МИНОБРНАУКИ РОССИИ

ФГБОУ ВО Череповецкий государственный университет

Институт информационных технологий

Кафедра: Математическое и программное обеспечение ЭВМ

Дисциплина: Объектно-ориентированное программирование

ЛАБОРАТОРНАЯ РАБОТА № 7

Программирование обработки исключительных ситуаций

Выполнил:

студент гр.

1ПИб-01-21оп

МартынюкМ.В.

Проверил:

ЕршовЕ.В.

Череповец, 2020 г.

**Цель работы:**

Изучить синтаксис и семантику определения и вызова исключений, синтаксис обработчика и спецификации исключений; приобрести практические навыки запуска исключений; изучить особенности применения стандартных библиотечных исключений.

**Задание:**

Используя модифицированные классы из 5 или 6 лабораторной работы, обработайте все возможные исключительные ситуации. Проверьте работоспособность АТД на тестовом наборе данных.

Возможные исключительные ситуации:

1. ввод отрицательных значений
2. попытка использования не выделенной памяти с помощью оператора new

**Код программы:**

#include"iostream"

#include<string>

#include<string.h>

#include<cstring>

#include<new>

using namespace std;

class bad\_value : public exception {};

template<class T>

class printer {

public:

printer(); //по умолчанию

printer(int a, int n, T b, T c, T d, T e, char\* f, char\* g); //конструктор с параметрами

printer(const printer& m); //копирующий конструктор

printer& operator=(const printer& m);

~printer(); //деструктор

void print();

printer& operator+=(const printer& m2);

printer& operator>(const printer& m);

printer& operator+(const printer& m);

protected:

int font\_size;//размершрифта

int number\_of\_pages;//количествостраниц

T indent\_on\_the\_left;//отступслева

T indent\_on\_the\_right;//отступсправа

T indent\_on\_the\_from\_above;//отступсверху

T indent\_on\_the\_from\_below;//отступснизу

char\* color;//цветшрифта

char\* text;// текст

private:

};

template<class T>printer<T>::printer() {

font\_size = 1;

number\_of\_pages = 2;

indent\_on\_the\_left = 3;

indent\_on\_the\_right = 4;

indent\_on\_the\_from\_above = 5;

indent\_on\_the\_from\_below = 6;

char f[100] = "not text";

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

strcpy(color, f);

char g[100] = "not text";

text = new char[strlen(g) + 1];

strcpy(text, g);

}

template<class T>printer<T>::printer(int a, int n, T b, T c, T d, T e, char\* f, char\* g) {

if (a < 0) throw bad\_value();

font\_size = a;

if (n < 0) throw bad\_value();

number\_of\_pages = n;

if (b < 0) throw bad\_value();

indent\_on\_the\_left = b;

if (c < 0) throw bad\_value();

indent\_on\_the\_right = c;

if (d < 0) throw bad\_value();

indent\_on\_the\_from\_above = d;

if (e < 0) throw bad\_value();

indent\_on\_the\_from\_below = e;

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

strcpy(color, f);

text = new char[strlen(g) + 1];

strcpy(text, g);

}

template<class T>printer<T>::printer(const printer<T>& m) {

if (&m != this) {

font\_size = m.font\_size;

number\_of\_pages = m.number\_of\_pages;

indent\_on\_the\_left = m.indent\_on\_the\_left;

indent\_on\_the\_right = m.indent\_on\_the\_right;

indent\_on\_the\_from\_above = m.indent\_on\_the\_from\_above;

indent\_on\_the\_from\_below = m.indent\_on\_the\_from\_below;

color = new char[strlen(m.color) + 1];

strcpy(color, m.color);

text = new char[strlen(m.text) + 1];

strcpy(text, m.text);

}

}

template <class T> printer<T>::~printer() { delete[]color; delete[]text; }

template<class T>printer<T>& printer<T>::operator=(const printer<T>& m) {

if (&m != this) { //если адрес не равен содержимому адресу this А если равен ?

font\_size = m.font\_size;

number\_of\_pages = m.number\_of\_pages;

indent\_on\_the\_left = m.indent\_on\_the\_left;

indent\_on\_the\_right = m.indent\_on\_the\_right;

indent\_on\_the\_from\_above = m.indent\_on\_the\_from\_above;

indent\_on\_the\_from\_below = m.indent\_on\_the\_from\_below;

color = new char[strlen(m.color) + 1];

strcpy(color, m.color);

text = new char[strlen(m.text) + 1];

strcpy(text, m.text);

