¿Qué podría ser C++17?

using std::cpp 2015

J. Daniel Garcia
josedaniel.garcia@uc3m.es

Grupo ARCOS
Universidad Carlos III de Madrid

18 de noviembre de 2015

Aviso

1 Estado actual

2 Cambios menores

3 Conceptos

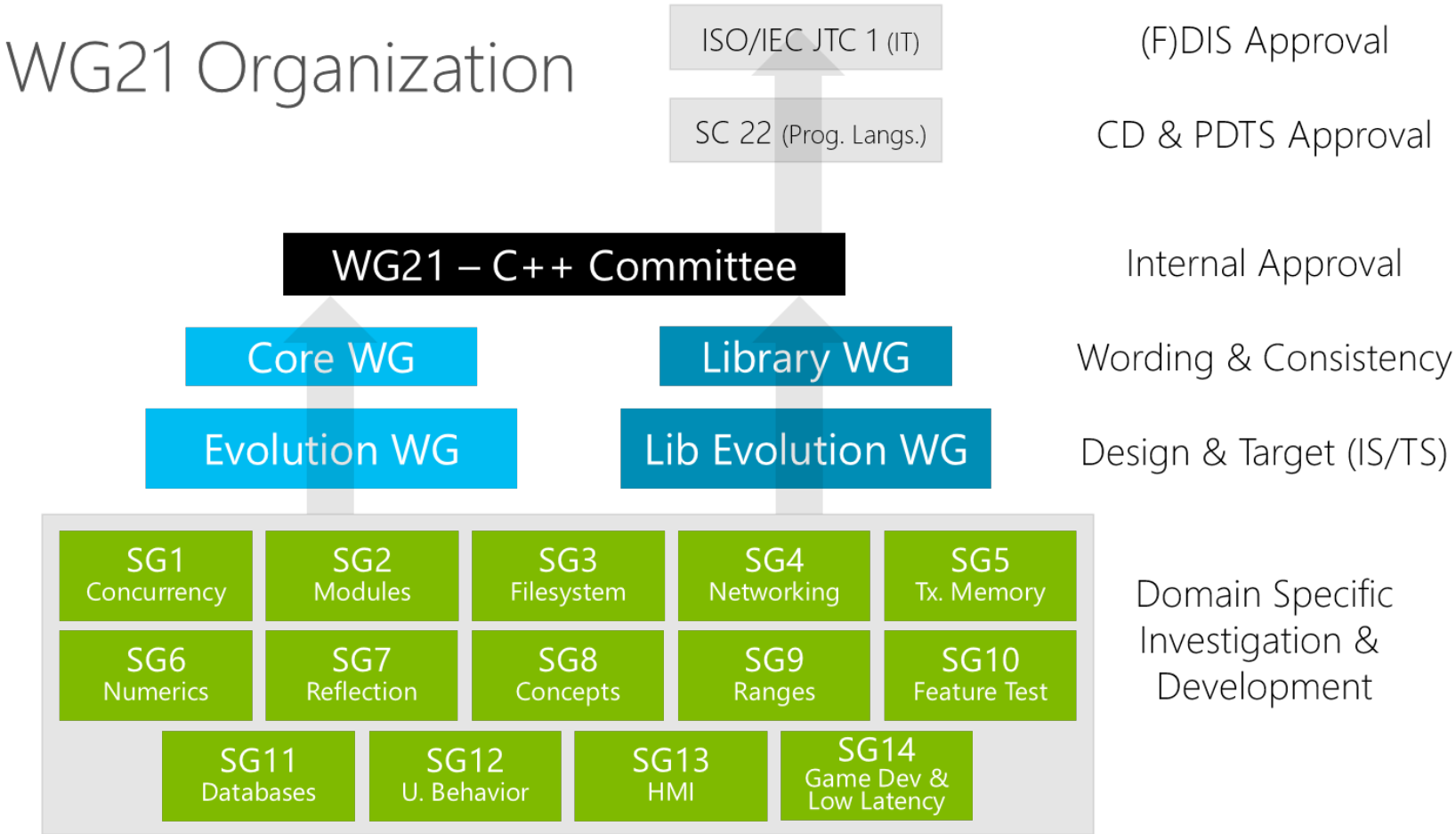
4 Paralelismo y concurrencia

5 Programación basada en contratos

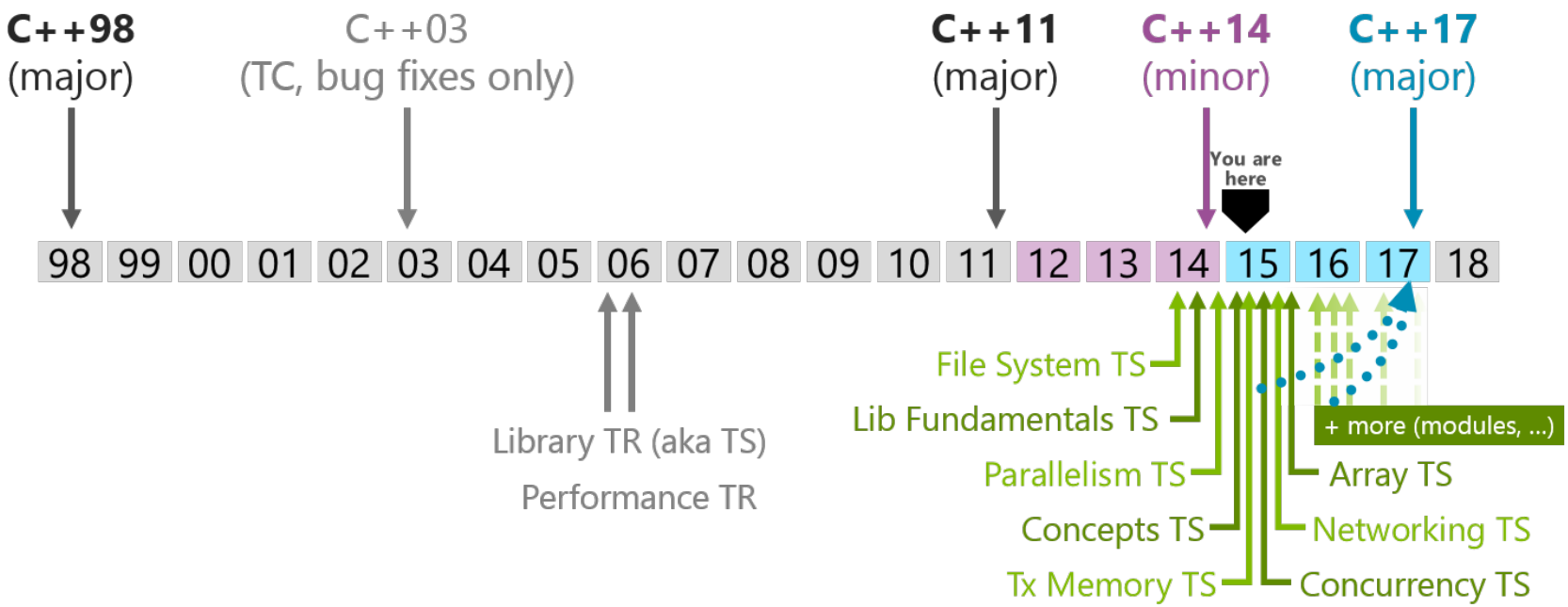
6 Corrutinas

7 Módulos

Estructura del comité



Planes


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
Array TS
Networking TS
Concurrency TS
+ more (modules, ...)

¿Cómo será C++ después de 2017

¿Cómo será C++ después de 2017

- Difícil de saber.

¿Cómo será C++ después de 2017

- Difícil de saber.
  - Mucho trabajo en marcha.
  - Requiere alcanzar consensos.

¿Cómo será C++ después de 2017

- Difícil de saber.
  - Mucho trabajo en marcha.
  - Requiere alcanzar consensos.

- ¿Será C++ una *major release*?
  - No está claro si se mira únicamente a la norma 14882.
  - Pero se debe mirar también a las TS que la acompañan.

Estado

- Norma ISO/IEC 14882
  - Previsible Draft en junio de 2016.

Estado

- Norma ISO/IEC 14882
  - Previsible Draft en junio de 2016.

- File System: ISO/IEC TS 18822:2015
  - Manipulación de archivos y directorios.
  - Basado en *Boost Filesystem V3*.

Library fundamentals

- Library Fundamentals: ISO/IEC TS 19568:2015.
  - Primera versión completada (espacio **std::experimental**).
    - Clases **optional** y **any**.
    - Vista sobre cadena **string_view**.
    - *Allocators* polimórficos (**std::experimental::pmr**).
    - Soporte para *arrays* en **shared_ptr** y **weak_ptr**.
    - Nuevos algoritmos **sample()** y de búsqueda.
    - Función **apply()**.
  - Se está trabajando en una segunda versión.

