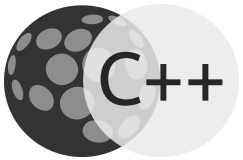
C++ y mucho más

Un vistazo al universo de bibliotecas disponibles

Martín Knoblauch Revuelta
http://www.mkrevuelta.com @mkrevuelta mkrevuelta@gmail.com

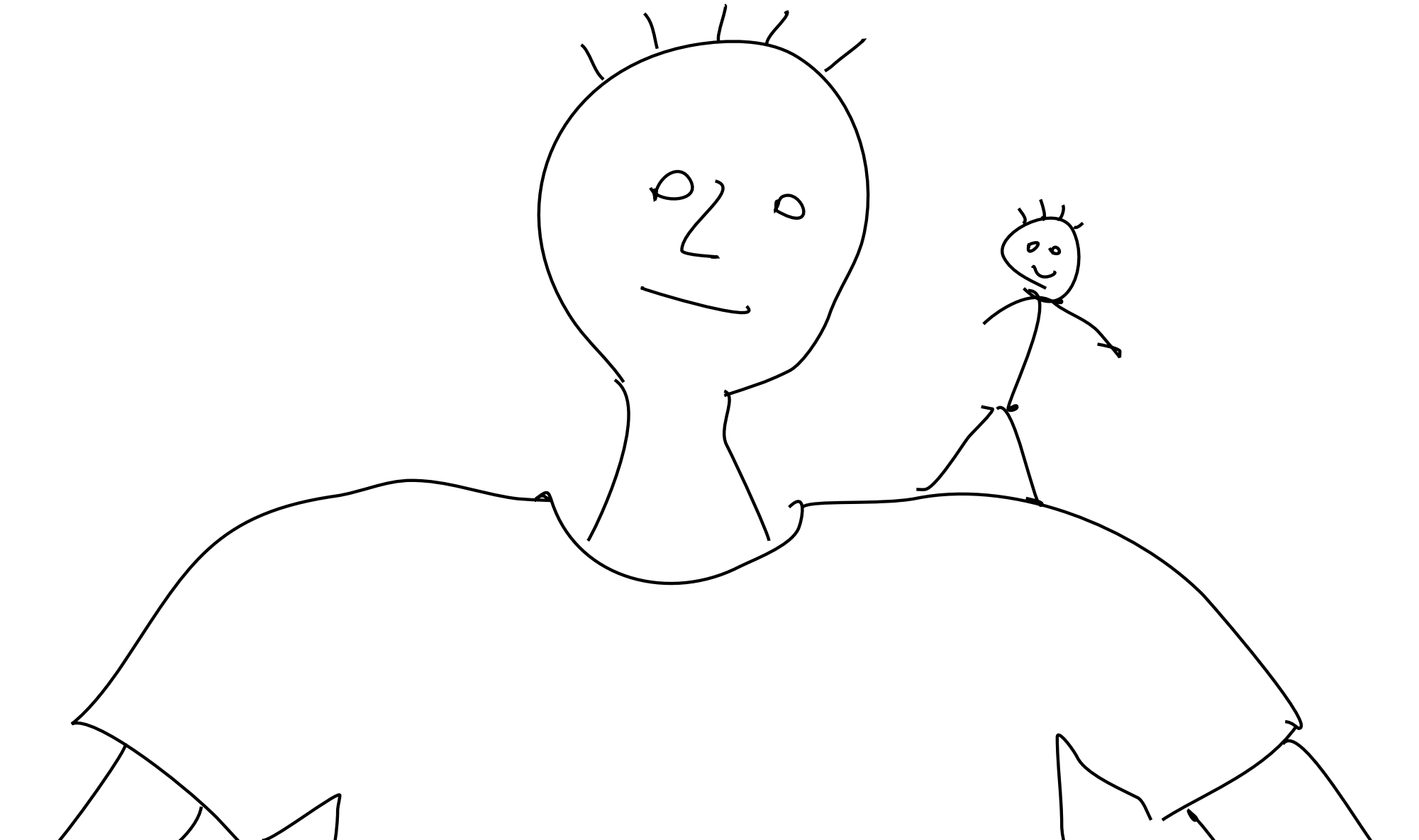
indizen  using std::cpp

Universidad Carlos III de Madrid,24 de noviembre de 2016

Index

Introducción

A hombros de gigantes



Viejos gigantes

Casi mayores de edad

- gSOAP (Genivia)
- Xerces (Apache)

¿Por qué los seguimos usando?

