ГУАП

КАФЕДРА № 14

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Ассистент		Н.Ю. Чумакова
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

НАСЛЕОВАНИЕ В C++

по курсу: ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 1042		Н. Ю. Лега
	подпись, дата	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 23 – Подсчет суммы четных элементов

2. Формализация задачи

Программа разбита на пять файлов: два файла с определениями (Queue.h и Queue1.h), два реализационных файла (Queue.cpp и Queue1.cpp) и один главный (main.cpp). Элемент очереди, реализованный с помощью структуры под именем Unit, состоит из данных целочисленного типа, а также указателя на предыдущий такой же элемент.

Число элементов очереди заранее неизвестно и вводится пользователем с клавиатуры. Поэтому реализовано динамическое выделение памяти не только для количества очередей, но и для самой очереди. Также реализовано динамическое удаление; в базовом классе parQueue прописаны конструкторы и деструкторы.

Реализованы два пользовательских меню – выбор модификатора доступа производного класса, с которым хочет работать пользователь и выбор действий, которые будут применены по отношению к очередям: 1 – Добавление элемента очереди; 2 – Извлечение элемента очереди; 3 – Вывод очереди на экран; 4 – Нахождение суммы четных 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 - в меню выбора класса.

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

В данном методе выделяются две переменные: указатель на тип данных Unit (равный последнему элементу очереди), целочисленный sum(сумма элементов). Сначала проверяем на нуль переменную size. Если она равна нулю, то очередь пуста, иначе работаем. Пока указатель не равен нулевой строке, проверяем элемент на четность, если он четный, то прибавляем к сумме. Далее возвращаем результат суммы. Так как мы работаем с одним из трёх классов (публичный, защищённый, приватный), то была объявлена и определена функция, принимающая шаблонный класс chosen_class, ничего не возвращающая.

Используя различные модификаторы доступа в наследных классах, будет меняться видимость данных и методов базового класса для производных классов. Чтобы это продемонстрировать, 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 касса становятся приватными; приватные данные базового класса напрямую недоступны.

3. **Исходный код**

Файл Queue.h

```
typedef struct Unit //элемент очереди
{
	int data = 0; //значение
	Unit* prev = 0; //указатель на предыдущий элемент
} Unit;
class parQueue  //родительский класс
{
private:
	Unit* last = 0; //указатель на "конец" очереди
protected:
	int size; //размер очереди
public:
	parQueue(); //конструктор без параметров
	~parQueue(); //деструктор

	Unit* get_last(); //объявление функции взятия конца очереди

	int pop(); //удаление элемента очереди
	void push(int el); //добавление элемента очереди
	void print(); //вывод на консоль очереди
	void merge(parQueue& Q1); //слияние двух очередей
	void copy(parQueue& Q); //копирование очереди
	bool isEmpty(); //проверка на заполненность

};
```

Файл Queue.cpp

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

parQueue::parQueue() //конструктор - инициализация размер и последнего элемент
{
	size = 0;
	last = nullptr;
}
```

```
parQueue::~parQueue() //деструктор
{
	while (size > 0) //пока размер > 0
	{
		Unit* temp = last; //временной переменной присваиваем значение
последнего
		last = temp->prev; //последняя становится "предыдущей"
		delete temp; //удаляем последнюю
		size--; //уменьшаем размер
	}
}
Unit* parQueue::get_last()  //возвр. последний элемент
{
	return last;
}
int parQueue::pop()
{
	setlocale(LC_ALL, "Rus");
	Unit* tmp = last;
	int deldata = 0;

	if (tmp->prev == nullptr) //если в очереди один элемент (указатель на предыдущий
элемент нулевой), то
	{
		deldata = tmp->data;
		delete last; //удаляем "последний" элемент
		last = nullptr; //и присваиваем ему ноль
		size--; //уменьшаем размер
	}
	else
	{	// в последнем содержится указатель на "следующий" элемент очереди, но
не на последний
		while (tmp->prev->prev != nullptr) //идем по элементам до "второго" в
очереди
		{
			tmp = tmp->prev;
		}
		deldata = tmp->prev->data;
		delete tmp->prev;
		tmp->prev = nullptr;
		size--;
	}
	return deldata;
}
void parQueue::push(int el)  //добавление элемента
{
	Unit* new_unit = new Unit; //выделяем память под новый элемент
	new_unit->prev = last; //указатель нового элемента указывает на последний в
очереди
	last = new_unit; //теперь новый элемент стал последним
	last->data = el; //присваиваем данные
	size++;
}
void parQueue::print() //вывод
{
	Unit* tmp = last;
```

