Министерство образования Республики Беларусь

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра экономической информатики

ЛАБОРАТОРНАЯ РАБОТА №2

«ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ В ЯЗЫКЕ С++»

Вариант №22

|  |  |
| --- | --- |
| Выполнил: | Прудник Д.А.  гр. 024402 |
| Проверил: | Бич Н.В. |

Минск 2021

**Цель работы:** изучить особенности обработки исключительных ситуаций на языке С++

## 1.1. Индивидуальное задание

Разработать набор классов (минимум 5 классов, связи между

классами: агрегация, композиция, наследование) по предметной области

«Тренажерный зал». Функционал программы должен позволить собрать

данные о необходимых тренажерах. Сгенерировать минимум пять типов

исключительных ситуаций. Реализовать перенаправление исключительных

ситуаций. Сгенерировать минимум одну исключительную ситуацию с

оператором new. Создать исключительную ситуацию в конструкторе и

продемонстрировать вызов конструкторов и деструкторов. Задать

собственную функцию завершения. Создать собственный (пользовательский)

класс исключения, сгенерировать исключение этого типа и обработать его. В

отчете отобразить созданную диаграмму классов

**1.2. Теоретические сведения**

Механизм обработки исключительных ситуаций (exception handling) является неотъемлемой частью языка С++. Исключения позволяют отделять код обнаружения проблемы от кода ее решения. Часть программы, ответственная за обнаружение проблемы, может передать информацию о возникшей ситуации другой части программы, которая специально предназначена для решения подобных проблем. Механизм обработки исключительных ситуаций предоставляет программисту средство реагирования на нештатные события и позволяет преодолеть ряд принципиальных недостатков следующих возможных методов обработки ошибок: − возврат функцией кода ошибки как возвращаемого значения или через выходные аргументы функций; − использование глобальных переменных для хранения ошибок; − использование оператора безусловного перехода goto или функций setjmp/longjmp. Функция setjmp сохраняет значения окружения для возврата управления программой в точку сохранения. Эта функция принимает аргумент и наполняет его значениями состояний переменных окружения в этой точке кода, для того, чтобы, в случае необходимости, можно было восстановить значения переменных окружения в более позднем вызове функции longjmp; − использование макроса препроцессора assert. Макрос assert() добавляет к программе процедуру диагностики. Если выражение ложно, то есть, результат сравнения равен нулю, макрос assert() пишет информацию о вызове в поток stderr и вызывает функцию abort() для завершения программы.

## 1.3. Листинг

#include<iostream>

#include<string>

#include<vector>

#include<exception>

#include<Windows.h>

using namespace std;

//class Exception

class Exception

{

private:

string m\_message;

public:

Exception(const string& message) :m\_message(message) {}

string What()const { return this->m\_message; }

};

template<class T>

class SeasonTicket

{

private:

T m\_costOfTicket;

int m\_amountOfDays;

int m\_amountOfTrainings;

public:

SeasonTicket() :m\_amountOfDays(0), m\_amountOfTrainings(0), m\_costOfTicket(0) {}

SeasonTicket(T cost, int days, int trainings) :m\_costOfTicket(cost), m\_amountOfDays(days), m\_amountOfTrainings(trainings) {}

~SeasonTicket()

{

cout << "\nObject client was deleted due the ";

}

void SetCost(T cost) { this->m\_costOfTicket = cost; }

void SetDays(int days) { this->m\_amountOfDays = days; }

void SetTrainings(int trainings) { this->m\_amountOfTrainings = trainings; }

T GetCost() const { return this->m\_costOfTicket; }

int GetAmountOfDays() const { return this->m\_amountOfDays; }

int GetAmountOfTrainings() const { return this->m\_amountOfTrainings; }

};

class Client :public SeasonTicket<double>

{

private:

string m\_fullName;

string m\_regularCustomer;

public:

Client() {}

Client(double cost, int days, int trainings, string name, string regularity) :SeasonTicket(cost, days, trainings)

{

this->m\_fullName = name;

this->m\_regularCustomer = regularity;

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

{

if (name[i] > '0' && name[i] < '9') throw runtime\_error("There is numbers in the name!\n");

