C++ - Current language status

Robert Piszczatowski

DCC Labs (dcclabs.com)

24 february 2016

Who is that guy?

C++
THE C++ PROGRAMMING LANGUAGE
BJARNE STROUSTRUP
IBM
Programming – Principles and Practice Using C++

Pre iso times

- 1979 - c with classes

Pre iso times

- 1979 - c with classes
- 1983 - c with classes renamed to c++

Pre iso times

- 1979 - c with classes
- 1983 - c with classes renamed to c++
- 1985 - „The C++ Programming Language”

Pre iso times

- 1979 - c with classes
- 1983 - c with classes renamed to c++
- 1985 - „The C++ Programming Language”
- 1990 - „The Annotated C++ Reference Manual”

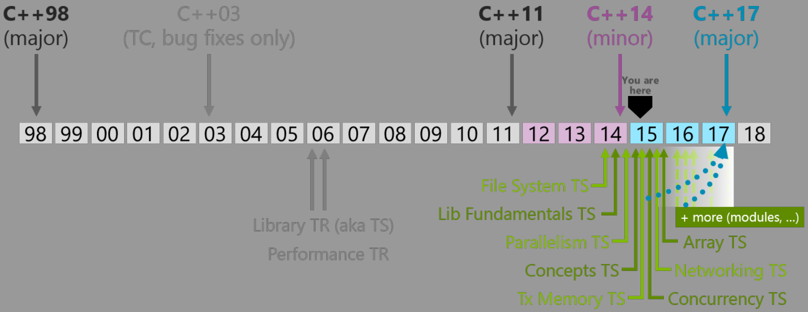
Pre iso times

- 1979 - c with classes
- 1983 - c with classes renamed to c++
- 1985 - „The C++ Programming Language”
- 1990 - „The Annotated C++ Reference Manual”
- 1991 - http://www.stroustrup.com/hopl2.pdf

Pre iso times

- 1979 - c with classes
- 1983 - c with classes renamed to c++
- 1985 - „The C++ Programming Language”
- 1990 - „The Annotated C++ Reference Manual”
- 1991 - http://www.stroustrup.com/hopl2.pdf
- 2005 - „The Design and Evolution of C++”

C++ timeline



C++98 (major)

C++03 (TC, bug fixes only)

C++11 (major)

C++14 (minor)

C++17 (major)

You are here

98 | 99 | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18

Library TR (aka TS)

Performance TR

File System TS

Lib Fundamentals TS

Parallelism TS

Concepts TS

Tx Memory TS

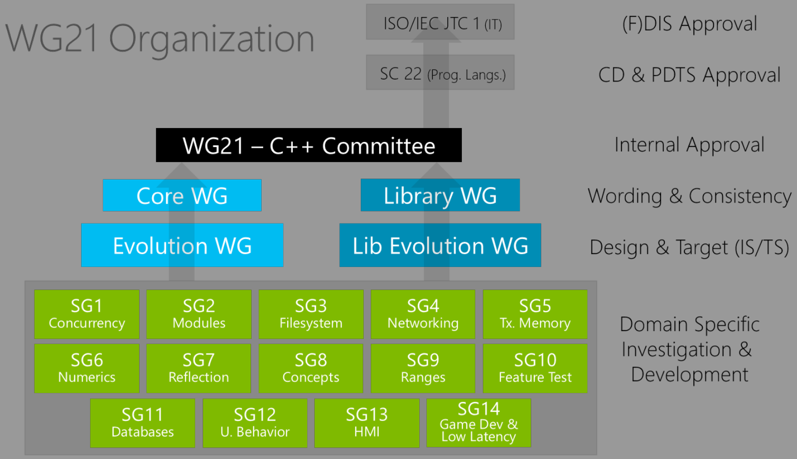
+ more (modules, ...)

Array TS

Networking TS

Concurrency TS

C++ committee structure



WG21 Organization

ISO/IEC JTC 1 (IT)

SC 22 (Prog. Langs.)

