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 №9
«Перегрузка операций»
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 изучения возможностей языка C++, обеспечивающих применение знаков операций к объектам пользовательских типов.

Задачи

Вариант 27.

SparseArray<T> – разреженный массив, отображающий неотрицательные целые числа в значения типа T. Массив должен быть реализован через хэш-таблицу. Требование к типу T: наличие конструктора по умолчанию (т.к. разреженный массив должен уметь создавать значения типа T). Операции, которые должны быть перегружены для SparseArray<T>:

1. «[ ]» – получение ссылки на i-тый элемент массива;
2. «( )» – формирование подмассива, содержащего элементы с индексами из указанного диапазона (принимает в качестве параметров границы диапазона, возвращает новый SparseArray<T>).
3. «==», «!=».

Решение

sparce_array.hpp

```
#ifndef LAB9_SPARCE_ARRAY_HPP_
#define LAB9_SPARCE_ARRAY_HPP_

#include <concepts>
#include <iostream>
#include <ranges>
#include <stdexcept>
#include <unordered_map>

namespace lab9 {

template <std::default_initializable T>
```

```cpp
class SparceArray;

template <std::default_initializable T>
bool operator==(const SparceArray<T>& lhs, const
 ↪  SparceArray<T>& rhs);

template <std::default_initializable T>
std::ostream& operator<<(std::ostream& out, const
 ↪  SparceArray<T>& arr);

template <std::default_initializable T>
class SparceArray {
public:
    using key_t = std::size_t;
    using container_t = std::unordered_map<key_t, T>;

private:
    container_t map;

public:
    SparceArray() = default;
    SparceArray(const container_t&);

    T& operator[](std::size_t idx);
    const T& operator[](std::size_t idx) const;
    SparceArray operator()(key_t l, key_t r) const;

    friend bool operator==<>(const SparceArray& lhs, const
     ↪  SparceArray& rhs);
    friend std::ostream& operator<<<>(std::ostream& out,
     ↪  const SparceArray& arr);

    void output();
};

template <std::default_initializable T>
using key_t = typename SparceArray<T>::key_t;

template <std::default_initializable T>
using container_t = typename SparceArray<T>::container_t;
```

```
template <std::default_initializable T>
SparceArray<T>::SparceArray(const container_t& inputMap)
    : map(inputMap)
{
}

template <std::default_initializable T>
T& SparceArray<T>::operator[](std::size_t idx)
{
    return map[idx];
}

template <std::default_initializable T>
const T& SparceArray<T>::operator[](std::size_t idx) const
{
    return map[idx];
}

template <std::default_initializable T>
SparceArray<T> SparceArray<T>::operator()(key_t l, key_t r)
 ↪  const
{
    if (l > r) {
        throw
         ↪  std::range_error("SparceArray<T>::operator()");
    }
    auto elems = map | std::views::filter([&l, &r](const
     ↪  auto& el) -> bool { return el.first >= l && el.first
     ↪  <= r; });
    return container_t(std::ranges::begin(elems),
     ↪  std::ranges::end(elems));
}

template <std::default_initializable T>
bool operator==(const SparceArray<T>& lhs, const
 ↪  SparceArray<T>& rhs)
{
    return lhs.map == rhs.map;
}
```

```cpp
template <std::default_initializable T>
std::ostream& operator<<(std::ostream& out, const
 ↪  SparceArray<T>& arr)
{
    std::string delimeter = " ";
    out << '{';
    for (auto&& [key, value] : arr.map) {
        out << delimeter << key << ":" << value;
        delimeter = ", ";
    }
    out << " }";
    return out;
}

}; // lab9

#endif // LAB9_SPARCE_ARRAY_HPP_
```

main.cpp

```cpp
#include "sparce_array.hpp"

#include <iostream>

int main()
{
    {
        struct T {
            T() = delete;
        };
        /* constraints not satisfied: default_initializable
         ↪  */
        /* lab9::SparceArray<T> arr; */
    }
    {
        lab9::SparceArray<std::string> arr;
        arr[0] = "abcd";
        arr[2] = "degh";
        arr[4] = "abcd";
```

```
        arr[5] = "abcd";
        arr[6] = "degh";
        arr[7] = "degh";

        auto subarr = arr(3, 6);
        std::cout << arr << '\n'
                  << subarr << '\n';

        /* range_error */
        /* auto subarr2 = arr(6, 5); */

        /* OK */
        auto subarr3 = arr(10, 20);
        std::cout << subarr3 << '\n';

        std::cout << (subarr == arr) << ' ' << (arr != arr)
         ↪  << '\n';
    }
    return 0;
}
```

Пример вывода

```
{ 7:degh, 6:degh, 5:abcd, 4:abcd, 2:degh, 0:abcd }
{ 4:abcd, 5:abcd, 6:degh }
{ }
0 0
```