C++

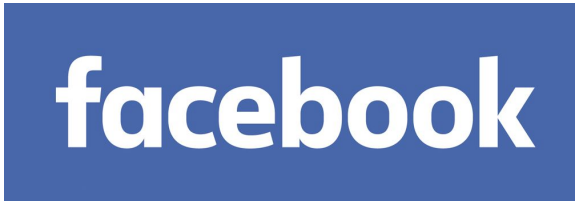
https://alexti.me/cpp/

¿Por qué C++?

- ➔ Estándar
- ➔ Rápido
- ➔ Influyente
- ➔ Poderoso
- ➔ Biblioteca Amplia

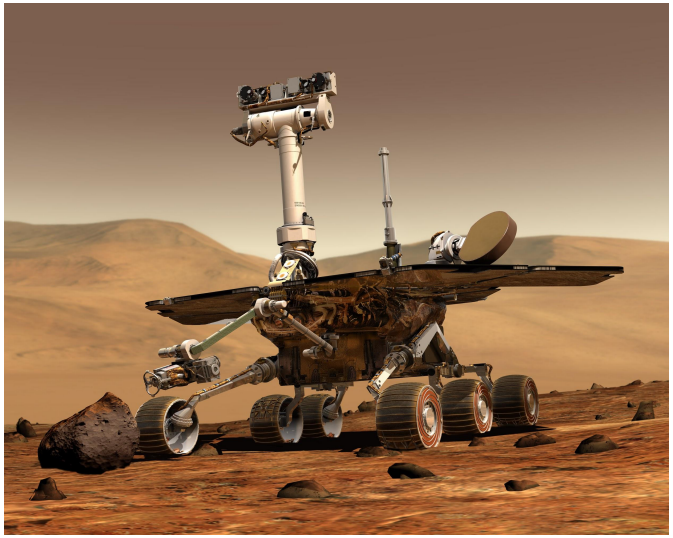
¿Quién usa C++?

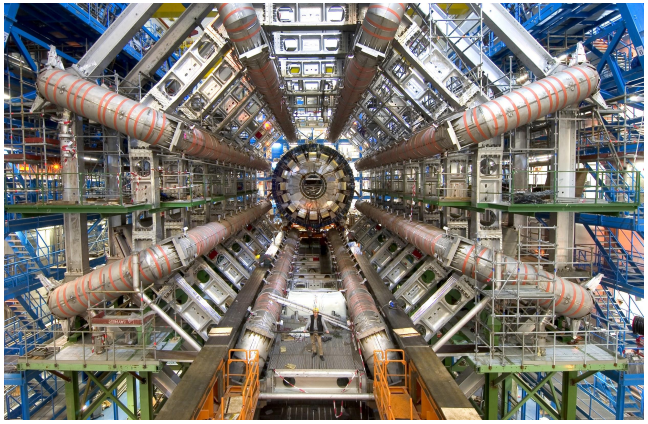
- Microsoft
- Adobe
- Google
- Facebook
- Amazon
- HP



¿Quién usa C++?

- ❏ Microsoft
- ❏ Adobe
- ❏ Google
- ❏ Facebook
- ❏ Amazon
- ❏ HP
- ❏ Lockheed-Martin
- ❏ NASA
- ❏ CERN
- ❏ Y muchos otros





Programación Competitiva

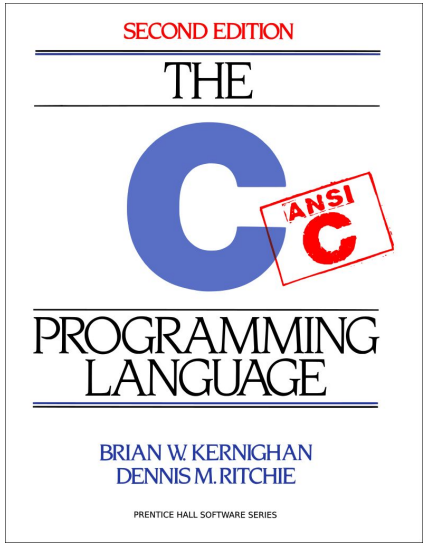
Nuestras soluciones:

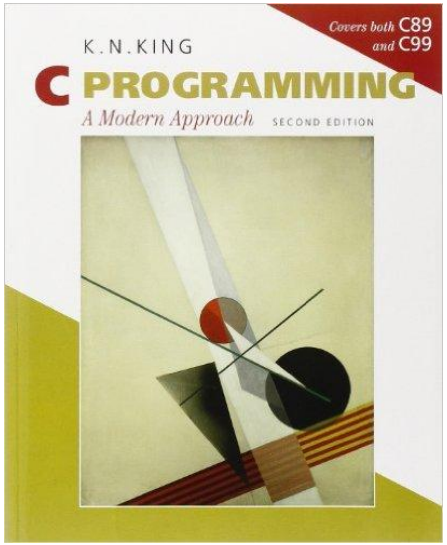
- Deben ser replicables en el sistema del evaluador.
- Deben correr rápido
- Deben ser fáciles de desarrollar

¿Que hay de C?

- C++ es un incremento de C. La librería estándar de C está incluida.
- En programación competitiva se usan algunas funciones de C por performance.
- Ejemplo:
  - printf vs. cout
  - scanf/getchar vs. cin
  - new/delete vs. malloc/free
  - realloc

