Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

ОТЧЕТ

по лабораторной работе

**«ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ В ЯЗЫКЕ С++»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 124404 |  | Н.А. МОЙСЕНОВИЧ |
| Проверила |  | К.А. БОРЩЕВИЧ |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Минск 2022

**Цель**

Изучить особенности обработки исключительных ситуаций на языке С++.

**Краткие теоретические сведения**

Исключения (exceptions) – это возникновение непредвиденных условий времени выполнения, например, деление на ноль, невозможность выделения динамической памяти для создания нового объекта, потеря подключения к базе данных или ввод непредвиденных данных, которые нарушают нормальное функционирование программы. Обычно эти условия завершают выполнение программы с системной ошибкой. Язык С++ позволяет восстанавливать программу из этих условий и продолжать ее выполнение.

Способ сообщения о проблеме не подразумевает знания о том, какая именно часть программы будет справляться с создавшейся ситуацией. Сообщив о случившемся, обнаружившая проблему часть кода завершает работу.

Обработка исключительных ситуаций С++ лишена недостатков вышеназванных методов реагирования на ошибки. Обработка исключительных ситуаций позволяет использовать для представления информации об ошибке объект любого типа. Поэтому можно создать иерархию классов, которая будет предназначена для обработки различных типов аварийных событий. Это упростит, структурирует программу.

Обнаружившая исключение часть программы использует оператор throw для создания исключения. Оператор throw состоит из ключевого слова throw, сопровождаемого выражением, которое определяет тип передаваемого исключения. Оператор throw, как правило, завершается точкой с запятой, что делает его выражением.

Когда тип исключения в блоке catch совпадает с типом сгенерированного исключения, выполняется блок, связанный с этим catch. При завершении выполнения кода соответствующего обработчика управление переходит к оператору, находящемуся после всех блоков catch.

Объявленные в блоке try переменные или объекты недоступны вне этого блока, даже в блоке catch.

Поиск обработчика осуществляется по цепочке обращений в обратном порядке. Сначала поиск обработчика исключения осуществляется в той функции, в которой оно было сгенерировано. Если соответствующего раздела catch не найдено, то работа функции завершается, а поиск продолжается в той функции, которая вызвала функцию, в которой было создано исключение.

Если блок catch с соответствующим типом данных так и не будет найден, то управление перейдет к библиотечной функции terminate(), которая определена в заголовке exception. Поведение этой функции зависит от системы, но обычно она завершает выполнение программы.

Иногда возникает ситуация, при которой необходимо обработать исключительную ситуацию сначала на более низком уровне вложенности блока try, а затем передать ее на более высокий уровень для продолжения обработки. Для этого необходимо использовать оператор throw без аргументов.

При генерации исключения внутри конструктора процесс создания объекта прекращается.

**Вариант 18**

**Общая постановка задачи**



**Листинг файла Source.cpp**

#include "Shop.h"

#include "Input.h"

void Term()

{

cout << "There are some unprocessed exceptions, tell administrator about this issue " << endl;

exit(-1);

}

void BuyBoots(Bill\* bill,Cart\* cart)

{

Boots boots;

string brand;

float cost;

float size;

bool isRetry = true;

while (isRetry)

{

try

{

StringInput(&brand, "brand");

Input(&cost, "boots cost", InputType(Number));

Input(&size, "boots size", InputType(Number));

boots.SetInfo(size, brand, cost);

cart->Add(boots);

bill->AddSum(cost);

isRetry = false;

}

catch (const exception& ex)

{

cout << ex.what() << endl;

}

catch (InputException& ex)

{

cout << ex.what() << endl;

cout << "Wrong data was " << ex.GetErrorData() << endl;

}

}

}

void BuyHat(Bill\* bill,Cart\* cart)

{

Hat hat;

string brand;

string color;

float cost;

bool isRetry = true;

while (isRetry)

{

try

{

StringInput(&brand, "brand");

StringInput(&color, "color");

Input(&cost, "hat cost", InputType(Number));

hat.SetInfo(color, brand, cost);

cart->Add(hat);

bill->AddSum(cost);

isRetry = false;

}

catch (const exception& ex)

{

cout << ex.what() << endl;

}

catch (InputException& ex)

