Let’s Talk About C++ Abstraction Layers

Inbal Levi
Let’s Talk About C++
Abstraction Layers

Inbal Levi
Who Am I

- Lead C++ Developer at MPGC Services Ltd
- Active member of ISO C++ work group (WG21):
  - Israeli NB Chair
  - Ranges SG Chair
- C++ International Conferences
  - Core C++ 2023 Organizer
  - C++Now 2023 Program Chair
- I love language & software design, I also love cataloguing
What This Talk Is About

I. What are Abstraction Layers?

II. Abstraction Layers model for C++

III. Existing solutions

IV. Future solutions -or- How can we do better?
Part I: What Are Abstraction Layers
Part 0: What Are Abstractions

- Software development is all about communicating logic to the computer.
- To achieve that, we need to apply some level of abstraction.

Abstraction (computer science)

- The process of removing or generalizing physical, spatial, or temporal details or attributes in the study of objects or systems to focus attention on details of greater importance; it is similar in nature to the process of generalization;
- the creation of abstract concept-objects by mirroring common features or attributes of various non-abstract objects or systems of study— the result of the process of abstraction.
Part 0: What Are Abstractions

```c
int main()
{
    int arr[] = {1, 2, 3};
    int* ptr = arr;
    for (auto &ai : arr)
    {
        printf("Int: \t%d\n", ai);
        // C++23
    }
}
```
Part 0: What Are Abstractions

- Choosing messaging technique to use for the following system?
  1. Push
  2. Pull
  3. Message board
  4. Something else...?

- We need more information...
  a. Which technology?
  b. Which components?
  c. Which latency requirements?
Part 0: What Are Abstractions

Under Abstraction

The essence of abstraction is preserving information that is relevant in a given context, and forgetting information that is irrelevant in that context.

– John V. Guttag  
(former head of EE and CS at MIT)
Part I: What Are Abstraction Layers

```c
int main()
{
    char arr[] = {'a', 'b'};
    char* ptr = arr;
    for (auto i = 0; i < SIZE; i++)
    {
        printf("Char: \t%c, pointer location: %X\n", arr[i], ptr);
        ptr++;
    }
}
```

<table>
<thead>
<tr>
<th>Types</th>
<th>Memory Layout (Bytes, bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Char: a</td>
<td>78744098</td>
</tr>
<tr>
<td>Char: b</td>
<td>78744099</td>
</tr>
<tr>
<td>Int: 1</td>
<td>C0ED03E0</td>
</tr>
<tr>
<td>Int: 2</td>
<td>C0ED03E4</td>
</tr>
</tbody>
</table>
Part I: What Are Abstraction Layers

```c
int main()
{
    int i = 0;
    // std::cout << &i;              // 0x7ffc8584085c
    *(int*)0x7ffc8584085c = 1;    // UB (when don’t store the value)
    return i;
}
```

- The example is about:
  - The duality of int and memory address
  - The invalidity of the address
  - The UB created by using the address

- P2434: Nondeterministic pointer provenance
Part I: What Are Abstraction Layers

- We don’t care about this, we’re modern C++ developers… right?
- Wrong.
  - N2443: Lifetime-End Pointer Zap (2019)
  - P2318: A Provenance-aware Memory Object Model for C (2021)
  - P1726: Pointer lifetime-end zap and provenance, too (2021)
  - P2434: Nondeterministic pointer provenance (2022)
Part II: Abstraction Layers Model for C++
Part II: Abstraction Layers Model for C++

- We will analyze C++ language and library, building a “layers” model
- We want to identify the borders between layers
- We want to recognize the “dangerous” parts, where bugs occur
Part II: Abstraction Layers Model for C++

- The example is about three topics:
  - The invalidity of the address
  - The duality of int and memory address
  - The UB created by using the address

```cpp
int main()
{
    int i = 0;
    std::cout << &i;
    *(int*)0x7ffc8584085c = 1;
    return i;
}
```
Part II: Abstraction Layers Model for C++