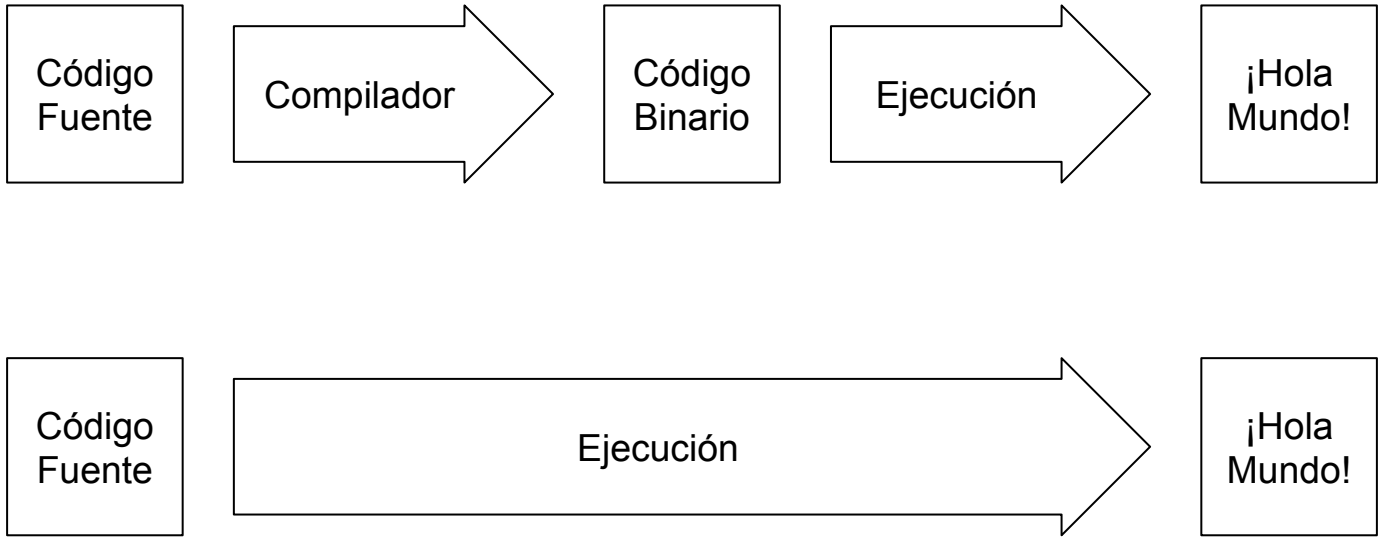
¡C++ no es C!



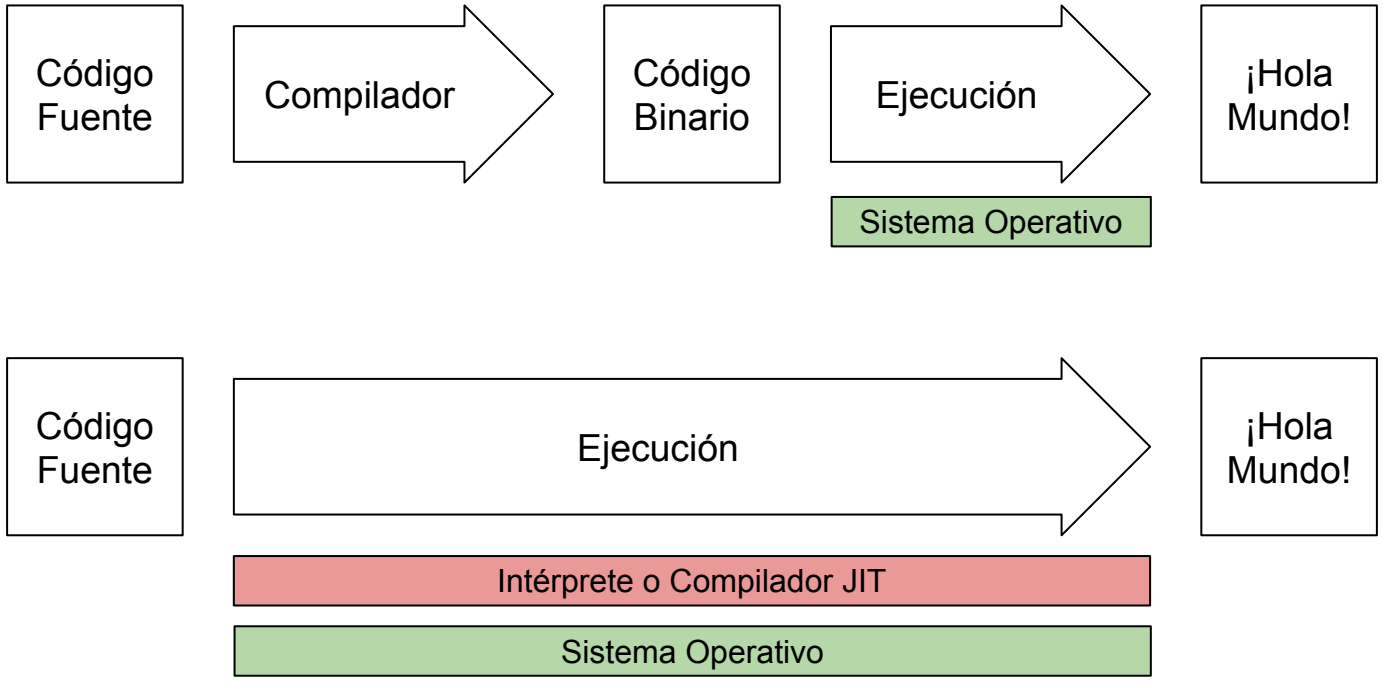


El entorno C++

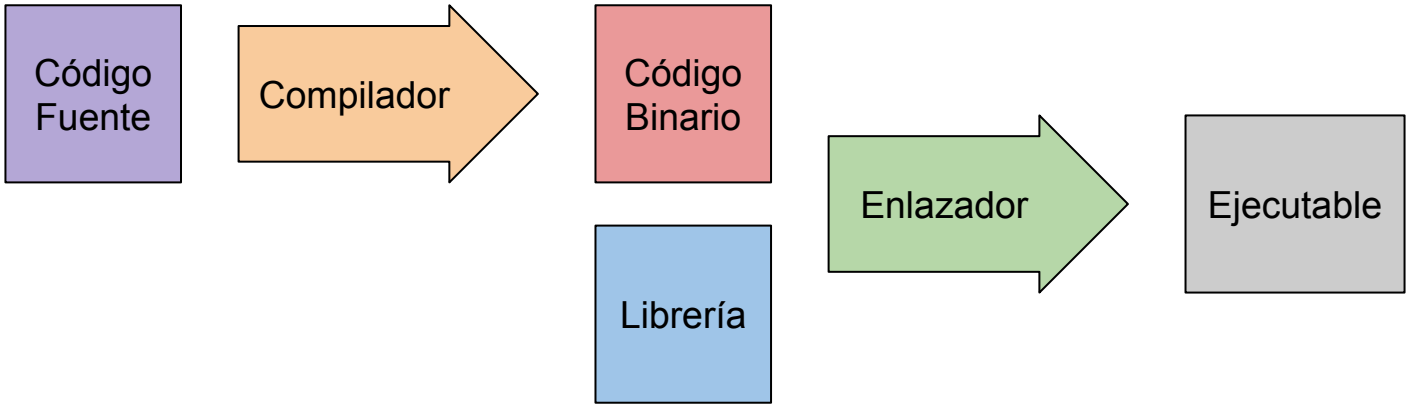
C++ es compilado



C++ es compilado



C++ es compilado



¡Hola Mundo!

