***Лабораторная работа 5.***

***Программная реализация отношений простого наследования***

***Задача:***

1. Создайте производный класс для АТД, реализованного по заданию лабораторной работы 3, используя одиночное наследование.

2. Проверьте работоспособность АТД и производного класса на тестовом наборе данных.

***Решение:***

#include <iostream>

using namespace std;

class Fireplaces {

private:

int quantity;

int view;

char\* name;

public:

Fireplaces(){}

Fireplaces(int quantity, int view) : quantity(quantity), view(view) {}

void input(int qua, int vie, char\* nam) {

quantity = qua;

view = vie;

name = new char[strlen(nam) + 1];

strcpy(name, nam);

}

void print() {

cout << "view = " << view << endl << "quantity = " << quantity << endl;

}

void operator = (const Fireplaces& f) {

name = new char[strlen(f.name) + 1];

strcpy(name, name);

}

void operator + (const Fireplaces& f) {

quantity = quantity + f.quantity;

}

void operator \* (const Fireplaces& f) {

quantity = f.quantity \* quantity;

}

void operator - (const Fireplaces& f) {

quantity = f.quantity - quantity;

}

};

class DuHast:public Fireplaces {

private:

double dudu;

float agaga;

public:

DuHast(int quantity, int view, double dudu, float agaga) : Fireplaces(quantity, view), agaga(agaga), dudu(dudu) {}

void print2() {

cout << "agaga = " << agaga << endl << "dudu = " << dudu << endl;

}

};

int main()

{

setlocale(LC\_ALL, "russian");

Fireplaces fire;

fire.input(10101010, 2020202, (char\*)"printer");

DuHast duhast(1000, 757000, 8756565.856575, 100.09);

DuHast duhast2(4500, 677000, 1116565.111575, 111.09);

cout << "DUHAST: \_\_\_\_\_\_\_\_\_\_" << endl;

duhast.print();

cout << endl << "DUHAST2: \_\_\_\_\_\_\_\_\_\_" << endl;

duhast.print2();

duhast + duhast2;

cout << endl << "DUHAST: \_\_\_\_\_\_\_\_\_\_ \\после сложения" << endl;

duhast.print();

cout << endl << "FIRE: \_\_\_\_\_\_\_\_\_\_" << endl;

fire.print();

fire + duhast;

cout << endl << "FIRE: \_\_\_\_\_\_\_\_\_\_\\после сложения" << endl;

fire.print();

