Кафедра: Математическое и программное обеспечение ЭВМ

Дисциплина: ООП

ЛАБОРАТОРНАЯ РАБОТА № 6

«Исключительные ситуации»

Выполнил:

студент гр. 1ПИб-01-21оп

Коровин Даниил Андреевич

Проверил:

Доктор техничеких наук, Ершов Е.В.

Череповец, 2020 г

# Задание

Используя модифицированный АТД, обработайте все возможные исключительные ситуации.

# 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?

Код программы:

#include <iostream>

using namespace std;

// Первый базовый класс

template <class T1, class T2>

class Oven {

protected:

T1 year, month, day;

char\* name;

public:

void input();

void print();

Oven();

Oven(const char\* n, T2 date\_year, T1 date\_month, T1 date\_day);

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

~Oven<T1, T2>();

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

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

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

};

template <class T1, class T2>

void Oven<T1, T2>::input() {

cout << "Введите название вашей печи: ";

cin >> name;

cout << "Введите год изобретения вашей печи: ";

cin >> year;

cout << "Введите месяц изобретения вашей печи: ";

cin >> month;

cout << "Введите день изобретения вашей печи: ";

cin >> day;

}

template <class T1, class T2>

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

cout << "Название вашей печи - " << name << endl;

cout << "Год изобретения вашей печи - " << year << endl;

cout << "Месяц изобретения вашей печи - " << month << endl;

cout << "День изобретения вашей печи - " << day << endl;

cout << endl;

}

template <class T1, class T2>

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

name = new char[11]{ "Неизвестно" };

year = (T2)0;

month = (T1)0;

day = (T1)0;

}

template <class T1, class T2>

Oven<T1, T2>::Oven(const char\* n, T2 date\_year, T1 date\_month, T1 date\_day) {

if (n == "")

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

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

strcpy(name, n);

if (date\_year < 0)

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

year = date\_year;

if (date\_month < 0)

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

month = date\_month;

if (date\_day < 0)

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

day = date\_day;

}

template <class T1, class T2>

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

if (a == NULL)

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

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

strcpy(name, a.name);

year = a.year;

month = a.month;

day = a.day;

}

template <class T1, class T2>

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

delete[]name;

}

template <class T1, class T2>

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

if (this == &a)

return \*this;

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

strcpy(name, a.name);

year = a.year;

month = a.month;

day = a.day;

return \*this;

}

template<class T1, class T2>

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

{

Oven<T1, T2> t;

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

strcpy(t.name, name);

for (int i = strlen(name); i < strlen(name) + strlen(a.name); i++) {

t.name[i] = a.name[i - strlen(name)];

}

t.name[strlen(name) + strlen(a.name)] = '\0';

t.year = year + a.year;

t.month = month + a.month;

t.day = day + a.day;

return (t);

}

template<class T1, class T2>

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

return (year + month + day == a.year + a.month + a.day);

}

// Второй базовый класс

template <class T1, class T2>

class TypeOven {

protected:

char\* type;

public:

TypeOven<T1, T2>();

TypeOven<T1, T2>(const char\* typed);

void inputType();

void printType();

};

template<class T1, class T2>

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

type = new char[11]{ "Неизвестно" };

}

template<class T1, class T2>

TypeOven<T1, T2>::TypeOven(const char\* typed) {

if (typed == "")

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

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

strcpy(type, typed);

}

template<class T1, class T2>

void TypeOven<T1, T2>::inputType() {

cout << "Введите тип вашей печи:";

cin >> type;

}

template<class T1, class T2>

void TypeOven<T1, T2>::printType() {

cout << "Тип вашей печи: " << type << endl;

}

// Класс-наследник

template<class T1, class T2>

class Material : public Oven<T1, T2>, public TypeOven<T1, T2> {

public:

void input();

void print();

Material();

Material(const char\* n, T2 date\_year, T1 date\_month, T1 date\_day, const char\* material);

~Material();

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

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

protected:

char\* material;

};

template<class T1, class T2>

void Material<T1, T2>::input() {

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

TypeOven<T1, T2>::inputType();

cout << "Введите название материала вашей печи : ";

cin >> material;

}

template<class T1, class T2>

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

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

cout << endl;

TypeOven<T1, T2>::printType();

cout << "Материал вашей печи - " << material << endl;

}

template<class T1, class T2>

Material<T1, T2>::Material() : Oven<T1, T2>() {

material = new char[11]{ "Неизвестно" };

}

template<class T1, class T2>

Material<T1, T2>::Material(const char\* n, T2 date\_year, T1 date\_month, T1 date\_day, const char\* material) {

if (material = "")

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

this->material = material;

}

template<class T1, class T2>

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

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

}

template<class T1, class T2>

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

if (a == NULL)

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

material = a.material;

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

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

this->year = a.year;

this->month = a.month;

this->day = a.day;

}

template<class T1, class T2>

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

material = a.material;

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

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

this->year = a.year;

this->month = a.month;

this->day = a.day;

return (\*this);

}

int main()

{

setlocale(LC\_ALL, "Russian");

cout << "Ошибки в программе:" << endl;

// Пустое название печи

try {

Oven<int, double> a("", 1725, 10, 15);

}

catch (const char\* e) {

cout << e << endl;

}

// Отрицательная дата года

try {

Oven<int, double> b("Russian", -1111, 10, 15);

}

catch (const char\* e) {

cout << e << endl;

}

// Отрицательная дата месяца

try {

Oven<int, double> c("Russian", 1111, -10, 15);

}

catch (const char\* e) {

cout << e << endl;

}

// Отрицательная дата дня

try {

Oven<int, double> d("Russian", 1111, 10, -15);

}

catch (const char\* e) {

cout << e << endl;

}

// Пустое название типа печи

try {

TypeOven<int, double> f("");

}

catch (const char\* e) {

cout << e << endl;

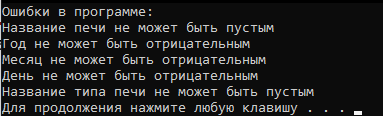
}

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().