ГУАП

КАФЕДРА № 14

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | Н.Ю. Чумакова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1 |
| НАСЛЕДОВАНИЕ В C++ |
| по курсу: ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТГР. № | 1042 |  |  |  | A.А. Мамонтова |
|  |  |  | 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) исходной очереди. Порядок следования элементов должен быть сохранен. Создать функцию слияния двух очередей. Результатом должна быть очередь, состоящая элементов первой очереди 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 14 - Подсчет среднего арифметического для элементов с нечетным порядковым номером

1. **Формализация задачи**

Данная программа разбита на пять файлов: два головных файла (Queue.h иQueue1.h), два реализационных файла (Queue.cpp иQueue1.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 Unit, 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 было реализовано 2 меню: меню выбора модификатора доступа для дочерних классов и меню, включающее в себя список действий(функций), которые пользователь может применять к очередям. А именно: 1 – Добавление элемента очереди, 2 – Извлечение элемента очереди, 3 –Вывод очереди на экран, 4 – Подсчет среднего арифметического для элементов с нечетным порядковым номером(задание по варианту), 5 – Создание копии очереди, 6 – Слияние двух очередей, 7 – Выбор иной очереди, 8 – «С какой очередью я сейчас работаю?», 0 – Вернутся к выбору класса наследника.

Добавление, удаление и вывод очереди на экран реализованы методами push(in el), pop() и print(), соответственно. Данные методы характерны для таковых при работе с очередью, поэтому алгоритм их работы здесь приведен не будет.

Перейдем к заданию по варианту 14. Было необходимо реализовать алгоритм вычисления среднего арифметического элементов, находящихся на нечетных позициях. В данном алгоритме было объявлено и определено 5 переменных: переменная-указатель типа Unit равная последнему элементу очереди, целочисленные arf(среднее арифметическое - результат), sum – сумма элементов на нечетных позициях, k – переменная для определения четности-нечетности позиции, сnt – переменная-счетчик элементов на нечетных позициях для дальнейшего вычисления среднего арифметического. Итак, если список не пуст(size!=0), мы переходим к циклу while, который выполняется, пока последний элемент не будет указывать на nullptr. Если k будет нечетным, то в переменная sum увеличивается на значение текущего элемента(sum += last->data) и счетчик cnt увеличивается на 1. Затем переходим на следующий элемент и k увеличиваем на 1.

После выхода из цикла находим среднее арифметическое(сумму всех элементов, удовлетворяющих условию, делим на их количество).

Так как мы работаем с одним из трех классов (публичный, защищенный, 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. 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 касса становятся приватными; приватные данные базового класса напрямую недоступны.

1. **Исходный код**

Файл Queue.h

#pragma once

using namespace std;

typedef struct Unit// структура элемента очереди Unit

{

int data=1;//значение

Unit\* prev = 0;//указатель на предыдущий элемент

}Unit;

class parQueue//родительский класс "Очередь"

{

private:

Unit\* last = 0;//указатель на конец очереди

protected:

int size = 0;//размер очереди

public:

parQueue();

~parQueue();

Unit\* get\_last();//взятие конца очереди

void push(int el);//добавление элемента

int pop();//удаление элемента

void copy(parQueue &Q);//копирование очуреди

void merge(parQueue &Q1); //слияние двух очередей

void print();//вывод очереди на консоль

bool isEmpty(); //проверка на заполненность

};

Файл Queue.cpp

#include<iostream>

#include "Queue.h"

using namespace std;

parQueue::parQueue()

{

size = 0;

last = nullptr;

}

parQueue::~parQueue()

{

while (size > 0)

{

Unit\* tmp = last;

last = tmp->prev;

delete tmp;

size--;

}

}

Unit\* parQueue::get\_last()

{

return last;

}

void parQueue::push(int el)

{

Unit\* new\_unit = new Unit;//выделяем память под новый элемент

new\_unit->prev = last;// новый элемент хранит в себе в качестве указателя на предыдущий элемент адрес "старого" последнего элемента

last = new\_unit;//добавленный элемент становится последним

last->data = el; //присваиваем данные

size++;//увеличиваем количество элементов

}

int parQueue::pop()//удаление

{

Unit\* tmp = last;

int del = 0;//значение удаляемого элемента

if (tmp->prev == nullptr)//если в очереди только один элемент, т.е указатель на предыдущий элемент нулевой

{

del = tmp->data;

delete last;

last = nullptr;

size--;

}

else

{

while (tmp->prev->prev!=nullptr)

{

tmp = tmp->prev;

}

del = tmp->prev->data;

delete tmp->prev;

tmp->prev = nullptr;

size--;

}

return del;

}

void parQueue::copy(parQueue& Q)

{

setlocale(LC\_ALL, "Rus");

int\* buf = new int[Q.size];//создаем массив-буфер

Unit\* elQueue = Q.last;// новая переменная-очередь

for (int i=Q.size - 1; i >= 0; i--)

{

buf[i] = elQueue->data;//в буферный массив заносим элементы очереди, которая подлежит копированию

elQueue = elQueue->prev;//переходим к следующему элементу

}

for (int i = 0; i < Q.size; i++)

{

this->push(buf[i]);

}

delete[] buf;

cout << "Копирование очереди завершено!" << endl;

}

void parQueue::merge(parQueue& Q1)

{

setlocale(LC\_ALL, "Rus");

int\* buf1 = new int[Q1.size];//создаем массив-буфер

Unit\* elQueue1 = Q1.last;// новая переменная-очередь

for (int i = Q1.size - 1; i >= 0;i--)