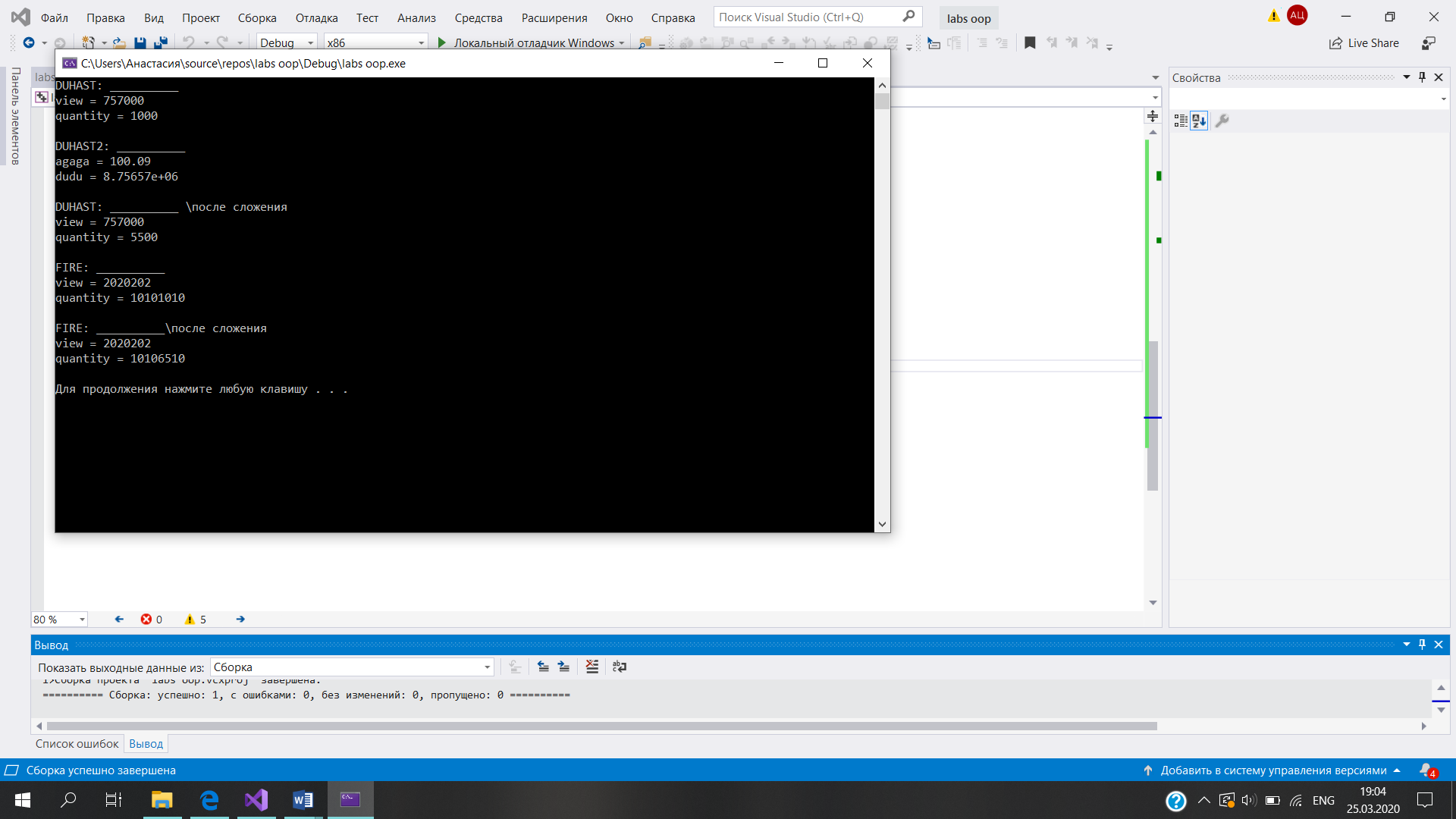
cout << endl;

system("pause");

return 0;

}

***Тестирование:***



***Контрольные вопросы:***

***1. Дайте определение наследования:***

Наследование (inheritance) – это механизм получения нового класса на основе существующего класса.

***2. Какие модификаторы прав доступа к производному классу вы знаете и в чем их особенности:***

Модификаторы прав доступа к членам класса (public, protected и private) могут применяться в объявлении класса в любом порядке и сколько угодно раз. Открытый член доступен во всей области видимости, где виден класс. Закрытый член доступен другим функциям-членам своего класса. Защищенный член доступен не только другим функциям-членам своего класса, но и функциям-членам класса, унаследованного непосредственно от данного класса. Если все члены открыты, то ключевое слово class можно заменить ключевым словом struct.

***3. Как выполняется конструктор при наследовании:***

При наследовании и инициализации членов класса конструкторы выполняются в следующем порядке:

1. Базовые классы инициализируются в порядке объявления.

2. Члены инициализируются в порядке объявления.

***4. Дайте определение чисто виртуальной функции:***

Чисто виртуальная функция – это функция, в объявлении которой присутствует как ключевое слово virtual, так и инициализатор 0. Такая функция переопределяется в классах наследниках, а базовый класс превращает в абстрактный, т.е. содержащий минимально-необходимый функционал класс. Пример: ***virtual void print() = 0;***

***5. Какие операторы используются для разыменования указателя на член класса?***

Для разыменования указателя на член класса используются два оператора: .\* и →\*.

***Лабораторная работа 6.***

***Программная реализация множественного наследования***

***Задание:***

1. Используя предыдущую программу, создайте новый производный класс с применением множественного наследования.

2. Проверьте работоспособность АТД и производных классов на тестовом наборе данных.

