МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЕТ

по лабораторной работе № 2

по дисциплине «Алгоритмы и структуры данных»

Тема: «Множество как объект»

Студенты гр. 3315 _____ Моторин Д.М. Герасимов Т .Е.

Преподаватель _____ Аббас Саддам

Санкт-Петербург

2024

Цель работы:

Сравнение процедурного и объектно-ориентированного подходов на примере задачи обработки множеств.

Задание (Вариант 8):

Множество, содержащее буквы, имеющиеся во множестве *A*, но не являющиеся общими для *B* и *C*, и все буквы из *D.*В данной работе проведем сравнение между процедурного и объектно-ориентированного подходов на примере задачи обработки множеств.

Результаты эксперимента на примере скриншота:

```
Choose input method:
1. Enter sets manually
2. Generate sets randomly
1
Enter set A: rthvu
Enter set B: utrec
Enter set C: poiqzcv
Enter set D: retqip

Testing SetArray...
SetArray 1 created
Unique characters (SetArray): r t h v u e q i p
Execution time using SetArray: 1.4417e-05 seconds
SetArray 1 destroyed
Set destroyed

Testing SetList...
Custom new for SetList
SetList 1 created
Unique characters (SetList): r t h v u e q i p
Execution time using SetList: 7.541e-06 seconds
SetList 1 destroyed
Set destroyed
Custom delete for SetList

Testing SetBitset...
SetBitset 1 created
Unique characters (SetBitset): e h i p q r t u v
Execution time using SetBitset: 2.792e-06 seconds
SetBitset 1 destroyed
Set destroyed

Testing SetMachineWord...
SetMachineWord 1 created
Unique characters (SetMachineWord): e h i p q r t u v
Execution time using SetMachineWord: 1.125e-06 seconds
SetMachineWord 1 destroyed
Set destroyedProgram ended with exit code: 0
```

Рис. 1 результат с отслеживанием вызовов функций

```
Choose input method:
1. Enter sets manually
2. Generate sets randomly
2
Enter size of set A: 4
Enter size of set B: 10
Enter size of set C: 7
Enter size of set D: 6
Generated sets:
A: mjct
B: zfqiejrxya
C: ckbjxls
D: bprsyt

Testing SetArray...
SetArray 1 created
Unique characters (SetArray): m c t b p r s y
Execution time using SetArray: 1.541e-06 seconds
SetArray 1 destroyed
Set destroyed

Testing SetList...
Custom new for SetList
SetList 1 created
Unique characters (SetList): m c t b p r s y
Execution time using SetList: 9.125e-06 seconds
SetList 1 destroyed
Set destroyed
Custom delete for SetList

Testing SetBitset...
SetBitset 1 created
Unique characters (SetBitset): b c m p r s t y
Execution time using SetBitset: 2.917e-06 seconds
SetBitset 1 destroyed
Set destroyed

Testing SetMachineWord...
SetMachineWord 1 created
Unique characters (SetMachineWord): b c m p r s t y
Execution time using SetMachineWord: 1.083e-06 seconds
SetMachineWord 1 destroyed
Set destroyedProgram ended with exit code: 0
```

Рис. 2 результат с отслеживанием вызовов функций

Результаты эксперимента в таблице процедурного подхода:

Способ хранения множеств	Среднее время (в секундах) по результатам 3 тестов
массив	3.5e-06
Список	5.8e-06
Массив битов	1.6e-06
Машинное слово	1.5e-06

Результаты эксперимента в таблице объектно-ориентированного подхода:

Способ хранения множеств	Среднее время (в секундах) по результатам 3 тестов
массив	5.2e-06
Список	6.6e-06
Массив битов	3.2e-06
Машинное слово	3.0 e-06

Вывод:

В ходе работы был проведен анализ процедурного и объектно-ориентированного подходов на примере обработки множеств. Как и в первой лабораторной работе программа быстрее выполняется при использовании машинного слова для хранения множеств. Выявлен более универсальный, удобный подход. Работа подтвердила преимущество объектно-ориентированного подхода в читаемости кода.

