|  |  |
| --- | --- |
| C:\Documents and Settings\admin\Рабочий стол\МИРЭА_ЭМБЛЕМА_приказ.JPG | |
| МИНОБРАНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  «**МОСКОВСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»**  **МИРЭА** | |
| Институт информационных технологий  Кафедра математического обеспечения и стандартизации информационных технологий | |
| **Отчет по лабораторной работе №9**  **Тема работы «Структуры данных. STL-контейнеры.»** | |
| Выполнил: |  |
| Студент группы ИКБО-07-17  Отчет принял:  Преподаватель кафедры МОСИТ | Акжигитов Р. Р.  Миронов Антон Николаевич |
| Москва  2018 | |

Оглавление

[Задание 1 3](#_Toc517782041)

[Условия задания 3](#_Toc517782042)

[Декомпозиция 3](#_Toc517782043)

[Определения классов 3](#_Toc517782044)

[Тестирование 5](#_Toc517782045)

[Кодирование алгоритма программы 5](#_Toc517782046)

[Вывод 8](#_Toc517782047)

# Задание 1

# Условия задания

**Для стека:** втолкнуть элемент, вытолкнуть, определить количество элементов в стеке.

Стек – структура данных с дисциплиной обслуживания: последний пришел первый вышел.

**Класс:** Студент (Фамилия, имя, номер группы).

# Декомпозиция

1. Класс Стек на динамическом массиве.
2. При добавлении элемента он помещается в начало, остальные элементы сдвигаются на 1 позицию вправо, длина увеличивается на 1.
3. При получении элемента, последний элемент достается из массива, длина массива уменьшается на 1.
4. Класс Студент с перегруженным выводом и присваиванием, так как в стеке data[0] = element (Student), требует этого оператора.
5. Класс Итератор. Перегружены арифметические операции, присваивание и неравенство.

# Определения классов

template<class T>

class Iterator {

public:

Iterator() {}

Iterator(T\* iter) {

this->iter = iter;

}

void operator==(Iterator elem) {

this->iter = elem->iter;

}

void operator==(T\* elem) {

this->iter = elem;

}

Iterator operator++(int) {

Iterator oldValue = \*this;

this->iter++;

return oldValue;

}

Iterator operator--(int) {

Iterator oldValue = \*this;

this->iter--;

return oldValue;

}

Iterator& operator++() {

this->iter++;

return \*this;

}

Iterator& operator--() {

this->iter--;

return \*this;

}

friend bool operator!=(Iterator elem1, Iterator elem2) {

return elem1.iter != elem2.iter;

}

T\* getIter() {

return this->iter;

}

private:

T\* iter = 0;

};

template<class T>

class Stack {

public:

Stack() {}

~Stack() { free(data); }

void push(T element) {

data = (T\*)realloc(data, sizeof(T) \* ++length);

for (int i = length - 2; i >= 0; i--)

data[i + 1] = data[i];

data[0] = element;

}

T pop() {

T temp = data[--length];

data = (T\*)realloc(data, sizeof(T) \* length);

return temp;

}

void print() {

for (int i = 0; i < length; i++)

cout << data[i] << " ";

cout << endl;

}

int getLength() {

return length;

}

T\* begin() {

return data;

}

T\* end() {

return data + length;

}

private:

T\* data = 0;

int length = 0;

};

class Student {

public:

Student(const char f[255], const char i[255], const char group[255]) {

strcpy\_s(this->f, f);

strcpy\_s(this->i, i);

strcpy\_s(this->group, group);

}

friend ostream& operator<<(ostream& os, Student& st) {

os << "Student: " << st.f << " " << st.i << "\nGroup: " << st.group << endl;

return os;

}

Student& operator=(Student& st) {

strcpy\_s(this->f, st.f);

strcpy\_s(this->i, st.i);

strcpy\_s(this->group, st.group);

return \*this;

}

private:

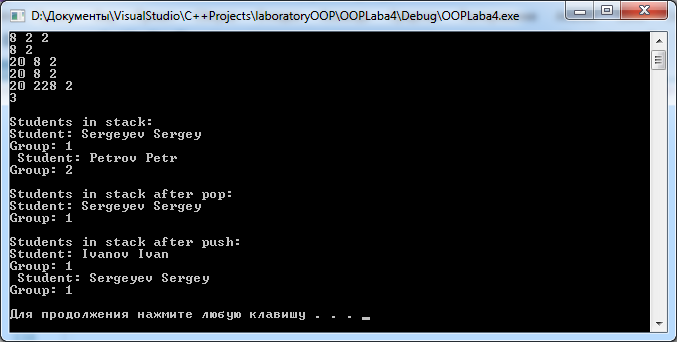
char f[255],

i[255],

group[255];

};

# Тестирование



Добавление в Стек чисел (2, 2, 8), получение из Стека числа (2), добавление числа (20).

Проход с выводом в консоль с помощью Итератора. Изменение по индексу 1 числа 8 на число 228 с помощью Итератора.

Добавление 2-х студентов в Стек. Удаление одного, добавление следующего.

# Кодирование алгоритма программы

#include <iostream>

#include <vector>

#include <string>

using namespace std;

template<class T>

class Iterator {

public:

Iterator() {}

Iterator(T\* iter) {

this->iter = iter;

}

void operator==(Iterator elem) {

this->iter = elem->iter;

}

void operator==(T\* elem) {

this->iter = elem;

}

Iterator operator++(int) {

Iterator oldValue = \*this;

this->iter++;

return oldValue;

}

Iterator operator--(int) {

Iterator oldValue = \*this;

this->iter--;

return oldValue;

}

Iterator& operator++() {

this->iter++;

return \*this;

}

Iterator& operator--() {

this->iter--;

return \*this;

}

friend bool operator!=(Iterator elem1, Iterator elem2) {

return elem1.iter != elem2.iter;

}

T\* getIter() {

return this->iter;

}

private:

T\* iter = 0;

};

template<class T>

class Stack {

public:

Stack() {}

~Stack() { free(data); }

void push(T element) {

data = (T\*)realloc(data, sizeof(T) \* ++length);

for (int i = length - 2; i >= 0; i--)

data[i + 1] = data[i];

data[0] = element;

}

T pop() {

T temp = data[--length];

data = (T\*)realloc(data, sizeof(T) \* length);

return temp;

}

void print() {

for (int i = 0; i < length; i++)

cout << data[i] << " ";

cout << endl;

}

int getLength() {

return length;

}

T\* begin() {

return data;

}

T\* end() {

return data + length;

}

private:

T\* data = 0;

int length = 0;

};

class Student {

public:

Student(const char f[255], const char i[255], const char group[255]) {

strcpy\_s(this->f, f);

strcpy\_s(this->i, i);

strcpy\_s(this->group, group);

}

friend ostream& operator<<(ostream& os, Student& st) {

os << "Student: " << st.f << " " << st.i << "\nGroup: " << st.group << endl;

return os;

}

Student& operator=(Student& st) {

strcpy\_s(this->f, st.f);

strcpy\_s(this->i, st.i);

strcpy\_s(this->group, st.group);

return \*this;

}

private:

char f[255],

i[255],

group[255];

};

int main() {

Stack<int> st;

st.push(2);

st.push(2);

st.push(8);

st.print();

st.pop();

st.print();

st.push(20);

st.print();

for (Iterator<int> it = st.begin(); it != Iterator<int>(st.end()); it++)

cout << \*it.getIter() << " ";

cout << endl;

Iterator<int> it = st.begin() + 1;

\*it.getIter() = 228;

st.print();

cout << st.getLength();

cout << "\n\n";

Student stud("Ivanov", "Ivan", "1");

Student stud1("Petrov", "Petr", "2");

Student stud2("Sergeyev", "Sergey", "1");

Stack<Student> studStack;

studStack.push(stud1);

studStack.push(stud2);

cout << "Students in stack:\n";

studStack.print();

studStack.pop();

cout << "Students in stack after pop:\n";

studStack.print();

studStack.push(stud);

cout << "Students in stack after push:\n";

studStack.print();

system("pause");

return 0;

}

# Вывод

В данной лабораторной работе я научился создавать свои контейнеры STL и работать с уже готовыми, написал свой Итератор, и проверил его работоспособность (проход по массиву с помощью итераторов, изменение конкретного индекса с помощью итератора).