exeption.hpp:

#pragma once

//базовий класс виключень

class MyException {

const char \*message = nullptr;

public:

MyException(const char \*message = "Unknown exception") : message(message) {}

const char\* what() const {

return message;

}

};

//виключення: "Не вірний аргумент!"

struct Wrong\_Argument\_Exeption : MyException {

Wrong\_Argument\_Exeption(const char \*message = "Wrong argument exception") : MyException(message) {}

};

//виключення: "Вихід за межі!"

struct Out\_Of\_Range\_Exception : MyException {

Out\_Of\_Range\_Exception(const char \*message = "Out of range exception") : MyException(message) {}

};

//виключення: "Поганий файл!"

struct Bad\_File\_Exeption : MyException {

Bad\_File\_Exeption(const char \*message = "Bad file exception") : MyException(message) {}

};

//виключення: "Поганий файл для запису!"

struct Bad\_File\_Write\_Exeption : MyException {

Bad\_File\_Write\_Exeption(const char \*message = "Bad file write exception") : MyException(message) {}

};

//виключення: "Поганий файл для зчитування!"

struct Bad\_File\_Read\_Exeption : MyException {

Bad\_File\_Read\_Exeption(const char \*message = "Bad file read exception") : MyException(message) {}

};

Date.h:

#pragma once

#include <iostream>

#include <iomanip>

#include "exeption.hpp"

using namespace std;

class Date {/\*(Рік, Місяць, День)\*/

private:

int day,

month,

year;

public:

//конструктор за замовчуванням

Date() {

day = 1 + rand() % 31;

month = 1 + rand() % 12;

year = 1980 + rand() % (2017 - 1980);

}

//конструктор ініціалізації

Date(int a\_day, int a\_month, int a\_year) {

if (day < 0 || day > 31 || month < 0 || month > 12 || year > 2017 || year < 1980)

throw Wrong\_Argument\_Exeption();

this->day = a\_day;

this->month = a\_month;

this->year = a\_year;

}

//конструктор копіювання

Date(const Date &copy) {

day = copy.day;

month = copy.month;

year = copy.year;

}

//перевантажений оператор виведення в потік

friend ostream& operator << (ostream &os, const Date &o\_date) {

os << setw(10) << o\_date.day;

os << setw(3) << o\_date.month;

os << setw(5) << o\_date.year;

return os;

}

//перевантажений оператор введення з потоку

friend istream& operator >> (istream &is, Date &o\_date) {

is >> o\_date.day;

is >> o\_date.month;

is >> o\_date.year;

return is;

}

void Set(); //метод змінення полів

void Show();//метод який виводить на консоль таблицю

int getDay() {

return day;

}

int getMonth() {

return month;

}

int getYear() {

return year;

}

//деструктор

~Date() {}

};

//реалізація методу змінення полів

void Date::Set() {

bool flag = true;

cout << " Введите год: ";

while (flag) {

cin.clear();

rewind(stdin);

cin >> year;

if (!cin || year < 1980 || year > 2017 )

cout << " Неверное значение, введите снова: ";

else flag = false;

}

flag = true;

cout << " Введите месяц: ";

while (flag) {

cin.clear();

rewind(stdin);

cin >> month;

if (!cin || month < 1 || month > 12 || !cin.good())

cout << " Неверное значение, месяц должен быть от 1 до 12: ";

else flag = false;

}

flag = true;

cout << " Введите день: ";

if (month == 2 && year % 4 == 0) {

while (flag) {

cin.clear();

rewind(stdin);

cin >> day;

if (!cin || day < 1 || day > 29)

cout << " Неверное значение, в этом месяце 29 дней: ";

else flag = false;

}

}

else if (month == 2 && year % 2 == 1) {

while (flag) {

cin.clear();

rewind(stdin);

cin >> day;

if (!cin || day < 1 || day > 28)

cout << " Неверное значение, в этом месяце 28 дней: ";

else flag = false;