***Решение:***

#include <iostream>

using namespace std;

class Fireplaces {

private:

int quantity;

int view;

char\* name;

public:

Fireplaces(){}

Fireplaces(int quantity, int view) : quantity(quantity), view(view) {}

void input(int qua, int vie, char\* nam) {

quantity = qua;

view = vie;

name = new char[strlen(nam) + 1];

strcpy(name, nam);

}

void print() {

cout << "view = " << view << endl << "quantity = " << quantity << endl;

}

void operator = (const Fireplaces& f) {

name = new char[strlen(f.name) + 1];

strcpy(name, name);

}

void operator + (const Fireplaces& f) {

quantity = quantity + f.quantity;

}

void operator \* (const Fireplaces& f) {

quantity = f.quantity \* quantity;

}

void operator - (const Fireplaces& f) {

quantity = f.quantity - quantity;

}

};

class DuHast {

public:

double dudu;

float agaga;

DuHast(double dudu, float agaga) : agaga(agaga), dudu(dudu) {}

void print2() {

cout << "agaga = " << agaga << endl << "dudu = " << dudu << endl;

}

void operator + (const DuHast& dh) {

dudu += dh.dudu;

}

};

class Koleso : public Fireplaces, public DuHast {

private:

int camera;

float mamont;

public:

Koleso(int camera, float mamont, int quantity, int view, double dudu, float agaga) : Fireplaces(quantity,view), DuHast(dudu,agaga), camera(camera),mamont(mamont){}

void print3() {

cout << "camera = " << camera << endl << "mamont = " << mamont << endl;

}

void summa() {

mamont += dudu + agaga;

cout << "Сложение элементов dudu и agaga с элементом mamont в классе Koleso = " << mamont << endl;

}

};

int main()

{

setlocale(LC\_ALL, "russian");

Fireplaces fire;

fire.input(10101010, 2020202, (char\*)"printer");

DuHast duhast(8756565.856575, 100.09);

Koleso koleso(8980, 8767.675, 1000, 1203, 65453.78685, 10102.7464);

cout << "DUHAST: \_\_\_\_\_\_\_\_\_\_" << endl;

duhast.print2();

cout << endl << "FIRE: \_\_\_\_\_\_\_\_\_\_" << endl;

fire.print();

cout << endl << "KOLESO: \_\_\_\_\_\_\_\_\_\_//элементы класса Koleso" << endl;

koleso.print3();

cout << endl << "KOLESO: \_\_\_\_\_\_\_\_\_\_//элементы класса Fireplaces через конструктор Koleso" << endl;

koleso.print();

cout << endl << "KOLESO: \_\_\_\_\_\_\_\_\_\_//элементы класса DuHast через конструктор Koleso" << endl;

koleso.print2();

cout << endl << "Koleso: \_\_\_\_\_\_\_\_\_\_//вызов функции сумма класса Koleso" << endl;

koleso.summa();

cout << endl << "Koleso: \_\_\_\_\_\_\_\_\_\_ \\после сложения" << endl;

koleso.print3();

cout << endl << "FIRE: \_\_\_\_\_\_\_\_\_\_\\после сложения" << endl;

fire + koleso;

fire.print();

cout << endl << "Duhast: \_\_\_\_\_\_\_\_\_\_\\после сложения" << endl;

duhast + koleso;

fire.print();

cout << endl << "FIRE: \_\_\_\_\_\_\_\_\_\_\\после вычитания" << endl;

fire - koleso;

fire.print();

