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-30 Суханова Евгения Валерьевна
Ссылка на репозиторий:
https://github.com/MUCTR-IKT-CPP/EVSuhanova_30/blob/main/Algoritms/laba3.cpp

Приняли: Пысин Максим Дмитриевич
Краснов Дмитрий Олегович

Дата сдачи: 03.03.2023

Оглавление

Описание задачи.

В рамках лабораторной работы необходимо изучить и реализовать одну из трёх структур, 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 !=)
- 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 создает 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 подсчетом 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.
- Инвертировать содержимое контейнера заполненного отсортированными по возрастанию элементами не используя операцию перемещения при помощи итератора, а только операторы изъятия и вставки.

Описание метода/модели.

Стек сам по себе является частным случаем списка, поэтому часто его реализация является просто модификацией над уже существующими в библиотеке типом списка, которая просто запрещает доступ к определенным операциям, и добавляя новую.

Каждый элемент стека состоит из поля хранящего значение элемента стека, и поле указателя на следующий элемент стека. Это соответствует однонаправленному списку, однако в отличии от него, начало стека является его же и концом.

Вставка

Как правило эта операция называется push.

Алгоритм вставки выглядит следующим образом:

- Получаем новое значение
- Создаем новый элемент стека с полями значения и указателя на последующий элемент
- Присваиваем значение к полю значения.
- Присваиваем к указателю на последующий элемент указатель на текущий первый элемент стека
- Присваиваем к указателю на текущий первый элемент, указатель на новосозданный элемент

Удаление

Как правило эта операция называется pop.

Алгоритм удаления выглядит следующим образом:

- Берем первый(верхний) элемент стека
- В указатель на верхний элемент стека записываем значение поля указателя на следующий элемент взятое из удаляемого первого элемента
- Возвращаем значение удаленного элемента

Выполнение задачи.

Для реализации стека использовался язык программирования C++.

Для начала нам необходимо реализовать с помощью контейнера все функции стека, которые пригодятся нам при выполнении лабораторной работы:

- функция проверки на пустоту;
- функция вставки нового элемента;
- функция получения верхнего элемента;
- функция удаления верхнего элемента;
- функция получения размера стека;
- функция сортировки стека;
- функция вывода в файл;
- указатели на начало и конец стека;
- оператор перемещения;
- оператор возврата значения;
- оператор сравнения.

Далее были реализованы функции для демонстрации работы созданного контейнера.

1. В функции filling() мы заполняем стек, используя оператор push(), случайными целыми числами в пределах [-1000; 1000].
2. С помощью функции summ() мы находим сумму всех элементов, максимальный, минимальный элементы и среднее значение всех элементов.

```
Сумма, макс, мин, сред

min: -997
max: 1000
aver: -13
summ: -13770
```

3. Данный массив был отсортирован с помощью оператора sorting().

```
Отсортированный стек

-997
-995
-993
-993
-993
-992
-990
-987
-984
-984
-984
-980
-979
-972
-970
```

начало

```
961
961
961
963
969
971
975
976
977
981
985
991
992
994
997
1000
```

конец

4. Было произведено сравнение сортировки стека, созданного с помощью шаблона, и стека из стандартной библиотеки. В результате мы получили, что стеки сортируются верно.

```
Стеки равны
```

5. Далее была выполнена инверсия отсортированного стека

```
Инверсия стека

1000
997
994
992
991
985
981
977
976
975
971
969
963
961
961
```

начало

```
-968
-970
-972
-979
-980
-984
-984
-984
-987
-990
-992
-993
-993
-993
-995
-997
```

конец

6. Была выполнена проверка вставки и изъятия элементов из стека 10 строковых элементов.

```
Добавлен элемент: 1 строка
Добавлен элемент: 2 строка
Добавлен элемент: 3 строка
Добавлен элемент: 4 строка
Добавлен элемент: 5 строка
Добавлен элемент: 6 строка
Добавлен элемент: 7 строка
Добавлен элемент: 8 строка
Добавлен элемент: 9 строка
Добавлен элемент: 10 строка
```

```
Изъят элемент: 10 строка
Изъят элемент: 9 строка
Изъят элемент: 8 строка
Изъят элемент: 7 строка
Изъят элемент: 6 строка
Изъят элемент: 5 строка
Изъят элемент: 4 строка
Изъят элемент: 3 строка
Изъят элемент: 2 строка
Изъят элемент: 1 строка
```

