Пермский национальный исследовательский политехнический университет

**Лабораторная работа №9**

**Дисциплина: Основы алгоритмизации и**

**программирования**

**Вариант № 9**

**"Обработка исключительных ситуаций"**

Выполнил: Зайченко Никита Геннадьевич

Проверила: Доцент кафедры ИТАС Полякова О. А.

Пермь 2022

**Содержание отчета**

1) Постановка задачи (общая и конкретного варианта).

2) Словесное описание исключительных ситуаций.

3) Определение класса error или иерархии пользовательских классов для определения исключительных ситуаций (если есть необходимость).

4) Описание класса-контейнера.

5) Определение компонентных функций для класса-конетейнера.

6) Функция main().

7) Объяснение результатов работы программы.

8) Ответы на контрольные вопросы.

**Постановка задачи (общая и конкретного варианта)**

1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.

2) Разработка программы, обрабатывающей исключительные ситуации.

3) Класс- контейнер МНОЖЕСТВО с элементами типа int.

Реализовать операции:

[] – доступа по индексу;

!= - проверка на неравенство;

< число – принадлежность числа множеству;

+ n – переход вправо к элементу с номером n .

(1 и 3 реализация)

**Словесное описание исключительных ситуаций.**

В 1 варианте реализации обработаны исключения:

1) Значение индекса больше размера множества.

2) Некорректные параметры (ровны nullptr), что может означать, что объект не был инициализирован и не содержит корректных данных для выполнения операции.

3) Если значение ровно 0.

В 3 варианте реализации обработаны исключения:

1) Ошибка в размере Множества.

2) Пустое множество.

3) Индекс меньше нуля.

4) Индекс больше текущего размера вектора.

**Определение класса error или иерархии пользовательских классов для определения исключительных ситуаций (если есть необходимость).**

class Error//базовый класс

{

public:

virtual void what() {};

};

class IndexError :public Error //ошибка в индексе множества

{

protected:

string msg;

public:

IndexError() { msg = "Index Error\n"; }

virtual void what() { cout << msg; }

};

class SizeError :public Error //ошибка в размере Множества

{

protected:

string msg;

public:

SizeError() { msg = "Size error\n"; }

virtual void what() { cout << msg; }

};

class EmptySizeError :public SizeError //пустое множество

{

protected:

string msg\_;

public:

EmptySizeError() { SizeError(); msg\_ = "Set is empty\n"; }

virtual void what() { cout << msg << msg\_; }

};

class IndexError1 :public IndexError //индекс меньше нуля

{

protected:

string msg\_;

public:

IndexError1() { IndexError(); msg\_ = "index < 0\n"; }

virtual void what() { cout << msg << msg\_; }

};

class IndexError2 :public IndexError //индекс больше текущего размера вектора

{

protected:

string msg\_;

public:

IndexError2() { IndexError(); msg\_ = "index > size\n"; }

virtual void what() { cout << msg << msg\_; }

};

****

**Описание класса-контейнера.**

class Iterator

{

friend class Set;//дружественный класс

public:

Iterator() { elem = 0; }//конструктор без параметров

Iterator(const Iterator& it) { elem = it.elem; }//конструктор копирования

int& operator \*() const { return\*elem; } //перегруженная операция разыменования

private:

int\* elem; //указатель на элемент типа int

};

class Set

{

public:

Set(int s, int k = 0); //конструктор с параметрами: выделяет память под s элементов и заполняет их значением k

Set(const Set& a); //конструктор с параметрами

~Set(); //деструктор

Set& operator=(const Set& a); //оператор присваивания

int& operator[](int index); //операция доступа по индексу

int operator()(); //операция, возвращающая длину вмножества

bool operator<(int value) const; //проверка принадлежности числа множеству

bool operator!=(const Set&); // проверка на неравенство

int operator+(int n) const; //переход вправо к элементу с номером n

friend ostream& operator<<(ostream& out, const Set& a); //перегруженные операции вывода

friend istream& operator>>(istream& in, Set& a); //перегруженные операции ввода

Iterator first() { return beg; } //возвращает указатель на первый элемент

Iterator last() { return end; } //возвращает указатель на элемент следующий за последним

private:

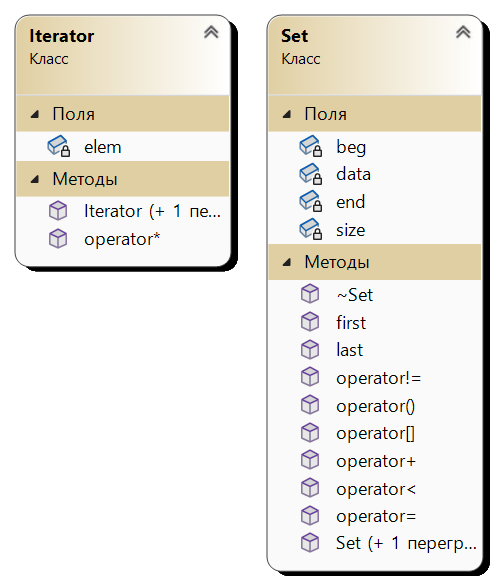
int size; //размер множества

int\* data; //укзатель на динамический массив значений вектора

Iterator beg; //указатель на первый элемент множества

Iterator end; //указатель на элемент следующий за последним

};



**Определение компонентных функций для класса-конетейнера.**

Set::Set(int s, int k)

{

size = s;

data = new int[size];

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

data[i] = k;

beg.elem = &data[0];

end.elem = &data[size];

}

//конструктор копирования

Set::Set(const Set& a)

{

size = a.size;

data = new int[size];

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

data[i] = a.data[i];

beg = a.beg;

end = a.end;

}

//деструктор

Set::~Set() { delete[]data; data = 0; }

//операция присваивания

Set& Set::operator=(const Set& a)

{

if (this == &a)return \*this;

size = a.size;

if (data != 0) delete[]data;

data = new int[size];

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

data[i] = a.data[i];

return \*this;

beg = a.beg;

end = a.end;

return \*this;

}

//операция доступа по индексу

int& Set::operator[](int index)

{

if (index > size) throw 1;

else return data[index];

}

//операция для получения длины вектора

int Set::operator ()() { return size; }

//проверка принадлежности числа множеству

bool Set::operator<(int value) const

{

for (int i = 0; i < size; ++i) {

if (data[i] == NULL) { throw 3; }

else

{

if (data[i] >= value)

{

return data[i] == value;

}

}

}

return false;

}

// проверка на неравенство

bool Set::operator!=(const Set& a) {

if (!this || !&a) {

throw 2;

}

return data != a.data;

}

int Set:: operator+(int n) const

{

if (n > size) throw 1;

else return data[n];

} //переход вправо к элементу с номером n

//операции для ввода-выода

ostream& operator<<(ostream& out, const Set& a)

{

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

out << a.data[i] << " ";

return out;

}

istream& operator>>(istream& in, Set& a)

{

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

in >> a.data[i];

return in;

}

**Функция main().**

**1 вариант реализации:**

int main()

{

setlocale(LC\_ALL, "Russian");

int n, k, l, j, s;

//контролируемый блок

try

{

Set a(5);

cout << "Множнство a: " << endl;

cout << a << endl;

cin >> a;

cout << "Введите элемент котрый хотите заменить: " << endl;

cin >> n;

cout << "Введите новое значение: " << endl;

cin >> k;

a[n - 1] = k;

cout << "Множнство a: " << endl;

cout << a << endl;

cout << endl << "Проверяем равенство первого и второго элемента множества a: " << endl;

cout << "Введите номера элемнтов для сравнения: " << endl;

cin >> l; cin >> j;

cout << endl << a[l-1] << " != " << a[j-1] << endl;

if (a[l-1] != a[j-1]) { cout << "Элементы не равны" << endl; }

else { cout << "Элементы равны" << endl; }

cout << endl << "Введите элимент для проверки принодлежности множеству: ";

cin >> s;

if (a < s) { cout << "Элементы принадлежит множеству a" << endl; }

else { cout << "Элементы не принадлежит множеству a" << endl; }

cout << "Переход вправо к элементу с номером n, введите n: " << endl;

cin >> j; j = j - 1;

cout << "Элемент с номером n: " << a+j;

}

//обработчик исключения

catch (int)

{

cout << "ERROR!!!" << endl; //сообщение об ошибке

}

return 0;

}

**3 вариант реализации:**

int main()

{

setlocale(LC\_ALL, "Russian");

int n, k, l, j, s;

