РЕАЛИЗАЦИЯ ОПЕРАТОРОВ СРАВНЕНИЯ ДЛЯ ТИПА INTVECTOR

ОБЪЯВЛЕНИЕ ТИПА INTVECTOR

```
class IntVector {
private:
  int *arr;
  size_t size_;
  size_t capacity_;
public:
  /* constructors */
  IntVector();
  IntVector(const std::initializer_list<int> &init_list);

  /* Rule of 5 */
  IntVector(const IntVector &v_copy);
  IntVector(IntVector &&v_copy);
  IntVector &operator=(const IntVector &vec);
  IntVector &operator=(IntVector &&vec);
  ~IntVector();

  size_t size() const;
  size_t capacity() const;
  int *begin() const;
  int *end() const;
  int &operator[](const size_t index) noexcept;
};
```

INTVECTOR: RULE OF FIVE

В нашем классе IntVector мы сами управляем ресурсами, такими как выделение и освобождение памяти.
Соответственно, следуя правилу пяти, мы должны написать свои специальные методы:

Деструктор
- ~IntVector();

Конструктор копирования
- IntVector(const IntVector &v_copy);

Конструктор перемещения
- IntVector(IntVector &&v_copy);

Оператор присваивания копированием
- IntVector &operator=(const IntVector &vec);

Оператор присваивания перемещением
- IntVector &operator=(IntVector &&vec);

ОПРЕДЕЛЕНИЕ ОПЕРАТОРОВ СРАВНЕНИЯ ДЛЯ ТИПА INTVECTOR

Определим операторы сравнения “равно” и “меньше”, на основе которых будут реализованы все остальные операторы сравнения.

```
inline bool operator==(const IntVector &lhs, const IntVector &rhs) {
  return (
      lhs.size() == rhs.size() &&
      std::equal(lhs.begin(), lhs.end(), rhs.begin()));
}
```

```
inline bool operator<(const IntVector &lhs, const IntVector &rhs) {
  return std::lexicographical_compare(
      lhs.begin(), lhs.end(), rhs.begin(), rhs.end());
}
```

ОПЕРАТОР==

Два объекта IntVector равны, если они содержат одинаковое количество элементов и соответствующие элементы имеют одинаковые значения.
В противном случае они не равны.

```
inline bool operator==(const IntVector &lhs, const IntVector &rhs) {
  return (
      lhs.size() == rhs.size() &&
      std::equal(lhs.begin(), lhs.end(), rhs.begin()));
}
```

- Метод std::equal сравнивает два диапазона поэлементно на признак равенства.

std::equal - cppreference.com

STD::EQUAL

Defined in header <algorithm>

std::equal сравнивает два диапазона поэлементно на признак равенства, либо по заданному предикату.

- Return value:
  Если два диапазона поэлементно равны то возвращается: true.
  В ином случае: false.
- Time complexity: O(n)

[std::equal - cppreference.com]

Possible implementation

First version

```
template<class InputIt1, class InputIt2>
bool equal(InputIt1 first1, InputIt1 last1,
           InputIt2 first2)
{
    for (; first1 != last1; ++first1, ++first2) {
        if (!(*first1 == *first2)) {
            return false;
        }
    }
    return true;
}
```

Second version

```
template<class InputIt1, class InputIt2, class BinaryPredicate>
bool equal(InputIt1 first1, InputIt1 last1,
           InputIt2 first2, BinaryPredicate p)
{
    for (; first1 != last1; ++first1, ++first2) {
        if (!p(*first1, *first2)) {
            return false;
        }
    }
    return true;
}
```

Примечание: **std::equal** не следует использовать для сравнения диапазонов, сформированных итераторами из std::unordered_set, std::unordered_multiset, std::unordered_map или std::unordered_multimap, потому что порядок, в котором элементы хранятся в этих контейнерах, может отличаться, даже если два контейнера хранят одни и те же элементы.

ОПЕРАТОР<

Проверяет, меньше ли объект слева от оператора, чем объект справа от оператора.

```
inline bool operator<(const IntVector &lhs, const IntVector &rhs) {
  return std::lexicographical_compare(
      lhs.begin(), lhs.end(), rhs.begin(), rhs.end());
}
```

- Метод lexicographical_compare() возвращает true, если первый диапазон лексикографически меньше второго; в противном случае возвращается false.

std::lexicographical_compare - cppreference.com

STD::LEXICOGRAPHICAL_COMPARE

Defined in header <algorithm>

std::lexicographical_compare возвращает true, если первый диапазон лексикографически меньше второго, в противном случае false.

Time complexity: O(n)

[std::lexicographical_compare - cppreference.com]

Possible implementation

lexicographical_compare (1)

```
template<class InputIt1, class InputIt2>
bool lexicographical_compare(InputIt1 first1, InputIt1 last1,
                             InputIt2 first2, InputIt2 last2)
{
    for (; (first1 != last1) && (first2 != last2); ++first1, (void) ++first2)
    {
        if (*first1 < *first2)
            return true;
        if (*first2 < *first1)
            return false;
    }

    return (first1 == last1) && (first2 != last2);
}
```

lexicographical_compare (3)

```
template<class InputIt1, class InputIt2, class Compare>
bool lexicographical_compare(InputIt1 first1, InputIt1 last1,
                             InputIt2 first2, InputIt2 last2, Compare comp)
{
    for (; (first1 != last1) && (first2 != last2); ++first1, (void) ++first2)
    {
        if (comp(*first1, *first2))
            return true;
        if (comp(*first2, *first1))
            return false;
    }

    return (first1 == last1) && (first2 != last2);
}
```

ЛЕКСИГРАФИЧЕСКОЕ СРАВНЕНИЕ

Лексикографический порядок — отношение линейного порядка на множестве слов над некоторым упорядоченным алфавитом

Лексикографическое сравнение - это способ сравнения строк, чисел или других объектов на основе лексикографического порядка, который определяет порядок следования элементов в словаре.
При лексикографическом сравнении два объекта сначала сравниваются по первому символу или цифре: если они не равны, то возвращается результат этого сравнения; если они равны, то сравниваются следующие символы или цифры и так далее до тех пор, пока получится результат.

ОПЕРАТОРЫ > <= >= !=

```
inline bool operator!=(const IntVector &lhs, const IntVector &rhs) {
  return !(lhs == rhs);
}
```

```
inline bool operator>(const IntVector &lhs, const IntVector &rhs) {
  return (rhs < lhs);
}
```

```
inline bool operator<=(const IntVector &lhs, const IntVector &rhs) {
  return !(rhs < lhs);
}
```

```
inline bool operator>=(const IntVector &lhs, const IntVector &rhs) {
  return !(lhs < rhs);
}
```

Полная реализация с тестами :
DSA/IntVector at main · coldysplash/DSA (github.com)

Q&A