¡Porque funcionan!

Valgrind

Valgrind

Más que una biblioteca de funciones, es una herramienta para el desarrollador

Búsqueda de errores http://valgrind.org

- Memcheck:
  - Lagunas de memoria
  - Liberación incorrecta
  - Desbordamiento de los bloques reservados
  - Uso peligroso de datos sin inicializar

Valgrind

Además de Memcheck:

- Cachegrind: cache profiler
  Callgrind: grafo de llamadas
- Massif: heap profiler
- Helgrind: threads, bloqueos y exclusión mutua

Soportado en:

Linux, MacOSX, Solaris, Android...

pero no Windows

Conan

Conan.io

Gestor de paquetes C/C++ https://conan.io

- Dependencias
- Descentralizado
- Fuentes / binarios
- Scripts en Python
- *Free hosting service for free software*

Conan.io

Distintas...

- Versiones de paquete
- Compiladores
- Arquitecturas
- Sistemas operativos
- Generadores de proyectos

"Hola Boost" con Conan (1/8)

Vamos a hacer un "hola mundo" que use Boost

1. Buscamos "boost" en conan.io
2. Primera opción (más descargas):
   `Boost/1.60.0@lasote/stable`
3. ¿Seguimos las instrucciones? ¡No!

Leer:

http://docs.conan.io/en/latest/examples/boost.html
http://docs.conan.io/en/latest/manage_deps/conanfile_txt.html

"Hola Boost" con Conan (2/8)

Preparamos los fuentes:

```
$ mkdir holaBoost
$ cd holaBoost
$ mkdir src
$ cd src
$ vi hola.cpp
$ vi CMakeLists.txt
$ vi conanfile.txt
$ cd ..
```

"Hola Boost" con Conan (3/8)

src/hola.cpp

```
#include <iostream>
#include <boost/filesystem/operations.hpp>

int main ()
{
   std::cout << "¡Hola! " << std::endl
   << boost::filesystem::current_path ()
   << std::endl;
}
```

"Hola Boost" con Conan (4/8)

src/CMakeLists.txt

```
project(holaBoost)
cmake_minimum_required(VERSION 2.8.12)

include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
conan_basic_setup()

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall")
add_executable(hola hola.cpp)
target_link_libraries(hola ${CONAN_LIBS})
```

"Hola Boost" con Conan (5/8)

src/conanfile.txt

```
[requires]
Boost/1.60.0@lasote/stable

[generators]
cmake

[imports]
bin, *.dll -> ./bin
lib, *.dylib* -> ./bin
```

"Hola Boost" con Conan (6/8)

Instalamos: (La 1ª vez descarga Boost)

```
$ mkdir debug
$ cd debug
$ export CC=/usr/bin/gcc-4.8
$ export CXX=/usr/bin/g++-4.8
$ conan install ../src                    \
        -s compiler=gcc                   \
        -s compiler.version=4.8
[..]
```

"Hola Boost" con Conan (7/8)

Dejamos que CMake despliegue su magia...

```
$ cmake ../src                                  \
        -G "Unix Makefiles"                     \
        -DCMAKE_BUILD_TYPE=Debug
[..]
```

"Hola Boost" con Conan ($8/_8$)

¡Ya podemos compilar!

```
$ cmake --build .
[..]
$ bin/hola
¡Hola!
"/home/martin/experimentos/holaBoost/debug"
```

Google

Protocol Buffers

Serialización de datos estructurados para RPC

- Datos en binario
- Mensaje sin definición de su estructura
- La estructura se define en lenguaje "proto"
- Generador de código: `protobuf-compiler`
- El código generado empaqueta / desempaqueta y ofrece "getters" y "setters"
- C++, C#, Go, Java, Python y más

https://developers.google.com/protocol-buffers

Flat Buffers

Similar a Protocol Buffers pero...