//контролируемый блок

try

{

Set a(5);

cout << "Множнство a: " << endl;

cout << a << endl;

cin >> a;

cout << "Введите элемент котрый хотите заменить: " << endl;

cin >> n;

cout << "Введите новое значение: " << endl;

cin >> k;

a[n - 1] = k;

cout << "Множнство a: " << endl;

cout << a << endl;

cout << endl << "Проверяем равенство первого и второго элемента множества a: " << endl;

cout << "Введите номера элемнтов для сравнения: " << endl;

cin >> l; cin >> j;

cout << endl << a[l - 1] << " != " << a[j - 1] << endl;

if (a[l - 1] != a[j - 1]) { cout << "Элементы не равны" << endl; }

else { cout << "Элементы равны" << endl; }

cout << endl << "Введите элимент для проверки принодлежности множеству: ";

cin >> s;

if (a < s) { cout << "Элементы принадлежит множеству a" << endl; }

else { cout << "Элементы не принадлежит множеству a" << endl; }

cout << "Переход вправо к элементу с номером n, введите n: " << endl;

cin >> j; j = j - 1;

cout << "Элемент с номером n: " << a + j;

}

//обработчик исключения

catch (Error& e)

{

e.what();

}

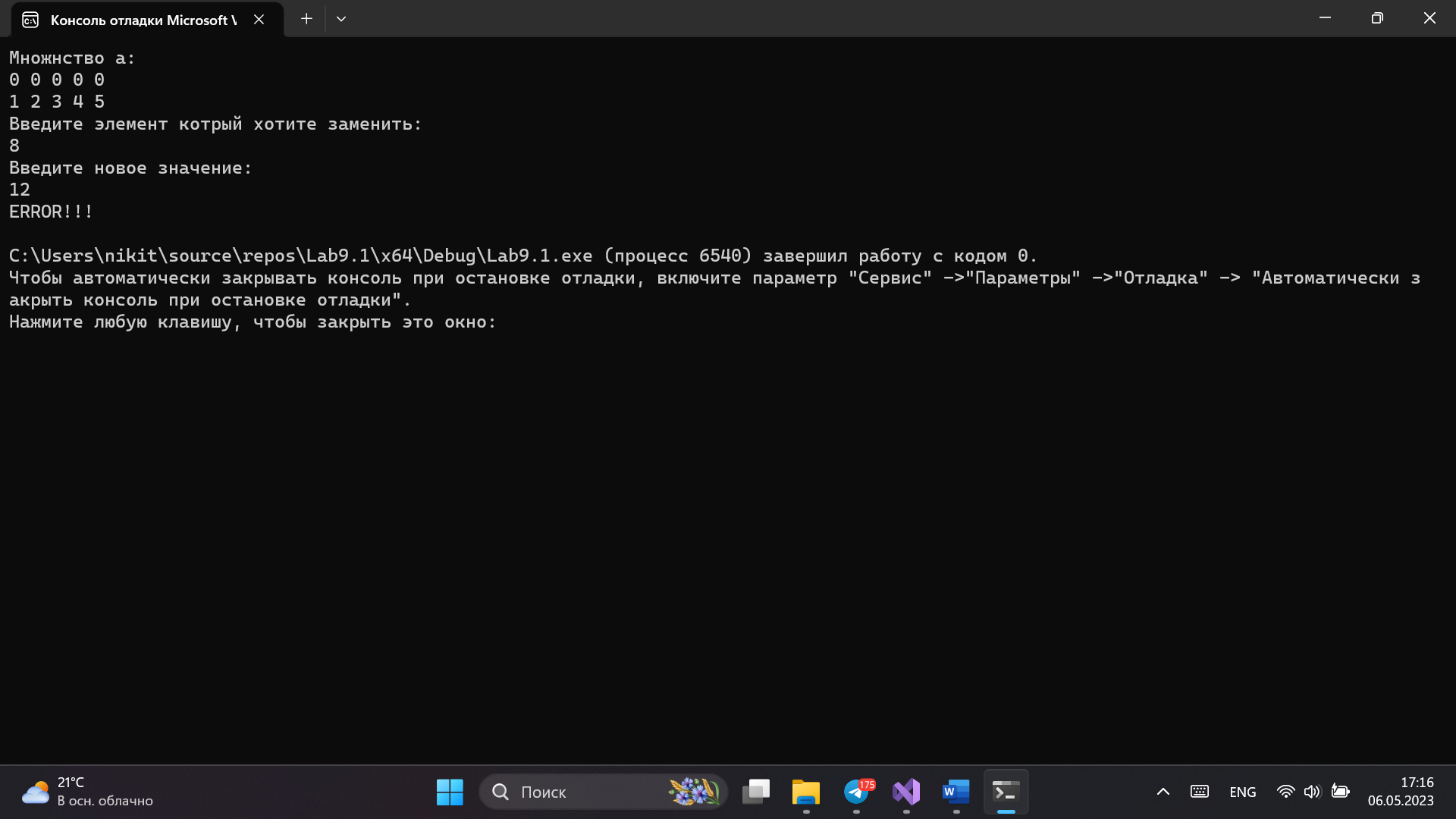
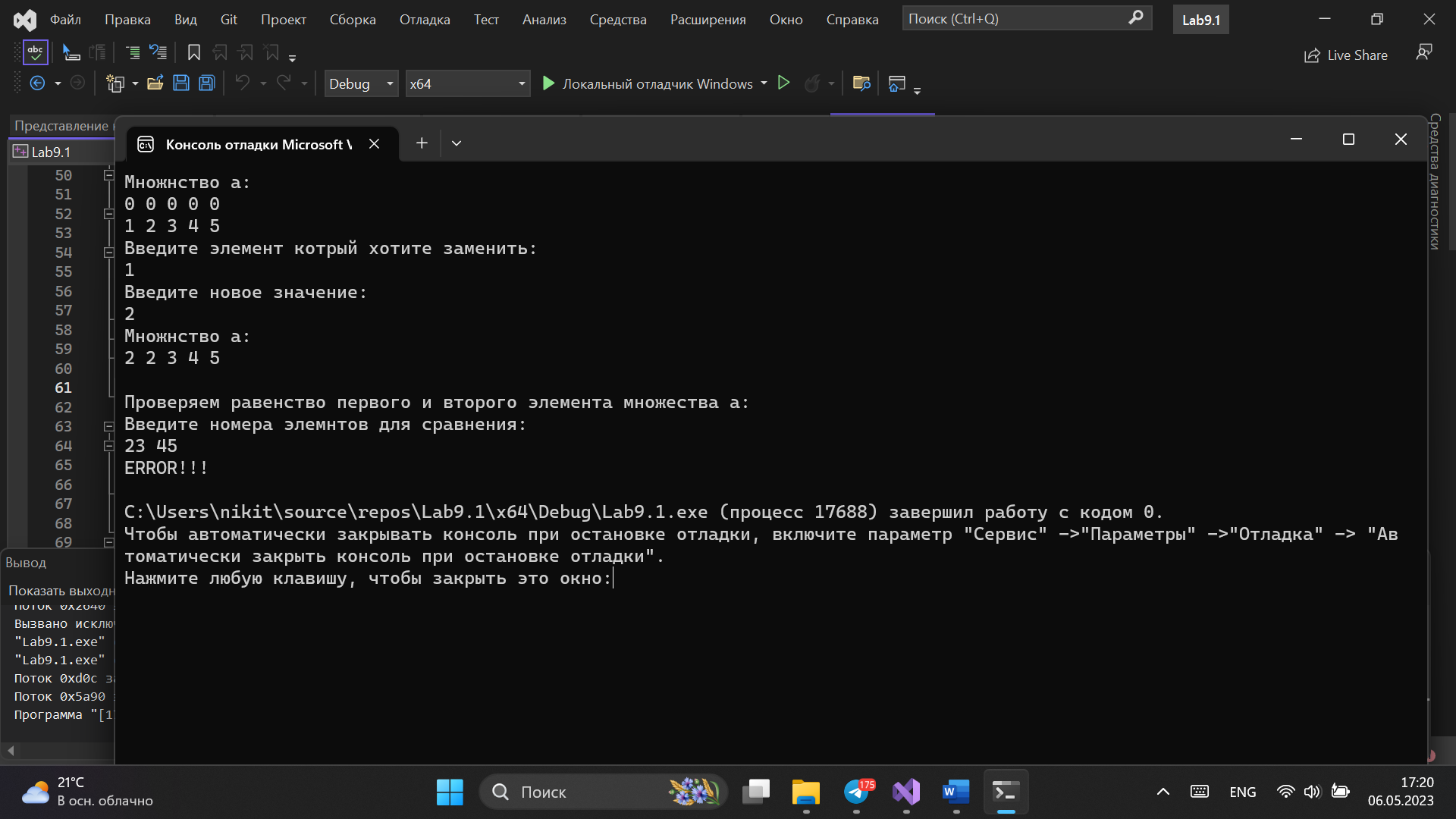
return 0;

