***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 {

protected:

int quantity;

int view;

char\* name;

public:

Fireplaces(){}

Fireplaces(int quantity, int view, char\* name) : quantity(quantity), view(view), name(name) {}

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

void print();

};

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

quantity = qua;

view = vie;

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

strcpy(name, nam);

}

void Fireplaces::print() {

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

}

class Fire :public Fireplaces {

private:

double brightness;

float heat;

public:

Fire(){}

Fire(int quantity, int view, char\* name, double brightness, float heat) : Fireplaces(quantity, view, name), brightness(brightness), heat(heat) {}

void input(int quantity, int view, char\* name, double brightness, float heat);

void print();

void operator =(const Fire& f);

Fire operator +(const Fire& f);

Fire operator \*(const Fire& f);

};

void Fire::input(int quantity, int view, char\* name, double brightness, float heat) {

Fireplaces::quantity = quantity;

Fireplaces::view = view;

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

strcpy(Fireplaces::name, name);

this->brightness = brightness;

this->heat = heat;

}

void Fire::print() {

cout << "quantity = " << Fireplaces::quantity << endl << "view = " << Fireplaces::view

<< endl << "name = " << Fireplaces::name << endl << "brightness = " << brightness

<< endl << "heat = " << heat << endl;

}

void Fire::operator =(const Fire& f) {

quantity = f.quantity;

view = f.view;

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

strcpy(name, f.name);

brightness = f.brightness;

heat = f.heat;

}

Fire Fire::operator +(const Fire& f) {

Fire fire1(this->quantity + f.quantity, this->view + f.view, strcat(this->name, f.name), this->brightness + f.brightness, this->heat + f.heat);

return fire1;

}

Fire Fire::operator \*(const Fire& f) {

Fire fire1(this->quantity \* f.quantity, this->view \* f.view, strcat(this->name, f.name), this->brightness \* f.brightness, this->heat \* f.heat);

return fire1;

}

int main()

{

setlocale(LC\_ALL, "russian");

char text1[100]; cout << "text1 = "; cin >> text1;

char text2[100]; cout << "text2 = "; cin >> text2;

Fire fire1(10, 20, text1, 18.09, 3);

Fire fire2(2, 3, text2, 98.76, 653);

Fire fire3;

Fire fire4;

cout << "Fire1:" << endl;

fire1.print();

cout << "Fire2:" << endl;

fire2.print();

cout << "Сложение:" << endl;

fire3 = fire1 + fire2;

cout << "Результат сложения:" << endl;

fire3.print();

cout << "Умножение:" << endl;

fire4 = fire1 \* fire2;

cout << "Результат умножения:" << endl;

fire4.print();

