Diferenças entre as versões de C++

Dr. Rodrigo Mologni Gonçalves dos Santos

Conteúdo

1. Evolução do C++
2. C++ Moderno
3. C++ Antigo vs. C++ Moderno

Evolução do C++

C++ Origem

1979 C com classes

1985 Cfront 1.0

1989 Cfront 2.0

1991 Cfront 3.0
Comitê C++ da ISO

C++ Padrão

1992 STL

1998 C++98

1999 Boost

2003 C++03

2007 TR1

C++ Moderno

2011 C++11

2014 C++14

2017 C++17

2020 C++20

C++ Moderno

C++11

Um grande número de mudanças foi introduzido para padronizar as práticas existentes e melhorar as abstrações disponíveis para os programadores C++.

C++14

Foram realizadas pequenas melhorias e correções de defeitos.

C++17

Os seguintes recursos foram incorporados: biblioteca *filesystem*, extensões da biblioteca padrão C++, extensão para paralelismo e funções matemáticas especiais.

C++20

O padrão mais recente do C++.*

C++ reference

C++11, C++14, C++17, C++20, C++23, C++26 | Compiler support C++11, C++14, C++17, C++20, C++23, C++26

- Freestanding implementations
- ASCII chart

Language
- Basic concepts
- Keywords
- Preprocessor
- Expressions
- Declarations
- Initialization
- Functions
- Statements
- Classes
- Overloading
- Templates
- Exceptions

Standard library (headers)

Named requirements

Feature test macros (C++20)

Language support library
- source_location (C++20)
- Type support
- Program utilities
- Coroutine support (C++20)
- Three-way comparison (C++20)
- numeric_limits — type_info
- initializer_list (C++11)

Concepts library (C++20)

Diagnostics library
- exception — System error
- basic_stacktrace (C++23)

Memory management library
- unique_ptr (C++11)
- shared_ptr (C++11)
- Low level management

Metaprogramming library (C++11)
- Type traits — ratio
- integer_sequence (C++14)

General utilities library
- Function objects — hash (C++11)
- Swap — Type operations (C++11)
- Integer comparison (C++20)
- pair — tuple (C++11)
- optional (C++17)
- expected (C++23)
- variant (C++17) — any (C++17)
- String conversions (C++17)
- Formatting (C++20)
- bitset — Bit manipulation (C++20)

Strings library
- basic_string — char_traits
- basic_string_view (C++17)
- Null-terminated strings: byte — multibyte — wide

Containers library
- array (C++11)
- vector — deque
- list — forward_list (C++11)
- set — multiset
- map — multimap
- unordered_map (C++11)
- unordered_multimap (C++11)
- unordered_set (C++11)
- unordered_multiset (C++11)
- stack — queue — priority_queue
- flat_set (C++23)
- flat_multiset (C++23)
- flat_map (C++23)
- flat_multimap (C++23)
- span (C++20) — mdspan (C++23)

Iterators library

Ranges library (C++20)

Algorithms library
- Execution policies (C++17)
- Constrained algorithms (C++20)

Numerics library
- Common math functions
- Mathematical special functions (C++17)
- Mathematical constants (C++20)
- Numeric algorithms
- Pseudo-random number generation
- Floating-point environment (C++11)
- complex — valarray

Date and time library
- Calendar (C++20) — Time zone (C++20)

Localizations library
- locale — Character classification

Input/output library
- Print functions (C++23)
- Stream-based I/O — I/O manipulators
- basic_istream — basic_ostream
- Synchronized output (C++20)

Filesystem library (C++17)
- path

Regular expressions library (C++11)
- basic_regex — algorithms

Concurrency support library (C++11)
- thread — jthread (C++20)
- atomic — atomic_flag
- atomic_ref (C++20)
- memory_order — condition_variable
- Mutual exclusion — Semaphores (C++20)
- future — promise — async
- latch (C++20) — barrier (C++20)

Technical specifications

Standard library extensions (library fundamentals TS)
- resource_adaptor — invocation_type

Standard library extensions v2 (library fundamentals TS v2)
- propagate_const — ostream_joiner — randint
- observer_ptr — Detection idiom

Standard library extensions v3 (library fundamentals TS v3)
- scope_exit — scope_fail — scope_success — unique_resource

Parallelism library extensions v2 (parallelism TS v2)
- simd

Concurrency library extensions (concurrency TS) — **Transactional Memory** (TM TS)

Reflection (reflection TS)

External Links — Non-ANSI/ISO Libraries — Index — std Symbol Index

Vantagens

- Fazer uso dos novos recursos da linguagem permite criar programas **mais complexos**, com **menos código**, de forma **mais segura** e mantendo a **mesma velocidade**.

C++ Antigo vs. C++ Moderno

O que significa programar em C++ Moderno?

Fazer uso dos recursos, tais como:

- Aquisição de Recurso é Inicialização (RAII)
- As bibliotecas da STL
- Funções e classes genéricas
- Semântica de transferência
- Ponteiros inteligentes
- Funções lambdas
- etc.

Ponteiros inteligentes

```
void func()
{
    int* valuePtr = new int(15);
    int x = 45;
    if (x == 45)
        return;
    delete valuePtr;
}
```

```
void func()
{
    std::unique_ptr<int> valuePtr(new int(15));
    int x = 45;
    if (x == 45)
        return;
}
```

Declaração for **baseada em intervalo**

```
for (int n = 0; n != sizeof(array); ++n) {
    /* variable declaration */ = array[n];
    // statements
}
```

```
for (auto item : items) {
    // statements
}
```

Inicializadores designados

```
struct Position { int x; int y; int z; };

Position a { 0, 1, 2}; // x=0 y=1 z=2
Position b { 0, 2 };   // x=0 y=2 z=?
```

```
struct Position { int x; int y; int z; };

Position a { .y = 1, .z = 2 }; // x=0
Position b { .x = 0, .z = 2 }; // y=0
```

Declaração vinculativa estruturada

```cpp
unordered_map<string, pair<int, int>> hash_table;

pair<int, int> elem = hash_table["Up"];
std::cout << elem.first;
std::cout << elem.second;
```

```cpp
unordered_map<string, pair<int, int>> hash_table;

auto [id, size] = hash_table["Up"];
std::cout << id;
std::cout << size;
```

Referências

Chapter 16: History and Compatibility