

ГУАП

КАФЕДРА № 14

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ  
ПРЕПОДАВАТЕЛЬ

ассистент

\_\_\_\_\_  
должность, уч. степень, звание

\_\_\_\_\_  
подпись, дата

Н.Ю. Чумакова

\_\_\_\_\_  
инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

НАСЛЕДОВАНИЕ В C++

по курсу: ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

1042

/ 25.02

\_\_\_\_\_  
подпись, дата

Д.А. Васейко

\_\_\_\_\_  
инициалы, фамилия

Санкт-Петербург 2022

## 1. Постановка задачи

Создать родительский класс «Очередь» с функциями инициализации очереди, добавления элемента в очередь и извлечения элемента из очереди. Создать метод создания копии очереди. Результатом должен стать новый экземпляр класса «Очередь», состоящий из элементов (копий элементов) исходно очереди. Порядок следования элементов должен быть сохранен. Создать функцию слияния двух очередей. Результатом должна быть очередь, состоящая из элементов первой и второй очереди. Порядок следования элементов должен быть сохранен. На основе родительского класса «Очередь» создать дочерний класс «Очередь1» с функциями нахождения и отображения на экране требуемого в соответствии с вариантом задания значения.

Вариант 3 – Подсчет размаха ряда элементов – разности между максимальным и минимальным .

## 2. Формализация задачи

Данная программа разбита на пять файлов: два головных файла (ptQueue.h и chdQueue.h), два реализационных файла (ptQueue.cpp и chdQueue.cpp) и один главный (main.cpp) – это файлы с объявлением, определением методов класса соответственно, а также файл с управляющей функцией.

Элемент очереди, реализованный с помощью структуры под именем cell, состоит из данных целочисленного типа, а также указателя на предыдущий такой же элемент.

Число элементов очереди (как и само количество очередей) заранее неизвестно и вводится пользователем вручную с клавиатуры. Вследствие этого реализовано динамическое выделение памяти не только для количества очередей, но и для самой очереди. Наряду с этим реализовано динамическое удаление; в базовом классе parQueue прописаны конструкторы и деструкторы.

В программе реализованы два пользовательских меню – выбор модификатора доступа производного класса (очереди), с которым хочет работать пользователь и собственно выбор действий, которые будут применены по отношению к очередям: 1 – Добавление элемента очереди; 2 – Извлечение элемента очереди; 3 – Вывод очереди на экран; 4 – Подсчет размаха ряда элементов; 5 – Создание копии очереди; 6 – Слияние двух очередей; 7 – Выбор иной очереди; 8 – С какой очередью я сейчас работаю?; 0 – Вернуться к выбору класса-наследника. Выход из программы осуществляется в меню выбора класса.

Все вышеописанные методы (удаление, добавление и т.д.) характерны для таковых при работе с очередью, поэтому алгоритм их работы здесь приведен не будет. Алгоритм же реализации метода №4 (соответствует варианту задания) будет описан ниже.

В данном методе выделяются четыре переменных: указатель на тип данных cell (равный последнему элементу очереди), целочисленный max, min, result(размах). Сначала проверяем на нуль переменную size. Если она равна нулю, то очередь пуста, иначе работаем. Пока указатель не равен нулевой строке, ищем максимальный и минимальный элементы; находим размах путем вычитания из максимума минимума.

Так как мы работаем с одним из трех классов (публичный, защищенный, приватный), то была объявлена и определена функция, принимающая шаблонный класс choozen\_queue, ничего не возвращающая.

Используя различные модификаторы доступа в наследных классах, будет меняться видимость данных и методов базового класса для производных классов. Ради этой

демонстрации в базовом классе прописаны все три спецификатора доступа: в частном доступе находится переменная элемента очереди, в защищенном доступе – размер очереди, в публичном – геттер элемента, конструктор, деструктор, прочие методы. Когда мы используем публичный класс-наследник, то все публичные методы и данные базового класса также становятся публичными и для наследника; все защищенные данные и методы – защищенными в наследнике; приватные данные и методы напрямую недоступны из базового класса. Когда мы используем защищенный класс-наследник, то все публичные и защищенные данные и методы базового класса становятся здесь защищенными; приватные данные напрямую недоступны. Когда мы используем приватный класс-наследник, то все данные и методы базового класса становятся приватными; приватные данные базового класса напрямую недоступны.