}

return \*this;

}

template<class T>printer<T>& printer<T>::operator>(const printer<T>& m)

{

if (font\_size < m.font\_size) cout << "Размершрифта = small" << endl;

if (font\_size > m.font\_size) cout << "Размершрифта = big" << endl;

else cout << "Размершрифта = ravni" << endl;

if (number\_of\_pages < m.number\_of\_pages) cout << "Количествостраниц = small" << endl;

if (number\_of\_pages > m.number\_of\_pages) cout << "Количествостраниц = big" << endl;

else cout << "Количество страниц = ravni" << endl;

if (indent\_on\_the\_left < m.indent\_on\_the\_left)cout << "Отступслева = small" << endl;

if (indent\_on\_the\_left > m.indent\_on\_the\_left)cout << "Отступслева = big" << endl;

else cout << "Отступслева = ravni" << endl;

if (indent\_on\_the\_right < m.indent\_on\_the\_right)cout << "Отступсправа = small" << endl;

if (indent\_on\_the\_right > m.indent\_on\_the\_right)cout << "Отступсправа = big" << endl;

else cout << "Отступсправа = ravni" << endl;

if (indent\_on\_the\_from\_above < m.indent\_on\_the\_from\_above)cout << "Отступсверху = small" << endl;

if (indent\_on\_the\_from\_above > m.indent\_on\_the\_from\_above)cout << "Отступсверху = big" << endl;

else cout << "Отступсверху = ravni" << endl;

if (indent\_on\_the\_from\_below < m.indent\_on\_the\_from\_below)cout << "Отступснизу = small" << endl;

if (indent\_on\_the\_from\_below > m.indent\_on\_the\_from\_below)cout << "Отступснизу = big" << endl;

else cout << "Отступснизу = ravni" << endl;

if (strlen(text) < strlen(m.text))cout << "Длинатекста = small" << endl;

if (strlen(text) > strlen(m.text))cout << "Длинатекста = big" << endl;

else cout << "Длина текста = ravni" << endl;

return (\*this);

}

template<class T>printer<T>& printer<T>::operator+=(const printer<T>& m)

{

int dlin;

font\_size = font\_size + m.font\_size;

number\_of\_pages = number\_of\_pages + m.number\_of\_pages;

indent\_on\_the\_left = indent\_on\_the\_left + m.indent\_on\_the\_left;

indent\_on\_the\_right = indent\_on\_the\_right + m.indent\_on\_the\_right;

indent\_on\_the\_from\_above = indent\_on\_the\_from\_above + m.indent\_on\_the\_from\_above;

indent\_on\_the\_from\_below = indent\_on\_the\_from\_below + m.indent\_on\_the\_from\_below;

dlin = strlen(color) + strlen(m.color);

if (strlen(color) < strlen(m.color)) color = new char[strlen(m.color) + 1];

else color = new char[strlen(color) + 1];

sprintf(color, "%d", dlin);

dlin = strlen(text) + strlen(m.text);

if (strlen(text) < strlen(m.text)) text = new char[strlen(m.text) + 1];

else text = new char[strlen(text) + 1];

sprintf(text, "%d", dlin);

return (\*this);

}

template<class T>printer<T>& printer<T>::operator+(const printer<T>& m)

{

printer n;

int dlin;

n.font\_size = font\_size + m.font\_size;

n.number\_of\_pages = number\_of\_pages + m.number\_of\_pages;

n.indent\_on\_the\_left = indent\_on\_the\_left + m.indent\_on\_the\_left;

n.indent\_on\_the\_right = indent\_on\_the\_right + m.indent\_on\_the\_right;

n.indent\_on\_the\_from\_above = indent\_on\_the\_from\_above + m.indent\_on\_the\_from\_above;

n.indent\_on\_the\_from\_below = indent\_on\_the\_from\_below + m.indent\_on\_the\_from\_below;

dlin = strlen(color) + strlen(m.color);

if (strlen(color) < strlen(m.color)) n.color = new char[strlen(m.color) + 1];

else color = new char[strlen(color) + 1];

sprintf(n.color, "%d", dlin);

dlin = strlen(text) + strlen(m.text);

if (strlen(text) < strlen(m.text)) n.text = new char[strlen(m.text) + 1];

else n.text = new char[strlen(text) + 1];

sprintf(n.text, "%d", dlin);

return n;