}

}

if (month % 2 == 0 && month != 2) {

while (flag) {

cin.clear();

rewind(stdin);

cin >> day;

if (!cin || day < 1 || day > 30)

cout << " Неверное значение, в этом месяце 30 дней: ";

else flag = false;

}

}

else {

while (flag) {

cin.clear();

rewind(stdin);

cin >> day;

if (!cin || day < 1 || day > 31) {

cout << " Неверное значение, в месяце 30 дней: ";

}

else flag = false;

}

}

}

//реалізація виводу таблички на консоль

void Date::Show() {

cout << " " << setw(2) << setfill('0') << day << "." << setw(2) << setfill('0') << month << "." << setw(4) << year << "|";

}

Person.h:

#pragma once

#pragma warning(disable : 4996)

#include "Date.h"

#include <iostream>

using namespace std;

//масив імен за замовчуванням

char \*d\_names[] = { "Алексей", "Денис", "Юрий", "Иван", "Илья", "Андрей", "Богдан", "Владислав", "Анатолий", "Александр", "Максим", "Евгений", "Виктор", "Павел" };

//масив прізвищ за замовчуванням

char \*d\_surnames[] = { "Гарник", "Трухан", "Данько", "Засека", "Артамонов", "Мартыненко", "Скороход", "Юрченко", "Бочок", "Шматко", "Бевза", "Войтович", "Колотуша", "Ташу" };

class Person {/\*(Прізвище, Ім’я, Дата (народження))\*/

private:

Date birthday\_date;

char \*name,

\*surname;

public:

//конструктор зазамовчуванням

Person() : birthday\_date() {

int i = rand() % 14;

name = new char[strlen(d\_names[i]) + 1];

strcpy(name, d\_names[i]);

i = rand() % 14;

surname = new char[strlen(d\_surnames[i]) + 1];

strcpy(surname, d\_surnames[i]);

}

//конструктор ініціалізації

Person(int a\_day, int a\_month, int a\_year, char \*a\_name, char \*a\_surname) : birthday\_date(a\_day, a\_month, a\_year) { //конструктор ініціалізації

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

strcpy(this->name, a\_name);

this->surname = new char[strlen(a\_surname) + 1];

strcpy(this->surname, a\_surname);

}

//конструктор копіювання

Person(const Person &copy) {

this->birthday\_date = copy.birthday\_date;

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

strcpy(name, copy.name);

surname = new char[strlen(copy.surname) + 1];

strcpy(surname, copy.surname);

}

//перевантажений оператор виведення в потік

friend ostream& operator << (ostream &os, const Person &o\_person) {

os << setw(15) << o\_person.name << endl;

os << setw(15) << o\_person.surname << endl;

return os;

}

//перевантажений оператор введення з потоку

friend istream& operator >> (istream &is, Person &o\_person) {

is >> o\_person.name;

is >> o\_person.surname;

return is;

}

virtual void Set();//метод змінення полів

virtual void Show();//метод який виводить на консоль таблицю

char \*getName() {

return name;

}

char\* getSurname() {

return surname;

}

Date getBDate() {

return birthday\_date;

}

//віртуальний деструктор

virtual ~Person() {

if (name)

delete[] name;

if (surname)

delete[] surname;

}

};

//реалізація методу змінення полів

void Person::Set() {

int i;

bool flg = true;

name = new char[30];

cout << " Введите имя: ";

while (flg) {

rewind(stdin);

cin.clear();

cin >> name;

for (i = 0; name[i]; i++)

if (name[i] >= '!' && name[i] <= '@' || name[i] >= '[' && name[i] <= '`' || name[i] == '~' || name[i] == ',' || name[i] == ' ')

break;

if (name[i])

cout << " Введены неверные символы! Введите имя еще раз: ";

else flg = false;

}

flg = true;

surname = new char[30];

cout << " Введите фамилию: ";

while (flg) {

rewind(stdin);

cin.clear();

cin >> surname;

for (i = 0; surname[i]; i++)

if (surname[i] >= '!' && surname[i] <= '@' || surname[i] >= '[' && surname[i] <= '`' || surname[i] == '~' || surname[i] == ',' || name[i] == ' ')

break;

if (surname[i])

cout << " Введены неверные символы! Введите фамилию еще раз: ";

else flg = false;

