**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.

**Код программы:**

#include "pch.h"

#include <iostream>

#include <string.h>

using namespace std;

template <class T = int, class F = float> //шаблонный БК

class sendM

{

public:

sendM<T, F>(const char\* c, T b, F d, const char\* y);

sendM<T, F>(const sendM<T, F> &a);

sendM<T, F>();

~sendM<T, F>();

void print();

bool operator==(sendM<T, F> a);

sendM operator+(sendM<T, F> &a);

sendM &operator=(const sendM<T, F> &a);

protected:

char\* type;

T weight;

F price;

char\* marka;

};

class sendIN { // 2 БК

public:

sendIN(bool u);

sendIN();

sendIN(const sendIN &a);

~sendIN();

void print();

protected:

bool USB;

};

template <class T, class F> // реализация шаблонного БК (1 БК)

sendM<T, F>::sendM() {

type = new char[10];

strcpy(type, "0");

weight = 0;

price = 0;

marka = new char[10];

strcpy(marka, "0");

}

template <class T, class F>

sendM<T, F>::sendM(const sendM<T, F> &a) {

type = new char[strlen(a.type) + 1];

strcpy(type, a.type);

marka = new char[strlen(a.marka) + 1];

strcpy(marka, a.marka);

weight = a.weight;

price = a.price;

}

template <class T, class F>

sendM<T, F>::sendM(const char\* c, T b, F d, const char\* y)

{

if (c == "0")

throw "Ошибка: не введен тип модема.";

type = new char[strlen(c) + 1];

strcpy(type, c);

if (b < 0)

throw "Ошибка: вес не может иметь отрицательное значение или равняться 0.";

weight = b;

if (d <= 0)

throw "Ошибка: цена не может иметь отрицательное значение или равняться 0.";

price = d;

if (y == "0")

throw "Ошибка: не введено наименование марки.";

marka = new char[strlen(y) + 1];

strcpy(marka, y);

}

template <class T, class F>

sendM<T, F>::~sendM() {

type = NULL; marka = NULL;

}

template <class T, class F>

void sendM<T, F>::print()

{

cout << "Тип:" << type << endl;

cout << "Вес:" << weight << endl;

cout << "Стоимость:" << price << endl;

cout << "Марка:" << marka << endl;

}

template <class T, class F>

bool sendM<T, F>::operator==(sendM<T, F> a) {

return ((!strcmp(type, a.type)) &&

weight == a.weight && price == a.price &&

(!strcmp(type, a.type)));

}

template <class T, class F>

sendM<T, F> sendM<T, F>::operator+(sendM<T, F> &a) {

sendM s;

s.type = new char[strlen(a.type) + strlen(type) + 1];

strcpy(s.type, type);

strcat(s.type, a.type);

s.weight = weight + a.weight;

s.price = price + a.price;

s.marka = new char[strlen(a.marka) + strlen(marka) + 1];

strcpy(s.marka, marka);

strcat(s.marka, a.marka);

return s;

}

template <class T, class F>

sendM<T, F> &sendM<T, F>::operator=(const sendM<T, F> &a)

{

type = NULL;

type = a.type;

type = new char[strlen(a.type) + 1];

strcpy(type, a.type);

weight = a.weight;

price = a.price;

marka = NULL;

marka = a.marka;

marka = new char[strlen(a.marka) + 1];

strcpy(marka, a.marka);

return (\*this);

}

// реализация БК без шаблона (2 БК)

sendIN::sendIN() {

USB = 0;

}

sendIN::sendIN(bool u)

{

if (u < 0 || u>1)

throw "Ошибка: значение должно быть 0 или 1";

USB = u;

}

sendIN::~sendIN() {

}

void sendIN::print()

{

cout << "Вход USB:\nДа=1\nНет=0\nВпишите значение=" << USB << endl;

}

// ПК без множественного наследования

template <class T, class F>

class sendrate : public sendM<double, int > {

public:

sendrate<T, F>(const char\* c, T b, F d, const char\* y, int r);

sendrate<T, F>(const sendrate<T, F> &a);

sendrate<T, F>();

~sendrate<T, F>();

void print();

sendrate& operator = (const sendrate<T, F> &a);

protected:

int R;

};

template <class T, class F>

sendrate<T, F>::sendrate(const char\* c, T b, F d, const char\* y, int r) :sendM<T, F>(c, b, d, y)