- No necesita desempaquetar el mensaje
  - Ahorro de memoria
  - Menor fragmentación
  - Acceso directo a datos específicos
- Menos código

https://google.github.io/flatbuffers

¿Queda alguna razón para usar Protocol Buffers?

Zero Message Queue

Zero Message Queue

Mensajería asíncrona de alto rendimiento

- API algo similar a Sockets
- Patrones de comunicación
  - Solicitud - respuesta
  - Publicación - suscriptores
  - Push - pull (pipeline)
  - Par exclusivo

http://zeromq.org/

Zero Message Queue

¡No sólo para comunicar distintas máquinas!

Proponen usarlo:

- Entre procesos
- Entre hilos del mismo proceso
- Como mecanismo de sincronización
- Para obtener soluciones realmente escalables

Garantías

Garantías sobre entrega de los mensajes

- No hay garantía de entrega
- No hay ACK
- El mensaje se entrega de una pieza...

  o no se entrega

Los ACKs y reenvíos tienen un coste

ZMQ lo deja en manos de la aplicación

Intel Thread Building Blocks

Thread Building Blocks

Biblioteca de paralelismo para aprovechar los procesadores multi-núcleo

https://www.threadingbuildingblocks.org

https://software.intel.com/en-us/tbb-documentation

Operaciones básicas: for

`parallel_for`

- Cuerpo del bucle: objeto-función / lambda
- Granularidad orientativa (por defecto 1 iteración)
- Espacio de iteración
  1D / 2D / definido por el usuario

Distribución del trabajo

Troceado del trabajo

- `simple_partitioner` → Según granul.
- `auto_partitioner` → Heurísticas
- `affinity_partitioner` → ¡Caché!
- `static_partitioner` → Sin robo

Operaciones básicas: reduce

`parallel_reduce`

- Objeto-función con estado (acumulador)
- Operador() no const
- "`split` constructor"
  - Copia la parte constante
  - Pone a cero el acumulador
- `join`
  - `acum += otro.acum;`

Espacios de iteración a medida

Espacios de iteración definidos por el usuario

- También a base de objetos-función
- "`split` constructor":
  dos partes iguales
- "`proportional split` constructor":
  dos partes distintas según proporción
  indicada por el particionador

Otros bucles...

`parallel_do`

- Listas enlazadas

  Al menos un hilo recorre la lista secuencialmente
- Árboles

  Cada hilo puede añadir trabajos de los nodos descendientes

Hay mucho más

"[Data] Flow Graphs"

Contenedores: hash map

`concurrent_hash_map`

- Bloqueo de grano fino
- A nivel de elemento
- Un escritor / varios lectores
- RAII
  - `accessor`
  - `const_accessor`

Contenedores: vector ($1/_2$)

`concurrent_vector`

- Memoria no necesariamente consecutiva
- Los objetos no se mueven
- Bloqueo de grano fino
  - `push_back (x)`
  - `grow_by (n)`
  - `grow_to_at_least (n)`
- ¡Varios hilos pueden añadir a la vez!