WG21 – C++ Committee

Core WG

Library WG

Evolution WG

Lib Evolution WG

SG1 Concurrency

SG2 Modules

SG3 Filesystem

SG4 Networking

SG5 Tx. Memory

SG6 Numerics

SG7 Reflection

SG8 Concepts

SG9 Ranges

SG10 Feature Test

SG11 Databases

SG12 U. Behavior

SG13 HMI

SG14 Game Dev & Low Latency

(F)DIS Approval

CD & PDTS Approval

Internal Approval

Wording & Consistency

Design & Target (IS/TS)

Domain Specific Investigation & Development

SG1 - Concurrency

- parallelism TS 1 and 2

SG1 - Concurrency

- parallelism TS 1 and 2
- concurrency TS 1 and 2

SG1 - Concurrency

Parallelism TS

- parallel versions of standard algorithms (for_each)

SG1 - Concurrency
Parallelism TS

- parallel versions of standard algorithms (for_each)
- execution policies (seq, par, par_vec)

SG1 - Concurrency

Parallelism TS

- parallel versions of standard algorithms (for_each)
- execution policies (seq, par, par_vec)
- task regions (Parallelism TS 2)

SG1 - Concurrency

Parallelism TS - task region

```
void quicksort(int *v, int start, int end) {
   if (start < end) {

         int pivot = partition(v, start, end);


            quicksort(v, start, pivot - 1);



            quicksort(v, pivot + 1, end);


   }
}
```

SG1 - Concurrency

Parallelism TS - task region

```
void quicksort(int *v, int start, int end) {
   if (start < end) {
      task_region([&] (auto &r) {
         int pivot = partition(v, start, end);

         r.run([&] {
            quicksort(v, start, pivot - 1);
         });

         r.run([&] {
            quicksort(v, pivot + 1, end);
         });
      });
   }
}
```

SG1 - Concurrency

Concurrency TS

- std::future improvements (.then(), when_all(), when_any())

SG1 - Concurrency
Concurrency TS

- std::future improvements (.then(), when_all(), when_any())
- latches and barriers

SG1 - Concurrency

Concurrency TS

- std::future improvements (.then(), when_all(), when_any())
- latches and barriers
- atomic shared pointers

SG1 - Concurrency

Concurrency TS

- std::future improvements (.then(), when_all(), when_any())
- latches and barriers
- atomic shared pointers
- resumable functions (await)

SG1 - Concurrency

Concurrency TS - future improvements

```cpp
std::future<std::string> page = get("google.pl");
```

SG1 - Concurrency

Concurrency TS - future improvements

```cpp
std::future<std::string> page = get("google.pl");

std::future<int> someValue =
   page.then([](std::future<std::string> s) {
      int ret = use(s.get());
      return ret;
   });
```

SG1 - Concurrency

Concurrency TS - future improvements

```
std::future<std::string> page = get("google.pl");

std::future<int> someValue =
   page.then([](std::future<std::string> s) {
      int ret = use(s.get());
      return ret;
   });

someValue.then([](std::future<int> value) {
  // ... use value
});
```

SG1 - Concurrency

Concurrency TS - future improvements

```
auto page = get("google.pl");

auto someValue = page.then([](auto s) {
   int ret = use(s.get());
   return ret;
});

someValue.then([](auto value) {
  // ... use value
});
```

SG1 - Concurrency

Concurrency TS - resumable functions

```
                    void tcp_reader(int total) {
  char buf[64 * 1024];
  auto conn =      Tcp::Connect("127.0.0.1", 1337);
  do {
    auto bytesRead =      conn.read(buf, sizeof(buf));
    total -= bytesRead;
  } while (total > 0);
}

int main() { tcp_reader(1000 * 1000 * 1000)      ; }
```

SG1 - Concurrency

Concurrency TS - resumable functions

```cpp
std::future<void>      tcp_reader(int total) {
  char buf[64 * 1024];
  auto conn = await Tcp::Connect("127.0.0.1", 1337);
  do {
    auto bytesRead = await conn.read(buf, sizeof(buf));
    total -= bytesRead;
  } while (total > 0);
}

int main() { tcp_reader(1000 * 1000 * 1000).get(); }
```