### 3. Исходный код

Файл ptQueue.h

```
#pragma once

typedef struct cell { //элемент
    int data = 0; //значение элемента
    cell* ptrPrev = nullptr; //указатель на предыдущий элемент
}cell;

class ptQueue {
private:
    cell* last = 0; //указатель на конец очереди
protected:
    int size; //размер очереди
public:
    ptQueue(); //конструктор
    ~ptQueue(); //деконструктор

    cell* get_last(); //прототип функции получения указателя на конец очереди
    void push(int elem); //прототип функции добавления элемента в очередь
    int pop(); //прототип функции удаления элемента из очереди
    void print(); //прототип функции вывода очереди на экран
    void merge(ptQueue& Q); //прототип функции слияния очередей
    void copy(ptQueue& Q); //прототип функции копирования очередей
    bool blSmthng(); //прототип функции копирования очередей
    int specFunction(); //прототип функции выполнения задачи
};
```

Файл ptQueue.cpp

```
#include<iostream>
#include"ptQueue.h"

using namespace std;

ptQueue::ptQueue() { last = 0; size = 0; } //конструктор
ptQueue::~~ptQueue() //деконструктор
{
    while (size > 0)
    {
        cell* temp = last;
        last = temp->ptrPrev;
        size--;
        delete temp;
    }
}
```

```

}

cell* ptQueue::get_last() { return last; } //getter конца очереди

void ptQueue::push(int elem) //метод добавления элемента в очередь
{
    cell* newCell = new cell;
    newCell->ptrPrev = last;
    last = newCell;
    last->data = elem;
    size++;
}

int ptQueue::pop() //метод изъятия элемента из очереди
{
    cell* temp = last;
    int tmpData = 0;

    if (temp->ptrPrev == nullptr)
    {
        tmpData = temp->data;
        delete last;
        last = nullptr;
        size = 0;
    }
    else
    {
        while (temp->ptrPrev->ptrPrev != nullptr) { temp = temp->ptrPrev; }
        tmpData = temp->ptrPrev->data;
        delete temp->ptrPrev;
        temp->ptrPrev = nullptr;
        size--;
    }
    return tmpData;
}

void ptQueue::print() //метод вывода очереди на экран
{
    cell* temp = last;
    while (temp->ptrPrev != nullptr)
    {
        cout << temp->data << "-->";
        temp = temp->ptrPrev;
    }
    cout << temp->data << endl;
}

void ptQueue::merge(ptQueue& Q) //метод слияния очереди
{
    int* buffData = new int[Q.size];
    cell* temp = Q.last;
    for (int i = Q.size - 1; i >= 0; i--)
    {
        buffData[i] = temp->data;
        temp = temp->ptrPrev;
    }
    for (int i = 0; i < Q.size; i++) { this->push(buffData[i]); }
    delete[] buffData;
}

void ptQueue::copy(ptQueue& Q) //метод копирования очереди
{
    int* buffData = new int[Q.size];
    cell* temp = Q.last;
    for (int i = Q.size - 1; i >= 0; i--)

```

```

    {
        buffData[i] = temp->data;
        temp = temp->ptrPrev;
    }
    for (int i = 0; i < Q.size; i++) { this->push(buffData[i]); }
    delete[] buffData;
}

bool ptQueue::bIsMthng() { return size == 0 ? true : false; } //метод проверки на
заполненность

int ptQueue::specFunction() //метод выполнения задачи
{
    cell* last = get_last();
    int max = get_last()->data;
    int min = get_last()->data;
    int result = 0;

    if (size == 0){cout << "Queue is clear" << endl;}
    else
    {
        while (last != nullptr)
        {
            if (last->data > max)
                max = last->data;
            if (last->data < min)
                min = last->data;
            last = last->ptrPrev;
        }
        result = max - min;
    }
    return result;
}

```

Файл chdQueue.h

```

#pragma once
#include"ptQueue.h"

class chdQueue_pri : private ptQueue //дочерний класс с модификатором приват
{
public: //те же методы, только в публице
    int specFunction();
    void push(int elem);
    int pop();
    void print();
    void merge(chdQueue_pri& Q);
    void copy(chdQueue_pri& Q);
    bool bIsMthng();
};

class chdQueue_pro : protected ptQueue //дочерний класс с модификатором протект
{
public://те же методы, только в публице
    int specFunction();
    void push(int elem);
    int pop();
    void print();
    void merge(chdQueue_pro& Q);
    void copy(chdQueue_pro& Q);
    bool bIsMthng();
};

class chdQueue_pab : public ptQueue //дочерний класс с модификатором публик

