Exceptions

```cpp
int main() {
    int a = 5;
    try {
        int b = 10;
        try {
            if (a > 0)
                throw 1;
            if (b < 0)
                throw new exception();
        }
        catch (int me) {
            cout << "I got " << me;
        }
        cout << "Continue Here";
    }
    catch (const char *WhereAmI) {
        cout << "Do something\n";
        throw;
    }
    catch (...) {
        cout << "I catch anything";
    }
}
```
Exceptions and Initialization

class Foo {
public:
    int* k;
    Foo(int);
};

Foo::Foo(int start): k(new int(start)) {
}

class Foo {
public:
    int* k;
    Foo(int);
};

Foo::Foo(int start)
try : k(new int(start))
{
}
catch(...) {}
Exception Specification

```c
void bar(int x) {  
    bar code here  
}
```

```c
void foo(int x) throw(runtime_error) {  
    foo code here  
}
```

any exception can be raised in bar

States only runtime_error will be raised in foo

The compiler can not check to insure that only runtime_error will be raised in foo, so if another exception is raised ...
void foo(int x) throw() {
    foo code here
}

States no exceptions will be raised in foo
What Happens Here?

```cpp
void foo(int x) throw() {
    throw exception();
}

void foo(int x) throw(logic_error) {
    throw runtime_error();
}
```
void foo() {
    throw new exception();
}

void bar() {
    int* x = new int(3);
    foo();
    cout << "ready to delete\n";
    delete x;
}

int main() {
    try {
        bar();
    }
    catch (...) {
        cout << "What happened to delete?";
    }
}
Standard Template Library
Standard Library
# Standard (Template) Library

<table>
<thead>
<tr>
<th>Containers</th>
<th>Associative containers</th>
<th>Container Adapters</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector</td>
<td>set</td>
<td>queue</td>
</tr>
<tr>
<td>deque</td>
<td>multiset</td>
<td>priority_queue</td>
</tr>
<tr>
<td>list</td>
<td>map</td>
<td>stack</td>
</tr>
</tbody>
</table>

**Iterators**
- input
- forward
- bidirectional
- random access

**Algorithms**
- for_each
- find
- find_if
- find_first_of
- count
- copy
- replace
- sort
- etc.

**Functors (Function Objects)**
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> sample(2);
    sample[0] = 1;
    sample[1] = 3;
    sample.push_back(5);

    for (int k = 0; k < sample.size(); k++)
        cout << sample[k] << endl;
}
#include <iostream>
#include <vector>
#include <iterator>
using namespace std;

int main() {
    vector<int> sample(2);
    sample[0] = 1;
    sample[1] = 3;
    sample.push_back(5);

    reverse(sample.begin(), sample.end());

    vector<int>::const_iterator elements;
    for (elements = sample.begin(); elements != sample.end(); elements++)
        cout << *elements << endl;
}
```cpp
#include <iostream>
#include <vector>
#include <iterator>
using namespace std;

int main() {
    vector<int> sample(2);
    sample[0] = 1;
    sample[1] = 3;
    sample.push_back(5);

    vector<int>::iterator position = sample.begin();
    position++;
    sample.insert(position,10);
    sort(sample.begin(), sample.end());
    copy (sample.begin(), sample.end(), ostream_iterator<int> (cout, "\n"));
}
```
```cpp
#include <iostream>
#include <vector>
#include <iterator>
using namespace std;

template<class T> struct print : public unary_function<T, void> {
    print(ostream& out) : os(out), count(0) {}
    void operator() (T x) { os << x << ' '; ++count; }
    ostream& os;
    int count;
};

int main() {
    vector<int> sample(3);
    sample[0] = 1;
    sample[1] = 3;
    sample[2] = 5;

    print<int> printer = for_each(sample.begin(), sample.end(), print<int>(cout));
    cout << endl << printer.count << " objects printed." << endl;
}
```

Example Modified from http://www.sgi.com/tech/stl/for_each.html
#include <iostream>
using namespace std;

struct foo : public binary_function<int, int, int> {
    int operator() (int a, int b) const {
        return a + b;
    }
};

int main() {
    std::binder1st<foo> adder = bind1st(foo(), 10);
    cout << adder(5) << endl;
}
STLFilt
An STL Error Message Decryptor for C++

http://www.bdsoft.com/tools/stlfilt.html

"Renders many of even the most cryptic diagnostics comprehensible"
Templates & Code size

Templates increase
  Code size
  Memory requirements
  Run time
Misc
```c
#include <cstdio>

int main() {
    printf("Like C %s", "cat");
}
```
#import <iostream>

int main() {
    std::cout << "hi\n";
}

#include <iostream>

int main() {
    std::cout << "hi\n";
}
C++ Whining
Compiles without Errors or Warnings

```cpp
void foo(const std::string &) {}

int main() {
    foo(false);
}
```
What is $x \cdot y(z)$

```cpp
int y(int w) { return w; }

int main() {
    int x, z;
    x * y(z);
}
```

```cpp
int main() {
    struct x { x(int) {} } *z;
    x * y(z);
}
```
No compile time encapsulation

Change in a private member of a class

Requires recompiling all code that uses the class
Outstandingly complicated grammar

So

C++ compiles slowly
Error messages can be obscure
Parsing C++ is hard so compilers differ on interpretation
No way to locate definitions

AA BB(CC);

In Java AA is defined in
  The current package
  An imported package

In C++ AA is defined in
  The current file
  In included files
  In files included by the included files
  In files included by included files included by the included files
  In files included by any file included anywhere in the include chain

http://www.yosefk.com/c++fqa/defective.html
No run time encapsulation

pointer arithmetic
array bounds not checked
Very complicated type system

built-in types,
structures,
enumerations,
unions,
classes with single, multiple, virtual and non-virtual inheritance,
const and volatile qualifiers,
pointers, references and arrays,
typedefs,
global and member functions and function pointers, and
templates, which can have specializations on types (or integral constants),

http://www.yosefk.com/c++fqa/defective.html
What is this?

std::map<std::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::less<std::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::allocator<std::pair<std::basic_string<char, std::char_traits<char>, std::allocator<char> > const, std::basic_string<char, std::char_traits<char>, std::allocator<char> > > > >
Manual memory management

No garbage collection
| const Fred* p | Fred* const p | const Fred* const p |

http://www.yosefk.com/c++fqa/const.html
Does "Fred& const x" make any sense?