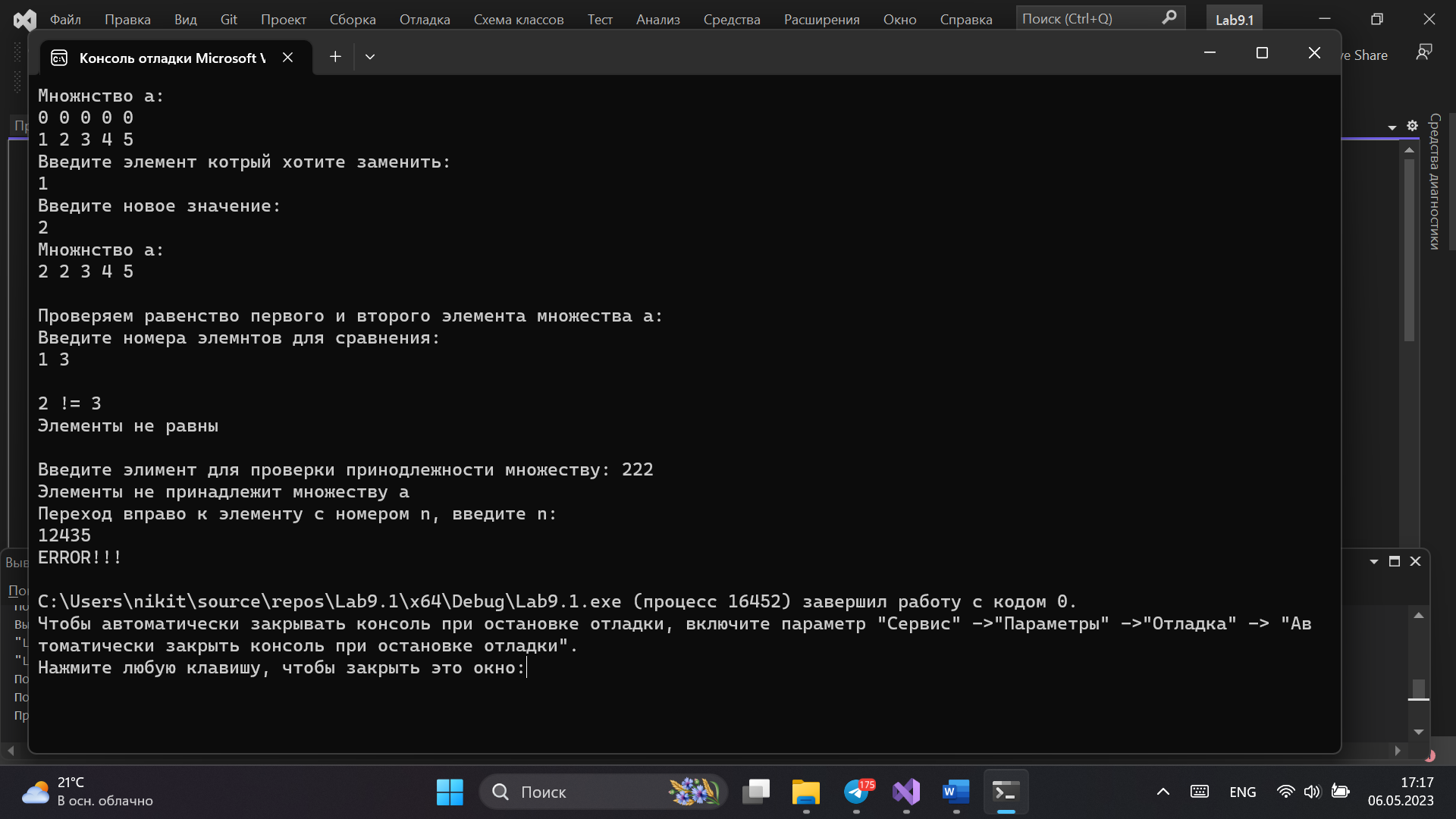
}

**Объяснение результатов работы программы.**

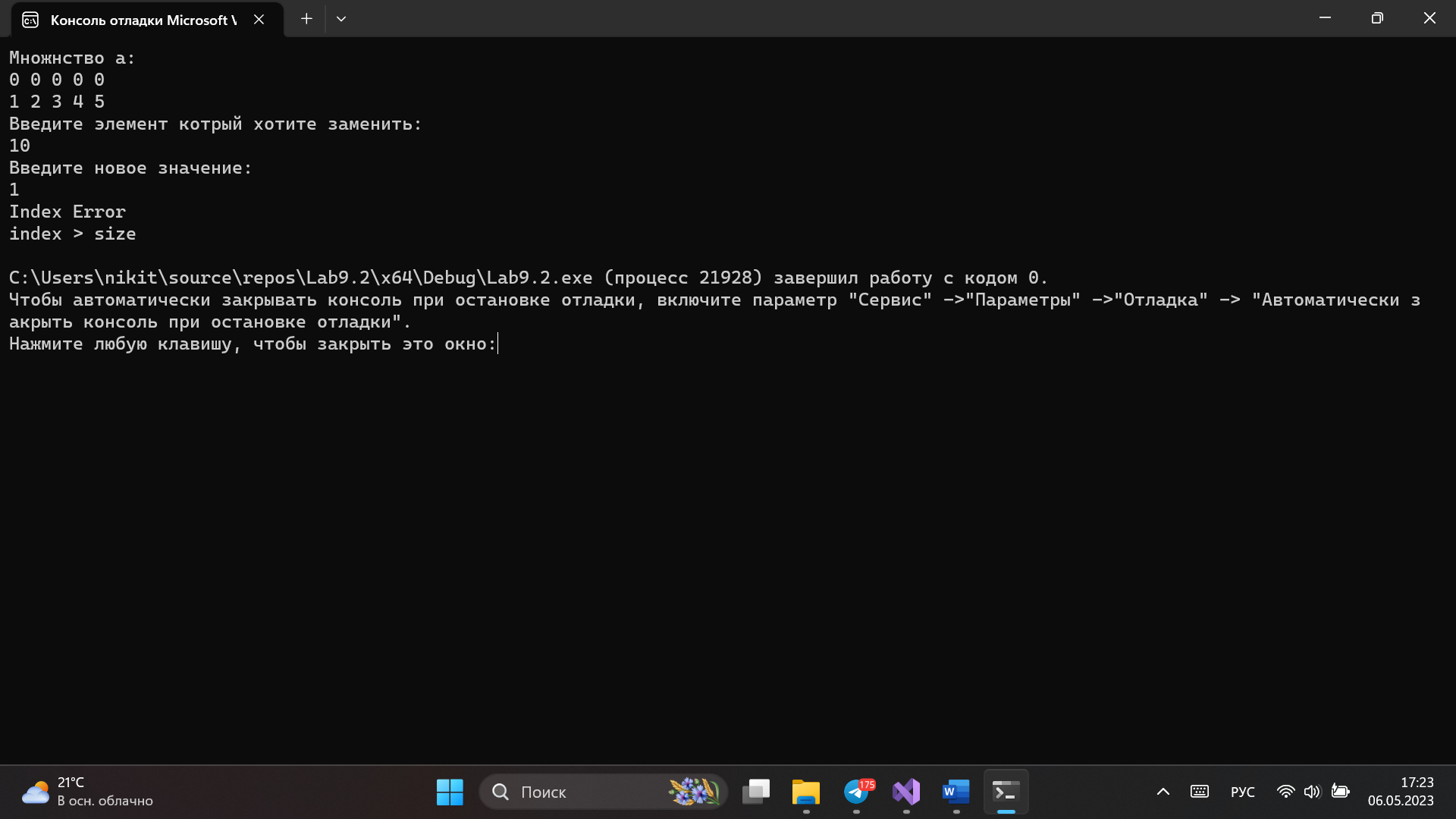
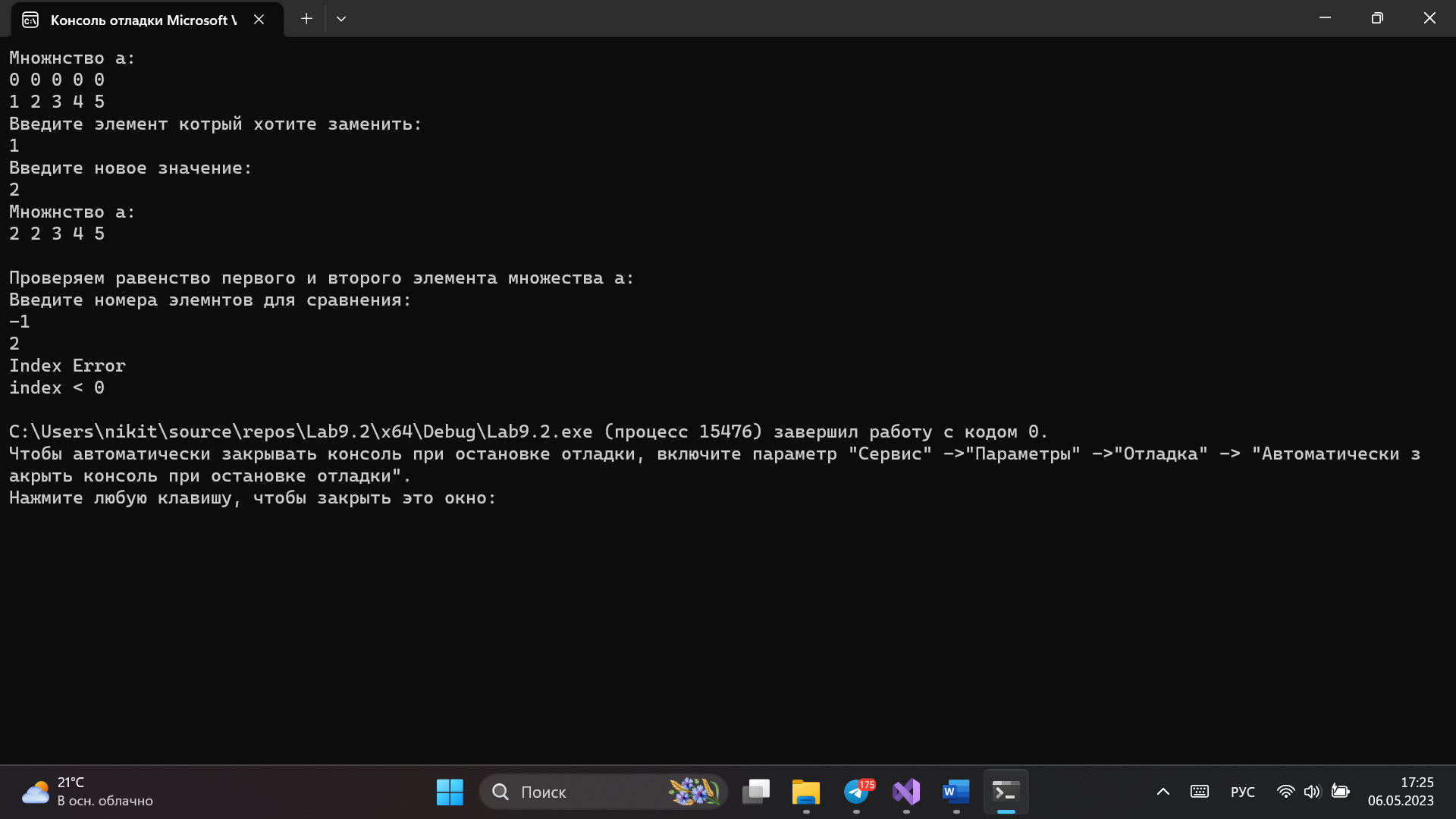
Данная программа демонстрирует, обработку исключительных ситуаций 2 вариантами реализации, 1 вариант информация об исключительных ситуациях передается с помощью стандартного типа данных, во 2 варианте реализации, информация об исключительных ситуациях передается с помощью иерархии пользовательских классов.Создается базовый класс Error и наследуемые классы с обработкой исключений.

