#include <iostream>

#include <vector>

#include <list>

#include <fstream>

#include <map>

using namespace std;

class Produs {

protected:

// Variabile

string nume\_produs;

float pret;

int cantitate;

public:

// Getters

string getNume\_produs() {

return nume\_produs;

}

float getPret() {

return pret;

}

int getCantitate() {

return cantitate;

}

// Setters

void setNume\_produs(string nume\_p) {

nume\_produs = nume\_p;

}

void setPret(float pret\_p) {

pret = pret\_p;

}

void setCantitate(int cant\_p) {

cantitate = cant\_p;

}

// Constructorul fara parametrii

Produs() {

this->nume\_produs = "Fara nume";

this->pret = 0;

this->cantitate = 0;

}

// Constructorul de copiere

Produs(const Produs& p) {

this->nume\_produs = p.nume\_produs;

this->pret = p.pret;

this->cantitate = p.cantitate;

}

// Constructorul cu toti parametrii

Produs(string nume\_produs, float pret, int cantitate) {

this->nume\_produs = nume\_produs;

this->pret = pret;

this->cantitate = cantitate;

}

// Forma supraincarcata a operatorului =

Produs& operator=(const Produs& p) {

if (this != &p) {

this->nume\_produs = p.nume\_produs;

this->pret = p.pret;

this->cantitate = p.cantitate;

}

return \*this;

}

// Supraincarcarea operatorului >>

friend istream& operator>>(istream& in, Produs& p) {

// Nu citesc de la tastatura nume\_medicament pentru ca am scris o functie mai jos care

// ia numele medicamentelor din lista\_medicamente dintr-o variabila de tip Stocuri

// si le pune intr-un vector de tip Medicament.

// Practic in vectorul de tip Medicament nu se pot afla nume de medicamente care nu exista

// in lista de medicamente din Stocuri.

cout << "\nProdusul " << p.nume\_produs << " are pretul: "; in >> p.pret;

cout << "\nCantitate Produs: "; in >> p.cantitate;

return in;

}

// Supraincarcarea operatorului <<

friend ostream& operator<<(ostream& out, const Produs& p) {

out << "\nNume produs: " << p.nume\_produs;

out << "\nPret produs: " << p.pret;

out << "\nCantitate ramasa in stoc: " << p.cantitate;

return out;

}

// Operatorul ==

bool operator==(const Produs& p) {

if (this->pret == p.pret) return true;

return false;

}

// Operatorul +

Produs operator+(float p) {

Produs aux(\*this);

aux.pret = aux.pret + p;

return aux;

}

// Operatorul -

Produs operator-(float p) {

Produs aux(\*this);

aux.pret = aux.pret - p;

return aux;

}

// Destructorul

~Produs() {}

void afisareProdus() {

cout << "\nNume produs: " << nume\_produs;

cout << "\nPret produs: " << pret;

cout << "\nCantitate ramasa in stoc: " << cantitate;

}

void editareProdus() {

cout << "\nProdus " << nume\_produs << " are pretul: "; cin >> pret;

cout << "\nCantitate produs: "; cin >> cantitate;

}

};

//Interfata

class Discount {

public:

virtual void afisareInfoClient() {

cout << "\nTip client: Tip nespecificat";

}

virtual float pretFinal() = 0;

};

class Client : public Discount {

protected:

// Variabile

char\* nume\_client;

const int id\_client;

float\* pretCumparaturi;

int nr\_produse;

string\* cumparaturi;

public:

static int nr\_client;

// Getters

char\* getNume\_client() {

return this->nume\_client;

}

float\* getPretCumparaturi() {

return this->pretCumparaturi;

}

string\* getCumparaturi() {

return this->cumparaturi;

}

int getNr\_produse() {

return nr\_produse;

}

const int getIdClient() {

return id\_client;

}

// Setters

void setNume\_client(char\* nume\_client) {

if (this->nume\_client != NULL)

delete[] this->nume\_client;

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

strcpy(this->nume\_client, nume\_client);

}

void setPretCumparaturi(float\* c, int n) {

this->nr\_produse = n;

if (this->pretCumparaturi != NULL)

delete[] this->pretCumparaturi;

this->pretCumparaturi = new float[n];

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

this->pretCumparaturi[i] = c[i];

}

void setCumparaturi(string c, int n) {

this->nr\_produse = n;

if (this->cumparaturi != NULL)

delete[] this->cumparaturi;

this->cumparaturi = new string[n];

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

this->cumparaturi[i] = c;

}

// Constructorul fara parametrii

Client() :id\_client(nr\_client++) {

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

strcpy(this->nume\_client, "Fara nume");

this->pretCumparaturi = NULL;

this->nr\_produse = 0;

this->cumparaturi = NULL;

