Міністерство освіти і науки України

Національний технічний університет України „КПІ”

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки

інформації та управління

**ЗВІТ**

до лабораторної роботи № 7

з дисципліни ООП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Виконав**  **студент** |  | *ІП-61 Кушка Михайло Олександрович* |  |  |
|  |  | (№ групи, прізвище, ім’я, по батькові ) |  |  |
|  |  |  |  |  |
| **Прийняв** |  | *Головченко М.М.* |  |  |
|  |  | (посада, прізвище, ім’я, по батькові ) |  |  |

Київ 2017

ЗМІСТ

1. Мета роботи 3

2. Постановка задачі 4

3. Аналітичні викладки 5

4. UML-діаграма класів 6

5. Вихідний код програми 7

prototypes.cpp 7

prototypes.hpp 8

stdafx.hpp 10

main.cpp 10

6. Приклади роботи програми 13

7. Висновки 14

# Мета роботи

Мета роботи - вивчити особливості STD бібліотеки в C++ 11 та навчитися її використовувати, а саме: застосовувати модуль random та “розумні” покажчики.

# Постановка задачі

**Стоматологічна клініка.** Пацієнт звертається в стоматологічну клініку зі скаргою. Консультант опитує пацієнта, фіксує його скарги, проводить огляд, після чого призначає необхідні процедури, лікування і вартість.

Після консультації пацієнт направляється до одного з лікарів (виходячи з захворювань пацієнта), де проходить необхідний курс лікування.

Сформувати колекцію даних з інформацією про пацієнтів їх захворюваннях і лікуючих лікарів.

Дані про предметну область повинні генеруватися випадковим чином. Для створення динамічних об'єктів використовувати Smart pointers.

# Аналітичні викладки

У STL є як велика кількість шаблонів, так і класів і функцій. Ми можемо їх використовувати з ООП або без нього. У STL є 3 основні компоненти.

* Ітератори
* Контейнери
* Алгоритми

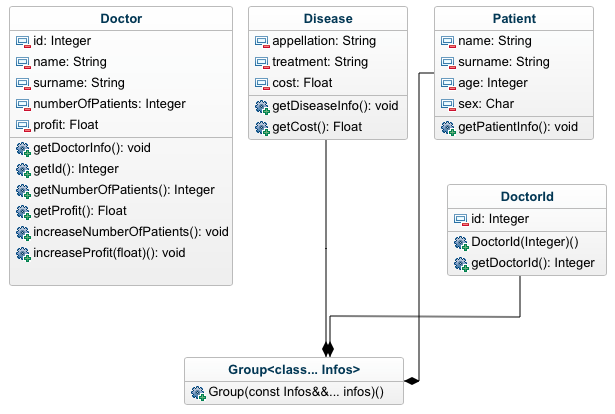
Ітератор - це аналог покажчика, за допомогою якого можна отримувати доступ до різних елементів даних. Можна використовувати і пару ітераторів для задання діапазону. Як і для покажчика, для отримання даних з ітераторів їх необхідно розіменувати за допомогою операції \*.

Контейнери - це структури даних, такі як списки, черги і тому подібне. Доступ до даних, що знаходяться всередині контейнера здійснюється за допомогою ітераторів. Виділяють наступні контейнери:

* vector - лінійний масив;
* list – двозв’язний список;
* deque - черга з двостороннім доступом;
* set - асоціативний масив унікальних ключів;
* multiset - асоціативний масив з можливістю дублювання ключів;
* map - асоціативний масив з унікальними ключами і значеннями;
* multimap -ассоціатівний масив з можливістю дублювання ключів і значень;
* stack - структура даних типу стек;
* queue - структура даних типу черга.

Алгоритми - це шаблони функцій, за допомогою яких проводяться операції по роботі з даними.

# UML-діаграма класів



# Вихідний код програми

## prototypes.cpp

//

// prototypes.cpp

// Lab6

//

// Created by Kushka Misha on 12/6/17.

// Copyright © 2017 Kushka Misha. All rights reserved.