7. Далее мы создаем стек структур Person с полями ФИО и дата рождения.
8. С помощью функции filter() мы выбрали тех людей, чей возраст в пределах от 20 до 30 лет, и записали их в отдельный стек структур, а также вывели этот стек в файл.

```
Фильтр на возраст

Nikolaev Petr Petrovich 25/7/1994
Ivanov Ivan Borisovich 19/2/2002
Borisov Semen Nikolaevich 16/7/1995
Nikolaev Petr Semenovich 24/7/1997
Borisov Nikolay Petrovich 28/11/1999
Smirnov Petr Borisovich 23/10/1995
Smirnov Nikolay Petrovich 27/12/1994
Nikolaev Ivan Semenovich 5/7/1994
Petrov Ivan Nikolaevich 7/6/1992
Smirnov Semen Borisovich 4/8/1998
```

9. Также была произведена проверка на количество вошедших по условию в новый стек и количество не попавших в него.

```
Текущая дата: 03/03/2023
Фильтрация выполняется верно
```

Далее представлен код на языке C++:

```
#include <iostream>
#include <Windows.h>
#include <algorithm>
#include <stdlib.h>
#include <time.h>
#include <string>
#include <fstream>
#include <stack>

using namespace std;

/* Структура данных о человеке */
struct Person {
    string surname; // Фамилия
    string name; // Имя
    string patronymic; // Отчество
    int day; // День рождения
    int month; // Месяц рождения
    int year; // Год рождения
};

//Шаблонный класс
template <typename T>
class Stack {
public:
    T* arr; //Контейнер для хранения элементов
    int count; //Количество переменных

    class Iterator {
    private:
        Stack<T>* mystack; //Объект стэка
        int index; //Индекс
    public:
        Iterator(int index, Stack<T>* mystack) :
            //Задаем стэк и начальный индекс
            mystack(mystack),
```

```
        index(index > mystack->size() ? -1 : index) {
    }

    //Оператор перемещения
    Iterator& operator++() {
        //Если есть следующий элемент, то идем к нему
        if (index != mystack->size() - 1) {
            index++;
        }
        //Иначе элементов нет
        else {
            index = -1;
        }
        //Разыменование указателя
        return *this;
    }

    //Оператор возврата значения
    T operator*() {
        return mystack->arr[index];
    }

    //Оператор сравнения
    bool operator !=(Iterator& it) {
        return it.index != index;
    }
};

//Создание пустого стека
Stack() {
    arr = nullptr;
    count = 0;
}

//Проверка, пустой ли стек
bool empty() {
    //Если стек пуст, то вернется true
```

```
    return count == 0;
}

//Добавить элемент наверх
void push(T x) {
    /*
    *@param x - элемент, который помещаем в стек
    *@param temp - временный контейнер
    */
    T* temp;

    temp = arr;

    //Выделяем новую память на 1 элемент больше
    arr = new T[count + 1];
    //arr = (T*)malloc((count + 1) * sizeof(T));

    //Увеличиваем количество элементов
    count++;

    //Перезаписываем основной контейнер с помощью временного
    for (int i = 0; i < count - 1; i++) {
        arr[i] = temp[i];
    }
    //Добавляем новый элемент
    arr[count - 1] = x;

    //Очищаем временный контейнер
    delete[] temp;

}

//Получить элемент сверху
T pop() {

    //Если стек не пуст
    if (!empty()) {
```

```
        //Возвращаем верхний элемент
        return arr[count - 1];
    }

    //Иначе он пуст
    else {
        cout << "Стэк пуст" << endl;
        return 0;
    }
}

//Удалить элемент сверху
void del() {
    /*
    *@param temp - временный контейнер
    */
    T* temp;

    temp = arr;

    //Выделяем новую память на 1 элемент меньше
    arr = new T[count - 1];
    //arr = (T*)malloc((count - 1) * sizeof(T));

    //Уменьшаем размер стека
    count--;

    //Перезаписываем основной контейнер с помощью временного
    for (int i = 0; i < count; i++) {
        arr[i] = temp[i];
    }

    //Удаляем временный контейнер
    delete[] temp;
}

//Получить размер стэка
```

```
int size() {
    return count;
}

// Сортировка
void sorting() {
