#include<iostream>

using namespace std;

class Persoana {

char\* nume;

protected:

int varsta;

public:

Persoana() {

this->nume = new char[strlen("numeDefault") + 1];

strcpy\_s(this->nume, strlen("numeDefault") + 1, "numeDefault");

this->varsta = 0;

}

Persoana(const char\* nume, int varsta) {

if (nume == nullptr) {

throw new exception("Nu poti seta nullptr pe char\* nume.");

}

if (varsta < 6) {

throw new exception("Persoana trebuia sa fie la gradinita.");

}

this->nume = new char[strlen(nume) + 1];

strcpy\_s(this->nume, strlen(nume) + 1, nume);

this->varsta = varsta;

}

Persoana(const Persoana& p) {

this->nume = new char[strlen(p.nume) + 1];

strcpy\_s(this->nume, strlen(p.nume) + 1, p.nume);

this->varsta = p.varsta;

}

Persoana& operator=(const Persoana& p) {

if (this != &p) {

delete[] this->nume;

this->nume = new char[strlen(p.nume) + 1];

strcpy\_s(this->nume, strlen(p.nume) + 1, p.nume);

this->varsta = p.varsta;

return \*this;

}

}

~Persoana() {

delete[]this->nume;

}

char\* getNume(char\* nume) {

return this->nume;

}

void setNume(char\* nume) {

if (nume == nullptr) {

throw exception("Numele nu poate avea dimensiunea 0.");

}

else {

delete[] this->nume;

this->nume = new char[strlen(nume) + 1];

strcpy\_s(this->nume, strlen(nume) + 1, nume);

}

}

int getVarsta(int varsta) {

return this->varsta;

}

void setVarsta(int varsta) {

if (varsta>80) {

throw exception("Varsta trebuie sa fie mai mica decat 80.");

}

this->varsta = varsta;

}

friend ostream& operator<<(ostream& out, const Persoana& p);

friend istream& operator>>(istream& in, Persoana& p);

};

ostream& operator<<(ostream& out, const Persoana& p) {

out << "Nume: " << p.nume << ", varsta: " << p.varsta << endl << endl;

return out;

}

istream& operator>>(istream& in, Persoana& p) {

char aux[100];

cout << "Nume: "; in >> aux;

delete[] p.nume;

p.nume = new char[strlen(aux) + 1];

strcpy\_s(p.nume, strlen(aux) + 1, aux);

cout << "Varsta: "; in >> p.varsta;

return in;

}

class Angajat : public Persoana {

private:

string nume\_manager;

const int id\_angajat;

const double salariuMinimAcceptabil;

protected:

double salariu;

public:

Angajat() : Persoana(), id\_angajat(0), salariuMinimAcceptabil(0.0){

this->nume\_manager = "numeManagerDefault";

this->salariu = 0.0;

}

Angajat(const char\* nume, int varsta, string nume\_manager, double salariu, int id\_angajat, double salariuMinimAcceptabil) : Persoana(nume, varsta), id\_angajat(id\_angajat), salariuMinimAcceptabil(salariuMinimAcceptabil) {

if (nume\_manager.empty()) {

throw exception("Numele nu poate avea dimensiunea 0.");

}

if (salariu == 0.0) {

throw exception("Salariul nu poate fi inexistent.");

}

this->nume\_manager = nume\_manager;

this->salariu = salariu;

}

Angajat(const Angajat& a) : Persoana(a), id\_angajat(id\_angajat), salariuMinimAcceptabil(salariuMinimAcceptabil) {

this->nume\_manager = a.nume\_manager;

this->salariu = a.salariu;

}

Angajat& operator=(const Angajat& a) {

if (this != &a) {

Persoana::operator=(a);

this->nume\_manager = a.nume\_manager;

this->salariu = a.salariu;

}

return\*this;

}

string getNume\_manager() {

return this->nume\_manager;

}

void setNume\_manager(string nume\_manager) {

if (nume\_manager.empty()) {

throw exception("Numele nu poate avea dimensiunea 0.");

}

this->nume\_manager = nume\_manager;

}

double getSalariu() {

return this->salariu;

}

void setSalariu() {

if (salariu == 0.0) {

throw exception("Salariul nu poate fi inexistent.");

}

this->salariu = salariu;

}

friend ostream& operator<<(ostream& out, const Angajat& a);

friend istream& operator>>(istream& in, Angajat& a);

friend void metodaPrietena();

friend class Departament;

};

ostream& operator<<(ostream& out, const Angajat& a) {

out << (Persoana)a;

out << "Nume manager: " << a.nume\_manager<<", "

<< "Salariul este: " << a.salariu<<", "

<< "Id angajat: " << a.id\_angajat<<", "

<< "Salariu minim acceptabil: " << a.salariuMinimAcceptabil << endl;

return out;