//

#include "prototypes.hpp"

// display all data from Patient class

void Patient::getPatientInfo() {

cout << "Name: " << name << endl

<< "Surname: " << surname << endl

<< "Age: " << age << endl

<< "Sex: " << sex << endl

<< "Doctor's id: " << id + 1 << endl << endl;

}

// display all data from Disease class

void Disease::getDiseaseInfo() {

cout << "Appellation: " << appellation << endl

<< "Treatment: " << treatment << endl

<< "Cost: " << cost << endl << endl;

}

// get cost from Disease class

float Disease::getCost() {

return cost;

}

// display all data from Doctor class

void Doctor::getDoctorInfo() {

cout << "=========== Doctor " << id + 1 << " ===========" << endl

<< "Name: " << name << endl

<< "Surname: " << surname << endl

<< "Number of patients: " << numberOfPatients << endl

<< "Profit: " << profit << endl << endl;

}

// get doctor's id

int Doctor::getId() {

return id;

}

// get Doctor's number of patients

int Doctor::getNumberOfPatients() {

return numberOfPatients;

}

// get Doctor's profit

float Doctor::getProfit() {

return profit;

}

// increase Doctor number of patients by 1

void Doctor::increaseNumberOfPatients() {

numberOfPatients += 1;

}

// Increase Doctor profit by value

void Doctor::increaseProfit(float additionalProfit) {

profit += additionalProfit;

}

## prototypes.hpp

//

// prototypes.hpp

// Lab6

//

// Created by Kushka Misha on 12/6/17.

// Copyright © 2017 Kushka Misha. All rights reserved.

//

#ifndef prototypes\_hpp

#define prototypes\_hpp

#include "stdafx.hpp"

/\*\*

\* Patient class

\*/

class Patient {

string name;

string surname;

string sex;

int age;

int id;

public:

Patient(string \_name, string \_surname, int \_age, string \_sex, int \_id) :

name(\_name), surname(\_surname),age(\_age), sex(\_sex), id(\_id) {};

void getPatientInfo();

};

/\*\*

\* Disease class

\*/

class Disease {

string appellation;

string treatment;

float cost;

public:

Disease(string \_appellation, string \_treatment, float \_cost) :

appellation(\_appellation), treatment(\_treatment), cost(\_cost) {};

void getDiseaseInfo();

float getCost();

};

/\*\*

\* Doctor class

\*/

class Doctor {

int id;

string name;

string surname;

int numberOfPatients;

float profit;

public:

Doctor(int \_id, string \_name, string \_surname, int \_numberOfPatients=0, float \_profit=0) :

id(\_id), name(\_name), surname(\_surname),numberOfPatients(\_numberOfPatients), profit(\_profit) {};

void getDoctorInfo();

int getId();

int getNumberOfPatients();

float getProfit();

void increaseNumberOfPatients();

void increaseProfit(float);

};

/\*\*

\* DoctorId class

\*/

class DoctorId {

int id;

public:

DoctorId(int \_id) : id(\_id) {};

int getDoctorId() {

return id;

}

};

/\*\*

\* Variational template class

\*/

template<class... Infos>

class Group : public Infos...

{

public:

Group(const Infos&&... infos) : Infos(infos)... {};

};

#endif /\* prototypes\_hpp \*/

## stdafx.hpp

//

// stdafx.hpp

// Lab6

//

// Created by Kushka Misha on 12/6/17.

// Copyright © 2017 Kushka Misha. All rights reserved.

//

#ifndef stdafx\_hpp

#define stdafx\_hpp

#include <iostream>

#include <string>

#include <vector>

#include <random>

using namespace std;

#endif /\* stdafx\_hpp \*/

## main.cpp

//

// main.cpp

// Lab6

//

// Created by Kushka Misha on 12/5/17.

// Copyright © 2017 Kushka Misha. All rights reserved.

//

#include "prototypes.hpp"

int main() {

Doctor doctors[] = {Doctor(0, "Nicola", "Tesla"), Doctor(1, "Franz", "Kafka")};

int num\_of\_doctors = 2;

// Data to random generate patient and it's disease

vector<vector<string>> names = {{"Adam", "m"}, {"Karl", "m"}, {"Lisa", "f"}, {"Gabbie", "f"}, {"David", "m"}};

vector<string> surnames = {"Vozniak", "Jobs", "Karter", "Duglas", "Twein"};

vector<vector<string>> disease\_treatment = {{"Caries", "Filling"}, {"Crack in the tooth", "Remove the tooth"}};

string cont = "y";

string name, surname, sex, disease, treatment, str\_n;

int age, n, id, pos, doctor\_id;

float cost;

// Init Mersenne-Twister random

random\_device rd;

mt19937 rng(rd());

while(cont == "y") {

while (true) {

try {

cout << endl << "Enter number of patients\n> ";

cin >> str\_n;

n = stoi(str\_n);

break;

} catch (...) {

cout << "Please enter a number, not string" << endl;

}

}

unique\_ptr<Group<Patient, Disease, DoctorId>> patient[n];