```
	while (tmp->prev != nullptr)
	{
		cout << tmp->data << " --> "; //выводим значения (с конца), пока указатель на следующий элемент не будет нулевым
		tmp = tmp->prev;
	}
	cout << tmp->data << endl; //вывели последний элемент
}
void parQueue::copy(parQueue& Q) // копирование очереди
{
	int* buff = new int[Q.size]; //создаем буферный массив
	Unit* el_queue = Q.last; //создаем переменную очереди

	for (int i = Q.size - 1; i >= 0; i--)
	{
		buff[i] = el_queue->data; //во временный массив заносим данные от "конца" текущей очереди
		el_queue = el_queue->prev; //переходим дальше по очереди
	}

	for (int i = 0; i < Q.size; i++)
	{
		this->push(buff[i]);
	}

	delete[] buff;
}
void parQueue::merge(parQueue& Q1) //слияние 2х очередей
{
	int* buff1 = new int[Q1.size];
	Unit* el_queue1 = Q1.last;

	for (int i = Q1.size - 1; i >= 0; i--)
	{
		buff1[i] = el_queue1->data;
		el_queue1 = el_queue1->prev;
	}

	for (int i = 0; i < Q1.size; i++)
	{
		this->push(buff1[i]);
	}
	delete[] buff1;
	cout << "Слияние успешно завершено!" << endl;
}

bool parQueue::isEmpty() //проверка
{
	if (size == 0)
	{
		return true;
	}
	else
	{
		return false;
	}
}
```

Файл Queue1.h

```
#include "Queue.h"

class sonQueue_private : private parQueue
{
public:
	int function23();
	int pop(); //удаление элемента очереди
	void push(int el); //добавление элемента очереди
	void print(); //вывод на консоль очереди
	void merge(sonQueue_private& Q1); //слияние двух очередей
	void copy(sonQueue_private& Q); //копирование очереди
	bool isEmpty(); //проверка на заполненность
};

class sonQueue_protected : protected parQueue
{
public:
	int function23();
	int pop(); //удаление элемента очереди
	void push(int el); //добавление элемента очереди
	void print(); //вывод на консоль очереди
	void merge(sonQueue_protected& Q1); //слияние двух очередей
	void copy(sonQueue_protected& Q); //копирование очереди
	bool isEmpty(); //проверка на заполненность
};

class sonQueue_public : public parQueue
{
public:
	int function23();

};
```

Файл Queue1.cpp

```
#include <iostream>
#include "Queue1.h"
using namespace std;

int sonQueue_private::function23()
{
	Unit* last = get_last();  //указатель на тип данных
	int sum = 0; //инициализирую сумму

	if (size == 0)
	{
		cout << "Очередь пуста!" << endl;
	}
	else
	{
		while (last != nullptr)
		{
			if (last->data % 2 == 0) {
				sum += last->data;
			}
			last = last->prev;
		}
	}
```

```
	return sum;
}
int sonQueue_private::pop() { return parQueue::pop(); }
void sonQueue_private::push(int el) { return parQueue::push(el); }
void sonQueue_private::print() { return parQueue::print(); }
void sonQueue_private::merge(sonQueue_private& Q1) { return parQueue::merge(Q1); }
void sonQueue_private::copy(sonQueue_private& Q1) { return parQueue::copy(Q1); }
bool sonQueue_private::isEmpty() { return parQueue::isEmpty(); }

int sonQueue_protected::function23()
{
	Unit* last = get_last();  //указатель на тип данных
	int sum = 0; //инициализирую сумму

	if (size == 0)
	{
		cout << "Очередь пуста!" << endl;
	}
	else
	{
		while (last != nullptr)
		{
			if (last->data % 2 == 0)
				sum = sum + last->data;
		}
	}
	return sum;
}
int sonQueue_protected::pop() { return parQueue::pop(); }
void sonQueue_protected::push(int el) { return parQueue::push(el); }
void sonQueue_protected::print() { return parQueue::print(); }
void sonQueue_protected::merge(sonQueue_protected& Q1) { return parQueue::merge(Q1); }
void sonQueue_protected::copy(sonQueue_protected& Q1) { return parQueue::copy(Q1); }
bool sonQueue_protected::isEmpty() { return parQueue::isEmpty(); }

