ФГАОУ ВО Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа киберфизических систем и управления

**ОТЧЕТ**

**по лабораторной работе №10**

**по теме «Множественное наследование»**

по дисциплине «Практикум по программированию»

Выполнил

студент гр.23533/2 А.Д. Шурак

Проверил

Ассистент В.Э. Ковалевский

Санкт-Петербург

2019

Задание

На основе лабораторной работы по работе со списками продемонстрировать множественное наследование.

Разработка программы

Для добавления работников университета создаем новый класс Worker, который содержит род деятельности и стаж работы.

class Worker

{

protected:

std::string career;

int experience;

public:

};  
  
Теперь класс Teacher наследует не только от одного, но и сразу от двух классов.

class Teacher : public People, public Worker

{

private:

Subject \*subj;

public:

bool Scan(std::ifstream &f);

std::ostream& OutData(std::ostream &f)

{

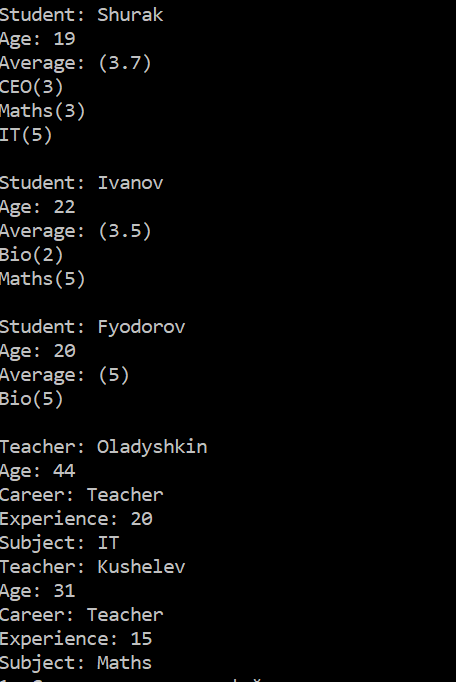
f << "Teacher: " << name << "\nAge: " << age << "\nCareer: " << career << "\nExperience: " << experience << "\nSubject: " << subj << "\n";

return f;

}

При добавлении нового преподавателя нужно ввести стаж работы, во время вывода выводятся все атрибуты.

Вся остальная программа в целом остается без изменений.



Заключение

В ходе выполнения лабораторной работы продемонстрировано множественное наследование.

**Приложение**

#include "People.h"

#include "Exams.h"

#include "Subject.h"

bool People::PScan(std::ifstream &f)

{

try

{

f >> name;

if (name.length() == 0)

return 1;

f >> age;

return 0;

}

catch (...)

{

return 1;

}

}

std::ostream& operator<<(std::ostream & f, People &data)

{

data.OutData(f);

return f;

}

bool Student::Scan(std::ifstream &f)

{

try

{

if (PScan(f))

return 1;

f >> size;

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

{

Exams nexam;

bool flag = nexam.Scan(f);

if (!flag)

{

pass.push\_back(nexam);

}

}

average();

sort(pass.begin(), pass.end());

return 0;

}

catch (...)

{

return 1;

}

}

void Student::average()

{

float result = 0;

for (auto i = pass.begin(); i < pass.end(); i++)

result += i->getMark();

avg = round((result/size) \* 10) / 10;

}

void Student::Edit()

{

char select = '0';

do {

std::cout << "1.Добавить экзамен студенту.\n2.Исправить экзамен студента.\n";

std::cin >> select;

} while ((select != '1') && (select != '2'));

std::cout << "Список экзаменов:\n";

OutPass(std::cout);

std::string name1;

std::cout << "Общий список предметов:\n";

Exams::OutSubj();

std::vector<Subject>::iterator chs = Exams::SubjList1.end();

std::cout << "Предмет, по которому сдан экзамен: ";

do

{

std::cin >> name1;

chs = find(Exams::SubjList1.begin(), Exams::SubjList1.end(), name1);

if (chs == Exams::SubjList1.end())

std::cout << "Предмет не обнаружен.\n";

} while (chs == Exams::SubjList1.end());

switch (select)

{

case '1':

{

std::vector<Exams>::iterator h = find(pass.begin(), pass.end(), name1);

if (h != pass.end())

{

std::cout << "Экзамен уже сдан.\n";

return;

