Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

Выполнил студент группы.....КС-38.....(Нергарян Геворг Гарегинович)
Ссылка на репозиторий: .....(https://github.com/MUCTR-IKT-CPP/Nergaryan_KC-38_Algos)

Приняли: .....Пысин Максим Дмитриевич
.....Краснов Дмитрий Олегович
.....Лобанов Алексей Владимирович
.....Крашенинников Роман Сергеевич

Дата сдачи: …….16.03.2023……..

Оглавление

Описание задачи.

Вариант 2.

В рамках лабораторной работы необходимо изучить и реализовать одну из трёх структур(двухсвязный список, стек, очередь), в соответствии со своим вариантом, при этом, все структуры должны:

Использовать шаблонный подход, обеспечивая работу контейнера с произвольными данными.

Реализовывать свой итератор предоставляющий стандартный для языка механизм работы с ним(для C++ это операции ++ и операция !=, для python это)

Обеспечивать работу стандартных библиотек и конструкции for each если она есть в языке, если их нет, то реализовать собственную функцию использующую итератор.

Проверку на пустоту и подсчет количества элементов.

Операцию сортировки с использованием стандартной библиотеки.

Стек операции:

- добавление в начало
- взятие из начала

Для демонстрации работы структуры необходимо создать набор тестов(под тестом понимается функция, которая создаёт структуру, проводит операцию или операции над структурой и удаляет структуру):

заполнение контейнера 1000 целыми числами в диапазоне от -1000 до 1000 и подсчет их суммы, среднего, минимального и максимального.

Провести проверку работы операций вставки и изъятия элементов на коллекции из 10 строковых элементов.

заполнение контейнера 100 структур содержащих фамилию, имя, отчество и дату рождения(от 01.01.1980 до 01.01.2020) значения каждого поля генерируются случайно из набора заранее заданных. После заполнение необходимо найти всех людей младше 20 лет и старше 30 и создать новые структуры содержащие результат фильтрации, проверить выполнение на правильность подсчётом кол-ва элементов не подходящих под условие в новых структурах.

Заполнить структуру 1000 элементов и отсортировать ее, проверить правильность использую структуру из стандартной библиотеки и сравнив результат.

(Стек и Очередь) Инверсировать содержимое контейнера заполненного отсортированными по возрастанию элементами не используя операцию перемещения при помощи итератора, а только операторы изъятия и вставки.

(Список) Перемешать все элементы отсортированного списка в случайном порядке.

Описание метода/модели.

Структура - это совокупность элементов, которые могут быть связаны друг с другом определенным образом. Структуры используются для организации и хранения данных в компьютерных программах.

Структуры могут быть определены в различных программах и языках программирования. Например, в языке C структура определяется с помощью ключевого слова **struct**, а в языке Python она может быть определена с помощью классов или словарей.

Структура может включать в себя различные типы данных, такие как целые числа, дробные числа, строки, булевы значения и т.д. Элементы структуры могут быть доступны по отдельности или как часть структуры.

Структуры могут использоваться для организации данных в более сложные структуры, например, списки, деревья, графы и т.д. Они также могут использоваться для передачи данных между функциями или объектами в программе.

Стек - это структура данных, которая представляет собой коллекцию элементов, где доступ к последнему добавленному элементу (вершине стека) осуществляется только через его удаление из структуры данных. Таким образом, стек работает по принципу LIFO (Last-In-First-Out) - последним пришел, первым вышел.

В отличие от списков, мы не можем получить доступ к произвольному элементу стека. Мы можем только добавлять или удалять элементы с помощью специальных методов. У стека нет также метода Contains, как у списков. Кроме того, у стека нет итератора.

Стек может быть реализован как структура данных в памяти компьютера или в виде абстрактного типа данных. Он предоставляет две основные операции: push (добавление элемента на вершину стека) и pop (удаление элемента с вершины стека). Кроме того, стек может также предоставлять операцию peek (просмотр элемента на вершине стека без его удаления).

Стек широко используется в программировании для хранения временных данных, вызова и возврата функций, обработки рекурсивных алгоритмов и т.д.

Выполнение задачи.

Использовался язык C++.

