A C++ library wish list

Bjarne Stroustrup

Texas A&M University

http://www.research.att.com/~bs

Overview

- What do we want?
  - A survey
- What would *I* like?
  - Some suggestions
- Teaching C++
  - To beginners
- What is “a system”?
  - An analogy
- A system for C++ libraries

4

What we have

- Lots
  - The C++ standard library
    - C++98
    - C++03
  - Boost
  - Corporate foundation libraries
  - GNU
  - Lots of open-source libraries
  - Lots of commercial libraries
- We do not have a system
  - We (mostly) have an unordered sea of non-interoperable libraries
    - The standard library was supposed to help
    - Boost was supposed to help

5

What libraries do “people” want?

- A quick survey
  - Only 30-odd libraries
    - Some huge
    - Some with dozens of significant sub-parts
  - Far too narrow a group
    - Just 15 responders
    - Too many
      - students
      - WG21 fans
    - Too few
      - industrial users
      - scientists and engineers
      - “mainline PC application builders”
  - Not a bad “top of the ice berg”

6

What do “people” want?

- “People” don’t clearly distinguish between
  - Library components
  - Language features
    - E.g., dynamic linking and loading support
  - Tools
    - E.g., leak detector
  - Programming environment
    - E.g., debugging support
- For many, those distinctions are artificial
  - Solutions tend to cut across the distinctions
    - E.g., is “lambda” a library or a language feature?
- Here, I’ll stick to traditional libraries
  - But great libraries may require tool and/or language support

What do people want
(C++0x and TR proposals)

- Networking
- Threading
- Unicode
  - “much more than C++0x offers”
- Date and time
- File system
  - “more protocols than boost”
- Hash tables
- Random numbers
- Safe casts

- Small surprises: I was never asked for
  - A smart pointer
  - A library relating to support of a particular programming style
  - A library aimed at supporting library building

Graphics/GUI

- GUI (gtkmm?)
- 2D rendering (Cairo?) Bezier curves
- 3D rendering (OpenGL?)
- 2D layout engine
- Bridge to other systems/libraries
- Computational geometry
- Rational numbers, real, fixed-point math

9

Business

- XML (parsing, generating, validating, transforming)
- Web programming (HTTP, HTTPS, email)
- Web services (SOAP, WSDL, UDDI)
- Bridge to other systems/libraries
- Plugin framework
- GUI
- Distribution (communications, serialization, resource discovery)
- Generic database connectivity and transactions
- Cryptography
- Authentication
- Generic scripting language interface (call, load and execute)
- Audio/video streams
- VB-like string manipulation

10

Concurrency/distribution

- Concurrency “beyond locks and semaphores”
  - lock-free, wait-free containers and algorithms
- Generic database connectivity and transactions
- Web programming
  - HTTP, HTTPS, email
- Distribution
  - communications, serialization, resource discovery
- Web services
  - SOAP, WSDL, UDDI
- Cryptography
- Authentication
- Shared memory
- Library for querying machine about hardware and OS resources
- Bridge to other systems/libraries

11

Math

- Matrix library
- Bigint, rational numbers, real, fixed-point math
- Numerical methods
- Computational geometry
- A Mathlab library
- Physical units
- Better formatting (“type-safe printf”)
- Bridge to other systems/libraries
- Math special functions (as in TR1)

12

Etc.

- GUI (gtkmm?)
- Generic database connectivity and transactions
- VB-like string manipulation
- Command-line parser
- Neural networks
- Library for querying machine about hardware and OS resources
- Plug-in framework
- C++ parser and transformer
- Memory mapped files, random-access

13

Obvious observations

- Different people needs different parts of that
- We are not going to get all of that this year
  - What do we *really* want?
  - What do we want *first*?
- Anything even a hundreds of that size needs an overall structure
  - It is hard to use separately developed libraries
    - Error handling
    - Whose vector/array/list/iterator/smart-pointer do I use?
    - What’s an “event”?
    - How do you download and install?
    - How do you know if two libraries will work together?

14

What would *I* want?