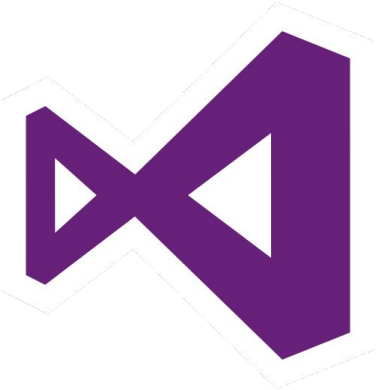
```cpp
#include <iostream>

int main(void) {
    std::cout << "¡Hola Mundo!" << std::endl;
    return 0;
}
```

¡Necesitamos un Compilador!

ORACLE

SOLARIS STUDIO



Linux

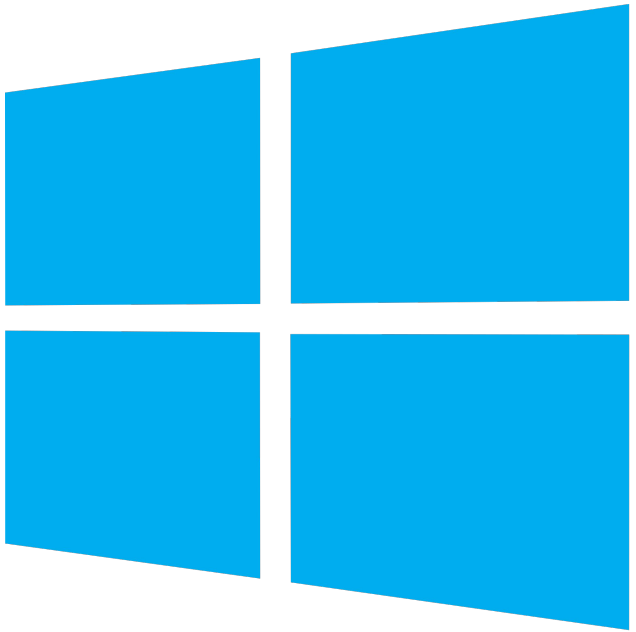
```
$ sudo apt install build-essential make gcc g++
$ g++ -v
```

Windows

https://nuwen.net/mingw.html

Abrir open_distro_window.bat

```
X:\MinGW>g++ --version
```



Editores e IDEs

- Vim
- Emacs
- Geany
- Eclipse
- QtCreator
- Visual Studio
- CLion

¡Hola Mundo!

```cpp
#include <iostream>

int main(void) {
    std::cout << "¡Hola Mundo!" << std::endl;
    return 0;
}

// Guardar como hola.cpp
```

El proceso de compilación

```
$ cpp hola.cpp > expand.cpp                          # Pre-procesar
$ g++ -o hola.o -std=c++14 -O2 -Wall -c expand.cpp   # Compilar
$ g++ -o hola.exe hola.o                             # Enlazar

En un solo paso:
$ g++ -o hola.exe -std=c++14 -O2 -Wall hola.cpp
```

`-std=c++14:` Compilar bajo el estándar de 2014 (Por defecto es 1998)
`-O2:` Nivel medio de optimización (O0 a 03)
`-Wall:` Mostrar todas las warning

Otro ejemplo

```
#include <iostream>

int main(void) {
    std::string s;
    std::cin >> s;
    std::cout << "¡Hola " << s << "!" << std::endl;
    return 0;
}
```

El entorno C++

Estructura de un programa

Variables

- Tienen un **tipo** (`int`, `float`)
- Tienen un **nombre**
- Pueden ser:
  - Locales
  - Globales

```
int x;
float y;
```

Funciones

- Tienen un **tipo** (`int`, `float`, `void`)
- Tienen un **nombre**
- Toda función que no tenga tipo `void` retorna un valor.
- Toda función puede aceptar parámetros.
- Si la función no toma parámetros, usar `void` como parámetro para que el compilador lo tome en consideración.

```
int f() {
    return 0;
}
float g() {
    return 0.0f;
}
void h() {}
```

Nombres

- ➔ Toda función o variable tiene un nombre
- ➔ Caracteres alfanuméricos
- ➔ Se permite _
- ➔ Un nombre no puede iniciar con un número
- ➔ No iniciar con _, el compilador hace uso de identificadores iniciando con _

- `int name;` **✓ Es válido**
- `int 9name;` **✗ Inválido**
- `int name_1;` **✓ Es válido**
- `int _name;` **✗ No usar**

Keywords

- No usar keywords como identificadores, da error de compilación.
- Un editor con syntax highlighting resalta keywords.

alignas (since C++11)	dynamic_cast	reinterpret_cast
alignof (since C++11)	else	requires (concepts TS)
and	enum	return
and_eq	explicit	short
asm	export(1)	signed
atomic_cancel (TM TS)	extern(1)	sizeof(1)
atomic_commit (TM TS)	false	static
atomic_noexcept (TM TS)	float	static_assert (since C++11)
auto(1)	for	static_cast
bitand	friend	struct(1)
bitor	goto	switch
bool	if	synchronized (TM TS)
break	import (modules TS)	template
case	inline(1)	this
catch	int	thread_local (since C++11)
char	long	throw
char16_t (since C++11)	module (modules TS)	true
char32_t (since C++11)	mutable(1)	try
class(1)	namespace	typedef
compl	new	typeid
concept (concepts TS)	noexcept (since C++11)	typename
const	not	union
constexpr (since C++11)	not_eq	unsigned
const_cast	nullptr (since C++11)	using(1)
continue	operator	virtual
decltype (since C++11)	or	void
default(1)	or_eq	volatile
delete(1)	private	wchar_t
do	protected	while
double	public	xor
	register(2)	xor_eq

Operaciones con Nombres

- Declaración

```
int x;
double suma(double, double);
```

- Definición

```
double suma(double a, double b) {
    return a + b;
}
```

- Asignación

```
int x = 10;
```

Reglas

- Una variable local no inicializada contiene valores indefinidos.
- **No inicializar variables y recibir respuestas inesperadas en un error común.**
- Las funciones deben ser al menos declaradas antes de su uso.
- La definición de una función puede encontrarse separada de su declaración.