}

}

~Client()

{

cout << "\nObject client was deleted!\n";

}

void SetFullName(string name) { this->m\_fullName = name; }

void SetRegularity(string state) { this->m\_regularCustomer = state; }

string GetName() const { return this->m\_fullName; }

string GetRegularity() const { return this->m\_regularCustomer; }

};

class Trainer

{

private:

string m\_nameOfTrainer;

int m\_amount;

public:

Trainer() :m\_amount(0), m\_nameOfTrainer("NULL") {}

Trainer(string name, int amount) :m\_nameOfTrainer(name), m\_amount(amount) {}

Trainer(const Trainer& obj)

{

this->m\_amount = obj.m\_amount;

this->m\_nameOfTrainer = obj.m\_nameOfTrainer;

}

Trainer& operator++()

{

this->m\_amount++;

return \*this;

}

void DisplayTrainerInformation()

{

cout << "Description - " << this->m\_nameOfTrainer

<< "\n Amount - " << this->m\_amount << endl;

}

void SetNameOfTrainer(string name) { this->m\_nameOfTrainer = name; }

void SetAmount(int amount) { this->m\_amount = amount; }

string GetNameOfTrainer() const { return this->m\_nameOfTrainer; }

int GetAmount() const { return this->m\_amount; }

};

class Database

{

private:

template<class T>

class EnteringOperations

{

public:

void CleareBuffer()

{

if (char(cin.peek()) == '\n')

cin.ignore();

if (cin.fail())

{

cin.clear();

cin.ignore(32767, '\n');

}

}

//Simple exception

string GetCorrectAnswer()

{

string userInput;

bool isFinded = true;

while (isFinded)

{

getline(cin, userInput);

try

{

if (userInput == "Y" || userInput == "y" || userInput == "Yes" || userInput == "yes")

{

isFinded = false;

}

else if (userInput == "N" || userInput == "n" || userInput == "no" || userInput == "No")

{

isFinded = false;

}

else

{

throw runtime\_error("Not correct answer!");

cin.clear();

}

}

catch (runtime\_error exception)

{

cout << "Error: " << exception.what() << "\nTry again:";

}

}

return userInput;

}

string GetCorrectValue(int minValue, int middleValue, int maxValue, int userValue, bool& isRight)

{

string clientTicket;

if (userValue == minValue)

{

clientTicket = "Standart";

isRight = true;

}

else if (userValue == middleValue)

{

clientTicket = "Special";

isRight = true;

}

else if (userValue == maxValue)

{

clientTicket = "Premium";

isRight = true;

}

else

{

isRight = false;

}

return clientTicket;

}

T GetValue()

{

T userValue;

while (!(cin >> userValue) && cin.get() != '\n')

{

cout << "Erorr connected with your input,try again: ";

cin.clear();

while (cin.get() != '\n');

}

return userValue;

}

};

vector<Client\*> m\_clients;

EnteringOperations<int> m\_operations;

public:

void CleareBuffer()

{

m\_operations.CleareBuffer();

}

bool IsEmpty()

{

if (m\_clients.size() == 0) return true;

else return false;

}

string GetCorrectAnswer()

{

return m\_operations.GetCorrectAnswer();

}

string GetCorrectValue(int minValue, int middleValue, int maxValue, int userValue, bool& isRightCost)

{

return m\_operations.GetCorrectValue(minValue, middleValue, maxValue, userValue, isRightCost);

}

bool GetCorrectAmountOfDays(int userValue)

{

if (userValue >= 1 && userValue <= 31) return true;

else

{

cout << "\nNot correct value,try again\n";

return false;

}

}

int GetValue()

{

return m\_operations.GetValue();

}

//Constructor exeption

void InsertNewPersonInDatabase()

{

string name = "", regularity;

int maxPrice = 300, middlePrice = 200, minPrice = 100, cost = 0, amountOfDays = 0, amountOfTrainings = 0;

bool isRightCost = false;

CleareBuffer();

cout << "Please, enter full name of new Client: ";

getline(cin, name);

cout << "Is this client regular in trainigs(Y/N): ";

bool isWorking = true;

regularity = GetCorrectAnswer();

cout << "Choose the cost of Season ticket: "