{

buf1[i] = elQueue1->data;

elQueue1 = elQueue1->prev;

}

for (int i = 0; i < Q1.size; i++)

{

this->push(buf1[i]);

}

delete[] buf1;

cout << "Слияние очередей завершено!" << endl;

}

void parQueue::print()

{

Unit\* tmp = last;

while (tmp->prev != nullptr)

{

cout << tmp->data << " "; //значение выводятся с конца, пока указатель на предыдущий элемент не будет нулевым

tmp = tmp->prev;

}

cout << tmp->data << endl;//вывод последнего элемента отдельно, т.к. в предыдущем цикле мы не сможем вывести его из-за условия tmp->prev != nullptr

}

bool parQueue::isEmpty()

{

if (size == 0)

{

return true;

}

else

{

return false;

}

}

Файл Queue1.h

#pragma once

#include"Queue.h"

class sonQueuePublic :public parQueue

{

public:

int myfunc();//функция по заданию

};

class sonQueuePrivate :private parQueue

{

public:

int myfunc();//функция по заданию

int pop();

void push(int el);

void copy(sonQueuePrivate& Q);

void merge(sonQueuePrivate& Q1);

void print();

bool isEmpty(); //проверка на заполненность

};

class sonQueueProtected : protected parQueue

{

public:

int myfunc();//функция по заданию

int pop();

void push(int el);

void copy(sonQueueProtected& Q);

void merge(sonQueueProtected& Q1);

void print();

bool isEmpty(); //проверка на заполненность

};

Файл Queue1.cpp

#include<iostream>

#include"Queue1.h"

#include"Queue.h"

using namespace std;

//#14. Подсчет среднего арифметического для элементов с нечетным порядковым номером.

int sonQueuePrivate::myfunc()

{

setlocale(LC\_ALL, "Rus");

Unit\* last = get\_last();

int arf;//среднее арифметическое элементов с неч. порядковым номером

int sum = 0;//сумма элементов с неч. порядковым номером

int cnt = 0;//счетчик нечетных позиций

int k = 0;

if (size == 0)

{

cout << "Очередь пуста!" << endl;

}

else

while (last != nullptr)

{

if (k % 2 == 1)

{

sum += last->data;

cnt++;

}

last = last->prev;

k++;

}

arf = sum / cnt;

return arf;

}

int sonQueuePrivate::pop() { return parQueue::pop(); }

void sonQueuePrivate::push(int el) { return parQueue::push(el); }

void sonQueuePrivate::copy(sonQueuePrivate& Q1) { return parQueue::copy(Q1); }

void sonQueuePrivate::print() { return parQueue::print(); }

void sonQueuePrivate::merge(sonQueuePrivate& Q1) { return parQueue::merge(Q1); }

bool sonQueuePrivate::isEmpty() { return parQueue::isEmpty(); }

int sonQueueProtected::myfunc()

{

setlocale(LC\_ALL, "Rus");

Unit\* last = get\_last();

int arf;//среднее арифметическое элементов с неч. порядковым номером

int sum = 0;//сумма элементов с неч. порядковым номером

int cnt = 0;//счетчик нечетных позиций

int k = 0;

if (size == 0)

{

cout << "Очередь пуста!" << endl;

}

else

while (last != nullptr)

{

if (k % 2 == 1)

{

sum += last->data;

cnt++;

}

last = last->prev;

k++;

}

arf = sum / cnt;

return arf;

}

int sonQueueProtected::pop() { return parQueue::pop(); }

void sonQueueProtected::push(int el) { return parQueue::push(el); }

void sonQueueProtected::print() { return parQueue::print(); }

void sonQueueProtected::merge(sonQueueProtected& Q1) { return parQueue::merge(Q1); }

void sonQueueProtected::copy(sonQueueProtected& Q1) { return parQueue::copy(Q1); }

bool sonQueueProtected::isEmpty() { return parQueue::isEmpty(); }

int sonQueuePublic::myfunc()

{

setlocale(LC\_ALL, "Rus");

Unit\* last = get\_last();

int arf;//среднее арифметическое элементов с неч. порядковым номером

int sum = 0;//сумма элементов с неч. порядковым номером

int cnt = 0;//счетчик нечетных позиций

int k = 0;

if (size == 0)

{

cout << "Очередь пуста!" << endl;

}

else

while (last != nullptr)

{

if (k % 2 == 1)

{

sum += last->data;

cnt++;

}

last = last->prev;

k++;

}

arf = sum / cnt;

return arf;

}

Файл main.cpp

#include<iostream>

#include <locale>

#include "Queue.h"

#include "Queue1.h"

using namespace std;

//#14. Подсчет среднего арифметического для элементов с нечетным порядковым номером.

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;//значение элемента, который введет пользователь

int index = 0;//номер очереди, с которой работаем

int count = 1; //кол-во очередей, с которыми работает пользователь

int res = 0; //переменная

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 - Подсчет среднего арифметического для элементов с нечетным порядковым номером" << 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 << "Введите номер желаемого пункта -> ";

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 уже 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");

res = q[index].myfunc();

cout << "Среднее арифметическое для элементов с нечетным порядковым номером: " << res << endl;

cout << "\n";

}

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. Теперь существует две равных очереди.\n" << endl;

count++; //теперь у нас есть как минимум две очереди с индексами 0 и 1...

}

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]);

//теперь существует как минимум три очереди с индексами 0, 1, 2...

