`std::iterator_traits`

Типы-члены для итераторов

Каждый класс итератора из стандартной библиотеки определяет типы-члены:

- `difference_type` (в чем измерять расстояние между итераторами)
- `value_type` (тип элемента под итератором)
- `pointer` (что возвращает `->`)
- `reference` (что возвращает `*`)
- `iterator_category` (категория итератора)

Категория итератора может быть одним из следующих пустых типов:
`input_iterator_tag`, `forward_iterator_tag` (наследник `input_iterator_tag`),
`bidirectional_iterator_tag` (наследник `forward_iterator_tag`),
`random_access_iterator_tag` (наследник `bidirectional_iterator_tag`),
`contiguous_iterator_tag` (наследник `random_access_iterator_tag`)

Типы-члены для итераторов

Как найти расстояние между итераторами?

```
template <class Iterator>
??? Distance(Iterator begin, Iterator end) {
  return ???;
}
```

Типы-члены для итераторов

```
template <class Iterator>
typename Iterator::difference_type Distance(Iterator begin, Iterator end) {
  if (Iterator is RandomAccessIterator) {
    return end - begin;
  }
  typename Iterator::difference_type distance = 0;
  for (; begin != end; ++begin) {
    ++distance;
  }
  return distance;
}
```

Типы-члены для итераторов

```
template <class Iterator>
typename Iterator::difference_type
DistanceImpl(Iterator begin, Iterator end, std::input_iterator_tag) {
  typename Iterator::difference_type distance = 0;
  for (; begin != end; ++begin) {
    ++distance;
  }
  return distance;
}

template <class Iterator>
typename Iterator::difference_type
DistanceImpl(Iterator begin, Iterator end, std::random_access_iterator_tag) {
  return end - begin;
}

template <class Iterator>
typename Iterator::difference_type Distance(Iterator begin, Iterator end) {
  return DistanceImpl(begin, end, typename Iterator::iterator_category{});
}
```

Типы-члены для итераторов

В современном C++ (C++17) все проще

```
template <class Iterator>
typename Iterator::difference_type Distance(Iterator begin, Iterator end) {
  using category = typename Iterator::iterator_category;
  if constexpr (std::is_base_of_v<std::random_access_iterator_tag, category>) {
      return end - begin;
  }
  typename Iterator::difference_type distance = 0;
  for (; begin != end; ++begin) {
    ++distance;
  }
  return distance;
}
```

Указатель

Является ли указатель итератором?

Указатель

Является ли указатель итератором? - Да.

Какой категории итераторов принадлежит указатель?

Указатель

Является ли указатель итератором? - Да.

Какой категории итераторов принадлежит указатель? - Contiguous.

Можно ли передать указатель в предыдущую функцию `Distance`?

Указатель

Является ли указатель итератором? - Да.

Какой категории итераторов принадлежит указатель? - Random access (contiguous).

Можно ли передать указатель в предыдущую функцию `Distance`? - Нет.

Как так?

`std::iterator_traits`

Более унифицированным способом получения свойств итераторов является использование шаблона `std::iterator_traits`:

```
template <class Iterator>
typename std::iterator_traits<Iterator>::difference_type
Distance(Iterator begin, Iterator end)
```

std::iterator_traits

Возможная реализация

```
template <class Iterator>
class iterator_traits {
    using difference_type   = typename Iterator::difference_type;
    using value_type        = typename Iterator::value_type;
    using reference         = typename Iterator::reference;
    using pointer           = typename Iterator::pointer;
    using iterator_category = typename Iterator::iterator_category;
};

template <class T>
class iterator_traits<T*> {  // специализация для указателей
    using difference_type   = std::ptrdiff_t;
    using value_type        = T;
    using reference         = T&;
    using pointer           = T*;
    using iterator_category = std::contiguous_iterator_tag;
};
```

`std::iterator_traits`

`std::iterator_traits` позволяет использовать указатели в качестве итераторов для стандартных контенеров, в частности итераторы `std::vector` и `std::array` могут быть определены следующим образом:

```
template <class T>
class vector {
    // ...
    using iterator = T*;
    using const_iterator = const T*;
    // ...
};
```