{

if (r < 1 || r>5)

throw "Ошибка: неправильное значение рейтинга(правильное: 1-5)";

R = r;

}

template <class T, class F>

sendrate<T, F>::sendrate(const sendrate<T, F> &a) {

R = a.R;

this->type = new char[strlen(a.type) + 1];

strcpy(this->type, a.type);

this->marka = new char[strlen(a.marka) + 1];

strcpy(this->marka, a.marka);

this->weight = a.weight;

this->price = a.price;

}

template <class T, class F>

sendrate<T, F>& sendrate<T, F>::operator=(const sendrate &a) {

R = a.R;

this->price = a.price;

this->weight = a.weight;

this->type = new char[strlen(a.type) + 1];

strcpy(this->type, a.type);

this->marka = new char[strlen(a.marka) + 1];

strcpy(this->marka, a.marka);

return(\*this);

};

template <class T, class F>

sendrate<T, F>::sendrate() : sendM<T, F>() {

R = 0;

}

template <class T, class F>

void sendrate<T, F>::print()

{

sendM::print();

cout << "Рейтинг:" << R << endl;

}

template <class T, class F>

sendrate<T, F>::~sendrate() {

sendM<T, F>::~sendM();

}

template <class T, class F> // ПК с множ. наследованием

class sendPC :public sendIN, public sendM<T, F> {

public:

sendPC<T, F>(const char\* c, T b, F d, const char\* y, int r, bool u);

sendPC<T, F>();

sendPC<T, F>(const sendPC<T, F> &a);

void print();

~sendPC();

protected:

int R;

};

template <class T, class F>

sendPC<T, F>::sendPC() : sendIN(), sendM<T, F>()

{

R = 0;

USB = 0;

}

template <class T, class F>

sendPC<T, F>::sendPC(const char\* c, T b, F d, const char\* y, int r, bool u) :sendIN(u), sendM<T, F>(c, b, d, y) {

if (r < 1 || r>5)

throw "Ошибка: неправильное значение рейтинга(правильное: 1-5)";

R = r;

if (u < 0 || u>1)

throw "Ошибка: значение должно быть 0 или 1";

USB = u;

}

template <class T, class F>

sendPC<T, F>::sendPC(const sendPC<T, F> &a) {

R = a.R;

this->type = new char[strlen(a.type) + 1];

strcpy(this->type, a.type);

this->marka = new char[strlen(a.marka) + 1];

strcpy(this->marka, a.marka);

this->weight = a.weight;

this->price = a.price;

this->USB = a.USB;

}

template <class T, class F>

void sendPC<T, F>::print()

{

sendM<T, F>::print();

sendIN::print();

cout << "Рейтинг:" << R << endl;

}

template <class T, class F>

sendPC<T, F>::~sendPC() {

sendM<T, F>::~sendM();

sendIN::~sendIN();

}

int main()

{

setlocale(LC\_ALL, "Russian");

char\* type = new char[10];

char\* marka = new char[10];

int weight, R;

double price;

bool USB;

cout << "======================Проверка на наличие ошибок=======================\n";

try

{

sendM<double, int> e("0", 10, 100, "HTC");

}

catch (const char\* fail)

{

cout << fail << '\n';

}

try

{

sendrate<double, int> a("4G", 10, 1, "0", 5);

}

catch (const char\* fail)

{

cout << fail << '\n';

}

try

{

sendPC<double, int> c("3G", 100, 0, "Digma", 3, 1);

}

catch (const char\* fail)

{

cout << fail << '\n';

}

try

{

sendPC<double, int> c("3G", -10, 10, "Digma", 3, 1);

}

catch (const char\* fail)

{

cout << fail << '\n';

}

try

{

sendPC<double, int> c("3G", 100, 10, "Digma", 0, 1);

}

catch (const char\* fail)