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;

sonQueuePrivate\* q1 = NULL;

sonQueueProtected\* q2 = NULL;

sonQueuePublic\* q3 = NULL;

cout << "Введите количество очередей: " << endl;

cin >> num\_q;

system("cls");

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

while (flag == 1)

{

cout << "1 - Приватный" << endl;

cout << "2 - Защищенный" << endl;

cout << "3 - Публичный" << endl;

cout << "0 - Выход из программы" << endl;

cout << "-> ";

cin >> c;

switch (c)

{

case 1:

q1 = new sonQueuePrivate[num\_q];

system("cls");

chosen\_class(q1, num\_q);

delete[] q1;

break;

case 2:

q2 = new sonQueueProtected[num\_q];

system("cls");

chosen\_class(q2, num\_q);

delete[] q2;

break;

case 3:

q3 = new sonQueuePublic[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;

}

1. **Результаты работы программы**

Итак, запускаем программу. Для начала нужно ввести количество очередей(Рис. 1). Введем 3 очереди, т.к. у нас будет 2 очереди, а 3 очередь будет результатом слияния двух других.

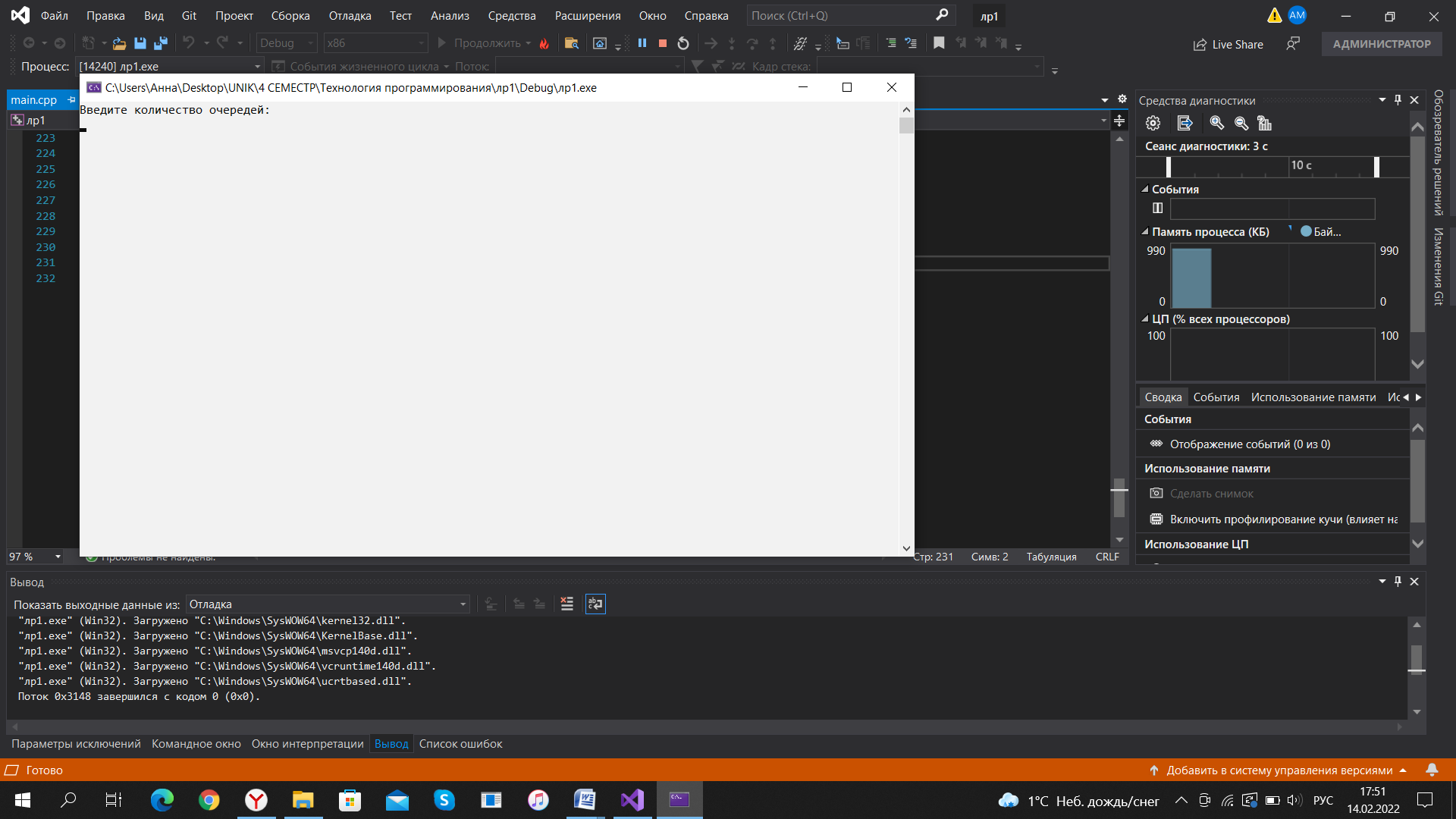


Рисунок 1 – Запуск программы

Далее предлагает выбрать модификатор доступа дочернего класса(Рис.2). Например, выберем публичный.

