors

- `vector<vector<int>> vv;`

missing space 

lexer thinks it is a right-shift.

- any error message with `pair<...>`  
`map<a, b>` implemented with `pair<a, b>`

# STL versus Java Containers

## STL

- Holds any type
- No virtual function calls
- Static type-checking

## Java Containers

- Holds things derived from Object
- Virtual Function Call overhead
- No Static type-checking

# Other data structures

- `set`, `multiset`, `multimap`
- `queue`, `priority_queue`
- `stack`, `deque`
- `slist`, `bitset`, `valarray`

# Generic Programming Resources

- STL Reference Pages  
[\*\*www.sgi.com/tech/stl/\*\*](http://www.sgi.com/tech/stl/)

# More Generic Programming

- GTL : Graph Template Library
- BGL : Boost Graph Library
- MTL : Matrix Template Library
- ITL : Iterative Template Library