    sort(arr, arr + size());
}

//Функция вывода стека
void print() {
    /*
    *@param p - хранилище элементов
    */
    T* p = arr;

    //Если не пуст
    if (!empty()) {
        //Проходимся по количеству элементов
        for (int i = 0; i < count; i++) {

            ofstream out;

            out.open("C:\\Users\\evgen\\Desktop\\Алгоритмы\\data_3laba.txt", ios::app);
            if (out.is_open())
            {
                out << *p << endl;
            }

            //Выводим текущий элемент
            //cout << *p << endl;

            //Идем к следующему
            p++;
        }
    }
    //Инач стек пуст
```

```
        else {
            cout << "Стек пуст" << endl;
        }
    }

    //Указатель на начало
    Iterator& begin() {
        return *(new Iterator(0, this));
    }

    //Указатель на конец
    Iterator& end() {
        return *(new Iterator(-1, this));
    }

};

void filling(); // Заполнение стека числами
void summ(Stack<int> myStack, int count); // Нахождение суммы, максимального, минимального, среднего элементов
void pasteAndDel(); // Проверка вставки и  изъятия
Stack<int> sortTest(Stack<int> myStack, stack<int> standart_stack); // Проверка сортировки
Stack<int> reverseStack(Stack<int> myStack); //Инверсия стека
void generationPersons(); // Генерация данных о людях
string generationSurname(); // Генератор фамилии
string generationName(); // Генератор имени
string generationPatronymic(); // Генератор отчества
int generationYear(); // Генератор года рождения
int generationMonth(); // Генератор месяца рождения
int generationDay(int month, int year); // Генератор дня рождения
void filter(Stack<Person> persons, int count); // Фильтрация
void clearFile(); // Очистка файла

int main()
{
    SetConsoleCP(1251); //Поддержка ввода на русском языке
    setlocale(LC_ALL, "Russian"); //Установка русской локали
```

```
    srand(time(0));
    clearFile();

    filling(); // Заполнение стека
    pasteAndDel(); // Втавка и изъятие
    generationPersons(); // Данные о людях
}

/* Заполнение стека
*
* @param myStack созданный стек
* @param standart_stack стек из стандартной библиотеки
* @param count количество элементов в стеке
*/
void filling() {
    Stack<int> myStack;
    stack<int> standart_stack;
    int count = 1000;
    int temp;
    for (int i = 0; i < count; i++) {
        temp = -1000 + rand() % 2001;
        standart_stack.push(temp); // Добавление элемента в стандартный стек
        myStack.push(temp); // Добавление элемента в созданный стек
    }
    ofstream out;
    out.open("C:\\Users\\evgen\\Desktop\\Алгоритмы\\data_3laba.txt", ios::app);
    if (out.is_open())
    {
        out << "Отсортированный стек\n" << endl;
    }
    myStack = sortTest(myStack, standart_stack); // Проверка сортировки стека на правильность
    myStack.print(); // Вывод стека

    if (out.is_open())
    {
        out << "Инверсия стека\n" << endl;
    }
```

```
    myStack = reverseStack(myStack); // Инверсия стека
    myStack.print(); // Вывод стека

    cout << "\nСумма, макс, мин, сред\n";
    summ(myStack, count);
}

/* Нахождение суммы, максимального, минимального, среднего элементов
*
* @param myStack созданный стек
* @param count длина стека
* @param s сумма элементов
* @param min минимальный элемент
* @param max максимальный элемент
*/
void summ(Stack<int> myStack, int count) {
    int s = 0;
    int el, min = myStack.pop(), max = myStack.pop();
    // Пока стек не пуст
    while (!myStack.empty()) {
        el = myStack.pop(); // Элемент
        if (min > el) min = el;
        if (max < el) max = el;
        s += el;
        myStack.del(); // Удаление элемента
    }
    cout << "\nmin: " << min << "\nmax: " << max << "\naver: " << s/count << "\nsumm: " << s << endl;
}

/* Проверка вставки и  изъятия
*
* @param myStack созданный стек
* @param count длина стека
*/
void pasteAndDel() {
