C++ overview

ERNESTO BASCÓN PANTOJA

There are only two kinds of languages: the ones people complain about and the ones nobody uses.

- BJARNE STROUSTRUP

El episodio de hoy llega gracias a:

Course contents (1/3)

1. Introduction to C++.
2. Stack and heap / Passing parameters by value / Pointers and references.
3. C strings / More on pointers / arrays.
4. Dynamic memory allocation / classes / structs
5. Methods / constructors / destructors / access modifiers

Course contents (2/3)

6. Inheritance and polymorphism / casting operators
7. Introduction to templates
8. `std::string` / iterators / exceptions
9. `std::vector` / `std::map` / `std::unordered_map`
10. Function pointers / functors / lambda expressions

Course contents (3/3)

11. `std::tuple` / `std::optional` / `std::any` / `std::variant`
12. Smart pointers
13. Pimpl idiom. Microsoft specifics: Libraries & DLLs
14. Windows specifics: BSTR, MFC strings & CLI/CLR

Scoring

Homeworks

- Exercises (40/100)
- Project (60/100)

Tools

1. Any C++ compiler (g++ 10+; clang 11+; Visual Studio 2019)
2. Any text editor or IDE
3. For last week, only Visual Studio.

Introduction to C++

Contents

Timeline

Timeline

Year	
1979	**Bjarne Stroustrup** created “C with classes”, that basically was C… with classes. It included: • Classes • Constructors and destructors • Derived classes • Public / private access modifiers • Type checking
1981	• Inline functions • Default arguments • Overload of assignment operator

Timeline

Year	
1982	“C with classes” successor: C++: • virtual functions • Function and operator overloading • References • const
1983	“cfront” was released. First commercial C++ compiler.
1985	“The C++ programming language” reference book 1st edition was published
1989	C++ 2.0: • Multiple inheritance • Abstract classes • Static member functions • Const member functions • Protected members

Timeline

Year	
~1990	• Templates • Exceptions • Namespaces • New casts • bool
1998	C++98 ISO standard was released.
2003	C++03 revision (ISO/IEC 14882:2003)
2011	C++11: A lot of new modern features (lambda expressions, auto, move semantics)
2014	C++14: Update released
2015	“The C++ core guidelines” started (https://isocpp.github.io/CppCoreGuidelines/)
2017	C++17: constexpr, std::string_view, std::optional, std::variant, std::any
2020	C++20: Modules, concepts, ranges, space-ship operator

Features

Features

1. Highly compatible with C.
2. Zero-cost abstractions (you don’t pay for what you don’t use)
3. Devoted to performance.
4. Extremely backwards compatible.
5. High/mid/low level programming language.
6. C++ compiler produces native binaries.
7. Highly optimized compilers

Features

1. A lot of control is left to the programmer (for performance's sake) [boundaries checking, memory handling, etc.].
2. "Undefined behavior"
3. Thin standard library

“Hello world”

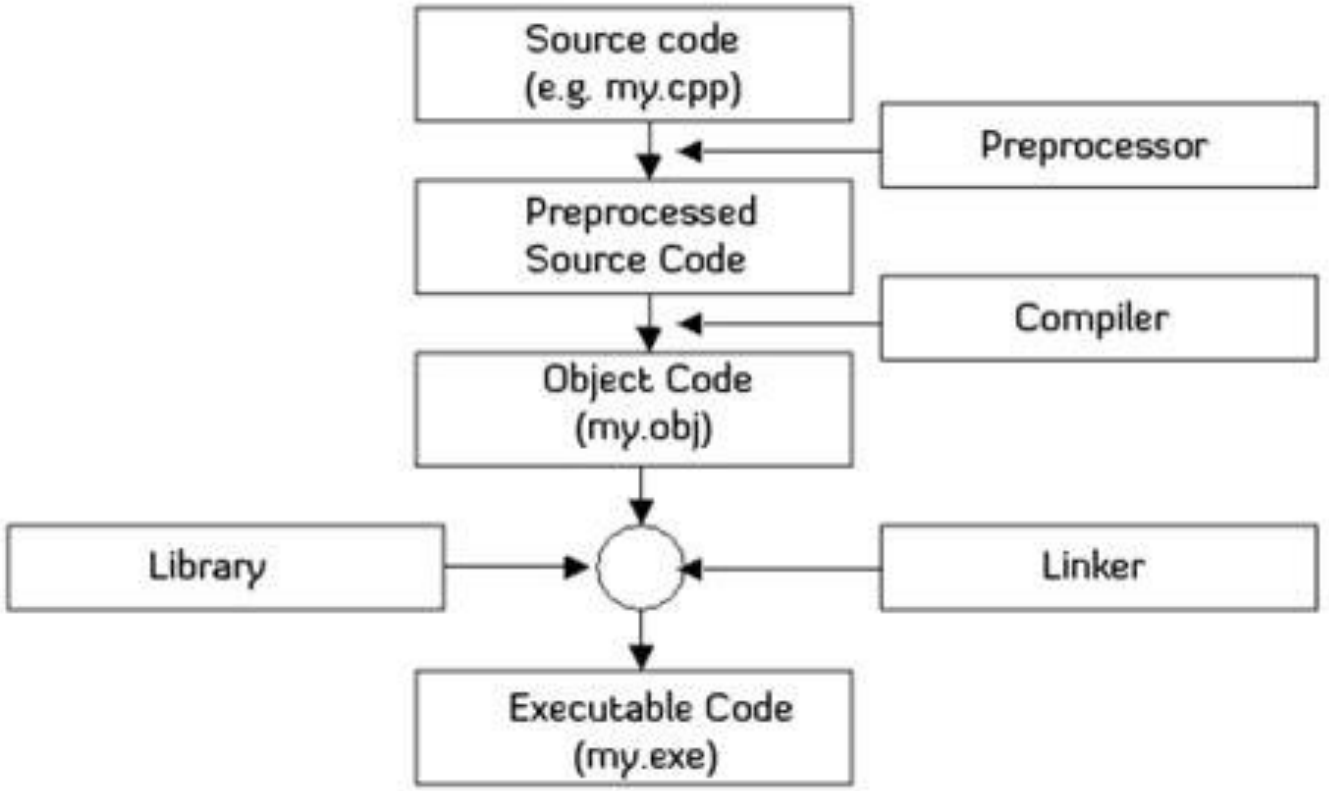
“Hello world”

```
#include <iostream>

int main()
{
    std::cout << "Hello world\n";
    return 0;
}
```

