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**

#include <iostream>

#include <stdio.h>

#include <string.h>

using namespace std;

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

class Software {

public:

Software<T1, T2>(const char\* soft, T1 c, T2 s);

Software<T1, T2>();

void print();

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

bool operator==(const Software& a);

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

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

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

~Software();

protected:

char\* programm; T1 cost; T2 size;

};

template <class T1, class T2>

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

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

strcpy(programm, "Неизвестно");

cost = 0;

size = 0;

};

template <class T1, class T2>

Software<T1, T2>::Software<T1, T2>(const char\* soft, T1 c, T2 s) {

if (soft=="")

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

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

strcpy(programm, soft);

if (c < 0)

throw "Стоимость не может быть отрицательной"; //Ошибка

cost = c;

if (s < 0)

throw "Размер не может быть отрицательным"; //Ошибка

size = s;

};

template <class T1, class T2>

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

cout << "Программа-" << programm << endl;

cout << "Стоимость-" << cost << endl;

cout << "Размер- " << size << "Мб" << endl << endl;

};

template <class T1, class T2>

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

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

strcpy(programm, a.programm);

cost = a.cost;

size = a.size;

};

template <class T1, class T2>

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

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

cost == a.cost &&

size == a.size);

};

template <class T1, class T2>

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

delete[] programm;

};

template <class T1, class T2>

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

Software<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>

Software<T1, T2>& Software<T1, T2>::operator=(const Software & 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>

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

cost += 1;

size += 1;

return(\*this);

};

//Второй базовый класс

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

class BaseСlassSoftware {

public:

BaseСlassSoftware<T1, T2>(int type);

BaseСlassSoftware<T1, T2>();

void printClass();

protected:

int typeClass;

};

template <class T1, class T2>

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

typeClass = 1;

};

template <class T1, class T2>

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

if (type > 3)

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

typeClass = type;

};

template <class T1, class T2>

void BaseСlassSoftware<T1, T2>::printClass() {

if (typeClass == 1)cout << "Класс программного обеспечения-Системное ПО" << endl;

if (typeClass == 2)cout << "Класс программного обеспечения-Прикладное ПО" << endl;

if (typeClass == 3)cout << "Класс программного обеспечения-Системы программирования" << endl;

};

//Производный класс

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

class PKSoftware : public Software<T1, T2> {

public:

PKSoftware<T1, T2>();

PKSoftware<T1, T2>(T1 stars, bool license, const char\* comment, const char\* soft, T1 c, T2 s);

~PKSoftware<T1, T2>();

void conclusion();

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

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

protected:

T1 stars; bool license; char\* comment;

};

template <class T1, class T2>

PKSoftware<T1, T2>::PKSoftware<T1, T2>() : Software<T1, T2>() {

stars = 0;

license = false;

comment = new char[strlen("не добавлено") + 1];

strcpy(comment, "не добавлено");

};

template <class T1, class T2>

PKSoftware<T1, T2>::PKSoftware<T1, T2>(T1 stars, bool license, const char\* comment, const char\* soft, T1 c, T2 s) : Software<T1, T2>(soft, c, s) {

if (stars < 0 || stars > 10)

throw "Ошибка - рейтинг не может быть больше 10 и меньше 0"; //Ошибка

this->stars = stars;

this->license = license;

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

strcpy(this->comment, comment);

};

template <class T1, class T2>

void PKSoftware<T1, T2>::conclusion() {

cout << "Вывод " << endl;

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

cout << endl << "Рейтинг-" << stars << endl;

cout << "Лицензия-";

if (license) cout << "Есть" << endl;

else cout << "Нет" << endl;

cout << "Комментарий- " << comment << endl;

};

template <class T1, class T2>

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

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

strcpy(comment, a.comment);

stars = a.stars;

license = a.license;

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>

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

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

strcpy(comment, a.comment);

stars = a.stars;

license = a.license;

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>

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

delete[] comment;

};

//Второй производный класс с множественым наследованием

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

class Apply : public Software<T1, T2>, public BaseСlassSoftware<T1, T2>

{

public:

Apply<T1, T2>(const char\* soft, T1 c, T2 s, int type, char\* spec);

Apply<T1, T2>();

void printApply();

protected:

char\* specialization;

};

template <class T1, class T2>

Apply<T1, T2>::Apply<T1, T2>() : Software<T1, T2>(), BaseСlassSoftware<T1, T2>() {

specialization = new char[strlen("неизвестно") + 1];

strcpy(specialization, "неизвестно");

};

template <class T1, class T2>

Apply<T1, T2>::Apply<T1, T2>(const char\* soft, T1 c, T2 s, int type, char\* spec) : Software<T1, T2>(soft, c, s), BaseСlassSoftware<T1, T2>(type) {

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

strcpy(specialization, spec);

};

template <class T1, class T2>

void Apply<T1, T2>::printApply() {

cout << "Вывод " << endl;

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

BaseСlassSoftware<T1, T2>::printClass();

cout << "Данное ПО специализируется на - " << specialization << endl;

};

int main()

