МИНОБРНАУКИ РОССИИ

ФГБОУ ВО Череповецкий государственный университет

Институт информационных технологий

Кафедра: Математическое и программное обеспечение ЭВМ

Дисциплина: Объектно-ориентированное программирование

*Лабораторная работа* *10*

**ПРОГРАММИРОВАНИЕ ОБРАБОТКИ**

**ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ**

Выполнил:

студент гр. 1ПИб-01-21оп Викторов Д.А.

Проверил:

зав. кафедрой Ершов Е.В.

Череповец, 2020 г.

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.

# 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,который следует сразу за пробным блоком.

1. Опишите синтаксис обработчика исключения.

catch (формальный\_аргумент)

составная\_конструкция

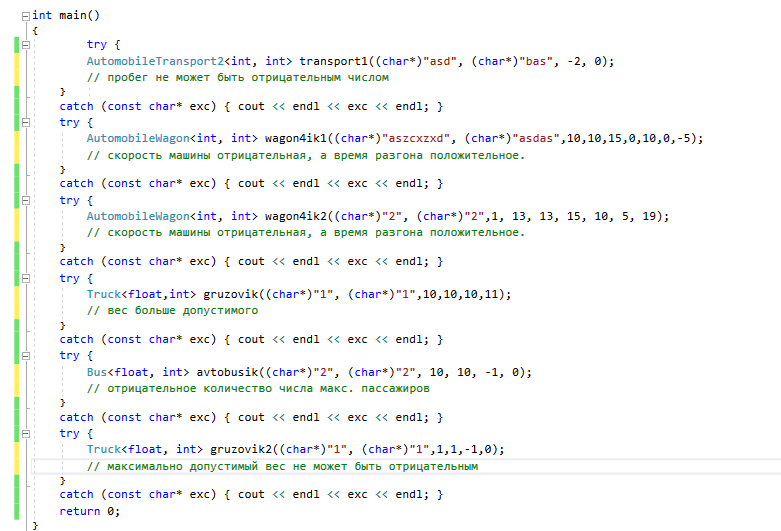
1. Что такое спецификация исключения?

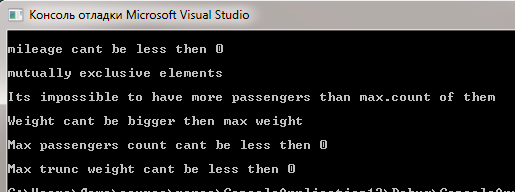
Синтаксически спецификация исключения является частью объявления и определения функции. Если спецификация исключения опущена, то предполагается, что такой функцией может быть возбуждено произвольное исключение. Например:

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().

Тесты





Код:

#include "pch.h"

#include <iostream>

#include <cstdlib>

#include <exception>

using namespace std;

template <class T1, class T2>

class AutomobileTransport2 { // базовый класс

public:

void virtual Print();

AutomobileTransport2();

AutomobileTransport2(char\* CarBrand, char\* CarModel, T1 km, T2 time);

AutomobileTransport2(const AutomobileTransport2<T1, T2> &a);

~AutomobileTransport2();

bool Equal(const  AutomobileTransport2<T1, T2>& b);

void Copy(const  AutomobileTransport2<T1, T2>& a);

AutomobileTransport2<T1, T2>Summa(const  AutomobileTransport2<T1, T2>& c);

 AutomobileTransport2<T1, T2>operator =(const  AutomobileTransport2<T1, T2>& a);

AutomobileTransport2<T1, T2>operator +(const  AutomobileTransport2<T1, T2>& c);

bool operator >(const  AutomobileTransport2<T1, T2>& a);

bool operator <(const  AutomobileTransport2<T1, T2>& a);

bool operator ==(const  AutomobileTransport2<T1, T2>& a);

bool operator !=(const  AutomobileTransport2<T1, T2>& a);

bool operator >=(const  AutomobileTransport2<T1, T2>& a);

bool operator <=(const  AutomobileTransport2<T1, T2>& a);

protected:

char\* carModel;

char\* carBrand;

T1 mileage;

T2 time0To100;

};

template <class T1, class T2> // наследуемый класс

class Truck : public virtual  AutomobileTransport2<T1, T2> { //одиночное наследование

public:

Truck(const Truck<T1, T2> &a); //копирующий конструктор

Truck<T1, T2>operator =(const  Truck<T1, T2>& a); // перегрузка оператора "равно"

Truck() // конструктор по умолчанию

:AutomobileTransport2<T1, T2>::AutomobileTransport2()