- Who?
  - Bjarne the language designer?
  - Bjarne the standards geek?
  - Bjarne the teacher?
  - Bjarne the computer scientist?
  - Bjarne the programmer?
- You will face similar choices/alternatives
  - Draw from your personal experience, but work for the larger community

15

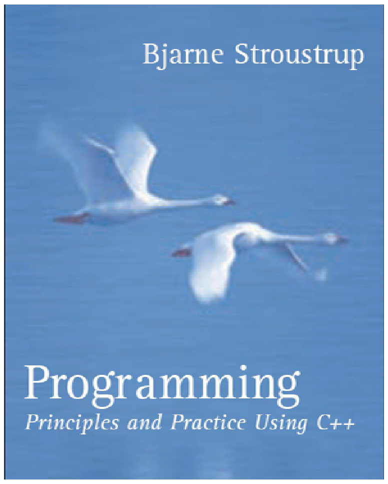
What *I* would like

- A "standard" platform for portable C++
  - Systems programming
    - Operating system access
    - Networking
    - …
  - Support for foundation areas
    - Text processing
    - Graphics
    - Linear algebra
    - …
  - Support for inter-library communication
- Much already exists, but
  - I can't find "it"
  - "those components" don't interoperate
  - I have to figure out how to install it

16

Libraries in teaching programming

- What I have been up to
  - Teach good concepts
  - Teach good techniques
  - Teach what scales
  - Teach novices
    - Some have never seen a line of code
    - Some “know everything”
  - Teach what have a chance of getting understood
    - Given just three months
  - Teach in a way that encourages hard work



17

Teaching

- Libraries are key
  - day 1: **string**
  - day 2: **vector**
  - week 3: Exceptions (my **vector** is range checked)
  - week 3: **sort()**
  - week 5: **iostreams**
  - week 6: Graphics
  - week 8: GUI
  - week 9: Pointer/array/**new**
  - week 10: **vector**, **list**, **array**
  - week 11: **find()**, **sort()**, **accumulate()**, **map**, **set**
  - week 12+: **regexp**, **Matrix**, etc. for “special projects”
  - Random numbers are needed for exercises (from about week 8)

18

Teaching

- Yes that can be done
  - Barely
  - 1000++ students have passed over three years
  - Also without me as the teacher

Teaching

- “I use
    - **#include “std_lib_facilities”**
      - Indirectly use of lots of std libraries
      - From day #1
    - **#include “Graph.h”**
      - Indirectly **FLTK.h**
    - **#include “GUI.h”**
      - Indirectly **FLTK.h**
    - **#include “Matrix.h”**
      - “homebrew” n-D matrices with simple mathematical operations
    - **#include “boost/regex.h”**

Teaching

- "Private libraries" are a real turnoff/barrier-to-use
  - Some students think that "not part of the standard" means
    - "not standard"=="not portable"=="not good"=="not worth learning"
  - Professors reject texts based on "private libraries"
    - Student reasons + too much work
- Distribution and installation are problems
  - My students use Windows, Macs, Linux, Unix
  - "self study" learners will use a wider range of systems/compilers
  - Even **#include** is a barrier
- Documentation is a problem
  - No standard
  - Typically written so that it's useful only late in the learning process
  - Hard to find a path from text/code to relevant documentation

21

We need "a system"

- We need
  - some order/structure among "interchangeable parts"
  - a comprehensible set of "distributions"
  - a (relatively) simple definition of "interchangeable part"
  - an (almost) universal set of guarantees and conventions

22

What is “a system”?

- How has other industries provided “systems”?
  - A good analogy would be a field where system is something in which
    - a tool is composed out of compatible parts
    - A tool is used by one or more people
      - with varying interests and skill levels

“Entry level”

- Good value for money
- Lots of support for novices
  - Basic use very easy
  - If the novice can’t produce good results “right out of the box” sales will plummet
- Lots of features
  - Something for everyone
  - A need to impress
- “Looks” matter
- “not too big or heavy”
  - “Smallest SLR” is an advertising line

“Enthusiasts”

- Emphasis on quality of results
  - Less emphasis on “features”
  - More emphasis on “quality of features”
- Serious user skills needed for best results
- Some support for relative novices
  - But not much, less than in “enthusiast”
- Emphasis on available add-on features

P.S. I do not own or use a dSLR

25

Professional

- Designed to produce superb results in the hands of a pro
- Endless features for specialized tasks
  - Many optional
- Built like a tank
  - For tasks where failure is expensive
- Expensive
  - The cost of equipment doesn’t matter if it makes the difference between success and failure

26

"The system" offers endless add-ons

- Different
  - "things"
  - qualities

27

One size does *not* fit all

- "Bigger" isn't always better
- "Small" isn't always good
- "Easy" isn't always good
- "Lots of control" isn't always good

The best is the enemy of the good

28

“The masses”

- Cheap
- Easy
- “Cute”
- Tuned for instant gratification
- Lots of features
  - Rarely used
  - Essential for sales
- Self-contained
  - Few if any add-ons
- A complete unit
  - Nothing interchangeable here
- “Brand”
- Some similarity to SLRs
  - Technology
  - User interfaces

29

Distribution and installation

- “It’s all in the box”
  - For some definition of “all”
  - For a variety of boxes
  - Note the manual and tutorial

30

How is a programming like a photography?