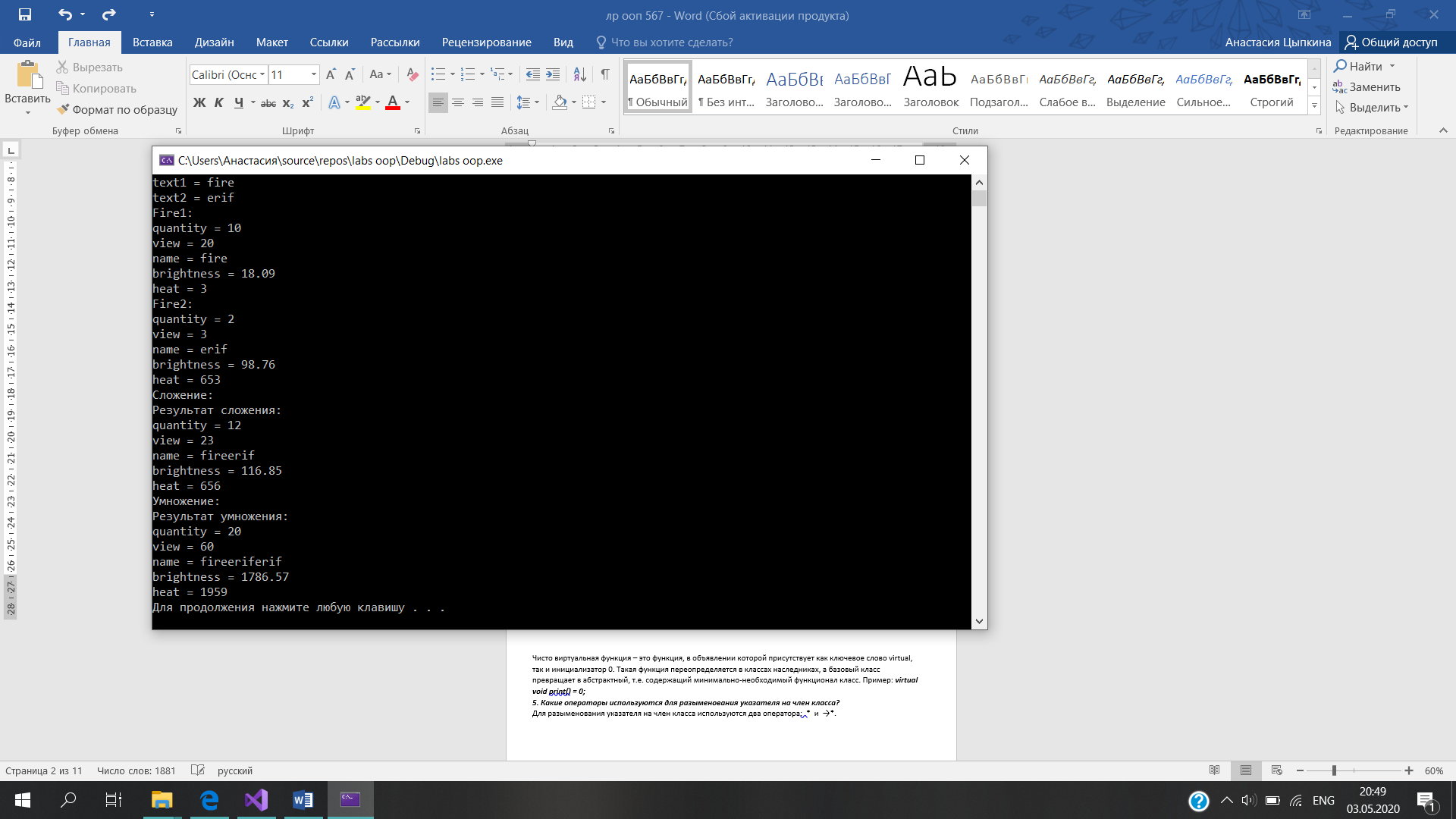
system("pause");

return 0;

}

***Тестирование:***

на след. странице



***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 {

protected:

int quantity;

int view;

char\* name;

public:

Fireplaces(){}

Fireplaces(int quantity, int view, char\* name) : quantity(quantity), view(view), name(name) {}

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

void print();

};

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

quantity = qua;

view = vie;

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

strcpy(name, nam);

}

void Fireplaces::print() {

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

}

class Fire {

protected:

double brightness;

float heat;

public:

Fire(){}

Fire(double brightness, float heat) : brightness(brightness), heat(heat) {}

void input(int quantity, int view, char\* name, double brightness, float heat);

void print();

};

void Fire::input(int quantity, int view, char\* name, double brightness, float heat) {

this->brightness = brightness;

this->heat = heat;

}

void Fire::print() {

cout << "brightness = " << brightness << endl << "heat = " << heat << endl;

}

class Billets : public Fire, public Fireplaces {

protected:

bool wet;

int count;

public:

Billets(){}

Billets(int quantity, int view, char\* name, double brightness, float heat, bool wet, int count) : Fireplaces(quantity, view, name), Fire(brightness, heat), wet(wet), count(count) {}

void input(int quantity, int view, char\* name, double brightness, float heat, bool wet, int count);

void print();

void operator =(const Billets& b);

Billets operator +(const Billets& b);

Billets operator \*(const Billets& b);

};

void Billets::input(int quantity, int view, char\* name, double brightness, float heat, bool wet, int count) {

Fireplaces::quantity = quantity;

Fireplaces::view = view;

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

strcpy(Fireplaces::name, name);

Fire::brightness = brightness;

Fire::heat = heat;

this->wet = wet;

this->count = count;

}

void Billets::print() {

cout << "quantity = " << quantity << endl << "view = " << view << endl << "name = "

<< name << endl << "brightness = " << brightness << endl << "heat = " << heat

<< endl << "wet = " << wet << endl << "count = " << count << endl;

}

void Billets::operator =(const Billets& b) {

quantity = b.quantity;

view = b.view;