Код программы:

```
#include "SetClasses.h"
#include <iostream>
#include <chrono>
#include <ctime>


using namespace std;
```

```
int main() {
    srand(static_cast<unsigned int>(time(0)));

    string A, B, C, D;
    int choice;

    cout << "Choose input method:\n";
    cout << "1. Enter sets manually\n";
    cout << "2. Generate sets randomly\n";
    cin >> choice;

    if (choice == 1) {
        cout << "Enter set A: ";
        cin >> A;
        cout << "Enter set B: ";
        cin >> B;
        cout << "Enter set C: ";
        cin >> C;
        cout << "Enter set D: ";
        cin >> D;
    }
    else if (choice == 2) {
        int sizeA, sizeB, sizeC, sizeD;
        cout << "Enter size of set A: ";
        cin >> sizeA;
        cout << "Enter size of set B: ";
        cin >> sizeB;
        cout << "Enter size of set C: ";
        cin >> sizeC;
        cout << "Enter size of set D: ";
        cin >> sizeD;

        generateRandomSet(A, sizeA);
        generateRandomSet(B, sizeB);
        generateRandomSet(C, sizeC);
        generateRandomSet(D, sizeD);

        cout << "Generated sets:\n";
        cout << "A: " << A << "\nB: " << B << "\nC: " << C << "\nD: " << D << "\n";
    }
    else {
        cerr << "Invalid choice!" << endl;
        return 1;
    }
```

```
    // Измерение времени выполнения для различных реализаций

    Set* set = nullptr;

    // Вариант с массивом
    cout << "\nTesting SetArray...\n";
    set = new SetArray(); // создаем новый объект с использованием оператора new, который выделяет память
    auto start = chrono::high_resolution_clock::now();// сохраняем текущее время
    set->findUnique(A, B, C, D);// ищем уникальные символы в A B C D
    auto end = chrono::high_resolution_clock::now();// сохраняем конечное время
    double timeArray = chrono::duration<double>(end - start).count();// считаем время выполнения
    cout << "Unique characters (SetArray): ";// выводим уникальные символы
    set->print();// выводим уникальные символы
    cout << "Execution time using SetArray: " << timeArray << " seconds\n"; // выводим время выполнения
    delete set;// освобождаем память выделенную под set

    // Вариант со списком
    cout << "\n\nTesting SetList...\n";
    set = new SetList();
    start = chrono::high_resolution_clock::now();
    set->findUnique(A, B, C, D);
    end = chrono::high_resolution_clock::now();
    double timeList = chrono::duration<double>(end - start).count();
    cout << "Unique characters (SetList): ";
    set->print();
    cout << "Execution time using SetList: " << timeList << " seconds\n";
    delete set;

    // Вариант с битовым набором
    cout << "\n\nTesting SetBitset...\n";
    set = new SetBitset();
    start = chrono::high_resolution_clock::now();
    set->findUnique(A, B, C, D);
    end = chrono::high_resolution_clock::now();
    double timeBitset = chrono::duration<double>(end - start).count();
    cout << "Unique characters (SetBitset): ";
    set->print();
    cout << "Execution time using SetBitset: " << timeBitset << " seconds\n";
    delete set;

    // Вариант с машинным словом
    cout << "\n\nTesting SetMachineWord...\n";
```

```
    set = new SetMachineWord();
    start = chrono::high_resolution_clock::now();
    set->findUnique(A, B, C, D);
    end = chrono::high_resolution_clock::now();
    double timeMachineWord = chrono::duration<double>(end - start).count();
    cout << "Unique characters (SetMachineWord): ";
    set->print();
    cout << "Execution time using SetMachineWord: " << timeMachineWord << "
seconds\n";
    delete set;

    return 0;
}
#define SET_CLASSES_H
#pragma once
#include <iostream>
#include <string>
#include <list>
#include <bitset>
#include <algorithm>
using namespace std;

const int ALPHABET_SIZE = 26; // размер алфавита
const char START_CHAR = 'a';  // начальный символ алфавита