{

maxTruncCar = 2;

currentWeight = 1;

};

Truck(char\* CarBrand, char\* CarModel, T1 km, T2 time, int MaxTruncCar, int CurrentWeight) // конструктор с параметрами

:AutomobileTransport2<T1, T2>::AutomobileTransport2(CarBrand, CarModel, km, time)

{

maxTruncCar = MaxTruncCar;

currentWeight = CurrentWeight;

if (maxTruncCar < 0)

throw "Max trunc weight cant be less then 0"; // Логическая ошибка: макс. вместимость багажника не может быть отрицательной

if (currentWeight > maxTruncCar)

throw "Weight cant be bigger then max weight"; // Логическая ошибка: вес находящийся в багажнике машины не может быть больше чем её макс. вместимость

};

void Print() //компонетная функция

{

AutomobileTransport2<T1, T2>::Print();

cout << ", MaxTrunc: " << maxTruncCar << ", Current weight: " << currentWeight;

}

protected: // поля принадлежащие этому классу

int maxTruncCar;

int currentWeight;

};

template<class T1, class T2>

Truck<T1, T2>::Truck(const Truck<T1, T2>& a)

{

Truck<T1, T2>::time0To100 = a.Truck<T1, T2>::time0To100;

Truck<T1, T2>::mileage = a.Truck<T1, T2>::mileage;

Truck<T1, T2>::maxTruncCar = a.Truck<T1, T2>::maxTruncCar;

Truck<T1, T2>::currentWeight = a.Truck<T1, T2>::currentWeight;

Truck<T1, T2>::carBrand = new char[strlen(a.Truck<T1, T2>::carBrand) + 1];

strcpy(Truck<T1, T2>::carBrand, a.Truck<T1, T2>::carBrand);

Truck<T1, T2>::carModel = new char[strlen(a.Truck<T1, T2>::carModel) + 1];

strcpy(Truck<T1, T2>::carModel, a.Truck<T1, T2>::carModel);

}

template<class T1, class T2>

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

{

if (this == &a)

return \*this;

if ((char\*)a.carBrand == "" || (char\*)a.carModel == "")

throw "Impossible to assign the characterestics of an empty truck"; // не возможно присвоить значения машины, у которой нет названия марки изготовителя и марки самого грузовика

Truck<T1, T2>::time0To100 = a.Truck<T1, T2>::time0To100;

Truck<T1, T2>::mileage = a.Truck<T1, T2>::mileage;

Truck<T1, T2>::maxTruncCar = a.Truck<T1, T2>::maxTruncCar;

Truck<T1, T2>::currentWeight = a.Truck<T1, T2>::currentWeight;

Truck<T1, T2>::carBrand = new char[strlen(a.Truck<T1, T2>::carBrand) + 1];

strcpy(Truck<T1, T2>::carBrand, a.Truck<T1, T2>::carBrand);

Truck<T1, T2>::carModel = new char[strlen(a.Truck<T1, T2>::carModel) + 1];

strcpy(Truck<T1, T2>::carModel, a.Truck<T1, T2>::carModel);

return \*this;

}

template <class T1, class T2>

class Bus : public virtual  AutomobileTransport2<T1, T2> { // одиночное наследование

public:

Bus(const Bus<T1, T2> &a)

{

Bus<T1, T2>::time0To100 = a.Bus<T1, T2>::time0To100;

Bus<T1, T2>::mileage = a.Bus<T1, T2>::mileage;

Bus<T1, T2>::maxPassengers = a.Bus<T1, T2>::maxTruncCar;

Bus<T1, T2>::currentPassengers = a.Bus<T1, T2>::currentPassengers;

Bus<T1, T2>::carBrand = new char[strlen(a.Bus<T1, T2>::carBrand) + 1];

strcpy(Bus<T1, T2>::carBrand, a.Bus<T1, T2>::carBrand);

Bus<T1, T2>::carModel = new char[strlen(a.Bus<T1, T2>::carModel) + 1];

strcpy(Bus<T1, T2>::carModel, a.Bus<T1, T2>::carModel);

};//копирующий конструктор

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

{

if (this == &a)

return \*this;

if ((char\*)a.carBrand == "" || (char\*)a.carModel == "")

throw "Impossible to assign the characterestics of an empty bus";// не возможно присвоить значения машины, у которой нет названия марки изготовителя и марки самого автобуса

Bus<T1, T2>::time0To100 = a.Bus<T1, T2>::time0To100;