}

cout << " Введите дату рождения" << endl;

birthday\_date.Set();

}

//реалізація виводу таблички на консоль

void Person::Show() {

cout << setw(10) << setfill(' ') << name << " ";

cout << setw(10) << surname << "|";

birthday\_date.Show();

}

Celner.h:

#pragma once

#include "Person.h"

#include <iostream>

using namespace std;

//масив категорій кельнерів

char \*d\_category[] = { "Первая", "Вторая", "Третья", "Четвертая", "Пятая" };

class Celner : public Person {/\*(Персона, Категорія)\*/

private:

char \*category;

public:

//конструктор за замовчуванням

Celner() : Person() {

int i = rand() % 5;

category = new char[strlen(d\_category[i]) + 1];

strcpy(category, d\_category[i]);

}

//конструктор ініціалізації

Celner(int a\_day, int a\_month, int a\_year, char \*a\_name, char \*a\_surname, char \*a\_category) : Person(a\_day, a\_month, a\_year, a\_name, a\_surname) { //конструктор ініціалізації

this->category = new char[strlen(a\_category) + 1];

strcpy(this->category, a\_category);

}

//конструктор копіювання

Celner(const Celner &copy) : Person(copy) {

category = new char[strlen(copy.category) + 1];

strcpy(category, copy.category);

}

//перевантажений оператор виведення в потік

friend ostream& operator << (ostream &os, Celner &o\_celner) {

os << o\_celner.getName();

os << o\_celner.getSurname();

os << o\_celner.getBDate();

os << o\_celner.category;

return os;

}

//перевантажений оператор введення з потоку

friend istream& operator >> (istream &os, Celner &o\_celner) {

os >> o\_celner.getName();

os >> o\_celner.getSurname();

os >> o\_celner.getBDate();

os >> o\_celner.category;

return os;

}

void Set();//метод змінення полів

void Show();//метод який виводить на консоль таблицю

char \* getCategory() {

return category;

}

//деструктор

~Celner() {

if (category)

delete[] category;

}

};

//реалізація методу змінення полів

void Celner::Set() {

cout << " Введите данные о кельнере" << endl;

Person::Set();

int i;

bool flg = true;

category = new char[30];

cout << " Введите категорию кельнера: ";

while (flg) {

rewind(stdin);

cin.clear();

cin >> category;

for (i = 0; category[i]; i++)

if (category[i] >= '!' && category[i] <= '@' || category[i] >= '[' && category[i] <= '`' || category[i] == '~' || category[i] == ',')

break;

if (category[i])

cout << " Введены неверные символы! Введите категорию еще раз: ";

else flg = false;

}

}

//реалізація виводу таблички на консоль

void Celner::Show() {

Person::Show();

cout << setw(14) << setfill(' ') << category << "|";

}

Type\_Of\_Dish.h:

#pragma once

#include <iostream>

char \*d\_type[] = { "Первое", "Второе", "Десерт" };

using namespace std;

class TypeOfDish {/\*(перша, друга, десерт)\*/

private:

char \*type;

public:

//конструктор за замовчуванням

TypeOfDish() {

int i = rand() % 3;

type = new char[strlen(d\_type[i]) + 1];

strcpy(type, d\_type[i]);

}

//конструктор ініціалізації

TypeOfDish(char \*a\_type) {

this->type = new char[strlen(a\_type) + 1];

strcpy(this->type, a\_type);

}

//конструктор копіювання

TypeOfDish(const TypeOfDish &copy) {

type = copy.type;

}

//перевантажений оператор виведення в потік

friend ostream& operator << (ostream &os, const TypeOfDish &o\_type\_of\_dish) {

os << setw(15) << o\_type\_of\_dish.type;

return os;

}

//перевантажений оператор введення з потоку

friend istream& operator >> (istream &is, TypeOfDish &o\_type\_of\_dish) {

is >> o\_type\_of\_dish.type;

return is;