}

// Constructorul de copiere

Client(const Client& c) :id\_client(c.id\_client) {

this->nr\_produse = c.nr\_produse;

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

strcpy(this->nume\_client, c.nume\_client);

this->pretCumparaturi = new float[c.nr\_produse];

for (int i = 0; i < c.nr\_produse; i++)

this->pretCumparaturi[i] = c.pretCumparaturi[i];

this->cumparaturi = new string[c.nr\_produse];

for (int i = 0; i < c.nr\_produse; i++)

this->cumparaturi[i] = c.cumparaturi[i];

}

// Constructorul cu toti parametrii

Client(char\* nume\_client, int nr\_produse, float\* pretCumparaturi, string\* cumparaturi) :id\_client(nr\_client++) {

this->nr\_produse = nr\_produse;

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

strcpy(this->nume\_client, nume\_client);

this->pretCumparaturi = new float[nr\_produse];

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

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

this->cumparaturi = new string[nr\_produse];

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

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

}

// Forma supraincarcata a operatorului =

Client& operator=(const Client& c) {

if (this != &c) {

if (this->nume\_client != NULL)

delete[] this->nume\_client;

if (this->pretCumparaturi != NULL)

delete[] this->pretCumparaturi;

if (this->cumparaturi != NULL)

delete[] this->cumparaturi;

this->nr\_produse = c.nr\_produse;

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

strcpy(this->nume\_client, c.nume\_client);

this->pretCumparaturi = new float[c.nr\_produse];

for (int i = 0; i < c.nr\_produse; i++)

this->pretCumparaturi[i] = c.pretCumparaturi[i];

this->cumparaturi = new string[c.nr\_produse];

for (int i = 0; i < c.nr\_produse; i++)

this->cumparaturi[i] = c.cumparaturi[i];

}

return \*this;

}

// Supraincarcarea operatorului >>

friend istream& operator>>(istream& in, Client& c) {

cout << "\nNume: ";

char n[100];

in >> n;

if (c.nume\_client != NULL)

delete[] c.nume\_client;

c.nume\_client = new char[strlen(n) + 1];

strcpy(c.nume\_client, n);

cout << "\nCate produse ati cumparat? "; in >> c.nr\_produse;

cout << "\nNume produse: ";

if (c.cumparaturi != NULL)

delete[] c.cumparaturi;

c.cumparaturi = new string[c.nr\_produse];

for (int i = 0; i < c.nr\_produse; i++)

in >> c.cumparaturi[i];

return in;

}

// Supraincarcarea operatorului <<

friend ostream& operator<<(ostream& out, const Client& c) {

out << "\nID client: " << c.id\_client;

out << "\nNume client: " << c.nume\_client;

out << "\nNumar de produse cumparate: " << c.nr\_produse;

out << "\nProduse: ";

for (int i = 0; i < c.nr\_produse; i++)

out << c.cumparaturi[i] << " ";

//out<<"\nPreturi: ";

//for(int i=0; i<c.nr\_produse; i++)

//out<<c.pretCumparaturi[i]<<" ";

return out;

}

// Operatorul []

string operator[](int i) {

if (i >= 0 && i < this->nr\_produse) return this->cumparaturi[i];

else cout << "\nIndex gresit";

}

// Destructorul

~Client() {

if (this->nume\_client != NULL)

delete[] this->nume\_client;

if (this->pretCumparaturi != NULL)

delete[] this->pretCumparaturi;

if (this->cumparaturi != NULL)

delete[] this->cumparaturi;

}

float pretInitial() {

float t = 0;

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

t = t + pretCumparaturi[i];

return t;

}

void afisareCumparaturi() {

cout << "\nCumparaturi: ";

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

cout << cumparaturi[i] << " ";

}

void stergeProdus(string produs, Client& cl) {

string\* aux;

int ok = 0, k = 0;

aux = new string[nr\_produse - 1];

for (int i = 0; i < cl.nr\_produse; i++) {

if (cl.cumparaturi[i] == produs && ok == 0) ok = 1;

else aux[k++] = cl.cumparaturi[i];

}

cl.nr\_produse--;

if (cl.cumparaturi != NULL)

delete[] cl.cumparaturi;

cl.cumparaturi = new string[cl.nr\_produse];

for (int i = 0; i < cl.nr\_produse; i++) {

cl.cumparaturi[i] = aux[i];

}

}

};

class PersoanaFizica :public Client {

protected:

bool card\_fidelitate;

public:

bool getCard\_fidelitate() {

return card\_fidelitate;

}

void setCard\_fidelitate(bool cardF) {

card\_fidelitate = cardF;

}

PersoanaFizica() :Client() {

this->card\_fidelitate = false;

}

PersoanaFizica(const PersoanaFizica& pf) :Client(pf) {

this->card\_fidelitate = pf.card\_fidelitate;