1) Код:

```
#include <iostream>
#include <vector>
#include <stdexcept>
#include <algorithm>
#include <cstdlib>
#include <ctime>
```

```cpp
#include <string>
#include <stack>
#include <chrono>
#include <deque>
#include <random>
using namespace std;


template<typename T>
class Stack {
private:
    std::vector<T> data; // контейнер для хранения данных стека
    size_t count = 0; // переменная-счетчик
public:
    Stack() {} // конструктор по умолчанию

    void push(T element) { // добавление элемента в конец стека
        data.push_back(element);
        count++;
    }

    void push_front(T element) { // добавление элемента в начало стека
        data.insert(data.begin(), element);
        count++;
    }

    T pop() { // удаление верхнего элемента из стека
        if (data.empty()) {
            throw std::out_of_range("Stack is empty");
        }
        T element = data.back();
        data.pop_back();
        count--;
        return element;
    }

    T pop_front() { // удаление первого элемента из стека
        if (data.empty()) {
            throw std::out_of_range("Stack is empty");
        }
        T element = data.front();
        data.erase(data.begin());
        count--;
        return element;
    }

    T top() const { // получение верхнего элемента стека
        if (data.empty()) {
            throw std::out_of_range("Stack is empty");
        }
        return data.back();
    }

    T front() const { // получение первого элемента стека
        if (data.empty()) {
            throw std::out_of_range("Stack is empty");
        }
        return data.front();
    }

    bool empty() const { // проверка стека на пустоту
        return data.empty();
    }

    size_t size() const { // получение размера стека
        return count;
    }

    void sort() { // сортировка элементов стека
```