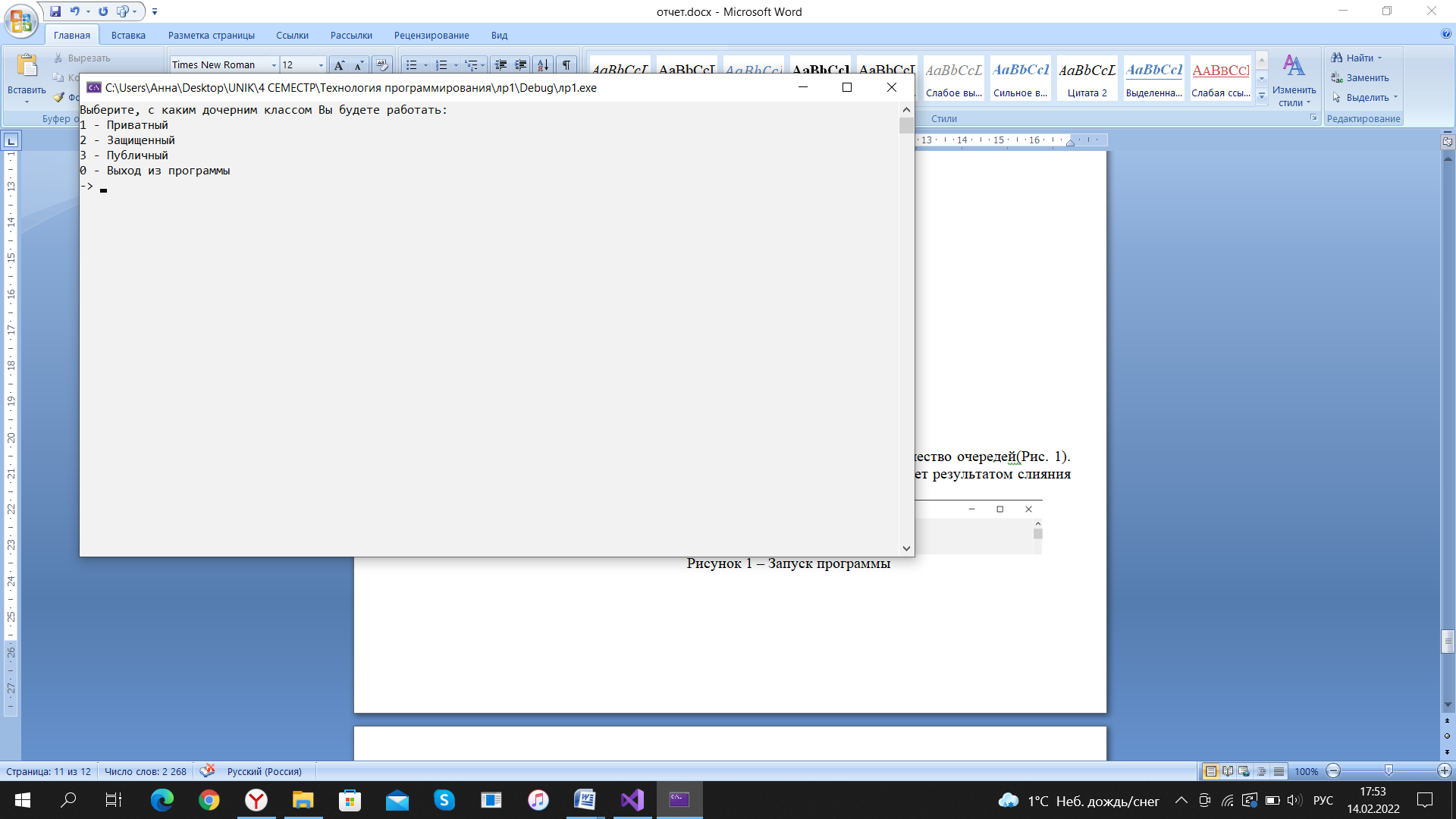


Рисунок 2 – Выбор модификатора доступа дочернего класса

После выбора перед нами появляется меню(Рис. 3). Для начала добавим элементы в очередь. Пусть будет последовательность элементов 1 2 3. И выведем ее на экран с помощью пункта №3 меню (Рис. 4).

