Лабораторная работа 7

**ПРОГРАММИРОВАНИЕ ОБРАБОТКИ**

**ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ**

**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 модифицированный АТД, обработайте все возможные исключительные ситуации.

**Ход работы:**

В качестве исключительных ситуаций было выбрано две ситуации:

1. При создании или изменении объекта, у любого из обрабатываемых объектов отсутствует имя или страна-производитель;
2. Момент, когда количество предметов мебели в объекте подкласса Setofitems превышает 50 или меньше 1;

**Код программы:**

#include "stdafx.h"

#include <iostream>

#include <string>

using namespace std;

class Setinterface {

public:

virtual void additems(int s)=0;

virtual void definesepsale(bool k) = 0;

};

template <class T>

class Offurniture {

private:

T ID;

protected:

char \*name;

char \*country;

T cost;

public:

Offurniture();

Offurniture(char \*n);

Offurniture(char \*n, T a, char \*co, T c);

Offurniture(char \*n, char \*co, T c);

void print()const;

Offurniture(const Offurniture &m);

void input(char \*n, T a, char \*co, T c);

Offurniture &copy(const Offurniture &m);

Offurniture sum(Offurniture b);

bool comp(Offurniture b);

~Offurniture();

Offurniture operator +(const Offurniture m);

Offurniture operator +(T m);

Offurniture operator -(const Offurniture m);

Offurniture operator -(T m);

Offurniture operator =(const Offurniture m);

Offurniture operator =(T m);

bool operator <(Offurniture m);

bool operator <(T m);

bool operator >(Offurniture m);

bool operator >(T m);

bool operator ==(Offurniture m);

bool operator ==(T m);

};

class Setofitems : public Offurniture<double>, virtual public Setinterface {

private:

Offurniture::name;

Offurniture::country;

Offurniture::cost;

int amountofitems = 0;

bool partsale = 0;

public:

void print();

Setofitems(char\* n, char\* co, double c, int aoi, bool ps);

Setofitems();

Setofitems(const Setofitems& m);

Setofitems &copy(Setofitems &m);

void additems(int s) {

if (amountofitems+s > 50 || amountofitems + s <1) throw 1;

amountofitems += s;

};

void definesepsale(bool k) {

partsale = k;

};

Setofitems summary(Setofitems b);

void input(char\* n, char\* co, double c, int aoi, bool ps);

};

Setofitems Setofitems::summary(const Setofitems b) {

Setofitems h;

h.cost = cost + b.cost;

strcpy(h.name, "-");

strcpy(h.country, "-");

if (amountofitems+b.amountofitems > 50 || amountofitems + b.amountofitems <1) throw 1;

h.amountofitems = amountofitems + b.amountofitems;

h.partsale = partsale + b.partsale;

return h;

}

template <class T>

bool Offurniture<T>::comp(Offurniture<T> b) {

return (cost > b.cost);

}

template <class T>

Offurniture<T> &Offurniture<T>::copy(const Offurniture<T> &m) {

if (m.name == NULL || m.country == NULL) throw 2;

if (&m != this) {

delete[]name;

delete[]country;

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

strcpy(name, m.name);

ID = m.ID;

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

strcpy(country, m.country);

cost = m.cost;

}

return(\*this);

}

template <class T>

void Offurniture<T>::input(char \*n, T a, char \*co, T c) {

if (n == NULL) throw 2;

if (co == NULL) throw 2;

strcpy(name, n);

ID = a;

strcpy(country, co);

cost = c;

}

template <class T>

Offurniture<T>::Offurniture<T>() {

name = new char[3];

strcpy(name, "-");

ID = 0;

country = new char[3];

strcpy(country, "-");

cost = 0;

}

template <class T>

Offurniture<T>::Offurniture<T>(char\* n) {

if (n == NULL) throw 2;

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

strcpy(name, n);

ID = 0;

country = new char[2];

strcpy(country, "-");

cost = 0;

}

template <class T>

Offurniture<T>::~Offurniture<T>() {

delete[]name;

delete[]country;

}

template <class T>

Offurniture<T> Offurniture<T>::sum(const Offurniture<T> b) {

Offurniture h;

h.cost = cost + b.cost;

strcpy(h.name, "-");

strcpy(h.country, "-");

return h;

}

// Перегрузка операторов

template <class T>

Offurniture<T> Offurniture<T>::operator +(const Offurniture<T> m) {

Offurniture h;

h.cost = cost + m.cost;

strcpy(h.name, "-");

strcpy(h.country, "-");

return h;