}

PersoanaFizica(char\* nume\_client, int nr\_produse, float\* pretCumparaturi, string\* cumparaturi, bool card\_fidelitate) :Client(nume\_client, nr\_produse, pretCumparaturi, cumparaturi) {

this->card\_fidelitate = card\_fidelitate;

}

PersoanaFizica& operator=(const PersoanaFizica& pf) {

if (this != &pf) {

Client::operator=(pf);

this->card\_fidelitate = pf.card\_fidelitate;

}

return \*this;

}

friend ostream& operator<<(ostream& out, const PersoanaFizica& pf) {

out << (Client&)pf;

out << "\nCard de fidelitate: ";

if (pf.card\_fidelitate) out << "da";

else out << "nu";

}

friend istream& operator>>(istream& in, PersoanaFizica& pf) {

in >> (Client&)pf;

cout << "\nCarnet de fidelitate (0-nu/1-da): ";

in >> pf.card\_fidelitate;

return in;

}

~PersoanaFizica() {

}

void afisareInfoClient() {

cout << "\nTip client: Persoana fizica";

}

float pretFinal() {

float t = 0;

int n;

float\* pc;

n = this->getNr\_produse();

pc = this->getPretCumparaturi();

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

t = t + pc[i];

if (card\_fidelitate == 1) t = t - t / 10; //Reducere 10%;

return t;

}

};

class PersoanaJuridica :public Client {

protected:

string nume\_firma;

string nr\_telefon;

public:

string getNume\_firma() {

return nume\_firma;

}

string getNr\_telefon() {

return nr\_telefon;

}

void setNume\_firma(string nf) {

nume\_firma = nf;

}

void setNr\_telefon(string nrtel) {

nr\_telefon = nrtel;

}

PersoanaJuridica() :Client() {

this->nume\_firma = "Fara nume";

this->nr\_telefon = "Fara numar de telefon";

}

PersoanaJuridica(const PersoanaJuridica& pj) :Client(pj) {

this->nume\_firma = pj.nume\_firma;

this->nr\_telefon = pj.nr\_telefon;

}

PersoanaJuridica(char\* nume\_client, int nr\_produse, float\* pretCumparaturi, string\* cumparaturi, string nume\_firma, string nr\_telefon) :Client(nume\_client, nr\_produse, pretCumparaturi, cumparaturi) {

this->nume\_firma = nume\_firma;

this->nr\_telefon = nr\_telefon;

}

PersoanaJuridica& operator=(const PersoanaJuridica& pj) {

if (this != &pj) {

Client::operator=(pj);

this->nume\_firma = pj.nume\_firma;

this->nr\_telefon = pj.nr\_telefon;

}

return \*this;

}

friend ostream& operator<<(ostream& out, const PersoanaJuridica& pj) {

out << (Client&)pj;

out << "\nNume firma: " << pj.nume\_firma;

out << "\nNumar de telefon: " << pj.nr\_telefon;

return out;

}

friend istream& operator>>(istream& in, PersoanaJuridica& pj) {

in >> (Client&)pj;

cout << "\nNume firma: ";

in >> pj.nume\_firma;

cout << "\nNumar de telefon: ";

in >> pj.nr\_telefon;

return in;

}

~PersoanaJuridica() {

}

void afisareInfoClient() {

cout << "\nTip client: Persoana juridica";

}

float pretFinal() {

float ti = 0, tf;

int n;

float\* pc;

n = this->getNr\_produse();

pc = this->getPretCumparaturi();

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

ti = ti + pc[i];

if (nume\_firma == "SweetEvy" || nume\_firma == "Carina's Sweets") {

tf = ti - ti \* 2 / 10; // Reducere 20%;

return tf;

}

else return ti;

}

};

int Client::nr\_client = 0;

ifstream f("date.in");

ofstream g("date.out");

int main()

{

vector <Produs\*> listaProduse;

vector <PersoanaFizica\*> listaPF;

vector <PersoanaJuridica\*> listaPJ;

Produs p, pv[100];

map <string, float> firmeClienti;

int com;

int ok = 1;

while (ok == 1) {

cout << "\nApasa 1 pentru MAGAZIN";

cout << "\nApasa 2 pentru CLIENT";

cout << "\nApasa 10 pentru EXIT";

cin >> com;

switch (com) {

case 1: {

system("CLS");

int ok1 = 1;

int com1;

while (ok1 == 1) {

cout << "\nApasa 1 pentru AFISARE PRODUSE RAMASE";

cout << "\nApasa 2 pentru ADAUGARE PRODUSE";

cout << "\nApasa 3 pentru EDITARE PRODUSE";

cout << "\nApasa 4 pentru STERGERE PRODUSE";

cout << "\nApasa 5 pentru PRELUCRARE COMENZI";

cout << "\nApasa 6 pentru REALIZARE RAPORT";

cout << "\nApasa 10 pentru LOG OUT MAGAZIN";

cin >> com1;

switch (com1) {

case 1: {

system("CLS");