- Types
  - bool, char, `<stdfloat>`, etc.
  - casts, `<charconv>`
  - cv qualifiers
  - `<limits>`

- Layout Control:
  - Pointers, References
  - `alignas`, `alignof` - query and set alignment of primitives and structs

- Memory Allocation
  - `delete`, `new`
  - `<memory_resource>`
  - `<scoped_allocator>`
Part II: Abstraction Layers Model for C++

- Binary Level Queries & Modification
  - asm - inline assembly block
  - inline
  - goto
  - <source_location>

- Linkage & Modules
  - extern
  - export, import, module
Part II: Abstraction Layers Model for C++
Part II: Abstraction Layers Model for C++
Part II: Abstraction Layers Model for C++
Part II: Keywords by Layers

A. Memory Access:
- register (2)
- mutable (1)
- static (B)
- thread_local (C++11) (B)
- volatile (1)

B. Memory Allocation:
- delete (1)
- new

C. Memory Layout Control (Bytes, bits):
- alignas (C++11)
- alignof (C++11)

D. Binary Level Modifications:
- asm
- inline (1)
- goto

E. Logic operators:
- and
- and_eq
- bitand
- bitor
- compl
- false
- true
- or
- or_eq
- not
- not_eq
- switch
- xor
- xor_eq

1. Types:
- bool
- char
- char8_t (C++20)
- char16_t (C++11)
- char32_t (C++11)
- const (A)
- double
- enum
- float
- int
- long
- nullptr (C++11)
- short
- signed (C)
- unsigned (C)
- union
- const_cast
- dynamic_cast (b)
- reinterpret_cast (A)
- static_cast (A)
- void
- wchar_t

2. Error Handling:
- catch
- try
- throw
- noexcept (C++11)

A. Object Oriented:
- class (1)
- default (1)
- explicit
- friend
- private
- protected
- public
- operator
- struct (1)
- this (4)
- namespace
- using (1)
- virtual

1. Types (A)
- enum
- float
- int
- long
- short
- signed (C)
- unsigned (C)
- union
- const_cast
- dynamic_cast (b)
- reinterpret_cast (A)
- static_cast (A)
- void
- wchar_t

2. Error Handling:
- catch
- try
- throw
- noexcept (C++11)

A. Object Oriented:
- class (1)
- default (1)
- explicit
- friend
- private
- protected
- public
- operator
- struct (1)
- this (4)
- namespace
- using (1)
- virtual

1. Types (A)
- enum
- float
- int
- long
- short
- signed (C)
- unsigned (C)
- union
- const_cast
- dynamic_cast (b)
- reinterpret_cast (A)
- static_cast (A)
- void
- wchar_t

2. Error Handling:
- catch
- try
- throw
- noexcept (C++11)

A. Object Oriented:
- class (1)
- default (1)
- explicit
- friend
- private
- protected
- public
- operator
- struct (1)
- this (4)
- namespace
- using (1)
- virtual

1. Types (A)
- enum
- float
- int
- long
- short
- signed (C)
- unsigned (C)
- union
- const_cast
- dynamic_cast (b)
- reinterpret_cast (A)
- static_cast (A)
- void
- wchar_t

2. Error Handling:
- catch
- try
- throw
- noexcept (C++11)

a) Control Flow:
- break
- case
- continue
- do
- else
- for
- if
- return
- while
- (Pattern Matching)
b) Linkage Control & Modules:
- import
- module
- export (1) (3)
- extern (1)
c) Coroutines:
- co_await (C++20)
- co_return (C++20)
- co_yield (C++20)
d) Experimental (non-layer):
- atomic_cancel (TM TS)
- atomic_commit (TM TS)
- atomic_noexcept (TM TS)
- reflexexpr (reflection TS)
- synchronized (TM TS)
Part II: Library Headers by Layers

