C++26:

Ein Überblick

Rainer Grimm

Schulung, Mentoring und Technologieberatung

C++26

Kernsprache

- [ ] Reflektion
- [ ] Contracts
- [ ] Platzhalter
- [ ] Template-Verbesserungen
- [ ] `delete` mit Grund

Bibliothek

- [ ] `std::inplace_vector`
- [ ] Unterstützung der linearen Algebra
- [ ] `std::submdspan`
- [ ] Unterstützung bei der Fehlersuche

Concurrency

- [ ] `std::execution`
- [ ] Data-Parallel Typen (SIMD)

C++26

Kernsprache

- [x] Reflektion
- [x] Contracts
- [x] Platzhalter
- [x] Template-Verbesserungen
- [x] `delete` mit Grund

Bibliothek

- [ ] `std::inplace_vector`
- [ ] Unterstützung der linearen Algebra
- [ ] `std::submdspan`
- [ ] Unterstützung bei der Fehlersuche

Concurrency

- [ ] `std::execution`
- [ ] Data-Parallel Typen (SIMD)

Reflektion

Reflektion ist die Fähigkeit eines Programms, seine Struktur und sein Verhalten zu untersuchen, zu analysieren und zu ändern.

```
int main() {
    constexpr auto r = ^^int;
    typename[:r:] x = 42;          // Same as: int x = 42;
    typename[:^^char:] c = '*';    // Same as: char c = '*';

    static_assert(std::same_as<decltype(x), int>);
    static_assert(std::same_as<decltype(c), char>);
    assert(x == 42);
    assert(c == '*');
}
```

- **^^: Reflektion Operator** erzeugt einen Reflektion-Wert aus seinem Operanden (`^^int` und `^^char`)
- **`[:refl:]`: Splicer** erzeugt ein grammatisches Element aus einem Reflektion-Wert (`[:r:]` und `[:^^char:]`)
- **`Reflektion-Wert`** ist eine Darstellung von Programmelementen als konstanter Ausdruck

Reflektion

- Reflektion
  - Proposal P2996R5
  - ist ein minimal viable product
  - unterstützt viele Metafunktionen

- Metafunktionen
  - als `consteval` deklariert
  - den Reflektion-Wert `std::meta::info` akzeptieren

- Reflektion-Operator (`^^`)
  - erstellt `std::meta::info`

daveed.cpp
getSize.cpp
-

Contracts

Ein **Contract** spezifiziert Schnittstellen für Softwarekomponenten auf präzise und überprüfbare Weise.

- Die Softwarekomponenten sind Funktionen und Methoden, die preconditions, postconditions und invariants erfüllen müssen.
  - Eine **preconditions**: ein Prädikat, das bei der Eingabe in eine Funktion gelten soll.
  - Eine **postconditions**: ein Prädikat, das beim Verlassen der Funktion gelten soll.
  - Eine **invariant**: ein Prädikat, das an seinem Punkt in der Berechnung gelten soll.

- Die Contracts basieren auf dem Proposal P2961R2.

Contracts

```
int f(int i)
    pre (i >= 0)
    post (r: r > 0)
{
    contract_assert (i >= 0);
    return i+1;
}
```

`pre` und `post`

- fügt eine precondition (postconditions) hinzu. Eine Funktion kann eine beliebige Anzahl von preconditions (postconditions) haben. Sie können beliebig miteinander vermischt werden.
- sind kontextbezogene Schlüsselwörter
- stehen am Ende der Funktionsdeklaration

`post`

- kann einen Rückgabewert haben. Dem Prädikat muss ein Bezeichner vorangestellt werden, gefolgt von einem Doppelpunkt.

`contract_assert`

- ist ein Schlüsselwort. Andernfalls könnte es nicht von einem Funktionsaufruf unterschieden werden.

contract.cpp

Platzhalter

Platzhalter sind eine gute Möglichkeit, nicht mehr benötigte Variablen hervorzuheben.

Platzhalter

- ist der Unterstrich (_)
- kann beliebig oft verwendet werden
- gibt keine Warnung aus, wenn sie nicht verwendet wird
- wird häufig in Python verwendet

platzhalter2.cpp

Template-Verbesserungen

Pack Indexing ermöglicht den Indexzugriff auf die Elemente des Parameterpacks.

Indizierung der Packs

- Kann Ihre Lieblings-Template-Verbesserung sein, wenn Sie ein Freund der Template-Metaprogrammierung sind
- basiert auf dem Proposal P2662R3

packIndexing.cpp

`delete` mit Grund

Mit C++26 können Sie einen Grund für Ihr `delete` angeben.