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

cout << (\*listaProduse[i]) << " ";

break;

}

case 2: {

system("CLS");

Produs pr;

cin >> pr;

listaProduse.push\_back(&pr);

break;

}

case 3: {

system("CLS");

p.editareProdus();

break;

}

case 4: {

system("CLS");

cout << "\nCe produs doriti sa stergeti: ";

string produs;

cin >> produs;

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

{

if (produs == (\*listaProduse[i]).getNume\_produs())

listaProduse.erase(listaProduse.begin() + i);

}

break;

}

case 5: {

system("CLS");

break;

}

case 6: {

system("CLS");

break;

}

case 10: {

ok1 = 0;

break;

}

default: {

cout << "\nCOMANDA INCORECTA";

break;

}

}

}

}

case 2: {

system("CLS");

int ok2 = 1;

int com2;

cout << "\nApasa f pentru PERSOANA FIZICA/j pentru PERSOANA JURIDICA: ";

char c;

cin >> c;

if (c == 'f') {

PersoanaFizica pf;

while (ok2 == 1) {

cout << "\nApasa 1 pentru VIZUALIZARE PRODUSE";

cout << "\nApasa 2 pentru SELECTARE PRODUSE";

cout << "\nApasa 3 pentru STERGERE PRODUSE";

cout << "\nApasa 4 pentru COMPLETARE DATE PERSONALE";

cout << "\nApasa 5 pentru AFISARE PRET INITIAL/PRET FINAL";

cout << "\nApasa 10 pentru FINALIZARE COMANDA";

cin >> com2;

switch (com2) {

case 1: {

system("CLS");

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

cout << (\*listaProduse[i]) << " ";

break;

}

case 2: {

system("CLS");

break;

}

case 3: {

system("CLS");

break;

}

case 4: {

system("CLS");

char c;

cin >> c;

PersoanaFizica pf;

cin >> pf;

listaPF.push\_back(&pf);

break;

}

case 5: {

system("CLS");

cout << "\nPret initial: " << pf.pretInitial() << " RON";

cout << "\nPret final: " << pf.pretFinal() << " RON";

break;

}

case 10: {

system("CLS");

ok2 = 0;

break;

}

default: {

cout << "\nCOMANDA INCORECTA";

break;

}

}

}

}

if (c == 'j') {

PersoanaJuridica pj;

cin >> pj;

while (ok2 == 1) {

cout << "\nApasa 1 pentru VIZUALIZARE PRODUSE";

cout << "\nApasa 2 pentru SELECTARE PRODUSE";

cout << "\nApasa 3 pentru STERGERE PRODUSE";

cout << "\nApasa 4 pentru COMPLETARE DATE PERSONALE";

cout << "\nApasa 5 pentru AFISARE PRET INITIAL/PRET FINAL";

cout << "\nApasa 10 pentru FINALIZARE COMANDA";

cin >> com2;

switch (com2) {

case 1: {

system("CLS");

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

cout << (\*listaProduse[i]) << " ";

break;

}

case 2: {

system("CLS");

break;

}

case 3: {

system("CLS");

break;

}

case 4: {

system("CLS");

char c;

cin >> c;

PersoanaJuridica pj;

cin >> pj;

listaPJ.push\_back(&pj);

break;

}

case 5: {

system("CLS");

cout << "\nPret initial: " << pj.pretInitial() << " RON";

cout << "\nPret final: " << pj.pretFinal() << " RON";

break;

}

case 10: {

system("CLS");

ok2 = 0;

break;

}

default: {

cout << "\nCOMANDA INCORECTA";

break;

}

}

}

}

break;

}

case 10: {

ok = 0;

break;

}

default: {

cout << "\nCOMANDA INCORECTA";

break;

}

}

}

}

system("CLS");

string pr;

int cant;

float prt;

cout << "\nCe produs doriti sa editati?";

cin >> pr;

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

if (pr == (\*listaProduse[i]).getNume\_produs()) {

cout << "\nPretul intial este " << (\* listaProduse[i]).getPret();

cout << "\nCantitatea intiala este " << (\*listaProduse[i]).getCantitate();

cout << "\nIntroduceti pret nou: "; cin >> prt;

cout << "\nIntroduceti cantitate noua: "; cin >> cant;

(\*listaProduse[i]).setPret(prt);

(\*listaProduse[i]).setCantitate(cant);

break;

system("CLS");

string raport;

cin >> raport;

g << "\nRaport magazin: ";

g << raport;

break;