The C++ Standard Library

Ray Toal
Loyola Marymount University
and CitySearch, Inc.
May 13, 1998
Outline

• Background
• What is in the Standard Library
• Organization of the Library
• Tour of the Library
  – Overview of the Modules
  – Code Examples
• Concluding Remarks
Goals and Objectives

- To present the overall organization and examples of the use of the C++ Standard Library so that
  - Programmers will be able to start using the library right away
  - Programmers will be able to get rid of tons of poorly commented, under-tested, *non-standard*, container libraries that defy (large-scale) reuse
What This Talk is About

- What is in the Standard Library and how the library is organized
- Why the Standard Library looks the way it does
- How to write code using the Standard Library (via examples)
- Helping you to become a better C++ programmer
What This Talk is NOT About

• Introductory C++ Programming
• Object Oriented Programming (the library purposely has a very evident non-object-oriented feel!)
• Detailed contents of the headers (we prefer code samples)
• Language Wars
• Alexander Stepanov
C++
ISO C++?

- C++ will be accepted as an official ISO standard sometime in 1998.
- Has been a moving target for too long: implementers attempt to keep up (sort of); developers face incompatibility problems.
- Old compilers and legacy code with outdated language features still in use :-(

Evolution of C++

- There have been many language changes since 1990 that many people are not aware of, such as templates, exceptions, `bool, true, false, explicit`, new-style casts, *The Standard Library*, namespaces, RTTI, member templates, `typename`, declarations in `if` and `while` conditions, explicit instantiation, new keywords, ...
Simple Example 1

```cpp
#include <iostream>
#include <string>

int main(int argc, char** argv)
{
    std::string name;
    if (argc > 1) name = argv[1];
    else std::cin >> name;
    std::cout << "Hello, " + name;
    return 0;
}
```
```
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char** argv)
{
    string name;
    if (argc > 1) name = argv[1];
    else cin >> name;
    cout << "Hello, " + name;
    return 0;
}
```
LIBRARY OVERVIEW
Motivation

- C++ is too popular to not have a standard library
- Everyone, it seems, has written wrappers for everything (witness too many incompatible and buggy string classes)
- The Standard C++ Library should contain the Standard C Library as a subset
Standard Library Design (1 of 2)

- Provides support for language features (e.g. RTTI, memory management)
- Supplies implementation-dependent information (like limits)
- Supplies functions that you wouldn’t write in C++ itself so they can be optimized for a particular platform (e.g., sqrt, memmove)
Standard Library Design (2 of 2)

• Supplies non-primitive facilities to encourage portability (e.g. containers, sort functions, I/O streams)
• Has conventions for extending the facilities it does provide
• Is not stuffed with non-universal facilities such as graphics and pattern matching
Structure of the Library

• The Standard Library is comprised of 50 modules (18 are from C):

  <algorithm>, <bitset>, <cassert>, <cctype>, <cerrno>, <cfloat>, <ciso646>,
  <climits>, <locale>, <cmath>, <complex>, <csetjmp>, <csignal>, <cstdarg>,
  <cstddef>, <cstdio>, <cstdlib>, <cstring>, <ctime>, <cwchar>, <cwctype>,
  <deque>, <exception>, <fstream>, <functional>, <iomanip>, <iostream>,
  <istream>, <iterator>, <limits>, <list>, <map>,<memory>, <new>, <numeric>, <ostream>, <queue>, <set>, <sstream>,
  <stack>, <stdexcept>, <streambuf>, <string>, <typeinfo>, <utility>, <valarray>,
  <vector>
Logical Organization

- It is useful to group the 50 modules into ten informal categories:
  - Containers
  - General Utilities
  - Iterators
  - Algorithms
  - Diagnostics
  - Strings
  - Input / Output
  - Localization
  - Language Support
  - Numerics
TOUR OF THE LIBRARY
Containers

- The Standard Library's container classes use templates (genericity) and *not* inheritance! (No abstract base container class: containers simply support a standard, recognizable set of basic operations)
- Design is "the result of a single-minded search for uncompromisingly efficient and generic algorithms"
Containers

- `<vector>` one-dimensional arrays
- `<list>` doubly-linked lists
- `<deque>` double-ended queues
- `<queue>` FIFO queues and priority queues
- `<stack>` stacks
- `<map>` dictionaries (associative arrays)
- `<set>` sets
- `<bitset>` bit sequences
#include <iostream>
#include <list>
#include <string>
using namespace std;

int main(int, char**)
{
    list<string> names; // default constructor makes it empty
    names.push_back("dva"); names.push_front("odin"); names.push_back("tri");
    for (list<string>::iterator i = names.begin(); i != names.end(); i++)
        cout << *i << '
';
    return 0;
}
```cpp
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main(int, char**) {
    map<string, int> m;  m["juan"] = 19; m["svetlana"] = 26;
    cout << m["ciaran"] << '\n';
    map<string, int>::iterator i = m.find("juan");
    if (i != m.end()) cout << (*i).second << '\n' << m.size() << '\n';
}
```
Container Interface