Contenedores: vector ($2/_2$)

Si varios hilos añaden a la vez:

1. Esperar a que `v.size > i`
2. Esperar a que `v[i]` esté construido
   Truco:
   - `zero_allocator`
   - Elementos `atomic != 0`

Contenedores: colas

`concurrent_queue`

- Crece dinámicamente
- Sin bloqueos
- `push`, `try_pop`

`concurrent_bounded_queue`

- Se puede especificar la capacidad
- Añade operaciones bloqueantes

Exclusión mutua

Varias clases de mutex, según:

- Simple / escritor+lectores
- Bloqueos recursivos / simples
- Reparto equitativo / desigual

nvidia CUDA

nvidia CUDA

No sólo un lenguaje o una API, sino:

- Plataforma de computación + modelo de programación
- Hacen elegante el uso de una GPU para propósito general
- C, C++, Fortran

http://www.nvidia.com/object/cuda_home_new.html

nvidia CUDA

Hola mundo: "SAXPY" (Sum A×X Plus Y)

- Preparar memoria en CPU y GPU
- Inicializar datos en CPU
- Transferir a GPU
- Ejecutar uno o más "kernels" (pequeñas funciones ¿estilo SIMD?) en GPU
- Transferir resultado a CPU

https://devblogs.nvidia.com/parallelforall/easy-introduction-cuda-c-and-c

nvidia CUDA

Thrust

- Biblioteca de algoritmos paralelos
- Interfaz similar a C++ STL
- Vectores
  - `host_vector`
  - `device_vector`
- Algoritmos
- Iteradores

https://thrust.github.io

Boost

Para muestra, un botón

```
#include <iostream>
#include <boost/lexical_cast.hpp>

int main (int argc, char * argv[])
{
  for (int i=1; i<argc; ++i)
    std::cout <<
      boost::lexical_cast<unsigned>(argv[i]);
}
```

Compatible hacia atrás

Alternativas std a lexical_cast

- `sscanf`: char*, muy aparatoso
- `istringstream`: muy aparatoso
- `strtoul`: char*
- `stoul`: C++11
- `from_chars`: C++17, char*, no lanza

Ojo

¡Todos convierten -1 a unsigned!
(lexical_cast también)

Vuelta a Boost: numeric_cast (1/2)

```
#include <iostream>
#include <boost/lexical_cast.hpp>
#include <boost/numeric/conversion/cast.hpp>

template<class Final, class Interm>
Final miLexCast (const std::string & s)
{
  return
    boost::numeric_cast<Final> (
      boost::lexical_cast<Interm> (s) );
}
```

Vuelta a Boost: numeric_cast (2/2)

```
int main (int argc, char * argv[])
{
  for (int i=1; i<argc; ++i)
    std::cout <<
      miLexCast<unsigned,
                long long>(argv[i]);
}
```

¿Es una chapuza? Quizás, pero...

$-1 \rightarrow$ boost::numeric::negative_overflow

Boost Asio

Entrada/salida síncrona/asíncrona

- `io_service`: bucle de eventos multihilo
- `deadline_timer` → callback(s)
- `strand`: ejecución secuencial
- `tcp/udp::socket`, IPv4/IPv6
- Fin de operación E/S → callback
- `socket::cancel()`, `socket::close()`
- Nada pte. → `io_service::run()` retorna
- Más: Bind, Coroutine... ver ejemplos

Boost Filesystem

Rutas de directorios y archivos (Windows, Unix...)

- `current_path`
- `concat` / `operator+=`
- `is_regular_file`, `is_directory`
- `canonical`
- ...

Habrá un `std::filesystem` en C++17

Boost Program Options

Clases para manejar cómodamente los argumentos en línea de órdenes de un programa

- `options_description`: nombre, tipo de valor y descripción de cada argumento posible
- `variables_map`: valores de las opciones (rellenado a partir de `argc`, `argv` y `options_description`)
- Se pueden enganchar funciones de validación o combinar opciones de otras fuentes

Boost Container

Entre muchas más cosas: `flat_map` y `flat_set`

- Variantes de `std::map` y `std::set`
- Memoria contigua en vez de árboles
- Misma interfaz, distinto rendimiento
  - Inserción y extracción rápidas en orden, muy lentas en el caso general
  - Menos memoria, menos fragmentada
  - Iteración y búsqueda más rápidas (factor cte.)