A. Memory Access
   - <memory>
   - <new>
   - <memory_resource>
   - <scoped_allocator>

B. Memory Allocation:
   - <memory>
   - <new>
   - <memory_resource>
   - <scoped_allocator>

C. Memory Layout Control
   - <source_location>
   - <stacktrace>

D. Binary Level Modifications:
   - <source_location>
   - <stacktrace>

E. Logic operators:
   - <regex>

F. Algorithms:
   - <algorithm>
   - <compare>
   - <initializer_list>
   - <utility>
   - <vector>

G. Iterators:
   - <iterator>

H. Ranges:
   - <ranges>

I. Numerics:
   - <complex>
   - <numeric>
   - <numbers>
   - <random>
   - <ratio>
   - <valarray>

J. File System:
   - <filesystem>
   - <fstream>

1. Types:
   - <bit> (C)
   - <limits>
   - <charconv>
   - <stdfloat>

2. Error Handling:
   - <expected>
   - <exception>
   - <stdexcept>
   - <system_error>

3. Containers & Adaptors:
   - <array>
   - <bitset>
   - <deque>
   - <forward_list>
   - <list>
   - <map>
   - <queue>
   - <set>
   - <stack>
   - <string>
   - <unordered_map>
   - <unordered_set>
   - <tuple>
   - <vector>
   - <string_view>
   - <span>
   - <mdspan>
   - <flat_set>
   - <flat_map>

a. Object Oriented:
   - <any>
   - <optional>
   - <variant>
   - <typeinfo> [1]

b. Type Query:
   - <type_traits> [c]
   - <typeindex>

c. Compile Time Conditions
   - Compile Time Hints:
     - <version>
     - Concepts:
       - <concepts>

d. Compile Time Placeholders

e. Localization & Time:
   - <locale>
   - <codecvt>
   - <chrono>

f. I/O:
   - <format>
   - <ios>
   - <iostream>
   - <istream>
   - <ostream>
   - <streetview>
   - <streambuf>
   - <stringstream>
   - <strstream>

a) Control Flow
b) Linkage Control & Modules
c) Coroutines:
   - <coroutine>
   - <generator> [H]
d) Concurrency / parallelism:
   - <atomic>
   - <barrier>
   - <condition_variable>
   - <thread>
   - <latch>
   - <mutex>
   - <semaphore>
   - <shared_mutex>
   - <stop_token>

e) Async
   - <future>
   - <execution> [F.]

01. C library (non-layer):
   - <cassert>
   - <clocale>
   - <cstdarg>
   - <cstring>
   - <cctype>
   - <cmath>
   - <cstddef>
   - <ctime>
   - <cerrno>
   - <csetjmp>
   - <cstdio>
   - <cwchar>
   - <cfloat>
   - <csignal>
   - <cstdlib>
   - <cwctype>
   - <climits>
We can identify different parts of the program with different layers.
```cpp
#include <sstream>  // I/O

int main()
{
    auto iss = std::istringstream("0 1 2");
    auto j = 0;
    while (iss >> j)
    {
        std::cout << "j: " << j << '\n';
    }
}
```

j: 0
j: 1
j: 2
```cpp
#include <sstream>   // I/O
#include <ranges>    // Ranges

int main()
{
    auto iss = std::istringstream("0 1 2");
    for (auto i : rn::istream_view<int>(iss) | rv::take(1))
        std::cout << "j in loop: " << i << '\n';

    auto j = 0;
    iss >> j; // Extraction
    std::cout << "j after loop: " << j << '\n';
}
```
Part II: Abstraction Layers Model for C++