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

strcpy(name, b.name);

brightness = b.brightness;

heat = b.heat;

wet = b.wet;

count = b.count;

}

Billets Billets::operator +(const Billets& b) {

Billets bil1(this->quantity + b.quantity, this->view + b.view, strcat(this->name, b.name), this->brightness + b.brightness, this->heat + b.heat, this->wet + b.wet, this->count + b.count);

return bil1;

}

Billets Billets::operator \*(const Billets& b) {

Billets bil1(this->quantity \* b.quantity, this->view \* b.view, strcat(this->name, b.name), this->brightness \* b.brightness, this->heat \* b.heat, this->wet \* b.wet, this->count \* b.count);

return bil1;

}

int main()

{

setlocale(LC\_ALL, "russian");

char text1[100]; cout << "text1 = "; cin >> text1;

char text2[100]; cout << "text2 = "; cin >> text2;

Billets bil1(10, 20, text1, 18.09, 3, 0, 10);

Billets bil2(2, 3, text2, 98.76, 653, 1, 900);

Billets bil3;

Billets bil4;

cout << "Bil1:" << endl;

bil1.print();

cout << "Bil2:" << endl;

bil2.print();

cout << "Сложение:" << endl;

bil3 = bil1 + bil2;

cout << "Результат сложения:" << endl;

bil3.print();

cout << "Умножение:" << endl;

bil4 = bil1 \* bil2;

cout << "Результат умножения:" << endl;

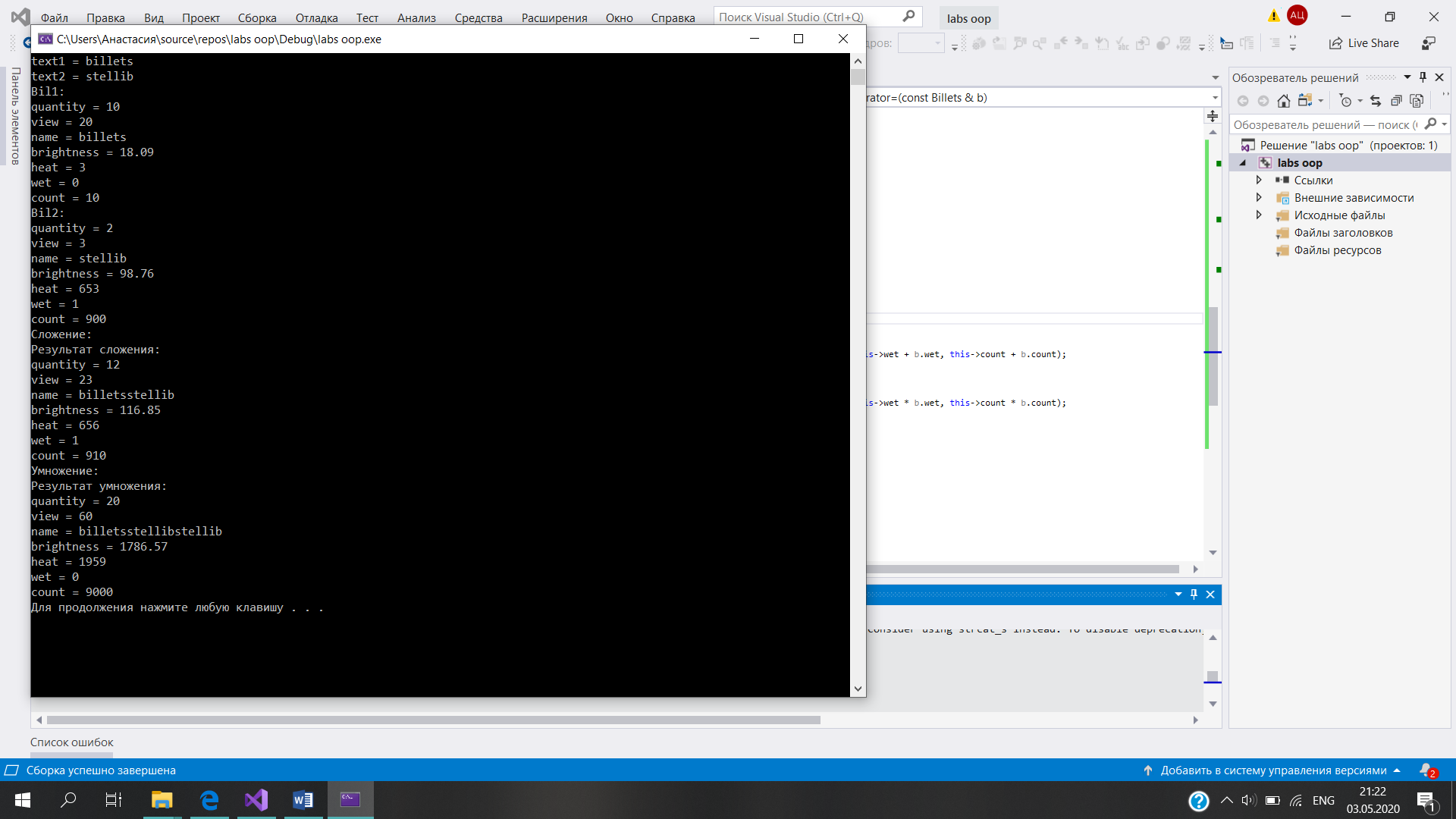
bil4.print();