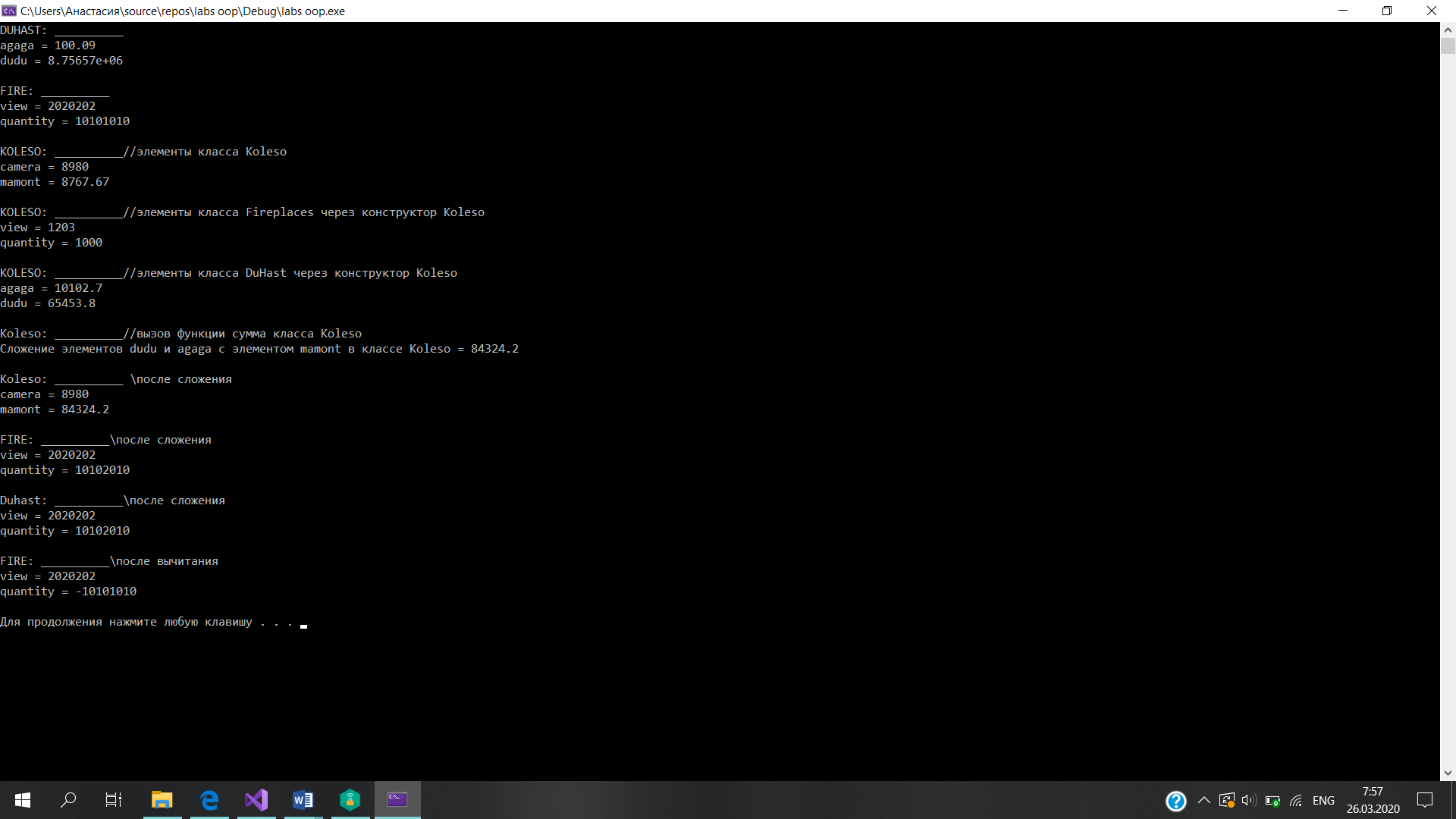
cout << endl;

system("pause");

return 0;

}

***Тестирование:***



***Контрольные вопросы:***

***1. С какой целью и в каких случаях используется множественное наследование:***

Множественное наследование (multiple inheritance) делает возможным получение производного класса от нескольких базовых классов. Множественное наследование позволяет соединить интерфейс с реализацией

***2. Опишите синтаксис заголовка производного класса при множественном наследовании:***

Синтаксис заголовка класса расширяется, чтобы можно было использовать список базовых классов с атрибутами доступа. Например:

class shape {};

class tview {};

class tshape: public shape, private tview {};

***3. Дайте определение ориентированного ациклического графа:***

Ориентированный ациклический граф (DAG) – это граф, узлы которого являются классами, а ориентированные ребра направлены от производных классов к базовым.

***4. Где инициируются виртуальные базовые классы:***

Виртуальные базовые классы инициализируются (вызывается void-конструктор) перед любыми невиртуальными базовыми классами и в том порядке, в котором они появляются в ПАГе наследования при просмотре его снизу-вверх и слева направо.

***5. Что будет, если из объявлений классов Cow и Buffalo убрать ключевое слово virtual (см. последний пример в теоретических положениях):***

Ключевое слово virtual в классе Cow и классе Buffalo предотвращает многократное копирование полей данных weight, price, color из предков класса Beefalo, а значит удаление ключевого слова virtual приведет к понижению степени повторного использования кода, что нарушает принцип объектно-ориентированного программирования и одну из важнейших функций наследования.

***Лабораторная работа 7.***

***Программирование обработки исключительных ситуаций***

***Задание:***

Используя модифицированный АТД, обработайте все возможные исключительные ситуации.

