Кафедра: Математическое и программное обеспечение ЭВМ

Дисциплина: ООП

ЛАБОРАТОРНАЯ РАБОТА № 6

«Исключительные ситуации»

Выполнил:

студент гр. 1ПИб-01-21оп

Балакирев Яков Александрович

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 DB1, class DB2>

class RGD {

protected:

DB1 spead, god, width;

char\* poezd;

public:

void vvod();

void vivod();

RGD();

RGD(const char\* n\_poezd, DB1 n\_spead, DB1 n\_god, DB1 n\_width);

RGD(const RGD<DB1, DB2>& temp);

~RGD<DB1, DB2>();

RGD<DB1, DB2> operator = (const RGD<DB1, DB2>& rgd);

bool operator == (const RGD<DB1, DB2>& rgd);

};

template <class DB1, class DB2>

void RGD<DB1, DB2>::vvod()

{

cout << "Введите название: ";

cin >> poezd;

cout << "Введите скорость: ";

cin >> spead;

cout << "Введите год: ";

cin >> god;

cout << "Введите ширину: ";

cin >> width;

}

template <class DB1, class DB2>

void RGD<DB1, DB2>::vivod()

{

cout << poezd << endl;

cout << "Скорость:" << spead << endl;

cout << "Год:" << god << endl;

cout << "Ширина:" << width << endl;

cout << endl;

}

template <class DB1, class DB2>

RGD<DB1, DB2>::RGD()

{

poezd = new char[11]{ "swallow" };

spead = (DB2)140;

god = (DB1)2011;

width = (DB1)3480;

}

template <class DB1, class DB2>

RGD<DB1, DB2>::RGD(const char\* n\_poezd, DB1 n\_spead, DB1 n\_god, DB1 n\_width)

{

if (n\_poezd == "")

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

poezd = new char[strlen(n\_poezd) + 1];

strcpy(poezd, n\_poezd);

if (n\_spead < 0)

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

spead = n\_spead;

if (n\_god < 0)

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

god = n\_god;

if (n\_width < 0)

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

width = n\_width;

}

template <class DB1, class DB2>

RGD<DB1, DB2>::RGD(const RGD<DB1, DB2>& temp)

{

if (temp == NULL)

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

poezd = new char[strlen(temp.poezd) + 1];

strcpy(poezd, temp.poezd);

spead = temp.spead;

god = temp.god;

width = temp.width;

}

template <class DB1, class DB2>

RGD<DB1, DB2>::~RGD()

{

delete[]poezd;

}

template <class DB1, class DB2>

RGD<DB1, DB2> RGD<DB1, DB2> :: operator = (const RGD& rgd)

{

if (this == &rgd)

return \*this;

poezd = new char[strlen(rgd.poezd) + 1];

strcpy(poezd, rgd.poezd);

spead = rgd.spead;

god = rgd.god;

width = rgd.width;

return \*this;

}

template<class DB1, class DB2>

bool RGD<DB1, DB2> :: operator == (const RGD<DB1, DB2>& rgd)

{

return (spead + god + width == rgd.spead + rgd.god + rgd.width);

}

// Второй базовый класс

template <class DB1, class DB2>

class Material

{

protected:

char\* material;

public:

Material<DB1, DB2>();

Material<DB1, DB2>(const char\* typed);

void vvod\_M();

void vivod\_M();

};

template<class DB1, class DB2>

Material<DB1, DB2>::Material()

{

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

}

template<class DB1, class DB2>

Material<DB1, DB2>::Material(const char\* temp)

{

if (temp == "")

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

material = new char[strlen(temp) + 1];

strcpy(material, temp);

}

template<class DB1, class DB2>

void Material<DB1, DB2>::vvod\_M()

{

cout << "Введите материал вагона:";

cin >> material;

}

template<class DB1, class DB2>

void Material<DB1, DB2>::vivod\_M()

{

cout << "Материал: " << material << endl;

}

// Класс-наследник

template<class DB1, class DB2>

class Service : public RGD<DB1, DB2>, public Material<DB1, DB2>

{

public:

void vvod();

void vivod();

Service();

Service(const char\* n\_poezd, DB1 n\_spead, DB1 n\_god, DB1 n\_width, const char\* service);

~Service();

Service<DB1, DB2>(const Service& temp);

Service& operator = (const Service<DB1, DB2>& rgd);

protected:

char\* service;

};

template<class DB1, class DB2>

void Service<DB1, DB2>::vvod()

{

RGD<DB1, DB2>::vvod();

Service<DB1, DB2>::vvod\_M();

cout << "Введите род службы: ";

cin >> service;

}

template<class DB1, class DB2>

void Service<DB1, DB2>::vivod()

{

RGD<DB1, DB2>::vivod();

cout << endl;

Material<DB1, DB2>::vivod\_M();

cout << "Род службы:" << service << endl;

}

template<class DB1, class DB2>

Service<DB1, DB2>::Service() : RGD<DB1, DB2>()

{

service = new char[11]{ "passenger" };

}

template<class DB1, class DB2>

Service<DB1, DB2>::Service(const char\* n\_poezd, DB1 n\_spead, DB1 n\_god, DB1 n\_width, const char\* service)

{

if (service = "")

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

this->service = service;

}

template<class DB1, class DB2>

Service<DB1, DB2>::~Service()

{

RGD<DB1, DB2>::~RGD();

}

template<class DB1, class DB2>

Service<DB1, DB2>::Service<DB1, DB2>(const Service& temp)

{

if (temp == NULL)

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

service = temp.service;

this->poezd = new char[strlen(temp.poezd) + 1];

strcpy(this->poezd, temp.poezd);

this->spead = temp.spead;

this->god = temp.god;

this->width = temp.width;

}

template<class DB1, class DB2>

Service<DB1, DB2>& Service<DB1, DB2> :: operator = (const Service& rgd)

{

service = rgd.service;

this->poezd = new char[strlen(rgd.poezd) + 1];

strcpy(this->poezd, rgd.poezd);

this->spead = rgd.spead;

this->god = rgd.god;

this->width = rgd.width;

return (\*this);

}

int main()

{

setlocale(LC\_ALL, "Russian");

cout << "Ошибки:" << endl;

// Пустое название

try

{

RGD<int, double> a("", 140, 2011, 3480);

}

catch (const char\* e)

{

cout << e << endl;

}

// Отрицательная скорость

try

{

RGD<int, double> b("Swallow", -140, 2011, 3480);

}

catch (const char\* e)

{

cout << e << endl;

}

// Отрицательный год

try

{

RGD<int, double> c("Swallow", 140, -2011, 3480);

}

catch (const char\* e)

{

cout << e << endl;

}

// Отрицательная ширина

try

{

RGD<int, double> d("Swallow", 140, 2011, -3480);

}

catch (const char\* e)

{

cout << e << endl;

}

// Пустое название материала вагона

try

{

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

}

catch (const char\* e)

{

cout << e << endl;

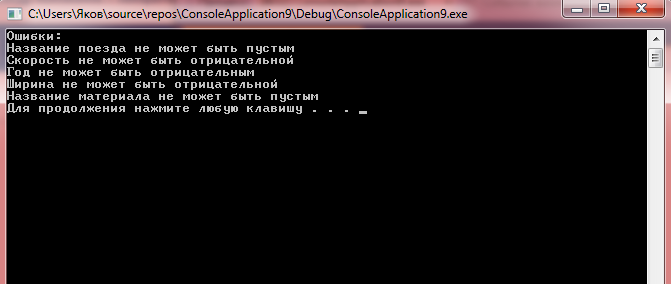
}

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.

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

В блоке catch.

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.

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().