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 s;

float s1;

class Fireplaces {

protected:

int quantity;

int view;

char\* name;

public:

Fireplaces(){}

Fireplaces(int quantity, int view, char\* name) : quantity(quantity), view(view), name(name) {}

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

void print();

};

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

quantity = qua;

view = vie;

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

strcpy(name, nam);

}

void Fireplaces::print() {

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

}

class Fire {

protected:

double brightness;

float heat;

public:

Fire(){}

Fire(double brightness, float heat) : brightness(brightness), heat(heat) {}

void input(int quantity, int view, char\* name, double brightness, float heat);

void print();

};

void Fire::input(int quantity, int view, char\* name, double brightness, float heat) {

this->brightness = brightness;

this->heat = heat;

}

void Fire::print() {

cout << "brightness = " << brightness << endl << "heat = " << heat << endl;

}

class Billets : public Fire, public Fireplaces {

public:

bool wet;

int count;

Billets(){}

Billets(int quantity, int view, char\* name, double brightness, float heat, bool wet, int count) : Fireplaces(quantity, view, name), Fire(brightness, heat), wet(wet), count(count) {}

void input(int quantity, int view, char\* name, double brightness, float heat, bool wet, int count);

void print();

void operator =(const Billets& b);

Billets operator +(const Billets& b);

Billets operator \*(const Billets& b);

};

void Billets::input(int quantity, int view, char\* name, double brightness, float heat, bool wet, int count) {

Fireplaces::quantity = quantity;

Fireplaces::view = view;

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

strcpy(Fireplaces::name, name);

Fire::brightness = brightness;

Fire::heat = heat;

this->wet = wet;

this->count = count;

}

void Billets::print() {

cout << "quantity = " << quantity << endl << "view = " << view << endl << "name = "

<< name << endl << "brightness = " << brightness << endl << "heat = " << heat

<< endl << "wet = " << wet << endl << "count = " << count << endl;

}

void Billets::operator =(const Billets& b) {

quantity = b.quantity;

view = b.view;

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

strcpy(name, b.name);

brightness = b.brightness;

heat = b.heat;

wet = b.wet;

count = b.count;

}

Billets Billets::operator +(const Billets& b) {

Billets bil1(this->quantity + b.quantity, this->view + b.view, strcat(this->name, b.name), this->brightness + b.brightness, this->heat + b.heat, this->wet + b.wet, this->count + b.count);

return bil1;

}

Billets Billets::operator \*(const Billets& b) {

Billets bil1(this->quantity \* b.quantity, this->view \* b.view, strcat(this->name, b.name), this->brightness \* b.brightness, this->heat \* b.heat, this->wet \* b.wet, this->count \* b.count);

return bil1;

}

int main()

{

try {

setlocale(LC\_ALL, "russian");

char text1[100]; cout << "text1 = "; cin >> text1;

char text2[100]; cout << "text2 = "; cin >> text2;

int n; cout << "Тип ошибки (1 или 2) = "; cin >> n;

Billets bil1;

switch (n) {

case 1: {

bil1.input(10, 20, text1, 18.09, 3, 0, 10);

break;

}

case 2: {

bil1.input(10, 20, text1, 18.09, 3, 0, -10);

break;

}

}

Billets bil2(2, 3, text2, 98.76, 653, 1, 900);