SG2 - Modules

- inclusion vs. semantic model of access to libraries APIs

DCCLABS

SG2 - Modules

- inclusion vs. semantic model of access to libraries APIs
- separation of definition and usage (ODR violations)

SG2 - Modules

- inclusion vs. semantic model of access to libraries APIs
- separation of definition and usage (ODR violations)
- compilation time (don't pay for what you don't use)

SG2 - Modules

- inclusion vs. semantic model of access to libraries APIs
- separation of definition and usage (ODR violations)
- compilation time (don’t pay for what you don’t use)
- two implementations: Microsoft and Google

SG2 - Modules

- inclusion vs. semantic model of access to libraries APIs
- separation of definition and usage (ODR violations)
- compilation time (don’t pay for what you don’t use)
- two implementations: Microsoft and Google
- the „import” keyword

SG2 - Modules

- inclusion vs. semantic model of access to libraries APIs
- separation of definition and usage (ODR violations)
- compilation time (don’t pay for what you don’t use)
- two implementations: Microsoft and Google
- the „import” keyword
- problem with binary format of module description

SG3 - Filesystem

- based on boost::filesystem

SG3 - Filesystem

- based on boost::filesystem
- directory, hard link, soft link, regular file

SG3 - Filesystem

- based on boost::filesystem
- directory, hard link, soft link, regular file
- path, perms

SG3 - Filesystem

- based on boost::filesystem
- directory, hard link, soft link, regular file
- path, perms
- operations: copying files, creating links, deletion, renaming, etc.

SG4 - Networking

- based on boost::asio

SG4 - Networking

- based on boost::asio
- networking using TCP and UDP, including support for multicast

SG4 - Networking

- based on boost::asio
- networking using TCP and UDP, including support for multicast
- client and server applications

SG4 - Networking

- based on boost::asio
- networking using TCP and UDP, including support for multicast
- client and server applications
- scalability to handle many concurrent connections

SG4 - Networking

- based on boost::asio
- networking using TCP and UDP, including support for multicast
- client and server applications
- scalability to handle many concurrent connections
- protocol independence between IPv4 and IPv6

SG4 - Networking

- based on boost::asio
- networking using TCP and UDP, including support for multicast
- client and server applications
- scalability to handle many concurrent connections
- protocol independence between IPv4 and IPv6
- name resolution (DNS)

SG4 - Networking

- based on boost::asio
- networking using TCP and UDP, including support for multicast
- client and server applications
- scalability to handle many concurrent connections
- protocol independence between IPv4 and IPv6
- name resolution (DNS)
- timers

SG4 - Networking

- based on boost::asio
- networking using TCP and UDP, including support for multicast
- client and server applications
- scalability to handle many concurrent connections
- protocol independence between IPv4 and IPv6
- name resolution (DNS)
- timers
- may use coroutines internally

SG5 - Transactional memory

- „synchronized” and „atomic” blocks

SG5 - Transactional memory

- „synchronized” and „atomic” blocks
- threading policies for atomic blocks

SG5 - Transactional memory

```
int sequence() {
   static int i = 0;
   synchronized { // begin transaction
      printf("before %d\n", i);
      ++i;
      printf("after %d\n", i);
      return i; // commit transaction
   }
}
```

SG5 - Transactional memory

```
// each call to f() retrieves a unique value of i, even when done in parallel
int f() {
  static int i = 0;
  atomic_noexcept { // begin transaction
// printf("before %d\n", i); // error: cannot call a non transaction-safe function
    ++i;
    return i; // commit transaction
  }
}
```

SG6 - Numerics

- early development

SG6 - Numerics

- early development
- overflow detecting

SG6 - Numerics

- early development
- overflow detecting
- rounding control

SG6 - Numerics

- early development
- overflow detecting
- rounding control
- multiword integer operations

SG6 - Numerics

- early development
- overflow detecting
- rounding control
- multiword integer operations
- bounded and unbounded types

