МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Череповецкий государственный университет»

**Лабораторная работа № 3**

**«ПРОГРАММИРОВАНИЕ ОБРАБОТКИ**

**ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ»**

**Выполнил:**

студент гр. 1ИВТпб-01-21оп

Климов А.Г.  
**Проверил:** преподаватель

Ганичева О.Г.  
Отметка о зачете:

Череповец

2017 год

**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.

**Вариант**

A 16. Осветительные приборы.

**Задания**

Используя модифицированный АТД, обработайте все возможные исключительные ситуации.

**Ход работы**

**Текст программы:**

**ClassLight.h**

// ClassLight.h

// Блок защиты от повторного включения:

#ifndef LIGHT\_H\_INCLUDED

#define LIGHT\_H\_INCLUDED

#include <assert.h>

template <class T> class light

{

public:// набор функций

void add\_cost\_light(float x) {

assert(x > 0); // Если значение цены меньше 0, то программа прекращает работу

cost\_light = x;

}

float get\_cost\_light() {

return cost\_light;

}

light& operator=(const light &m);

friend light operator-(light a, light b);

light operator++();

//

light(); //конструктор по умолчанию

light(char \*nm1, char nm2, T i, float j); //конструктор с параметрами

light(const light &t); //конструктор с const, меняющий содержимое, к которому получает доступ (копирующий конструктор)

~light() { delete[]name1; } //деструктор

//

void input(char \*nm1, char nm2, int i, float j);

void print();

void prisv(light b);

bool sravn(light b);

void destr() { delete[]name1; } // уничтожение объекто

private:

char \*name1; // название осветительного прибора

char name2; // тип лампочки

T k; // размер цоколя

float d; // яркость света

protected:

float cost\_light; // цена осветительного прибора

};

#endif //LIGHT\_H\_INCLUDED

**ClassLight.cpp**

#include "ClassLight.h"

#include <iostream>

#include <string>

template<typename T> void light<T>::input(char \*nm1, char nm2, int i, float j)

{

name1 = new char[strlen(nm1) + 1];

strcpy(name1, nm1);

name2 = nm2;

k = i;

d = j;

}

template<typename T> void light<T>::print()

{

std::cout << "Название: " << name1 << std::endl;

std::cout << "Цоколь: " << name2 << k << std::endl;

std::cout << "Яркость света: " << d << std::endl;

}

template<typename T> void light<T>::prisv(light b)

{

delete[]name1;

name1 = new char[strlen(b.name1) + 1];

strcpy(name1, b.name1);

name2 = b.name2;

k = b.k;

d = b.d;

}

template<typename T> bool light<T>::sravn(light b)

{

return ((strcmp(name1, b.name1) == 0) && (name2 == b.name2) && (k == b.k) && (d == b.d)) ? true : false;

}

//конструкторы

template<typename T> light<T>::light() {

name1 = new char[strlen("Unknown") + 1];

strcpy(name1, "Unknown");

name2 = 'E';

k = 0;

d = 0;

cost\_light = 555;

}

template<typename T> light<T>::light(char \*nm1, char nm2, T i, float j) {

name1 = new char[strlen(nm1) + 1];

strcpy(name1, nm1);

name2 = nm2;

k = i;

d = j;

}

template<typename T> light<T>::light(const light &t) {

name1 = new char[strlen(t.name1) + 1];

strcpy(name1, t.name1);

name2 = t.name2;

k = t.k;

d = t.d;

}

//перегрузка операторов

template<typename T> light<T> light<T>::operator++() {

k++;

d++;

return (\*this);

}

template<typename T> light<T> operator-(light<T> a, light<T> b) {

light<T> h;

h.k = a.k - b.k;

h.d = a.d - b.d;

return(h);

}

template<typename T> light<T>& light<T>::operator=(const light &m) {

k = m.k;

d = m.d;

delete[]name1;

name1 = new char[strlen(m.name1) + 1];

strcpy(name1, m.name1);

return(\*this);

}

template class light<int>;

template class light<float>;

template class light<char>;

**ClassLamps.h**

// ClassLamps.h

// Блок защиты от повторного включения:

#ifndef LAMPS\_H\_INCLUDED

#define LAMPS\_H\_INCLUDED

#include "ClassLight.h"

#include <iostream>

#include <assert.h>

#include <signal.h>

class lamps :public light<int> {

public:

void add\_cost\_lamp(float y) {

if (y < 0) throw (y); // Выражение исключение

cost\_lamp = y;

}

float get\_cost\_lamp() {

return cost\_lamp;

}

lamps() {

name3 = new char[strlen("Lamp name") + 1];

strcpy(name3, "Lamp name");

j = 0;

cost\_lamp = 18;

};

lamps(char \*name1, char name2, int k, float d, char \*nm3, float j) :light(name1, name2, k, d) {

name3 = new char[strlen(nm3) + 1];

strcpy(name3, nm3);

this->j = j;

}

void print\_m();

~lamps() { delete[]name3; } //деструктор

private:

char\* name3;

float j;

int \*p;

int size;

protected:

float cost\_lamp=0; // цена лампы прибора

float cost\_lamp\_2=20;

};

#endif //MY\_SYMBOL\_H

**ClassLamps.cpp**

#include "ClassLamps.h"

#include <iostream>

void lamps::print\_m() {

light::print();

std::cout << name3<< " " << j;

std::cout << std::endl;

}

**ClassTest.h**

#include "ClassLight.h"

#include "ClassLamps.h"

