***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 <string.h>

#include <iostream>

using namespace std;

template <class T1 = int, class T2 = float>

class MemoryDevice {

public:

MemoryDevice(const char\* programa, T1 v, T2 p);

MemoryDevice<T1, T2>();

void print();

MemoryDevice<T1, T2>(const MemoryDevice& a);

bool operator==(const MemoryDevice& a);

MemoryDevice<T1, T2> operator+(const MemoryDevice& a);

MemoryDevice<T1, T2>& operator++();

MemoryDevice<T1, T2>& operator=(const MemoryDevice& a);

~MemoryDevice<T1, T2>();

protected:

char\* programm; T1 cost; T2 size;

};

template <class T1, class T2>

MemoryDevice<T1, T2>::MemoryDevice<T1, T2>() {

programm = new char[strlen("EVA 255") + 1];

strcpy(programm, "EVA 255");

cost = 200;

size = 300;

};

template <class T1, class T2>

MemoryDevice<T1, T2>::MemoryDevice<T1, T2>(const char\* programa, T1 v, T2 p) {

if (programa == "")

throw "Название не может быть пустым"; //Ошибка

programm = new char[strlen(programa) + 1];

strcpy(programm, programa);

if (v < 0) throw "Стоимость не может быть отрицательной"; //Ошибка

cost = v;

if (p < 0) throw "Размер не может быть отрицательным"; //Ошибка

size = p;

};

template <class T1, class T2>

void MemoryDevice<T1, T2>::print() {

cout << "Название: " << programm << endl;

cout << "Стоимость: " << cost << endl;

cout << "Размер: " << size << endl;

};

template <class T1, class T2>

MemoryDevice<T1, T2>::MemoryDevice<T1, T2>(const MemoryDevice& a) {

if (a == NULL)

throw "Ссылка указывает на несуществующий элемент"; //Ошибка

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

strcpy(programm, a.programm);

cost = a.cost;

size = a.size;

};

template <class T1, class T2>

bool MemoryDevice<T1, T2>::operator==(const MemoryDevice& a) {

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

cost == a.cost &&

size == a.size);

}

template <class T1, class T2>

MemoryDevice<T1, T2>::~MemoryDevice<T1, T2>() {

delete[] programm;

};

template <class T1, class T2>

MemoryDevice<T1, T2> MemoryDevice<T1, T2>::operator+(const MemoryDevice & a) {

MemoryDevice <T1, T2> v;

v.programm = new char[strlen(a.programm) + strlen(programm) + 2];

strcpy(v.programm, programm);

strcat(v.programm, ",");

strcat(v.programm, a.programm);

v.cost = cost + a.cost;

v.size = size + a.size;

return v;

};

template <class T1, class T2>

MemoryDevice<T1, T2>& MemoryDevice<T1, T2>:: operator=(const MemoryDevice & a) {

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

strcpy(programm, a.programm);

cost = a.cost;

size = a.size;

return(\*this);

};

template <class T1, class T2>

MemoryDevice<T1, T2>& MemoryDevice<T1, T2>::operator++() {

cost += 1;

size += 1;

return(\*this);

};

//Второй базовый класс

template <class T1 = int, class T2 = float>

class ElectronicMedia {

public:

ElectronicMedia<T1, T2>(int type);

ElectronicMedia<T1, T2>();

void printEM();

protected:

int typeEM;

};

template <class T1, class T2>

ElectronicMedia<T1, T2>::ElectronicMedia<T1, T2>() {

typeEM = 1;

};

template <class T1, class T2>

ElectronicMedia<T1, T2>::ElectronicMedia<T1, T2>(int type) {

if (type > 3)

throw "Ошибка класс ПО не может быть больше 3"; //Ошибка

typeEM = type;

};

template <class T1, class T2>

void ElectronicMedia<T1, T2>::printEM() {

if (typeEM == 1)cout << "Класс запоминающего устройства-Оптическое" << endl;

if (typeEM == 2)cout << "Класс запоминающего устройства-Полупроводниковые" << endl;

if (typeEM == 3)cout << "Класс запоминающего устройства-Магнитные" << endl;

};

//Производный класс

template <class T1 = int, class T2 = float>

class korpys : public MemoryDevice<T1, T2> {

public:

korpys<T1, T2>();

korpys(T1 seria, bool сooling, const char\* inf, const char\* programa, T1 v, T2 p);