- **P2406**: Add `lazy_counted_iterator` (2023)
- **P2799**: Closed ranges may be a problem (...) (2023)
- **P2846**: size_hint: Eagerly reserving memory for non-quite-sized lazy ranges (2023)
- More to follow...
Part II: Abstraction Layers Model for C++

```cpp
int main()
{
    auto iss = std::istringstream("0 1 2"); // I/O
    for (auto i : rn::istream_view<int>(iss) | rv::take(1)) // Ranges
        std::cout << "In loop: " << i << '\n';
    auto j = 0;
    iss >> j; // I/O
    std::cout << "After loop: " << j << '\n';
}
```

```
In loop: 0
After loop: 2
```

- `ranges::views::take` modifies the layer type of `iss`
Part III: Existing Solutions
Part III: Existing Solutions

- C++ “operates” in especially large number of layers
- Abstraction Layers are the glue areas between features
- These interaction are “dangerous” areas, on which bugs occur
- Our logic should take the transformation under consideration
Part III: Existing Solutions

- **Solution I: Create better code**
  - Create boundaries: Encapsulation, Namespaces, Headers, Modules, etc.
  - Use code guidelines, tools, etc. to enforce those

```cpp
int main()
{
    auto iss = std::istringstream("0 1 2");
    for (auto i : rn::istream_view<int>(iss) | rv::take(1))
        std::cout << "In loop: " << i << '
';
    auto j = 0;
    iss >> j;  // Error
    std::cout << "After loop: " << j << '
';
}
```
Part III: Existing Solutions

- **Solution I: Create better code**
  - Create boundaries: Encapsulation, Namespaces, Headers, Modules, etc.
  - Use code guidelines, tools, etc. to enforce those
  - Example of such: CppCon 2021: Up to Code / David Sankel
  - **Upsides:**
    - Existing and familiar idioms, utilities, guidelines, tools 🌛
  - **Downsides:**
    - May create unacceptable overhead 🚫
    - Enforcing is a challenge, especially in large projects, multiple teams 🚫
    - Does not help with cross-boundaries code 🚫
Part III: Existing Solutions

- **Solution II: Use a different language for higher level logic**
  - Modern C++ provides a partial solution for shifting between abstractions
  - Following modern idioms will minimize logic errors in our code

- **Upsides:**
  - Discard previous error-prone code “for free” 🍍

- **Downsides:**
  - Learn (and deploy!) a “new language” with each standard release 🍋
  - We didn’t provide a solution for existing code 🍋
  - We didn’t provide a solution for Assembly-C-C++ code bases 🍋
  - Does not help with cross-boundaries code 🍋
Part IV: Future solutions

-or-

How can we do better?
Part IV: How Can We Do Better?

- We need to apply this model on our code
- Apply to language / library syntax creates a new language (or use python 😁)
- Solution: apply the logic in the error messages level
- Applying this model to errors/warnings preserves syntax compatibility, but at the same time - moves us to a higher level of analytics
Part IV: How Can We Do Better?

- Static Analysis Tool: Classify tokens according to layers

```cpp
#include <iostream>

int main()
{
    std::array<int, SIZE> arr = {0, 1, 2};
    std::cout << sizeof(arr);
}
```

Warning! On line 6, variable `arr` changes abstraction layer from ~Containers to ~Types
Part IV: How Can We Do Better?

```cpp
int main()
{
    std::istringstream iss("0 1 2");
    for (int i : rn::istream_view<int>(iss) | rv::take(1))
        std::cout << "In loop: " << i << '\n';
    int j = 0;
    iss >> j;
    std::cout << "After loop: " << j << '\n';
}
```

Warning! On line 4, variable `iss` changes abstraction layer from ~IO to ~Ranges

Warning! On line 7, variable `iss` changes abstraction layer from ~Ranges to ~IO
### Part IV: How Can We Do Better?

```cpp
#include <sstream>
#include <ranges>

int main()
{
    std::stringstream iss("0 1 2");
    for (int i : rn::istream_view<int>(iss)) { // line 7
        std::cout << "j in loop: " << i << \n);
    }
    int j = 0;
    iss >> j; // line 11
    std::cout << "j after loop: " << j << \n);
}
```