***Решение:***

#include <iostream>

using namespace std;

int k;

float j;

class Fireplaces {

private:

int quantity;

int view;

char\* name;

public:

Fireplaces(){}

Fireplaces(int quantity, int view) : quantity(quantity), view(view) {}

void input(int qua, int vie, char\* nam) {

quantity = qua;

view = vie;

name = new char[strlen(nam) + 1];

strcpy(name, nam);

}

void print() {

cout << "view = " << view << endl << "quantity = " << quantity << endl;

}

void operator = (const Fireplaces& f) {

name = new char[strlen(f.name) + 1];

strcpy(name, name);

}

void operator + (const Fireplaces& f) {

quantity = quantity + f.quantity;

}

void operator \* (const Fireplaces& f) {

quantity = f.quantity \* quantity;

}

void operator - (const Fireplaces& f) {

quantity = f.quantity - quantity;

}

};

class DuHast {

public:

double dudu;

float agaga;

DuHast(double dudu, float agaga) : agaga(agaga), dudu(dudu) {}

void print2() {

cout << "agaga = " << agaga << endl << "dudu = " << dudu << endl;

}

void operator + (const DuHast& dh) {

dudu += dh.dudu;

}

};

class Koleso : public Fireplaces, public DuHast {

private:

int camera;

float mamont;

public:

Koleso(int camera, float mamont, int quantity, int view, double dudu, float agaga) : Fireplaces(quantity,view), DuHast(dudu,agaga), camera(camera),mamont(mamont){

if (dudu > 5000.50) throw k;

}

void print3() {

cout << "camera = " << camera << endl << "mamont = " << mamont << endl;

}

void summa() {

if (agaga > mamont) throw j;

mamont += dudu + agaga;

cout << "Сложение элементов dudu и agaga с элементом mamont в классе Koleso = " << mamont << endl;

}

};

int main()

{

setlocale(LC\_ALL, "russian");

try {

int n;

cout << "Введите n = 1 для ошибки типа J и n = 2 для ошибки K: " << endl;

cin >> n;

Fireplaces fire;

fire.input(10101010, 2020202, (char\*)"printer");

DuHast duhast(8756565.856575, 100.09);

Koleso koleso(8980, 8767.675, 1000, 1203, 453.78685, 10102.7464);

cout << "DUHAST: \_\_\_\_\_\_\_\_\_\_" << endl;

duhast.print2();

cout << endl << "FIRE: \_\_\_\_\_\_\_\_\_\_" << endl;

fire.print();

cout << endl << "KOLESO: \_\_\_\_\_\_\_\_\_\_//элементы класса Koleso" << endl;

koleso.print3();

cout << endl << "KOLESO: \_\_\_\_\_\_\_\_\_\_//элементы класса Fireplaces через конструктор Koleso" << endl;

koleso.print();

cout << endl << "KOLESO: \_\_\_\_\_\_\_\_\_\_//элементы класса DuHast через конструктор Koleso" << endl;

koleso.print2();

if (n == 1) {

cout << endl << "Koleso: \_\_\_\_\_\_\_\_\_\_//вызов функции сумма класса Koleso" << endl;

koleso.summa();

cout << endl << "Koleso: \_\_\_\_\_\_\_\_\_\_ \\после сложения" << endl;

koleso.print3();

}

else Koleso koleso2(8980, 8767.675, 1000, 1203, 55453.78685, 10102.7464);

cout << endl << "FIRE: \_\_\_\_\_\_\_\_\_\_\\после сложения" << endl;

fire + koleso;

fire.print();

cout << endl << "Duhast: \_\_\_\_\_\_\_\_\_\_\\после сложения" << endl;

duhast + koleso;

fire.print();

cout << endl << "FIRE: \_\_\_\_\_\_\_\_\_\_\\после вычитания" << endl;

fire - koleso;

fire.print();

cout << endl;

}

catch (int k) {

cout << "ERROR: Dudu превышает порог допустимого значения у объекта KOLESO2" << endl;

}

catch (float j) {

cout << "ERROR: Значение agaga больше mamont - что недопустимо для данной функции" << endl;