}

void Set();//метод змінення полів

void Show();//метод який виводить на консоль таблицю

char \*getType() {

return type;

}

//деструктор

~TypeOfDish() {

if (type)

delete[] type;

}

};

//реалізація методу змінення полів

void TypeOfDish::Set() {

bool flg = true;

type = new char[10];

cout << " Выберите тип блюда" << endl;

cout << "\n 1. Первое"<< endl;

cout << "\n 2. Второе" << endl;

cout << "\n 3. Десерт" << endl;

int a;

cout << "\n Ваш выбор: ";

while (flg == true) {

rewind(stdin);

cin.clear();

cin >> a;

switch (a) {

case 1:

type = d\_type[0];

flg = false;

break;

case 2:

type = d\_type[1];

flg = false;

break;

case 3:

type = d\_type[2];

flg = false;

break;

default:

cout << "Такого типа блюда нету. Введите еще раз." << endl;

}

}

}

//реалізація виводу таблички на консоль

void TypeOfDish::Show() {

cout << setw(9) << type << "|";

}

Dish.h:

#pragma once

#include "Type\_of\_dish.h"

#include <iostream>

using namespace std;

//масив назв страв за замовчуванням

char \*d\_nameOfDish[] = { "Бендерики", "Бифштекс", "Борщ", "Винегрет", "Гриль", "Драники", "Жаркое", "Жульен", "Запеканка", "Зразы", "Капустняк", "Кебаб", "Лагман", "Лазанья", "Лапша", "Мафины", "Налистники", "Оливье", "Паштет", "Харчо", "Хачапури", "Шаурма", "Шашлык", "Яичница" };

class Dish {/\*(Назва, Ціна, Тип страви)\*/

private:

TypeOfDish type\_of\_dish;

char \*nameOfDish;

int price;

public:

//конструктор за замовчуванням

Dish() : type\_of\_dish() {

int i = rand() % 24;

nameOfDish = new char[strlen(d\_nameOfDish[i]) + 1];

strcpy(nameOfDish, d\_nameOfDish[i]);

price = rand() % 10001;

}

//конструктор ініціалізації

Dish(char \*a\_type, char \*a\_nameOfDish, int a\_price) : type\_of\_dish(a\_type){

if (a\_price < 0)

throw Wrong\_Argument\_Exeption();

this->nameOfDish = a\_nameOfDish;

this->price = a\_price;

}

//конструктор копіювання

Dish(const Dish &copy) : type\_of\_dish(copy.type\_of\_dish) {

nameOfDish = new char[strlen(copy.nameOfDish) + 1];

strcpy(nameOfDish, copy.nameOfDish);

price = copy.price;

}

//перевантажений оператор виведення в потік

friend ostream& operator << (ostream &os, const Dish &o\_dish) {

os << setw(15) << o\_dish.type\_of\_dish;

os << setw(15) << o\_dish.nameOfDish;

os << setw(15) << o\_dish.price;

return os;

}

//перевантажений оператор введення з потоку

friend istream& operator >> (istream &is, Dish &o\_dish) {

is >> o\_dish.type\_of\_dish;

is >> o\_dish.nameOfDish;

is >> o\_dish.price;

return is;

}

void Set();//метод змінення полів

void Show();//метод який виводить на консоль таблицю

int getPrice() {

return price;

}

char \*getNameOfDish() {

return nameOfDish;

}

char \*getTypeOfDish() {

return type\_of\_dish.getType();

}

~Dish() {

if (nameOfDish)

delete[] nameOfDish;

}

};

//реалізація методу змінення полів

void Dish::Set() {

int i;

bool flg = true;

nameOfDish = new char[30];

cout << " Введите название блюда: ";

while (flg) {

rewind(stdin);

cin.clear();

cin >> nameOfDish;

for (i = 0; nameOfDish[i]; i++)

if (nameOfDish[i] >= '!' && nameOfDish[i] <= '@' || nameOfDish[i] >= '[' && nameOfDish[i] <= '`' || nameOfDish[i] == '~' || nameOfDish[i] == ',' || nameOfDish[i] == ' ')

break;

if (nameOfDish[i])

cout << " Введены неверные символы! Введите название еще раз: ";

else flg = false;

