Выполнили:

студенты группы 19ВВ2

Гусев Д.О.

Кубасов И.М.

Приняли:

Синев М.П.

Дорошенко И.Н.

Пенза 2021

ОТЧЕТ

по лабораторной работе №7

по курсу «Технологии программирования»

на тему «Динамическая структура данных»

Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

### Название

Динамическая структура данных.

### Цель работы

Изучение вложенных классов и принципов инкапсуляции и полиморфизма в языке С++. Повышение навыков работы с виртуальными функциями.

### Требования к классу

- Класс в соответствии с заданием должен формировать динамическую структуру данных из объектов Derived1, Derived2 и Derived3. Для этого в классе должен быть реализован вложенный класс <Элемент структуры>.

- Класс <Элемент структуры> является служебным классом, который описывает одну единицу данных, входящую в структуру данных. Класс должен включать в себя весь необходимый набор переменных для организации структуры данных (указатель на следующий/предыдущий элемент данных и т.п.). Одним из полей класса <Элемент структуры> должна быть переменная типа Interface\*. Данная переменная должна использоваться для размещения адреса объекта классов Derived1, Derived2 и Derived3.

- В классе <Структура> должны быть реализованы:

* Функции, указанные в варианте задания.
* Функции обхода всех элементов структуры данных: GetFirst и GetNext. Функция GetFirst должна возвращать указатель типа Interface\* на первый элемент данных. Функция GetNext должна возвращать указатель типа Interface\* на следующий элемент данных. При этом для линейных структур данных выход за границу структуры должен вызывать ошибку (функция GetFirst и/или GetNext должны возвращать NULL). Для циклических структур данных выход за границу структуры должен вызывать переход к первому/последнему элементу.

- В классе <Структура> в деструкторе должно выполняться удаление всех элементов структуры.

### Требования к программе

- В основной программе должен быть реализован цикл с возможными операциями:

1. добавление элемента в структуру;
2. удаление элемента из структуры;
3. вывод структуры на экран.

- Добавление элемента в структуру должно предлагать пользователю выбрать тип создаваемого объекта: Derived1, Derived2 или Derived3. Для размещенного объекта должна быть вызвана функция Input.

- Вывод структуры на экран должен быть выполнен на базе функций GetFirst и GetNext, реализованных в классе структуры. С использованием данных функции необходимо перебирать все элементы данных в структуре и поочередно печатать их на экран. Печать каждого элемента должна выполняться через вызов функции Output. Печать структуры должна быть реализована в 2-х режимах:

* целиком вся структура данных;
* отдельно по элементам, чтобы была продемонстрирована ситуация выхода за границу структуры.

### Лабораторное задание



### Листинг

Для экономии месте приведен только код относящийся в лб№7.

class Queue

{

class queue\_element

{

public:

Interface\* element;

};

private:

int MAX\_SIZE;

int FIRST, LAST;

int flag;

queue\_element\* queue;

public:

Queue(int size)

{

MAX\_SIZE = size;

FIRST = -1;

LAST = -1;

flag = 0;

queue = new queue\_element[MAX\_SIZE];

}

~Queue()

{

delete[] queue;

}

bool full()

{

return (FIRST == 0 and LAST == MAX\_SIZE - 1) or (FIRST == LAST + 1);

}

bool empty()

{

return FIRST == -1;

}

void push(int type)

{

if (full())

{

cout << "Очередь заполнена";

return;

}

else

{

Interface\* obj = nullptr;

if (FIRST == -1)

{

FIRST = 0;

}

LAST = (LAST + 1) % MAX\_SIZE;

switch (type)

{

case 1:

{

obj = new derived1;

obj->Input();

break;

}

case 2:

{

obj = new derived2;

obj->Input();

break;

}

case 3:

{

obj = new derived3;

obj->Input();

break;

}

}

queue[LAST].element = obj;

flag++;

}

}

void pop()

{

if (empty())

{

cout << "Очередь пуста";

return;

}

else

{

delete queue[FIRST].element;

if (LAST == FIRST)

{

FIRST = -1;

LAST = -1;

}

else

{

FIRST = (FIRST + 1) % MAX\_SIZE;

flag--;

}

}

}

Interface\* GetNext(int index)

{

return queue[(index + 1) % MAX\_SIZE].element;

}

Interface\* GetFirst()

{

return queue[FIRST].element;

}

Interface\* GetLast()

{

return queue[LAST].element;