int sonQueue_public::function23()
{
	Unit* last = get_last();  //указатель на тип данных
	int sum = 0; //инициализирую сумму

	if (size == 0)
	{
		cout << "Очередь пуста!" << endl;
	}
	else
	{
		while (last != nullptr)
		{
			if (last->data % 2 == 0)
				sum = sum + last->data;
		}
	}
	return sum;  }
```

Файл main.cpp

```
#include <iostream>
```

```
#include <locale>
#include "Queue.h"
#include "Queue1.h"

using namespace std;
template <class T>
void chosen_class(T* q, int n_q) //шаблонная функция для  работы с классом
{
	setlocale(LC_ALL, "Rus");
	int c;
	int flag = 1;
	int value; //введенное значение пользователем
	int index = 0; //номер очереди, с которой работаем
	int count = 1; //кол-во очередей, с которыми работает пользователь
	int res4 = 0; //переменная для выполнения пункта задания
	int chosen_q; //номер выбранной очереди

	while (flag == 1)
	{
		cout << "1 - Добавление элемента очереди" << endl;
		cout << "2 - Извлечение элемента очереди" << endl;
		cout << "3 - Вывод очереди на экран" << endl;
		cout << "4 - Нахождение суммы четных элементов" << endl;
		cout << "5 - Создание копии очереди" << endl;
		cout << "6 - Слияние двух очередей" << endl;
		cout << "7 - Выбор иной очереди" << endl;
		cout << "8 - С какой очередью я сейчас работаю?" << endl;
		cout << "0 - Вернуться к выбору класса-наследника" << endl;
		cout << "-> ";
		cin >> c;

		switch (c)
		{
		case 1:
			system("cls");
			cout << "Введите значение: ";
			cin >> value;
			q[index].push(value);
			cout << "Значение добавлено в очередь.\n " << endl;
			break;
		case 2:
			if (q[index].isEmpty())
			{
				cout << "Очередь пуста, извлекать нечего.\n " << endl;
				system("pause");
				break;
			}
			else
			{
				system("cls");
				value = q[index].pop();
				cout << "Извлеченный элемент: " << value << endl;
				cout << "\n";
			}
			break;
		case 3:
			if (q[index].isEmpty())
```

```
		{
			cout << "Очередь пуста, выводить нечего.\n" << endl;
			system("pause");
			break;
		}
		else
		{
			system("cls");
			q[index].print();
			cout << "\n";
		}
		break;
	case 4:
		if (q[index].isEmpty())
		{
			cout << "Очередь пуста, задание выполнить невозможно.\n"
<< endl;
			system("pause");
			break;
		}
		else
		{
			system("cls");
			res4 = q[index].function19();
			cout << "Сумма четных элементов: " << res4 << endl;
			cout << "\n";
		}
		break;
	case 5:
		if (q[index].isEmpty())
		{
			cout << "Очередь пуста, копировать нечего.\n" << endl;
			system("pause");
			break;
		}
		else
		{
			if (count == n_q)
			{
				cout << "Невозможно создать копию очереди, так
как количество очередей достигло максимума.\n" << endl;
				system("pause");
				break;
			}

			system("cls");
			q[count].copy(q[index]);
			cout << "Очередь успешно скопирована.\n" << endl;
			count++;   }
		break;
	case 6:
		if (count == 1)
		{
			cout << "Существует только одна очередь.\n" << endl;
			system("pause");
			break;  }
		else
```

```
                {
                        cout << "С какой очередью произвести слияние?" << endl;
                        cin >> chosen_q;

                        if ((chosen_q < 0) || (chosen_q >= n_q) || (chosen_q == index))
                        {
                                cout << "Некорректное значение.\n" << endl;
                                system("pause");
                                break;  }

                        if (q[chosen_q].isEmpty())
                        {
                                cout << "Невозможно произвести слияние, так как
вторая очередь пуста.\n" << endl;
                                system("pause");
                                break;
                        }
                        system("cls");
                        q[index].merge(q[chosen_q]);