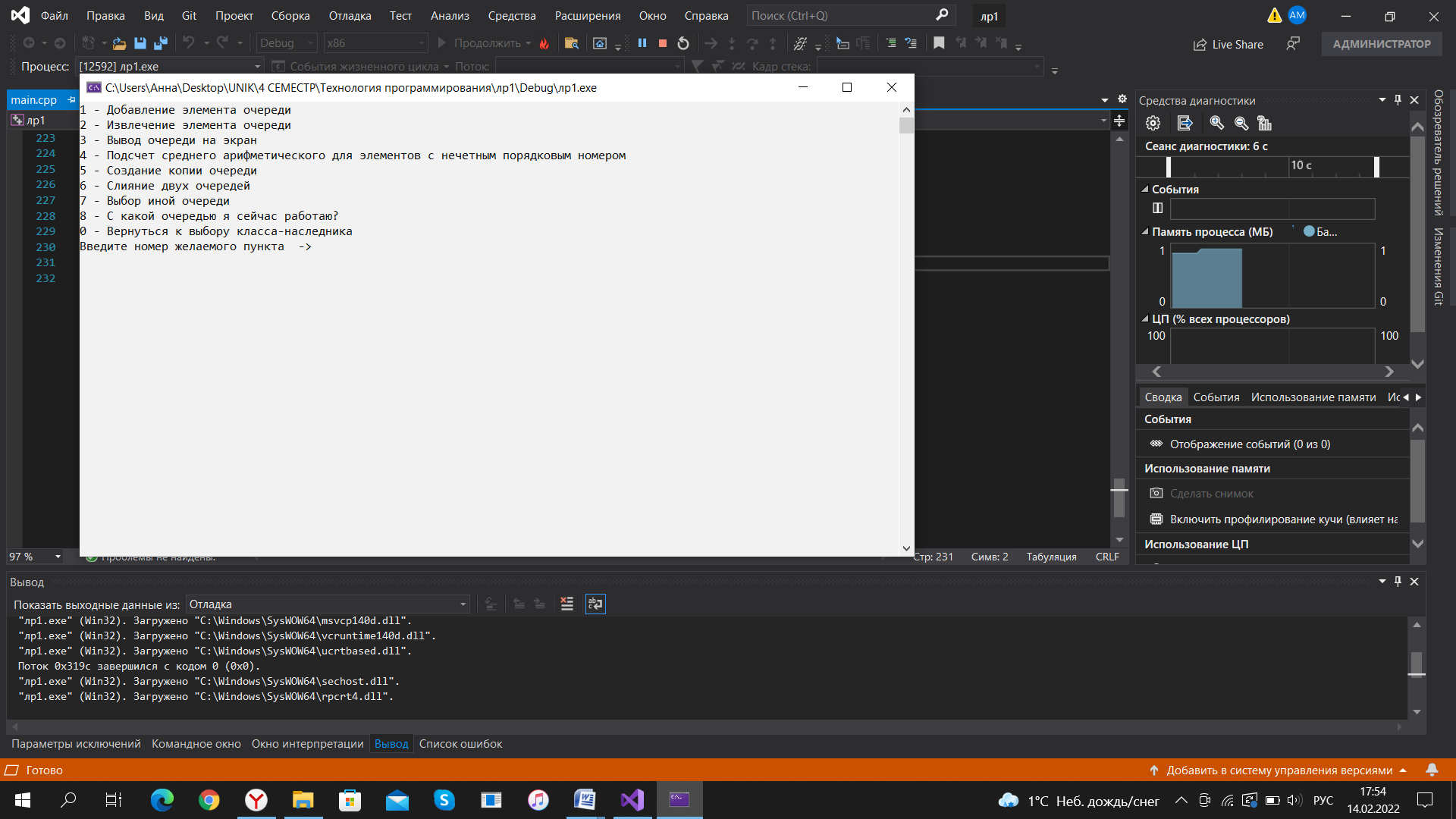
****

Рисунок 3 – Меню

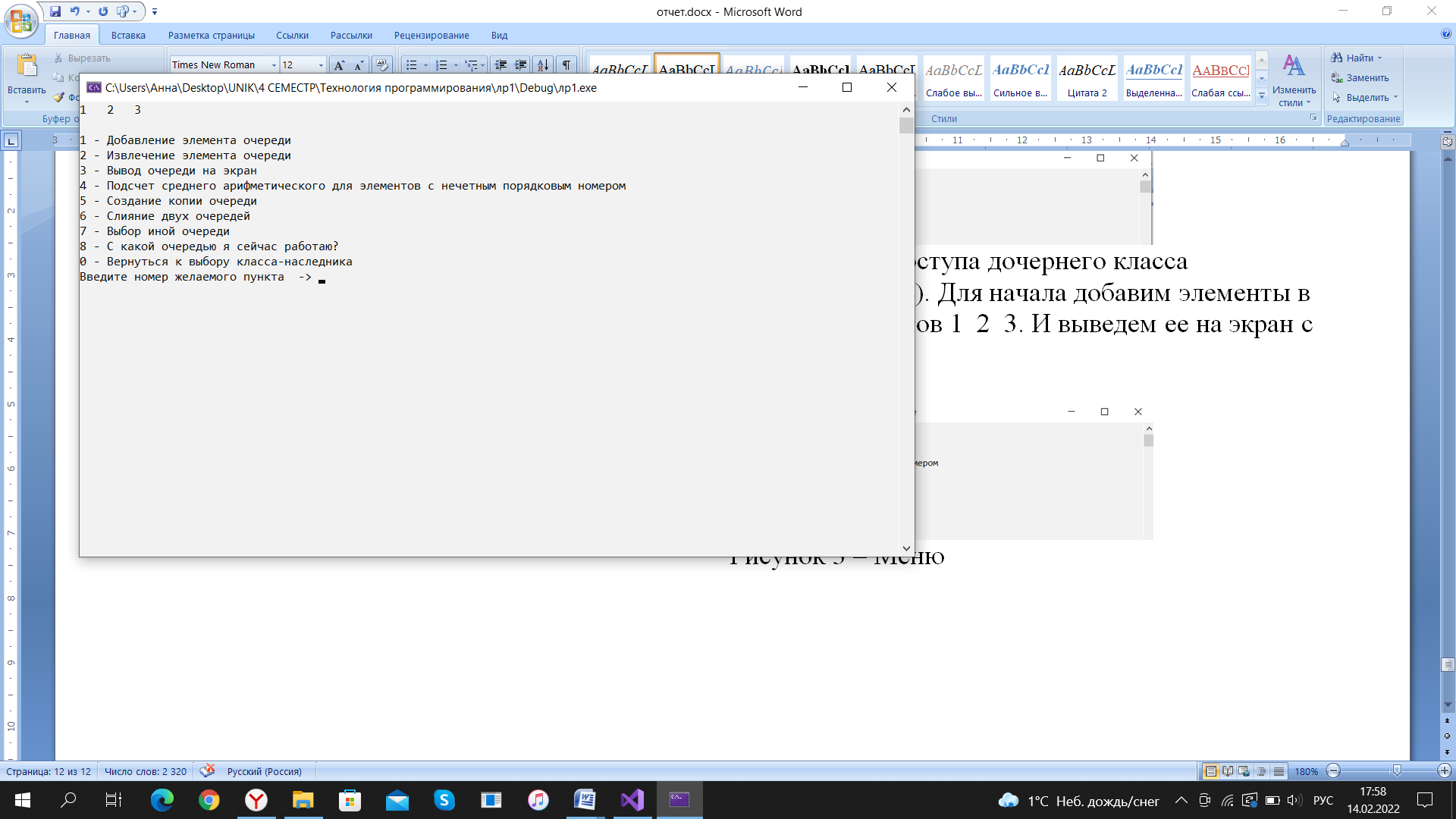


Рисунок 4 – Выполнение пунктов 1 и 3

Также программа позволяет удалить элемент (Рис. 5).