Concurrencia y paralelismo

- Paralelismo: ISO/IEC TS 19570:2015.
  - Solución de biblioteca para el paralelismo.
  - Ofrece muchos algoritmos de la STL de forma paralela y varias políticas de ejecución
  - Se está trabajando en una segunda versión.
    - Paralelismo de tareas con modelo *fork-join*.
    - Múltiples políticas de planificación (*parent-stealing*, *child-stealing*, . . .).

Concurrencia y paralelismo

- Paralelismo: ISO/IEC TS 19570:2015.
  - Solución de biblioteca para el paralelismo.
  - Ofrece muchos algoritmos de la STL de forma paralela y varias políticas de ejecución
  - Se está trabajando en una segunda versión.
    - Paralelismo de tareas con modelo *fork-join*.
    - Múltiples políticas de planificación (*parent-stealing*, *child-stealing*, . . .).
- Concurrencia: ISO/IEC TS 19571:XXXX.
  - Extensiones de concurrencia para C++.
    - Mejora a **std::future** y APIs asociadas.
    - *latches* y barreras reusables.
    - Punteros inteligentes de tipo atómico (**atomic_shared_ptr** y **atomic_weak_ptr**).
  - Publicación inminente (2015 o 2016).

Memoria transaccional

- Memoria transaccional: ISO/IEC TS 19841:2015.
  - Ofrece aproximación distinta para evitar condiciones de carrera.
  - Evita el uso de cerrojos y otros mecanismos de sincronización.

Redes

- Library Fundamentals: ISO/IEC TS 19216:XXXX.
  - Primera versión en marcha(espacio **std::experimental::net**).
    - Entrada/salida asíncrona.
    - Temporizadores.
    - Búfers y flujos orientados a búfer.
    - Sockets y direcciones de red.
  - En proceso (2016/2017?)

Conceptos y Rangos

- C++ extensions for concepts: ISO/IEC TS 19217:2015.
  - Permite restringir los tipos con los que se instancia una plantilla.

Conceptos y Rangos

- C++ extensions for concepts: ISO/IEC TS 19217:2015.
  - Permite restringir los tipos con los que se instancia una plantilla.

- Especificación Técnica para Rangos: ISO/IEC TS YYYYY:XXXX.
  - Añade el concepto de rango para operar sobre un conjunto de datos.
  - Hace uso de la especificación de conceptos en su definición.

1 Estado actual

2 Cambios menores

3 Conceptos

4 Paralelismo y concurrencia

5 Programación basada en contratos

6 Corrutinas

7 Módulos

Eliminando cosas antiguas

- **auto_ptr**
- **random_shuffle**
- Partes obsoletas de **<functional>**.
- **register**.
- **operator++(bool)**.
- Alias de **iostream**s obsoletos (ej: **ios_base::io_mode**, ...).

Otras mejoras en la biblioteca

- Concepto de iterador contiguo.
- Mejoras en conversiones **unique_ptr<T[]>**.
- Mejoras en las especificaciones **noexcept** de la biblioteca estándar.
- Mejoras en la inserción en mapas (**try_emplace()**).
- Nuevo tipo **void_t**.
- Función global **size()** (**size(x)** $\rightarrow$ **x.size()**).
- Constructores implícitos de **tuple** y **pair**.
- Nuevo tipo **bool_constant<val>**.
- Nuevo mutex **shared_mutex** (problema de los múltiples lectores).

Otras mejoras en la biblioteca

- *Traits* de valor (ej: **is_scalar_v<T>**, **is_arithmetic_v<T>**, . . .) tomados de TS emphlibrary fundamentals.
- Funciones sobre tiempos de **chrono** (**floor**, **ceil**, **round**, **abs**).
- Función **as_const(x)**.
- **lock_guard<>** ahora acepta múltiples **mutex** (*variadic*).

Otros cambios

- Las especificaciones de excepciones pasan a ser parte del sistema de tipos.
- Nueva directiva **__has_include()**.

1 Estado actual

2 Cambios menores

3 Conceptos

4 Paralelismo y concurrencia

5 Programación basada en contratos

6 Corrutinas

7 Módulos

Conceptos

- **Objetivo**: Hacer al programación genérica más simple.

Conceptos

- **Objetivo**: Hacer al programación genérica más simple.

- Un largo camino.
  - Un nuevo comienzo después de la retirada de C++11.
  - Una de las características más revisadas que conozco.

¿Por qué?

- Especificación de restricciones sobre argumentos de plantilla en la declaración.
- Sobrecarga de funciones genéricas usando restricciones.
- Especialización parcial de plantillas de clases y de variables usando restricciones.
- Nueva sintaxis para especificar conceptos y aplicar restricciones a plantillas.
- Sintaxis simplificada para plantillas.
- Mejora sustancial de mensajes de error.