```

```
{
public: //тут не надо, и так все открыто и доступно

};
```

Файл chdQueue.cpp

```
#include <iostream>
#include "chdQueue.h"

using namespace std;

int chdQueue_pri::specFunction() { return ptQueue::specFunction(); }
int chdQueue_pri::pop() { return ptQueue::pop(); }
void chdQueue_pri::push(int elem) { return ptQueue::push(elem); }
void chdQueue_pri::print() { return ptQueue::print(); }
bool chdQueue_pri::blSmthng() { return ptQueue::blSmthng(); }
void chdQueue_pri::merge(chdQueue_pri& Q) { return ptQueue::merge(Q); }
void chdQueue_pri::copy(chdQueue_pri& Q) { return ptQueue::copy(Q); }

int chdQueue_pro::specFunction() { return ptQueue::specFunction(); }
int chdQueue_pro::pop() { return ptQueue::pop(); }
void chdQueue_pro::push(int elem) { return ptQueue::push(elem); }
void chdQueue_pro::print() { return ptQueue::print(); }
bool chdQueue_pro::blSmthng() { return ptQueue::blSmthng(); }
void chdQueue_pro::merge(chdQueue_pro& Q) { return ptQueue::merge(Q); }

void chdQueue_pro::copy(chdQueue_pro& Q) { return ptQueue::copy(Q); }
```

Файл main.cpp

```
#include<iostream>
#include<locale>
#include"chdQueue.h"
using namespace std;

template <class T>

void choozen_queue(T* queue, int qNumb)//шаблончик
{
    int c;
    int flag = 1;
    int temp = 0; //введенное значение пользователем
    int index = 0; //номер очереди, с которой работаем
    int res = 0; //переменная для выполнения пункта задания
    int count = 1; //кол-во очередей, с которыми работает пользователь
    int chosen_q; //номер выбранной очереди

    while (flag == 1)
    {
        cout << "1 - Add new element of queue(push)" << endl;
        cout << "2 - Extract element of queue (pop)" << endl;
        cout << "3 - Print queue" << endl;
        cout << "4 - Find the range" << endl;
        cout << "5 - Copy queue" << endl;
        cout << "6 - Merge queue" << endl;
        cout << "7 - Choose another queue" << endl;
        cout << "8 - Which queue is choosen?" << endl;
        cout << "0 - Choose access modifier" << endl;
        cout << "-> ";
        cin >> c;
        switch (c)
        {
            case 1:
                system("cls");
```

```

        cout << "Input ur value: ";
        cin >> temp;
        queue[index].push(temp);
        cout << "Ur value was added.\n " << endl;
        break;

case 2:
    if (queue[index].blSmthng())
    {
        cout << "Queue is free, nothing to extract.\n " << endl;
        system("pause");
        break;
    }
    else
    {
        system("cls");
        temp = queue[index].pop();
        cout << "Extract element is: " << temp << endl;
        cout << "\n";
    }
    break;

case 3: //вывод очереди на экран
    if (queue[index].blSmthng())
    {
        cout << "Queue is free, nothing to print.\n " << endl;
        system("pause");
        break;
    }
    else
    {
        system("cls");
        queue[index].print();
        cout << "\n";
    }
    break;

case 4:
    if (queue[index].blSmthng())
    {
        cout << "Queue is free, task can't be completed.\n " << endl;
        system("pause");
        break;
    }

    else
    {
        system("cls");
        res = queue[index].specFunction();
        cout << "The range is: " << res << endl;
        cout << "\n";
    }
    break;

case 5: //создание копии очереди
    if (queue[index].blSmthng())
    {
        cout << "Queue is free, nothing to copy \n " << endl;
        system("pause");
        break;
    }

    else
    {