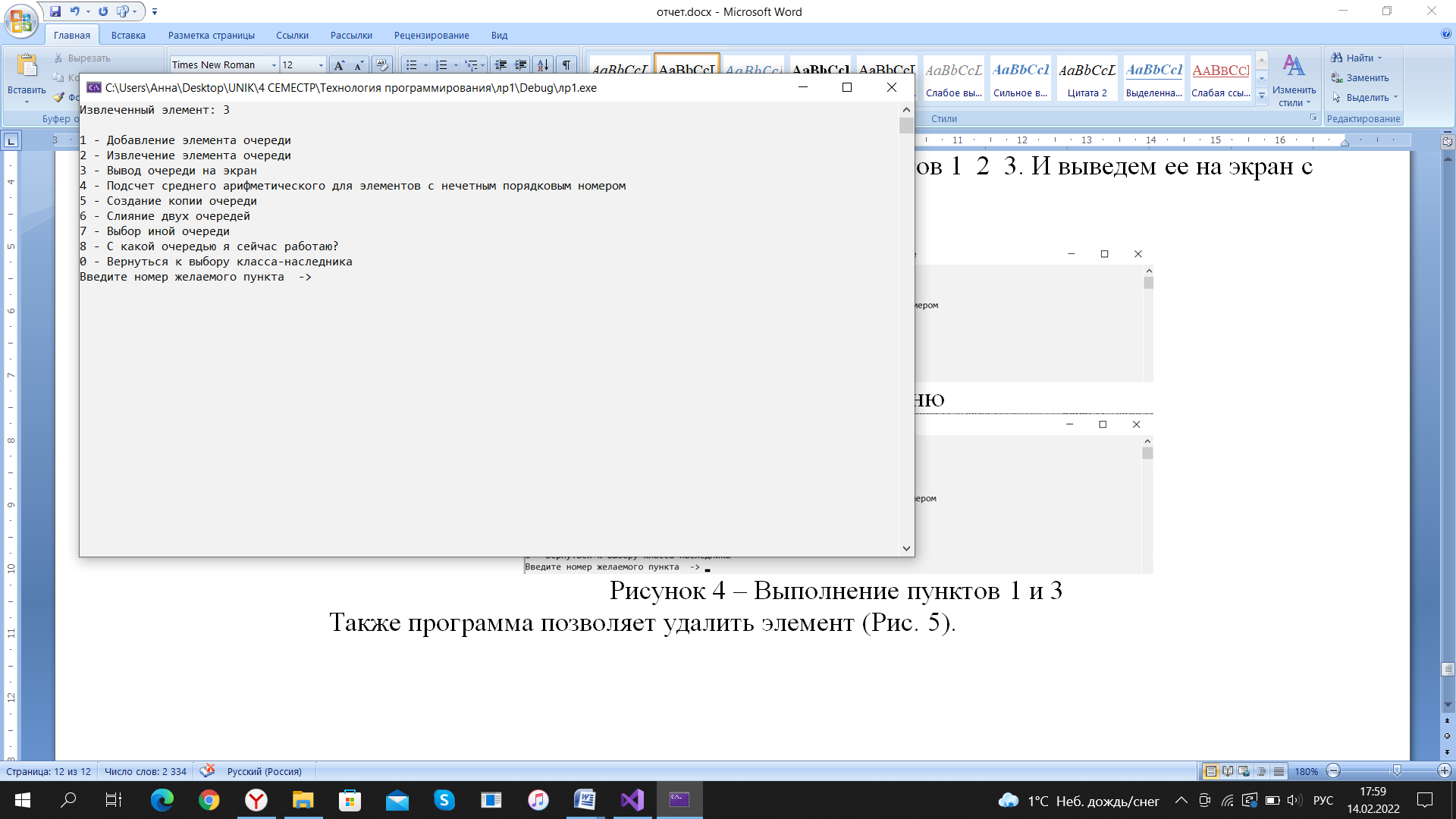


Рисунок 5 – Извлечение элемента

Теперь скопируем очередь.