}

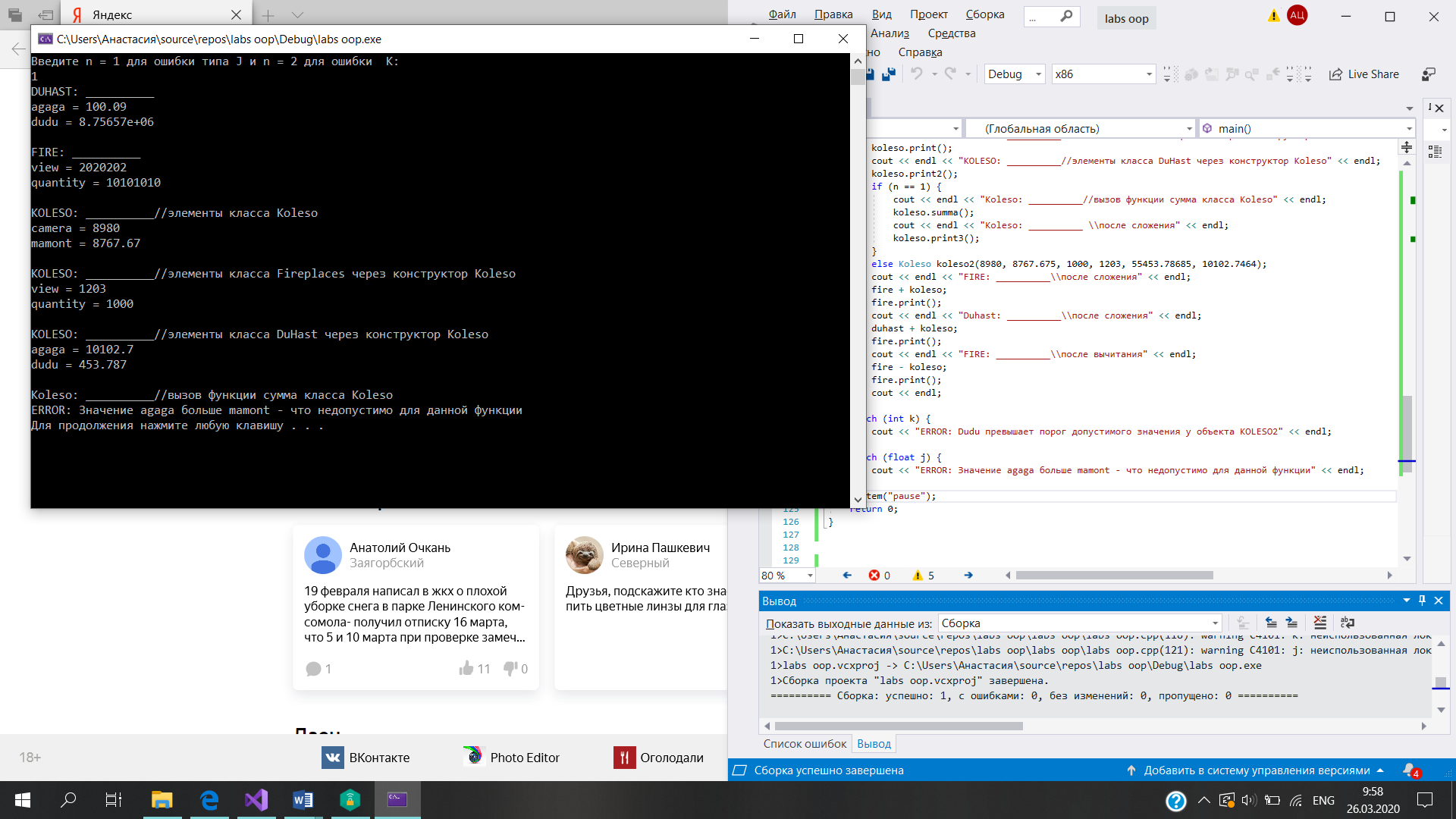
system("pause");

return 0;