                        cout << "\n";   }
                break;
        case 7:
                system("cls");
                cout << "Сейчас вы работаете с очередью №" << index << endl;
                cout << "Введите номер очереди (от 0 до " << n_q - 1 << ") , на
которую хотите переключиться: ";
                cin >> chosen_q;

                if ((chosen_q < 0) || (chosen_q >= n_q) || (chosen_q == index))
                {
                        cout << "Некорректное значение или количество очередей
превышено.\n" << endl;
                        system("pause");
                }
                else
                {
                        index = chosen_q;
                        count++;
                        system("cls");
                        cout << "Вы переключились на очередь №" << index << endl;
                        cout << "\n";
                }
                break;
        case 8:
                system("cls");
                cout << "Вы сейчас работаете с очередью №" << index << endl;
                cout << "\n";
                break;
        case 0:
                cout << "\n";
                flag = 0;
                break;   }
    }
}
int main()  {
    setlocale(LC_ALL, "Rus");
```

```
int num_q; //переменная количества очередей
int c;
int flag = 1;

sonQueue_private* q1 = NULL;
sonQueue_protected* q2 = NULL;
sonQueue_public* q3 = NULL;

cout << "Введите колчество очередей: ";
cin >> num_q;
system("cls");

cout << "Выберите, с каким классом-наследником Вы будете работать: " << endl;

while (flag == 1)
{
	cout << "1 - private" << endl;
	cout << "2 - protected" << endl;
	cout << "3 - public" << endl;
	cout << "0 - Выход из программы" << endl;
	cout << "-> ";
	cin >> c;