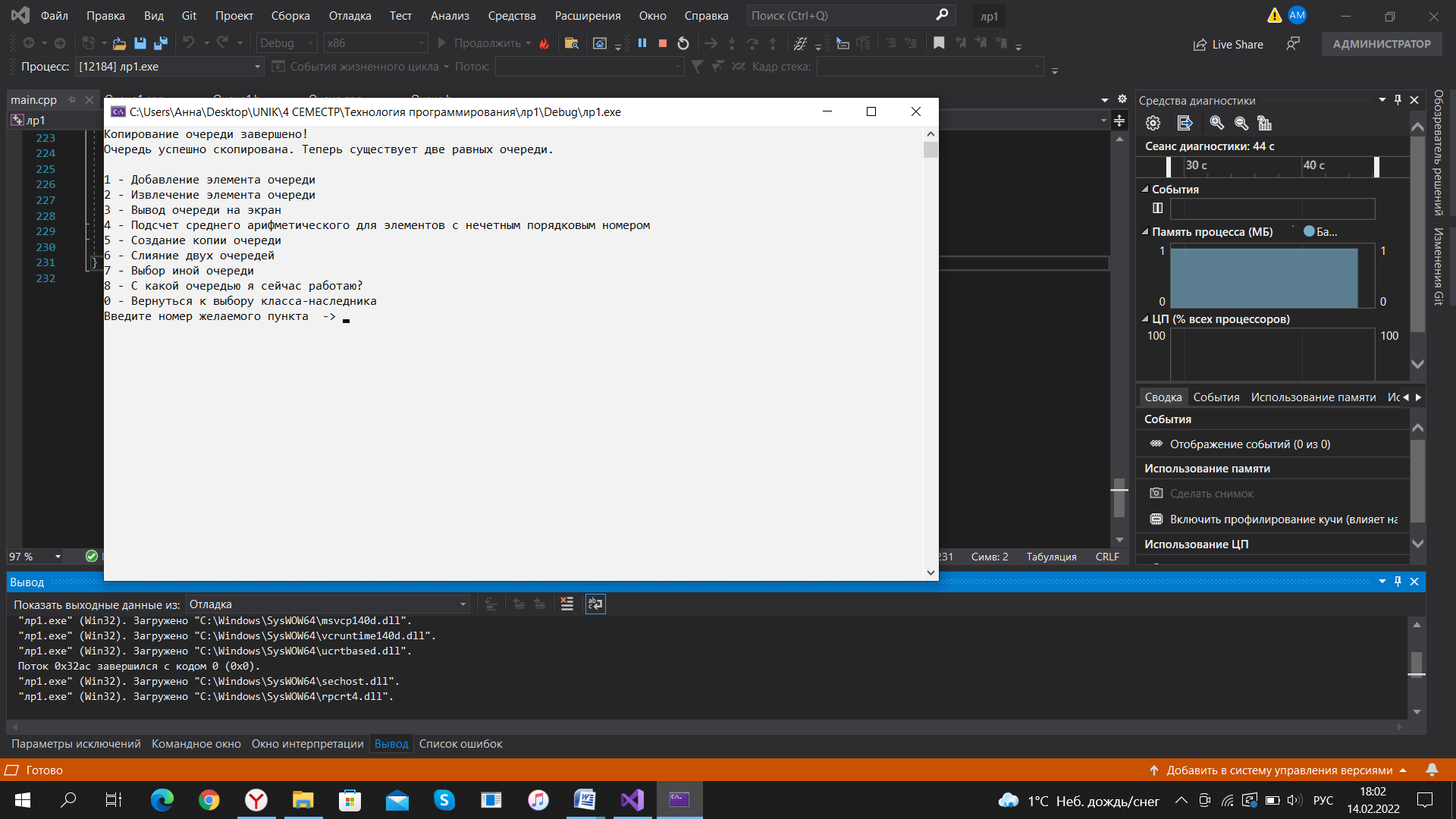


Рисунок 6 – Копирование очереди

Теперь у нас получилось две одинаковых очереди. Выполним их слияние. Получим третью очередь.

