Evolving a language in and for the real world

Bjarne Stroustrup
Texas A&M University
http://www.research.att.com/~bs
Overview

• 1951-1978: Prehistory – Aims and Ideals
• 1979-1990: The early years – C with Classes and C++
• 1998-2008: Living in the real world – C++0x
8000+ Programming Languages

- C++’s family tree (part of)

- And this is a gross oversimplification!
Programming languages

- A programming language exists to help people express ideas
  - Programming language features exist to serve design and programming techniques
  - The real measure of value is the number, novelty, and quality of applications
Assembler – 1951

- Machine code to assembler and libraries
  - Abstraction
  - Efficiency
  - Testing
  - documentation

THE USE OF SUB-ROUTINES IN PROGRAMMES

D. J. Wheeler

Cambridge & Illinois Universities

Worthwhile the prime objectives to be born in mind when constructing them are simplicity of use, correctness of codes and accuracy of description. All complexities should-if possible—be buried out of sight.
Fortran –1956

• A notation fit for humans
  – For a specific application domain
    • \( A(I) = B(I) + C \times D(I) \)
  – Efficiency a premium
  – Portability
Simula –1967

• Organize code to model “the real world”
  – Object-oriented design

• Let the users define their own types (classes)
  – In general: concepts map to classes
  – “Data abstraction”

• Organize classes into hierarchies
  – Object-oriented programming
C – 1974

- An simple and general notation for systems programming
  - Somewhat portable
  - Direct mapping of objects and basic operations to machine
    - Performance becomes somewhat portable
C with Classes –1980

- General abstraction mechanisms to cope with complexity
  - From Simula
- General close-to-hardware machine model for efficiency
  - From C
  - Became C++ in 1984
  - Commercial release 1985
ISO Standard C++

• C++ is a general-purpose programming language with a bias towards systems programming that
  – is a better C  
  – supports data abstraction  
  – supports object-oriented programming  
  – supports generic programming

• A multi-paradigm programming language
  – The most effective styles use a combination of techniques
C++ applications
(www.research.att.com/~bs/applications.html)

- Telecommunications
- Google, Amazon, …
- Microsoft applications and GUIs
- Linux tools and GUIs
- Financial
- Games
- PhotoShop
- Most browsers
- …

- Mars Rovers
- Marine diesel engines
- Cell phones
- Human genome project
- High-energy physics
- Micro electronics design and manufacturing
- …
What’s distinctive about C++?

• Stability
  – Essential for real-world software
  – 1985-2008
  – 1978-2008 (C and C with Classes)

• Non-proprietary
  – Yet almost universally supported
  – ISO standard from 1998

• Direct interface to other languages
  – Notably C, assembler, Fortran

• Abstraction + machine model
  – Zero overhead principle
    • For basic operations (e.g. memory access) and abstraction mechanisms
  – User-defined types receive the same support as built-in types
  – Standard library written in the language itself
    • And most non-standard libraries
Aims for C++

• Support real-world software developers
  – “better software now”
  – by “better” I mean correct, maintainable, efficient, portable, …

• Change the way people think about software
  – Object-oriented programming
  – Generic programming
  – Resource management
  – Error handling

• Functional, not academic, beauty
  – “even I could have designed a much prettier language” – B.S. 1984 or so
Ideals

• The fundamental ideals for good design
  – Represent ideas directly in code
  – Represent independent ideas independently in code
  – Represent relationships among ideas directly in code
    • Hierarchical
    • Parametric
  – Combine ideas expressed in code freely
    • where and only where combinations make sense

• C++
  – Make these ideals viable for the largest possible range of application areas
    • “viable” includes “affordable” and “on available hardware”
    • “viable” includes “performs as well as the gold standard in a given area”
      – e.g. Fortran for scientific computation and C for systems programming
    • “viable” includes “in the hands of ordinary programmers”
Language features – 1979-1990

- **C with Classes (1979-84)**
  - Function argument declarations and checking
  - `const` (also in constant expressions)
  - Classes
  - Derived classes
  - Constructors, destructors
  - `new` and `delete`
  - Inline functions

- **C++ (in 1983-86)**
  - Overloading (incl. `=`, `[]`, and `()`)
  - `virtual` functions
  - Type-safe linkage