Bus<T1, T2>::mileage = a.Bus<T1, T2>::mileage;

Bus<T1, T2>::maxPassengers = a.Bus<T1, T2>::maxPassengers;

Bus<T1, T2>::currentPassengers = a.Bus<T1, T2>::currentPassengers;

Bus<T1, T2>::carBrand = new char[strlen(a.Bus<T1, T2>::carBrand) + 1];

strcpy(Bus<T1, T2>::carBrand, a.Bus<T1, T2>::carBrand);

Bus<T1, T2>::carModel = new char[strlen(a.Bus<T1, T2>::carModel) + 1];

strcpy(Bus<T1, T2>::carModel, a.Bus<T1, T2>::carModel);

return \*this;

}

; // перегрузка оператора "равно"

Bus() // конструктор по умолчанию

:AutomobileTransport2<T1, T2>::AutomobileTransport2()

{

maxPassengers = 0;

currentPassengers = 0;

};

Bus(char\* CarBrand, char\* CarModel, T1 km, T2 time, int MaxPassengers, int CurrentPassengers) // конструктор с параметрами

:AutomobileTransport2<T1, T2>::AutomobileTransport2(CarBrand, CarModel, km, time)

{

if (MaxPassengers < 0)

throw "Max passengers count cant be less then 0"; // максимальное количество пассажиров не может быть отрицательным

if (CurrentPassengers > MaxPassengers)

throw "Its impossible to have more passengers than max.count of them"; // невозможно перевозить больше пассажиров чем максимально возможно.

maxPassengers = MaxPassengers;

currentPassengers = CurrentPassengers;

};

void Print() // компонетная функция

{

AutomobileTransport2<T1, T2>::Print();

cout << "Max passengers count:" << maxPassengers << ", Current passengers count:"

<< currentPassengers;

}

protected: // поля принадлежащие этому классу

int maxPassengers;

int currentPassengers;

};

// Множественное наследование

template <class T1, class T2>

class AutomobileWagon : public Bus<T1, T2>, public Truck<T1, T2> {

public:

AutomobileWagon() :Truck<T1, T2>::Truck(), Bus()  // конструктор по умолчанию

{

averageSpeed = 0;

};

AutomobileWagon(char\* CarBrand, char\* CarModel, T1 km, T2 time, int MaxPassengers, int CurrentPassengers, int MaxTruncCar, int CurrentWeight, int AverageSpeed) // конструктор с параметрами

:Truck<T1, T2>::Truck(CarBrand, CarModel, km, time, MaxTruncCar, CurrentWeight), Bus<T1, T2>::Bus(CarBrand, CarModel, km, time, MaxPassengers, CurrentPassengers)

{

if (AverageSpeed <= 0 && (int)time > 0 || AverageSpeed > 0 && (int)time <= 0)

throw "mutually exclusive elements"; // обнаружены два взаимоисключающих элемента(машина не может разгонятся до 100кмчас за какое-то время если средняя скорость равна или меньше нуля и наоборот

averageSpeed = AverageSpeed;

};

AutomobileWagon(const AutomobileWagon<T1, T2> &a)

{

AutomobileWagon<T1, T2>::time0To100 = a.AutomobileWagon<T1, T2>::time0To100;

AutomobileWagon<T1, T2>::mileage = a.AutomobileWagon<T1, T2>::mileage;

AutomobileWagon<T1, T2>::maxTruncCar = a.AutomobileWagon<T1, T2>::maxTruncCar;

AutomobileWagon<T1, T2>::currentWeight = a.AutomobileWagon<T1, T2>::currentWeight;

AutomobileWagon<T1, T2>::maxPassengers = a.AutomobileWagon<T1, T2>::maxTruncCar;

AutomobileWagon<T1, T2>::currentPassengers = a.AutomobileWagon<T1, T2>::currentPassengers;

AutomobileWagon<T1, T2>::carBrand = new char[strlen(a.AutomobileWagon<T1, T2>::carBrand) + 1];

strcpy(AutomobileWagon<T1, T2>::carBrand, a.AutomobileWagon<T1, T2>::carBrand);

AutomobileWagon<T1, T2>::carModel = new char[strlen(a.AutomobileWagon<T1, T2>::carModel) + 1];

strcpy(AutomobileWagon<T1, T2>::carModel, a.AutomobileWagon<T1, T2>::carModel);

};//копирующий конструктор

AutomobileWagon<T1, T2>operator =(const  AutomobileWagon<T1, T2>& a) // перегруженный оператор"="

{

if (this == &a)

return \*this;

if ((char\*)a.carBrand == "" || (char\*)a.carModel == "")

throw "Impossible to assign the characterestics of an empty bus";// не возможно присвоить значения машины, у которой нет названия марки изготовителя и марки самого универсального автомобиляъ.