SG7 - Reflection

- compile time reflection capabilities

SG7 - Reflection

- compile time reflection capabilities
- generation of common functions (getters, = and == operators)

SG7 - Reflection

- compile time reflection capabilities
- generation of common functions (getters, = and == operators)
- type transformations (mocking, delegates, Struct-of-Arrays vector)

SG7 - Reflection

- compile time reflection capabilities
- generation of common functions (getters, = and == operators)
- type transformations (mocking, delegates, Struct-of-Arrays vector)
- compile-time context information (better static-assert)

SG7 - Reflection

- compile time reflection capabilities
- generation of common functions (getters, = and == operators)
- type transformations (mocking, delegates, Struct-of-Arrays vector)
- compile-time context information (better static-assert)
- ORM

SG8 - Concepts

- associate syntactic constraints with template type parameters

SG8 - Concepts

- associate syntactic constraints with template type parameters
- improve code readability

SG8 - Concepts

- associate syntactic constraints with template type parameters
- improve code readability
- improve compiler diagnostics

SG8 - Concepts

- associate syntactic constraints with template type parameters
- improve code readability
- improve compiler diagnostics
- new syntax for defining concepts

SG8 - Concepts

- associate syntactic constraints with template type parameters
- improve code readability
- improve compiler diagnostics
- new syntax for defining concepts
- „auto” - empty constraint

SG8 - Concepts

Simple constraints and overloading

```
void f(EqualityComparable&&); // (1) declaration of a constrained function

```

SG8 - Concepts

Simple constraints and overloading

```
void f(EqualityComparable&&); // (1) declaration of a constrained function
f(MyNoncomparableType()); // ill-formed, simple compiler diagnostic

```

SG8 - Concepts

Simple constraints and overloading

```
    void f(EqualityComparable&&); // (1) declaration of a constrained function
    f(MyNoncomparableType()); // ill-formed, simple compiler diagnostic
    void f(Incrementable&&); // (2) declaration of a constrained function

```

SG8 - Concepts

Simple constraints and overloading

```cpp
void f(EqualityComparable&&); // (1) declaration of a constrained function
f(MyNoncomparableType()); // ill-formed, simple compiler diagnostic
void f(Incrementable&&); // (2) declaration of a constrained function
f("string-literal"s); /* calls (1) - no ambiguity */
```

SG8 - Concepts

Placeholders

```
   std::pair<auto, auto> p2 = std::make_pair(0, ’a’);
```

SG8 - Concepts

Placeholders

```
std::pair<auto, auto> p2 = std::make_pair(0, ’a’);
auto x = f(2);
```

SG8 - Concepts

Placeholders

```
    std::pair<auto, auto> p2 = std::make_pair(0, 'a');
    auto x = f(2);
    Sortable x = f(y);
```

SG8 - Concepts

Placeholders

```cpp
std::pair<auto, auto> p2 = std::make_pair(0, 'a');
auto x = f(2);
Sortable x = f(y);
auto f(Container) -> Sortable;
```

SG8 - Concepts
Placeholders

```
    std::pair<auto, auto> p2 = std::make_pair(0, ’a’);
    auto x = f(2);
    Sortable x = f(y);
    auto f(Container) -> Sortable;
    void f(std::pair<auto, EqualityComparable>);
```

SG8 - Concepts

Short notation

```
    void g1(const EqualityComparable*, Incrementable&);
```

SG8 - Concepts

Short notation

```cpp
void g1(const EqualityComparable*, Incrementable&);

template<EqualityComparable T, Incrementable U>
void g1(const T*, U&);
```

SG8 - Concepts

Short notation

```
    void g1(const EqualityComparable*, Incrementable&);

    template<EqualityComparable T, Incrementable U>
    void g1(const T*, U&);

    template<typename T, typename U>
    void g1(const T*, U&) requires EqualityComparable<T>
        && Incrementable<U>;
```

SG8 - Concepts

Concept definition

```
// variable concept from the standard library
template <class T, class U>
concept bool Derived = std::is_base_of<U, T>::value;

// function concept from the standard library
template <class T>
concept bool EqualityComparable() {
   return requires(T a, T b) { {a == b} -> Boolean;
        {a != b} -> Boolean; };
}
```

