Improving Standard C++ for the Physics Community

CHEP 2004

Interlaken, Switzerland

Marc Paterno & W. E. Brown

Fermi National Accelerator Laboratory

30 September 2004

Outline

1. Goals, motivation, overview

2. A sampling of what's likely ahead for C++
   - Enhanced function-declarations to improve performance
   - Random-number toolkit to improve domain support
   - Mathematical special functions to improve domain support
   - Shared-ownership pointers to improve interoperability
   - Move semantics to improve performance

3. Further developments
   - Additional proposals
   - In the standard library
   - In the core language
   - On the horizon

4. Conclusion

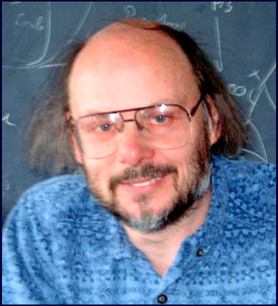
How did Fermilab get involved?

- Then Walter invited his friend Bjarne Stroustrup to speak at Fermilab for ACAT 2000 . . .

- . . . and Bjarne said if the scientific community wants changes, we should get involved — so Fermilab did!

How did Fermilab get involved?

- Then Walter invited his friend Bjarne Stroustrup to speak at Fermilab for ACAT 2000 . . .



- . . . and Bjarne said if the scientific community wants changes, we should get involved — so Fermilab did!

Goals for this talk

- To inform the physics community of:
  - The possible future directions of Standard C++
  - Fermilab's role in influencing this direction
- To encourage greater participation in setting this direction:
  - So that features important to *us* are included
  - So that these features are compatible with *our* use
- To note proposed features of special interest to *us*

C++ standards bodies and their work

- The "standards committee" is really multiple committees:
  - ISO JTC1-SC22/WG21 is the international standards committee
    - Its members are national standards bodies (currently 15)
  - ANSI NCITS/J16 is the US national standards committee
    - Fermilab is a voting member of J16
- From the 1998 publication of the C++ Standard until 2001 was a "period of calm to enhance the stability of the language"*
  - Nonetheless busy: identified, evaluated, consolidated many hundreds of editorial and minor technical improvements
  - Resulted in an updated standard: C++03
- Since 2001 the committee has been working toward C++0*x*:
  - Still soliciting and evaluating proposals for extensions to the language and to the standard library
  - A Technical Report on the standard library is likely to be voted out of committee in October 2004

*B. Stroustrup

Outline

1. Goals, motivation, overview
2. A sampling of what’s likely ahead for C++
   - Enhanced function-declarations to improve performance
   - Random-number toolkit to improve domain support
   - Mathematical special functions to improve domain support
   - Shared-ownership pointers to improve interoperability
   - Move semantics to improve performance
3. Further developments
   - Additional proposals
   - In the standard library
   - In the core language
   - On the horizon
4. Conclusion

Enhanced function-declarations to improve performance

- **Observation**: Compilers are often unable to “see through” function calls in order to optimize code:
  - They may *not* safely assume that function calls are benign
  - Whole-program analysis is expensive and often impossible
  - Optimization opportunities are lost
- **Observation**: Programmers frequently have the information the compiler lacks but needs for better optimizations
- **Proposal**: Introduce new qualifiers pure[†] and nothrow:
  - To let a programmer declare a function’s relevant characteristics
  - To have the compiler verify this claimed behavior
  - To obtain better-optimized code from compiler analysis of this new information at each function call site

```
z = f(x) + f(y) // can this be done in parallel?
```

[†] pure = has no side-effects

Random-number toolkit to improve domain support

- **Observation**: High-quality pseudorandom-number generation is required in many fields:

  gaming, testing, security, numerics, . . . , physics
- **Observation**: The existing standard facility (`rand` and `srand`) is grossly inadequate for common uses:
  - Sequences are not reproducible across implementations
  - Typical implementations exhibit poor “randomness”
  - Only uniformly distributed random integers are provided