~korpys<T1, T2>();

void print1();

korpys<T1, T2>(const korpys& a);

korpys& operator = (const korpys<T1, T2>& a);

protected:

T1 seria; bool cooling; char\* inf;

};

template <class T1, class T2>

korpys<T1, T2>::korpys<T1, T2>() : MemoryDevice<T1, T2>() {

seria = 0;

cooling = false;

inf = new char[strlen("нету") + 1];

strcpy(inf, "нету");

};

template <class T1, class T2>

korpys<T1, T2>::korpys<T1, T2>(T1 seria, bool cooling, const char\* inf, const char\* programa, T1 v, T2 p) : MemoryDevice<T1, T2>(programa, v, p) {

if (seria < 0 || seria > 100)

throw "Ошибка - серия не может быть больше 100 и меньше 0"; //Ошибка

this->seria = seria;

this->cooling = cooling;

this->inf = new char[strlen(inf) + 1];

strcpy(this->inf, inf);

};

template <class T1, class T2>

void korpys<T1, T2>::print1() {

cout << "Вывод " << endl;

korpys<T1, T2>::print();

cout << endl << "Серия-" << seria << endl;

cout << "Охлаждение-";

if (cooling) cout << "Есть" << endl;

else cout << "Нет" << endl;

cout << "Информация - " << inf << endl;

};

template <class T1, class T2>

korpys<T1, T2>::korpys<T1, T2>(const korpys& a) {

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

strcpy(inf, a.inf);

seria = a.seria;

cooling = a.cooling;

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

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

this->cost = a.cost;

this->size = a.size;

};

template <class T1, class T2>

korpys<T1, T2>& korpys<T1, T2>::operator=(const korpys& a) {

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

strcpy(inf, a.inf);

seria = a.seria;

cooling = a.cooling;

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

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

this->cost = a.cost;

this->size = a.size;

return(\*this);

};

template <class T1, class T2>

korpys<T1, T2>::~korpys() {

delete[] inf;

};

// Второй производный класс с множественым наследованием

template <class T1 = int, class T2 = float>

class View : public MemoryDevice<T1, T2>, public ElectronicMedia<T1, T2>

{

public:

View<T1, T2>(const char\* programa, T1 v, T2 p, int type, char\* vie);

View<T1, T2>();

void printView();

protected:

char\* spec;

};

template <class T1, class T2>

View<T1, T2>::View<T1, T2>() : MemoryDevice<T1, T2>(), ElectronicMedia<T1, T2>() {

spec = new char[strlen("EVA 255") + 1];

strcpy(spec, "EVA 255");

};

template <class T1, class T2>

View<T1, T2>::View<T1, T2>(const char\* programa, T1 v, T2 p, int type, char\* vie) : MemoryDevice<T1, T2>(programa, v, p), ElectronicMedia<T1, T2>(type) {

spec = new char[strlen(vie) + 1];

strcpy(spec, vie);

};

template <class T1, class T2>

void View<T1, T2>::printView() {

cout << "Вывод " << endl;

MemoryDevice<T1, T2>::print();

ElectronicMedia<T1, T2>::printEM();

cout << "Данное запоминающее устройство принадлежит - " << spec << endl;

};

int main()

{

setlocale(LC\_ALL, "Russian");

char\* programa = new char[10];

char\* inf = new char[10];

double cost, size, seria;

bool cooling;

cout << endl << "Ошибки: " << endl;

try

{

MemoryDevice<float, int>z1("", 1, 2);

}

catch (const char\* e)

{

cout << e << endl;

}

try

{

MemoryDevice<float, int>z2("mc5", -1, 2);

}

catch (const char\* e)

{

cout << e << endl;

}

try

{

MemoryDevice<float, int>z3("mc5", 1, -2);

}

catch (const char\* e)

{

cout << e << endl;

}

try

{

korpys<float, int>z4(11, 5, "good", "mc6", 10000, 128);

}

catch (const char\* e)

{

cout << e << endl;

}

try

{

ElectronicMedia<float, int> z5(5);

}

catch (const char\* e)

{

cout << e << endl;

}

return 0;

}

**Контрольные вопросы:**

1. **Дайте определение исключения.**

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.

1. **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

1. **Опишите синтаксис обработчика исключения.**

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.

1. **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. 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().

1. **Для каких целей используется функция unexpected?**

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().

**Результат работы программы:**