Compilation process

Compilation process



Datatypes

Datatypes

1. Null pointer type:
   - `nullptr_t`
2. Boolean values:
   - `bool`
3. Character types:
   - `char` -> 8-bit always
   - `wchar_t` -> depends on platform

Datatypes

1. Integer types:
   - `[signed] short` / `unsigned short` (generally: 16 bit)
   - `int` / `unsigned int` (generally: 32 bit)
   - `long` / `unsigned long` (Windows: 32 bit, Linux: 64 bit)
   - `long long` / `unsigned long long` (64 bit)
2. Floating-point types:
   - `float` (generally: 32 bit)
   - `double` (generally: 64 bit)
   - `long double` (generally: 80 bit)

Datatypes

1. Size type:
   - `size_t`
2. Fixed size integers types:
   - `int8_t` / `uint8_t` (8 bit)
   - `int16_t` / `uint16_t` (16 bit)
   - `int32_t` / `uint32_t` (32 bit)
   - `int64_t` / `uint64_t` (64 bit)

Literals

Literals

1. Boolean literals:
   - `true`
   - `false`
2. Character (char) literals:
   - `'a'`
   - `'\0'`

Literals

1. Wide character (wchar_t) literals:
   - `L'a'`
   - `L'\0'`
2. UTF-8, UTF-16, UTF-32 literals (since C++17):
   - `u8'a' //UTF-8`
   - `u'ठ' //UTF-16`
   - `U'居' //UTF-32`

Literals

1. int literals
   - `12385`
   - `-234245`
   - `12’634 (since C++11)`
   - `0x1F2A32BC (hexadecimal)`
   - `012532 (octal)`
   - `0b0111010111 (since C++14) (base-2)`

Literals

1. unsigned int literals
   - `12385U`
   - `-234245U`
   - `12’634U (since C++11)`

Literals

1. long literals
   - `123851234123L`
   - `-23424585622L`
   - `92’634’524’723’184L (since C++11)`
   - `0x1F2A32BC0A00B0L`
   - `012532231234L`
   - `0b01110101110101100l1L (since C++14)`

Literals

1. unsigned long literals
   - `123851234123UL`
   - `-23424585622UL`
   - `92’634’524’723’184UL (since C++11)`
   - `0x1F2A32BC0A00B0UL`
   - `012532231234UL`

Literals

1. float literals
   - `32423.2f`
   - `2E10f (2*10^2)`
   - `2.32E-5f`
2. double literals
   - `32423.23423`
   - `2E10`
   - `2.323222E-5`

Literals

1. “String” literals (const char*)
   - “Hello world”
2. “Wide-string” literals (const wchar_t*)
   - L”Hello world”
3. std::string literals:
   - “Hello world”s
4. UTF-8, UTF-16, UTF-32 “string” literals:
   - u8”Hello world”
   - u”Hello world”
   - U”Hello world”

Literals

1. Null pointer (`nullptr_t`)literal
   - `nullptr reemplaza a NULL`

Operators

Operators

1. C++ has the same operators, with the same semantics than C, Java, C#, etc.
2. C++ provides a set of new operators:
   - `new / delete`
   - `static_cast / const_cast / reinterpret_cast / dynamic_cast`
   - `::`
   - `sizeof…`
   - `typeid`
   - `noexcept`
3. Almost all C++ operators can be overloaded for custom datatypes

Arithmetic operators

- `+, +=`
- `-, -=`
- `*, *=`
- `/, /=`
- `%, %=`
- `++`
- `--`

Relational operators

- `==`
- `!=`
- `>`
- `<`
- `>=`
- `<=`
- `<=>` (C++20, not included in C)

Logical operators

- &&
- ||
- !

Bitwise operators

- `&, &=`
- `|, |=`
- `~`
- `^`
- `<<, <<=`
- `>>, >>=`

Other operators

- `::`
- `sizeof`
- `sizeof…`
- `alignof`
- `typeid`
- `new, new[]`
- `delete, delete[]`
- `noexcept`
- `static_cast / const_cast / reinterpret_cast / dynamic_cast`