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: 06.03.2023

Оглавление

Описание задачи. **2**
Описание метода/модели. **3**
Выполнение задачи. **4**
Заключение. **9**

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++.

Класс стека имеет параметры самого контейнера arr и количество переменных в этом стеке count.

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

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

- функция проверки на пустоту;

```
//Проверка, пустой ли стек
bool empty() {
    //Если стек пуст, то вернется true
    return count == 0;
}
```

- функция вставки нового элемента;

```
void push(T x) {
    T* temp;
    temp = arr;
    arr = new T[count + 1];
    count++;
    for (int i = 0; i < count - 1; i++)
        arr[i] = temp[i];
    arr[count - 1] = x;
    delete[] temp;

}
```

- функция получения верхнего элемента;

```
T pop() {
    if (!empty()) {
        return arr[count - 1];
    }
    else {
        cout << "Стэк пуст" << endl;
        return 0;
    }
}
```

- функция удаления верхнего элемента;

```
void del() {
    T* temp;
    temp = arr;
    arr = new T[count - 1];
    count--;
    for (int i = 0; i < count; i++)
        arr[i] = temp[i];
    delete[] temp;
}
```

- функция получения размера стека;

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

- функция сортировки стека;

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

- функция вывода в файл;

```
void print() {
    T* p = arr;
    if (!empty()) {
        for (int i = 0; i < count; i++) {
            ofstream out;
            out.open(_Filename: "C:\\Users\\evgen\\Desktop\\Алгоритмы\\data_3laba.txt", _Mode: ios::app);
            if (out.is_open())
                out << *p << endl;
            p++;
        }
    }
    else
        cout << "Стек пуст" << endl;
}
```

- указатели на начало и конец стека;

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

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

- оператор перемещения;

```
//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;

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

- оператор сравнения.

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

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].

```
for (int i = 0; i < count; i++) {
    temp = -1000 + rand() % 2001;
    standart_stack.push(_val: temp); // Добавление элемента в стандартный стек
    myStack.push(x: temp); // Добавление элемента в созданный стек
}
```

2. С помощью функции summ() мы находим сумму всех элементов, максимальный, минимальный элементы и среднее значение всех элементов.

```
while (!myStack.empty()) {
    el = myStack.pop(); // Элемент
    if (min > el) min = el;
    if (max < el) max = el;
    s += el;
    myStack.del(); // Удаление элемента
}
```

```
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.

```
if(standart_stack.size() == myStack.count){
    for (int i = 0; i < count; i++) {
        TestStack.push(x: 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";
```

```
Стеки равны
```

5. Далее была выполнена инверсия отсортированного стека

```
Stack<int> reverseStack(Stack<int> myStack) {
    Stack<int> TestStack;
    while (!myStack.empty()) {
        TestStack.push(x: myStack.pop());
        myStack.del();
    }
    return TestStack;
}
```

```
Инверсия стека

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

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

начало конец

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

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

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

```
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.

```
/* 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
};
```

```
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(x: new_person); // Добавление нового человека в стек
}
```

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.

```
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( x: persons.arr[i]); // Добвление в стек
        // Запись в файл
        if (out.is_open()) { ... }
        count_filter++; // Прибавление длины стека
    }
    else count_not_filter++;
}
```

```
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.

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

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

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.