¿Qué podría pasar?

- Es probable que el próxima reunión (marzo 2016) se discuta su incorporación a ISO/IEC 14882.
- Puede ser un debate no exento de polémicas.
  - Posiciones divergentes dentro del comité.
  - Necesidad de consenso.
- En mi opinión, se han incorporado al lenguaje características con menor nivel de *perfeccionamiento*.

¿Qué podría pasar?

- Es probable que el próxima reunión (marzo 2016) se discuta su incorporación a ISO/IEC 14882.
- Puede ser un debate no exento de polémicas.
  - Posiciones divergentes dentro del comité.
  - Necesidad de consenso.
- En mi opinión, se han incorporado al lenguaje características con menor nivel de *perfeccionamiento*.
- Para saber más:
  - 17:00 – 17:45: Concepts Lite.

1 Estado actual

2 Cambios menores

3 Conceptos

4 Paralelismo y concurrencia

5 Programación basada en contratos

6 Corrutinas

7 Módulos

Paralelismo

- ISO/IEC TS 19570:2015 ya aprobada.
  - Solución de biblioteca para algoritmos paralelos.
  - Múltiples políticas de ejecución (secuencial, paralela, paralela y vectorial).
  - Aproximación a factor común de varias soluciones industriales.

```
vector<int> v = get_the_vector();
count_if(par, begin(v), end(v), [](auto x) { return x>0; });

auto m = reduce(par, begin(v), end(v), v[0],
   [] (auto x, auto y) {
     return max(x,y);
   }
);
```

¿Qué podría pasar?

- Se ha propuesto la incorporación a la ISO/IEC 14882.
  - Pendiente de discusión en profundidad.

¿Qué podría pasar?

- Se ha propuesto la incorporación a la ISO/IEC 14882.
  - Pendiente de discusión en profundidad.
- Argumentos a favor:
  - Existen implementaciones de terceras partes.
  - Experiencia avalada por implementaciones industriales previas.
  - Simplificación de una clase de programas paralelos.
  - Fácil transición de programas que usan STL.

¿Qué podría pasar?

- Se ha propuesto la incorporación a la ISO/IEC 14882.
  - Pendiente de discusión en profundidad.
- Argumentos a favor:
  - Existen implementaciones de terceras partes.
  - Experiencia avalada por implementaciones industriales previas.
  - Simplificación de una clase de programas paralelos.
  - Fácil transición de programas que usan STL.
- Argumentos en contra:
  - Insuficiente experiencia de implementación.
  - Falta de integración en distribucions principales.

¿Qué podría pasar?

- Se ha propuesto la incorporación a la ISO/IEC 14882.
  - Pendiente de discusión en profundidad.
- Argumentos a favor:
  - Existen implementaciones de terceras partes.
  - Experiencia avalada por implementaciones industriales previas.
  - Simplificación de una clase de programas paralelos.
  - Fácil transición de programas que usan STL.
- Argumentos en contra:
  - Insuficiente experiencia de implementación.
  - Falta de integración en distribucions principales.
- Para saber más:
  - 12:45 – 13:30: Paralelismo en C++

1 Estado actual

2 Cambios menores

3 Conceptos

4 Paralelismo y concurrencia

5 Programación basada en contratos

6 Corrutinas

7 Módulos

Contratos

- Idea: Soporte a preccondiciones y postcondiciones.
  - Algo más evolucionado que **assert**.

Contratos

- Idea: Soporte a preccondiciones y postcondiciones.
  - Algo más evolucionado que **assert**.
- Ventajas potenciales:
  - Expresar la semántica de las operaciones.
  - Dar soporte al análisis estático de código.
  - Separar la gestión de errores de la detección de defectos.
  - ...

Contratos

- Idea: Soporte a preccondiciones y postcondiciones.
  - Algo más evolucionado que **assert**.
- Ventajas potenciales:
  - Expresar la semántica de las operaciones.
  - Dar soporte al análisis estático de código.
  - Separar la gestión de errores de la detección de defectos.
  - ...

```
template <typename T>
class bounded_queue {
//  ...
  void push(T x)
    [[ expects: ! full () ]]
    [[ ensures: !empty() ]];
//  ...
};
```

¿Qué podría pasar?

- Grupo de trabajo muy activo en los últimos meses.
  - Bloomberg, Morgan Stanley, Microsoft, Universidad Carlos III, . . .
  - Reuniones muy frecuentes.
  - Controversia en algunos puntos.

¿Qué podría pasar?