}

type\_of\_dish.Set();

flg = true;

cout << " Введите цену блюда: ";

while (flg) {

cin.clear();

rewind(stdin);

cin >> price;

if (!cin || price < 0 || price > 10000)

cout << " Неверное значение, введите снова: ";

else flg = false;

}

}

//реалізація виводу таблички на консоль

void Dish::Show() {

cout << setw(11) << nameOfDish << "|";

type\_of\_dish.Show();

cout << setw(9) << price << "|" << endl;

}

Order.h:

#pragma once

#include "Celner.h"

#include "Dish.h"

#include "exeption.hpp"

#include <iostream>

using namespace std;

class Order : public Celner{/\*(Список страв, Кельнер, Дата (замовлення))\*/

private:

Dish \*\*listOfDishes;

Date order\_date;

int size;

public:

//конструктор за замовчуванням

Order() : Celner(), order\_date() {

size = 1 + rand() % 5;

listOfDishes = new Dish\*[size];

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

listOfDishes[i] = new Dish;

}

//конструктор за замовчуванням в якому є можливість задати кількість страв

Order(int n) : Celner(), order\_date() {

if (n < 0)

throw Wrong\_Argument\_Exeption();

size = n;

listOfDishes = new Dish\*[size];

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

listOfDishes[i] = new Dish;

}

//конструктор ініціалізації

Order(Dish \*\*a\_listOfDishes, int count\_of\_dishes, int year\_of\_order, int month\_of\_order, int day\_of\_order, int day\_of\_celner, int month\_of\_celner, int year\_of\_celner, char \*celner\_name, char \*celner\_surname, char \*celner\_category) :

Celner(day\_of\_celner, month\_of\_celner, year\_of\_celner, celner\_name, celner\_surname, celner\_category), order\_date(year\_of\_order, month\_of\_order, day\_of\_order) {

this->size = count\_of\_dishes;

listOfDishes = new Dish\*[size];

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

listOfDishes[i] = new Dish(\*a\_listOfDishes[i]);

}

//конструктор копіювання

Order(const Order &copy) {

size = copy.size;

listOfDishes = new Dish\*[size];

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

listOfDishes[i] = new Dish(\*copy.listOfDishes[i]);

}

//перевантажений оператор індексації

Dish &operator [](int index) {

if (index < 0 || index >= size)

throw Out\_Of\_Range\_Exception("Ошибка! Обращение к несуществующему элементу массива.");

else

return \*listOfDishes[index];

}

//перевантажений оператор виведення в потік

friend ostream& operator << (ostream &os, Order &o\_order) {

os << setw(15) << o\_order.order\_date;

os << setw(15) << o\_order.getName();

os << setw(15) << o\_order.getSurname();

os << o\_order.getBDate();

os << setw(15) << o\_order.getCategory();

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

os << \*o\_order.listOfDishes[i] << endl;

return os;

}

//перевантажений оператор введення з потоку

friend istream& operator >> (istream &is, Order &o\_order) {

is >> o\_order.order\_date;

is >> o\_order.getName();

is >> o\_order.getSurname();

is >> o\_order.getCategory();

is >> o\_order.getCategory();

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

is >> \*o\_order.listOfDishes[i];

return is;

}

void Set();//метод змінення полів

void Show();//метод який виводить на консоль таблицю

bool CompareNum(int);//метод порівнювання чисел який створений для пошуку по числовому полю

void CompareStr(char\*);//метод порівнювання символьних полів який створений для пошуку по символьному полю

Dish \*\*getArr() {

return listOfDishes;

}

int getSize() {

return size;

}

//деструктор

~Order() {

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

delete[] listOfDishes[i];

}

};