```

```

        if (count == qNumb)
        {
            cout << "U cant add new queue, cause number of queues is
max.\n" << endl;

            system("pause");
            break;
        }
        system("cls");
        queue[count].copy(queue[index]);
        cout << "Queue has been copied. U get 2 the same queues.\n" <<
endl;

        count++;
    }
    break;
case 6:
    if (count == 1)
    {
        cout << "U have just one queue, nothing to merge.\n" << endl;
        system("pause");
        break;
    }
    else
    {
        cout << "Which queue u wanna merge?" << endl;
        cin >> chosen_q;
        if ((chosen_q < 0) || (chosen_q >= qNumb) || (chosen_q == index))
        {
            cout << "Somethig wrong!\n" << endl;
            system("pause");
            break;
        }
        if (queue[index].blSmthng())
        {
            cout << "U can't merge, cause second queue is free.\n" <<
endl;

            system("pause");
            break;
        }
        system("cls");
        queue[index].merge(queue[chosen_q]);
        cout << "Queues has been merged.\n" << endl;
    }
    break;
case 7:
    system("cls");
    cout << "Queue is choosen number>>" << index << endl;
    cout << "Input number of queue (by 0 to " << qNumb << ") , that u wanna
choose: ";

    cin >> chosen_q;
    if ((chosen_q < 0) || (chosen_q >= qNumb) || (chosen_q == index))
    {
        cout << "Something wrong!\n" << endl;
        system("pause");
    }
    else
    {
        index = chosen_q;
        system("cls");
        cout << "U switched to queue number>>" << index << endl;
        cout << "\n";
    }
    break;
case 8:

```



```

        system("cls");
        cout << "Queue is choosen number>>" << index << endl;
        cout << "\n";
        break;
    case 0:
        cout << "\n";
        flag = 0;
        break;
    }
}
}
int main()
{
    int num_q;
    int c;
    int flag = 1;
    chdQueue_pri* q1 = NULL;
    chdQueue_pro* q2 = NULL;
    chdQueue_pab* q3 = NULL;

    cout << "This is consol program for 1st lab!" << endl;
    cout << "Input number of queues ";
    cin >> num_q;
    system("cls");

    cout << "Choose, what access modifiera u wanna get" << endl;
    while (flag == 1)
    {
        cout << "1 - private" << endl;
        cout << "2 - protect" << endl;
        cout << "3 - public" << endl;
        cout << "0 - Exit" << endl;
        cout << "-> ";
        cin >> c;
        switch (c)
        {
            case 1:
                q1 = new chdQueue_pri[num_q];
                system("cls");
                choozen_queue(q1, num_q);
                delete[] q1;
                break;
            case 2:
                q2 = new chdQueue_pro[num_q];
                system("cls");
                choozen_queue(q2, num_q);
                delete[] q2;
                break;
            case 3:
                q3 = new chdQueue_pab[num_q];
                system("cls");
                choozen_queue(q3, num_q);
                delete[] q3;
                break;
            case 0:
                flag = 0;
                break;
            default:
                system("cls");
                cout << "Somethig wrong, try again!" << endl;
                break;
        }
    }
}

```

```

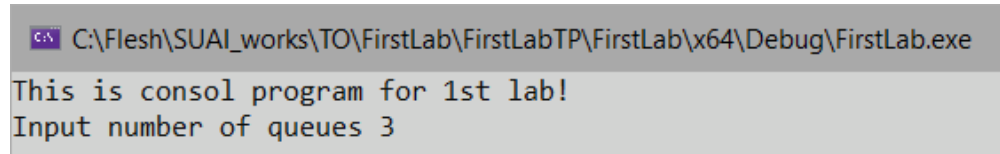
    }
    }
    return 0;
}

```

## 1. Результаты работы программы

Для демонстрации результатов работы программы введем последовательно очередь: 1->2->3->3->4->5. Учитывая особенности работы с очередью, впоследствии будет удален первый введенный элемент – 5. Размах данной очереди:  $4 - 1 = 4$ ; Проверим это:

При включении программы, нам предлагают ввести количество очередей. Их будет три для демонстрации работы метода копирования и слияния (Рис. 1)



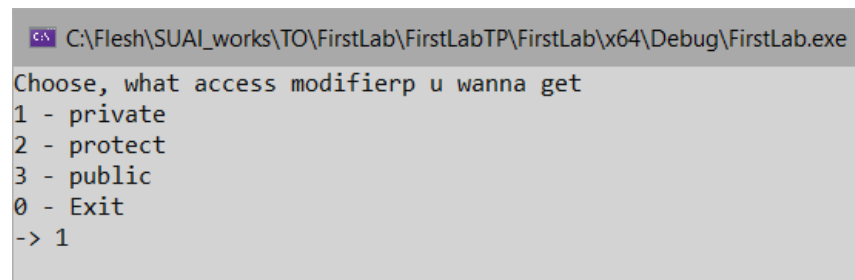
```

C:\Flesh\SUAI_works\TO\FirstLab\FirstLabTP\FirstLab\x64\Debug\FirstLab.exe
This is consol program for 1st lab!
Input number of queues 3

```

Рисунок 1 – Выбор количества очередей.