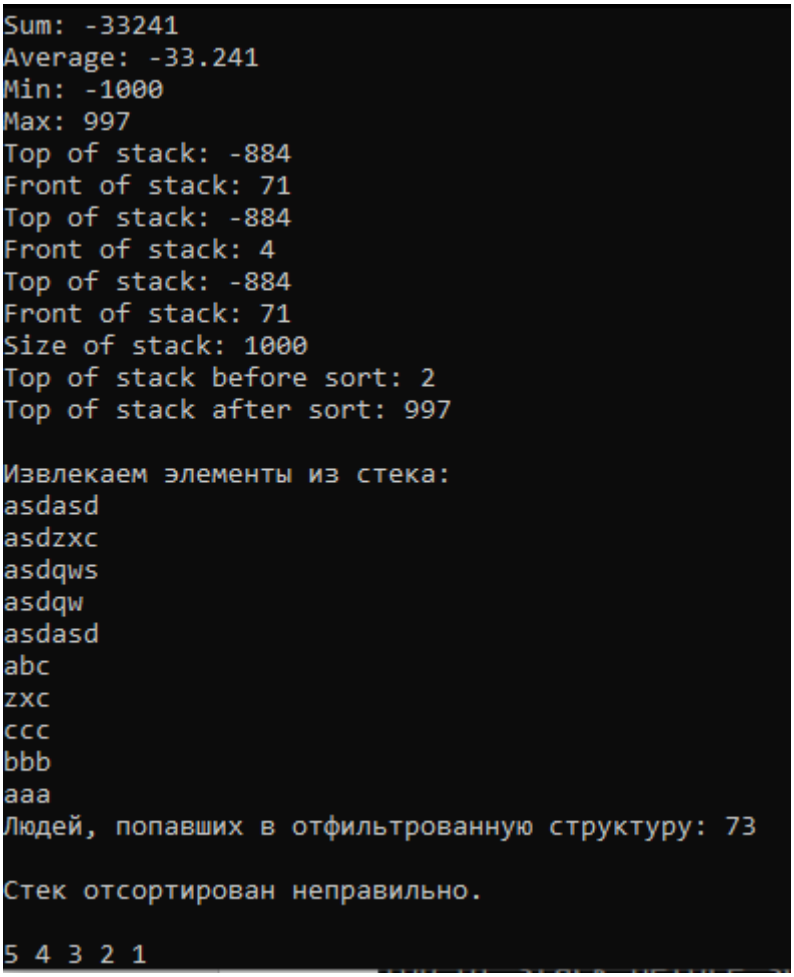
```
        std::sort(data.begin(), data.end());
    }

    T sum() const { // сумма элементов стека
        T result = 0;
        for (const auto& element : data) {
            result += element;
        }
        return result;
    }

    double average() const { // среднее значение элементов стека
        if (data.empty()) {
            return 0;
        }
        return static_cast<double>(sum()) / data.size();
    }

    T min() const { // минимальное значение элементов стека
        if (data.empty()) {
            throw std::out_of_range("Stack is empty");
        }
        T result = data.front();
        for (const auto& element : data) {
            if (element < result) {
                result = element;
            }
        }
        return result;
    }

    T max() const { // максимальное значение элементов стека
        if (data.empty()) {
            throw std::out_of_range("Stack is empty");
        }
        T result = data.front();
        for (const auto& element : data) {
            if (element > result) {
                result = element;
            }
        }
        return result;
    }
};

struct Person {
    string surname;
    string name;
    string patronymic;
    string birth_date;
    int year_date;
};

void invertStack(stack<int>& st) {
    stack<int> temp; // создаем временный стек для хранения элементов

    // Извлекаем элементы из исходного стека и вставляем их в стек temp
    while (!st.empty()) {
        temp.push(st.top());
        st.pop();
    }

    // Извлекаем элементы из стека temp и вставляем их в исходный стек
    while (!temp.empty()) {
        st.push(temp.top());
        temp.pop();
    }
}
```

```
int main() {
    setlocale(LC_ALL, "Russian");
    // Пункт 1
    Stack<int> intStack;
    std::srand(std::time(nullptr)); // инициализация генератора случайных чисел
    for (int i = 0; i < 1000; i++) {
        intStack.push(std::rand() % 2001 - 1000);
    }



    // вывод результатов
    std::cout << "Sum: " << intStack.sum() << std::endl;
    std::cout << "Average: " << intStack.average() << std::endl;
    std::cout << "Min: " << intStack.min() << std::endl;
    std::cout << "Max: " << intStack.max() << std::endl;



    std::cout << "Top of stack: " << intStack.top() << std::endl; // вывод верхнего элемента
    std::cout << "Front of stack: " << intStack.front() << std::endl; // а тут первого
    intStack.push_front(4);       //добавляем в начало
    std::cout << "Top of stack: " << intStack.top() << std::endl; //вывод верхнего элемента
    std::cout << "Front of stack: " << intStack.front() << std::endl; //  а тут первого
    intStack.pop_front();        //удаляем первый элемент
    std::cout << "Top of stack: " << intStack.top() << std::endl; // вывод верхнего элемента
    std::cout << "Front of stack: " << intStack.front() << std::endl; //а тут первого
    std::cout << "Size of stack: " << intStack.size() << std::endl; // подсчет количества
элементов

    intStack.push(2);

    std::cout << "Top of stack before sort: " << intStack.top() << std::endl; // верхний
элемент до сортировки

    intStack.sort();

    std::cout << "Top of stack after sort: " << intStack.top() << std::endl; // тоже самое
после

    //Пункт 2
    //Провести проверку работы операций вставки и изъятия элементов на коллекции из 10
строковых элементов.
    std::stack<std::string> myStack;
    std::string myString;

    // Вставляем 10 элементов в стек
    myStack.push("aaa");
    myStack.push("bbb");
    myStack.push("ccc");
    myStack.push("zxc");
    myStack.push("abc");
    myStack.push("asdasd");
    myStack.push("asdqw");
    myStack.push("asdqws");
    myStack.push("asdzxc");
    myStack.push("asdasd");

    /*
    for (int i = 0; i < 10; i++) {
        std::cout << "Введите строку #" << i + 1 << ": ";
        std::cin >> myString;
        myStack.push(myString);
    }
    */

    // Извлекаем элементы из стека
```

```
    std::cout << "\nИзвлекаем элементы из стека:\n";
    while (!myStack.empty()) {
        std::cout << myStack.top() << std::endl;
        myStack.pop();
    }

    // Пункт 3

    // Создаем вектор структур
    vector<Person> people(100);

    // Набор значений для генерации случайных данных
    vector<string> surnames = { "Ivanov", "Petrov", "Sidorov", "Smirnov", "Kuznetsov" };
    vector<string> names = { "Ivan", "Peter", "Alexey", "Maxim", "Sergey" };
    vector<string> patronymics = { "Ivanovich", "Petrovich", "Sidorovich", "Maximovich",
"Sergeevich" };
    vector<int> days = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 };
    vector<int> months = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    vector<int> years;
    for (int year = 1980; year <= 2020; year++) {
        years.push_back(year);
    }

    // Заполняем вектор случайными данными
    for (Person& person : people) { //итерация по всем эл-там вектора people, где каждый
элемент является структурой Person
        // Генерируем случайные значения для каждого поля
        person.surname = surnames[rand() % surnames.size()];
        person.name = names[rand() % names.size()];
        person.patronymic = patronymics[rand() % patronymics.size()];
        int day = days[rand() % days.size()];
        int month = months[rand() % months.size()];
        int year = years[rand() % years.size()];
        person.birth_date = to_string(day) + "." + to_string(month) + "." + to_string(year);
        person.year_date = year;
    }

    // Создаем вектор для хранения отфильтрованных структур
    vector<Person> filtered_people;

    // Фильтруем структуры по возрасту
    for (const Person& person : people) {
        int birth_year = person.year_date;
        int current_year = 2023;
        int age = current_year - birth_year;
        if (age > 30 || age < 20) {
            filtered_people.push_back(person);
        }
    }

    // Подсчитываем количество элементов не подходящих под условие
    int count = people.size() - filtered_people.size(); // всего людей - прошедших фильтрацию

    cout << "Людей, попавших в отфильтрованную структуру: " << filtered_people.size() << endl
<< endl;

    // Пункт 4

    vector<int> temp_vector;
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(1, 1000);
    for (int i = 0; i < 1000; i++) {
        temp_vector.push_back(dis(gen));
    }

    sort(temp_vector.begin(), temp_vector.end());
    stack<int> std_stack;
```

```
    for (int i = 0; i < temp_vector.size(); i++) {
        std_stack.push(temp_vector[i]);
    }

    stack<int> my_stack;
    for (int i = 0; i < 1000; i++) {
        my_stack.push(dis(gen));
    }
    vector<int> temp_vector_2;
    while (!my_stack.empty()) {
        temp_vector_2.push_back(my_stack.top());
        my_stack.pop();
    }
    sort(temp_vector_2.begin(), temp_vector_2.end());
    for (int i = 0; i < temp_vector_2.size(); i++) {
        my_stack.push(temp_vector_2[i]);
    }
    bool is_sorted_correctly = true;
    while (!std_stack.empty() && !my_stack.empty()) {
        if (std_stack.top() != my_stack.top()) {
            is_sorted_correctly = false;
            break;
        }
        std_stack.pop();
        my_stack.pop();
    }
    if (is_sorted_correctly) {
        cout << "Стек отсортирован правильно." << endl << endl;
    }
    else {
        cout << "Стек отсортирован неправильно." << endl << endl;
    }

    // Пункт 5

    stack<int> st;

    // Заполняем стек отсортированными по возрастанию элементами
    st.push(1);
    st.push(2);
    st.push(3);
    st.push(4);
    st.push(5);

    // Инвертируем содержимое стека
    invertStack(st);

    // Выводим на экран содержимое инвертированного стека
    while (!st.empty()) {
        cout << st.top() << " ";
        st.pop();
    }




    return 0;
}
```