- Grupo de trabajo muy activo en los últimos meses.
  - Bloomberg, Morgan Stanley, Microsoft, Universidad Carlos III, . . .
  - Reuniones muy frecuentes.
  - Controversia en algunos puntos.
- Posibilidades abiertas:
  - Incorporación a C++17.
  - Especificación técnica.
  - Status quo.

¿Qué podría pasar?

- Grupo de trabajo muy activo en los últimos meses.
  - Bloomberg, Morgan Stanley, Microsoft, Universidad Carlos III, . . .
  - Reuniones muy frecuentes.
  - Controversia en algunos puntos.
- Posibilidades abiertas:
  - Incorporación a C++17.
  - Especificación técnica.
  - Status quo.
- Para saber más:
  - 15:45 – 16:30: ¿Programación basada en contratos para C++17?

1 Estado actual

2 Cambios menores

3 Conceptos

4 Paralelismo y concurrencia

5 Programación basada en contratos

6 Corrutinas

7 Módulos

Corrutinas

- Origen:
  - Idea orginal: Melvin Conway (1958).
  - Generalización del concepto de subrutina (Knuth, 1968).
- Objetivos:
  - Escalable: Millones de corrutinas.
  - Eficiente: Coste equivalente a subrutina.
  - Usable en entornos donde no se pueden usar excepciones.
- Idea básica:
  - Un subprograma que se puede suspender y al que se puede volver.
- Nuevas palabras clave: **co_await**, **co_yield** y **co_return**.

Generando una serie

Generador

```
generator<int> fib(int max) {
  int a = 0;
  int b = 1;
  while (a <= max) {
    co_yield a; // Suspensión
    int t = a + b;
    a = b;
    b = t;
  }
}
```

Uso

```
void f() {
  for co_await (auto && x : fib(1000)) {
    cout << x << endl;
  }
}
```

¿Qué podría pasar?

- Propuesto para C++17.
  - Propuesta reciente de especificación técnica.
  - Interacción con propuestas de entrada/salida asíncrona.

- Existe propuesta formal redactada.

1 Estado actual

2 Cambios menores

3 Conceptos

4 Paralelismo y concurrencia

5 Programación basada en contratos

6 Corrutinas

7 Módulos

¿Por qué?

- Directiva **#include** basada en sustitución de texto.
  - ¿Cuántas líneas tengo que compilar con mi *"Hello World"*?.
  - Problemas de compilación y enlazado (*one definition rule*, guardas de inclusión, símbolos múltiplemente definidos, ...).
  - Tiempos de compilación muy largos.
  - No existe el concepto de **componente**.
  - Problemas con el preprocesador (tenemos que vivir con marcos).

¿Por qué?

- Directiva **#include** basada en sustitución de texto.
  - ¿Cuántas líneas tengo que compilar con mi *"Hello World"*?.
  - Problemas de compilación y enlazado (*one definition rule*, guardas de inclusión, símbolos múltiplemente definidos, ...).
  - Tiempos de compilación muy largos.
  - No existe el concepto de **componente**.
  - Problemas con el preprocesador (tenemos que vivir con marcos).

- Desafios:
  - Organización de código en proyectos de gran escala.
  - Mejor interacción con herramientas.
  - Reducción del tiempo de construcción.

Ejemplo

Usando módulos

```
import std.io;
import graphics.image;

int  main() {
   using namespace graphics2d;
   image img;
   ifstream  ifs {"foto.jpg"};
   img.load(ifs);
   img.to_gray_scale();
   ofstream ofs{"fotogs.jpg"}
   img.store(ofs);
   return 0;
}
```

La interfaz

```
import std.io;
import graphics.image;

module graphics.image;

namespace graphics2d {
   export class image {
      //  ...
   };
};
```

¿Qué podría pasar?

- Eventual inclusión en C++17.

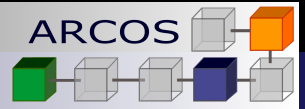
¿Qué podría pasar?

- Eventual inclusión en C++17.

- Aspectos a considerar:
  - Experinecia de implementación: Microsoft y clang.
  - Divergencias menores entre ambas aproximaciones.
  - Beneficios importantes.

¿Qué podría pasar?

- Eventual inclusión en C++17.
- Aspectos a considerar:
  - Experinecia de implementación: Microsoft y clang.
  - Divergencias menores entre ambas aproximaciones.
  - Beneficios importantes.
- Implementación de Microsoft disponible en Visula Studio 2015.



¿Qué podría ser C++17?

using std::cpp 2015

J. Daniel Garcia
josedaniel.garcia@uc3m.es

Grupo ARCOS
Universidad Carlos III de Madrid

18 de noviembre de 2015