    Stack<string> myStack;
    int count = 10;
```

```
        for (int i = 0; i < count; i++) {
            myStack.push(to_string(i+1) + " строка"); // Вставка элемента
            cout << "Добавлен элемент: " << myStack.pop() << endl;
        }

        cout << "Изъятие элементов\n";
        while (!myStack.empty()) {
            cout << "Изъят элемент: " << myStack.pop() << endl; // Изъятие элемента
            myStack.del(); // Удаление элемента из стека
        }
    }

    /* Проверка сортировки
    *
    * @param myStack созданный стек
    * @param TestStack Стек для тестирования
    * @param standart_stack стек из стандартной библиотеки
    * @param count длина стека
    * return myStack возвращает отсортированный стек
    */
    Stack<int> sortTest(Stack<int> myStack, stack<int> standart_stack) {
        Stack<int> TestStack;
        myStack.sorting(); // Сортировка стека
        using cnt = decltype(standart_stack)::container_type;
        // Сортировка стека из стандартной библиотеки
        sort(reinterpret_cast<cnt*>(&standart_stack)->begin(),
reinterpret_cast<cnt*>(&standart_stack)->end());
        int count = standart_stack.size(); // Длина стека
        // Проверка совпадения длин стеков
        if(standart_stack.size() == myStack.count){
            for (int i = 0; i < count; i++) {
                TestStack.push(myStack.pop()); // Запись в тестовый стек
                if (myStack.pop() != standart_stack.top()) break; // Проверка совпадения элементов
                myStack.del();
                standart_stack.pop();
            }
        }
```

```
    // Проверка на совпадение стеков
    if (standart_stack.empty() && myStack.empty()) cout << "Стеки равны\n";
    else cout << "Стеки неравны\n";
    return reverseStack(TestStack);
}

/* Инверсия стека
*
* @param myStack созданный стек
* @param TestStack Стек для тестирования
* return myStack возвращает инвертированный стек
*/
Stack<int> reverseStack(Stack<int> myStack) {
    Stack<int> TestStack;
    while (!myStack.empty()) {
        TestStack.push(myStack.pop());
        myStack.del();
    }
    return TestStack;
}

/* Генерация данных о человеке
*
* @param persons стек людей
* @param new_person новый человек
* @param count количество людей
*/
void generationPersons() {
    int count = 100;
    Stack<Person> persons;
    Person new_person;

    for (int i = 0; i < count; i++) {

        new_person.name = generationName(); // Генрация имени
        new_person.surname = generationSurname(); // Генрация фамилии
        new_person.patronymic = generationPatronymic();  // Генрация отчества
```

```
            new_person.year = generationYear();  // Генрация года рождения
            new_person.month = generationMonth(); // Генрация месяца рождения
            new_person.day = generationDay(new_person.month, new_person.year); // Генрация дня
рождения
            persons.push(new_person); // Добавление нового человека в стек

            /*cout << persons.arr[i].surname << " " << persons.arr[i].name << " " << persons.arr[i].patronymic
<< " ";
            cout << persons.arr[i].day << "/" << persons.arr[i].month << "/" << persons.arr[i].year << endl;*/
        }
        filter(persons, count); // Фильтрация по возрасту
    }

    /* Генерация фамилии
    *
    * @param surnames массив фамилий
    */
    string generationSurname() {
        string surnames[] = {"Ivanov", "Petrov", "Smirnov", "Borisov", "Nikolaev"};
        return surnames[rand() % 5];
    }

    /* Генерация имен
    *
    * @param names массив имен
    */
    string generationName() {
        string names[] = { "Ivan", "Petr", "Semen", "Boris", "Nikolay" };
        return names[rand() % 5];
    }

    /* Генерация отчества
    *
    * @param patronymic массив отчеств
    */
    string generationPatronymic() {
        string patronymic[] = { "Ivanovich", "Petrovich", "Semenovich", "Borisovich", "Nikolaevich" };