AutomobileWagon<T1, T2>::time0To100 = a.AutomobileWagon<T1, T2>::time0To100;

AutomobileWagon<T1, T2>::mileage = a.AutomobileWagon<T1, T2>::mileage;

AutomobileWagon<T1, T2>::maxPassengers = a.AutomobileWagon<T1, T2>::maxPassengers;

AutomobileWagon<T1, T2>::maxTruncCar = a.AutomobileWagon<T1, T2>::maxTruncCar;

AutomobileWagon<T1, T2>::currentWeight = a.AutomobileWagon<T1, T2>::currentWeight;

AutomobileWagon<T1, T2>::currentPassengers = a.AutomobileWagon<T1, T2>::currentPassengers;

AutomobileWagon<T1, T2>::carBrand = new char[strlen(a.AutomobileWagon<T1, T2>::carBrand) + 1];

strcpy(AutomobileWagon<T1, T2>::carBrand, a.AutomobileWagon<T1, T2>::carBrand);

AutomobileWagon<T1, T2>::carModel = new char[strlen(a.AutomobileWagon<T1, T2>::carModel) + 1];

strcpy(AutomobileWagon<T1, T2>::carModel, a.AutomobileWagon<T1, T2>::carModel);

return \*this;

}

void Print() // компонентная функция

{

Truck<T1, T2>::Print();

cout << "Average speed: " << averageSpeed << ", Max passengers count:" << AutomobileWagon<T1, T2>::maxPassengers << ", Current passengers count:"

<< AutomobileWagon<T1, T2>::currentPassengers;

}

protected:

int averageSpeed;

};

int main()

{

try {

AutomobileTransport2<int, int> transport1((char\*)"asd", (char\*)"bas", -2, 0); // пробег не может быть отрицательным числом

}

catch (const char\* exc) { cout << endl << exc << endl; }

try {

AutomobileWagon<int, int> wagon4ik1((char\*)"aszcxzxd", (char\*)"asdas",10,10,15,0,10,0,-5); // скорость машины отрицательная, а время разгона положительное.

}

catch (const char\* exc) { cout << endl << exc << endl; }

try {

AutomobileWagon<int, int> wagon4ik2((char\*)"2", (char\*)"2",1, 13, 13, 15, 10, 5, 19); // скорость машины отрицательная, а время разгона положительное.

}

catch (const char\* exc) { cout << endl << exc << endl; }

try {

Truck<float,int> gruzovik((char\*)"1", (char\*)"1",10,10,10,11); // вес больше допустимого

}

catch (const char\* exc) { cout << endl << exc << endl; }

try {

Bus<float, int> avtobusik((char\*)"2", (char\*)"2", 10, 10, -1, 0); // отрицательное количество числа макс. пассажиров

}

catch (const char\* exc) { cout << endl << exc << endl; }

try {

Truck<float, int> gruzovik2((char\*)"1", (char\*)"1",1,1,-1,0); // максимально допустимый вес не может быть отрицательным

}

catch (const char\* exc) { cout << endl << exc << endl; }

return 0;

}

template<class T1, class T2>

void AutomobileTransport2<T1, T2>::Print()

{

if (carBrand == "" || carModel == "")

throw "names cant be empty";

cout << "Car brand:" << carBrand << ", Car model:" << carModel << ", mileage:" << mileage << ", Time from 0 to 100:" << time0To100;

}

template<class T1, class T2>

AutomobileTransport2<T1, T2>::AutomobileTransport2()

{

carBrand = new char[1];

strcpy(carBrand, "");

carModel = new char[1];

strcpy(carModel, "");

mileage = (T1)0;

time0To100 = (T2)0;

}

template<class T1, class T2>

AutomobileTransport2<T1, T2>::AutomobileTransport2(const AutomobileTransport2<T1, T2>& a)

{

if (&a != this) {

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

strcpy(carBrand, a.carBrand);

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

strcpy(carModel, a.carModel);

mileage = a.mileage;

time0To100 = a.time0To100;

}

}

template<class T1, class T2>

AutomobileTransport2<T1, T2>::~AutomobileTransport2()

{

delete[] carBrand;

delete[] carModel;