Condicionales e Iteración

```
if (condition) {

} else if (another_condition) {

} else {

}
```

```
switch (variable) {
    case A:
        break;
    case B:
        break;
    default:
        break;
}
```

```
for (initialize; condition; update) {

}
```

```
for (auto &variable : collection) {

}
```

```
while (condition) {

}
```

```
do {

} while (condition);
```

```
goto label;
```

Macros

- Un macro es una directiva del preprocesador.
- Se usan para:
  - **Incluir headers**
  - Compilación condicional
  - Reemplazar constantes

```
#include <iostream>
#define CANTIDAD 10

int main(void) {
    return 0;
}
```

Headers

- Un header contiene la definición de todas las funciones, variables y estructuras de datos públicas del módulo.
- Es una **interface para el módulo**.
- Se usa **header guards** para evitar su doble inclusión.

```
#ifndef MY_MODULE_H
#define MY_MODULE_H

float add(float, float);
float subtract(float, float);
float multiply(float, float);
float divide(float, float);

#endif
```

using

- using permite incluir miembros de otro namespace.
- Pueden incluirse miembros individuales o todos.
- Todos:

```
using namespace std;
```

- Individuales:

```
using std::cin;
```

Ejemplo

Una calculadora que reciba 3 argumentos y devuelva la operación pedida.

Input:

```
4 + 5
```

Output:

```
9
```

```cpp
#include <iostream>

using std::cin;
using std::cout;
using std::endl;

int main(void) {
    int val1, val2, res;
    char oper;
    cin >> val1 >> oper >> val2;
    switch (oper) {
        case '+':
            res = val1 + val2;
            break;
        case '-':
            res = val1 - val2;
            break;
        default:
            return -1;
    }
    cout << res << endl;
    return 0;
}
```

```cpp
// calculador.cpp
#include <iostream>
#include "operaciones.h"

using std::cin;
using std::cout;
using std::endl;

int main(void) {
    int val1, val2, res;
    char oper;
    cin >> val1 >> oper >> val2;
    switch (oper) {
        case '+':
            res = suma(val1, val2);
            break;
        case '-':
            res = resta(val1, val2);
            break;
        default:
            return -1;
    }
    cout << res << endl;
    return 0;
}
```

```cpp
// operaciones.h
#ifndef OPERACIONES_H
#define OPERACIONES_H

int suma(int, int);
int resta(int, int);

#endif
```

```cpp
// operaciones.cpp
#include "operaciones.h"

int suma(int a, int b) {
    return a + b;
}

int resta(int a, int b) {
    return a - b;
}

#endif
```

Ejercicios

1. Extender la calculadora para que soporte multiplicación y división.
2. Un programa que devuelva todos los factores de un número.

Input:

```
10
```

Output:

```
1 2 5 10
```

3. Imprimir los próximos 50 años bisiestos.
4. Reestructurar el programa para que toda la funcionalidad se encuentre en un segundo módulo.

Ejercicios

5. Se tiene el siguiente programa que imprime todo entero desde input hasta -input:

```
#include <iostream>

int main(void) {
    int input, neg_input, counter;
    std::cin >> input;
    neg_input = -input;
    while (counter >= neg_input) {
        std::cout << counter << " ";
        --counter;
    }
    std::cout << std::endl;
    return 0;
}
```

¿Hay algún error? ¿Cuál es?

Estructura de un programa

https://alexti.me/cpp/

Tipos y Valores

Tipos

- C++ está basado en C, un lenguaje para programación de sistemas.
- Los tipos primitivos de C++ tienen equivalencia directa en memoria.
- Las operaciones de C++ tienen equivalencia directa con instrucciones de CPU.

```cpp
#include <iostream>

using std::cin;
using std::cout;
using std::endl;

int main(void) {
    int val1, val2, res;
    char oper;
    cin >> val1 >> oper >> val2;
    switch (oper) {
        case '+':
            res = val1 + val2;
            break;
        case '-':
            res = val1 - val2;
            break;
        case '*':
            res = val1 * val2;
            break;
        case '/':
            res = val1 / val2;
            break;
        default:
            return -1;
    }
    cout << res << endl;
    return 0;
}
```

```
$ ./calc.exe
10 / 0
Floating point exception (core dumped)
```

Tipos

- bool (1 byte)
- char (1 byte)
- short (al menos 2 bytes)
- int (al menos 2 bytes)
- long (al menos 4 bytes)
- float
- double
- void

- short int (al menos 2 bytes)
- long int (al menos 4 bytes)
- long long int (al menos 8 bytes)
- Etc.