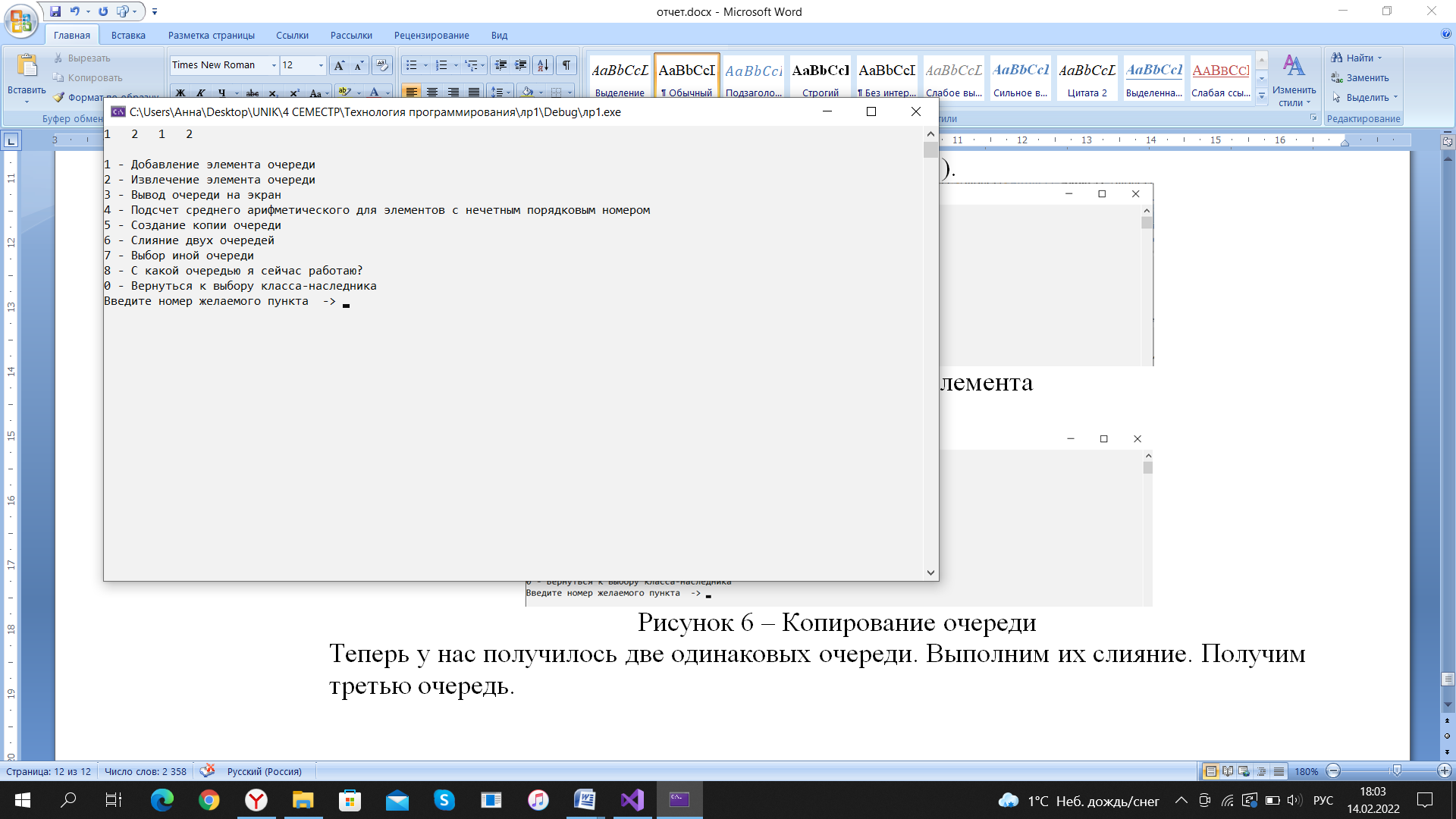


Рисунок 7 – Результат слияния двух очередей

Мы получили очередь из четырех элементов, посчитаем среднее арифметическое элементов, находящихся на нечетных позициях:

На нечетных позициях находятся элементы 2 и 2

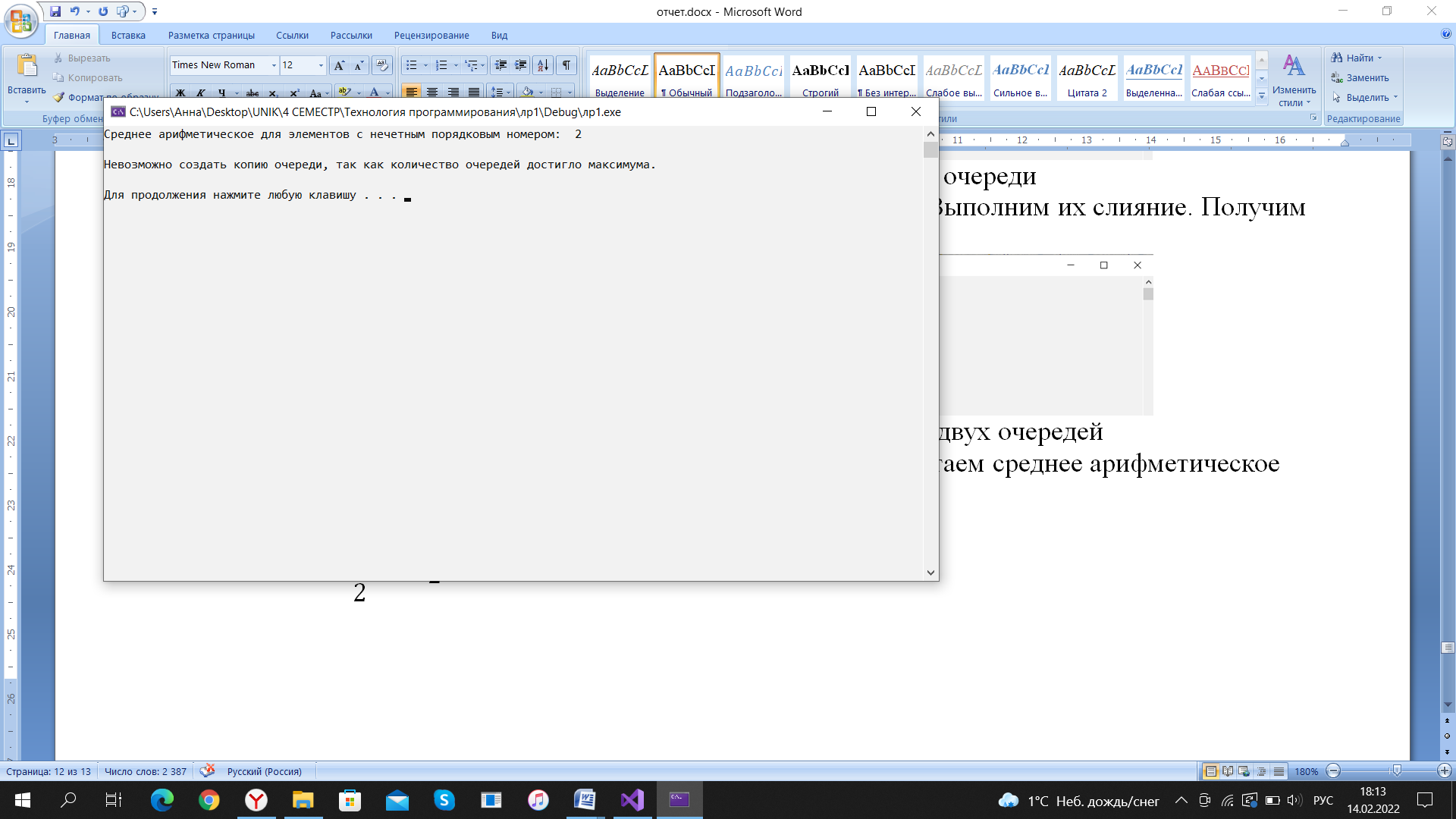


Рисунок 8 – Выполнение задания по варианту

Как мы видим, значения вычисленные программой и мною совпадают. Программа работает верно

1. **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.

Программа функционирует правильно, что подтверждается тестовыми примерами и полученными верными ответами, а также снимками экрана консоли. Можно сделать вывод о том, что лабораторная работа выполнена верно.