	switch (c)
	{
	case 1:
		q1 = new sonQueue_private[num_q];
		system("cls");
		chosen_class(q1, num_q);
		delete[] q1;
		break;
	case 2:
		q2 = new sonQueue_protected[num_q];
		system("cls");
		chosen_class(q2, num_q);
		delete[] q2;
		break;
	case 3:
		q3 = new sonQueue_public[num_q];
		system("cls");
		chosen_class(q3, num_q);
		delete[] q3;
		break;
	case 0:
		flag = 0;
		break;
	default:
		system("cls");
		cout << "Некорректный ввод!" << endl;
		break;
	}
}
return 0;  }
```

4. Результаты работы программы

Для демонстрации результатов работы программы введем последовательно очередь: 9 -> 4 -> 1 -> 3 -> 8 -> 5. Учитывая особенности

работы с очередью, впоследствии будет удален первый введенный элемент – 5. Сумма четных элементов: 4 + 8 = 12. Проверим это:

При включении программы, нам предлагают ввести количество очередей. Их будет две для демонстрации работы метода копирования и слияния:

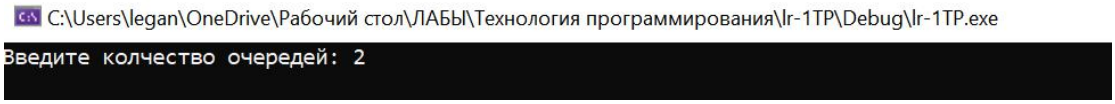

Рисунок 1 - Выбор количества очередей

Далее выбираем, с каким модификатором доступа работать:

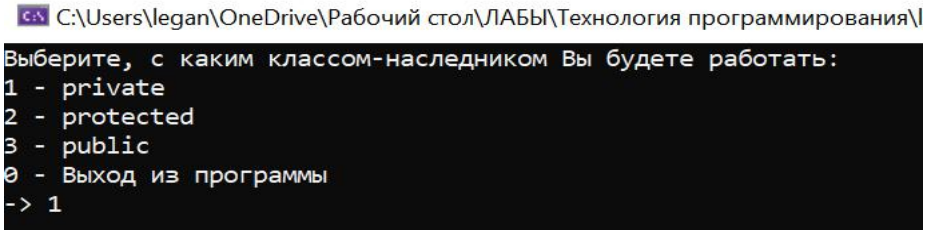

Рисунок 2 - Выбор модификатора доступа.

После выбора модификатора, перед нами появится основное меню работы. Начнем вносить элементы:

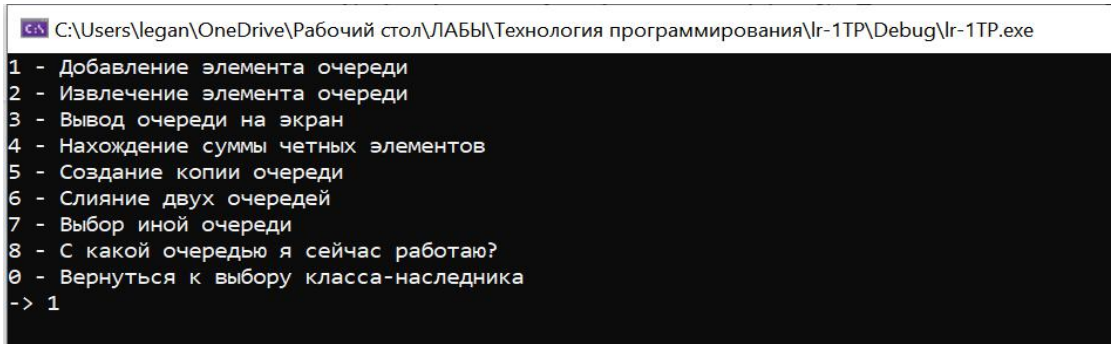

Рисунок 3 - Основное меню

Мы ввели очередь. Теперь удалим элемент :

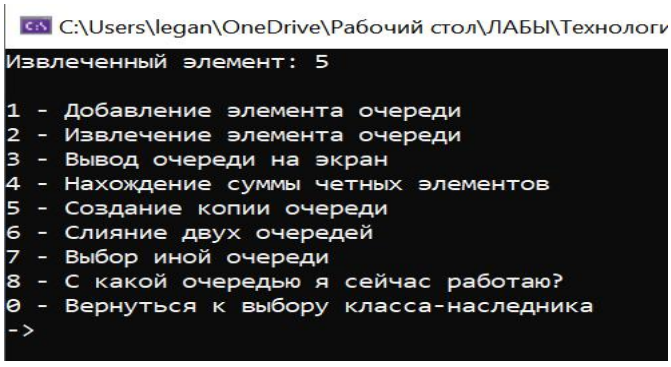

Рисунок 4 - Извлеченный элемент

Проверим вывод очереди:

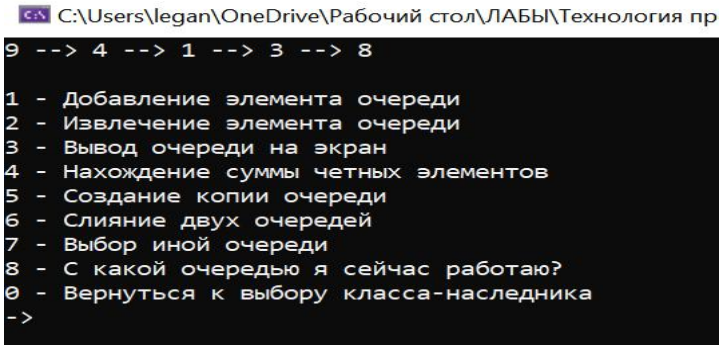

Рисунок 5 - Вывод очереди

Найдем элемент, согласно поставленному условию в задании работы:

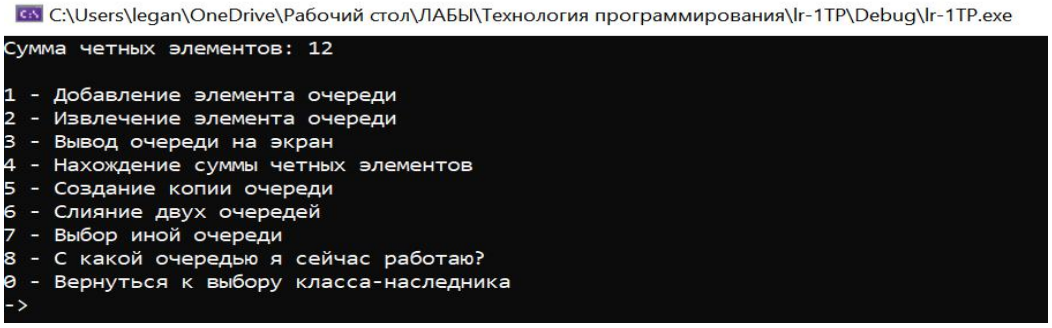

Рисунок 6 - Нахождение суммы

Создадим копию введенной очереди :

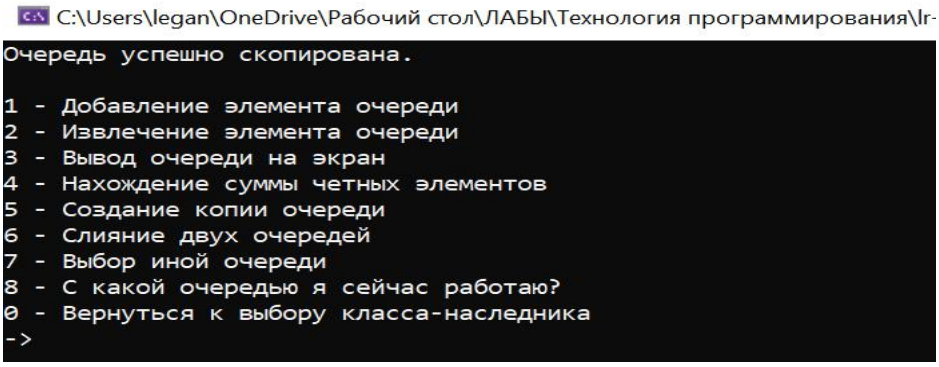

Рисунок 7 - Копия очереди

Слияние двух очередей:

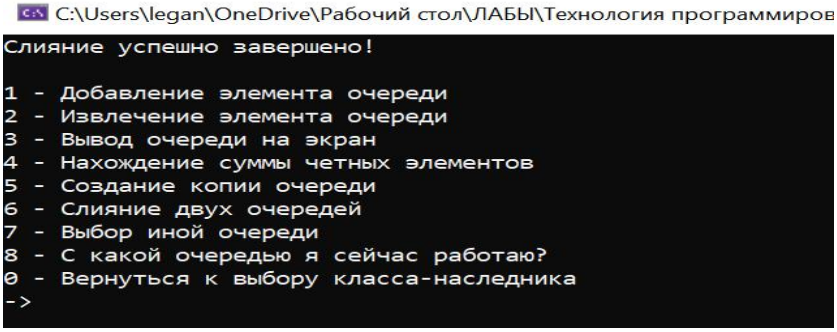

Рисунок 8 - Слияние очередей

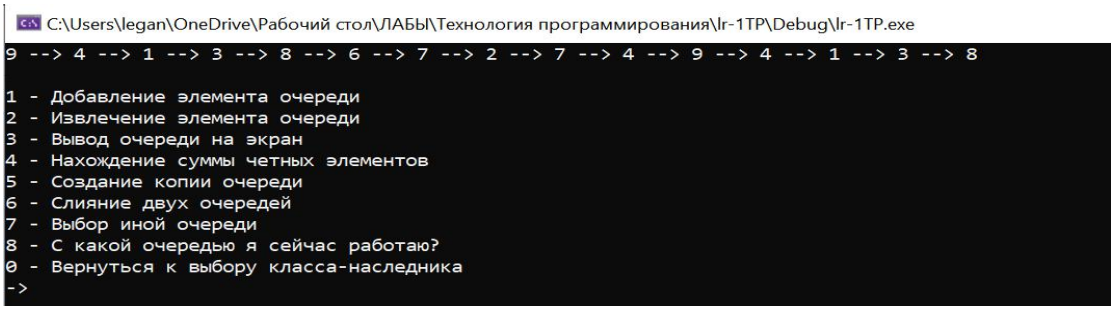

Рисунок 8 - Вывод слияния очередей

В программе есть возможность сменить номер очереди:

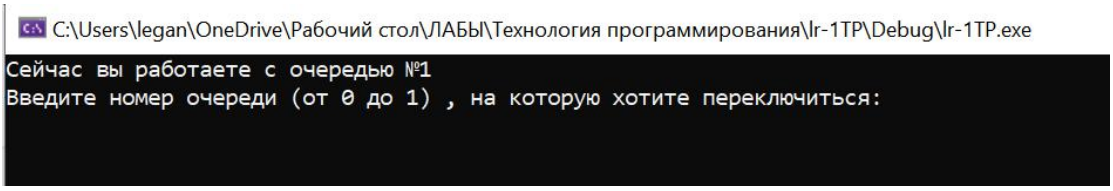

Рисунок 9 - Смена номер очереди

5. 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.