Features of the random-number toolkit

- **Proposal**: Provide a flexible and extensible framework for manipulating *engines* and *distributions*:
  - An engine is a “source of randomness”
  - A distribution creates, from the output of an engine, a stream of random variates with prescribed properties
  - It is easy to add user-defined distributions
  - Experts can add new engines
  - Added components work seamlessly with existing components
- **Proposal**: Include engines and distributions important to our community, and with characteristics important to us:
  - Engines’ outputs are guaranteed to be portable and reproducible
  - Distributions’ outputs are guaranteed to be reproducible

Show list of engines and distributions

Mathematical special functions to improve domain support

- **Observation**: Current mathematics support is scant — only a small handful of transcendental functions
- **Proposal**: Add support for some of the most important functions of mathematical physics
- This will be the first significant enhancement to `<math.h>` in circa 30 years
- Being (favorably) considered also by the C standards committees:
  - Designed to be compatible with C . . .
  - And thus is immediately compatible with many (most?) other programming languages

Features of the special functions proposal

- Why standardize special functions?
  - Quality and reliability; professional attention to important details often overlooked by many application programmers, *e.g.*:
    - Performance (both in speed and in space)
    - Corner cases that may need special handling
    - Error-reporting and -handling
  - Portability and re-use; let us focus on physics problems rather than on issues related to infrastructure or platform dependency
- Designed to follow C++ style, special functions are *functions*; other designs would:
  - Violate the zero-overhead principle
  - Treat users' extensions as second-class citizens

Show list of functions

Shared-ownership pointers to improve interoperability

- **Observation**: No pointer type having shared-ownership semantics is uniformly available today:
  - But such types are often needed, so . . .
  - We re-invent and produce unique versions, a situation akin to the days before `std::string`, but . . .
  - Implementation (even by experts) is known to be "exceedingly difficult,"[‡] especially in the presence of exceptions, so . . .
  - We waste time re-inventing the wheel (and sometimes we make square wheels), but then . . .
  - Different libraries can't communicate using them, because each has its own implementation
- **Proposal**: Provide a shared-ownership smart pointer:
  - Automate most details of dynamic lifetime management
  - Based on years of experience with Boost's[§] shared_ptr

[‡] H. Sutter

[§] `http://www.boost.org`

Move semantics to improve performance

- **Observation**: Copying an object can be expensive (*e.g.*, deep copies of contained objects)
- **Proposal**: Reduce cost by allowing choice of moving or copying
- Define move as the ability to cheaply transfer the value of an object from a source to a target, with no regard for the value of the source after the move
- Move semantics are typically applicable when the source object:
  - Will be destroyed shortly after the copy, or . . .
  - Will get a new value shortly after the copy
- Experimental implementation has seen (in realistic cases) a 10- to 20-fold speed increase[¶]

```
std::vector<std::string> greetings(10, "hello");
greetings.insert(greetings.begin(), "bonjour");
```

[¶] H. Hinnant, Metrowerks

Outline

1. Goals, motivation, overview

2. A sampling of what's likely ahead for C++
   - Enhanced function-declarations to improve performance
   - Random-number toolkit to improve domain support
   - Mathematical special functions to improve domain support
   - Shared-ownership pointers to improve interoperability
   - Move semantics to improve performance

3. Further developments
   - Additional proposals
   - In the standard library
   - In the core language
   - On the horizon

4. Conclusion

Additional proposals

- The committee is evaluating many other proposals
- We are still receiving new proposals
- In general, we intend to:
  - Keep to the zero-overhead principle
  - Minimize incompatibilities with C++03 and C99
  - Maintain or increase type safety
  - Improve portability, especially by minimizing “implementation-defined” and “undefined” behavior

Proposed additions 1: standard library

There are too many proposed additions to the standard library to discuss them all here . . .

- Random numbers
- Polymorphic function-object wrappers
- Type traits
- Enhanced function-binders
- Unordered (hashed) containers
- Regular expressions
- Mathematical special functions
- Tuple types
- Shared-ownership smart pointers
- Member-pointer adaptors
- Reference wrappers
- Function-result type-traits

Proposed additions 1: standard library

There are too many proposed additions to the standard library to discuss them all here . . .

