Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления
Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №10
«Реализация итераторов на языке C++»
по курсу: «Языки и методы программирования»

Выполнил:
Студент группы ИУ9-21Б
Старовойтов А. И.

Проверил:
Посевин Д. П.

Москва, 2022

Цели

Данная работа предназначена для приобретения навыков разработки контейнерных классов с итераторам.

Задачи

Согласно выбранному из таблицы описанию требуется составить контейнерный класс (или шаблон контейнерного класса) и итератор для перебора содержимого объектов этого класса. Если в варианте задания говорится о константном итераторе, значит итератор не должен поддерживать изменение содержимого объектов контейнерного класса.

Вариант 46.

Последовательность множеств с константным двунаправленным итератором по пересечениям соседних множеств последовательности. Обращение к элементам последовательности должно осуществляется с помощью перегруженной операции «[ ]».

Решение

set_sequence.hpp

```
#ifndef LAB10_SET_SEQUENCE_HPP_
#define LAB10_SET_SEQUENCE_HPP_

#include <algorithm>
#include <iterator>
#include <ranges>
#include <set>
#include <vector>

namespace lab10 {
template <typename T, typename Container =
 ↪  std::vector<std::set<T>>>
class SetSeq {
public:
    using container_type = Container;
```

```
    using value_type = typename Container::value_type;
    using size_type = typename Container::size_type;
    using const_reference = typename
     ↪  Container::const_reference;
    using const_iterator = typename
     ↪  Container::const_iterator;

private:
    container_type seq = container_type();
    container_type pairsIntersecions = container_type();

private:
    void calcIntersections()
    {
        std::ranges::transform(seq |
         ↪  std::ranges::views::take(seq.size() - 1),
            seq | std::ranges::views::drop(1),
            std::back_inserter(pairsIntersecions),
            [](const auto& a, const auto& b) {
                value_type res;
                std::ranges::set_intersection(a, b,
                 ↪  std::inserter(res, res.end()));
                return res;
            });
    }

public:
    SetSeq() = default;
    explicit SetSeq(const container_type& cont)
        : seq(cont)
    {
        calcIntersections();
    }
    explicit SetSeq(container_type&& cont)
        : seq(std::move(cont))
    {
        calcIntersections();
    }
    SetSeq(const SetSeq& other) = default;
    SetSeq(SetSeq&& other) = default;
```

```
    SetSeq& operator=(const SetSeq& other) = default;
    SetSeq& operator=(SetSeq&& other) = default;
    SetSeq(std::initializer_list<value_type> init)
        : seq(init)
    {
        calcIntersections();
    }

    auto size() const noexcept
    {
        return seq.size();
    }

    const_iterator cbegin() noexcept
    {
        return seq.cbegin();
    }
    const_iterator cend() noexcept
    {
        return seq.cend();
    }

    const_iterator cbeginIntersections() const noexcept
    {
        return pairsIntersecions.cbegin();
    }
    const_iterator cendIntersections() const noexcept
    {
        return pairsIntersecions.cend();
    }

    const_reference operator[](size_type pos) const
    {
        return seq[pos];
    }
};

}; // lab10

#endif // LAB10_SET_SEQUENCE_HPP_
```

main.cpp

```
#include "set_sequence.hpp"

#include <iostream>

int main()
{
    lab10::SetSeq<int> test { { 1, 2, 3 }, { 3, 4, 5 }, { 4
 ↪  }, { 1, 2, 3 }, { 1 } };
    for (auto it = test.cbeginIntersections(); it !=
     ↪  test.cendIntersections(); ++it) {
        for (auto& el : *it) {
            std::cout << el << ' ';
        }
        std::cout << '\n';
    }
    std::cout << "\ntest[1]: ";
    for (auto& el : test[1]) {
        std::cout << el << ' ';
    }
    std::cout << '\n';
    return 0;
}
```

Пример вывода

```
3
4

1

test[1]: 3 4 5
```