También hay tipos de tamaño fijo:

- int8_t
- int16_t
- int_32t

Definidos en <cinttypes>

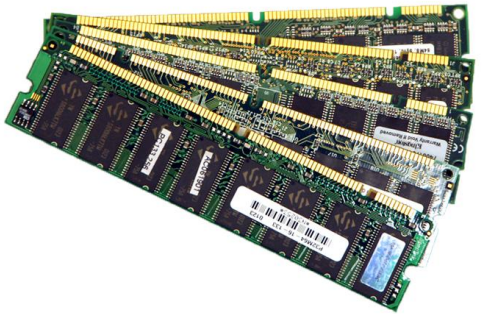
Seguridad de Tipos

- **Sin un tipo, el contenido de una variable son solo bits en memoria.**

`0110 0001`

Como char es 'a'

Como int es 97



- **La seguridad de tipos implica que todo objeto sólo podrá ser usado en las operaciones definidas para su tipo.**

Operadores

```cpp
#include <iostream>

int main(void) {
    std::cout << 1 + 2
              << std::endl;
    return 0;
}
```

```cpp
#include <iostream>

int main(void) {
    std::cout << "1" + "2"
              << std::endl;
    return 0;
}
```

Signed vs. Unsigned

signed int

16

`0000 0000 0001 0000`

-16

`1111 1111 1111 0000`

unsigned int

16

`0000 0000 0001 0000`

-16

`1111 1111 1111 0000`

Es decir: 4 294 967 280

Signed vs. Unsigned

signed int

4 294 967 280

`1111 1111 1111 0000`

Integer Overflow:

Comportamiento Indefinido

unsigned int

4 294 967 280

`1111 1111 1111 0000`

Ejemplo

```cpp
#include <iostream>
#include <cstdint>

int main(void) {
    int16_t x = 0b0111'1111'1111'1111;
    uint16_t y = 0b0111'1111'1111'1111;
    int16_t ox = 0b1111'1111'1111'1111;
    uint16_t oy = 0b1111'1111'1111'1111;

    std::cout << x << "\n"
              << y << "\n"
              << ox << "\n"
              << oy << std::endl;
    return 0;
}
```

Ejemplo de tipos

```
#include <iostream>

int main(void) {
    unsigned short edad;
    unsigned int dni;

    std::cout << "DNI: " << dni
              << ", Edad: " << edad
              << std::endl;
    return 0;
}
```

¡Cuidado!

```
#include <iostream>

int main(void) {
    unsigned short edad = 5;

    while (edad >= 0) {
        --edad;
    }
    return 0;
}

// ¿El programa termina?
```

Unsigned siempre es mayor que 0

No termina

Arrays estilo C

```
int arra[10];
int arrb[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

// Los array estilo C son una región contínua de memoria.
// Decaen a pointers al pasar a una función.
// Ocasionalmente se usan por performance/control.
```

Constantes

```
#include <iostream>
#define TEMPERATURA 35

int main(void) {
    std::cout << "Temp.: "
              << TEMPERATURA
                     + 2
              << std::endl;
    return 0;
}
```

Macros (Estilo C)

```
#include <iostream>

const int temperatura = 35;

int main(void) {
    std::cout << "Temp.: "
              << temperatura
                     + 2
              << std::endl;
    return 0;
}
```

Constantes

```cpp
#include <iostream>

constexpr int temperatura = 35;

int main(void) {
    std::cout << "Temp.: "
              << temperatura
                     + 2
              << std::endl;
    return 0;
}
```

- Los macros son reemplazados antes de la compilación (¡Sin seguridad de tipos!)
- `const` **puede** ser optimizado por el compilador, pero no da ninguna garantía.
- `constexpr` es compilado **siempre que sea posible**.

Ejemplo

```
#include <iostream>

int main(void) {
    const int x = 10;
    x = 4;
    std::cout << x << std::endl;
    return 0;
}

// ¿Compila?
```

Error de compilación

Referencias

- Una referencia es un alias para una variable.
- Una referencia no puede ser nula.
- Debe ser declarada y definida.

```
type &ref_name = variable;
```

Ejemplo:

```
int &ref = x; // Donde x es una variable tipo int
```

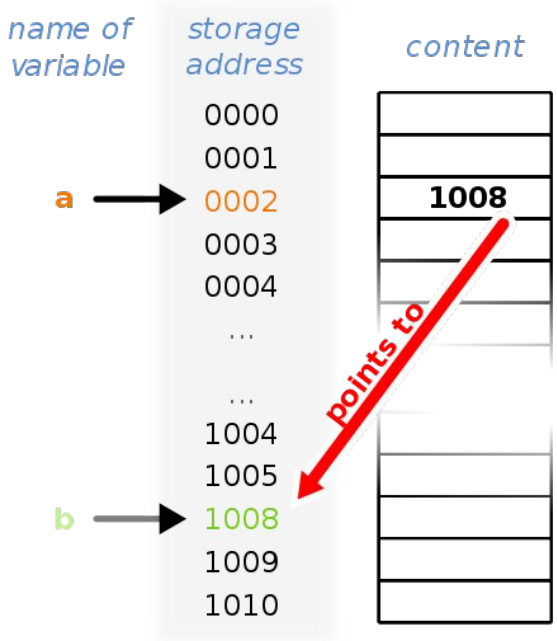
Ejemplo

```
#include <iostream>

int main(void) {
    int x = 10;
    int &ref = x;
    ref += 1;
    std::cout << "x=" << x << ", ref=" << ref<< std::endl;
    return 0;
}
```

Punteros

- Un puntero es una variable que contiene la dirección de otra variable



Punteros

Para declarar punteros en C/C++ se usa el operador de indirección:

```
type *ptr_name;
```

Ejemplo:

```
int *pointer;
```

Punteros

Para obtener la dirección en memoria de una variable se usa el operador unario:

```
ptr = &x;
```

Para acceder al valor de un puntero se usa el operador de indirección:

```
*ptr = y;
```

Ejemplo

```
#include <iostream>

int main(void) {
    int val = 10;
    int *ptr;
    ptr = &val;         // &x obtiene la dirección de x
    *ptr = 5;           // *ptr dereferencia el puntero
    std::cout << "val=" << val
              << ", ptr=" << ptr
              << ", *ptr=" << *ptr << std::endl;
    return 0;
}
```

¡Cuidado!

```
#include <iostream>

int main(void) {
    int val = 10;
    int *ptr = nullptr;
    *ptr = val;       // Intentando dereferenciar un puntero nulo
    std::cout << "val=" << val << ", ptr=" << ptr << std::endl;
    return 0;
}

// ¿Qué sucede?
```

Comportamiento Indefinido

```
$ ./punteros.exe
Segmentation fault (core dumped)
```