- Random numbers
- Polymorphic function-object wrappers
- Type traits
- Enhanced function-binders
- Unordered (hashed) containers
- Regular expressions
- Mathematical special functions
- Tuple types
- Shared-ownership smart pointers
- Member-pointer adaptors
- Reference wrappers
- Function-result type-traits

I discussed only a few of them.

Proposed additions 2: core language

There are also too many proposed additions to the core language to discuss them all here . . .

- Enhanced function-declarations
- Compile-time reflection
- Forwarding constructors
- *Concepts* for generic programming
- Static assertions
- `decltype` and `auto`

- Move semantics
- Local classes as template parameters
- Literals of user-defined types
- Generalized initializer-lists
- Null-pointer constant
- Template aliases
- Dynamic libraries

Proposed additions 2: core language

There are also too many proposed additions to the core language to discuss them all here . . .

- Enhanced function-declarations
- Compile-time reflection
- Forwarding constructors
- *Concepts* for generic programming
- Static assertions
- `decltype` and `auto`
- Move semantics
- Local classes as template parameters
- Literals of user-defined types
- Generalized initializer-lists
- Null-pointer constant
- Template aliases
- Dynamic libraries

I discussed only a few of them

A sample of what else is on the horizon

- Computer floating-point arithmetic has historically been largely based on binary representation
- A recently-promulgated ISO standard promotes the cause of decimal floating-point arithmetic:
  - Primarily motivated by financial applications, but . . .
  - Also of interest to the scientific community
- Vendors have committed to new hardware (!)
  in support of decimal floating-point arithmetic
- Long-term view suggests:
  - Binary floating-point arithmetic may stagnate/fossilize, and . . .
  - Decimal arithmetic may come to dominate numeric types
- C++ is exploring language and library support for decimal arithmetic; many thorny problems need to be addressed

Outline

1. Goals, motivation, overview
2. A sampling of what's likely ahead for C++
   - Enhanced function-declarations to improve performance
   - Random-number toolkit to improve domain support
   - Mathematical special functions to improve domain support
   - Shared-ownership pointers to improve interoperability
   - Move semantics to improve performance
3. Further developments
   - Additional proposals
   - In the standard library
   - In the core language
   - On the horizon
4. Conclusion

Why we should continue to participate

- C++ continues to be of significant interest to physics:
  - Expressiveness
  - Performance
  - Significant community experience
- C++ is being enhanced in directions of substantive interest to us. Fermilab has been actively nudging it in these directions!
- Standard components benefit us all:
  - Require less in-house development/maintenance
  - Enhance efforts to share code
  - Allow us to focus on physics, not infrastructure
- Walter and I hope to be able to continue supporting our community in the C++ standards effort—and welcome support from others.

Online references

- The C++ committee has a public web site at `http://www.open-std.org/jtc1/sc22/wg21/` (a very few pages are not public)
- Recent committee papers are found at:
  - `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2004/`
  - `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2003/`
- Walter and I can be reached via e-mail:
  - Marc Paterno: paterno@fnal.gov
  - Walter Brown: wb@fnal.gov

Engines and distributions provided

Engines:

- Basic engines are: linear congruential, Mersenne twister, subtract-with-carry (“ranlux”)
- Basic engines can be modified or combined, using: discard block, xor combine

Distributions:

- integer uniform, floating-point uniform
- Bernoulli, binomial, geometric, negative binomial
- Poisson, exponential, gamma, Weibull, extreme value
- normal, lognormal, χ^2, Breit-Wigner, Fisher F, Student t
- histogram sampling, cumulative distribution function sampling

Return to talk

Special functions provided

- Bessel and Neumann functions (cylindrical and spherical, 1st and 2nd kinds)
- Legendre and associated Legendre polynomials
- Spherical harmonics
- Hermite polynomials
- Laguerre and associated Laguerre polynomials
- Gamma function
- Complete and incomplete elliptic integrals (1st, 2nd and 3rd kinds)
- Euler beta function
- Exponential integral
- Riemann zeta function
- Error and complementary error function
- Hypergeometric and confluent hypergeometric functions

Return to talk