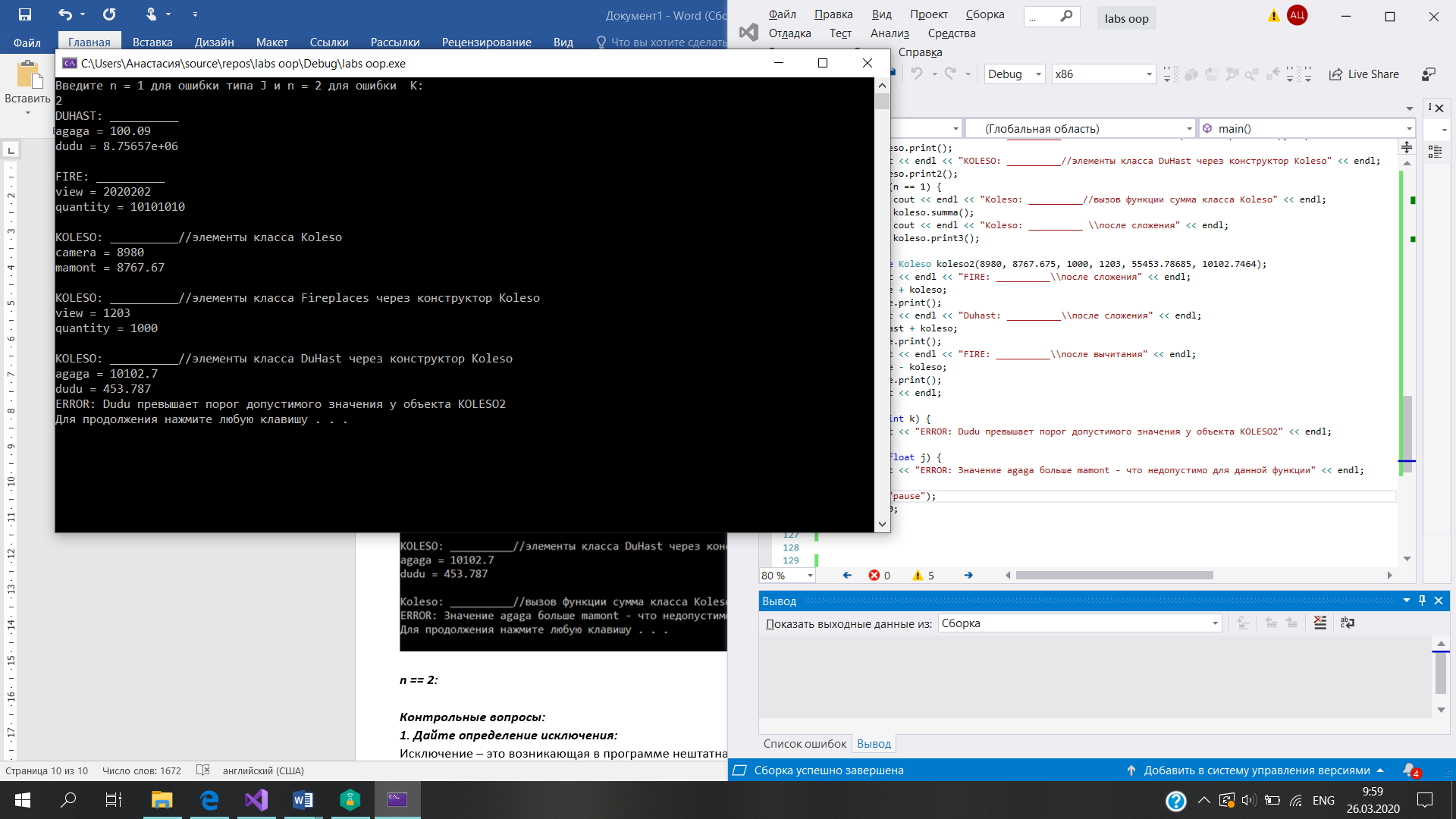
}

***Тестирование:***

***n == 1:***



***n == 2:***



***Контрольные вопросы:***

***1. Дайте определение исключения:***

Исключение – это возникающая в программе нештатная ситуация, с которой программа не может справиться.

***2. В каком блоке описывается список обработчиков для возбужденного исключения:***

Список обработчиков исключений описывается в специальном блоке catch, соответствующем типу возникнувшего исключения. Блок catch всегда описывается после блока try, т.е. вне его.

***3. Опишите синтаксис обработчика исключения:***

Синтаксически обработчик имеет вид:

catch (формальный\_аргумент)

составная\_конструкция

***4. Что такое спецификация исключения:***

Спецификация исключения – список типов, которые может иметь выражение throw внутри функции:

заголовок\_функции throw (список\_типов)

***5. Для каких целей используется функция unexpected:***

Предоставляемая системой функция unexpected() вызывается, когда она возбудила исключение, которое отсутствует в ее списке спецификации исключений.