Punteros

```
#include <iostream>

int main(void) {
    int arr[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int *ptr = arr;
    std::cout << "arr[0]=" << *ptr;
    ++ptr;
    std::cout << ", arr[1]=" << *ptr
              << ", arr[6]=" << *(ptr+5)
              << std::endl;
    return 0;
}
```

Estructuras de Datos

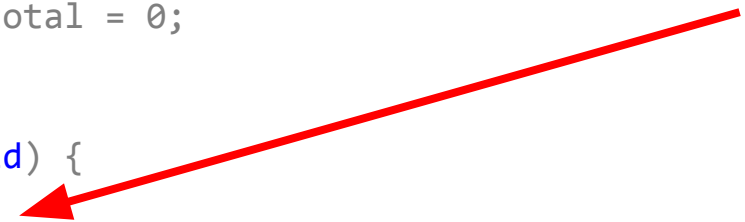
- No son clases
- No tienen métodos
- Todos sus miembros son públicos

```
struct ventas {
    std::string item = "";
    unsigned cantidad = 0;
    double precio = 0;
};
```

```cpp
#include <iostream>

struct venta {
    std::string item = "";
    unsigned cantidad = 0;
    double precio = 0;
    double total = 0;
};

int main(void) {
    venta v;
    std::cout << "Item, cantidad y precio" << std::endl;
    std::cin >> v.item >> v.cantidad >> v.precio;
    v.total = v.cantidad * v.precio;
    std::cout << "Total: $ " << v.total << std::endl;
    return 0;
}
```



RAII!

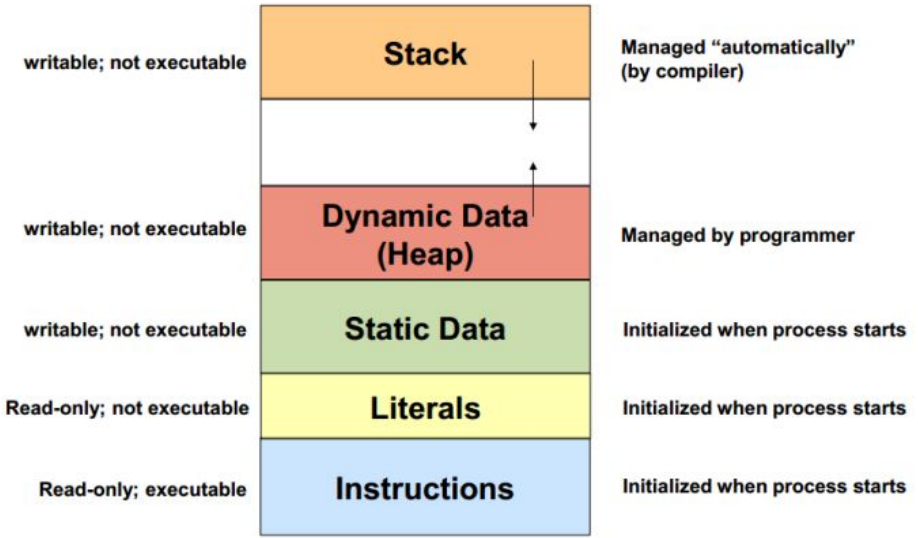
RAII

- ➔ **R**esource **A**cquisition **I**s **I**nitialization
- ➔ ¿Pero cuando usamos new?

Stack vs. Heap

Sin entrar a detalle:

- **Stack** es el espacio local de las funciones.
- **Heap** es el espacio común manejado por el programador.



```cpp
#include <iostream>

struct venta {
    std::string item = "";
    unsigned cantidad = 0;
    double precio = 0;
    double total = 0;
};

venta* registrar_venta(std::string, unsigned, double);
void imprimir_venta(venta &v);

int main(void) {
    std::string item;
    unsigned cantidad;
    double precio;

    std::cout << "Item, cantidad y precio"
              << std::endl;
    std::cin >> item >> cantidad >> precio;
    venta *v = registrar_venta(item, cantidad,
precio);
    imprimir_venta(*v);
    delete v;
    return 0;
}
```

Debemos borrar manualmente lo que alocamos en heap

```cpp
venta* registrar_venta(std::string item,
                       unsigned cantidad,
                       double precio) {
    venta *v = new venta;
    v->item = item;
    v->cantidad = cantidad;
    v->precio = precio;
    v->total = precio * cantidad;
    return v;
}
```

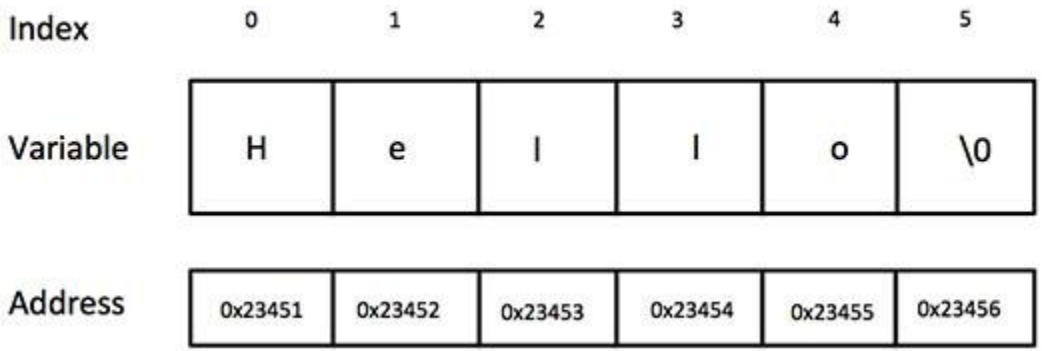
Devuelve un pointer alocado en heap

```cpp
void imprimir_venta(venta &v) {
    std::cout << "item: " << v.item
              << ", cantidad: " << v.cantidad
              << ", precio: " << v.precio
              << ", total: " << v.total
              << std::endl;
}
```

Recibe una referencia

Strings

- Tres tipos:
  - `char* str = "string"; // String estilo C literal`
  - `char str[] = "string"; // String estilo C arar`
  - `std::string str = "string"; // String estilo C++`