void generateRandomSet(std::string& set, int size);

// Базовый интерфейсный класс для всех реализаций множества
class Set {
public:
    virtual void findUnique(const string& A, const string& B, const string& C, const
string& D) = 0;
    virtual void print() const = 0;
    virtual ~Set() {
        cout << "Set destroyed";
    }
};

// Реализация класса Set для массива (SetArray)
class SetArray : public Set {
private:
    static int counter;  // отслеживаем количество созданных объектов
    int tag;  // Уникальный тег
    char elements[ALPHABET_SIZE];// хранение символов
    int size; // храним текущее количество символов
```

```
public:
    SetArray() : size(0), tag(++counter) { //  Конструктор инициализирует size нулем и увеличивает counter, присваивая его значение tag
        std::fill(std::begin(elements), std::end(elements), '\0'); // заполняем пустыми символами
        cout << "SetArray " << tag << " created\n";
    }

    ~SetArray() { // деструктор
        cout << "SetArray " << tag << " destroyed\n"; // выводим сообщение об уничтожении объекта
    }

    SetArray(const SetArray& other) : size(other.size), tag(++counter) {// конструктор создаем новый объект на основе существующего
        for (int i = 0; i < size; ++i) {
            elements[i] = other.elements[i];
        }
        cout << "SetArray " << tag << " copied from " << other.tag << "\n";
    }

    SetArray& operator=(const SetArray& other) { //  проверяем, что объект не присваивается самому себе
        if (this != &other) {
            cout << "SetArray " << tag << " assigned from " << other.tag << "\n";
            size = other.size;
            for (int i = 0; i < size; ++i) {
                elements[i] = other.elements[i];
            }
        }
        return *this;
    }

    void findUnique(const string& A, const string& B, const string& C, const string& D) override {
    size = 0;
    // Добавляем элементы из A, кроме тех, что есть в B и C
    for (char ch : A) {
        if (B.find(ch) == string::npos || C.find(ch) == string::npos) {
            elements[size++] = ch;
        }
    }
    // Добавляем все элементы из D
    for (char ch : D) {
```

```
        if (size < ALPHABET_SIZE && (B.find(ch) == string::npos || C.find(ch) ==
string::npos)) {
            elements[size++] = ch;
        }
    }
}

    void print() const override { // выводим уникальные элементы
        for (int i = 0; i < size; ++i) {
            cout << elements[i] << " ";
        }
        cout << endl;
    }
};
class SetList : public Set {
private:
    static int counter;  // Статический счетчик для тега
    int tag;  // Уникальный тег
    list<char> elements;
public:
    SetList() : tag(++counter) {
        cout << "SetList " << tag << " created\n";
    }

    ~SetList() {
        cout << "SetList " << tag << " destroyed\n";
    }

    SetList(const SetList& other) : elements(other.elements), tag(++counter) {
        cout << "SetList " << tag << " copied from " << other.tag << "\n";
    }

    SetList& operator=(const SetList& other) {
        if (this != &other) {
            cout << "SetList " << tag << " assigned from " << other.tag << "\n";
            elements = other.elements;
        }
        return *this;
    }

    void* operator new(size_t size) { // new используется для динамического
выделения памяти
        cout << "Custom new for SetList\n"; // При каждом вызове этого оператора
будет выводиться сообщение
        return ::operator new(size);
```

```
    }

    void operator delete(void* pointer) { //  delete используется для освобождения
памяти
        cout << "\nCustom delete for SetList";// При каждом вызове этого оператора
будет выводиться сообщение
        ::operator delete(pointer);
    }

    void findUnique(const string& A, const string& B, const string& C, const string&