{

setlocale(LC\_ALL, "Russian");

cout << endl << "Ошибки: " << endl;

try

{

Software<float, int>x1("", 1, 2);

}

catch (const char\* e)

{

cout <<"1 " << e << endl;

}

try

{

Software<float, int>x2("mc5", -1, 2);

}

catch (const char\* e)

{

cout << "2 " << e << endl;

}

try

{

Software<float, int>x3("mc5", 1, -2);

}

catch (const char\* e)

{

cout << "3 " << e << endl;

}

try

{

PKSoftware<float,int> x4(11, 5, "good", "mc6", 10000, 128);

}

catch (const char\* e)

{

cout << "4 " << e << endl;

}

try

{

BaseСlassSoftware<float, int> x5(5);

}

catch (const char\* e)

{

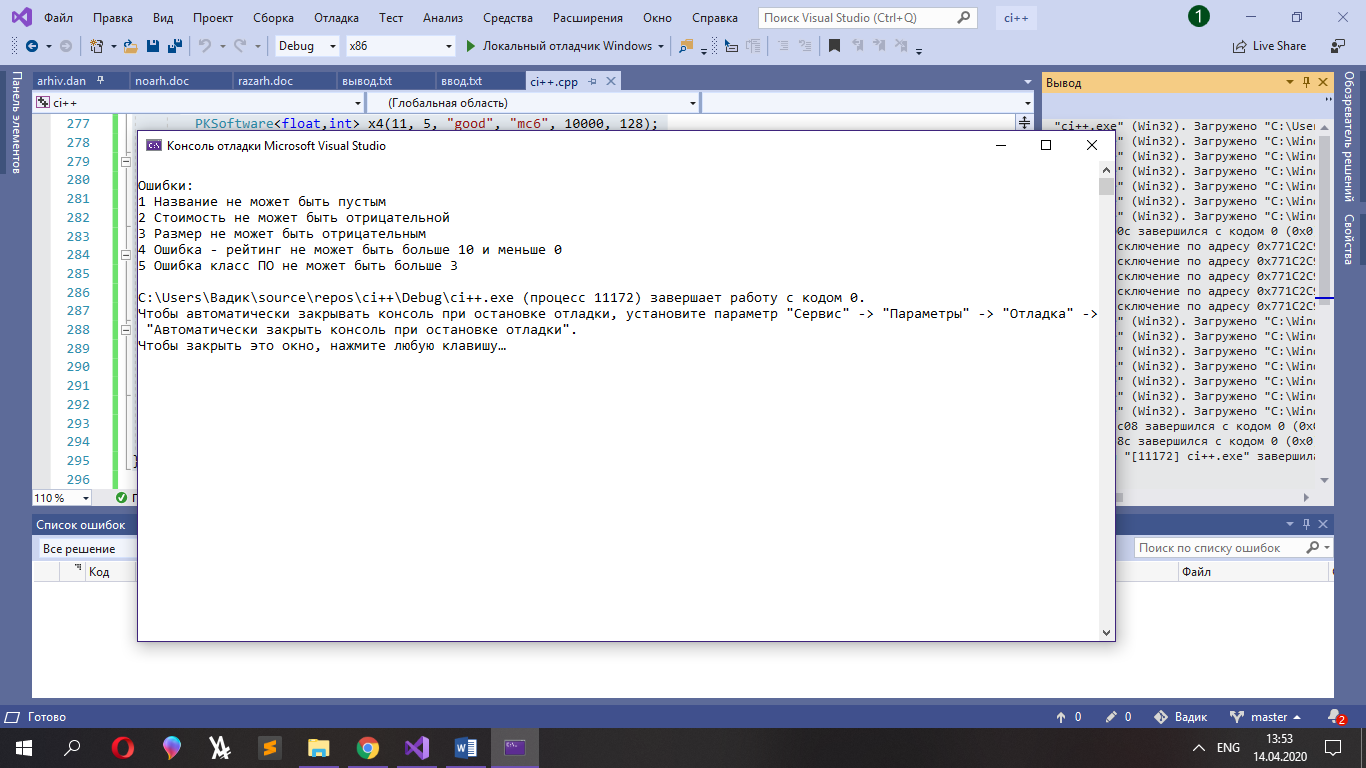
cout << "5 " << e << endl;

}

return 0;

}

**Пример работы**



**Контрольные** **вопросы**

1. Дайте определение исключения.

Исключение – это нештатная ситуация, с которой программа не может справиться. Например, при делении на ноль выполнение программы аварийно завершается системой.

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

В блоке try описывается, какие обработчики вызываются при возбуждении исключения. Порядок, в котором определены обработчики, задает очередность проверки обработчиков подходящего типа для возбуждения исключения.

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.

1. Что такое спецификация исключения?

Cпецификация исключения является частью объявления и определения функции.

1. Для каких целей используется функция unexpected?

Функция unexpected вызывается, когда вызвано исключение, которое отсутствует в списке спецификации исключений. Нарушение спецификаций исключений приводит к ошибкам на этапе выполнения. Эти ошибки отлавливаются функцией unexpected.