

Directions for C++0x

Bjarne Stroustrup

AT&T Research

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

Abstract

- The ISO C++ standard is now about two and a half years old. In about another two and a half years it could be revised, amended, or - of course - left alone. How should we expend our efforts to best help the C++ community? My view is that the primary effort should be directed at improving the standard libraries to be more supportive for general programming and also to provide a broader base for portable programs. I know that others have far more radical ideas and that others still are far more conservative. Are there general guiding principles that we can all agree on?
 - 45 minute+ discussion intro

Aim of this presentation

- Start discussion about the future of Standard C++
- Present one possible starting point
- Present a few concrete examples to “seed” technical discussion

Why mess with a good thing?

- The ISO Standard is good
 - but not perfect
- ISO rules require review
 - Community demands consideration of new ideas
- We face increasingly difficult tasks
 - We == programmers and system designers
- The world changes
 - and poses new challenges
- We have learned things since 1996
 - When the last of the ISO C++ features was proposed
- Stability is good
 - but the computing world craves novelty
 - Without challenges, the best people will depart for greener pastures

Prerequisites for goals

- Exciting enough to attract good people to the committee
- Ambitious enough to serve users well
- Compatible enough to make transition simple
- Specific enough to clearly state aim

If we do nothing about direction

- C++ will fossilize
- C++ will become de facto proprietary
- C++ will be dominated by bindings to external “standards”
- The committee will become a small club
 - Will focus on minute details of increasing irrelevance to programmers
- Changes will be made without direction

- Some say that this has already happened, but 1997-2000 was a deliberate period of calm to enhance stability
 - Now is the time to start discussing and planning
 - What’s the difference between stability and stagnation?

Prerequisites imply

- Focus on a few major topics
- Minor changes should be done to increase language and standard library uniformity and consistency
 - No major new language features are needed
- Changes should be focused on support for programming styles and for application areas
 - Not language technicalities

Overall goals

- Make C++ a better language for systems programming and library building
 - Rather than providing specialized facilities for a particular sub-community (e.g. numeric computation or Windows application development)
- Make C++ easier to teach and learn
 - Through increased uniformity, stronger guarantees, and facilities supportive of novices

Directions

- General
 - Minimize incompatibilities with C++98
 - Be as careful as the C committee was, or even more so
 - Zero-overhead principle
 - Maintain or increase type safety
- Language
 - Avoid major extensions
 - Make rules more general and uniform
 - Improve support for generic programming
 - Improve general guarantees (increase uniformity)
 - Improve support low-level embedded programming
- Library
 - Increase facilities of system-independent platform
 - Opportunistic, ambitious
 - Support distributed systems programming
 - Simple, clean, implementation-independent model
 - We need a notion of “if we support X, it must meet these requirements”
 - Not every system can support every standard library facility

Standard library ideas

- Elements of standard platform
 - set of resource handles supporting “resource acquisition is initialization”
 - directories, TCP/IP, advanced I/O (async, multiplex, memory map), ...
- Distributed computing
 - XTI (eXtended Type Information)
 - Threads
 - Remote invocation (incl. Async)
 - Remote instantiation, name server interface
- Make the standard library central to bindings to other systems
 - CORBA, SQL, ...
- Add a few “general utility” facilities
 - Hash_map
 - Pattern matching
 - “property”

Threads

- Standard library classes
- No core language facilities (no keywords)
 - Thin interface to platform threads
 - (local) lock objects
 - Example of resource object (RAII)
 - Language guarantees for general concurrency

RPC: Remote Procedure Call

- Message passing interface

```
// z = m.foo(x,y);
```

```
Handle<typeof(z)> h = client(m).send(message(&M::foo,x,y)); z = h.get();
```

- Async class

- // Handle<typeof(z)> h = async(m).foo(x,y);

```
Handle<typeof(z)> h = async(m).send(message(&M::foo,x,y));
```

```
// ...
```

```
if (h.ready()) z = h.get();
```

- Interface to a variety of implementations/libraries

- tool for mapping to message passing

- Can we standardize a tool?

- (sure: think of compiler, linker, ...)

XTI: eXtended Type Information

- A set of classes/objects representing C++ declarations:

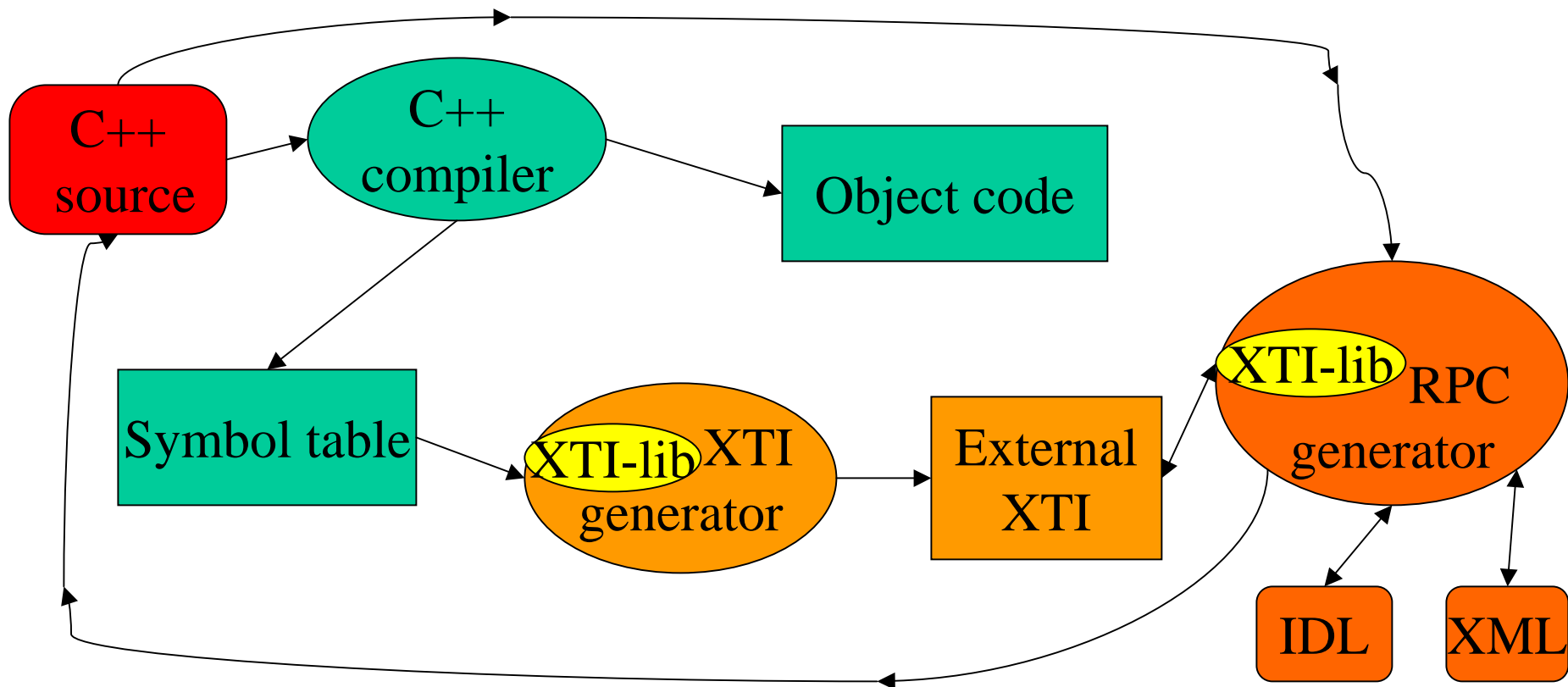
```
Program prog(“my_types”);  
if (prog.global_scope[“My_vec”].is_class()) { /* ... */ }  
for (scope::iterator p = prog.begin(); p!=prog.end(); ++p) p->xti_name();
```

- Represent anything that the C++ type system can
 - Classes, enumerations, typedefs, templates, namespaces, functions, non-local variables
 - Not code, local types, local variables

XTI: eXtended Type Information

- Use for run-time resolution
 - Extend `type_info`
 - or key on `typeid`
 - or lookup based on name
- Use for program analysis
 - E.g. ODR verification, version consistency checking
- Use for program transformation
 - E.g. IDL generation

Generation of inter-object communication code



How do we get libraries to include?

- The committee is not a good forum for design
 - Wait and hope?
 - Each of us go off and write one?
 - Look for existing ones to co-opt/adopt?
 - Write “requirements” and ask for proposals?
- Obvious potential problems
 - Lack of experience for new libraries
 - Lack of compatibility for old libraries
 - Proprietary aspects of libraries

Core language ideas

- Increase consistency
 - identical lookup for functions and function objects
 - decrease variation between implementations to increase portability
 - Minimize “implementation dependent/undefined/...”
- Improve support for generic programming
 - **Typedef templates**, maybe **typeof()**
- Remove embarrassments
 - Frequent questions, frequent novice errors

Avoid embarrassments

- Trivial solutions to trivial beginners' problems
 - **string** to **int** and **int** to **string** (without **stringstream**)
 - a **vector** and a **string** that are range checked by default
- Frequent questions, frequent novice errors
 - Prohibit default copying of objects with destructors
 - catch violations at point of use
 - Give a class with virtual functions a virtual destructor by default
 - Allow unions of arbitrary type
 - catch violations at point of use
 - **vector<list<int>>** (note: no space)
 - Prohibit hiding of virtual function
 - Introduce notation for “overload function that is (already) virtual”
- Explicitly admit GC as a valid implementation technique
 - Don't make the C++ semantics dependent on GC
 - Encourage GC as an option on an implementation

Core language ideas

- Encourage implementers to introduce new features in a common defined order
 - To avoid temporary decrease of portability due to different order of introduction
 - Assuming that implementers will “jump the gun”

C/C++ compatibility

- Offer merger with ISO C committee
 - A common language would benefit community
 - C/C++ isn't a language – the notion does harm
- Politically very difficult
 - Both sides must give up something
 - Have a small joint group draft principles
 - to be proposed to C and C++ groups separately
- Technically non-trivial
 - Obvious potential problems
 - Type-safety
 - C arrays

Not mentioned

- Standard GUI
 - Politically and technically too hard
 - Can XTI support GUI work? (yes)
 - Standard library callback mechanism?
 - Slots and signals?
- Persistence
 - Can XTI help? (yes)
- Module mechanism
 - Dynamic linking/loading
 - Way of controlling names being imported and exported
 - Incl. macros
- C++ ABI
 - A platform issue
- C++ virtual machine (VM++)

Think carefully

- Where will you be in 5 years? 10 years?
- Where would you like to be then?
- Where will your users/customers be then?

- Doing nothing is an option, but not an attractive long-term option
- We need to start planning/designing/experimenting years before the need for change becomes undeniable

- Significant change will happen whatever the committee does
- I'd like for the committee remain the/a center of the C++ community