#include <iostream>

class light\_lamps :public light<int>, public lamps {

public:

light\_lamps() {

std::cout << "Инициализация light\_lamps\n"; }

void light\_lamps\_addname(char\* nm4)

{

name4 = new char[strlen(nm4) + 1];

strcpy(name4, nm4);

}

void print\_ss() {

std::cout << "Название light\_lamps: " << name4 << std::endl;

};

int cost\_light\_and\_lamp() { return light<int>::cost\_light+lamps::cost\_lamp; }

private:

char \*name4;

};

**Main.cpp**

#include <iostream>

#include "ClassLight.h"

#include "ClassLamps.h"

#include "ClassTest.h"

using namespace std;

void main()

{

setlocale(LC\_ALL, "Russian");

light<int> s1("Светильник1", 'E', 77, 10.8);

s1.print();

cout << endl;

s1.add\_cost\_light(455.88);

//s1.add\_cost\_light(-455.88);

cout << "Цена осветительного прибора1: " << s1.get\_cost\_light() << endl;

cout << endl;

lamps l1("Светильник1", 'E', 40, 10.8, "Лампа", 22);

lamps l2("Светильник2", 'E', 40, 10.8, "Лампа", 22);

l1.print\_m();

cout << endl;

l2.print\_m();

cout << endl;

cout << "Цена лампы 1: " << l1.get\_cost\_lamp() << endl;

cout << "Цена лампы 2: " << l2.get\_cost\_lamp() << endl;

try // Обработчик исключений для цены. Если цена меньше нуля, то выводится сообщение об ошибке

{

float cost1 = 60.34, cost2=-60.34;

l1.add\_cost\_lamp(cost1);

l2.add\_cost\_lamp(cost2);

}

catch (float cost) { cout << "/// Ошибка при добавлении цены лампы: "<<cost<< " Значение цены сброшено!!! ///" << endl; float cost1 = 0, cost2 = 0; }

cout <<"Цена лампы 1: "<< l1.get\_cost\_lamp()<<endl;

cout << "Цена лампы 2: " << l2.get\_cost\_lamp()<<endl;

system("pause");

}

**Результаты тестирования:**

*Входные данные 1 (добавляем цену осветительного прибора):*

s1.add\_cost\_light(455.88);

*Результат 1 Рис.1:*

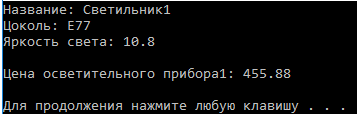


Рис. 1. Результат 1

*Входные данные 2 (добавляем цену осветительного прибора)::*

s1.add\_cost\_light(-455.88);

*Результат 2 Рис.2:*

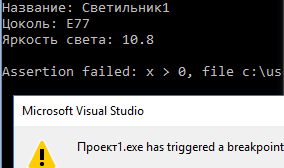


Рис. 2. Результат 2

*Входные данные 3:*

float cost1 = 60.34, cost2=-60.34;

l1.add\_cost\_lamp(cost1);

l2.add\_cost\_lamp(cost2);

*Результат 3 Рис.3:*

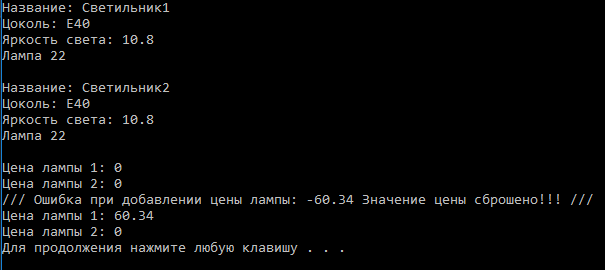


Рис. 3. Результат 3

**Ответы на 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.

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

Синтаксически выражение throw может принимать две фор-мы:

throw и throw выражение.

Конструкция «throw выражение» возбуждает исключение в пробном блоке. Для выбора инструкции catch, которая обрабатывает исключение, используется самый внутренний пробный блок (try-блок), в котором возбуждено исключение. А throw без аргумента может использоваться внутри catch для перезапуска текущего исключения или для вызова из недр первого обработчика второго обработчика при дальнейшей обработке исключения.

Запущенное с помощью throw выражение - это временный статический объект, существующий до тех пор, пока не будет произведен выход из обработчика исключения. Выражение отлавливается обработчиком, который может использовать этот временный объект. Неотловленное выражение завершает программу.

Целое значение, запущенное throw i, существует, пока не произойдет выход из обработчика catch(int n) с целой сигнатурой. Это выражение доступно для использования внутри обработчика как его аргумент.

Перезапуск исключений осуществляется таким образом:

catch (int n)

{

……….

throw; //перезапуск

}

Если запущенное выражение целого типа, то перезапущенное исключение – это тот же целый объект, который обрабатывается ближайшим обработчиком, подходящим для этого типа.

3. 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 информа-цию о типе.

4. 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().

5. Для каких целей используется функция unexpected?

Предоставляемая системой функция terminate() вызывается, если не было задано ни одного обработчика, знающего, как обращаться с исключением. По умолчанию вызывается функция abort(), которая немедленно завершает программу и возвращает управление операционной системе. С помощью set\_terminate() может быть задано какое-нибудь другое действие; тем самым задается обработчик. Объявления упомянутых функций находятся в except и except.h.

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().

**Вывод:** изучил синтаксис и семантику определения и вызова исключений, синтаксис обработчика и спецификации исключений; приобрёл практические навыки запуска исключений; изучил особенности применения стандартных библиотечных исключений.