D) override {
    elements.clear();
    for (char ch : A) {
        if (B.find(ch) == string::npos || C.find(ch) == string::npos) {
            elements.push_back(ch);
        }
    }
    for (char ch : D) {
        if (find(elements.begin(), elements.end(), ch) == elements.end()) {
            elements.push_back(ch);
        }
    }
}

    void print() const override {
        for (char ch : elements) {
            cout << ch << " ";
        }
        cout << endl;
    }
};
// Реализация класса Set для битовых наборов (SetBitset)
class SetBitset : public Set {
private:
    static int counter;  // Статический счетчик для тега
    int tag;
    bitset<ALPHABET_SIZE> elements;

public:
    SetBitset() : tag(++counter) {
        cout << "SetBitset " << tag << " created\n";
    }

    ~SetBitset() {
        cout << "SetBitset " << tag << " destroyed\n";
```

```
    }

    SetBitset(const SetBitset& other) : elements(other.elements), tag(++counter) {
        cout << "SetBitset " << tag << " copied from " << other.tag << "\n";
    }

    SetBitset& operator=(const SetBitset& other) {
        if (this != &other) {
            cout << "SetBitset " << tag << " assigned from " << other.tag << "\n";
            elements = other.elements;
        }
        return *this;
    }

    void findUnique(const string& A, const string& B, const string& C, const string&
D) override {
    elements.reset();
    for (char ch : A) {
        if (B.find(ch) == string::npos || C.find(ch) == string::npos) {
            elements.set(ch - START_CHAR);
        }
    }
    for (char ch : D) {
        elements.set(ch - START_CHAR);
    }
}

    void print() const override {
        for (int i = 0; i < ALPHABET_SIZE; ++i) {
            if (elements.test(i)) {
                cout << static_cast<char>(START_CHAR + i) << " ";
            }
        }
        cout << endl;
    }

};
// Реализация класса Set для машинного слова (SetMachineWord)
class SetMachineWord : public Set {
private:
    static int counter;  // Счетчик для тега
    int tag;
    unsigned int elements;
public:
    SetMachineWord() : elements(0), tag(++counter) {
```

```
        cout << "SetMachineWord " << tag << " created\n";
    }

    ~SetMachineWord() {
        cout << "SetMachineWord " << tag << " destroyed\n";
    }

    SetMachineWord(const SetMachineWord& other) : elements(other.elements),
tag(++counter) {
        cout << "SetMachineWord " << tag << " copied from " << other.tag << "\n";
    }

    SetMachineWord& operator=(const SetMachineWord& other) {
        if (this != &other) {
            cout << "SetMachineWord " << tag << " assigned from " << other.tag <<
"\n";
            elements = other.elements;
        }
        return *this;
    }

    void findUnique(const string& A, const string& B, const string& C, const string&
D) override {
    elements = 0;
    for (char ch : A) {
        if (B.find(ch) == string::npos || C.find(ch) == string::npos) {
            elements |= (1 << (ch - START_CHAR));
        }
    }
    for (char ch : D) {
        elements |= (1 << (ch - START_CHAR));
    }
}

    void print() const override {
        for (size_t i = 0; i < ALPHABET_SIZE; ++i) {
            if (elements & (1 << i)) {
                cout << char(START_CHAR + i) << " ";
            }
        }
        cout << endl;
    }
};
#include "SetClasses.h"
#include <cstdlib>  // Для rand()
```

```
#include <ctime>    // Для time()

int SetArray::counter = 0;
int SetList::counter = 0;
int SetBitset::counter = 0;
int SetMachineWord::counter = 0;

// Реализация функции generateRandomSet
void generateRandomSet(string& set, int size) {
    set.clear();
    while (set.size() < size) {
        char ch = START_CHAR + rand() % ALPHABET_SIZE;
        if (set.find(ch) == string::npos) {
            set += ch;
        }
    }
}
```