}

Exams a;

a.setSubj(&(\*chs));

int mark;

std::cout << "Введите оценку: ";

std::cin >> mark;

a.setMark(mark);

pass.push\_back(a);

size++;

break;

}

case '2':

{

std::vector<Exams>::iterator check = find(pass.begin(), pass.end(), name1);

if (check != pass.end())

{

int mark;

std::cout << "Введите оценку: ";

std::cin >> mark;

check->setMark(mark);

break;

}

std::cout << "Экзамен не обнаружен.\n";

return;

}

}

average();

sort(pass.begin(), pass.end());

}

void Student::DeleteExams()

{ pass.~vector(); }

bool Teacher::Scan(std::ifstream &f)

{

try

{

if (PScan(f))

return 1;

std::string nameofcareer;

f >> (nameofcareer);

if (nameofcareer.length() == 0)

return 1;

career = nameofcareer;

f >> experience;

std::string namesubj1;

f >> namesubj1;

subj = &(\*find(Exams::SubjList1.begin(), Exams::SubjList1.end(), namesubj1));

return 0;

}

catch (...)

{

return 1;

}

}

std::ostream& Student::OutPass(std::ostream &f)

{

for (auto i = pass.begin(); i < pass.end(); i++)

f << \*i;

f << std::endl;

return f;

}

void Student::AddExams()

{

std::cout << "Введите число экзаменов: ";

std::cin >> size;

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

{

Exams nexam;

nexam.AddData();

pass.push\_back(nexam);

}

average();

sort(pass.begin(), pass.end());

}

void Teacher::Delete()

{

subj = NULL;

}

void Teacher::AddSubject()

{

Exams::OutSubj();

std::cout << "Введите предмет: ";

std::string name;

subj = NULL;

do {

std::cin >> name;

std::vector<Subject>::iterator h = find(Exams::SubjList1.begin(), Exams::SubjList1.end(), name);

if (h == Exams::SubjList1.end())

std::cout << "Предмет не обнаружен.\n";

else

subj = &(\*h);

} while (subj == NULL);

} #include <vector>

#include <iostream>

#include "Exams.h"

#include "People.h"

#include "Subject.h"

#define INSTUD "students.txt"

#define INTEACHER "teachers.txt"

#define OUT "output.txt"

#define INSUBJECT "subjects.txt"

using namespace std;

std::vector<Subject> Exams::SubjList1;

template<class T>

void From(vector<T> &a, const char\* file)

{

ifstream fin(file);

while (fin)

{

T now;

bool flag = now.Scan(fin);

if (!flag)

{

a.push\_back(now);

}

}

fin.close();

}

template<class T>

void OutList(vector<T> &a)

{

for (int i = 0; i < (a.end() - a.begin()); i++)

cout << a[i];

}

template <class T>

void ToFile(vector<T>& a)

{

ofstream fout(OUT);

for (int i = 0; i < (a.end() - a.begin()); i++)

fout << a[i];

fout.close();

}

int main()

{

char slct, c;

string name1;

setlocale(LC\_ALL, "");

From(Exams::SubjList1, INSUBJECT);

vector<Student> S;

vector<Teacher> P;

do

{

cout << "1. Считать список из файла.\n";

cout << "2. Вывести список на экран.\n";

cout << "3. Записать список в файл.\n";

cout << "4. Удалить элемент из списка.\n";

cout << "5. Добавить элемент в список.\n";

cout << "6. Изменить информацию о студенте.\n";

cout << "7. Вывести общий список.\n";

cout << "8. Удалить студентов с определённой оценкой.\n";

cout << "9. Завершить выполнение программы.\n";

cin >> slct;

switch (slct)

{

case'1':

cout << "Выберите список для использования:\n1. Студенты.\n2. Преподаватели.\n";

cin >> c;

if (c == '1')

From(S, INSTUD);

if (c == '2')

From(P, INTEACHER);

break;

case'2':

cout << "Выберите список для использования:\n1. Студенты.\n2. Преподаватели.\n";

cin >> c;

if (c == '1')

{

if (S.empty())

{

cout << "Список пустой.\n"; break;

}

OutList(S);

}

if (c == '2')

{

if (P.empty())

{

cout << "Список пустой.\n"; break;

}

OutList(P);