- Results depends on equipment
- Results depends on the user
  - Teaching, training, manuals, tutorials, examples, etc.
- Lots of “components”
- Users differ
  - Patience
  - Skills
  - Ability/willingness to pay
  - Individual needs, tastes, and skills, change over time
- Users’ needs differ
  - Definition of and cost of failure
  - Definition of and rewards of success
- “The system” is a family of parts
  - You can move from one level to another
  - You can move within a level (add component)
  - Once you have trust in the brand or lots of parts you don’t leave

31

What *I* want: A libraries system

- Foundations
  - **#include<boost/start>**
  - **#include<boost/entry>**
  - **#include<boost/enthusiast>**
  - **#include<boost/professional>**

- Extensions (if not part of a foundation)
  - **#include<boost/graphics/2d>**
  - **#include<boost/graphics/3d>**
  - **#include<boost/professional/graphics/3d>**
  - **#include<boost/linear_algebra>**
  - **#include<boost/XML>**
  - **#include<3rd_party/image_filtering>**
  - …

32

Distribution and installation

- Click here to download **<boost/start>**
- **Install("<boost/start>")**
- Use

```
#include<boost/start>
int main()
{
    cout<<"enter file name:\n";
    string name;
    cin>>name;
    fistream in(name);              // name.c_str() if you must
    vector<double> v;
    double val;
    while (in>>val) v.push_back(val);
    // …
}
catch (…) {
    cerr << "oops, exception!\n";
}
```

33

What are the basic distributions?

- Suggestion
  - Four levels (self-contained distributions)
    - Start (analog of point-and-shoot)
    - Entry level (analog of first SLR)
    - Enthusiast
    - Professional
- Why four?
  - One is not enough
    - Don't overwhelm learners (and professors)
    - Don't over-constrain experts
    - I *think* I have technical reasons
  - The distributions are ideally compatible subsets
    - Small "incompatibilities" might be desirable
  - Should there be a (5th) separate "experimental" distribution?

34

Start (“point and shoot”): Rationale

- We need to get people to do simple useful things ASAP
  - Simple text and numeric processing
  - Give notions type (built-in and user-defined), value, object, loop, algorithm, function, and correctness
- We need to protect people against
  - “silly errors”
    - They’ll make enough anyway
    - Errors are natural and inevitable (they have to learn to find and remove them)
  - known distracting conceptual problems
    - Not yet: unsigned, many integer sizes
    - Not yet: pointers (references are hard enough)
    - Not yet: non-trivial use of namespaces
    - Not yet: Graphics
  - drowning in complexity
  - boredom

35

Start (“point and shoot”)

- For people writing their first line of C++
  - **iostream**s
    - Simpler/nicer FP formatting would be nice
  - **string** (range checked)
  - **vector** (range checked)
  - A few algorithms
    - **find()**, **count()**, **sort()**, **min()**, **max()**, **sqrt()**, **pow()**, …
  - Simple numeric range checking
    - **int x = numeric_cast<double>(d);** **//** may throw
  - Simplest random number generator
  - **Bigint**
  - No pointers, arrays, or **new**

36

Beginner ("first SLR"): Rationale

- For simple (mostly personal) real tasks
  - Primary use: 1st and 2nd year teaching and learning
  - Correctness and simplicity are more important than performance
    - Type safe and checked by default
      - First mechanism for unchecked code? (I think so)
    - But there is no reason to be seriously inefficient
  - Keep the connection from source code to execution straightforward
    - Moderate and straight-forward use of
      - Class hierarchies
      - Generic programming
  - Quick write, compile, test loop
  - Not overwhelming
    - Try not to drown people with "occasionally useful features"
    - documentation
  - Single installation
- Not just for novices
  - I expect to use this for simple everyday tasks
  - Could become *many* people's entry into C++

37

Entry level ("first SLR")