}

template <class T>

Offurniture<T> Offurniture<T>::operator +(T m) {

cost = cost + m;

return(\*this);

}

template <class T>

Offurniture<T> Offurniture<T>::operator -(const Offurniture<T> m) {

Offurniture h;

h.cost = cost - m.cost;

strcpy(h.name, "-");

strcpy(h.country, "-");

return h;

}

template <class T>

Offurniture<T> Offurniture<T>::operator -(T m) {

cost = cost - m;

return(\*this);

}

template <class T>

Offurniture<T> Offurniture<T>::operator =(const Offurniture<T> m) {

strcpy(name, m.name);

ID = m.ID;

strcpy(country, m.country);

cost = m.cost;

return(\*this);

}

template <class T>

Offurniture<T> Offurniture<T>::operator =(T m) {

cost = m;

return(\*this);

}

template <class T>

bool Offurniture<T>::operator <(const Offurniture<T> m) {

return(cost<m.cost);

}

template <class T>

bool Offurniture<T>::operator <(T m) {

return(cost<m);

}

template <class T>

bool Offurniture<T>::operator >(const Offurniture<T> m) {

return(cost>m.cost);

}

template <class T>

bool Offurniture<T>::operator >(T m) {

return(cost>m);

}

template <class T>

bool Offurniture<T>::operator ==(const Offurniture<T> m) {

return(cost == m.cost);

}

template <class T>

bool Offurniture<T>::operator ==(T m) {

return(cost == m);

}

template <class T>

Offurniture<T>::Offurniture<T>(const Offurniture<T> &m) {

if (m.name == NULL) throw 2;

if (m.country == NULL) throw 2;

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

strcpy(name, m.name);

ID = m.ID;

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

strcpy(country, m.country);

cost = m.cost;

}

template <class T>

Offurniture<T>::Offurniture<T>(char \*n, T a, char \*co, T c) {

if (n == NULL) throw 2;

if (co == NULL) throw 2;

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

strcpy(name, n);

ID = a;

country = new char[strlen(co) + 1];

strcpy(country, co);

cost = c;

}

template <class T>

Offurniture<T>::Offurniture<T>(char \*n, char \*co, T c) {

if (n == NULL) throw 2;

if (co == NULL) throw 2;

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

strcpy(name, n);

country = new char[strlen(co) + 1];

strcpy(country, co);

cost = c;

}

template <class T>

void Offurniture<T>::print()const {

cout << "Название предмета мебели: " << name << endl;

if (ID >= 0) cout << "ID предмета мебели:" << ID << endl;

cout << "Страна-производитель: " << country << endl;

cout << "Стоимость предмета мебели(в рублях) :" << cost << endl;

}

void Setofitems::print() {

Offurniture<double>::print();

cout << "Число предметов мебели в комплекте: " << amountofitems << endl;

if (partsale) cout << "Покупка отдельных элементов комплекта возможна" << endl;

else cout << "Покупка отдельных элементов комплекта невозможна" << endl;

}

Setofitems::Setofitems(char \*n, char \*co, double c, int aoi, bool ps):Offurniture<double>(n, co, c){

if (aoi > 50 || aoi <1) throw 1;

amountofitems = aoi;

partsale = ps;

}

Setofitems::Setofitems() : Offurniture<double>(){

amountofitems = 0;

partsale = 0;

}

Setofitems::Setofitems(const Setofitems& m) {

if (m.name == NULL || m.country == NULL) throw 2;

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

strcpy(name, m.name);

if (m.amountofitems > 50 || m.amountofitems <1) throw 1;

amountofitems = m.amountofitems;

partsale = m.partsale;

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

strcpy(country, m.country);

cost = m.cost;

}

Setofitems& Setofitems::copy(Setofitems& m) {

if (m.name == NULL || m.country == NULL) throw 2;

if (&m != this) {

delete[]name;

delete[]country;

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

strcpy(name, m.name);

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

strcpy(country, m.country);

cost = m.cost;

if (m.amountofitems > 50 || m.amountofitems <1) throw 1;

amountofitems = m.amountofitems;

partsale = m.partsale;

}

return(\*this);

}

void Setofitems::input(char\* n, char\* co, double c, int aoi, bool ps) {

if (n == NULL) throw 2;

if (co == NULL) throw 2;

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

strcpy(name, n);

if (aoi > 50 || aoi <1) throw 1;

amountofitems = aoi;

partsale = ps;

country = new char[strlen(co) + 1];

strcpy(country, co);

cost = c;