**Warning:** On line 7, variable `iss` changes abstraction layer from IO to Ranges.
Part IV: How Can We Do Better?

- Coroutine is a function which can suspend execution (C++20)

```cpp
Task doWork(); // Coroutine

struct Task {
    struct promise_type {
        HandleWrap get_return_object() { return HandleWrap(this); }
        std::suspend_always initial_suspend() { ... }
        struct HandleWrap {
            void resume() { std::cout << "Work\n"; mHandle.resume(); }
        }
    }
};

int main()
{
    auto work_handle = doWork();
    work_handle.resume();
}
```
Part IV: How Can We Do Better?

- `std::execution` is an async execution library, planned to go into C++26

```cpp
scheduler auto sch = thread_pool.scheduler(); // Scheduler
sender auto begin = schedule(sch);
sender auto doWork = then(schedule(sch), [] {
    std::cout << "Work\n";
});

int main()
{
    this_thread::sync_wait(doWork);
}
```
Part IV: How Can We Do Better?

```cpp
indic_dict = { "doWork": "Coroutines", ... }
token_dict = { "Task": "Coroutines", "sync_wait": "Async" ... }

Task doWork();  // Coroutine
struct Task {
  struct promise_type {
    HandleWrap get_return_object() { return HandleWrap(this); }
    std::suspend_always initial_suspend() { ... }
    void return_void { std::cout << "Work\n"; }
  }
};

int main()
{
  this_thread::sync_wait(doWork());  // Awaitable satisfies requirements for senders
}

Warning! On line 3, variable `doWork` changes abstraction layer from ~Coroutines to ~Async
```
Part IV: How Can We Do Better?

- Templates should be considered with the instantiation type

```cpp
#include <functional>
#include <string>
#include <filesystem>

void foo()
{
    std::function<std::filesystem::path()> fp{};
    std::function<std::string()> fs{};
    fp = fs; // Works in MSVC, clang, gcc
    fs = fp; // Works only in GCC and clang
}
```

*Example by Hana Dusíková*
Part IV: How Can We Do Better?

- False positives are common

```cpp
#include <iostream>

int main()
{
    int a = 5;
    int& ra = a;
    std::cout << ra;
}
```

Warning! On line 7, variable `ra` changes abstraction layer from ~Layout to ~IO
Part IV: How Can We Do Better?

- April 2023, committee ML: Can we use UTF8 strings with std::format()? (C++20)

```cpp
auto s = u8"Köln";
std::format("Stadt: {}\n", s); // ERROR
```

- P2728R0: Unicode in the Library, Part 1 & Part 2 (Zach Laine) (C++26)

```cpp
int main()
{
    std::string input = get_utf8_input();
    auto const utf16_view = std::uc::as_utf16(input);
    process_input(utf16_view.begin(), utf16.end()); // accepts UTF-16
}
```

- Formatters added in P2728 allow UTF views will be used in std::format()
Part IV: How Can We Do Better?

```cpp
#include <iostream>

int main()
{
    int a = 5;
    std::cout << a;
}
```

Compilation Errors

Runtime Errors

Logic Errors
Part IV: How Can We Do Better?

- Static Analysis Tool: Classify tokens according to layers

```c
int main()
{
    int i = 0;
    *(int*)0x7ffc8584085c = 1;
    return i;
}
```

Warning! On line 4, int literal changes abstraction layer from ~Type to ~Layout
Part IV: How Can We Do Better?