}

istream& operator>>(istream& in, Angajat& a) {

in >> (Persoana&)a;

in.ignore();

char aux[100];

cout << "Nume manager: "; in.getline(aux, 100);

a.nume\_manager = aux;

cout << "Salariul:";in >> a.salariu;

return in;

}

void metodaPrietena() {

Angajat a;

cout << "Pentru ca Angajat s-a imprietenit cu functia GLOBALA void metodaPrietena(); => ca avem acces la zona privata a clasei. Putem accesa nume\_manager si id\_angajat(privati): " << a.nume\_manager << " " << a.id\_angajat;

}

enum TIP {

fullstack = 2,

backend,

frontend

};

class Programator : public Angajat {

private:

char\* limbajProgramareFolosit;

TIP tip;

public:

Programator() : Angajat() {

this->limbajProgramareFolosit = new char[strlen("limbajProgramareFolositDefault") + 1];

strcpy\_s(this->limbajProgramareFolosit, strlen("limbajProgramareFolositDefault") + 1, "limbajProgramareFolositDefault");

this->tip = fullstack;

}

Programator(const char\* nume, int varsta, string nume\_manager, double salariu, int id\_angajat, double salariuMinimAcceptabil, const char\* limbajProgramareFolosit, TIP tip) :Angajat(nume, varsta, nume\_manager, salariu, id\_angajat, salariuMinimAcceptabil) {

this->limbajProgramareFolosit = new char[strlen(limbajProgramareFolosit) + 1];

strcpy\_s(this->limbajProgramareFolosit, strlen(limbajProgramareFolosit) + 1, limbajProgramareFolosit);

this->tip = tip;

}

Programator(const Programator& pr) :Angajat(pr) {

this->limbajProgramareFolosit = new char[strlen(pr.limbajProgramareFolosit) + 1];

strcpy\_s(this->limbajProgramareFolosit, strlen(pr.limbajProgramareFolosit) + 1, pr.limbajProgramareFolosit);

this->tip = pr.tip;

}

Programator& operator=(const Programator& pr) {

if (this != &pr) {

Angajat::operator=(pr);

this->limbajProgramareFolosit = new char[strlen(pr.limbajProgramareFolosit) + 1];

strcpy\_s(this->limbajProgramareFolosit, strlen(pr.limbajProgramareFolosit) + 1, pr.limbajProgramareFolosit);

this->tip = pr.tip;

}

return \*this;

}

~Programator() {

delete[]this->limbajProgramareFolosit;

}

char\* getLimbajProgramareFolosit(char\* limbajProgramareFolosit) {

return this->limbajProgramareFolosit;

}

void setLimbajProgramareFolosit(char\* limbajProgramareFolosit) {

this->limbajProgramareFolosit = new char[strlen(limbajProgramareFolosit) + 1];

strcpy\_s(this->limbajProgramareFolosit, strlen(limbajProgramareFolosit) + 1, limbajProgramareFolosit);

}

TIP getTip() {

return this->tip;

}

void setTip(TIP tip) {

this->tip = tip;

}

string tipToString(TIP tip) const {

switch (tip)

{

case fullstack:

return "Fullstack";

case backend:

return "Backend";

case frontend:

return "Frontend";

}

}

TIP stringToTIP(string value) {

if (value == "Fullstack") {

return fullstack;

}

else if (value == "Backend") {

return backend;

}

else {

return frontend;

}

}

friend ostream& operator<<(ostream& out, const Programator& pr);

friend istream& operator>>(istream& in, Programator& pr);

};

ostream& operator<<(ostream& out, const Programator& pr) {

out << (Angajat)pr;

out << endl << "Limbajul de programare folosit este: " << pr.limbajProgramareFolosit << endl;

out << "Tip programare: " << pr.tipToString(pr.tip) << endl;

return out;

}

istream& operator>>(istream& in, Programator& pr) {

in >> (Angajat&)pr;

char aux[100];

cout << "Limbajul de programare folosit: "; in >> aux;

delete[] pr.limbajProgramareFolosit;

pr.limbajProgramareFolosit = new char[strlen(aux) + 1];

strcpy\_s(pr.limbajProgramareFolosit, strlen(aux) + 1, aux);

cout << "Tip programare: "; in.getline(aux, 100);

pr.tip = pr.stringToTIP(aux);

return in;

}

class Departament {

char\* denumire;

Angajat\* angajati;

int nr\_angajati;

public:

static int angajatiToateDepartamentele;

Departament() {

this->denumire = new char[strlen("denumireDefault") + 1];

strcpy\_s(this->denumire, strlen("denumireDefault") + 1, "denumireDefault");

this->angajati = nullptr;

this->nr\_angajati = 0;

}

Departament(const char\* denumire) {

this->denumire = new char[strlen(denumire) + 1];

strcpy\_s(this->denumire, strlen(denumire) + 1, denumire);