Данные выводятся в консоль.



Сначала заполняем стек случайными значениями и выводим подсчет их суммы, среднего, минимального и максимального. Дальше вывод для наглядной демонстрации добавление в начало и взятие из начала.

Создаем новый стек и заполняем 10 строками (в коде уже введено, но при желании в комментарии остался вариант с ручным вводом каждой строки), после изымаем.

Создаем новый стек содержащих фамилию, имя, отчество и дату рождения(от 01.01.1980 до 01.01.2020) значения каждого поля генерируются случайно из набора заранее заданных. Дальше выводим количество элементов, попадающих под условие младше 20 лет или старше 30.

Заполняем стек 1000 элементов и сортируем, проверяем с сортировкой из стандартной библиотеки и выводим ответ(сортировка неправильная, потому что убывающая).

Заполняем новый стек 5 элементами по возрастанию, после инвертируем его при помощи операторов изъятия и вставки и выводим в консоль.

Заключение.

Работа со структурами и стеками на языке C++ может быть очень полезной и удобной при решении различных задач. Структуры позволяют объединять несколько переменных разных типов в одном объекте, что упрощает работу с данными в программе.

Стек является одной из наиболее распространенных структур данных в программировании. Он используется для хранения и управления данными в порядке Last-In-First-Out (LIFO), что может быть очень удобно в различных ситуациях.

Стандартная библиотека C++ предоставляет мощные и удобные инструменты для работы со структурами и стеками. Например, стек может быть реализован как контейнерный класс **std::stack**, который обеспечивает множество методов для добавления, извлечения и управления элементами стека.

Кроме того, в C++ есть возможность определять свои собственные структуры и классы, что позволяет создавать более сложные структуры данных, а также добавлять дополнительные методы и функции для работы с ними.

В целом, работа со структурами и стеками в C++ может быть очень полезной и удобной, особенно при работе с большими объемами данных или при решении сложных задач, связанных с управлением данными.