1 вариант реализации:



2 вариант реализации:

**Ответы на контрольные вопросы**

**1. Что представляет собой исключение в С++?**

**2. На какие части исключения позволяют разделить вычислительный процесс? Достоинства такого подхода?**

**3. Какой оператор используется для генерации исключительной ситуации?**

**4. Что представляет собой контролируемый блок? Для чего он нужен?**

**5. Что представляет собой секция-ловушка? Для чего она нужна?**

**6. Какие формы может иметь спецификация исключения в секции ловушке? В каких**

**ситуациях используются эти формы?**

**7. Какой стандартный класс можно использовать для создания собственной иерархии**

**исключений?**

**8. Каким образом можно создать собственную иерархию исключений?**

**9. Если спецификация исключений имеет вид: void f1()throw(int,double); то какие**

**исключения может прождать функция f1()?**

**10. Если спецификация исключений имеет вид: void f1()throw(); то какие исключения**

**может прождать функция f1()?**

**11. В какой части программы может генерироваться исключение?**

**12. Написать функцию, которая вычисляет площадь треугольника по трем сторонам (формула Герона).**

**Функцию реализовать в 4 вариантах:**

**a) Без спецификации исключений;**

**b) Со спецификацией throw();**

**c) С конкретной спецификацией с подходящим стандартным исключением;**

**d) Спецификация с собственным реализованным исключением.**

1. Исключение в С++ — это механизм, который позволяет обрабатывать ошибки и нестандартные ситуации в программе. Он позволяет прерывать выполнение программы в каком-либо месте и передавать управление в блок, который будет обрабатывать исключительную ситуацию.

2. Исключения в С++ можно разделить на три части: генерация исключения, передача исключения и обработка исключения. Правильное разделение на эти части позволяет более эффективно и надежно обрабатывать ошибки в программе.

3. Для генерации исключительной ситуации используется оператор throw.

4. Контролируемый блок — это участок кода, в котором может возникнуть исключительная ситуация. Для контролируемого блока определяется спецификация исключений, которые могут возникнуть в данном блоке.

5. Секция-ловушка — это блок кода, который обрабатывает исключение, которое было брошено где-то в предшествующих блоках. Секция-ловушка определяет, какие исключения будут обрабатываться и как этот блок кода будет обрабатывать исключение.

6. Спецификация исключений в секции-ловушке может иметь три формы:

a) catch - обрабатывает любое исключение, без указания типа.

b) catch (тип) - обрабатывает исключения только указанного типа.

c) catch (...) - обрабатывает любое исключение, но не служит для передачи информации об исключении.

7. Для создания собственной иерархии исключений можно использовать стандартный класс exception.

8. Для создания собственной иерархии исключений нужно создать класс исключения, который наследуется от класса exception, и определить необходимые методы.

9. Спецификация void f1() throw(int,double) говорит о том, что функция f1() может бросить исключения типа int или double.

10. Спецификация void f1() throw() говорит о том, что функция f1() не может бросить никаких исключений.

11. Исключение может возникнуть в любой части программы, где встретится оператор throw или при выполнении операций, которые могут привести к ошибке выполнения.

12. Функция для вычисления площади треугольника по трем сторонам:

**a) Без спецификации исключений:**

double triangleArea(double a, double b, double c) {

if (a + b <= c || b + c <= a || a + c <= b) {

throw "Invalid triangle sides";}

double p = (a + b + c) / 2;

double s = sqrt(p \* (p - a) \* (p - b) \* (p - c));

return s;}

**b) Cо спецификацией throw():**

double triangleArea(double a, double b, double c) throw() {

if (a + b <= c || b + c <= a || a + c <= b) {

throw "Invalid triangle sides";}

double p = (a + b + c) / 2;

double s = sqrt(p \* (p - a) \* (p - b) \* (p - c));

return s;}

**c) С конкретной спецификацией и использованием стандартного исключения:**

double triangleArea(double a, double b, double c) throw(invalid\_argument) {

if (a + b <= c || b + c <= a || a + c <= b) {

throw invalid\_argument("Invalid triangle sides");}

double p = (a + b + c) / 2;

double s = sqrt(p \* (p - a) \* (p - b) \* (p - c));

return s;}

**d) Со спецификацией исключения с использованием собственного исключения:**

class InvalidTriangleSidesException : public runtime\_error {

public:

InvalidTriangleSidesException() : runtime\_error("Invalid triangle sides"){}};

double triangleArea(double a, double b, double c) throw(InvalidTriangleSidesException) {

if (a + b <= c || b + c <= a || a + c <= b) {

throw InvalidTriangleSidesException();}

double p = (a + b + c) / 2;

double s = sqrt(p \* (p - a) \* (p - b) \* (p - c));

return s;}