|  |  |
| --- | --- |
| C:\Documents and Settings\admin\Рабочий стол\МИРЭА_ЭМБЛЕМА_приказ.JPG | |
| МИНОБРАНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  «**МОСКОВСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»**  **МИРЭА** | |
| Институт информационных технологий  Кафедра математического обеспечения и стандартизации информационных технологий | |
| **Отчет по лабораторной работе №6**  **Тема работы «Создание простого класса, его реализация и применение»** | |
| Выполнил: |  |
| Студент группы ИКБО-07-17  Отчет принял:  Преподаватель кафедры МОСИТ | Акжигитов Р. Р.  Миронов Антон Николаевич |
| Москва  2018 | |

Оглавление

[Задание 1 3](#_Toc511701826)

[Условия задания 3](#_Toc511701827)

[Декомпозиция 3](#_Toc511701828)

[Определения 3](#_Toc511701829)

[Реализация функций 4](#_Toc511701830)

[Функции ввода\вывода массива\значений 5](#_Toc511701831)

[Кодирование алгоритма программы 5](#_Toc511701832)

[Student.h 5](#_Toc511701833)

[InstituteGuy.h 7](#_Toc511701834)

[main.cpp 7](#_Toc511701835)

[Вывод 9](#_Toc511701836)

# Задание 1

# Условия задания

**Класс:** Студент

**Поля:** Номер зачетной книжки, Фамилия, Имя, Дата рождения, Номер телефона, Дата зачисления, Номер группы

**Дополнительные методы:**

1. По номеру зачетной книжки вернуть фамилию и дату рождения.
2. Определить, зачислены ли два студента в одну дату.

**Класс:** Студент – инстиута

**Поля:** Название института, кафедры

**Методы:**

1. По номеру зачетной книжки вернуть номер группы, институт, кафедру.
2. Определить, учатся ли два студента в одном институте.

# Декомпозиция

1. Произвести проход по массиву студентов. Сравнивать номера зачетной книжки (введенный номер со студенческими номерами), если такой нашелся, то возвращаем фамилию и дату рождения этого студента.
2. Пройти по массиву студентов от 0-го, до N-2-го (если N – длина массива), вложенный цикл будет проходить по значению +1 и до N–1-го, таким образом, каждый студент будет сравниваться с каждым, кроме себя самого, не будет повторений (напр., 1-ый со 2-ым и 2-ой с 1-ым). Проверять даты зачисления, искать хотя бы одно соответствие.
3. Аналогично 1-ому пункту, только возвращаются группа, институт и кафедра.
4. На вход подаются 2 студента, проверить по полям их институт и сравнить.

# Определения

struct Date {

int day, month, year;

};

class Student {

public:

Student() {

name = ""; surname = "";

orderNumber = ""; phoneNumber = ""; groupNumber = "";

enterDate = {0, 0, 0}; birthDate = {0, 0, 0};

}

Student(string name, string surname = "", string orderNumber = "",

string phoneNumber = "", string groupNumber = "",

Date enterDate = {0, 0, 0}, Date birthDate = {0, 0, 0}) {

this->name = name; this->surname = surname;

this->orderNumber = orderNumber; this->phoneNumber = phoneNumber; this->groupNumber = groupNumber;

this->enterDate = enterDate; this->birthDate = birthDate;

}

string getIntro() {

return this->name + " " + this->surname;

}

string getOrderNumber() { return this->orderNumber; }

string getSurname() { return this->surname; }

string getGroupNumber() { return this->groupNumber; }

Date getBirthDate() { return this->birthDate; }

Date getEnterDate() { return this->enterDate; }

private:

string orderNumber, name, surname, phoneNumber, groupNumber;

Date enterDate, birthDate;

};

class InstituteGuy : public Student {

public:

InstituteGuy():Student() {

this->instituteName = "";

this->department = "";

}

InstituteGuy(string name, string surname = "", string orderNumber = "",

string instituteName = "", string department = "", string phoneNumber = "", string groupNumber = "",

