Document number: P2819R1
Date: 2023-07-13
Project: Programming Language C++
Audience: LEWG, LWG
Reply-to: Michael Florian Hava[1] <mfh.cpp@gmail.com>
Christoph Hofer[2] <chofer.cpp@gmail.com>

Add tuple protocol to complex

Abstract

This paper proposes amending `complex` with the tuple protocol, enabling structured binding and easy referential access.

Tony Table

Before	Proposed
`complex<double> c{…};`	`complex<double> c{…};`
`auto & [r, i]{reinterpret_cast<double(&)[2]>(c)};`	`auto & [r, i]{c};`
`template<typename T>`	`template<typename T>`
`constexpr`	`constexpr`
`auto swap_parts(complex<T> c) -> complex<T> {`	`auto swap_parts(complex<T> c) -> complex<T> {`
`  if not consteval {`	
`    auto & [r, i]{reinterpret_cast<double(&)[2]>(c)};`	`  auto & [r, i]{c};`
`    swap(r, i);`	`  swap(r, i);`
`  } else {`	
`    //reinterpret_cast is ill-formed in constexpr…`	
`    const auto tmp{c.real()};`	
`    c.real(c.imag());`	
`    c.imag(tmp);`	
`  }`	
`  return c;`	`  return c;`
`}`	`}`
`vector<complex<double>> v{ … };`	`vector<complex<double>> v{ … };`
`auto reals{v | views::transform([](auto c) {`	`auto reals{v | views::elements<0>`
`                 return c.real();`	`             | ranges::to<vector>()};`
`               }`	
`             | ranges::to<vector>()};`	
`auto imags{v | views::transform([](auto c) {`	`auto imags{v | views::elements<1>`
`                 return c.imag();`	`             | ranges::to<vector>()};`
`               }`	
`             | ranges::to<vector>()};`	
`complex<double> c{…};`	`complex<double> c{…};`
`//interaction with pattern matching proposal P1371R3`	`//interaction with pattern matching proposal P1371R3`
`inspect(reinterpret_cast<double(&)[2]>(c)) {`	`inspect(c) {`
`  [0, 0] => { cout << "on origin"; }`	`  [0, 0] => { cout << "on origin"; }`
`  [0, i] => { cout << "on imaginary axis"; }`	`  [0, i] => { cout << "on imaginary axis"; }`
`  [r, 0] => { cout << "on real axis"; }`	`  [r, 0] => { cout << "on real axis"; }`
`  [r, i] => { cout << r << ", " << i; }`	`  [r, i] => { cout << r << ", " << i; }`
`};`	`};`
`//interaction with pattern matching proposal P2392R2`	`//interaction with pattern matching proposal P2392R2`
`inspect(reinterpret_cast<double(&)[2]>(c)) {`	`inspect(c) {`
`  is [0, 0]   => cout << "on origin";`	`  is [0, 0]   => cout << "on origin";`
`  is [0, _]   => cout << "on imaginary axis";`	`  is [0, _]   => cout << "on imaginary axis";`
`  is [_, 0]   => cout << "on real axis";`	`  is [_, 0]   => cout << "on real axis";`
`  [r, i] is _ => cout << r << ", " << i;`	`  [r, i] is _ => cout << r << ", " << i;`
`}`	`}`

[1] RISC Software GmbH, Softwarepark 32a, 4232 Hagenberg, Austria, michael.hava@risc-software.at

[2] RISC Software GmbH, Softwarepark 32a, 4232 Hagenberg, Austria, christoph.hofer@risc-software.at

Revisions

R0: Initial version

R1: Changes after LEWG review on 2023-06-12:

- Made `get` overloads hidden friends.
- Extending *tuple-like* concept to support tuple-based range algorithms.
- Amended proposed wording with entry to Annex C.

Motivation

Mathematically the set of complex numbers $\mathbb{C}$ is isomorphic to $\mathbb{R}^2$ as a vector space with the isomorphism $\Phi: \mathbb{C} \rightarrow \mathbb{R}^2$ such that $\Phi(a+bi) = (a,b)$. Therefore, complex numbers can be identified with tuples and should possess the same characteristics, which is covered by the tuple protocol.

Complex numbers can equivalently be represented in cartesian coordinates `(a,b)` as well as in polar coordinates `(r,θ)` using radius `r` and angle `θ`. However, alternative representations of complex numbers such as polar coordinates `(r,θ)` are prohibited by the requirement of matching C's `_Complex` *`floating-point`* feature.

As the respective getters do not expose referential access (changing them to do so would result in an ABI-break), the only way to get a reference to the real and imaginary parts of a `complex` is by performing a `reinterpret_cast` (mandated to be valid, see [complex.numbers.general]), which is not valid in a `constexpr` context. Supporting the tuple protocol enables structured binding and referential access to the components of a complex number in a `constexpr` compatible way.

Lastly, the current pattern matching proposals ([P1371R3] and [P2392R2]) allow inspection of *tuple-like* objects, the proposed changes make `complex` *tuple-like*.

Design Space

The tuple protocol traits (`tuple_size<T>` and `tuple_element<I, T>`) are partially specialized for `complex<U>` and four hidden friend function overloads of `get` are provided. Additionally, the exposition-only *`tuple-like`* concept is amended, enabling support for range algorithms like `std::views::elements`.

Impact on the Standard

This proposal is a library extension, that changes the meaning of *`tuple-like`*`<complex<T>>`.

Implementation Experience

The proposed design has been implemented at https://github.com/MFHava/STL/tree/P2819.

Proposed Wording

Wording is relative to [N4950]. Additions are presented like this, removals like ~~this~~ and drafting notes like **this**.