//реалізація методу змінення полів

void Order::Set() {

Celner::Set();

cout << " Введите дату заказа" << endl;

order\_date.Set();

cout << " Введите количество блюд: ";

bool flg = true;

while (flg) {

cin.clear();

rewind(stdin);

cin >> size;

if (!cin || size <= 0)

cout << " Ошибка, введите снова: ";

else flg = false;

}

for (int i = 0; i < size; i++) {

cout << "\n Блюдо №" << i + 1 << endl;

listOfDishes[i]->Set();

}

}

//реалізація виводу таблички на консоль

void Order::Show() {

order\_date.Show();

Celner::Show();

for (int i = 0; i < size; i++) {

if (i > 0)

cout << "|" << setw(86) << i + 1 << "|";

else

cout << setw(9) << i + 1 << "|";

listOfDishes[i]->Show();

}

}

//реалізація методу порівнювання чисел

bool Order::CompareNum(int tmp) {

if (tmp == order\_date.getDay() || tmp == order\_date.getMonth() || tmp == order\_date.getYear()) {

return true;

}

if (tmp == Celner::getBDate().getDay() || tmp == Celner::getBDate().getMonth() || tmp == Celner::getBDate().getYear()) {

Celner::Show();

}

for (int i = 0; i < size; i++){

if (tmp == listOfDishes[i]->getPrice()) {

listOfDishes[i]->Show();

}

}

}

//реалізація методу порівнювання символьних полів

void Order::CompareStr(char\* tmp) {

if (strcmp(tmp, Celner::getName()) == 0 || strcmp(tmp, Celner::getSurname()) == 0 || strcmp(tmp, Celner::getCategory()) == 0) {

Celner::Show();

cout << endl;

}

for (int i = 0; i < size; i++) {

if (strcmp(tmp, listOfDishes[i]->getNameOfDish()) == 0 || strcmp(tmp, listOfDishes[i]->getTypeOfDish()) == 0)

listOfDishes[i]->Show();

}

}

Daily\_Balance.h:

#pragma once

#include "Order.h"

#include "exeption.hpp"

#include <iostream>

#include <iomanip>

using namespace std;

class DailyBalance {/\*(Список замовлень страв)\*/

private:

Order \*\*ordList;

int size;

public:

//конструктор за замовчуванням

DailyBalance() {

size = 1 + rand() % 5;

ordList = new Order\*[size];

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

ordList[i] = new Order();

}

//конструктор за замовчуванням в якому є можливість задати кількість замовлень

DailyBalance(int n) {

if (n < 0)

throw Wrong\_Argument\_Exeption();

size = n;

ordList = new Order\*[size];

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

ordList[i] = new Order;

}

//конструктор за замовчуванням в якому є можливість задати кількість замовлень та страв у цих замовленнях

DailyBalance(int n, int n1) {

if (n < 0 || n1 < 0)

throw Wrong\_Argument\_Exeption();

size = n;

ordList = new Order\*[size];

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

ordList[i] = new Order(n1);

}

//конструктор ініціалізації

DailyBalance(int count\_of\_orders, Dish \*\*a\_listOfDishes, int count\_of\_dishes, int year\_of\_order, int month\_of\_order, int day\_of\_order, int day\_of\_celner, int month\_of\_celner, int year\_of\_celner, char \*celner\_name, char \*celner\_surname, char \*celner\_category) {

this->size = count\_of\_orders;

ordList = new Order\*[size];

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

ordList[i] = new Order(a\_listOfDishes, count\_of\_dishes, year\_of\_order, month\_of\_order, day\_of\_order, day\_of\_celner, month\_of\_celner, year\_of\_celner, celner\_name, celner\_surname, celner\_category);

}

//конструктор копіювання

DailyBalance(const DailyBalance &copy) {

size = copy.size;

ordList = new Order\*[size];

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

ordList[i] = new Order(\*copy.ordList[i]);

}

//перевантажений оператор індексації

Order &operator [](int index) {

if (index < 0 || index >= size)

throw Out\_Of\_Range\_Exception("Ошибка! Обращение к несуществующему элементу массива.");

else

return \*ordList[index];