- **C++ (1988-90)**
  - Templates
  - Exceptions

**Not in C until much later**

**Huge impact**

**Rather late**

Stroustrup - Columbia 9/30/9
Basic resource management

- A resource can be memory, file handle, lock, socket, etc.

```cpp
class vector {
    vector(int s); // constructor: validate arguments, acquire resources
    ~vector(); // destructor: release resources
    // ...
};

void f(int s) {
    vector v(s);
    // ...
}
```
Object-oriented programming

- Class hierarchies, dynamic lookup, and static interfaces

```cpp
class Shape {
    Point c; // common implementation detail: often a dumb idea
    Color col;
public: // common user interface
    virtual void draw();
    virtual void move(Point p) { c=p; }
    virtual void rotate(int deg);
    // ...
};

class Circle : public Shape {
    Circle(Point cc, Color co);
    void rotate(int) {} // nice optimal algorithm
    // ...
};
```
C++ ISO Standardization – Membership

• About 22 nations (8 to 12 at a meeting)
  – ANSI (US national committee) hosts the technical meetings
  – Other nations have further technical meetings

• Membership have varied
  – 100 to 200+
    • 200+ members currently
  – 40 to 100 at a meeting
    • ~60 currently

• Most members work in industry

• Most are volunteers
  – Even many of the company representatives

• Most major platform, compiler, and library vendors are represented
  – E.g., IBM, Intel, Microsoft, Sun

• End users are underrepresented
C++ ISO Standardization – Process

Formal, slow, bureaucratic, and democratic
  – “the worst way, except for all the rest”
    (apologies to W. Churchill)

Most technical work happens
  – in “working groups”
  – electronically between meetings
For C++, the ISO standards process is central

- Standard support needed for mainstream use
  - Huge potential for improvement of application code
  - For (far too) many “if it isn’t in the standard it doesn’t exist”
- Significant defense against vendor lock-in
- C++ has no rich owner
  - who can dictate changes, pay for design, implementation, marketing, etc.
- The C++ standards committee is the central forum of the C++ community
  - Endless discussions among people who would never meet otherwise
- The committee receives feedback from a broad section of the community
  - Much of it industrial
- The committee is somewhat proactive
  - Adds features not previously available in the C++ world
C++ ISO Standardization – Results

1998  ISO standard
      – 22-0 vote

2003  Technical Corrigenda
      – “bug fix release”; no new features

2008  Registration draft for C++0x
      – 2011?

• Technical reports
  – Decimal floating point (2008)
  – Library2
  – Modularity

1992  Covariant return types

1993  Run-time type identification (RTTI: `dynamic_cast`, `typeid`, and `type_info`)
      Declarations in conditions
      Overloading based on enumerations
      `namespaces`
      `mutable`
      New casts (`static_cast`, `reinterpret_cast`, and `const_cast`)
      A Boolean type (`bool`)
      Explicit template instantiation
      Explicit template argument specification in function template calls

1994  Member templates ("nested templates")
      Class templates as template arguments

1996  In-class member initializers
      Separate compilation of templates (`export`)
      Template partial specialization
      Partial ordering of overloaded function templates

The sum is far more significant than the parts
C++98 example: Resource management

- Standard library containers
  - with exception-safety guarantees (e.g., vector)
  - the techniques can be used by every user
- No resources are leaked
  - E.g. vector elements and file handles (handled by ifstream)
  - Destructors do cleanup
    - guaranteed, implicitly
  - Based on a simple and systematic view of resource management
    - Resources: e.g. locks, sockets, memory, thread handles, file handles
    - Exception safety guarantees
    - RAII

```cpp
void f(string s)
{
    vector<int> v;
    ifstream is(s);
    // ...
    int x;
    while (is>>x) {
        if (x<=0) throw Bad_value(x);
        v.push_back(x);
    }
    // ...
}
```
The STL

- Ideal: The most general and most efficient expression of an algorithm
  - Focus on algorithms
  - Separate algorithms from data
    - Using iterators
  - Go from the concrete to the abstract
    - Not the other way
  - Use compile-time resolution to eliminate overheads
    - Inlining and overloading
  - Where needed, parameterize with policies
    - E.g. sorting criteria
STL example: find_if

• Definition

```cpp
template<class Iter, class Pred>
Iter find_if(Iter first, Iter last, Pred p)
{
    while (first!=last && !p(*first)) // while not at end and predicate not met
        ++first; // advance to next element
    return first; // return the element reached
}
```

```cpp```
pi = find_if(v.begin(), v.end(), Less_than<int>(42));
if (pi!=v.end()) {
    // found it!
}
```cpp