Далее программа предложит пользователю, с каким модификатором доступа работать (Рис. 2)



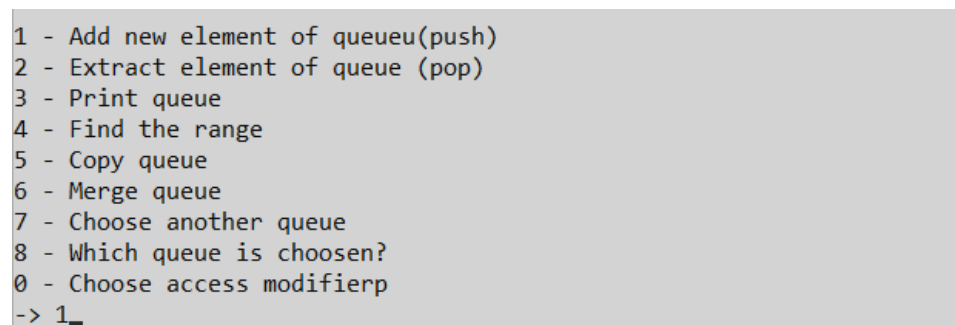
```

C:\Flesh\SUAI_works\TO\FirstLab\FirstLabTP\FirstLab\x64\Debug\FirstLab.exe
Choose, what access modifierp u wanna get
1 - private
2 - protect
3 - public
0 - Exit
-> 1

```

Рисунок 2 – Выбор модификатора доступа.

По выборе модификатора, перед нами появится основное меню работы. Начнем вносить элементы (Рис. 3)



```

1 - Add new element of queueu(push)
2 - Extract element of queue (pop)
3 - Print queue
4 - Find the range
5 - Copy queue
6 - Merge queue
7 - Choose another queue
8 - Which queue is choosen?
0 - Choose access modifierp
-> 1

```

Рисунок 3 – Основное меню пользователя.

Мы ввели очередь, указанную выше. Теперь удалим первый ее элемент (Рис. 4)

```
C:\Flesh\SUAL_works\TO\FirstLab\FirstLabTP\FirstLab\x64\Debug\FirstLab.exe
Extract element is: 5

1 - Add new element of queueu(push)
2 - Extract element of queue (pop)
3 - Print queue
4 - Find the range
5 - Copy queue
6 - Merge queue
7 - Choose another queue
8 - Which queue is choosen?
0 - Choose access modifierp
->
```

Рисунок 4 – Извлеченный элемент.

Давайте проверим, как выглядит наша очередь теперь (Рис. 5)

```
C:\Flesh\SUAL_works\TO\FirstLab\FirstLabTP\FirstLab\x64\Debug\FirstLab.exe
1-->2-->3-->3-->4

1 - Add new element of queueu(push)
2 - Extract element of queue (pop)
3 - Print queue
4 - Find the range
5 - Copy queue
6 - Merge queue
7 - Choose another queue
8 - Which queue is choosen?
0 - Choose access modifierp
->
```

Рисунок 5 – Вывод очереди на экран.

Найдем элемент, согласно поставленному условию в задании работы (Рис. 6)

```
C:\Flesh\SUAL_works\TO\FirstLab\FirstLabTP\FirstLab\x64\Debug\FirstLab.exe
The range is: 3

1 - Add new element of queueu(push)
2 - Extract element of queue (pop)
3 - Print queue
4 - Find the range
5 - Copy queue
6 - Merge queue
7 - Choose another queue
8 - Which queue is choosen?
0 - Choose access modifierp
-> █
```

Рисунок 6 – Нахождение размаха.

Создадим копию введенной очереди (Рис. 7)

```
C:\Flesh\SUAL_works\TO\FirstLab\FirstLabTP\FirstLab\x64\Debug\FirstLab.exe
Queue has been copied. U get 2 the same queues.

1 - Add new element of queueu(push)
2 - Extract element of queue (pop)
3 - Print queue
4 - Find the range
5 - Copy queue
6 - Merge queue
7 - Choose another queue
8 - Which queue is choosen?
0 - Choose access modifierp
->
```

Рисунок 7 – Создание копии очереди.