}

break;

case'3':

cout << "Выберите список для использования:\n1. Студенты.\n2. Преподаватели.\n";

cin >> c;

if (c == '1')

{

if (S.empty())

{

cout << "Список пустой.\n"; break;

}

ToFile(S);

}

if (c == '2')

{

if (P.empty())

{

cout << "Список пустой.\n"; break;

}

ToFile(P);

}

break;

case'5':

{

if (S.empty())

{

cout << "Список пуст.\n"; break;

}

cout << "Введите имя:\n";

cin >> name1;

vector<Student>::iterator much = find(S.begin(), S.end(), name1);

if (much == S.end())

{

cout << "Студент не обнаружен.\n"; break;

}

much->Edit();

break;

}

case'6':

{

cout << "Выберите список для использования:\n1. Студенты.\n2. Преподаватели.\n";

cin >> c;

if (c == '1')

{

if (S.empty())

{

cout << "Список пустой.\n"; break;

}

cout << "Введите имя:\n";

cin >> name1;

S.erase(remove(S.begin(), S.end(), name1), S.end());

}

if (c == '2')

{

if (P.empty())

{

cout << "Список пустой.\n"; break;

}

cout << "Введите имя:\n";

cin >> name1;

P.erase(remove(P.begin(), P.end(), name1), P.end());

}

break;

}

case'4':

{

cout << "Выберите список для использования:\n1. Студенты.\n2. Преподаватели.\n";

cin >> c;

if (c == '1')

{

cout << "Введите имя:\n";

cin >> name1;

if (find(S.begin(), S.end(), name1) == S.end())

{

Student now;

now.AddData(name1);

S.push\_back(now);

}

else

cout << "Cтудент уже есть в списке.\n";

}

if (c == '2')

{

cout << "Введите имя:\n";

cin >> name1;

if (find(P.begin(), P.end(), name1) == P.end())

{

Teacher now;

now.AddData(name1);

P.push\_back(now);

}

else

cout << "Преподаватель уже есть в списке.\n";

}

break;

}

case'7':

{

if (P.empty() && S.empty())

{

cout << "Списки пусты.\n"; break;

}

vector<People\*> H;

for (auto i = S.begin(); i < S.end(); i++)

{

H.push\_back(&\*i);

}

for (auto i = P.begin(); i < P.end(); i++)

{

H.push\_back(&\*i);

}

for (auto i = H.begin(); i < H.end(); i++)

cout << \*(\*i);

break;

}

case'8':

{

if (S.empty())

{

cout << "Список пустой.\n"; break;

}

int mark = 2;

cout << "Введите оценку:\n";

cin >> mark;

S.erase(remove(S.begin(), S.end(), mark), S.end());

break;

}

}

} while (slct != '9');

} #include "Exams.h"

bool Exams::Scan(std::ifstream &f)

{

std::string namesubj1;

f >> (namesubj1);

if (namesubj1.length() == 0)

return 1;

subj = &(\*find(SubjList1.begin(),SubjList1.end(), namesubj1));

f >> (mark);

return 0;

}

std::ostream& operator<<(std::ostream & f, Exams &data)

{

f << data.subj << "(" << data.mark << ")\n";

return f;

}

void Exams::AddData()

{

OutSubj();

std::cout << "Введите предмет: ";

std::string name;

do {

std::cin >> name;

std::vector<Subject>::iterator h = find(SubjList1.begin(), SubjList1.end(), name);

if (h == SubjList1.end())

std::cout << "Предмет не обнауржен.\n";

else

subj == &(\*h);

} while (subj == NULL);

std::cout << "Введите оценку: ";

std::cin >> mark;

} #include "Subject.h"

bool Subject::Scan(std::ifstream &f)

{

try

{

f >> (NameSubj);

if (NameSubj.length() == 0) return 1;

return 0;

}

catch (...)