{

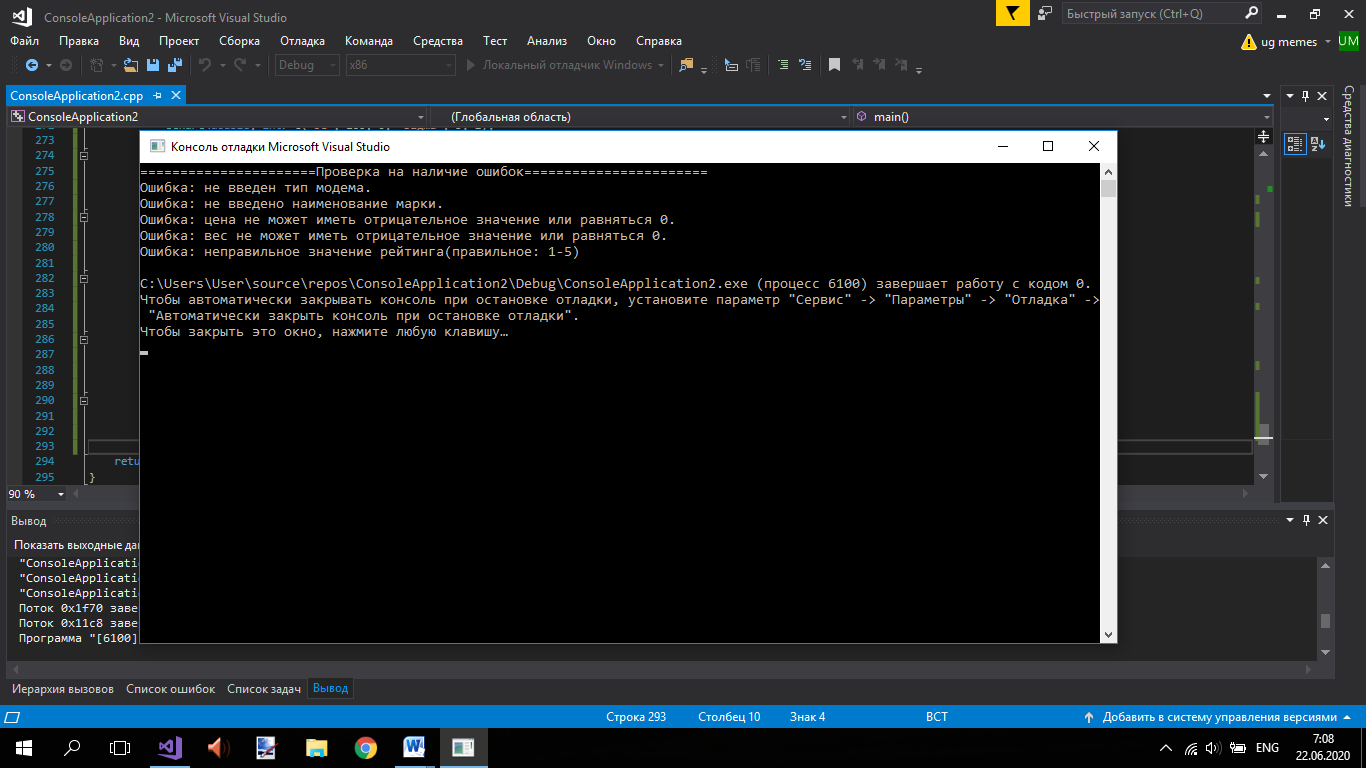
cout << fail << '\n';

}

return 0;

}

**Результат работы программы**



**Контрольные вопросы**

1. Дайте определение исключения.
2. В каком блоке описывается список обработчиков для возбужденного исключения?
3. Опишите синтаксис обработчика исключения.
4. Что такое спецификация исключения?
5. Для каких целей используется функция unexpected?

Ответы:

1) Обычно исключение – 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.

Код на С++ позволяет напрямую возбуждать исключения в пробном блоке с помощью запускающего выражения throw. Исключения обрабатываются с помощью вызова надлежащего обработчика, выбираемого из списка обработчиков, который следует сразу за пробным блоком.

2) Блок try является контекстом для определения того, какие обработчики вызываются при возбуждении исключения. Порядок, в котором определены обработчики, задает очередность проверки.

3) Синтаксически обработчик имеет вид:

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();

}

Допустима эллиптическая ( … ) сигнатура, совпадающая с аргументом любого типа. Кроме того, формальный аргумент может быть абстрактным объявлением, то есть может не задавать имя переменной, а только предоставлять информацию о типе.

4) 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. Например:

void foo() throw(int, over\_flow);

void noex(int i) throw();

Нарушение спецификаций исключений приводит к ошибкам на этапе выполнения. Эти ошибки отлавливаются функцией unexpected().

5) Предоставляемая системой функция unexpected() вызывается, когда она возбудила исключение, которое отсутствует в ее списке спецификации исключений. По умолчанию вызывается функция terminate(). Или же можно задать обработчик, воспользовавшись функцией set\_terminate().