Boost Flyweight

Caché para reutilizar objetos (const) medianos o grandes en vez de hacer copias

- `flyweight<T>` es pequeño y apunta al `T` correspondiente
- Convertible implícitmente a `const T&`
- Un contador por cada objeto `T`
- Cuando la cuenta llega a cero, se destruye
- Configurable: clave-valor, árbol/hash/otro, sin contadores, sin bloqueos...

Boost Graph

Tres formas de codificar un grafo:

1. `adjacency_list`:
   una lista de aristas por cada vértice
2. `adjacency_matrix`:
   matriz 2D, 1 bit por posible arista
3. `edge_list`: lista de aristas

Recordatorio para no matemáticos

Vértice: punto, circulito, nodo...

Arista: línea, flecha... ¿puntero?

Boost Interprocess

- Memoria compartida
- Proyección de archivos en memoria
- Semáforos, mutex, variables de condición... *
- * Variantes con nombre
- Bloqueo de archivos
- Punteros relativos
- Colas de mensajes

Más Boost

http://www.boost.org/doc/libs/

Accumulators, Algorithm, Align, Any, Array, **Asio**, Assert, Assign, Atomic, Bimap, Bind, Call Traits, Chrono, Circular Buffer, Compatibility, Compressed Pair, Compute, Concept Check, Config, **Container**, Context, Conversion, Convert, Core, Coroutine (deprecated), Coroutine2, CRC, Date Time, DLL, Dynamic Bitset, Enable If, Endian, Exception, Fiber, **Filesystem**, **Flyweight**, Foreach, Format, Function, Function Types, Functional, Functional/Factory, Functional/Forward, Functional/Hash, Functional/Overloaded Function, Fusion, Geometry, GIL, **Graph**, GraphParallel, Hana, Heap, ICL, Identity Type, In Place Factory, Typed In Place Factory, Integer, **Interprocess**, Interval, Intrusive, IO State Savers, Iostreams, Iterator, Lambda, **Lexical Cast**, Local Function, Locale, Lockfree, Log, Math, Math Common Factor, Math Octonion, Math Quaternion, Math/Special Functions, Math/Statistical Distributions, Member Function, Meta State Machine, Metaparse, Min-Max, Move, MPI, MPL, Multi-Array, Multi-Index, Multiprecision, **Numeric Conversion**, Odeint, Operators, Optional, Parameter, Phoenix, Pointer Container, Polygon, Pool, Predef, Preprocessor, **Program Options**, Property Map, Property Tree, Proto, Python, QVM, Random, Range, Ratio, Rational, Ref, Regex, Result Of, Scope Exit, Serialization, Signals (deprecated), Signals2, Smart Ptr, Sort, Spirit, Statechart, Static Assert, String Algo, Swap, System, Test, Thread, ThrowException, Timer, Tokenizer, TR1 (deprecated), Tribool, TTI, Tuple, Type Erasure, Type Index, Type Traits, Typeof, uBLAS, Units, Unordered, Utility, Uuid, Value Initialized, Variant, VMD, Wave, Xpressive

C++ Guidelines Support Library

No confundir con...

GSL - GNU Scientific Library

Biblioteca numérica para C/C++ (es decir, en C)

- Números aleatorios
- Ajuste de mínimos cuadrados
- Vectores, matrices, complejos
- Interpolación
- Montecarlo
- Ecuaciones diferenciales ...

https://www.gnu.org/software/gsl

C++ Core Guidelines - GSL

Core Guidelines

- Consejos para programar bien
- Muy, muy, **muy** recomendables

  http://isocpp.github.io/CppCoreGuidelines

Aquí sólo vamos a ver una parte de las GSL, que sólo son una parte de las Core Guidelines