}

//перевантажений оператор виведення в потік

friend ostream& operator << (ostream &os, const DailyBalance &o\_daily\_balance) {

for (int i = 0; i < o\_daily\_balance.size; i++)

os << \*o\_daily\_balance.ordList[i] << endl;

return os;

}

//перевантажений оператор введення з потоку

friend istream& operator >> (istream &is, DailyBalance &o\_daily\_balance) {

for (int i = 0; i < o\_daily\_balance.size; i++)

is >> \*o\_daily\_balance.ordList[i];

return is;

}

void Set();//метод змінення полів

void Show();//метод який виводить на консоль таблицю

Order \*\*getArr() {

return ordList;

}

int getSize() {

return size;

}

//деструктор

~DailyBalance() {

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

delete[] ordList[i];

}

};

//реалізація методу змінення полів

void DailyBalance::Set() {

cout << "Введите количество заказов: ";

bool flg = true;

while (flg) {

cin.clear();

rewind(stdin);

cin >> size;

if (!cin || size <= 0)

cout << "Ошибка, введите снова: ";

else flg = false;

}

for (int i = 0; i < size; i++) {

cout << "\nЗаказ №" << i + 1 << endl;

ordList[i]->Set();

}

}

//реалізація виводу таблички на консоль

void DailyBalance::Show() {

cout << "------------------------------------------------------------------------------------------------------------------------" << endl;

cout << "|" << setw(10) << "№ Заказа|" << setw(15) << "Дата заказа|" << setw(30) << "Кельнер" << setw(22) << "|";

cout << setw(23) << "Блюда" << setw(19) << "|" << endl;

cout << "------------------------------------------------------------------------------------------------------------------------" << endl;

cout << "|" << setw(10) << "-|" << setw(15) << "-|" << setw(22) << "Имя Фамилия|" << setw(15) << "Дата рождения|" << setw(15) << "Категория|";

cout << setw(10) << "№ Блюда|" << setw(12) << "Название|" << setw(10) << "Тип|" << setw(10) << "Цена(грн)|" << endl;

cout << "------------------------------------------------------------------------------------------------------------------------" << endl;

for (int i = 0; i < size; i++) {

cout << "|" << setw(9) << i + 1 << "|";

ordList[i]->Show();

cout << "------------------------------------------------------------------------------------------------------------------------" << endl;

}

}

main.cpp:

#include <locale.h>

#include <conio.h>

#include <iostream>

#include "Daily\_balance.h"

#include <fstream>

#include <string>

#include <windows.h>

#include "exeption.hpp"

using namespace std;

//у звіті обгрунтовано чому наступні дві функції не є шаблонними

//функція знаходження статистичних даних(середня кількість став у замовленні)

void Average\_Counts\_Of\_Dishes(DailyBalance &db) {

float sum = 0;

for (int i = 0; i < db.getSize(); i++)

sum += db[i].getSize();

sum /= db.getSize();

cout << "Среднее значение количества блюд в заказе: " << sum << endl;

}

//функція знаходження статистичних даних(середня вартість замовлень)

void Average\_Price(DailyBalance &db) {

float sum = 0;

for (int i = 0; i < db.getSize(); i++)

for (int j = 0; j < db[i].getSize(); j++)

sum += db[i][j].getPrice();

sum /= db.getSize();

cout << "Среднее значение стоимости заказов: " << sum << " грн." << endl;

}

//функція пошуку за числовим та текстовим полем

void Find(DailyBalance &db) {

int n;

int num\_tmp;

char \*str\_tmp = new char[30];

while (true) {

cout << "\n\t1. Поиск по числовому полю" << endl;

cout << "\t2. Поиск по строковому полю" << endl;

cout << "\t3. Выход в меню" << endl;

cout << "\nВаш выбор: ";

rewind(stdin);

cin.clear();

cin >> n;

switch (n) {

case 1:

cout << "\n Введите число: ";

cin >> num\_tmp;

for (int i = 0; i < db.getSize(); i++) {

if (db[i].CompareNum(num\_tmp) == true) {

db[i].Show();