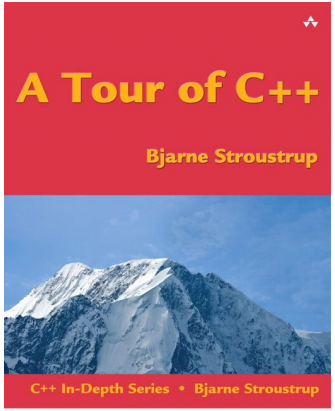
Tipos compuestos

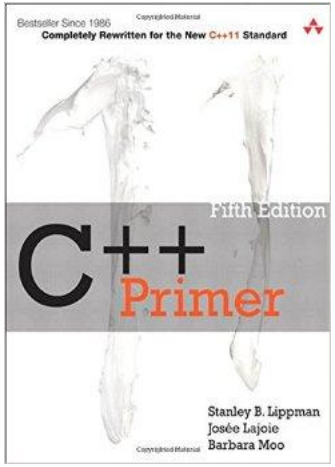
- C++ permite referencias a const, punteros const, punteros const a const, etc.
- C++ permite obtener el tipo de retorno de una función (decltype).
- C++ permite asumir automáticamente el tipo de una variable (auto).
- Etc.

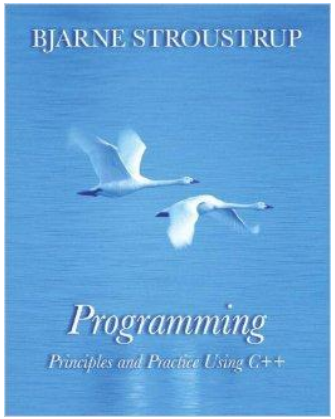

Use the
book,
Luke

Recursos

- http://es.cppreference.com/w/
- http://en.cppreference.com/w/
- http://www.cplusplus.com/







Ejercicios

1. Un programa que convierta una string constante en un número entero. Imprimir en formato hexadecimal (Usar `std::hex`). Usar tipos apropiados para evitar overflow.

Constante:

```
const std::string s = "4294967295";
```

Output:

```
ffffffff
```

Formato:

```
std::cout << std::hex << num
          << std::endl;
```

2. Un programa que dada una operación (venta o compra), una cantidad y un precio devuelva el total de ingreso o egreso. Debe usar estructuras de datos.

Input:

```
venta 5 10.0
```

Output:

```
50.0
```

Input:

```
compra 10 5.0
```

Output:

```
-50.0
```

Ejercicios

3. Implementar una lista enlazada de `int` usando estructuras de datos. Debe incluir las operaciones de inserción, acceso y borrado.
   Nota:
   Se puede declarar funciones miembro de un struct

```
struct my_type {
    int x;
    void f() {}
};
```

Tipos y valores

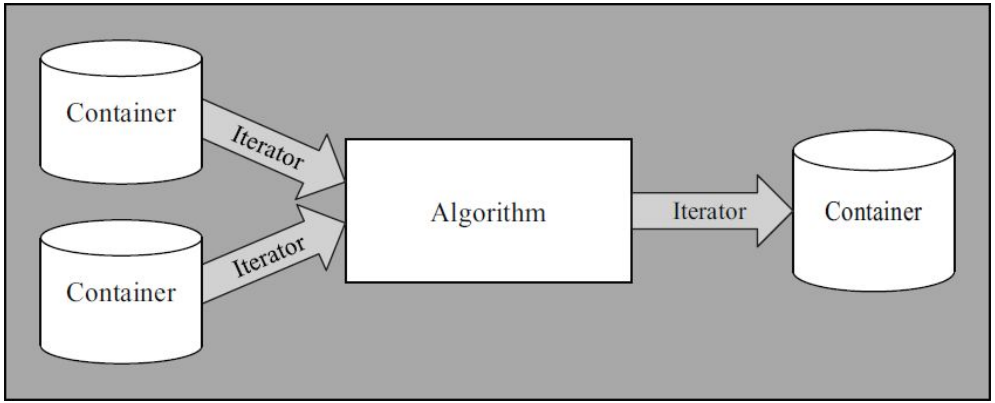
https://alexti.me/cpp/

Standard Template Library

Entrada y Salida

- cin y cout son streams estándar de entrada y salida.
- cin >> x lee todos los datos hasta el próximo espacio en blanco (tab, espacio, \n) y los guarda en la variable x.
- cout << x imprime el valor de x.
- endl hace flush del stream e imprime un salto de línea.

STL



- Tiene 4 partes:
  - Algoritmos
  - Containers
  - Iteradores
  - Funcional
- Usa templates: clases y funciones de tipo genérico.
- Los template de C++ se especializan durante la compilación.
  - Más eficiente que durante la ejecución

Un array dinámico

- Arreglos de C tienen tamaño fijo
- ¿Qué hago si no sé cuántos elementos habrá?

```
std::vector
```

Un array dinámico

```cpp
#include <iostream>
#include <vector>

int main(void) {
    int x, sum = 0;
    std::vector<int> v;
    while (std::cin >> x) {
        v.push_back(x);
    }
    for (auto it = v.begin(); it != v.end(); ++it) {
        sum += *it;
    }
    std::cout << "Total: " << sum << std::endl;
    return 0;
}
```

¿Qué hemos hecho?

- Creamos un **container** vector de int.
- Recorrimos el container con un **iterator**.
- *¿Y qué hay de algorithm?*

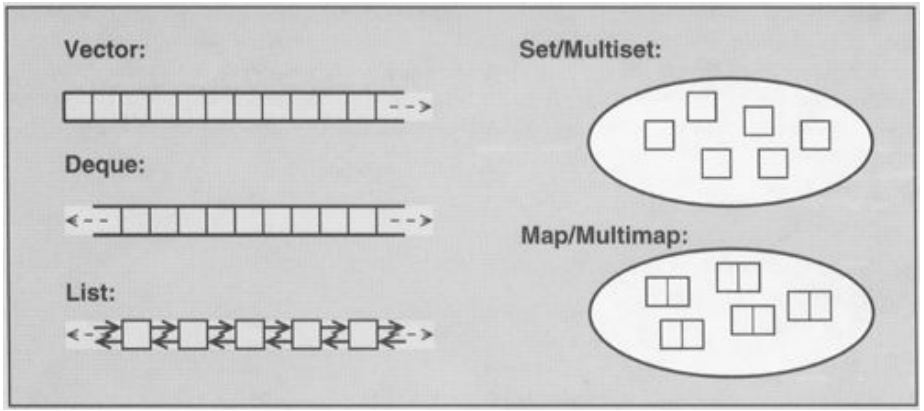
Usando un algoritmo de STL

```
#include <iostream>
#include <vector>
#include <numeric>

int main(void) {
    int x, sum;
    std::vector<int> v;
    while (std::cin >> x) {
        v.push_back(x);
    }
    sum = std::accumulate(v.begin(), v.end(), 0);
    std::cout << "Total: " << sum << std::endl;
    return 0;
}
```