}

void output\_queue()

{

int index = FIRST % MAX\_SIZE;

if (empty())

{

cout << "Очередь пуста" << endl;

}

else

{

cout << "Элементы в очереди" << endl;

GetFirst()->Output();

cout << endl;

while (index % MAX\_SIZE != LAST)

{

GetNext(index++)->Output();

cout << endl;

}

}

}

void output\_element()

{

int index;

if (empty())

{

cout << "Очередь пуста" << endl;

}

else

{

cout << "Введите индекс элемента очереди ";

cin >> index;

if ((index < 0) or (index > flag))

{

index = FIRST;

}

cout << "элемент стека под индексом " << index << endl;

GetNext(--index)->Output();

cout << endl;

}

}

};

**Source.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <conio.h>

#include <Windows.h>

#include <locale.h>

#include <iostream>

#include "phone.h"

int main() {

setlocale(LC\_ALL, "Russian");

Queue q(5);

cout << "Добавление в очередь" << endl;

q.push(1);

q.push(2);

q.push(3);

q.output\_queue();

q.output\_element();

q.pop();

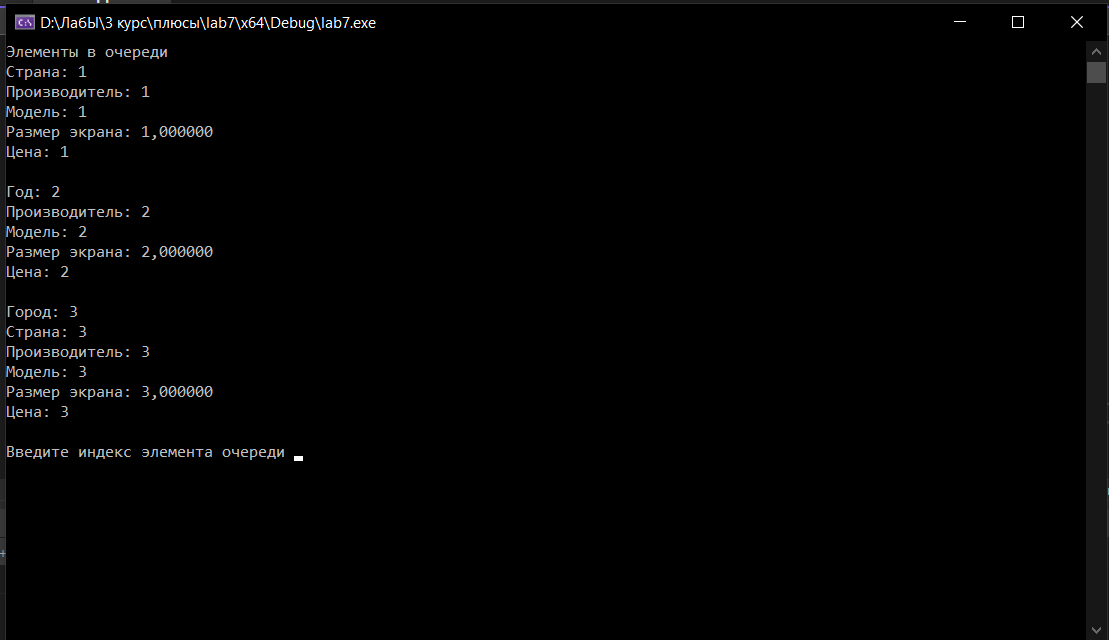
cout << "очередь после удаления элемента" << endl;

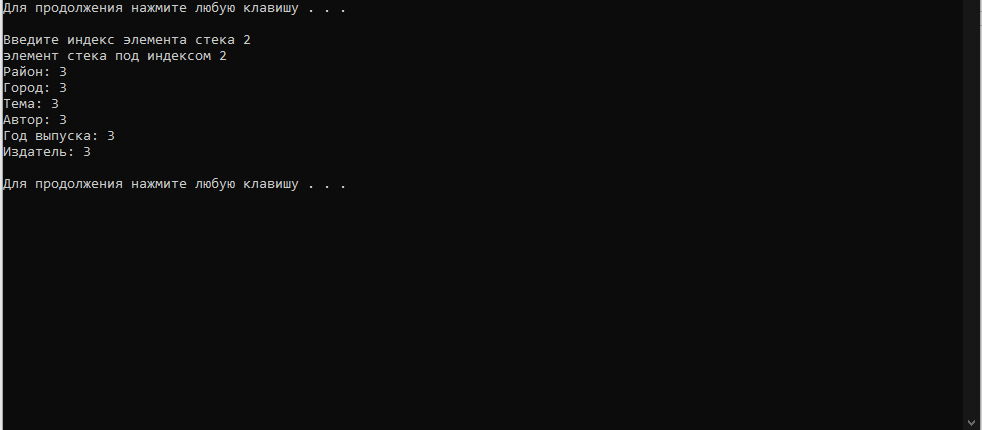
q.output\_queue();