- All of "start"
- Much of the STL (all?)
  - **vector**, **list**, **map**, **unordered_map**, **array**
  - Most or all algorithms
    - Do we need to add any for completeness?
- Regular expressions
- Better string manipulation
- Math
  - n-D Matrix class with basic arithmetic operations
  - Some basic linear algebra functions
  - **Real**
- Graphics (2D)
  - Basic shapes and operations
  - No memory leaks
  - On top of "real" libraries
    - note the plural
- Simple GUI
  - Event model, no memory leaks
  - Buttons, menus, inbox outbox
  - On top of "real" libraries
    - note the plural
- Simple measurement tools
  - **clock()**, …
- Networking (?)
- More?

38

Enthusiast: Rationale

- This is where most application developers should be
  - Here is where "whole system" issues become important
    - Concurrency
    - Data bases
  - Solid, well-performing libraries
  - Many optional add-ons
  - Not tools, just libraries (?)

39

Enthusiast

- All of "Entry level"
- Concurrency
  - C++0x threads, mutexes, async message exchange, etc.
  - Message queue
  - Task system (thread pool, work stealing)
- Networking (?)
- Serialization
- Database interface
- Scripting interface (?)
- XML
- Unit test framework
- Fixed-point type
- More – but what?

40

Professional: Rationale

- This is where the tool and library builders should be
  - Performance often critically important
  - “bleeding edge libraries” as add-ons
    - Experimentation (?)
      - it should be easier to get something into “professional” than “enthusiast”
  - “Whatever it takes to get the work done”
    - Correctness
    - Performance
    - Can/must assume skilled programmers
  - Tools

41

Professional

- Library building components and tools
- A typed abstract syntax tree for C++ (IPR)
- Parser library/tools
- Add-ons
  - Sparse matrices
  - Parallel programming
  - Business support
  - …

42

What makes a system?

- Interfaces and conventions for interchangeable parts
- User-interface elements

43

Common base

- Error and resource handling
  - Exceptions and RAII
    - Just don’t leak
- Data exchange (keep it simple and efficient)
  - **std::vector**
  - **std::array**
  - **std::string** (good enough)?
  - **std::pair<string,string>**?
  - Some kind of stream (for non-shared memory communication)
- Implementation information (tricky)
  - Operating system
  - Number of processors
  - Available memory

44

No hierarchical order

- You can't build professional libraries on top of beginner ones
  - The protection will get in the way
  - You need many more parameterization opportunities and features
- You can't build beginner libraries on top of professional ones
  - The complexity will "shine through"
    - Error messages
    - Debugging
    - Build times
  - Minimize the use advanced features in the interfaces to beginner libraries
    - The novices will want those interfaces explained
  - The size will be obvious

45

Common base

- Type safety
  - Statically safe if possible
    - But never assume your users understand type theory
  - Dynamically safe optionally
    - Make it easy to enable checking
      - 25% for checking is fine for most uses
      - 10* for checking is fine for novices and for debugging
  - No inherently unsafe interfaces
    - Always "pass" sufficient information for complete checking
  - For "professional level" only
    - Beat the gold standard in each field
      - in flexibility and performance
      - Whatever it takes
- Don't require derivation for everything
- Don't parameterize everything
- Use platform implementations of the standard library
  - Cooperate; don't compete

46

Common base – Stability?

- How important?
- Frequent updates (yearly?)
- Binary compatibility?

47

Purpose

- To have "the system" used to "do good", to
  - do what haven't been done before
  - do things cheaper
  - do things more reliably
  - increase quality
  - enable individuals do better
- ***Not*** "to have the most beautiful, fastest, most general, most buzz-word compliant system"

48

“Words of wisdom” (I hope)

- “Keep it as simple as possible, but no simpler”
  - A. Einstein
- “Strive for intellectual cohesiveness”
  - Lawrence Crowl
- “Simplicity is the ultimate sophistication”
  - Leonardo da Vinci
- “You can build a safe system on top of a fast system; you can’t build a fast system on top of a slow safe system”
  - anon
- “Make simple things simple”
  - B. Stroustrup
- “The best is the enemy of the good”
  - Voltaire

49

Thanks!

- Really, too many to list
  - “everyone who has ever built a library for use by others”
    - good or bad, we learn either way
  - David Wheeler: first paper on library design, 1951
  - …
  - Bell Labs
  - …
  - WG21
  - …
  - Boost
  - …

50