Containers

- Guardan una colección de objetos.
- Implementan estructuras de datos comúnmente usadas:
  - Arreglos dinámicos
  - Pilas
  - Colas
  - Listas enlazadas
  - Árboles
  - Arreglos Asociativos



Iterators

- Apuntan a un elemento dentro de una colección.
- Permiten aritmética de iteradores:
  - `v.begin() + v.size() / 2;` **✓ Es válido**
  - `++iterator` **✓ Es válido y frecuente**
- Pueden ser invalidados por cambios al container.
  - Depende del container.

Algorithms

- Funciones comúnmente usadas.
- Generalmente operan en rangos definidos por **Iterators**.
- Operaciones para:
  - Búsqueda
  - Ordenamiento
  - Contar
  - Manipular

Functional

- Encapsula funciones que son determinadas en tiempo de ejecución.
- Incluye definiciones de funciones típicas:
  - Suma
  - Resta
  - Igual
  - Etc.

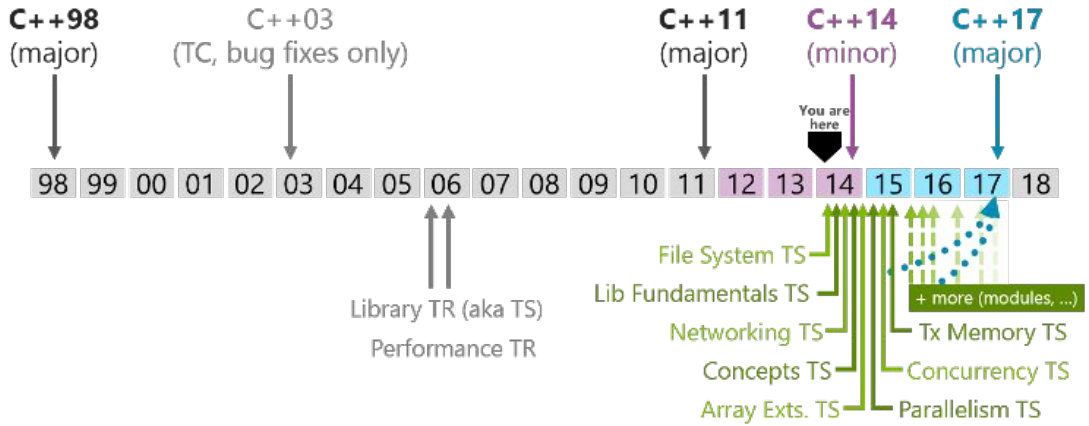
```cpp
#include <iostream>
#include <functional>

using std::cin;
using std::cout;
using std::endl;
using std::function;

int main(void) {
    int val1, val2;
    char oper;
    function<int(int, int)> func;
    cin >> val1 >> oper >> val2;
    switch (oper) {
        case '+':
            func = [](int a, int b) { return a + b; };
            break;
        case '-':
            func = [](int a, int b) { return a - b; };
            break;
        default:
            return -1;
    }
    cout << func(val1, val2) << endl;
    return 0;
}
```

STL

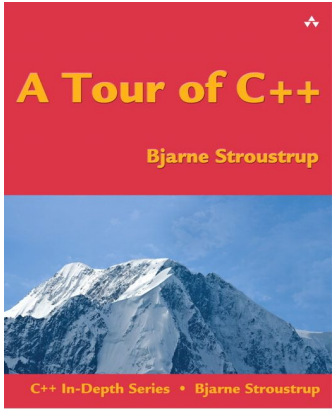
- La biblioteca STL es amplia.
- Nuevos estándares incrementan funcionalidad.
- Próxima revisión: C++17

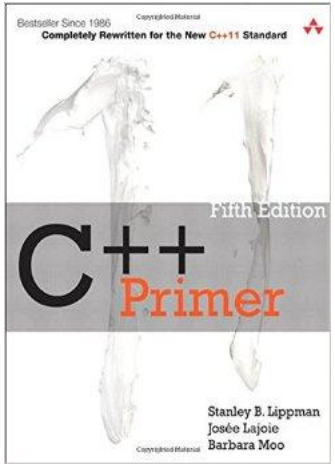


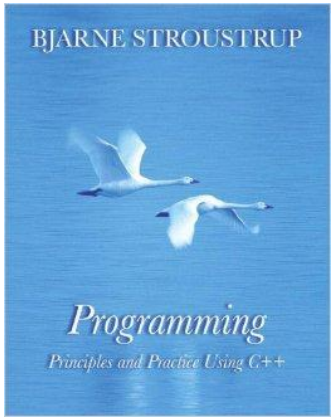

Seriously, use the book, Luke

¡Recursos!

- http://es.cppreference.com/w/
- http://en.cppreference.com/w/
- http://www.cplusplus.com/







Ejercicios

1. Calcular el producto de todos los números ingresados.
2. Una calculadora que acepte suma, resta, multiplicación, división y exponenciación (Solo 3 argumentos).
3. Un programa que lea un archivo con formato:

   ```
   nombre edad nota
   nombre edad nota
   ```

   Calcular la edad promedio, nota promedio y nota máxima.
4. Un programa que imprima todos los números primos desde el 2 hasta 100000.
5. Un programa que lea un archivo e imprime todo su contenido en orden inverso:

   ```
   Hola taller
   taller Hola
   ```

Standard Template Library