- `delete` mit Grund
  - wird zur Best Practice
  - stützt sich auf den Proposal p2573r2

deleteReason.cpp

C++26

Kernsprache

- [ ] Reflektion
- [ ] Contracts
- [ ] Platzhalter
- [ ] Template-Verbesserungen
- [ ] `delete` mit Grund

Bibliothek

- [x] `std::inplace_vector`
- [x] Unterstützung der linearen Algebra
- [x] `std::submdspan`
- [x] Unterstützung bei der Fehlersuche

Concurrency

- [ ] `std::execution`
- [ ] Data-Parallel Typen (SIMD)

`std::inplace_vector`

`std::inplace_vector`

- Vektor mit dynamischer Größenänderung und zur Compilezeit festgelegter Kapazität
- zusammenhängender eingebetteter Speicher, in dem die Elemente innerhalb des Vektorobjekts selbst gespeichert werden
- drop-in replacement für `std::vector`

- Wenn `std::inplace_vector`? (P0843R8)
  - Speicherzuweisung ist nicht möglich
  - Die Speicherzuweisung führt zu einer inakzeptablen Beeinträchtigung der Performanz
  - `std::array` ist keine Option, z. B. wenn nicht default konstruierbare Objekte gespeichert werden müssen
  - innerhalb von `constexpr`-Funktionen ist ein Array mit dynamischer Größenänderung erforderlich

Unterstützung für Lineare Algebra

`<linalg>` ist eine freie Schnittstelle für lineare Algebra, die auf BLAS basiert.

- BLAS: **B**asic **Li**near **A**lgebra **S**ubprograms ist eine Spezifikation, die eine Reihe von Low-Level-Routinen für die Durchführung gängiger linearer Algebra-Operationen vorschreibt
  - Vektoraddition
  - skalare Multiplikation
  - Linearkombinationen
  - Matrix-Multiplikation

- Diese Operationen sind de facto die grundlegenden Standardroutinen für lineare Algebra-Bibliotheken.

std::submdspan

std::submdspan

```
template<class T, class E, class L, class A,
         class ... SliceArgs)
auto submdspan(mdspan<T,E,L,A> x, SliceArgs ... args);
```

```
int* ptr = ...;
int N = ...;
mdspan a(ptr, N);

// subspan of a single element
auto a_sub1 = submdspan(a, 1);
static_assert(decltype(a_sub1)::rank() == 0);
assert(&a_sub1() == &a(1));

// subrange
auto a_sub2 = submdspan(a, tuple{1, 4});
static_assert(decltype(a_sub2)::rank() == 1);
assert(&a_sub2(0) == &a(1));
assert(a_sub2.extent(0) == 3);
```

```
// subrange with stride
auto a_sub3 = submdspan(a, strided_slice{1, 7, 2});
static_assert(decltype(a_sub3)::rank() == 1);
assert(&a_sub3(0) == &a(1));
assert(&a_sub3(3) == &a(7));
assert(a_sub3.extent(0) == 4);

// full range
auto a_sub4 = submdspan(a, full_extent);
static_assert(decltype(a_sub4)::rank() == 1);
assert(a_sub4(0) == a(0));
assert(a_sub4.extent(0) == a.extent(0));
```

Unterstützung bei der Fehlersuche

C++26 hat drei Funktionen für die Fehlersuche.

- `std::breakpoint`: Hält das laufende Programm an, wenn es aufgerufen wird, und übergibt die Kontrolle an den Debugger
- `std::breakpoint_if_debugging`: ruft `std::breakpoint` auf, wenn `std::is_debugger_present true` zurückgibt
- `std::is_debugger_present`: prüft, ob ein Programm unter der Kontrolle eines Debuggers läuft

C++26

Kernsprache

- ☐ Reflektion
- ☐ Contracts
- ☐ Platzhalter
- ☐ Template-Verbesserungen
- ☐ `delete` mit Grund

Bibliothek

- ☐ `std::inplace_vector`
- ☐ Unterstützung der linearen Algebra
- ☐ `std::submdspan`
- ☐ Unterstützung bei der Fehlersuche

Concurrency

- ■ `std::execution`
- ■ Data-Parallel Typen (SIMD)

`std::execution`

`std::execution` bietet "*ein Standard-C++-Framework für die Verwaltung der asynchronen Ausführung auf generischen Ausführungsressourcen*". (P2300R10)

- `std::execution`
  - früher bekannt als Executors oder Sender/Empfänger
  - stdexec ist die Referenzimplementierung dieses Proposals
  - Verfügt über drei zentrale Abstraktionen: Scheduler, Sender und Empfänger sowie eine Reihe von anpassbaren asynchronen Algorithmen.

stdexec.cpp

std::execution

Das Programm "Hello world" des Proposals P2300R10.

```
using namespace std::execution;

scheduler auto sch = thread_pool.scheduler();                          // 1

sender auto begin = schedule(sch);                                     // 2
sender auto hi = then(begin, []{                                       // 3
    std::cout << "Hello world! Have an int.";                          // 3
    return 13;                                                         // 3
});                                                                    // 3
sender auto add_42 = then(hi, [](int arg) { return arg + 42; });       // 4

auto [i] = this_thread::sync_wait(add_42).value();
```