Date enterDate = { 0, 0, 0 }, Date birthDate = { 0, 0, 0 })

:Student(name, surname, orderNumber, phoneNumber, groupNumber, enterDate, birthDate) {

this->instituteName = instituteName;

this->department = department;

};

string getInstituteName() { return this->instituteName; }

string getDepartment() { return this->department; }

private:

string instituteName, department;

};

# Реализация функций

// преобразование структуры даты в строковое представление

string dateToString(Date date, char delim = '.') {

return to\_string(date.day) + delim + to\_string(date.month) + delim + to\_string(date.year);

}

// проверка: была ли дата в объекте заполнена

bool isFilledDate(Date date) {

return date.day != 0 && date.month != 0 && date.year != 0;

}

// получение фамилии и даты рождения из массива студентов с помощью номера зачетной книжки

tuple<bool, string, Date> getStudentByOrder(vector<Student> sts, string order) {

tuple<bool, string, Date> temp;

for (auto s : sts) {

if (s.getOrderNumber() == order) {

tuple<bool, string, Date> temp(true, s.getSurname(), s.getBirthDate());

return temp;

}

}

return temp;

}

// получение студентов, которые были зачислены в одну и ту же дату

tuple<bool, Student, Student> inOneDate(vector<Student> sts) {

tuple<bool, Student, Student> temp;

for (int i = 0; i < sts.size() - 1; i++)

for (int j = i + 1; j < sts.size(); j++) {

if (dateToString(sts[i].getEnterDate()) == dateToString(sts[j].getEnterDate()) && isFilledDate(sts[i].getEnterDate())) {

tuple<bool, Student, Student> temp(true, sts[i], sts[j]);

return temp;

}

}

return temp;

}

// поиск в массиве по номеру книжки

tuple<bool, string, string, string> seekByOrder(vector<InstituteGuy> guys, string order) {

tuple<bool, string, string, string> temp;

for (auto guy : guys) {

if (guy.getOrderNumber() == order) {

tuple<bool, string, string, string> temp(true, guy.getGroupNumber(), guy.getInstituteName(), guy.getDepartment());

return temp;

}

}

return temp;

}

// учатся ли два студента института на одной кафедре

bool studyAtSameDepartment(InstituteGuy firstGuy, InstituteGuy secondGuy) {

return firstGuy.getDepartment() == secondGuy.getDepartment();

}

# Функции ввода\вывода массива\значений

void orderSearch(vector<Student> sts, string order) {

auto t = getStudentByOrder(sts, order);

cout << "Search order number: " + order << endl;

if (get<0>(t))

cout << "\tResult:\n\t\tSurname: " << get<1>(t) << "\n\t\tEnter Date: " << dateToString(get<2>(t)) << endl << endl;

else

cout << "\tStudent with that orderNumber hasn't found..." << endl << endl;

}

void beautifulSearch(vector<InstituteGuy> guys, string order) {

auto searchResult = seekByOrder(guys, order);

cout << "Search query: " + order + "\n";

if (get<0>(searchResult))

cout << "\tResult:\n\t\tGroup: " << get<1>(searchResult) << "\n\t\tInstitute: " << get<2>(searchResult) << "\n\t\tDepartment: " << get<3>(searchResult) << endl << endl;

else

cout << "\tStudent with that orderNumber hasn't found..." << endl << endl;

}

void formattedEnter(vector<Student> sts) {

auto temp = inOneDate(sts);

cout << "Did some students enter in one day?\n\t" << (get<0>(temp) ? "Yes, " + get<1>(temp).getIntro() + " and " + get<2>(temp).getIntro() + " did." : "No.") << endl << endl;

# Кодирование алгоритма программы

# Student.h

#pragma once

#include <string>

#include <tuple>

#include <vector>

using namespace std;

struct Date {

int day, month, year;

};

class Student {

public:

Student() {

name = ""; surname = "";

orderNumber = ""; phoneNumber = ""; groupNumber = "";

enterDate = {0, 0, 0}; birthDate = {0, 0, 0};

}

Student(string name, string surname = "", string orderNumber = "",

string phoneNumber = "", string groupNumber = "",

Date enterDate = {0, 0, 0}, Date birthDate = {0, 0, 0}) {

this->name = name; this->surname = surname;

this->orderNumber = orderNumber; this->phoneNumber = phoneNumber; this->groupNumber = groupNumber;

this->enterDate = enterDate; this->birthDate = birthDate;

}

string getIntro() {

return this->name + " " + this->surname;

}

string getOrderNumber() { return this->orderNumber; }

string getSurname() { return this->surname; }

string getGroupNumber() { return this->groupNumber; }

Date getBirthDate() { return this->birthDate; }

Date getEnterDate() { return this->enterDate; }

private:

string orderNumber, name, surname, phoneNumber, groupNumber;

Date enterDate, birthDate;

};

// преобразование структуры даты в строковое представление

string dateToString(Date date, char delim = '.') {

return to\_string(date.day) + delim + to\_string(date.month) + delim + to\_string(date.year);

}

// проверка: была ли дата в объекте заполнена

bool isFilledDate(Date date) {

return date.day != 0 && date.month != 0 && date.year != 0;

}

// получение фамилии и даты рождения из массива студентов с помощью номера зачетной книжки

tuple<bool, string, Date> getStudentByOrder(vector<Student> sts, string order) {

tuple<bool, string, Date> temp;

for (auto s : sts) {

if (s.getOrderNumber() == order) {

tuple<bool, string, Date> temp(true, s.getSurname(), s.getBirthDate());

return temp;

}

}

return temp;

}

// получение студентов, которые были зачислены в одну и ту же дату

tuple<bool, Student, Student> inOneDate(vector<Student> sts) {

tuple<bool, Student, Student> temp;

for (int i = 0; i < sts.size() - 1; i++)

for (int j = i + 1; j < sts.size(); j++) {

if (dateToString(sts[i].getEnterDate()) == dateToString(sts[j].getEnterDate()) && isFilledDate(sts[i].getEnterDate())) {

tuple<bool, Student, Student> temp(true, sts[i], sts[j]);

return temp;

}

}

return temp;

}

# InstituteGuy.h

#pragma once

#include "Student.h"

class InstituteGuy : public Student {

public:

InstituteGuy():Student() {

this->instituteName = "";

this->department = "";

}

InstituteGuy(string name, string surname = "", string orderNumber = "",

string instituteName = "", string department = "", string phoneNumber = "", string groupNumber = "",

Date enterDate = { 0, 0, 0 }, Date birthDate = { 0, 0, 0 })

:Student(name, surname, orderNumber, phoneNumber, groupNumber, enterDate, birthDate) {

this->instituteName = instituteName;

this->department = department;

};

string getInstituteName() { return this->instituteName; }

string getDepartment() { return this->department; }

private:

string instituteName, department;

};

// поиск в массиве по номеру книжки

tuple<bool, string, string, string> seekByOrder(vector<InstituteGuy> guys, string order) {

tuple<bool, string, string, string> temp;

for (auto guy : guys) {

if (guy.getOrderNumber() == order) {

tuple<bool, string, string, string> temp(true, guy.getGroupNumber(), guy.getInstituteName(), guy.getDepartment());

return temp;

}

}

return temp;

}

// учатся ли два студента института на одной кафедре

bool studyAtSameDepartment(InstituteGuy firstGuy, InstituteGuy secondGuy) {

return firstGuy.getDepartment() == secondGuy.getDepartment();

}

# main.cpp

#include "iostream"

#include "Student.h"

#include "InstituteGuy.h"

using namespace std;

void orderSearch(vector<Student> sts, string order) {

auto t = getStudentByOrder(sts, order);

cout << "Search order number: " + order << endl;

if (get<0>(t))

cout << "\tResult:\n\t\tSurname: " << get<1>(t) << "\n\t\tEnter Date: " << dateToString(get<2>(t)) << endl << endl;

else

cout << "\tStudent with that orderNumber hasn't found..." << endl << endl;

