***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 <stdlib.h>

#include "stdio.h"

#include <iostream>

using namespace std;

// Базовый класс

template <class T1, class T2>

class Agr {

public:

Agr<T1, T2>();

Agr<T1, T2>(const char\* name, T1 n, T2 t);

void input();

void print();

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

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

~Agr<T1, T2>();

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

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

protected:

char\* tractor; T1 mass; T2 cost;

};

template <class T1, class T2>

Agr<T1, T2>::Agr()

{

tractor = new char[30];

strcpy(tractor, "Tractor");

mass = 200;

cost = 600;

}

template <class T1, class T2>

Agr<T1, T2>::Agr<T1, T2>(const char\* name, T1 n, T2 t) {

if (name == "")

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

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

strcpy(tractor, name);

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

cost = t;

if (n < 0) throw "Масса не может быть отрицательной"; //Ошибка

mass = n;

};

template <class T1, class T2>

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

cout << endl;

cout << "Трактор: " << tractor << endl;

cout << "Масса: " << mass << " кг" << endl;

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

};

template <class T1, class T2>

void Agr<T1, T2>::input()

{

cout << "Введите название трактора: ";

cin >> tractor;

cout << "Введите массу трактора (кг): ";

cin >> mass;

cout << "Введите стоимость трактора (руб): ";

cin >> cost;

}

template <class T1, class T2>

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

if (a == NULL)

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

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

strcpy(tractor, a.tractor);

cost = a.cost;

mass = a.mass;

};

template <class T1, class T2>

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

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

cost == a.cost &&

mass == a.mass);

};

template <class T1, class T2>

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

delete[]tractor;

};

template <class T1, class T2>

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

Agr v;

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

strcpy(v.tractor, tractor);

strcat(v.tractor, ",");

strcat(v.tractor, a.tractor);

v.cost = cost + a.cost;

v.mass = mass + a.mass;

return v;

};

template <class T1, class T2>

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

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

strcpy(tractor, a.tractor);

cost = a.cost;

mass = a.mass;

return(\*this);

};

//Второй базовый класс

template <class T1, class T2>

class ClassDvig {

public:

ClassDvig<T1, T2>(const char\* dvig);

ClassDvig<T1, T2>();

void inputdvig();

void printdvig();

protected:

char\* dvigP;

};

template <class T1, class T2>

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

dvigP = new char[30];

strcpy(dvigP, "Колесный");

};

template <class T1, class T2>

ClassDvig<T1, T2>::ClassDvig<T1, T2>(const char\* dvig) {

if (dvig == "")

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

dvigP = new char[strlen(dvig) + 1];

strcpy(dvigP, dvig);

};

template <class T1, class T2>

void ClassDvig<T1, T2>::inputdvig()

{

cout << "Введите тип движителя: ";

cin >> dvigP;

}

template <class T1, class T2>

void ClassDvig<T1, T2>::printdvig() {

cout << "Тип движителя: " << dvigP << endl;

};

// Производный класс

template <class T1, class T2>

class Engine : public Agr<T1, T2>, public ClassDvig<T1, T2> {

public:

Engine();

Engine(const char\* name, T1 n, T2 t, const int pow);

~Engine();

void input();

void print();

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

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

protected:

double pow;

};

template <class T1, class T2>

Engine<T1, T2>::Engine() : Agr<T1, T2>(), ClassDvig<T1, T2>()

{

pow = 240;

}

template <class T1, class T2>

Engine<T1, T2>::Engine(const char\* name, T1 n, T2 t, const int pow) : Agr<T1, T2>(name, n, t), ClassDvig<T1, T2>()

{

if (pow < 0) throw "Мощность не может быть отрицательной"; //Ошибка

this->pow = pow;

}

template <class T1, class T2>

Engine<T1, T2>::~Engine()

{

Agr<T1, T2>::~Agr();

}

template <class T1, class T2>

void Engine<T1, T2>::input()

{

Agr<T1, T2>::input();

ClassDvig<T1, T2>::inputdvig();

cout << "Введите мощность двигателя (л.с.): ";

cin >> pow;

}

template <class T1, class T2>

void Engine<T1, T2>::print()

{

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

cout << endl;

ClassDvig<T1, T2>::printdvig();

cout << "Мощность двигателя: " << pow << " л.с." << endl;

}

template <class T1, class T2>

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

if (a == NULL)

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

pow = a.pow;

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

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

this->cost = a.cost;

this->mass = a.mass;

};

template <class T1, class T2>

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

pow = a.pow;

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

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

this->cost = a.cost;

this->mass = a.mass;

return(\*this);

};

int main()

{

setlocale(LC\_ALL, "Russian");

cout << endl << "Ошибки: " << endl;

//пустое название

try

{

Agr<int, double>a1("", 1, 1);

}

catch (const char\* e)

{

cout << e << endl;

}

//отрицательная стоимость

try

{

Agr<float, int>a2("LZ", -1, 1);

}

catch (const char\* e)

{

cout << e << endl;

}

//отрицательная масса

try

{

Agr<float, int>a3("LZ", 1, -1);

}

catch (const char\* e)

{

cout << e << endl;

}

//пустое название

try

{

ClassDvig<float, int>c1("");

}

catch (const char\* e)

{

cout << e << endl;

}

//отрицательная мощность

try

{

Engine<float, int>e1("LZ",1,1,-2);

}

catch (const char\* e)

{

cout << e << endl;

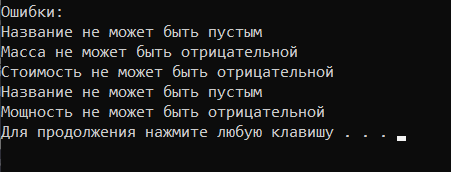
}

system("pause");

return 0;

}

**Результат работы программы:**



**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().