ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ

УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра Інформатики

**Звіт**

з лабораторної роботи № 3

Виконав Перевірила

ст.гр.ІТІНФ-20-1 Кириченко I. Ю.

Самченко С.О.

Харків 2021

Варіант 21

Виконати варіант завдання з лабораторної роботи 2, створивши шаблоный базовый абстрактный клас у якому реалізовані створення та операції з динамічним шаблоним масивом, об’явлені чисто віртуальні метод вводу виводу.

У класі нащадку, який свторений успадкування до заданого типу (відповідної структури), винести остальні методі. Продемонструвати роботу з конструкторами базового та успадкованого класів.

#include <iostream>

#include <string>

using namespace std;

struct Addres {

string City;

string Street;

int HouseNumber = 0;

};

ostream& operator <<(ostream& out, const Addres& adr);

istream& operator >>(istream& in, Addres& adr);

struct Department {

string DName;

string Specialization;

int CountEmp = 0;

int ActivityPr = 0;

};

ostream& operator <<(ostream& out, const Department& dep);

istream& operator >>(istream& in, Department& dep);

template <class T>

class Base {

protected:

T\* arr = NULL;

int size = 0;

public:

Base() {

arr = NULL;

size = 0;

cout << "base class created" << endl;

}

~Base() {

if (arr)

delete[] arr;

cout << "base class deleted" << endl;

}

virtual void input() = 0;

virtual void output() = 0;

};

class Company : public Base<Department> {

private:

string CompanyName;

string Activity;

int Year;

Addres Adr;

public:

Company() : Base() {

cout << "company created" << endl;

}

~Company() {

cout << "company deleted" << endl;

}

Company(const Company& comp) : Base() {

\*this = comp;

cout << "company created" << endl;

}

const Department\* getDeps() const { return arr; }

const string& getDepName(const int index) const { return arr[index].DName; }

const string& getSpec(const int index)const { return arr[index].Specialization; }

const int& getActPr(const int index)const { return arr[index].ActivityPr; }

const int& getEmpCount(const int index)const { return arr[index].CountEmp; }

const int& getNumbDep()const { return size; }

const string& getCompanyName() const { return CompanyName; }

const string& getAct() const { return Activity; }

const int& getYear()const { return Year; }

const Addres& getAdr() const { return (Adr); }

void setDeps(const Department\* deps, const int dnumb) {

delete[] arr;

arr = new Department[size = dnumb];

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

arr[i] = deps[i];

}

}

void setCompanyName(const string compname) { CompanyName = compname; }

void setAct(const string act) { Activity = act; }

void setYear(const int year) { Year = year; }

void setAdr(const string& city, const string& street, int house) { Adr.City = city; Adr.Street = street; Adr.HouseNumber = house; }

int avarageActivity()const;

bool isExistdDep(string depname)const;

bool addDep(string depname, string specialization, int empcount, int actpr);

bool addDep(const Department& dep);

bool deleteDep(string depname);

Department\* findMostActivDep()const;

int averageEmpCount()const;

void outPutDep(ostream& out)const;

Company& operator =(const Company& obj);

Company& operator +=(const Department& actor);

bool operator <(const Company& arg2) const;

operator int()const;

friend ostream& operator <<(ostream& out, const Company& comp);

friend istream& operator >>(istream& in, Company& accomptor);

virtual void input() {

cin >> \*this;

cout << "data was readed" << endl;

}

virtual void output() {

cout << \*this;

cout << "data was printed" << endl;

}

};

int main() {

setlocale(LC\_ALL, "ru");

Company company;

company.input();

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

cout << "Add new department: " << endl;

Department newDep;

cin >> newDep;

company += newDep;

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

cout << "Information about the company:\n";

cout << company;

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

cout << endl << endl;

cout << "Avarage count of workers: " << company.averageEmpCount() << endl;

cout << "Average workload in the company: " << company.avarageActivity() << endl;

Department\* MostActivDep = company.findMostActivDep();

if (MostActivDep) cout << "The most loaded department:\n" << \*MostActivDep << endl;

string DepName;

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

cout << "Enter the name of the department you want to delete:\n";

getline(cin, DepName);

bool result = company.deleteDep(DepName);

if (!result) cout << "There is no such department " << endl;

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

cout << "Residual information about the company:\n";

company.output();

system("pause");

return 0;

}

ostream& operator <<(ostream& out, const Department& dep)

{

out << "\nDepartment:\t" << dep.DName

<< "\nSpecialization:\t" << dep.Specialization

<< "\nCount of workers =\t" << dep.CountEmp

<< "\nWorkload in the company =\t" << dep.ActivityPr;

return out;

}

istream& operator >>(istream& in, Department& dep)