[version.syn]

```
#define __cpp_lib_complex_tuple YYYYMML //also in <complex>

#define __cpp_lib_tuple_like ~~202207L~~YYYYMML //also in <utility>, <tuple>, <map>, <unordered_map>
```

[DRAFTING NOTE: Adjust the placeholder value as needed to denote the proposal's date of adoption.]

[tuple.like]

??.??.? Concept *tuple-like* **[tuple.like]**

```
template<class T>
  concept tuple-like = see below; //exposition only
```

1 A type T models and satisfies the exposition-only concept *tuple-like* if `remove_cvref_t<T>` is a specialization of `array`, `complex`, `pair`, `tuple`, or `ranges::subrange`.

[complex.numbers]

??.??.? Header <complex> synopsis **[complex.syn]**

```
namespace std {
…
  // [complex.transcendentals], transcendentals
…
  template<class T> complex<T> tanh (const complex<T>&);

  // [complex.tuple], tuple interface
  template<class T> struct tuple_size;
  template<size_t I, class T> struct tuple_element;
  template<class T>
    struct tuple_size<complex<T>>;
  template<size_t I, class T>
    struct tuple_element<I, complex<T>>;

  // [complex.literals], complex literals
…
}
```

??.??.? Class template complex **[complex]**

```
namespace std {
  template<class T> class complex {
  public:
    using value_type = T;

    constexpr complex(const T& re = T(), const T& im = T());
    constexpr complex(const complex&) = default;
    template<class X> constexpr explicit(see below) complex(const complex<X>&);

    constexpr T real() const;
    constexpr void real(T);
    constexpr T imag() const;
    constexpr void imag(T);

    constexpr complex& operator= (const T&);
    constexpr complex& operator+=(const T&);
    constexpr complex& operator-=(const T&);
    constexpr complex& operator*=(const T&);
    constexpr complex& operator/=(const T&);

    constexpr complex& operator=(const complex&);
    template<class X> constexpr complex& operator= (const complex<X>&);
    template<class X> constexpr complex& operator+=(const complex<X>&);
    template<class X> constexpr complex& operator-=(const complex<X>&);
    template<class X> constexpr complex& operator*=(const complex<X>&);
    template<class X> constexpr complex& operator/=(const complex<X>&);

    template<size_t I>
      friend constexpr T& get(complex&) noexcept;
    template<size_t I>
      friend constexpr T&& get(complex&&) noexcept;
    template<size_t I>
      friend constexpr const T& get(const complex&) noexcept;
    template<size_t I>
      friend constexpr const T&& get(const complex&&) noexcept;
  };
}
```

…

??.??.? Non-member operations **[complex.ops]**

…

```
template<class T, class charT, class traits>
  basic_ostream<charT, traits>& operator<<(basic_ostream<charT, traits>& o, const complex<T>& x);
```

14 *Effects*: Inserts the complex number x onto the stream o as if it were implemented as follows:

```
basic_ostringstream<charT, traits> s;
s.flags(o.flags());
s.imbue(o.getloc());
s.precision(o.precision());
s << '(' << x.real() << ',' << x.imag() << ')';
return o << s.str();
```

15 [*Note 1*: In a locale in which comma is used as a decimal point character, the use of comma as a field separator can be ambiguous. Inserting `showpoint` into the output stream forces all outputs to show an explicit decimal point character; as a result, all inserted sequences of complex numbers can be extracted unambiguously. — *end note*]

```
template<size_t I>
  friend constexpr T& get(complex& x) noexcept;
template<size_t I>
  friend constexpr const T& get(const complex& x) noexcept;
template<size_t I>
  friend constexpr T&& get(complex&& x) noexcept;
template<size_t I>
  friend constexpr const T&& get(const complex&& x) noexcept;
```

16 *Mandates*: `I < 2` is `true`.

17 *Returns*: A reference to the real part of x if `I == 0` is `true`, otherwise a reference to the imaginary part of x.

??.??.? Value operations **[complex.value.ops]**

…

??.??.? Transcendentals **[complex.trancendentals]**

…

```
template<class T> complex<T> tanh(const complex<T>& x);
```

27 *Returns*: The complex hyperbolic tangent of x.

??.??.? Tuple interface **[complex.tuple]**

```
template<class T>
  struct tuple_size<complex<T>> : integral_constant<size_t, 2> {};

template<size_t I, class T>
  struct tuple_element<I, complex<T>> {
    using type = T;
  };
```

1 *Mandates*: `I < 2` is `true`.

??.??.? Additional overloads **[cmplx.over]**

[diff]

C.? C++ and ISO C++2023 **[diff.cpp23]**

1 Subclause [diff.cpp23] lists the differences between C++ and ISO C++ 2023 (ISO/IEC 14882:2023, *Programming Languages — C++*), by the chapters of this document.

C.?.? [complex]: Complex numbers library **[diff.cpp23.complex]**

1 **Affected subclause: [complex.numbers]**
Change: Enabling structured binding for complex numbers.
Rationale: Improve usability of complex numbers.
Effect on original feature: Valid C++ 2023 code may change meaning. For example:

```
  template<typename T>
  class is_tuple_like {
    template<typename U> static auto test(U *) -> decltype(tuple_size<U>::value, 0);
    template<typename>   static void test(...);
  public:
    static constexpr const bool value = !std::is_void<decltype(test<T>(nullptr))>::value;
  };

  bool value = is_tuple_like<complex<float>>::value; //value is true; previously was false
```

Acknowledgements

Thanks to RISC Software GmbH for supporting this work.