}

int main()

{

setlocale(LC\_ALL, "russian");

try

{

cout << "Объект set1 Производного класса Setofitems" << endl;

Setofitems set1("Комплект мебели NOVA S", "Россия", 100000, 7, 1);

set1.print();

cout << endl;

cout << "Объект set2 Производного класса Setofitems" << endl;

Setofitems set2("Комплект мебели Суперэконом", "Россия", 100, 70, 0);

set2.print();

cout << endl;

if (set1 > 100) cout << "set1 больше 100 по стоимости согласно оператору сравнения базового класса" << endl;

else cout << "set1 меньше 100 по стоимости согласно оператору сравнения базового класса" << endl;

cout << endl;

cout << "Объект set4 без имени и страны-производителя:" << endl;

Setofitems set4(NULL, NULL, 7, 7, 1);

set4.print();

}

catch (int i)

{

cout << "Ошибка - ";

switch (i)

{

case 1:

cout << "Число предметов мебели в комплекте не может превышать 50 или быть меньше 1" << endl;

break;

case 2:

cout << "У объекта отсутствует имя или страна-производитель" << endl;

break;

}

}

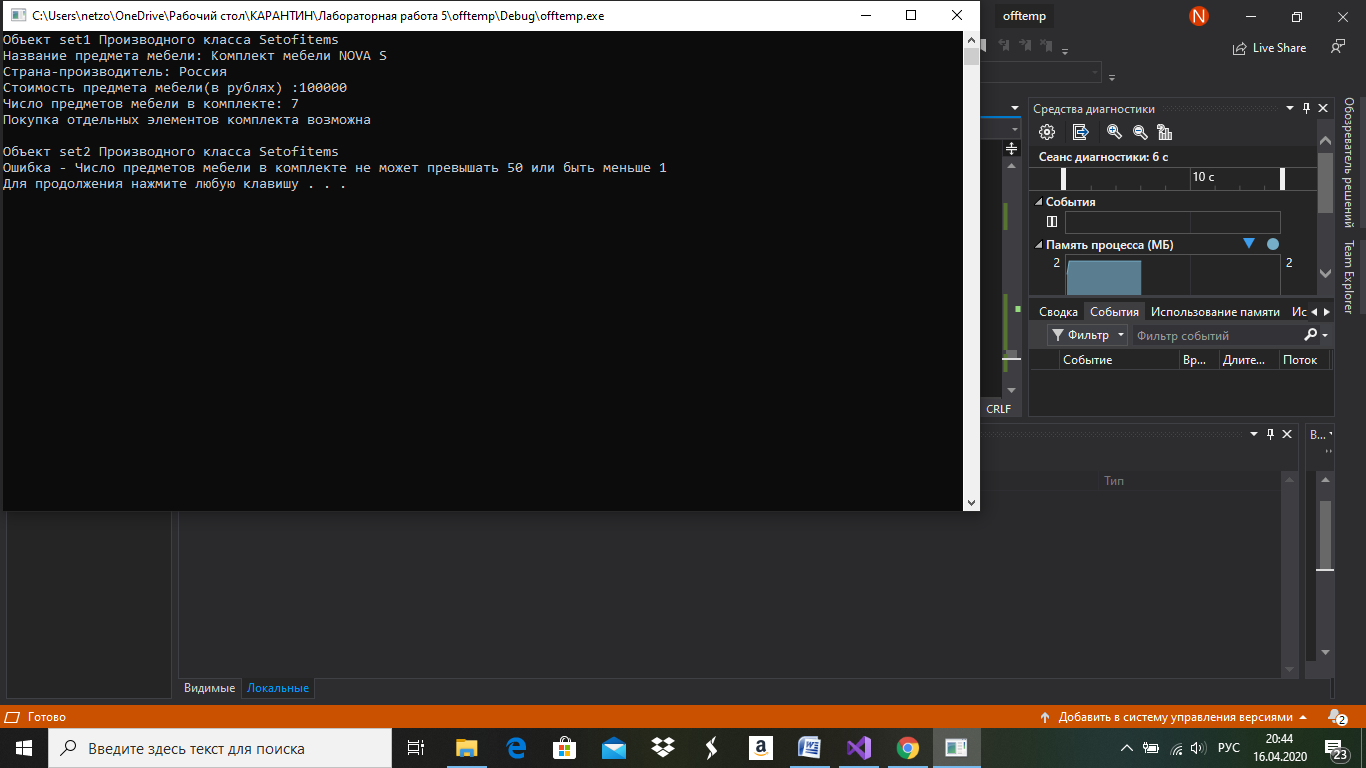
system("pause");

return 0;