}

template<class T>void printer<T>::print() { //определениефункции

cout << "Размершрифта " << font\_size << endl;//размершрифта

cout << "Количествостраниц " << number\_of\_pages << endl;

cout << "Отступслева " << indent\_on\_the\_left << endl;//отступслева

cout << "Отступсправа " << indent\_on\_the\_right << endl;//отступсправа

cout << "Отступсверху " << indent\_on\_the\_from\_above << endl;//отступсверху

cout << "Отступснизу " << indent\_on\_the\_from\_below << endl;//отступснизу

cout << "Цветшрифта " << color << endl;//цветшрифта

cout << "Текст " << text << endl;// текст

}

//-------------------------------сканер------------------------------------------

class scaner {

public://копирующий конструктор и деструктор, вызывающий деструктор базового класса

scaner();

scaner(int c\_c, char\* text);

scaner(const scaner& m); //копирующий конструктор

void print();

~scaner();

private:

int s\_s=1.5; //скорость сканирования

int g\_c=24; //глубина цвета

protected:

int c\_c; //количество копий

char\* text; //просканированный текст

};

scaner::scaner() {

c\_c = 0;

char f[100] = "not text";

text = new char[strlen(f) + 1];; //просканированный текст

strcpy(text, f);

}

scaner::scaner(int a, char\* b) {

if (a < 0) throw bad\_value();

c\_c = a;

text = new char[strlen(b) + 1]; //просканированный текст

strcpy(text, b);

}

scaner::~scaner() { delete[]text; } //!!!!

scaner::scaner(const scaner& m) {

if (&m != this) {

c\_c = m.c\_c;

text = new char[strlen(m.text) + 1];

strcpy(text, m.text);

}

}

void scaner::print() {

cout << "Cкорость сканирования" << s\_s << endl;

cout << "Глубина цвета " << g\_c << endl;

cout << "Количество копий " << c\_c << endl;

cout << "Просканированный текст " << text << endl;

}

//-------------------------------объединение----------------------------------------

class printer\_and\_scaner : public printer<int>, public scaner {

public://копирующий конструктор и деструктор, вызывающий деструктор базового класса

printer\_and\_scaner();

printer\_and\_scaner(int a, int n, int b, int c, int d, int e, char\* f, char\* g, int c\_c, char\* text, int k); //k -kol\_ener, s - skorost

printer\_and\_scaner(const printer\_and\_scaner& m); //копирующий конструктор

void print();

~printer\_and\_scaner();

private:

int kol\_ener;

};

printer\_and\_scaner::printer\_and\_scaner() :printer(), scaner() {

kol\_ener = 360;

}

printer\_and\_scaner::printer\_and\_scaner(int a, int n, int b, int c, int d, int e, char\* f, char\* g, int c\_c, char\* text, int k) : printer<int>(a, n, b, c, d, e, f, g), scaner(c\_c, text) {

if (k < 0) throw bad\_value();

kol\_ener = k;

}

printer\_and\_scaner::~printer\_and\_scaner() {

printer::~printer();

scaner::~scaner();

}

printer\_and\_scaner::printer\_and\_scaner(const printer\_and\_scaner& m) {

if (&m != this) {

printer::printer();

scaner::scaner();

kol\_ener = m.kol\_ener;

}

}

void printer\_and\_scaner::print() {

printer::print();

scaner::print();

cout << "kol\_ener " << kol\_ener << endl;

}

int main() {

setlocale(LC\_ALL, "Russian");

char col[10] = "red";

char tex[100] = "redtextred";

char tex1[100] = "redstextred";

bool exception;

try

{

cout << "Вводим информацию о первом объекте" << endl << endl;

printer\_and\_scaner print1(12, -25, 2.5, 3.2, 1.2, 2.5, col, tex, 20, tex1, 360);

cout << "Выводим информацию о первом объекте:" << endl << endl;

print1.print();

cout << endl;

cout << "Копируем информацию из первого объекта во второй" << endl << endl;

printer\_and\_scaner print2(print1);

cout << "Выводим информацию о втором объекте:" << endl << endl;

print2.printer::print();

print2.scaner::print();