}

Departament(const char\* denumire, Angajat\* angajati, int nr\_angajati) {

this->denumire = new char[strlen(denumire) + 1];

strcpy\_s(this->denumire, strlen(denumire) + 1, denumire);

this->nr\_angajati = nr\_angajati;

this->angajati = new Angajat[nr\_angajati];

for (int i = 0; i < nr\_angajati; i++) {

this->angajati[i] = angajati[i];

}

}

Departament(const Departament& d) {

this->denumire = new char[strlen(d.denumire) + 1];

strcpy\_s(this->denumire, strlen(d.denumire) + 1, d.denumire);

this->nr\_angajati = d.nr\_angajati;

this->angajati = new Angajat[d.nr\_angajati];

for (int i = 0; i < d.nr\_angajati; i++) {

this->angajati[i] = d.angajati[i];

}

}

Departament& operator=(const Departament& d) {

if (this != &d) {

delete[]this->denumire;

this->denumire = new char[strlen(d.denumire) + 1];

strcpy\_s(this->denumire, strlen(d.denumire) + 1, d.denumire);

this->nr\_angajati = d.nr\_angajati;

delete[]this->angajati;

this->angajati = new Angajat[d.nr\_angajati];

for (int i = 0; i < d.nr\_angajati; i++) {

this->angajati[i] = d.angajati[i];

}

}

return \*this;

}

char\* getDenumire(char\* denumire) {

return this->denumire;

}

void setDenumire(char\* denumire) {

if (denumire == nullptr) {

throw exception("Denumirea nu poate avea dimensiunea 0.");

}

else {

delete[] this->denumire;

this->denumire = new char[strlen(denumire) + 1];

strcpy\_s(this->denumire, strlen(denumire) + 1, denumire);

}

}

int getNr\_angajati() {

return this->nr\_angajati;

}

void setNr\_angajati(int nr\_angajati) {

this->nr\_angajati = nr\_angajati;

}

Angajat\* getAngajati() {

return this->angajati;

}

void setAngajati(Angajat\* angajati, int nr\_angajati) {

if (angajati == nullptr) {

throw exception("Nu poti seta nullptr pe vectorul de angajati.");

}

else {

delete[] this->angajati;

this->nr\_angajati = nr\_angajati;

this->angajati = new Angajat[nr\_angajati];

for (int i = 0; i < nr\_angajati; i++) {

this->angajati[i] = angajati[i];

}

}

}

float salariuTotal() const {

float totalSalariu = 0.0;

for (int i = 0; i < nr\_angajati; i++) {

totalSalariu += angajati[i].getSalariu();

}

return totalSalariu;

}

~Departament() {

delete[] this->denumire;

delete[] this->angajati;

}

friend ostream& operator<<(ostream& out, const Departament& d);

friend istream& operator>>(istream& in, Departament& d);

};

ostream& operator<<(ostream& out, const Departament& d) {

out << "-------\n";

out << "Denumirea departamentului este :" << d.denumire << endl;

out << "Numar angajati :" << d.nr\_angajati << endl;

out << "\nAngajati:\n";

for (int i = 0; i <d.nr\_angajati; i++) {

out << d.angajati[i];

}

out << "-------";

return out;

}

istream& operator>>(istream& in, Departament& d) {

char aux[100];

cout << "Denumirea departamentului este : "; in >> aux;

delete[] d.denumire;

d.denumire = new char[strlen(aux) + 1];

strcpy\_s(d.denumire, strlen(aux) + 1, aux);

cout << "Numar angajati: "; in >> d.nr\_angajati;

in.ignore();

cout << "Angajati:\n";

delete[] d.angajati;

d.angajati = new Angajat[d.nr\_angajati];

for (int i = 0; i < d.nr\_angajati; i++) {

cout << "Angajat[" << i << "]\n"; in >> d.angajati[i];

}

return in;

}

void main() {

//Persoana persoana1;

//cin >> persoana1;

Persoana p1("Ruxandra", 20);

cout<<p1;

try {

p1.setVarsta(88);

}

catch (exception e) {

cout << e.what()<<endl;

}

try {

p1.setNume(nullptr);

}

catch (exception e) {

cout << e.what() << endl;

};

//Angajat angajatul1;

//cin >> angajatul1;

Angajat a1( "Alexandra", 34, "manager1", 207.89, 87, 163.45);

cout << a1;

//Programator p1;

//cin >> p1;

Programator programatorul1("Denisa", 34, "manager2", 674.34, 943, 236.56, "c", backend);

cout << programatorul1;

Programator p2("Dani", 23, "manager3", 562.67, 898, 186.76, "c++", backend);

cout<<p2;

Angajat\* angajatPointer = new Angajat[2];

angajatPointer[0] = programatorul1;

angajatPointer[1] = p2;

Departament departamentul1("Departament", angajatPointer, 2);

cout << departamentul1;

delete[] angajatPointer;

cout << departamentul1 << endl;

cin >> departamentul1; cout << endl;

cout << departamentul1 << endl;

cout << "Salariul total al angajatilor din departament: " << departamentul1.salariuTotal() << endl;

}