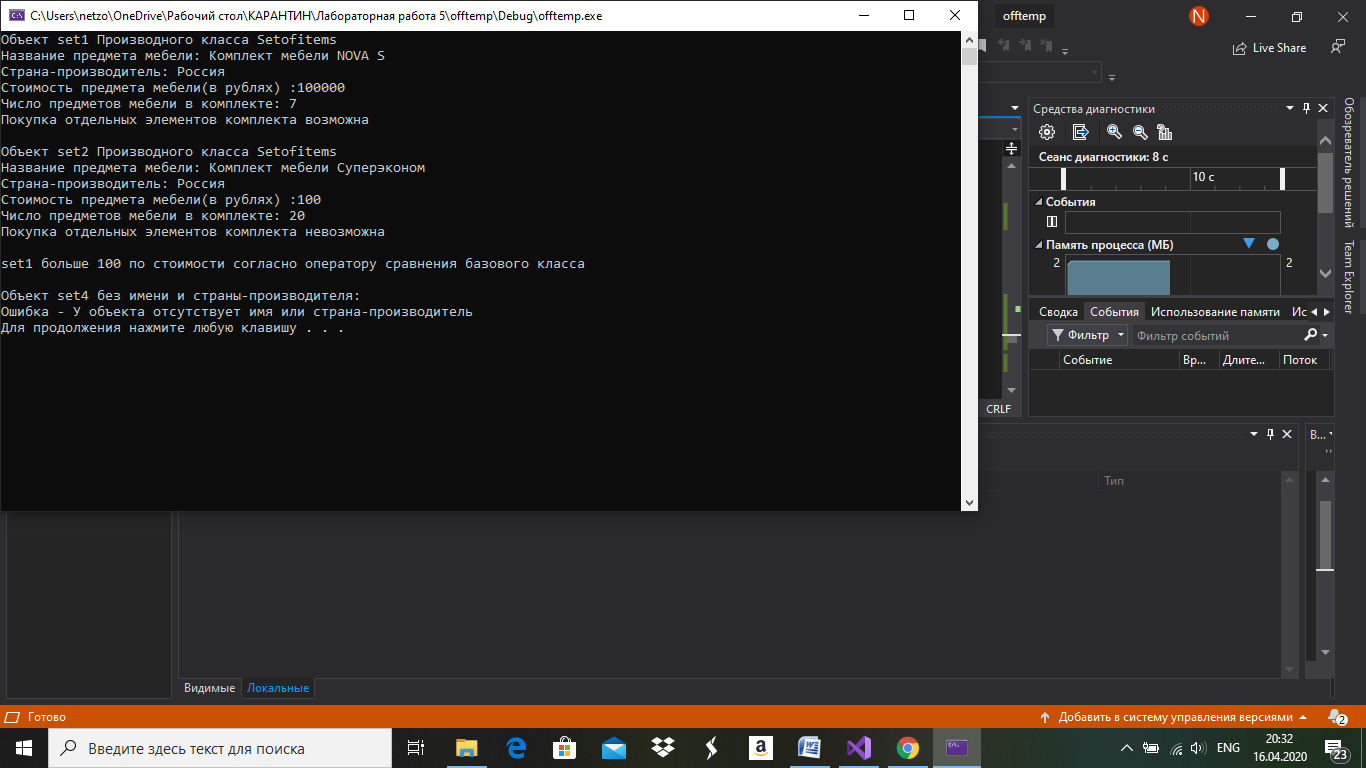
}

**Тестирование программы:**

Тест 1 – попытка присвоить объекту set2 количество предметов, равное 70:



Тест 2 – создание объекта set4 без имени и страны-производителя:



**Ответы на контрольные вопросы:**

1. Исключение – это возникающая в программе нештатная ситуация, с которой программа не может справиться.
2. Список обработчиков для возбужденного исключения обычно описывается сразу за соответствующим ему блоком try.
3. Синтаксически обработчик имеет вид: catch (формальный\_аргумент) { составная\_конструкция}
4. Спецификация исключения – это установление списка типов, которые может иметь выражение throw внутри функции. 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. Предоставляемая системой функция unexpected() вызывается, когда она возбудила исключение, которое отсутствует в ее списке спецификации исключений. Функция unexpected() перехватывает не перехваченные исключения и вызывает функцию terminate, что завершает программу, не позволяя ей работать дальше с необработанным исключением.