федеральное государственное бюджетное образовательное учреждение высшего образования
«Вологодский государственный университет»

Институт математики, естественных и компьютернных наук

Кафедра автоматики и вычислительной техники

Отчет по лабораторной работе №5

Дисциплина: «Программирование»

09.03.04	**43.10**		**01**	**2025**
Код направления подготовки/ специальности	Код выпускающей кафедры	Регистрационный номер по журналу	Код формы обучения	Год

Руководитель доц., Кочкин Дмитрий Валерьевич

Выполнил студент Тестов Максим Олегович

Группа, курс 4Б09 РПС-21

Дата сдачи _____

Дата защиты _____

Оценка по защите _____

Вологда 2025

1. Шаблонный класс "множество"

Напишите шаблонный класс Set для представления АТД "множество"на основе рандомизированного двоичного дерева, хеш-таблицы или другой структуры, обеспечивающей требуемую производительность (использование класса std::set не допускается)

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

template <class T>
class Set {

private:

class HashMap {
private:
    static const int SizeHashMap = 200;

    vector<vector<T>> buckets;

    mutable vector<T> cached_values;
    mutable bool cache_valid = false;
    mutable size_t total_count = 0;

public:
    HashMap() : buckets(SizeHashMap), cache_valid(false), total_count(0) {}

    // хеш функция
    int HashIndex(const T& Tvalue) const {
        const unsigned char* Cvalue = reinterpret_cast<const unsigned char*>
        size_t size = sizeof(T);

        unsigned int hash = 0;
        for (size_t i = 0; i < size; ++i) {
            hash = hash * 31 + Cvalue[i];
        }

        return (hash % SizeHashMap);
    }

    // добавление записей в таблицу
```

```
  void InsertValues(const T& value) {
      int bucketIndex = HashIndex(value);

      for (int i = 0; i < buckets[bucketIndex].size(); i++) {
          if (buckets[bucketIndex][i] == value) return;
      }

      buckets[bucketIndex].push_back(value);
      total_count++;
      cache_valid = false;
  }

  // удалить запись
  void DeleteValues(const T& value) {
      int bucketIndex = HashIndex(value);

      for (int i = 0; i < buckets[bucketIndex].size(); i++) {
          if (buckets[bucketIndex][i] == value) {
              buckets[bucketIndex].erase(buckets[bucketIndex].begin() + i)
              total_count--;
              cache_valid = false;
              return;
          }
      }
  }

  // проверка наличия записи
  bool ContainsValues(const T& value) const {
      int bucketIndex = HashIndex(value);

      for (int i = 0; i < buckets[bucketIndex].size(); i++) {
          if (buckets[bucketIndex][i] == value) return true;
      }
      return false;
  }

  // получить все записи
const vector<T>& GetAllValues() const {
  if (cache_valid == false) {
      size_t total_size = 0;
      for (int i = 0; i < buckets.size(); i++) {
          total_size += buckets[i].size();
      }
```

```
        // Обновляем кэш
        cached_values.clear();
        cached_values.reserve(total_size);

        for (int i = 0; i < buckets.size(); i++) {
            for (int j = 0; j < buckets[i].size(); j++) {
                cached_values.push_back(buckets[i][j]);
            }
        }

        cache_valid = true;
    }
    return cached_values;
  }

    size_t GetSize() const {
        return total_count;
    }

};

  HashMap HashTable;

  void insert(const T& value) {
    HashTable.InsertValues(value);
  }

  void remove(const T& value) {
    HashTable.DeleteValues(value);
  }

  bool contains(const T& value) const {
    return HashTable.ContainsValues(value);
  }

  const vector<T>& getterAll() const {
    return HashTable.GetAllValues();
  }

public:

  Set() {}

  Set(const Set &s) {
```

```
  const auto& values = s.getterAll();
  for (const auto& val : values) {
    insert(val);
  }
}

//вставка элемента в множество
Set operator + (const T &t) const {
  Set result = *this;
  result.insert(t);
  return result;
}

//Set<T> result = 42 + Set<T> others
friend Set operator + (const T &t, const Set &s) {
  Set newSet = s;
  newSet.insert(t);
  return newSet;
}

Set &operator += (const T &t) {
  insert(t);
  return *this;
}

//удаление элемента из множества
Set operator - (const T &t) const {
  Set result = *this;
  result.remove(t);
  return result;
}

Set &operator -= (const T &t) {
  remove(t);
  return *this;
}

//проверка наличия элемента в множестве
bool exists(const T &t) const {
  return contains(t);
}

//объединение множеств
Set operator + (const Set &s) const {
```

```
  Set<T> result = *this;
  const auto& values = s.getterAll();
  for (int i = 0; i < (int)values.size(); i++) {
    result.insert(values[i]);
  }
  return result;
}

Set& operator += (const Set &s) {
  const auto& values = s.getterAll();
  for (int i = 0; i < (int)values.size(); i++) {
    insert(values[i]);
  }
  return *this;
}
//разность множеств
Set operator - (const Set &s) const {
  Set<T> result = *this;
  const auto& values = s.getterAll();
  for (int i = 0; i < (int)values.size(); i++) {
    result.remove(values[i]);
  }
  return result;
}

Set& operator -= (const Set &s) {
  const auto& values = s.getterAll();
  for (int i = 0; i < (int)values.size(); i++) {
    remove(values[i]);
  }
  return *this;
}

//операции сравнения
bool operator == (const Set &s) const {
  const auto& v1 = getterAll();
  const auto& v2 = s.getterAll();
  if (v1.size() != v2.size()) return false;

  vector<T> sorted_v1 = v1;
  vector<T> sorted_v2 = v2;
  sort(sorted_v1.begin(), sorted_v1.end());
  sort(sorted_v2.begin(), sorted_v2.end());
  return sorted_v1 == sorted_v2;
```

```
  }

  bool operator != (const Set &s) const {
    return !(*this == s);
  }

  //мощность множества (число элементов)
  int size(void) const {
    return HashTable.GetSize();
  }

  //вывод элементов в порядке возрастания, разделяя их пробелом
  friend ostream& operator << (ostream& os, const Set &s) {
    vector<T> values = s.getterAll();

    if (!values.empty()) {
        sort(values.begin(), values.end());

        os << values[0];
        for (size_t i = 1; i < values.size(); ++i) {
            os << " " << values[i];
        }
    }
    return os;
  }

  //операция присваивания (если нужна)
  Set &operator = (const Set &s) {
    if (this == &s) return *this;
    HashTable = s.HashTable;
    return *this;
  }

  ~Set() {}
};

#include "set-test.h"
```