}

catch (bad\_value& e) {

cout << "Введено неверное значение." << endl;

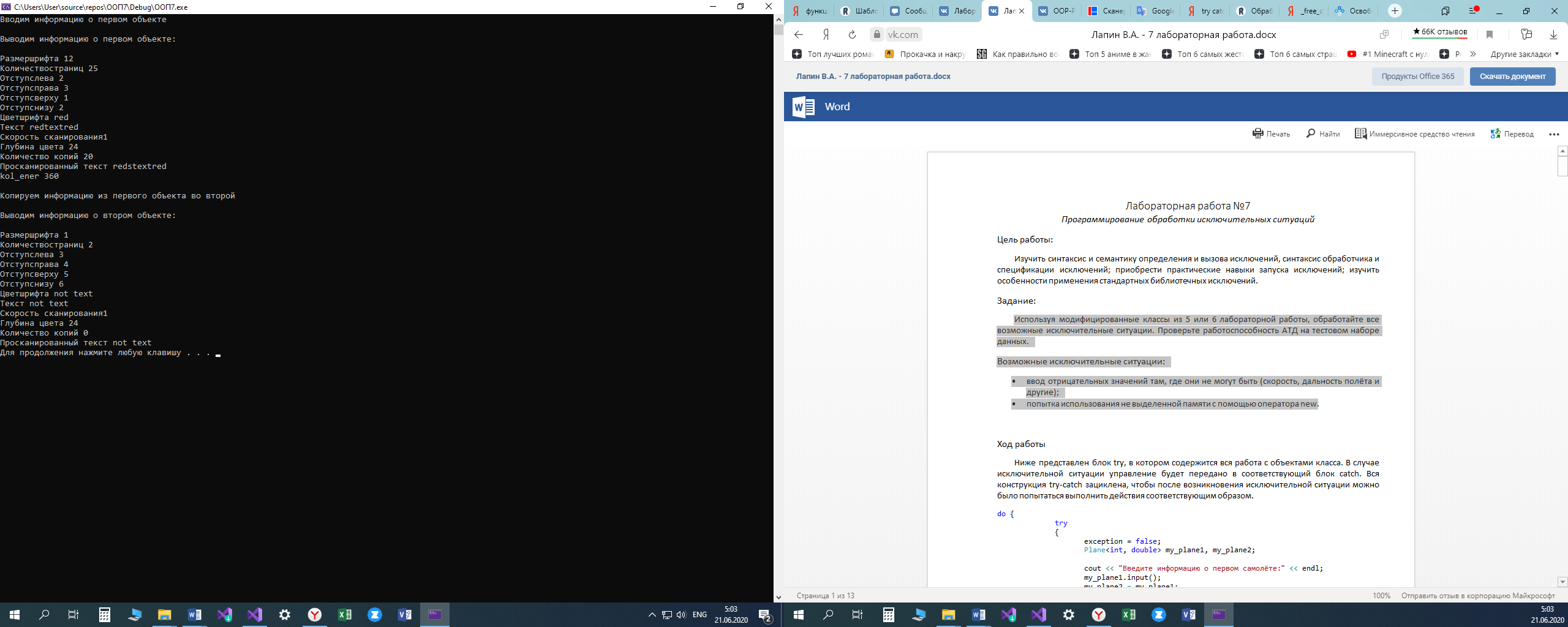
}

system("pause");