C++ Core Guidelines - GSL

Guidelines Support Library

- Apoyo para C++ Core Guidelines
- Sólo cabeceras
- Implementación de Microsoft (también funciona con GCC 5.1)

  https://github.com/Microsoft/GSL
- Otra para compiladores antiguos: gsl-lite

  https://github.com/martinmoene/gsl-lite

GSL - Ejemplo (1/12)

```
#include <iostream>
#include <cstring>
#include <gsl/gsl>

// Envoltorio de una biblioteca antigua
namespace wr
{
 gsl::owner<char*> reservar(std::size_t sz);
 void liberar (gsl::owner<char*> p);
}
```

GSL - Ejemplo (2/12)

```
// Función para convertir a mayúsculas
template <class SpanType>
void mayus (SpanType span);
```

Veremos esta función más adelante

GSL - Ejemplo (3/12)

```
int main ()
{
  // gsl::final_act<...>
  auto adios { gsl::finally ( []
  {
    std::cout << "Bye bye" << std::endl;
  }) };
```

La función lambda se ejecutará cuando el objeto `adios` salga de ámbito

GSL - Ejemplo (4/12)

```
// Cadena original
gsl::not_null<gsl::czstring<>> saludo
   { "Hello gsl world!" };

// Reservar espacio para una copia
gsl::owner<gsl::zstring<>> copia
   { wr::reservar (1+strlen(saludo)) };

auto liberador { gsl::finally (
   [&]{ wr::liberar (copia); } ) };
```

GSL - Ejemplo (5/12)

Operamos con `saludo` y `copia` como lo haríamos con simples punteros a `char`

```
// Hacer la copia
strcpy (copia, saludo);

// Mostrarla: Hello gsl world!
std::cout << copia << std::endl;
```

De hecho, son sólo punteros con anotaciones

GSL - Ejemplo (6/12)

`span`: trozo de array (lectura/escritura)

```
// Modificar parte de la cadena
gsl::span<char> trozo { copia + 6, 3 };
mayus (trozo);

// Mostrarla: Hello GSL world!
std::cout << copia << std::endl;
```

Es más seguro que pasar dos punteros, o un puntero y un tamaño

GSL - Ejemplo (7/12)

```
// Con unique_ptr en vez de
// owner + finally
std::unique_ptr
    <char, decltype(&wr::liberar)>
    otraCopia
    {
      wr::reservar (1+strlen(saludo)),
      &wr::liberar
    };
```

GSL - Ejemplo (8/12)

```
// Más correcto:
std::string ultimaCopia { saludo };

mayus (
   gsl::string_span<>{ultimaCopia}
   .subspan(6, 3) );

// Resultado: Hello GSL world!
std::cout << ultimaCopia << std::endl;
```

GSL - Ejemplo (9/12)

```
    // Fin de main():
    // Liberar memoria y decir "Bye bye"
}
```

Por fin se destruyen los objetos `ultimaCopia`, `otraCopia`, `liberador` y `adios`

La destrucción del resto (incluído `trozo`) es trivial

GSL - Ejemplo (10/12)

```
// Envoltorio de una biblioteca
// antigua ficticia

namespace wr {

gsl::owner<char*> reservar (std::size_t sz)
{
   return new char[sz];
}
```

Era de esperar, ¿no?

GSL - Ejemplo (11/12)

Sin sorpresas...

```
void liberar (gsl::owner<char*> p)
{
  delete [] p;
}

} // namespace wr
```

GSL - Ejemplo (12/12)

```
// Función para convertir a mayúsculas
template <class SpanType>
void mayus (SpanType span)
{
  for (auto & c: span)
    c = toupper (c);
}
```

Ojo: ¡Codificación de caracteres!

Otras bibliotecas

Otras bibliotecas (1/2)

Mundos completos

- Poco `https://pocoproject.org`

  Alternativa a Boost

- Qt `https://www.qt.io`

  Interfaces gráficas multiplataforma

Otras bibliotecas ($2/_2$)

"Creative coding"

- Cinder `https://libcinder.org`

  Windows / OS X

- Open Frameworks `http://openframeworks.cc`

  Muchas plataformas

- Cairo `https://www.cairographics.org`

  Gráficos vectoriales 2D

Muchas gracias

¿Preguntas?