Billets bil3;

Billets bil4;

cout << "Bil1:" << endl;

bil1.print();

if (bil1.count <= 0) throw s;

cout << "Bil2:" << endl;

bil2.print();

cout << "Сложение:" << endl;

bil3 = bil1 + bil2;

cout << "Результат сложения:" << endl;

bil3.print();

cout << "Умножение:" << endl;

bil4 = bil1 \* bil2;

cout << "Результат умножения:" << endl;

bil4.print();

if (bil4.count >= 9000) throw s1;

}

catch (int s) {

cout << "ERROR. Значение переменной count не может быть отрицательным." << endl;

}

catch (float s1) {

cout << "ERROR.Значение переменной count больше, либо равно 9000." << endl;

}

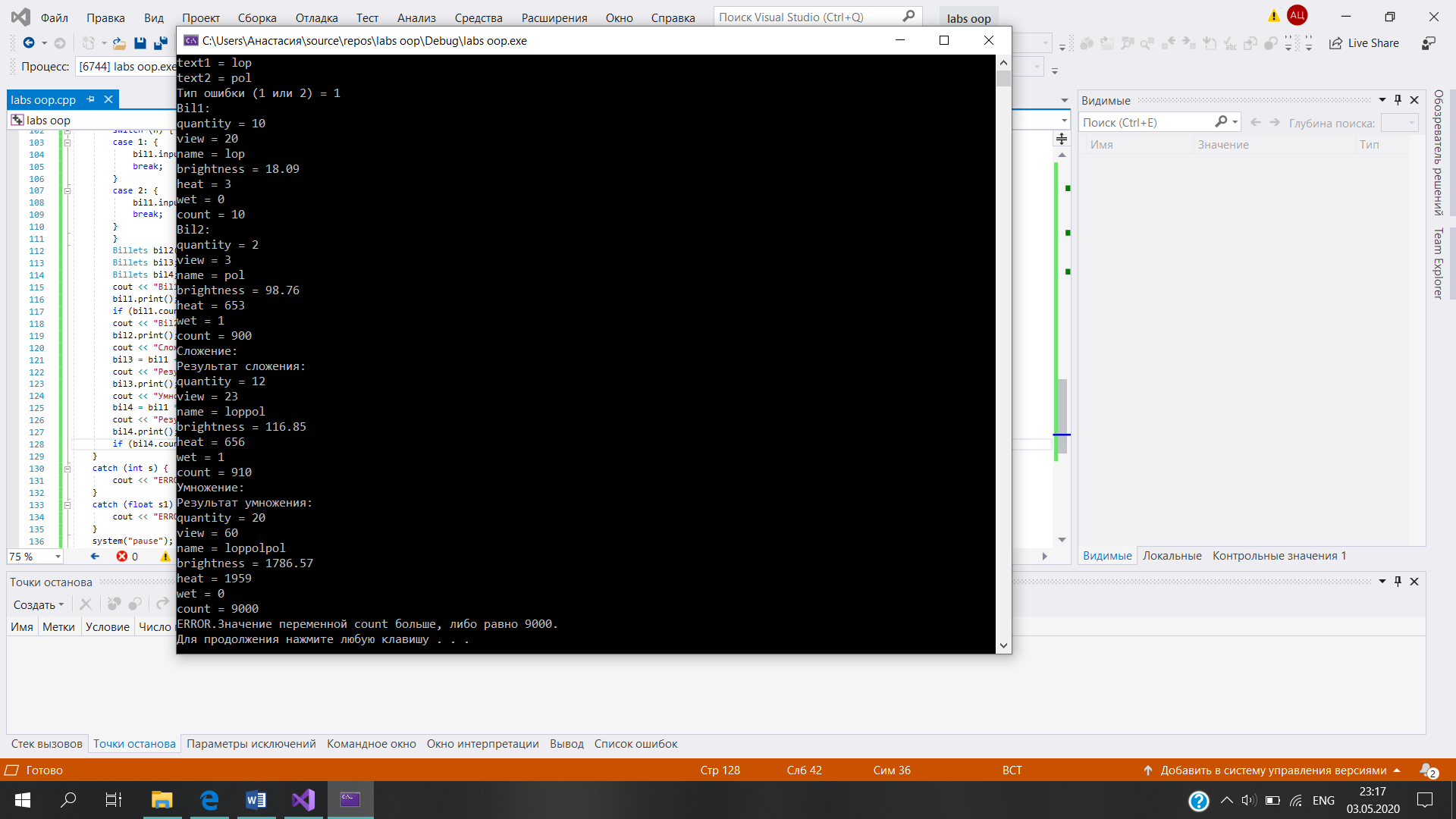
system("pause");

return 0;

