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

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

## 2. 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.

## 3. 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

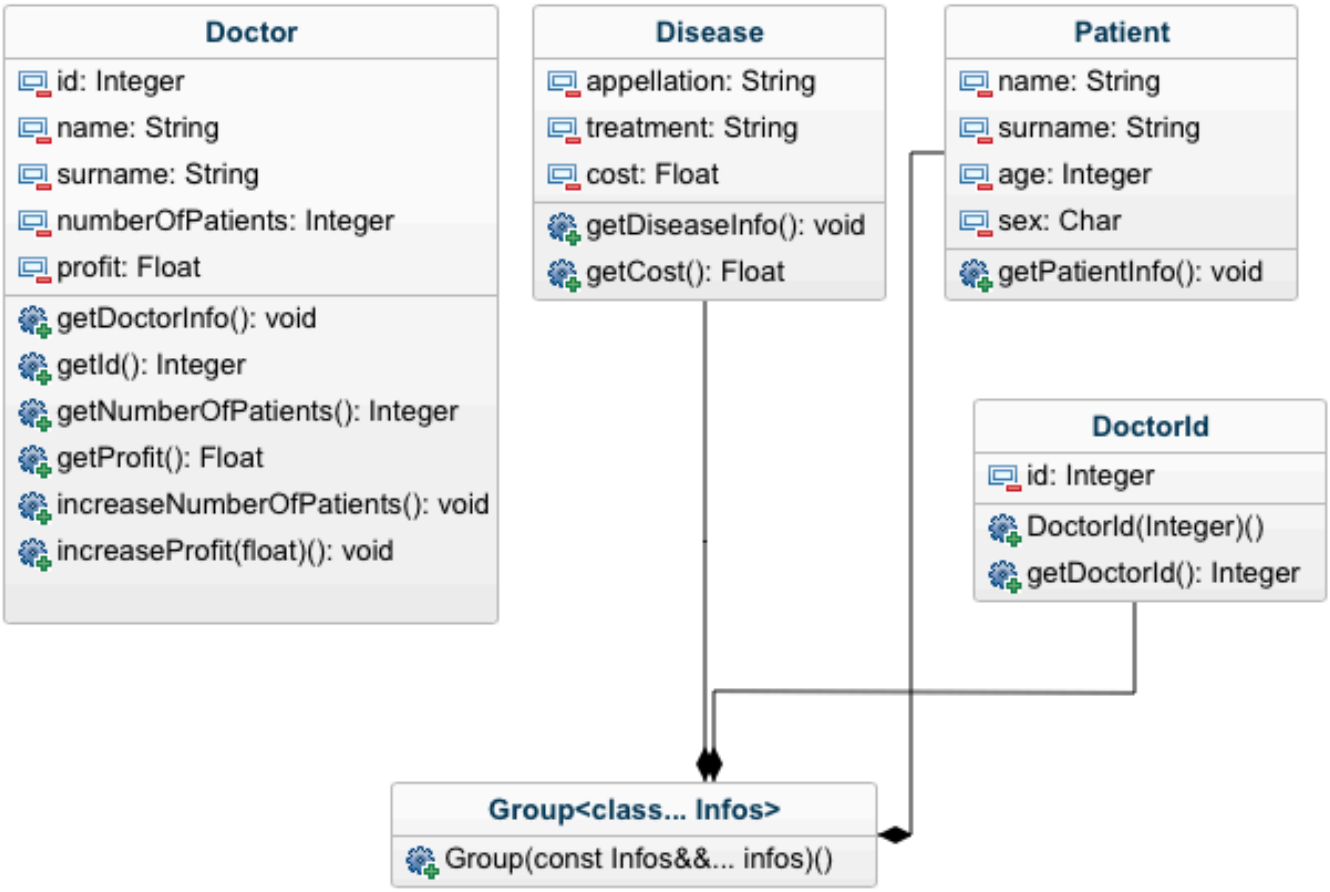
Ітератор - це аналог покажчика, за допомогою якого можна отримувати доступ до різних елементів даних. Можна використовувати і пару ітераторів для задання діапазону. Як і для покажчика, для отримання даних з ітераторів їх необхідно розіменувати за допомогою операції *.

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 -ассоціативний 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.

## 4. UML-ДІАГРАМА КЛАСІВ



## 5. 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;
}
```

# 6. ПРИКЛАДИ РОБОТИ ПРОГРАМИ

```
Enter number of patients
> 4
========== Patient 1==========
Name: David
Surname: Twein
Age: 18
Sex: m
Doctor's id: 2

Appellation: Caries
Treatment: Filling
Cost: 875

========== Patient 2==========
Name: Lisa
Surname: Twein
Age: 45
Sex: f
Doctor's id: 1

Appellation: Crack in the tooth
Treatment: Remove the tooth
Cost: 344

========== Patient 3==========
Name: Lisa
Surname: Vozniak
Age: 12
Sex: f
Doctor's id: 1

Appellation: Crack in the tooth
Treatment: Remove the tooth
Cost: 876

========== Patient 4==========
Name: Lisa
Surname: Karter
Age: 75
Sex: f
Doctor's id: 1

Appellation: Caries
Treatment: Filling
Cost: 773

========== Doctor 1 ==========
Name: Nicola
Surname: Tesla
Number of patients: 3
Profit: 1993

========== Doctor 2 ==========
Name: Franz
Surname: Kafka
Number of patients: 1
Profit: 875



Continue? (y / n)
> n
Program ended with exit code: 0
```

## 7. 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.