- Add a layer of analytics:
  
  I. **Compile time**: errors in syntax and software model (e.g. type system)
  
  II. **Abstractions resolution**: errors in logic and composition
  
  III. **Runtime**: errors in dynamic data

- Applying abstraction layers to the model exposes logic bugs on earlier stage

- **Do this by adding the abstraction layers classifications into**:
  
  I. Compilers
  
  II. Static Analysis Tools
  
  III. Other tools which generates AST (CastXML, etc.)
Part IV: How Can We Do Better - The Full Solution

● Focus on interfaces with the user, e.g:

   Create an “ergonomics” study group
   - David Sankel

● But also, we should:
  ○ Address abstraction layers model as developers
  ○ Address abstraction layers model as the standards committee
  ○ Examine every proposal not only for “local” usability, but also for integration

  ○ Add Abstraction Layers Error Messages to Our Tools

Thank you for listening!
Thanks!

Thank you for listening 😊

Special thanks to:
- Yehezkel Brant
- Corentin Jabot
- NYC++ Meetup group
- Barry Revzin
- Bryce Adelstein Lelbach
- Aditya - layers in IDE
- Vern - user defined layers
- Amir - teachability

Linkedin.com/inballevi
Twitter.com/Inbal_I
sinbal2leextra@gmail.com

Would love to get your input!
References & More Info

- **Papers:**
  - P2406R5: Add `lazy_counted_iterator` (Yehezkel Bernat)
  - P2846R0: size_hint: Eagerly reserving memory for non-quite-sized lazy ranges (Corentain Jabot)
  - P2728R0: Unicode in the Library, Part 1 & Part 2 (Zach Laine)
  - P2434: Nondeterministic pointer provenance

- **Blogs:**
  - What Color Is Your Function / Bob Nystorm
  - C++ Buffer Hardening / Jan Korous

- **Talks:**
  - CppCon 2021: Up to Code / David Sankel
  - CppNow 2022: Rust Features that I Want in C++ / David Sankel
  - CppNow 2023: Take Five Adventures with Taking Elements from an Input Stream / Barry Revzin

- **Books:**
  - Abstraction and Specification in Program Development / Barbara Liskov and John Guttag
Part II: Abstraction Layers Model for C++

Diagram showing the Abstraction Layers Model for C++. The layers include:

- **Concurrency/Parallelism**
- **Async**
- **Coroutines**

- **Localization & Time**
- **Algorithms**
- **Containers & Adaptors**
- **ranges**

- **File System**
- **I/O**
- **Error Handling**
- **Iterators**

- **Logic Op**
- **Control Flow**
- **Compile Time Conditions**
- **Compile Time placeholders**
- **Type Query**
- **Object Oriented**
- **Numerics**

- **Linkage & Modules**
- **Binary Modification**
- **Layout Control**
- **Types**
- **Memory Access**
- **Memory Allocation**