<< "\n1) - 100 (Standard)"

<< "\n2) - 200 (Special)"

<< "\n3) - 300 (Premium)"

<< "\n\nYour choice>> ";

cost = GetValue();

GetCorrectValue(minPrice, middlePrice, maxPrice, cost, isRightCost);

while (isWorking)

{

if (isRightCost == true) isWorking = false;

else

{

cout << "\nChoose right cost,try again: ";

cost = GetValue();

GetCorrectValue(minPrice, middlePrice, maxPrice, cost, isRightCost);

}

}

isWorking = true;

bool isRightAmountOfdays = false;

cout << "Please, enter amount of days you want to go(max 31 days): ";

amountOfDays = GetValue();

while (isWorking)

{

if (GetCorrectAmountOfDays(amountOfDays) == true) isWorking = false;

else

{

cout << "Enter value: ";

amountOfDays = GetValue();

}

}

isWorking = true;

cout << "Please, enter how much training will you have with the trainer(0 - 7): ";

amountOfTrainings = GetValue();

while (isWorking)

{

if (amountOfTrainings > 7 || amountOfTrainings < 0)

{

cout << "Not correct value, try again: ";

amountOfTrainings = GetValue();

}

else isWorking = false;

}

try

{

m\_clients.push\_back(new Client(cost, amountOfDays, amountOfTrainings, name, regularity));

}

catch (runtime\_error exception)

{

cout << "Error: " << exception.what() << endl;

}

}

void DeleteFromDatabase()

{

int numberOfClient;

int ordinalNumberInList = 1;

cout << "Enter number of client you would like to delete: " << endl;

cout << "\n\tDatabase of Clients" << endl;

cout << "===================================================" << endl << endl;

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

{

cout << ordinalNumberInList << ") " << m\_clients[i]->GetName() << endl;

ordinalNumberInList++;

}

cout << "\n===================================================" << endl;

cout << "\nYour choice>> ";

numberOfClient = GetValue();

numberOfClient--;

bool isFinded = false;

if (numberOfClient >= 0 && numberOfClient <= m\_clients.size())

{

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

{

if (i == numberOfClient)

{

isFinded = true;

}

else isFinded = false;

}

}

if (isFinded == true)

{

cout << "\nClient (" << m\_clients[numberOfClient]->GetName() << ") successfully deleted" << endl;

m\_clients.erase(m\_clients.begin() + numberOfClient);

}

else cout << "\nPerson with such number not finded!\n";

}

void DisplayPersonsDatabase()

{

int numberInList = 1;

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

{

cout << "\n\tNumber of Client: " << numberInList << "\nFull Name: " << m\_clients[i]->GetName() << endl;

cout << "Amount of days : " << m\_clients[i]->GetAmountOfDays() << endl;

cout << "Regular visitor: " << m\_clients[i]->GetRegularity() << endl;

numberInList++;

}

cout << "\nAmount of Clients: " << m\_clients.size() << endl;

}

void EditPersonInDatabase()

{

int numberOfClient = 0;

cout << "Enter number of client you would like to edit: ";

int ordinalNumberInList = 1;

cout << "\n\tDatabase of Clients" << endl;

cout << "===================================================" << endl << endl;

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

{

cout << ordinalNumberInList << ") " << m\_clients[i]->GetName() << endl;

ordinalNumberInList++;

}

cout << "\n===================================================" << endl;

cout << "\nYour choice>> ";

numberOfClient = GetValue();

numberOfClient--;

if (numberOfClient >= 0 && numberOfClient <= m\_clients.size())

{

string name = "";

string regularity = "";

int amountOfDays = 0;

int amountOfTrainings = 0;

int maxPrice = 300;

int minPrice = 100;

int cost = 0;

CleareBuffer();

cout << "Please, enter new full name of new Client: ";

getline(cin, name);

cout << "Is this client regular in trainigs(Y/N): ";

regularity = GetCorrectAnswer();

cout << "Choose the cost of Season ticket: "

<< "\n1) - 100 (Standard)"