Произведем слияние двух очередей, равных между собой (Рис. 8)

```
C:\Flesh\SUAL_works\TO\FirstLab\FirstLabTP\FirstLab\x64\Debug\FirstLab.exe
Queues has been merged.

1 - Add new element of queueu(push)
2 - Extract element of queue (pop)
3 - Print queue
4 - Find the range
5 - Copy queue
6 - Merge queue
7 - Choose another queue
8 - Which queue is choosen?
0 - Choose access modifierp
-> █
```

Рисунок 8 – Слияние двух очередей.

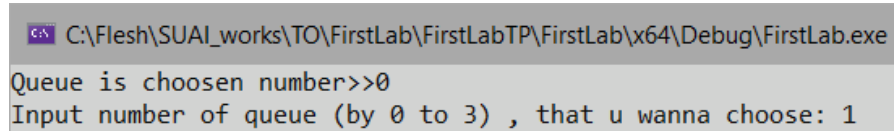
Давайте выведем получившуюся очередь на экран (Рис. 9)

```
C:\Flesh\SUAL_works\TO\FirstLab\FirstLabTP\FirstLab\x64\Debug\FirstLab.exe
1-->2-->3-->3-->4-->1-->2-->3-->3-->4

1 - Add new element of queueu(push)
2 - Extract element of queue (pop)
3 - Print queue
4 - Find the range
5 - Copy queue
6 - Merge queue
7 - Choose another queue
8 - Which queue is choosen?
0 - Choose access modifierp
->
```

Рисунок 9 – Вывод слившихся очередей.

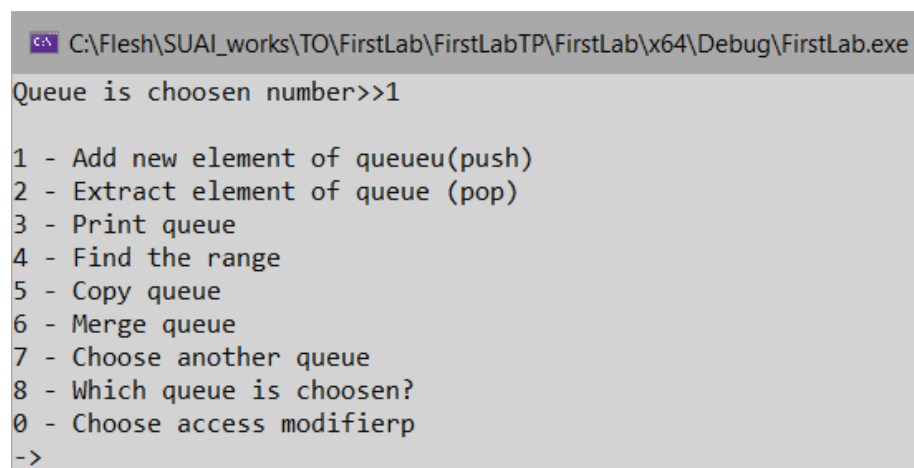
В программе есть возможность сменить номер очереди (Рис. 10)



```
C:\Flesh\SUAI_works\TO\FirstLab\FirstLabTP\FirstLab\x64\Debug\FirstLab.exe
Queue is choosen number>>0
Input number of queue (by 0 to 3) , that u wanna choose: 1
```

Рисунок 10 – Выбор иной очереди.

В программе есть возможность узнать номер очереди, с которой мы сейчас работаем (Рис. 11)



```
C:\Flesh\SUAI_works\TO\FirstLab\FirstLabTP\FirstLab\x64\Debug\FirstLab.exe
Queue is choosen number>>1
1 - Add new element of queueu(push)
2 - Extract element of queue (pop)
3 - Print queue
4 - Find the range
5 - Copy queue
6 - Merge queue
7 - Choose another queue
8 - Which queue is choosen?
0 - Choose access modifierp
->
```

Рисунок 11 – Номер очереди, с которой работает пользователь.

## 2. Выводы

В процессе выполнения лабораторной работы мы изучили различные модификаторы доступа при наследовании классов от одного базового, и то, как эти модификаторы влияют на работу с этими классами-наследниками. При выполнении тестов программа доказала свою правильную работоспособность.