Stroustrup - Columbia 9/30/9
C++0x: 2002-2008

• Overall goals
  – Make C++ a better language
    • for systems programming
    • for library building
  – Make C++ easier to teach and learn
    • generalization
    • better libraries

• Massive pressure for
  – More language features
  – Stability / compatibility
    • Incl. C compatibility

• Insufficient pressure for
  – More standard libraries
    • The committee doesn’t have the resources required for massive library development
C++0x: Areas of change

• Machine model and concurrency
  – Memory model
  – Threads library, asynchronous return
  – Atomic API
  – Thread-local storage
• Support for generic programming
  – `auto`, `decltype`, template aliases, Rvalue references, …
  – General and uniform initialization
  – Lambdas
• Etc.
  – improved `enums`
  – `long long`, C99 character types, etc.
  – …
• Libraries
  – Regular expressions
  – Hashed containers
  – …
C++0x: language features

- `decltype` and `auto` — type deduction from expressions
- Template aliases
- Move semantics (rvalue references)
- Static assertions (`static_assert`)
- `long long` and many other C99 features
- `>>` (without a space) to terminate two template specializations
- Unicode data types
- Variadic templates
- Generalized constant expressions (`constexpr`)
- Generalized initializer lists
- Scoped and strongly typed enumerations (`class enum`)
- Control of alignment
- `nullptr` — Null pointer constant
- A for-statement for ranges
- Delegating and forwarding constructors
- Thread-local storage (`thread_local`)
- Defaulting and inhibiting common operations
- Lambda expressions
- …

The whole is much more than its parts
Performance and convenience

template<class C, class V> vector<typename C::iterator> find_v(C& s, V v)
   // find all occurrences of v in s
{
    vector<C::iterator> res;
    for (auto p = s.begin(); p!=s.end(); ++p)
       if (*p==v) res.push_back(p);
    return res;
}

vector<string> m = { "Dennis", "Joe", "Brian", "Al", "Joe", "Bill" };  
for (auto x : find_v(m,"Bill"))
    if (x!= "Bill") cerr << "bug!\n";
Why did C++ succeed?

• Reasons
  – Low-level access plus abstraction mechanisms
    • Performance
    • Direct access to real hardware
    • Very general zero-overhead abstraction
  – C compatibility
  – A useful tool (from day #1)
  – Timing
  – Non-proprietary – ISO standard
  – Stable
  – Evolving

“Being best at one or two things is not enough, you must be good enough at everything someone consider important”
Why did C++ succeed?

- Popular non-reasons
  - Just luck
    - For 25 years!
  - AT&T’s marketing might
    - Must be a joke 😊
  - It was first
    - Except for Ada, CommonLoops, Smalltalk, Eiffel, Objective C, Modula-2, C, Fortran, ML, …
  - Just C compatibility
    - Never 100%
  - It was cheapest
    - Not for most of its lifetime (incl. all the early years)
What is C++?

Template meta-programming!

A multi-paradigm programming language

A hybrid language

It’s C!

Embedded systems programming language

Buffer overflows

Supports generic programming

Too big!

A random collection of features

An object-oriented programming language

Low level!
C++

A language for building software infrastructures and resource-constrained applications

A light-weight-abstraction programming language
Thanks!

- C and Simula
  - Brian Kernighan
  - Doug McIlroy
  - Kristen Nygaard
  - Dennis Ritchie
  - …
- ISO C++ standards committee
  - Steve Clamage
  - Francis Glassborow
  - Andrew Koenig
  - Tom Plum
  - Herb Sutter
  - …
- C++ compiler, tools, and library builders
  - Beman Dawes
  - David Vandevoorde
  - …
- Application builders
More information

• My HOPL-II and HOPL-III papers
• The Design and Evolution of C++ (Addison Wesley 1994)
• My home pages
  – Papers, FAQs, libraries, applications, compilers, …
    • Search for “Bjarne” or “Stroustrup”
• The ISO C++ standard committee’s site:
  – All documents from 1994 onwards
    • Search for “WG21”
• The Computer History Museum
  – Software preservation project’s C++ pages
    • Early compilers and documentation, etc.
      – http://www.softwarepreservation.org/projects/c_plus_plus/
      – Search for “C++ Historical Sources Archive”
C/C++ compatibility

- A constant sore point
  - Separate standards committees
    - A tragedy
  - Constant borrowing
    - Both ways
    - Often incompatibly
  - Widely demanded by users
    - Rightfully so
  - Widely despised by users
    - “Against OO”
    - “Against the spirit of C”