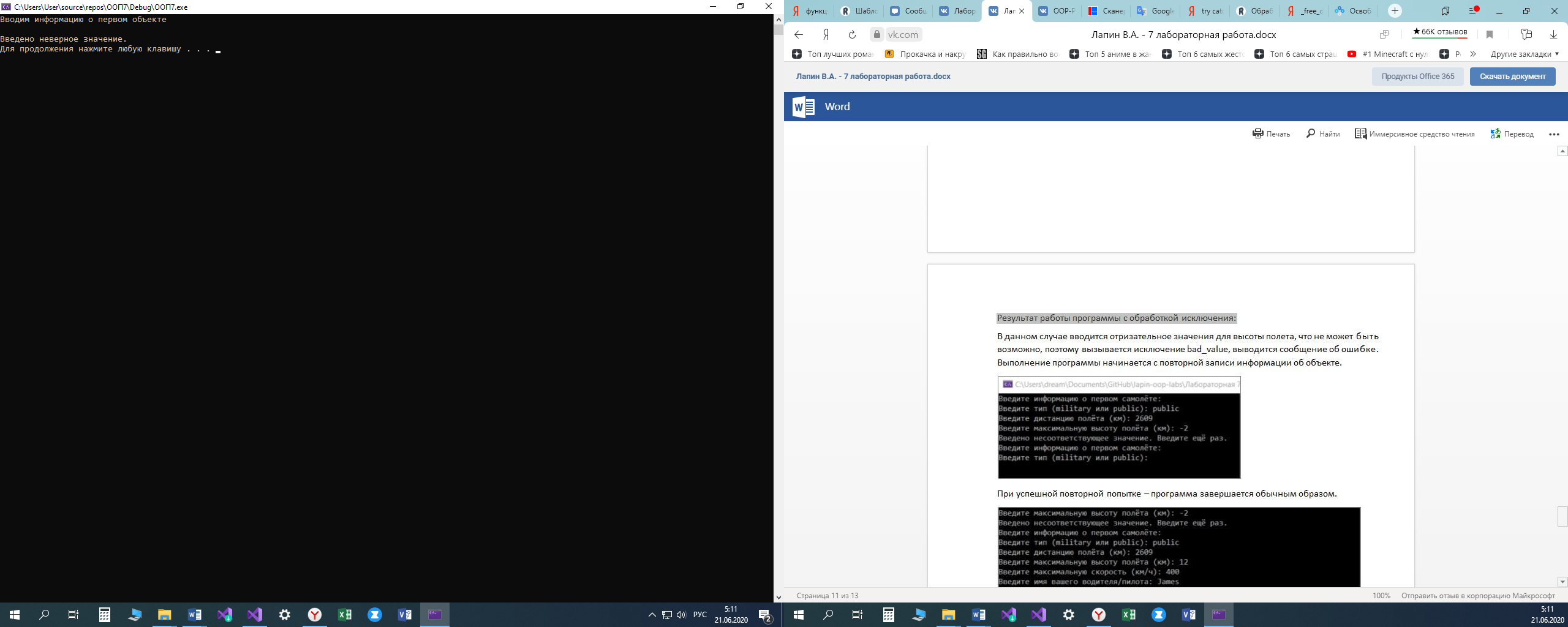
}

**Тестирование программы:**

Результат работы программы:



Результат работы программы с обработкой исключения:



**Контрольные вопросы:**

1.Дайте определение исключения.

Это возникающая в программе нештатная ситуация, с которой программа не может справиться. Например, при делении на ноль выполнение программы аварийно завершается системой. С++ дает возможность восстанавливать программу из ошибочных ситуаций и продолжать ее выполнение.

2.В каком блоке описывается список обработчиков для возбужденного исключения?

Блок try является контекстом для определения того, какие обработчики вызываются при возбуждении исключения. Порядок, в котором определены обработчики, задает очередность проверки обработчиков подходящего типа для возбуждения исключения.

3. Опишите синтаксис обработчика исключения.

catch (формальный\_аргумент) { составная\_конструкция }

Обработчик catch выглядит как объявление функции одного аргумента без возвращаемого типа. Например:

catch (const char\* message)

{

cerr << message << endl;

exit (1);

}

catch (…) //будет выполнено действие по умолчанию

{

cerr << “Вот и все!” << endl;

abort();

}

4.Что такое спецификация исключения?

Синтаксически спецификация исключения является частью объявления и определения функции и имеет следующий вид:

заголовок\_функции throw (список\_типов)

Здесь список\_типов – это список типов, которые может иметь выражение throw внутри функции. В объявлении и в определении функции спецификация исключения должна записываться одинаково. Если список пуст, компилятор полагает, что функцией не будет выполняться никакой throw (ни прямо, ни косвенно). Если спецификация исключения опущена, то предполагается, что такой функцией может быть возбуждено произвольное исключение.

5.Для каких целей используется функция unexpected?

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