{

return 1;

}

}

std::ostream& operator<<(std::ostream & f, Subject \*data)

{

f << data->NameSubj;

return f;

}

std::ostream& operator<<(std::ostream & f, Subject &data)

{

f << data.NameSubj << "\n";

return f;

}

#pragma once

#ifndef Subject\_H

#define Subject\_H

#include <iostream>

#include <fstream>

#include <string>

class Subject

{

private:

std::string NameSubj;

public:

bool operator==(std::string namesubj1) { return(NameSubj == namesubj1); }

friend std::ostream& operator<<(std::ostream & f, Subject \*data);

friend std::ostream& operator<<(std::ostream & f, Subject &data);

std::string GetNameSubj() { return NameSubj; }

bool Scan(std::ifstream &f);

};

#endif #pragma once

#ifndef People\_H

#define People\_H

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <algorithm>

#include "Exams.h"

#include "Subject.h"

class Exams;

class Subject;

class People

{

protected:

std::string name;

int age;

public:

bool operator==(std::string name1) { return(name == name1); }

bool PScan(std::ifstream &f);

virtual std::ostream& OutData(std::ostream &f)

{

f << "Name: " << name << "\nAge: " << age << "\n";

return f;

}

virtual void AddData(std::string inname)

{

name = inname;

std::cout << "Введите возраст: ";

std::cin >> age;

}

friend std::ostream& operator<<(std::ostream & f, People &data);

};

class Worker

{

protected:

std::string career;

int experience;

public:

};

class Student : public People

{

private:

int size;

float avg;

std::vector<Exams> pass;

public:

Student()

{

size = 0;

avg = 0;

}

bool operator==(std::string name1) { return(name == name1); }

bool operator==(int mark)

{

std::vector<Exams>::iterator it = find(pass.begin(), pass.end(), mark);

if ((it == pass.end()))

return false;

else

return true;

}

void DeleteExams();

std::ostream& OutData(std::ostream &f)

{

f << "Student: " << name << "\nAge: " << age << "\nAverage: (" << avg << ")\n";

OutPass(f);

return f;

}

void AddExams();

std::ostream& OutPass(std::ostream &f);

bool Scan(std::ifstream &f);

void average();

void AddData(std::string inname)

{

name = inname;

std::cout << "Введите возраст: ";

std::cin >> age;

AddExams();

sort(pass.begin(), pass.end());

}

void Edit();

};

class Teacher : public People, public Worker

{

private:

Subject \*subj;

public:

bool Scan(std::ifstream &f);

std::ostream& OutData(std::ostream &f)

{

f << "Teacher: " << name << "\nAge: " << age << "\nCareer: " << career << "\nExperience: " << experience << "\nSubject: " << subj << "\n";

return f;

}

void AddData(std::string input\_name)

{

name = input\_name;

career = "Teacher";

std::cout << "Введите возраст: ";

std::cin >> age;

std::cout << "Введите стаж: ";

std::cin >> experience;

AddSubject();

}

void Delete();

void AddSubject();

};

#endif #pragma once

#ifndef Exams\_H

#define Exams\_H

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include "Subject.h"

class Subject;

class Exams

{

private:

int mark;

Subject\* subj;

public:

static std::vector<Subject> SubjList1;

static void OutSubj()

{

for (int i = 0; i < (Exams::SubjList1.end() - Exams::SubjList1.begin()); i++)

std::cout << Exams::SubjList1[i];

}

static Subject\* FindSubject(std::string name1)

{

for (int i = 0; i < (SubjList1.end() - SubjList1.begin()); i++)

{

if (Exams::SubjList1[i] == name1)

return &Exams::SubjList1[i];

}

return NULL;

}

Exams()

{

subj = NULL;

mark = 0;

}

bool Scan(std::ifstream &f);

void AddData();

int getMark() { return mark; }

void setMark(int inname) { mark = inname; }

void setSubj(Subject \*insubj) { subj = insubj; }

bool operator!=(int mark1) { return(mark != mark1); }

bool operator==(int mark1) { return(mark == mark1); }

bool operator>(int mark1) { return(mark > mark1); }

bool operator<(int mark1) { return(mark < mark1); }

bool operator>=(int mark1) { return(mark >= mark1); }

bool operator<=(int mark1) { return(mark <= mark1); }

bool operator!=(Exams& A) { return(mark != A.getMark()); }

bool operator==(Exams& A) { return(mark == A.getMark()); }

bool operator>(Exams& A) { return(mark > A.getMark()); }

bool operator<(Exams& A) { return(mark < A.getMark()); }

bool operator>=(Exams& A) { return(mark >= A.getMark()); }

bool operator<=(Exams& A) { return(mark <= A.getMark()); }

bool operator==(std::string name1) { return(\*subj == name1); }

friend std::ostream& operator<<(std::ostream & f, Exams &data);

};

#endif