<< "\n2) - 200 (Special)"

<< "\n3) - 300 (Premium)"

<< "\n\nYour choice>> ";

cost = GetValue();

cout << "Please, enter amount of days you want to go: ";

amountOfDays = GetValue();

cout << "Please, enter how much training will you have with the trainer(0 - 7): ";

amountOfTrainings = GetValue();

bool isWorking = true;

while (isWorking)

{

if (amountOfTrainings < 0 && amountOfTrainings>7)

{

cout << "Try again: ";

amountOfDays = GetValue();

}

else isWorking = false;

}

cout << "\nClient (" << m\_clients[numberOfClient]->GetName() << ") successfully edited" << endl;

m\_clients[numberOfClient]->SetFullName(name);

m\_clients[numberOfClient]->SetRegularity(regularity);

m\_clients[numberOfClient]->SetTrainings(amountOfTrainings);

m\_clients[numberOfClient]->SetDays(amountOfDays);

}

}

void CleareListOfClients()

{

m\_clients.clear();

try

{

if (m\_clients.size() != 0) throw "All memory not clean!(Clients)\n";

}

catch (const char\* exception)

{

cout << "Error: " << exception << endl;

throw;

}

}

};

class Gym

{

private:

int m\_amountOfTrainers;

Database m\_database;

vector<Trainer\*> m\_trainers;

vector<string> m\_necassaryTrainers;

public:

void ShowClientsMenu()

{

cout << "\tMenu to manage clients"

<< "\n(1) - Add new client to database"

<< "\n(2) - Show list of clients"

<< "\n(3) - Delete client from database"

<< "\n(4) - Find and Edit characteristics of clients"

<< "\n(0) - Get back"

<< "\n\nYour choice>> ";

}

void ShowTrainingsMenu()

{

cout << "\tMenu to manage trainings"

<< "\n(1) - Add trainer to the gym"

<< "\n(2) - Display list of Necessary trainers"

<< "\n(3) - Display list of trainers"

<< "\n(0) - Get back"

<< "\n\nYour choice>> ";

}

void ShowCommonMenu()

{

cout << "\tCommon menu to manage Gym"

<< "\n(1) - Manage clients database"

<< "\n(2) - Replenish trainings database"

<< "\n(0) - Exit"

<< "\n\nYour choice>> ";

}

bool CheckForEnoughtOfTrainers(int minimalValue)

{

try

{

if (m\_trainers.size() != minimalValue)

{

throw runtime\_error("Not enought of trainers in the GYM\n");

}

}

catch (runtime\_error except)

{

cout << "Finded except: " << except.what() << endl;

}

}

bool IsEmpty() const

{

if (m\_trainers.size() == 0) return true;

else return false;

}

void CreateListOfNecessaryTrainers()

{

m\_necassaryTrainers.push\_back("Butterfly");

m\_necassaryTrainers.push\_back("Treadmill");

m\_necassaryTrainers.push\_back("Stepper");

m\_necassaryTrainers.push\_back("Orbitrek");

m\_necassaryTrainers.push\_back("Climber");

m\_necassaryTrainers.push\_back("Hammer");

m\_necassaryTrainers.push\_back("Smith trainer");

m\_necassaryTrainers.push\_back("Crossover");

m\_necassaryTrainers.push\_back("Gravitron");

m\_necassaryTrainers.push\_back("Dumbbell rack");

}

void DisplayListOfNecessaryTrainers()

{

int ordinalNumber = 1;

cout << "\tList of neccesery trainers" << endl;

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

{

cout << ordinalNumber << ") " << m\_necassaryTrainers[i] << endl;

ordinalNumber++;

}

}

void AddNewTrainerInGym()

{

string nameOfTrainer;

int amountOfTrainers;

Database operations;

operations.CleareBuffer();

cout << "Please, enter name of trainer: ";

getline(cin, nameOfTrainer);

cout << "Enter amount of " << nameOfTrainer << "(<10): ";

amountOfTrainers = operations.GetValue();

bool isWorking = true;

while (isWorking)

{

if (amountOfTrainers < 0 || amountOfTrainers > 10)