```

```
    return patronymic[rand() % 5];
}

/* Генерация года рождения */
int generationYear() {
    return 1980 + rand() % 41;
}

/* Генерация месяца рождения */
int generationMonth() {
    return 1 + rand() % 13;
}

/* Генерация дня рождения
*
* @param month месяц
* @param year год
*/
int generationDay(int month, int year) {
    if (month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month == 10 || month == 12)
{
        return 1 + rand() % 32; // январь, март, май, июнь, август, октябрь, декабрь
    }
    else if (month == 2) { // февраль
        if (year % 4 != 0 || (year % 100 == 0 && year % 400 != 0))
            return 1 + rand() % 29; // невисокосный год
        else
            return 1 + rand() % 30; // високосный год
    }
    else {
        return 1 + rand() % 31; // апрель, июнь, сентябрь, ноябрь
    }
}

/* Фильтрация
*
* @param persons стек людей
```

```
* @param count количество элементов в стеке
* @param max_age максимальный возраст
* @param min_age минимальный возраст
* @param age возраст
* @param count_filter количество подходящих людей
* @param count_not_filter количество не подходящих людей
* @param t текущее время
*/
void filter(Stack<Person> persons, int count) {
    ofstream out;
    out.open("C:\\Users\\evgen\\Desktop\\Алгоритмы\\data_3laba.txt", ios::app);
    if (out.is_open())
    {
        out << "Фильтр на возраст\n" << endl;
    }
    Stack<Person> filter_persons;
    cout << "------------------------\n";
    int max_age = 30, min_age = 20;
    int age, count_filter = 0, count_not_filter = 0;
    time_t t1 = time(NULL);
    struct tm t;
    localtime_s(&t, &t1);
    // Текущая дата
    printf("Текущая дата: %.2d/%.2d/%.2d\n", t.tm_mday, t.tm_mon + 1, t.tm_year + 1900);


    for (int i = 0; i < count; i++) {

        age = t.tm_year + 1900 - persons.arr[i].year; // Подсчет возраста

        if (t.tm_mon + 1 < persons.arr[i].month) age--; // Не было дня рождения в этом году
        else if (t.tm_mon + 1 == persons.arr[i].month)
            if (t.tm_mday < persons.arr[i].day) age--; // Не было дня рождения в этом году

        if (age <= max_age && age >= min_age) { // Проверка условий фильтрации
            filter_persons.push(persons.arr[i]); // Добвление в стек
            // Запись в файл
```

```
            if (out.is_open())
            {
                out << filter_persons.arr[count_filter].surname << " " << filter_persons.arr[count_filter].name
<< " " << filter_persons.arr[count_filter].patronymic << " ";
                out << filter_persons.arr[count_filter].day << "/" << filter_persons.arr[count_filter].month <<
"/" << filter_persons.arr[count_filter].year << endl;
            }
            count_filter++; // Прибавление длины стека
        }
        else count_not_filter++;
    }

    // Проверка количества отфильтрованных людей
    if (count == count_filter + count_not_filter) cout << "Фильтрация выполняется верно\n";
    else cout << "Фильтрация выполняется неверно\n";

}

// Очистка файла
void clearFile() {
    ofstream out;
    out.open("C:\\Users\\evgen\\Desktop\\Алгоритмы\\data_3laba.txt");
}
```

Заключение.

В заключении можно заметить, что реализованный стек имеет те же функции, что и стек из стандартной библиотеки. Аналогично стеку из библиотеки реализованный стек также сортируется. Также была сделана функция, инвертирующая стек без использования итератора. Успешно выполняются функции вставки и изъятия строковых элементов.

Помимо этого, была реализована функция, позволяющая создать стек структур, содержащих информацию о человеке (ФИО и дату рождения). Далее была выполнена фильтрация этого стека по условию, что возраст человека входит в пределы от 20 до 30 лет. Подходящие под это условие люди были записаны в новый стек структур. И была проведена проверка, которая показала, что фильтрация работает верно.