Evolving a widely used language
Why and how?

Bjarne Stroustrup
Texas A&M University
http://www.research.att.com/~bs
Beautiful Bergen
Thanks!
A programming language exists to help people express ideas

- To help build useful and/or interesting systems
- As problems change, a language must evolve (or die)
- Many more users implies different kinds of users and different problems
- Language features exist to serve design and programming techniques
C++ applications

www.research.stroustrup/~bs/applications.html

- Telecommunications
- Google
- Microsoft applications and GUIs
- Linux tools and GUIs
- Games
- Financial systems
- PhotoShop
- ...

- Mars Rovers
- Marine diesel engines
- Cell phones
- Human genome project
- Micro electronics design and manufacturing
- ...

Stroustrup - Bergen
The world of 1975-85

- Much work still done in assembler
  - Essentially all embedded systems programming
  - Most systems programming
  - Quite a lot of applications programming
- OOP – what’s that?
  - In both academia and industry
  - The few that have heard of it deem it
    - Just “slow graphics”
    - Unsuitable for ordinary mortals
    - Incapable of interoperate as part of a system
- Academia and industry are quickly drifting apart
  - It was not always so in CS; e.g. Dijkstra, Hoare, Backus, started out in industry
The idea of C++

• C + Simula
  – Direct map to hardware + abstraction
  – Efficiency + structure

• Known problems
  – Non-uniform handling of built-in and user-defined types in Simula
  – Lack of static type safety in C
  – No parameterized types or procedures in either

• Get something working and then improve on it
  – First non-research user after 6 months
  – C++ compiler in C++
  – Direct support of a variety of colleagues
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
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

Huge impact
Not in C until much later
Rather late
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
C++ ISO Standardization – Organization

• (ad hoc) Working groups
  – Core
  – Library
  – Evolution
    • Concurrency
• “mailings”
  – “papers” presenting issues and proposals
    • Hundreds each year; see WG21
• “reflectors”
  – Achieved mailing lists
• Many (even) more “ad hoc” activities
  – E.g. implementers presenting progress

Stroustrup - Bergen
C++ standardization – why bother?

- 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 massive 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
   – We hoped for C++09

2010  CD expected
   – Should lead to C++0B

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

- Stability and Compatibility
  - “make the language much better but don’t break my code”
- Scale
  - Million-line projects became common
  - Specification – precise and complete
  - Portability
- Resource management
  - Invariants, RAII
- Type safety
  - Containers
- Performance
  - Compactness
- Equal support for user-defined and built-in types
  - Value types, scoped objects
- User skills required
  - C++ should not be just expert friendly
Alternatives to ISO Standardization

• Corporate ownership
  – Maybe “softened” by customer involvement
• Tame standards bodies
• “Benign dictator for life”
  – “Benign”
• No change
  – Just get it right at first and don’t change
• Chaos
  – Dialects
What could be done: 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
**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
  - …
A feature too far

- Concepts
- High-level concurrency features
- Garbage collection
- Modules

- How much can be done to a widely used language?
  - We have pushed the envelope
  - Maybe that can’t continue?
    - If so, more for person issues than for technical issues
    - And the technical issues mostly relates to complexity from “feature interactions”
What kind of people participates?

• Idealists
  – To change the language and the world
  – Often busy at other things (essential), sometimes single issue (very bad), sometimes undisciplined (bad), often lasts just a couple of years (just enough to do something, good or bad)

• Damage controllers
  – To protect the language from the idealists

• Corporate representatives
  – To guard their corporation’s interests
  – Big companies differ from small companies

• Bureaucrats
  – People who like to go to meetings
  – Can be on time and keep long lists (essential skills)
What kinds of people participates?

- Lots of implementers
  - Compilers
  - Libraries
  - tools

- Few users (far too few)
  - Application builders
  - Educators
  - Researchers
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”