• Standard Containers are all template classes which contain
  – typedefs `iterator`, `reverse_iterator`, and others
  – `empty()`, `clear()`, `erase()`, `size()`, `max_size()`, `begin()`, `end()`, `rbegin()`, `rend()`, `swap()`, and `get_allocator()`

• Certain containers have other members

• There is no hierarchy of containers!
Utilities, Iterators and Algorithms

- `<utility>` operators and pairs
- `<functional>` function objects
- `<memory>` allocators for containers
- `<iterator>` iterators
- `<algorithm>` general algorithms

The header `<cstdlib>` contains `bsearch()` and `qsort()` which are underpowered, useless and inefficient.
Some Algorithms

- `<algorithm>` contains, among others,

  ```cpp
  for_each(), find(), find_if(), count(), count_if(), search(), equal(), copy(), swap(), replace(), fill(), remove(), remove_if(), unique(), reverse(), random_shuffle(), sort(), merge(), partition(), binary_search(), includes(), set_union(), make_heap(), min(), max(), next_permutation()
  ```
#include <iostream>
#include <algorithm>
#include <functional>
#include <vector>
using namespace std;

int main(int, char**) {
    vector<int> a; for (int i = 0; i < 100; i++) a.push_back(i);
    random_shuffle(a.begin(), a.begin()+75);
    for (int i = 0; i < a.size(); i++) cout << a[i] << ' ';
    sort(a.begin(), a.end(), greater<int>());
    for (int i = 0; i < a.size(); i++) cout << a[i] << ' ';
}
Diagnostics

- `<stdexcept>` defines some standard exception classes thrown by many library operations
- `<cassert>` contains the `assert()` macro
- `<cerrno>` C-style error handling, needed to support legacy code
Strings

- The header `<string>` defines the template class `basic_string` and the classes `string` and `wstring`, which are instantiations of `basic_string` with `char` and `wchar`
- Strings have real copy semantics, you can assign using `=`, compare with `<=` and `>`, etc.
- Prefer strings to error-prone C-style char pointers
#include <iostream>
#include <string>
using namespace std;

int main(int, char**)
{
    string s1 = "Hello", s2("Goodbye"), s3, s4(s2, 4,3);
    s3 = s1; s3[1] = 'u';
    cout << s1 << ' ' << s3 << s2.length() << '\n';
    string message = s1 + ',' + " then " + s2;
    message.replace(7, 4, "and");
    cout << message << s4 << ' ' << s2.find('y') << '\n';
}
Input/Output

- `<ios>` basic stream types and ops
- `<streambuf>` buffers for streams
- `<istream>` input stream template class
- `<ostream>` output stream template class
- `<iostream>` standard streams like cin and cout
- `<fstream>` files to/from streams
- `<sstream>` strings to/from streams
- `<iomanip>` some stream manipulators
int main(int, char**) {
  ifstream f; double x; f.open("numbers.txt");
  if (!f) throw new runtime_error("missing file");
  while (true) {
    f >> x;
    if (f.bad()) throw new runtime_error("corrupted");
    if (f.fail()) {
      if (f.eof()) break; else throw new runtime_error("junk");
    }
    cout << fixed << setprecision(4) << x << '\n';
  } // note stream f closed in destructor
} // note catching and reporting runtime_errors omitted for space
Localization

• The header `<locale>` contains a class called `locale`, other classes such as `money_get` and `money_put`, and a number of operations such as `isalpha()`, `isdigit()`, `isalnum()`, `isspace()`, `ispunct()`, `iscntrl()`, `isupper()`, `islower()`, `toupper()`, `tolower()`
Language Support

- `<limits>` numeric limits
- `<new>` dynamic memory management
- `<typeinfo>` RTTI support
- `<exception>` exception class

In addition there are several headers from the C library:
`<climits>`, `<cfloat>`, `<cstddef>`, `<cstdarg>`, `<csetjmp>`, `<cstdlib>`, `<ctime>`, `<csignal>`
Numerics

- `<complex>` a class for complex numbers and many global operations
- `<valarray>` numeric vectors and operations
- `<numeric>` generalized numeric operations: accumulate(), partial_sum(), adjacent_difference(), inner_product()
- `<cmath>` mathematical functions
- `<cstdlib>` C-style random numbers and abs(), fabs(), div()
CONCLUDING REMARKS
Advice

- Use the Standard Library in all your new work; port old code to practice if feasible
- Remember the "C-style" way is almost always inferior to the "C++-style"
- Compose your own quick-reference guide to library facilities
- Read the Advice sections (16.4, 17.7, 18.12, 19.5, 20.5, 21.10, 22.8) in Stroustrup's book
For More Information


(Credits: This whole talk is organized pretty much like Part III of the above book and borrows many of the reference tables from it)