}

void beautifulSearch(vector<InstituteGuy> guys, string order) {

auto searchResult = seekByOrder(guys, order);

cout << "Search query: " + order + "\n";

if (get<0>(searchResult))

cout << "\tResult:\n\t\tGroup: " << get<1>(searchResult) << "\n\t\tInstitute: " << get<2>(searchResult) << "\n\t\tDepartment: " << get<3>(searchResult) << endl << endl;

else

cout << "\tStudent with that orderNumber hasn't found..." << endl << endl;

}

void formattedEnter(vector<Student> sts) {

auto temp = inOneDate(sts);

cout << "Did some students enter in one day?\n\t" << (get<0>(temp) ? "Yes, " + get<1>(temp).getIntro() + " and " + get<2>(temp).getIntro() + " did." : "No.") << endl << endl;

}

int main() {

cout << "Akzhigitov Radmir (lab. OOP n. 1) 17.04.2018" << endl << endl;

Student newOne("Petr", "Ivanov", "123234");

Student newTwo("Maria", "Sokolova", "228", "8800", "123", { 17, 12, 2020 }, { 17, 12, 2000 });

Student newThree("Semen", "Bita", "123", "234", "123", { 17, 12, 2020 });

vector<Student> sts{ newOne, newTwo, newThree };

orderSearch(sts, "300");

orderSearch(sts, "228");

formattedEnter(sts);

InstituteGuy one("Pavel", "Chistyakov", "123");

InstituteGuy two("Ivan", "Mindsmell", "228", "Cyber", "Computer Security", "8123234123", "1115");

InstituteGuy three("Sasha", "Tudar", "1337", "IT", "Software Engineer", "88005553535", "0717");

InstituteGuy four("Boris", "Godunov", "1596", "PandaHack", "Software Engineer", "213234234");

vector<InstituteGuy> guys{ one, two, three, four };

beautifulSearch(guys, "1337");

beautifulSearch(guys, "2282");

beautifulSearch(guys, "228");

cout << "Do " << two.getIntro() << " and " << three.getIntro() << " study at same department? "

<< (studyAtSameDepartment(two, three) ? "Yes, they do." : "No, they don't.") << endl;

cout << "Do " << three.getIntro() << " and " << four.getIntro() << " study at same department? "

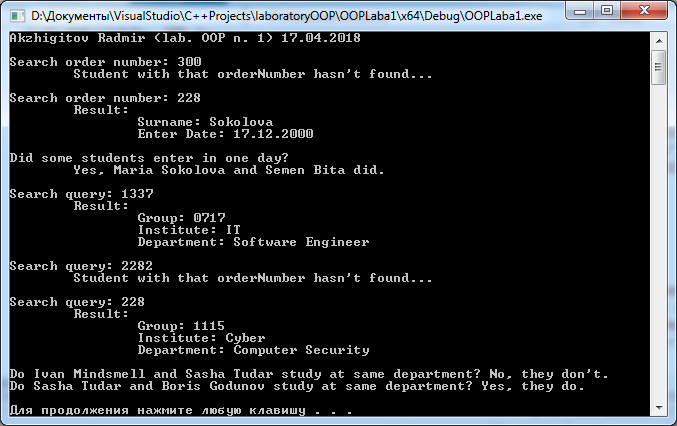
<< (studyAtSameDepartment(three, four) ? "Yes, they do." : "No, they don't.") << endl;

cout << endl;

system("pause");

return 0;

}



# Вывод

В этой работе необходимо было использовать классы и все множество понятий с ним связанных. Разработать структуру класса, определить какие поля – приватные, какие – открытые. Разработать конструкторы и методы классов, произвести операции наследования классов. Работа с некоторыми другими контейнерами, такими как: vector, tuple, String.