}

}

break;

case 2:

cout << "\n Введите слово: ";

cin >> str\_tmp;

for (int i = 0; i < db.getSize(); i++)

db[i].CompareStr(str\_tmp);

break;

case 3:

return;

break;

default:

cout << "\nТакого пункта нет в меню. Введите еще раз." << endl;

break;

}

}

delete[] str\_tmp;

}

template<typename T>//шаблонна функція запису до текстового файлу масиву обєктів

void Text\_Write(T \*\*arr, int n) {

ofstream Ft("Text.txt");

if (!Ft.is\_open())

throw Bad\_File\_Write\_Exeption("Ошибка! Текстовый файл не был открыт для записи!");

else {

for (int i = 0; i < n; i++)

Ft << \*arr[i] << endl;

cout << "\nТекстовый файл успешно создан!" << endl;

}

Ft.close();

}

template<typename T>//шаблонна функція зчитування з текстового файлу масиву обєктів до іншого массиву обєктів

void Text\_Read(T \*\*arr, int n) {

ifstream Ft("Text.txt");

if (!Ft.is\_open())

throw Bad\_File\_Read\_Exeption("Ошибка! Текстовый файл не был открыт для считывания!");

else {

for (int i = 0; i < n; i++) {

Ft >> \*arr[i];

arr[i]->Show();

}

cout << "\nТекстовый файл успешно считан!" << endl;

}

Ft.close();

}

template<typename T>//шаблонна функція запису до бінарного файлу масиву обєктів

void Binary\_Write(T \*\*arr, int n) {

ofstream Fb("Binary.dat", ios::binary);

if (!Fb.is\_open())

throw Bad\_File\_Write\_Exeption("Ошибка! Бинарный файл не был открыт для записи!");

else {

for (int i = 0; i < n; i++)

Fb.write((char\*)&arr[i], sizeof(&arr[i]));

cout << "\nБинарный файл успешно создан!" << endl;

}

Fb.close();

}

template<typename T>//шаблонна функція зчитування з бінарного файлу масиву обєктів до іншого массиву обєктів

void Binary\_Read(T \*\*arr, int n) {

ifstream Fb("Binary.dat", ios::binary);

if (!Fb.is\_open())

throw Bad\_File\_Read\_Exeption("Ошибка! Бинарный файл не был открыт для считывания!");

else {

cout << "\nБинарный файл успешно считан!" << endl;

for (int i = 0; i < n; i++) {

Fb.read((char\*)&arr[i], sizeof(T));

arr[i]->Show();

}

}

Fb.close();

}

void Menu() {

int n = 0;

DailyBalance db(10, 10);

DailyBalance db1(10, 10);

DailyBalance db2(10, 10);

while (true){

cout << "\nМеню" << endl;

cout << "\t1. Ввести данные" << endl;

cout << "\t2. Вывести данные в таблице" << endl;

cout << "\t3. Запись данных на диск" << endl;

cout << "\t4. Считывание данных" << endl;

cout << "\t5. Поиск" << endl;

cout << "\t6. Выход" << endl;

cout << "\nВаш выбор: ";

cin >> n;

switch (n){

case 1:

db.Set();

break;

case 2:

db.Show();

Average\_Counts\_Of\_Dishes(db);

Average\_Price(db);

break;

case 3:

Binary\_Write(db.getArr(), db.getSize());

Text\_Write(db.getArr(), db.getSize());

break;

case 4:

Binary\_Read(db1.getArr(), db1.getSize());

Text\_Read(db2.getArr(), db2.getSize());

break;

case 5:

Find(db);

break;

case 6:

exit(0);

break;

default:

cout << "\nТакого пункта нет в меню. Введите еще раз." << endl;

break;

}

}

}

int main() {

srand(time(NULL));

setlocale(LC\_ALL, "RUS");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

try {

Menu();

}

catch (MyException &ex) { //контролюючий блок який відловлює всі виключення(структури) які наслідуються від Базового классу виключень

cout << ex.what(); //показує яке саме виключення було схоплене

}

\_getch();

return 0;

}