HelloWorld.cpp

std::execution

- Ressourcen für die Ausführung
  - den Ort der Ausführung
  - brauchen keine Darstellung im Code

- Scheduler
  - die Ausführungsressource darstellen
  - Das Scheduler Concept wird durch einen einzigen Sendealgorithmus definiert: `schedule`.
  - Der Algorithmus `schedule` gibt einen Sender zurück, der auf einer vom Scheduler bestimmten Ausführungsressource fertiggestellt wird.

```
execution::scheduler auto sch = thread_pool.scheduler();
execution::sender auto snd = execution::schedule(sch);
// snd is a sender (see below) describing the creation of a new execution resource
// on the execution resource associated with sch
```

std::execution

- Sender beschreiben Arbeit
  - einige Werte zu senden, wenn ein mit diesem Sender verbundener Empfänger diese Werte schließlich erhalten wird

- Empfänger stoppt den Arbeitsablauf
  - er unterstützt drei Kanäle: Wert, Error, Stop

```
execution::scheduler auto sch = thread_pool.scheduler();
execution::sender auto snd = execution::schedule(sch);
execution::sender auto cont = execution::then(snd, []{
    std::fstream file{ "result.txt" };
    file << compute_result;
});

this_thread::sync_wait(cont);
// at this point, cont has completed execution
```

std::execution

- Sender Fabriken
  - execution::scheduler
  - execution::just
  - execution::just_error
  - execution::just_stopped
  - execution::read_env
- Sender Empfänger
  - this_thread::sync_wait
- Sender Adapter
  - execution::continues_on
  - execution::then
  - execution::upon_*
  - execution::let_*
  - execution::starts_on
  - execution::into_variant
  - execution::stopped_as_optional
  - execution::stopped_as_error
  - execution::bulk
  - execution::split
  - execution::when_all

Data-Parallel Typen (SIMD)

```
const int SIZE= 8;
int vec[]={1, 2 , 3, 4, 5, 6, 7, 8};
int res[SIZE]={0,};

int main(){
  for (int i= 0; i < SIZE; ++i){
    res[i]= vec[i] + 5;
  }
}
```

Nicht vektorisiert

```
movslq  -8(%rbp), %rax
movl    vec(,%rax,4), %ecx
addl    $5, %ecx
movslq  -8(%rbp), %rax
movl    %ecx, res(,%rax,4)
```

Vektorisiert

```
movdqa  .LCPI0_0(%rip), %xmm0   # xmm0 = [5,5,5,5]
movdqa  vec(%rip), %xmm1
paddd   %xmm0, %xmm1
movdqa  %xmm1, res(%rip)
paddd   vec+16(%rip), %xmm0
movdqa  %xmm0, res+16(%rip)
xorl    %eax, %eax
```

Data-Parallel Typen (SIMD)

- **Vektorisierbaren Typen:** Standard-Ganzzahltypen, Zeichentypen und die Typen `float` und `double`

- Das Klassen-Template `simd`

```
template< class T, class Abi = simd_abi::compatible >
class simd;
```

- Die Aliase `native_simd` und `fixed_size_simd`

```
template< class T, int N >
using fixed_size_simd = std::datapar::simd<T, std::datapar::simd_abi::fixed_size<N>>;

template< class T >
using native_simd = std::datapar::simd<T, std::datapar::simd_abi::native<T>>;
```

Data-Parallel Typen (SIMD)

Data-Parallel Typen

- besitzen vektorspezifische Operationen (Reduktion, Reduktion mit Maske).
- können auf die Funktionen in <cmath> angewandt werden (grundlegende, exponentielle, potenz, trigeometrische, hyperbolische oder gamma Funktionen).

Syntaktischer Zucker

```
void f(std::vector<float>& data) {
    std::for_each(std::execution::simd, data.begin(), data.end(), [](auto& v) {
        v = std::sin(v);
    });
}
```

simd.cpp

C++26

Kernsprache

- [ ] Reflektion
- [ ] Contracts
- [ ] Platzhalter
- [ ] Template-Verbesserungen
- [ ] `delete` mit Grund

Bibliothek

- [ ] `std::inplace_vector`
- [ ] Unterstützung der linearen Algebra
- [ ] `std::submdspan`
- [ ] Unterstützung bei der Fehlersuche

Concurrency

- [ ] `std::execution`
- [ ] Data-Parallel Typen (SIMD)

Blog: www.ModernesCpp.com
Mentoring: www.ModernesCpp.org

Rainer Grimm

Schulung, Mentoring und
Technologieberatung