{

cout << " \* Input department name: ";

getline(in, dep.DName);

cout << " \* Activity: ";

getline(in, dep.Specialization);

cout << " \* Count workers: ";

string strTmp;

getline(in, strTmp);

dep.CountEmp = atoi(strTmp.c\_str());

cout << " \* Activity percent: ";

getline(in, strTmp);

dep.ActivityPr = atoi(strTmp.c\_str());

return in;

}

ostream& operator <<(ostream& out, const Addres& adr)

{

out << " City " << adr.City

<< " Street " << adr.Street

<< " House № " << adr.HouseNumber;

return out;

}

istream& operator >>(istream& in, Addres& adr) {

cout << " \* Input city name: ";

getline(in, adr.City);

cout << " \* input street name: ";

getline(in, adr.Street);

cout << " \* input house: ";

string strTmp;

getline(in, strTmp);

adr.HouseNumber = atoi(strTmp.c\_str());

return in;

}

/\*-----------------------------------------------------------------------------------------------------\*/

void Company::outPutDep(ostream& out) const { for (int i = 0; i < size; ++i) out << arr[i] << endl; };

int Company::avarageActivity() const {

int result = 0;

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

result += (arr[i].ActivityPr);

result /= size;

return result;

}

/\*|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|\*/

bool Company::isExistdDep(string depname)const {

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

if (arr[i].DName == depname) {

return true;

}

}

return false;

}

/\*|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|\*/

bool Company::addDep(string depname, string specialization, int empcount, int actpr) {

Department newDep = { depname, specialization, empcount, actpr };

return addDep(newDep);

}

/\*|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|\*/

bool Company::addDep(const Department& dep) {

if (isExistdDep(dep.DName)) return false;

Department\* mas = new Department[size + 1];

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

mas[i] = arr[i];

mas[size] = dep;

delete[]arr;

arr = mas;

++size;

return true;

}

/\*|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|\*/

bool Company::deleteDep(string depname) {

if (!isExistdDep(depname)) return false;

Department\* mas = new Department[size - 1];

int j = 0;

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

{

if (arr[i].DName == depname)

{

continue;

}

mas[j++] = arr[i];

}

delete[]arr;

arr = mas;

--size;

}

/\*|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|\*/

Department\* Company::findMostActivDep()const {

if (size <= 0) return NULL;

Department\* MostActivDep = arr;

for (int i = 1; i < size; ++i)

{

if (arr[i].ActivityPr > MostActivDep->ActivityPr) MostActivDep = &arr[i];

}

return MostActivDep;

}

/\*|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|\*/

int Company::averageEmpCount()const {

int result = 0;

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

result += (arr[i].CountEmp);

result /= size;

return result;

}

/\*|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|\*/

Company& Company::operator =(const Company& obj) {

if (this == &obj) return (\*this);

this->~Company();

CompanyName = obj.CompanyName;

Activity = obj.Activity;

Year = obj.Year;

Adr = obj.Adr;

size = obj.size;

arr = new Department[size];

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

{

arr[i] = obj.arr[i];

}

}

/\*|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|\*/

Company& Company::operator +=(const Department& dep) {

addDep(dep);

return(\*this);

}

/\*|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|\*/

bool Company::operator < (const Company& com2) const {

return (this->Year < com2.Year);

}

/\*|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|\*/

Company::operator int()const {

int result = avarageActivity();

return result;

}

/\*|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|\*/

ostream& operator <<(ostream& out, const Company& comp) {

out << comp.getCompanyName()

<< "\n - kind of activity: " << comp.getAct()

<< "\n - year of foundation: " << comp.getYear()

<< "\n - address: " << comp.getAdr() << endl;

comp.outPutDep(out);

return out;

}

/\*|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|\*/

istream& operator >>(istream& in, Company& comp) {

string strTmp, strTmp1, strTmp2;

cout << "Input company name: ";

getline(in, strTmp);

comp.setCompanyName(strTmp);

cout << "enter the type of activity: ";

getline(in, strTmp);

comp.setAct(strTmp);

cout << "enter the year of foundation of the company: ";

getline(in, strTmp);

comp.setYear(atoi(strTmp.c\_str()));

cout << endl;

cout << "enter address:" << endl;

cout << " \* city ";

getline(in, strTmp);

cout << " \* street ";

getline(in, strTmp1);

cout << " \* house # ";

getline(in, strTmp2);

comp.setAdr(strTmp, strTmp1, atoi(strTmp2.c\_str()));

string ansver;

do {

cout << endl;

cout << "enter data about departments: " << endl;

Department dep;

in >> dep;

comp += dep;

cout << "want to add another department? (y/n):";

getline(in, ansver);

} while (ansver.size() != 0 && ansver[0] == 'y');

return in;

}