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

// Patient's name and sex

uniform\_int\_distribution<int> uni\_name(0, names.size() - 1);

pos = uni\_name(rng);

name = names[pos][0];

sex = names[pos][1];

// Patient's surname

uniform\_int\_distribution<int> uni\_surname(0, surnames.size() - 1);

surname = surnames[uni\_surname(rng)];

// Patient's disease and treatment

uniform\_int\_distribution<int> uni\_disease(0, disease\_treatment.size() - 1);

pos = uni\_disease(rng);

disease = disease\_treatment[pos][0];

treatment = disease\_treatment[pos][1];

// Patient's age

uniform\_int\_distribution<int> uni\_age(5, 80);

age = uni\_age(rng);

// Treatment cost

uniform\_int\_distribution<int> uni\_cost(100, 1000);

cost = uni\_cost(rng);

// Doctor's id

uniform\_int\_distribution<int> uni\_id(0, num\_of\_doctors - 1);

id = uni\_id(rng);

patient[i].reset(new Group<Patient, Disease, DoctorId> ({name, surname, age, sex, id}, {disease, treatment, cost}, {id}));

}

// Calculate profit and number of patients of every doctor

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

cost = patient[i]->getCost();

doctor\_id = patient[i]->getDoctorId();

doctors[doctor\_id].increaseNumberOfPatients();

doctors[doctor\_id].increaseProfit(cost);

}

// Display patient's and disease data

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

cout << "=========== Patient " << i + 1 << "===========" << endl;

patient[i]->getPatientInfo();

patient[i]->getDiseaseInfo();

}

// Display doctor's data

for(int i = 0; i < num\_of\_doctors; i++) {

doctors[i].getDoctorInfo();

}

// Restart program?

cout << endl << endl << "Continue? (y / n)\n> ";

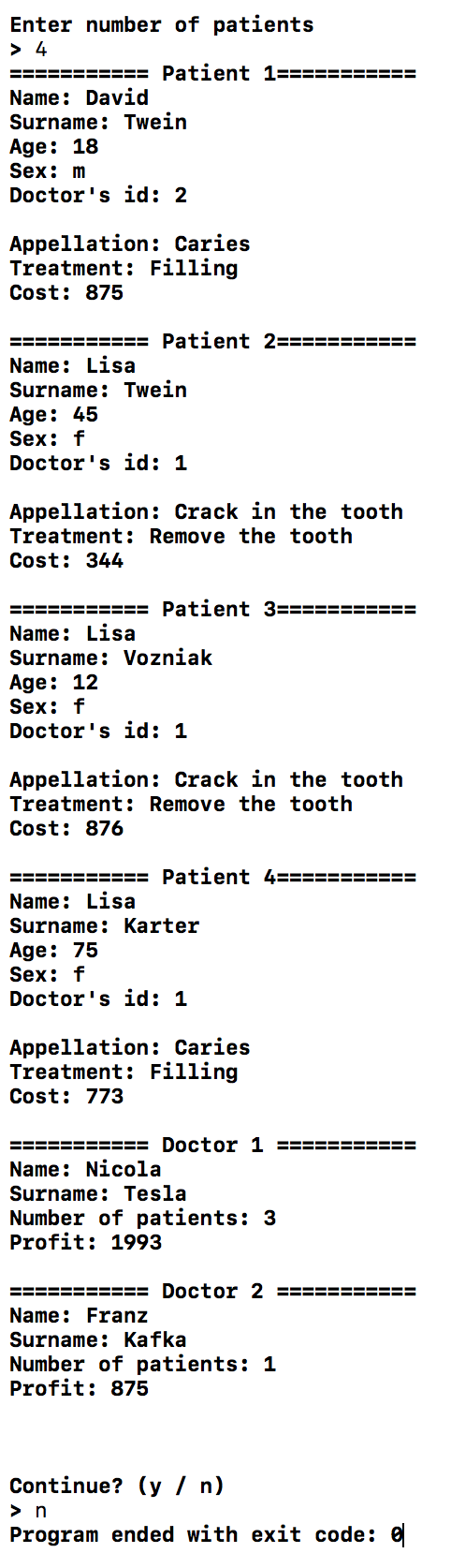
cin >> cont;

}

return 0;

}

# Приклади роботи програми



# Висновки

У даній лабораторній роботі я навчився користуватися основними контейнерами с STD бібліотеки, а саме з контейнерами з модуля random та “розумним покажчиком” unique\_ptr, що автоматично знищує після себе “сміття”, запобігаючи засмічення оперативної пам’яті.