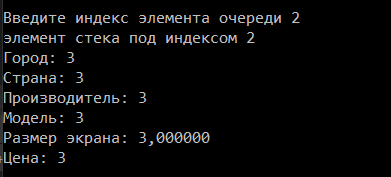
return 0;

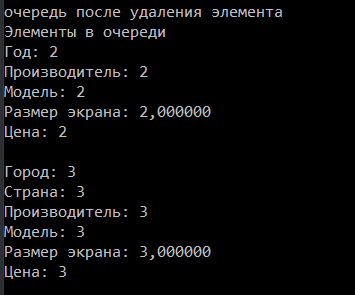
}

### Результаты работы программы

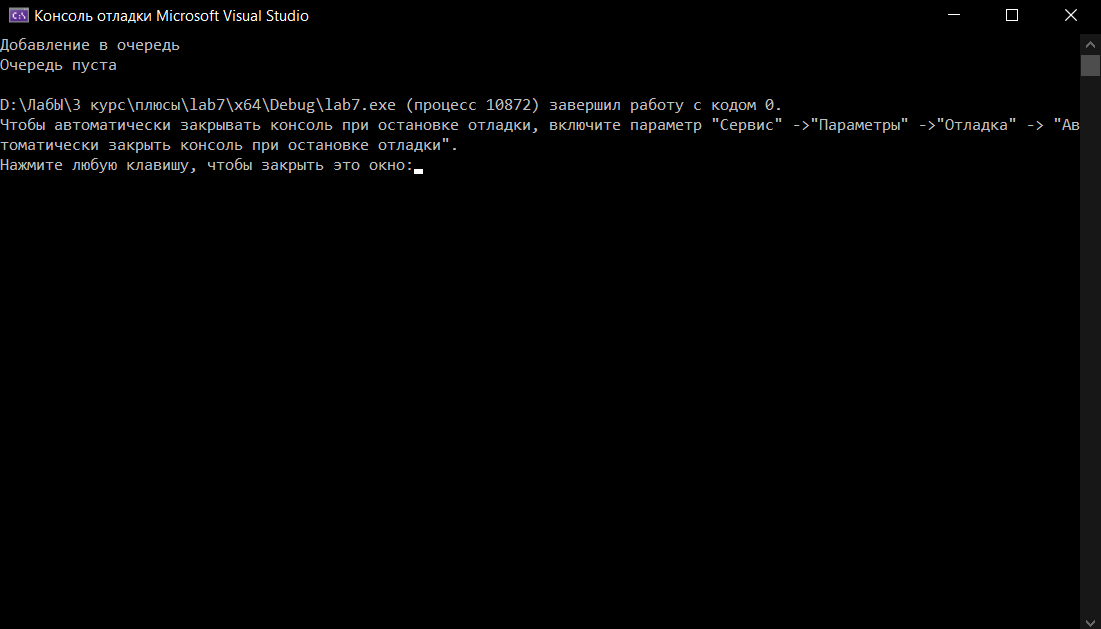




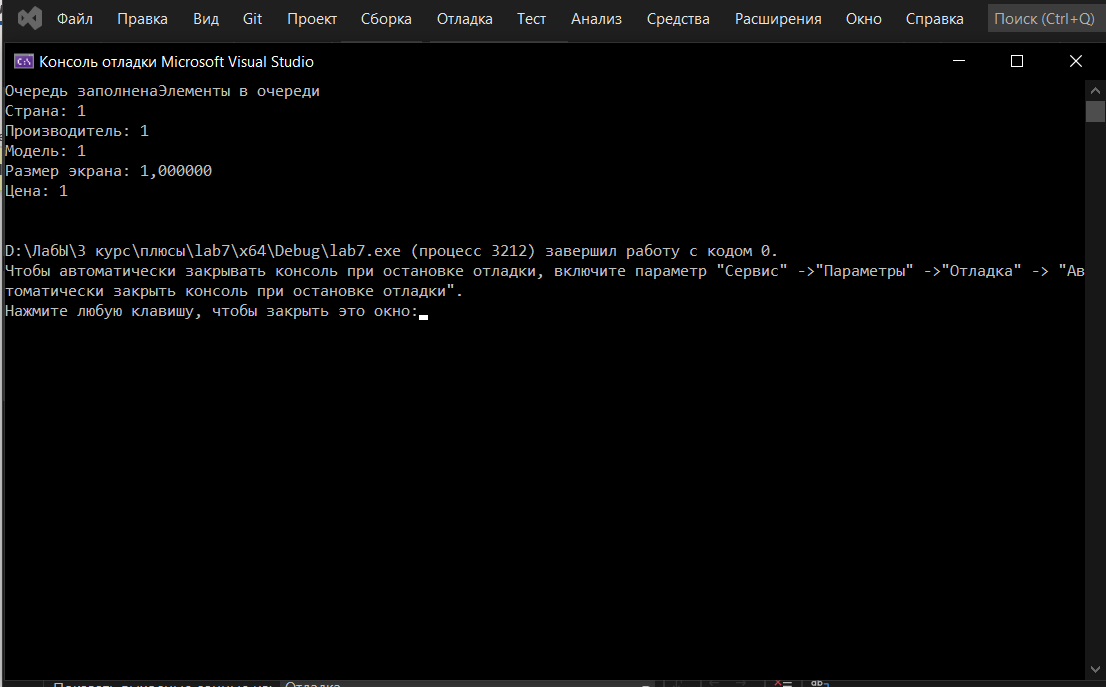




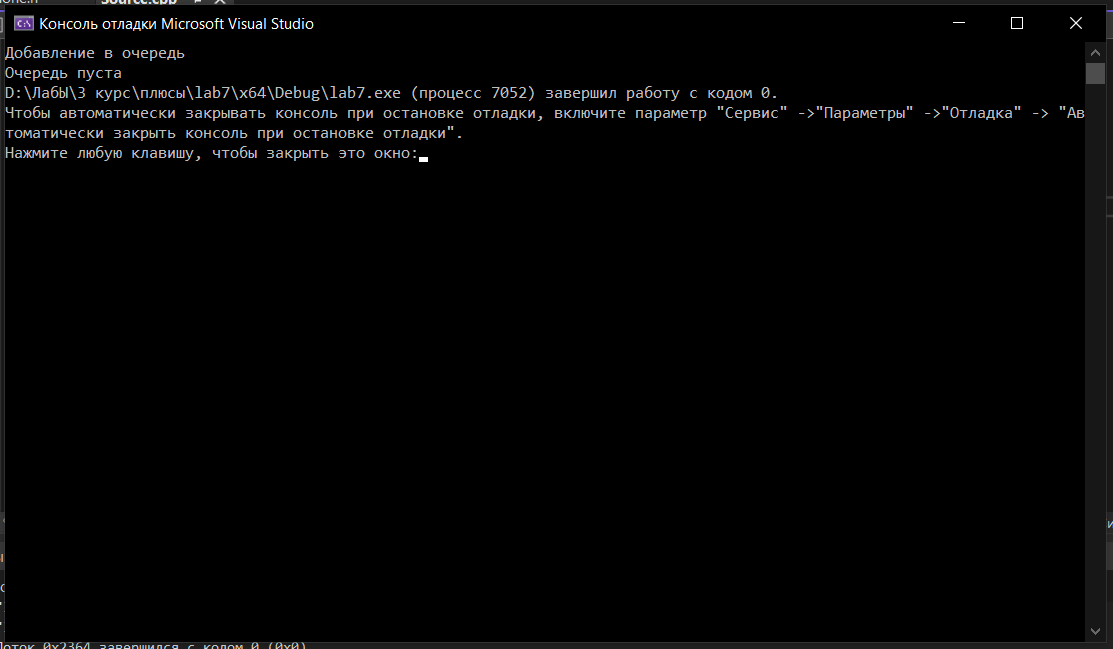
Попытка вывода пустой очереди



Попытка добавления элемента в заполненную очередь



Попытка удаления элемента из пустого стека



### Выводы

В ходе выполнения лабораторной работы были изучены механизмы виртуализации функций в языке С++, а также были приобретены навыки разработки и использования виртуальных функций и абстрактных классов.