- **Program & Source Code**
- **Memory Layout (Bytes, bits)**
C++ Keywords

- alignas (C++11)
- alignof (C++11)
- and
- and_eq
- asm
- atomic_cancel (TM TS)
- atomic_commit (TM TS)
- atomic_noexcept (TM TS)
- auto (1)
- bitand
- bitor
- bool
- break
- case
- catch
- char
- char8_t (C++20)
- char16_t (C++11)
- char32_t (C++11)
- class (1)
- compl
- concept (C++20)
- const
- consteval (C++20)
- constexpr (C++11)
- constinit (C++20)
- const_cast
- continue
- co_await (C++20)
- co_return (C++20)
- co_yield (C++20)
- decltype (C++11)
- default (1)
- delete (1)
- do
- double
- dynamic_cast
- else
- enum
- explicit
- export (1) (3)
- extern (1)
- false
- float
- for
- friend
- goto
- if
- inline (1)
- int
- long
- mutable (1)
- namespace
- new
- noexcept (C++11)
- not
- not_eq
- nullptr (C++11)
- operator
- or
- or_eq
- private
- protected
- public
- reflector (reflection TS)
- requires (C++20)
- return
- short
- signed
- sizeof (1)
- static
- static_assert (C++11)
- static_cast
- struct (1)
- switch
- synchronized (TM TS)
- template
- this (4)
- thread_local (C++11)
- throw
- true
- try
- typedef
- typeid
- typename
- union
- unsigned
- using (1)
- virtual
- void
- volatile
- wchar_t
- xor
- xor_eq
<table>
<thead>
<tr>
<th>Library Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;any&gt;</code></td>
</tr>
<tr>
<td><code>&lt;bitset&gt;</code></td>
</tr>
<tr>
<td><code>&lt;chrono&gt;</code></td>
</tr>
<tr>
<td><code>&lt;compare&gt;</code></td>
</tr>
<tr>
<td><code>&lt;concepts&gt;</code></td>
</tr>
<tr>
<td><code>&lt;coroutines&gt;</code></td>
</tr>
<tr>
<td><code>&lt;csetjmp&gt;</code></td>
</tr>
<tr>
<td><code>&lt;signal&gt;</code></td>
</tr>
<tr>
<td><code>&lt;stdarg&gt;</code></td>
</tr>
<tr>
<td><code>&lt;stddef&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cstdlib&gt;</code></td>
</tr>
<tr>
<td><code>&lt;ctime&gt;</code></td>
</tr>
<tr>
<td><code>&lt;expected&gt;</code></td>
</tr>
<tr>
<td><code>&lt;functional&gt;</code></td>
</tr>
<tr>
<td><code>&lt;initializer_list&gt;</code></td>
</tr>
<tr>
<td><code>&lt;optional&gt;</code></td>
</tr>
<tr>
<td><code>&lt;source_location&gt;</code></td>
</tr>
<tr>
<td><code>&lt;tuple&gt;</code></td>
</tr>
<tr>
<td><code>&lt;utility&gt;</code></td>
</tr>
<tr>
<td><code>&lt;variant&gt;</code></td>
</tr>
<tr>
<td><code>&lt;version&gt;</code></td>
</tr>
<tr>
<td><code>&lt;memory&gt;</code></td>
</tr>
<tr>
<td><code>&lt;memory_resource&gt;</code></td>
</tr>
<tr>
<td><code>&lt;new&gt;</code></td>
</tr>
<tr>
<td><code>&lt;scoped_allocator&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cfloat&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cinttypes&gt;</code></td>
</tr>
<tr>
<td><code>&lt;climits&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cstdint&gt;</code></td>
</tr>
<tr>
<td><code>&lt;limits&gt;</code></td>
</tr>
<tr>
<td><code>&lt;stdfloat&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cassert&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cerrno&gt;</code></td>
</tr>
<tr>
<td><code>&lt;exception&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cstring&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cwchar&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cwctype&gt;</code></td>
</tr>
<tr>
<td><code>&lt;charconv&gt;</code></td>
</tr>
<tr>
<td><code>&lt;ctype&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cstring&gt;</code></td>
</tr>
<tr>