{

cout << "\nNot correct value, try again: \n";

amountOfTrainers = operations.GetValue();

}

else isWorking = false;

}

m\_trainers.push\_back(new Trainer(nameOfTrainer, amountOfTrainers));

bool isFinded = false;

int numberOfTrainer;

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

{

if (nameOfTrainer == m\_necassaryTrainers[i])

{

isFinded = true;

numberOfTrainer = i;

}

}

if (isFinded == true)

{

m\_necassaryTrainers.erase(m\_necassaryTrainers.begin() + numberOfTrainer);

}

}

void DisplayListOfTrainersInGym()

{

int ordinalNumber = 1;

cout << "\tList of trainers in the Gym" << endl;

for (const auto& trainer : m\_trainers)

{

cout << ordinalNumber << ")";

trainer->DisplayTrainerInformation();

cout << endl;

ordinalNumber++;

}

}

void CleareListOfTrainers()

{

m\_trainers.clear();

try

{

if (m\_trainers.size() != 0) throw "All memory not clean(Trainers)!\n";

}

catch (const char\* exception)

{

cout << "Error: " << exception << endl;

throw;

}

}

bool IsEmptyNecessaryList()

{

if (m\_necassaryTrainers.size() == 0) return true;

else return false;

}

void SetAmountOfTrainers(int number) { this->m\_amountOfTrainers = number; }

int GetAmountOfTrainersByOnePosition() const { return this->m\_amountOfTrainers; }

};

//Shutdown function

void Quit()

{

exit(0);

}

void ShutdownProgram()

{

cout << "\nFunction of shuttingdown the program!\n";

set\_terminate(Quit);

throw 0;

}

int main()

{

Gym gym;

Database operations;

bool isWorking = true;

bool isWorkingClietnsMenu = true;

bool isWorkingTrainingsMenu = true;

gym.CreateListOfNecessaryTrainers();

while (isWorking)

{

gym.ShowCommonMenu();

int menuChoice = operations.GetValue();

switch (menuChoice)

{

case 1:

isWorkingClietnsMenu = true;

while (isWorkingClietnsMenu)

{

gym.ShowClientsMenu();

switch (menuChoice = operations.GetValue())

{

case 1:

operations.InsertNewPersonInDatabase();

break;

case 2:

if (operations.IsEmpty() == false) operations.DisplayPersonsDatabase();

else cout << "\nDatabase is empty!\n";

break;

case 3:

if (operations.IsEmpty() == false) operations.DeleteFromDatabase();

else cout << "\nDatabase is empty!\n";

break;

case 4:

if (operations.IsEmpty() == false) operations.EditPersonInDatabase();

else cout << "\nDatabase is empty!\n";

break;

case 0:

isWorkingClietnsMenu = false;

break;

default:

cout << "\nThere is no case with such input, try again!\n";

break;

}

system("pause");

system("cls");

}

break;

case 2:

isWorkingTrainingsMenu = true;

while (isWorkingTrainingsMenu)

{

gym.ShowTrainingsMenu();

switch (menuChoice = operations.GetValue())

{

case 1:

gym.AddNewTrainerInGym();

break;

case 2:

if (gym.IsEmptyNecessaryList() == false) gym.DisplayListOfNecessaryTrainers();

else cout << "\nList of necessary trainers is empty!\n";

break;

case 3:

if (gym.IsEmpty() == false) gym.DisplayListOfTrainersInGym();

else cout << "\nList of trainers is empty!\n";

break;

case 0:

isWorkingTrainingsMenu = false;

break;

default:

cout << "\nThere is no case with such input, try again!\n";

break;

}

system("pause");

system("cls");

}

break;

case 0:

try

{

ShutdownProgram();

}

catch (int exception)

{

}

isWorking = false;

break;

default:

cout << "\nThere is no case with such input, try again!\n";

break;

}

system("pause");

system("cls");

}

//Redirects exception

try

{

gym.CleareListOfTrainers();

operations.CleareListOfClients();

}

catch (const char\* exception)

{

cout << "Clean up memory!\n";

}

}

## 1.4. Вывод

Изучены особенности обработки исключительных ситуаций на языке С++