}

template<class T1, class T2>

bool AutomobileTransport2<T1, T2>::Equal(const AutomobileTransport2<T1, T2>& b)

{

return mileage == b.mileage && time0To100 == b.time0To100;

}

template<class T1, class T2>

void AutomobileTransport2<T1, T2>::Copy(const AutomobileTransport2<T1, T2>& a)

{

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

strcpy(carBrand, a.carBrand);

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

strcpy(carModel, a.carModel);

mileage = a.mileage;

time0To100 = a.time0To100;

}

template<class T1, class T2>

AutomobileTransport2<T1, T2> AutomobileTransport2<T1, T2>::Summa(const AutomobileTransport2<T1, T2>& c)

{

AutomobileTransport2<T1, T2> t;

char\* temp1 = new char[strlen(carBrand) + 1];

char\* temp2 = new char[strlen(carModel) + 1];

strcpy(temp1, carBrand);

strcpy(temp2, carModel);

t.mileage = mileage + c.mileage;

t.time0To100 = time0To100 + c.time0To100;

strcat(temp1, c.carBrand);

strcat(temp2, c.carModel);

t.carBrand = new char[strlen(temp1) + 1];

strcpy(t.carBrand, temp1);

t.carModel = new char[strlen(temp2) + 1];

strcpy(t.carModel, temp2);

return (t);

}

template<class T1, class T2>

AutomobileTransport2<T1, T2> AutomobileTransport2<T1, T2>::operator=(const AutomobileTransport2<T1, T2>& a)

{

if (this == &a)

return \*this;

if (a.carBrand == NULL ||a.carModel == NULL)

throw "Impossible to assign the characterestics of an empty automobile";// не возможно присвоить значения машины, у которой нет названия марки изготовителя и марки самого автомобиля

time0To100 = a.time0To100;

mileage = a.mileage;

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

strcpy(carBrand, a.carBrand);

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

strcpy(carModel, a.carModel);

return \*this;

}

template<class T1, class T2>

AutomobileTransport2<T1, T2> AutomobileTransport2<T1, T2>::operator+(const AutomobileTransport2<T1, T2>& c)

{

AutomobileTransport2<T1, T2> t;

char\* temp1 = new char[strlen(carBrand) + 1];

char\* temp2 = new char[strlen(carModel) + 1];

strcpy(temp1, carBrand);

strcpy(temp2, carModel);

t.mileage = mileage + c.mileage;

t.time0To100 = time0To100 + c.time0To100;

strcat(temp1, c.carBrand);

strcat(temp2, c.carModel);

t.carBrand = new char[strlen(temp1) + 1];

strcpy(t.carBrand, temp1);

t.carModel = new char[strlen(temp2) + 1];

strcpy(t.carModel, temp2);

return (t);

}

template<class T1, class T2>

bool AutomobileTransport2<T1, T2>::operator>(const AutomobileTransport2<T1, T2>& a)

{

return mileage + time0To100 > a.mileage + a.time0To100;

}

template<class T1, class T2>

bool AutomobileTransport2<T1, T2>::operator<(const AutomobileTransport2<T1, T2>& a)

{

return mileage + time0To100 < a.mileage + a.time0To100;

}

template<class T1, class T2>

bool AutomobileTransport2<T1, T2>::operator==(const AutomobileTransport2<T1, T2>& a)

{

return mileage + time0To100 == a.mileage + a.time0To100;

}

template<class T1, class T2>

bool AutomobileTransport2<T1, T2>::operator!=(const AutomobileTransport2<T1, T2>& a)

{

return mileage + time0To100 != a.mileage + a.time0To100;

}

template<class T1, class T2>

bool AutomobileTransport2<T1, T2>::operator>=(const AutomobileTransport2<T1, T2>& a)

{

return mileage + time0To100 >= a.mileage + a.time0To100;

}

template<class T1, class T2>

bool AutomobileTransport2<T1, T2>::operator<=(const AutomobileTransport2<T1, T2>& a)

{

return mileage + time0To100 <= a.mileage + a.time0To100;

}

template<class T1, class T2>

AutomobileTransport2<T1, T2>::AutomobileTransport2(char \* CarBrand, char \* CarModel, T1 km, T2 time)

{

carBrand = new char[strlen(CarBrand) + 1];

strcpy(carBrand, CarBrand);

carModel = new char[strlen(CarModel) + 1];

strcpy(carModel, CarModel);

mileage = km;

time0To100 = time;

if ((int)km < 0)

throw ("mileage cant be less then 0"); //пробег машины не может быть отрицательным.

}