<td><code>&lt;uchar&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cwchar&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cwctype&gt;</code></td>
</tr>
<tr>
<td><code>&lt;charconv&gt;</code></td>
</tr>
<tr>
<td><code>&lt;ctype&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cstring&gt;</code></td>
</tr>
<tr>
<td><code>&lt;uchar&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cwchar&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cwctype&gt;</code></td>
</tr>
<tr>
<td><code>&lt;string&gt;</code></td>
</tr>
<tr>
<td><code>&lt;string_view&gt;</code></td>
</tr>
<tr>
<td><code>&lt;iterator&gt;</code></td>
</tr>
<tr>
<td><code>&lt;generator&gt;</code></td>
</tr>
<tr>
<td><code>&lt;ranges&gt;</code></td>
</tr>
<tr>
<td><code>&lt;algorithm&gt;</code></td>
</tr>
<tr>
<td><code>&lt;execution&gt;</code></td>
</tr>
<tr>
<td><code>&lt;bit&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cfenv&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cmath&gt;</code></td>
</tr>
<tr>
<td><code>&lt;complex&gt;</code></td>
</tr>
<tr>
<td><code>&lt;numbers&gt;</code></td>
</tr>
<tr>
<td><code>&lt;numeric&gt;</code></td>
</tr>
<tr>
<td><code>&lt;random&gt;</code></td>
</tr>
<tr>
<td><code>&lt;ratio&gt;</code></td>
</tr>
<tr>
<td><code>&lt;valarray&gt;</code></td>
</tr>
<tr>
<td><code>&lt;locale&gt;</code></td>
</tr>
<tr>
<td><code>&lt;codecvt&gt;</code></td>
</tr>
<tr>
<td><code> (deprecated in C++17)</code></td>
</tr>
<tr>
<td><code>&lt;locale&gt;</code></td>
</tr>
<tr>
<td><code> (deprecated in C++98)</code></td>
</tr>
<tr>
<td><code>&lt;span&gt;</code></td>
</tr>
<tr>
<td><code>&lt;stack&gt;</code></td>
</tr>
<tr>
<td><code>&lt;unordered_map&gt;</code></td>
</tr>
<tr>
<td><code>&lt;unordered_set&gt;</code></td>
</tr>
<tr>
<td><code>&lt;vector&gt;</code></td>
</tr>
<tr>
<td><code>&lt;ios&gt;</code></td>
</tr>
<tr>
<td><code>&lt;iosfwd&gt;</code></td>
</tr>
<tr>
<td><code>&lt;iostream&gt;</code></td>
</tr>
<tr>
<td><code>&lt;fstream&gt;</code></td>
</tr>
<tr>
<td><code>&lt;iomanip&gt;</code></td>
</tr>
<tr>
<td><code>&lt;iostream&gt;</code></td>
</tr>
<tr>
<td><code>&lt;istream&gt;</code></td>
</tr>
<tr>
<td><code>&lt;ostream&gt;</code></td>
</tr>
<tr>
<td><code>&lt;spanstream&gt;</code></td>
</tr>
<tr>
<td><code>&lt;streambuf&gt;</code></td>
</tr>
<tr>
<td><code>&lt;strstream&gt;</code></td>
</tr>
<tr>
<td><code>&lt;future&gt;</code></td>
</tr>
<tr>
<td><code>&lt;latch&gt;</code></td>
</tr>
<tr>
<td><code>&lt;mutex&gt;</code></td>
</tr>
<tr>
<td><code>&lt;semaphore&gt;</code></td>
</tr>
<tr>
<td><code>&lt;shared_mutex&gt;</code></td>
</tr>
<tr>
<td><code>&lt;stop_token&gt;</code></td>
</tr>
<tr>
<td><code>&lt;thread&gt;</code></td>
</tr>
<tr>
<td><code>&lt;array&gt;</code></td>
</tr>
<tr>
<td><code>&lt;deque&gt;</code></td>
</tr>
<tr>
<td><code>&lt;flat_map&gt;</code></td>
</tr>
<tr>
<td><code>&lt;flat_set&gt;</code></td>
</tr>
<tr>
<td><code>&lt;forward_list&gt;</code></td>
</tr>
<tr>
<td><code>&lt;list&gt;</code></td>
</tr>
<tr>
<td><code>&lt;map&gt;</code></td>
</tr>
<tr>
<td><code>&lt;mdspan&gt;</code></td>
</tr>
<tr>
<td><code>&lt;queue&gt;</code></td>
</tr>
<tr>
<td><code>&lt;set&gt;</code></td>
</tr>
<tr>
<td><code>&lt;span&gt;</code></td>
</tr>
<tr>
<td><code>&lt;stack&gt;</code></td>
</tr>
<tr>
<td><code>&lt;unordered_map&gt;</code></td>
</tr>
<tr>
<td><code>&lt;unordered_set&gt;</code></td>
</tr>
<tr>
<td><code>&lt;vector&gt;</code></td>
</tr>
</tbody>
</table>