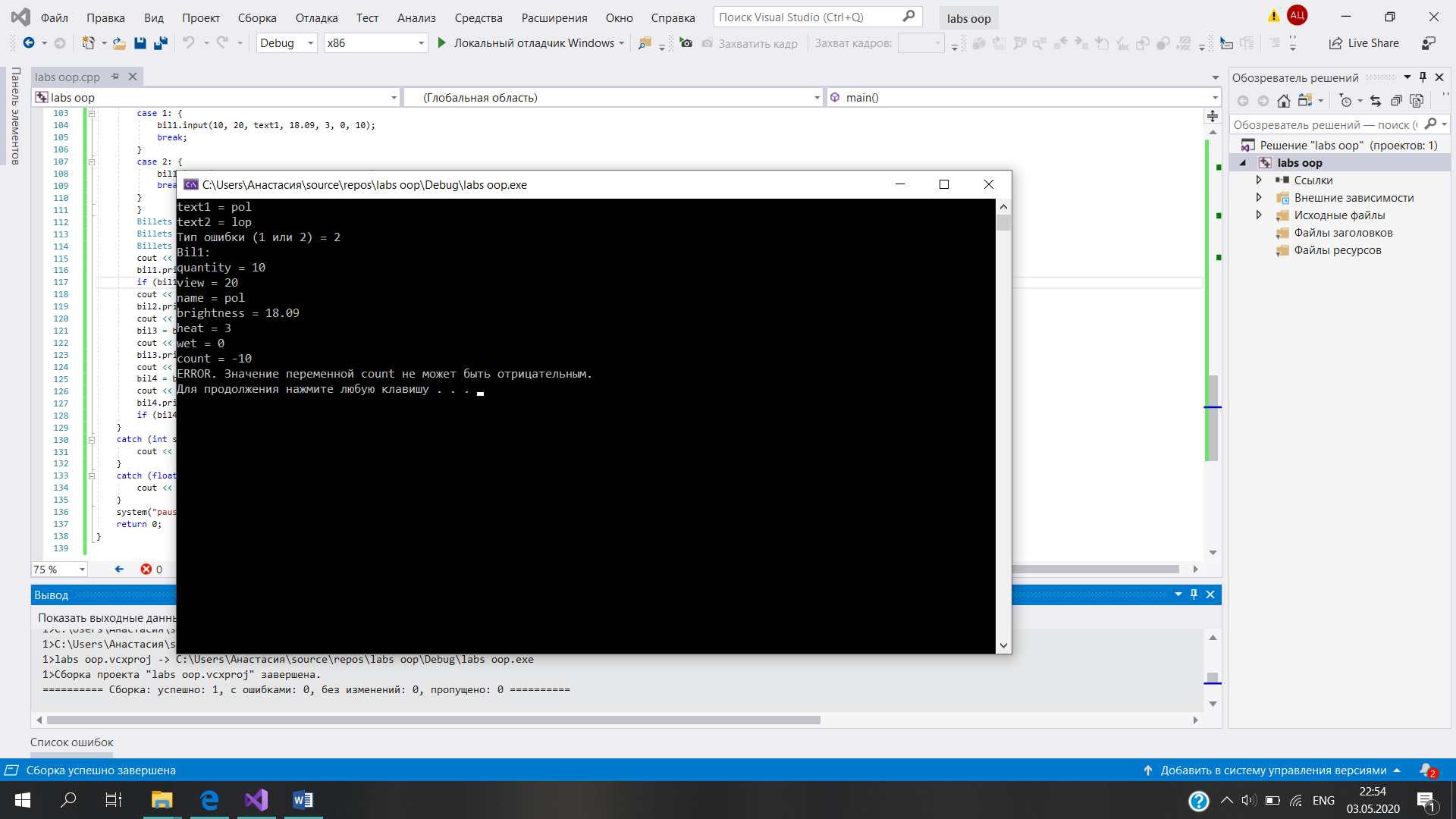
}

***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.