SG8 - Concepts

Concept definition

```
template <class T, class U> concept bool Same =
    std::is_same<T,U>::value;

template <class B> concept bool Boolean =
requires(B b1, B b2) {
   { bool(b1) }; // direct initialization constraint
   { !b1 } nothrow -> bool; // compound constraint
   requires Same<decltype(b1 && b2), bool>;
   requires Same<decltype(b1 || b2), bool>;
};
```

SG9 - Ranges

- represents range of elements

SG9 - Ranges

- represents range of elements
- single object replacement for pairs of iterators in STL (convenience, simplify interfaces)

SG9 - Ranges

- represents range of elements
- single object replacement for pairs of iterators in STL (convenience, simplify interfaces)
- range is a concept (or set of concepts)

SG9 - Ranges

- represents range of elements
- single object replacement for pairs of iterators in STL (convenience, simplify interfaces)
- range is a concept (or set of concepts)
- composability (views - lazy adaptation, actions - eager mutation)

SG9 - Ranges

- represents range of elements
- single object replacement for pairs of iterators in STL (convenience, simplify interfaces)
- range is a concept (or set of concepts)
- composability (views - lazy adaptation, actions - eager mutation)
- views can be potentially infinite (view::ints(1))

SG9 - Ranges

```
std::vector<int> v{/*...*/};
std::sort(v.begin(), v.end());
std::sort(v);
```

SG9 - Ranges

```cpp
std::vector<int> v{/*...*/};
std::sort(v.begin(), v.end());
std::sort(v);

auto rng = v
   | view::remove_if([](int i){return i % 2 == 0;});
```

SG9 - Ranges

```
std::vector<int> v{/*...*/};
std::sort(v.begin(), v.end());
std::sort(v);

auto rng = v
    | view::remove_if([](int i){return i % 2 == 0;})
    | view::transform([](int i){return ""s + i;});
```

SG9 - Ranges

```
std::vector<int> v{/*...*/};
std::sort(v.begin(), v.end());
std::sort(v);

auto rng = v
    | view::remove_if([](int i){return i % 2 == 0;})
    | view::transform([](int i){return ""s + i;})
    | view::take(10);
```

SG9 - Ranges

```cpp
std::vector<int> v{/*...*/};
std::sort(v.begin(), v.end());
std::sort(v);

auto rng = v
   | view::remove_if([](int i){return i % 2 == 0;})
   | view::transform([](int i){return ""s + i;})
   | view::take(10);

v |= action::sort | action::unique;
```

SG10 - Feature test

Investigation into whether and how to standardize a way for portable code to check whether a particular C++ product implements a feature yet, as standard is going to be extended.

SG11 - Databases

- compile time SQL expressions

SG11 - Databases

- compile time SQL expressions
- database vendor provided implementation of standard DB interface

SG12 - Unspecified and undefined behavior

A systematic review to catalog cases of undefined and unspecified behavior in the standard and recommend a coherent set of changes to define and/or specify the behavior.

SG13 - Human/machine interface

- generic API for drawing library

SG13 - Human/machine interface

- generic API for drawing library
- early development

SG14 - Game development and low latency

- find interfaces in STL that do not play well with low latency applications

SG14 - Game development and low latency

- find interfaces in STL that do not play well with low latency applications
- exceptions and RTTI, virtual functions

SG14 - Game development and low latency

- find interfaces in STL that do not play well with low latency applications
- exceptions and RTTI, virtual functions
- allocations (std::function)

SG14 - Game development and low latency

- find interfaces in STL that do not play well with low latency applications
- exceptions and RTTI, virtual functions
- allocations (std::function)
- std::async - will spawn new thread or not?

SG14 - Game development and low latency

- find interfaces in STL that do not play well with low latency applications
- exceptions and RTTI, virtual functions
- allocations (std::function)
- std::async - will spawn new thread or not?
- no worst case complexity for some std::algorithms

Thank you

Questions?