{

cout << ex.what() << endl;

cout << "Wrong data was " << ex.GetErrorData() << endl;

}

}

}

void CallCurrentFunction(Bill \*bill,Cart \*cart,int answer)

{

switch (answer)

{

case 1:

BuyHat(bill,cart);

break;

case 2:

BuyBoots(bill,cart);

break;

case 3:

(\*bill).Show();

break;

case 4:

(\*cart).Show();

break;

case 5:

(\*bill).PrepareBill(\*cart);

break;

case 6:

(\*bill).Write();

break;

case 7:

(\*bill).Read();

default:

exit(0);

break;

}

}

void ShowMenu(Bill\* bill, Cart\* cart)

{

float answer;

cout << "Choose an option " << endl;

cout << "1.Buy a hat" << endl;

cout << "2.Buy a pair of boots" << endl;

cout << "3.Show your bill" << endl;

cout << "4.Show your cart" << endl;

cout << "5.Prepare bill" << endl;

cout << "6.Write bill" << endl;

cout << "Or choose any other option to exit " << endl;

try

{

Input(&answer, "answer", InputType(Answer));

CallCurrentFunction(bill, cart, answer);

}

catch(const exception& ex)

{

cout << ex.what() << endl;

}

catch (InputException& ex)

{

cout << ex.what() << endl;

cout << ex.GetErrorData() << endl;

}

}

int main()

{

Bill bill;

Cart cart;

set\_terminate(Term);

while (true)

{

ShowMenu(&bill, &cart);

}

return 0;

}

**Листинг файла Shop.h**

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "json.hpp"

using std::vector;

using std::string;

using std::exception;

using std::ofstream;

using std::ifstream;

using std::cout;

using std::endl;

using nlohmann::json;

class Clothes

{

protected:

string Brand;

float Cost;

void SetBasicInfo(string brand, float cost);

void ShowBasicInfo();

public:

string GetBrand();

int GetCost();

};

class Hat: public Clothes

{

string \_color;

public:

void SetInfo(string color, string brand, float cost);

void Show();

string GetColor();

};

class Boots : public Clothes

{

float \_size;

public:

void SetInfo(float size, string brand, float cost);

void Show();

float GetSize();

};

class Cart

{

vector<Hat> \_hats;

vector<Boots> \_boots;

public:

Cart();

Cart(const Cart& other);

void Add(Hat hat);

void Add(Boots boots);

void Show();

bool Empty();

vector<Hat> GetHats();

vector<Boots> GetBoots();

};

class Bill

{

Cart \_cart;

float \_sum;

public:

Bill();

void AddSum(float value);

void PrepareBill(Cart cart);

void Show();

void Write();

void Read();

};

**Листинг файла Input.h**

#pragma once

#include <iostream>

#include <string>

using std::exception;

using std::numeric\_limits;

using std::streamsize;

using std::cout;

using std::cin;

using std::endl;

using std::string;

enum InputType

{

Number,

Age,

Answer

};

class InputException : public exception

{

float \_errorData;

public:

InputException(const char\* msg) : exception(msg){}

InputException(const char\* msg, float errorData) : exception(msg)

{

\_errorData = errorData;

}

float GetErrorData();

};

void StringInput(string\* data, string dataName);

void Input(float\* data, string dataName, InputType inputType);

**Листинг файла Input.cpp**

#include "Input.h"

void AgeInput(float tempData, string dataName)

{

if (tempData < 0 || tempData > 120)

throw InputException("Incorrect age amount", tempData);

}

void NumberInput(float tempData, string dataName)

{

if (tempData < 0 || tempData > INT\_MAX)

throw InputException("Incorrect data", tempData);

}

void AnswerInput(float tempData, string dataName)

{

if ((int)tempData != tempData)

throw InputException("Incorrect type of answer, try int num", tempData);

}

void Input(float\* data, string dataName, InputType inputType)

{

float tempData;

cout << "Enter " << dataName << endl;

cin >> tempData;

if (!cin)

{

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

throw exception("Incorrect data type input!");

}

switch (inputType)

{

case Number:

NumberInput(tempData, dataName);

\*data = tempData;

break;

case Age:

AgeInput(tempData, dataName);

\*data = tempData;

break;

case Answer:

AnswerInput(tempData, dataName);

\*data = tempData;

break;

default:

throw exception("Incorrect InputType!!!");

break;

}

}

void StringInput(string\* data, string dataName)

{

string tempData;

cout << "Enter " << dataName << endl;

cin >> tempData;

\*data = tempData;

}

float InputException::GetErrorData()

{

return \_errorData;

}

**Листинг файла Shop.h**

#include "Shop.h"

void Clothes::SetBasicInfo(string brand, float cost)

{

Cost = cost;

Brand = brand;

}

void Clothes::ShowBasicInfo()

{

cout << "Brand: " << Brand << "\tCost: " << Cost;

}

string Clothes::GetBrand()

{

return Brand;

}

int Clothes::GetCost()

{

return Cost;

}

void Boots::SetInfo(float size, string brand, float cost)

{

\_size = size;

SetBasicInfo(brand, cost);

}

void Boots::Show()

{

ShowBasicInfo();

cout << "\tSize: " << \_size << endl;

}

float Boots::GetSize()

{

return \_size;

}

void Hat::SetInfo(string color, string brand, float cost)

{

\_color = color;

SetBasicInfo(brand, cost);

}

void Hat::Show()

{

ShowBasicInfo();

cout << "\tColor: " << \_color << endl;

}

string Hat::GetColor()

{

return \_color;

}

Cart::Cart()

{

}

Cart::Cart(const Cart& other)

{

for (int i = 0; i < other.\_boots.size(); i++)

{

\_boots.insert(\_boots.begin(), other.\_boots.at(i));

}

for (int i = 0; i < other.\_hats.size(); i++)

{

\_hats.insert(\_hats.begin(), other.\_hats.at(i));

}

}

void Cart::Add(Hat hat)

{

\_hats.insert(\_hats.begin(), hat);

}

void Cart::Add(Boots boots)

{

\_boots.insert(\_boots.begin(), boots);

}

bool Cart::Empty()

{

if (\_hats.empty() && \_boots.empty())

return true;

return false;

}

void Cart::Show()

{

if (Empty())

throw exception("Trying to get access to empty cart");

Boots boots;

Hat hat;

for (int i = 0; i < \_boots.size(); i++)

{

boots = \_boots.at(i);

boots.Show();

}

for (int i = 0; i < \_hats.size(); i++)

{

hat = \_hats.at(i);

hat.Show();

}

}

vector<Hat> Cart::GetHats()

{

return \_hats;

}

vector<Boots> Cart::GetBoots()

{

return \_boots;

}

Bill::Bill()

{

\_cart = Cart();

}

void Bill::PrepareBill(Cart cart)

{

\_cart = cart;

}

void Bill::AddSum(float value)

{

\_sum += value;

}

void Bill::Show()

{

try

{

\_cart.Show();

cout << "Sum is " << \_sum << endl;

}

catch (const exception& ex)

{

cout << ex.what() << endl;

}

}

void Bill::Write()

{

json parsedBoot;

json parsedHat;

Hat unparsedHat;

Boots unparsedBoots;

ofstream fout;

fout.open("dataBase.json", ofstream::app);

for (int i = 0; i < \_cart.GetHats().size(); i++)

{

unparsedHat = \_cart.GetHats().at(i);

parsedHat["color"] = unparsedHat.GetColor();

parsedHat["brand"] = unparsedHat.GetBrand();

parsedHat["cost"] = unparsedHat.GetCost();

fout << parsedHat;

}

for (int i = 0; i < \_cart.GetBoots().size(); i++)

{

unparsedBoots = \_cart.GetBoots().at(i);

parsedBoot["size"] = unparsedBoots.GetSize();

parsedBoot["brand"] = unparsedBoots.GetBrand();

parsedBoot["cost"] = unparsedBoots.GetCost();

fout << parsedBoot;

}

fout.close();

}

void Bill::Read()

{

ifstream fin;

vector<json> parsedClothes;

fin.open("dataBase.json");

while (fin.end)

{

parsedClothes = json::parse(fin);

}

}

**Скриншоты**

**UML-диаграмма классов**



Рисунок 1 – Диаграмма классов

**Вывод**

Были изучены особенности обработки исключительных ситуаций на языке С++.