ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

ПОДАТОЧНИ СТРУКТУРИ И ПРОГРАМИРАЊЕ 2023/2024

АУДИТОРИСКИ ВЕЖБИ 8

КЛАСИ И ОБЈЕКТИ

Класи

Споредба помеѓу структури и класи

Структурите претставуваат стандардна репрезентација на кориснички дефиниран тип на податоци. Најпрвин се креира структурата и дополнително се пишуваат функции за манипулација со податоците кои се чуваат во неа.

```
struct Datum
{
	int d, m, y;
};
void init(Datum& d, int, int, int);
void dodadi_godina(Datum &d, int n);
void dodadi_mesec(Datum &d, int n);
void dodadi_den_day(Datum &d, int n);
```

Имплементацијата на функцијата за иницијализација би била:

```
void init(Datum & d, int dd, int mm, int yy){
	d.d = dd;
	d.m = mm;
	d.y = yy;
}
```

Како што се гледа од примерот, нема директна врска помеѓу структурата и функциите. Во C++ постои опција за вгнездување на функциите во структурата, односно:

```
struct Datum
{
	int d, m, y;
	void init(int dd, int mm, int yy);
	void dodadi_godina(int n);
	void dodadi_mesec(int n);
	void dodadi_den(int n);
};
```

Функцијата за иницијализација е дефинирана на следниов начин (ако дефиницијата на функцијата init е надвор од телото на структурата, тогаш мора да се специфицира името на структурата на која се однесува функцијата, со цел да се спречи конфликт на исти имиња на функции за различни структури):

```
void Datum::init(int dd, int mm, int yy)
{
	d = dd;
	m = mm;
	y = yy;
}
```

Како што се гледа од дефиницијата, при имплементирање на функции членки на структура, нема потреба од пренесување на референци за структурата. Бидејќи дефиницијата е `Datum::init`, кодот на функцијата директно може да пристапи до променливата-структура од која се повикува соодветната функција.

Истата декларација, меѓутоа сега со помош на класа, изгледа вака:

```
class Datum
{
private:
	int d, m, y;
public:
	void init(int dd, int mm, int yy);
	void dodadi_godina(int n);
	void dodadi_mesec(int n);
	void dodadi_den(int n);
};
```

Секоја класа се состои од два дела: приватен и јавен дел (за разлика од структурите кои имаат само јавен дел). Јавниот дел на класата е множество на функции преку кои објектите кои ќе произлезат од класата ќе комуницираат помеѓу себе.
ВАЖНО: Функциите кои не се членки на класата, не можат директно да пристапуваат до податоците во класата сместени во приватниот дел. Така, на пример функцијата `postavi()`, без разлика што зема референца од променливата од тип `Datum`, не може да пристапи до нејзините заштитени(приватни) вредности.

```
void postavi(Datum& d)
{
	d.y = 200; // error: Datum::y is private
}
```

За да може ова да се направи, треба да се обезбедат функции во класата кои ќе го имплементираат кодот од функцијата `postavi()`, а потоа функцијата `postavi()` ќе ги користи тие функции.

Конструктор

Во претходно дефинираната класа беше предвидена функција за иницијализација на вредностите дефинирани во класата. Програмерите понекогаш забораваат да ја иницијализираат почетната вредност на атрибутите на објектот кој го инстанцираат од класата. За да се спречи ова се воведува посебна функција која се нарекува конструктор на класата. Конструкторот секогаш има исто име со името на класата, а не враќа ништо. Ако е потребно да се пренесат аргументи на конструкторот, тогаш тие вредности се задаваат како предефинирани вредности на објектот кој се инстанцира.

```
Datum(int dd = 0, int mm = 0, int yy = 0); // прототип (декларација) на конструкторот
Datum::Datum(int dd = 0, int mm = 0, int yy = 0) // имплементација надвор од класата
{
	// ... проверка на валидноста на датумот ...
	d = dd; m = mm; y = yy;
}
```

Деструктор

Бидејќи со конструкторите се овозможува креирање на објекти од одредена класа, треба да се обезбеди и механизам преку кој ќе се ослободува меморијата која се зафаќа од објектот кој бил креиран. Овој механизам се обезбедува преку функција која се нарекува деструктор. Деструкторот има исто име како и класата, само што напред мора да стои симболот **(~)**. Така на пример за класата за работа со датуми, деструкторот би бил:

```
~Datum();
```

Дефинирање на класи

Класа дефинира множество на објекти што делат исти карактеристики, однесување и релации. Класата е шаблон, “калап”, според кој подоцна може да се генерираат објекти. Накратко речено, класата дефинира кои податоци се употребуваат во рамки на еден објект и кои операции можат да се применат врз тие податоци. Сите објекти ќе ги содржат истите видови на податоци дефинирани од класата, но секој објект – инстанца од класата, може да има различни вредности за тие податоци.

Пример:

```
#include <iostream>
using namespace std;
class Kvadar
{
private:
	int x, y, z;
public:
	Kvadar() {}; /*предефиниран (default) конструктор */
	Kvadar(int a, int b, int c); /* конструктор со аргументи */
	~Kvadar() {}; /* деструктор */
	int Povrsina() { return (2 * (x*y + x*z + y*z)); }
	int Volumen() { return (x*y*z); }
};
Kvadar::Kvadar(int a, int b, int c)
{
	x = a; y = b; z = c;
}
int main()
{
	Kvadar mojKvadar(3, 4, 5);  //Објект mojKvadar, инстанца од класата Kvadar
	cout << "Povrsinata e : " << mojKvadar.Povrsina() << endl << "Volumenot e : " <<
mojKvadar.Volumen() << endl;
}
```

Повикувањето на функциите членови на класата има синтакса слична на синтаксата за пристап на членови на структурата. Може да се декларира и друга функција со исто име надвор од класата или во некоја друга класа. Scope операторот :: овозможува да не се појави конфузија. Функциите на класата исто како и другите функции можат да бидат преоптоварени. Public членовите на класата се достапни и на функциите членови на класата и на сите останати функции во програмата. Често пати овие членови се нарекуваат интерфејс на класата.

Ако функцијата член е декларирана како private може да биде повикана само од други функции членови на истата класа. Конструкторот не враќа никаква вредност (дури ни void). Конструкторот се повикува на местата каде што се декларира објект од дадената класа. Конструкторот може да се преоптовари (да има повеќе верзии на конструкторот).

- Ако за една класа се дефинира конструктор со аргументи **без** предефинирани вредности, тогаш мора да се наведе и default конструкторот.
- Ако за една класа се дефинира конструктор со аргументи **со** предефинирани вредности, тогаш default конструкторот не треба да се пишува.
- Деструкторот извршува спротивни активности од конструкторот. Тој автоматски се повикува кога објектот се брише од меморијата. Деструкторот има исто име како класата со префикс **~**.

Полиња од објекти:

```
Kvadar kvadari[10];
Kvadar kv[3] = { Kvadar(1, 2, 3), Kvadar(10, 20, 30), Kvadar(3, 4, 5) };
kvadari[5] = kv[1];
cout << "Volumenot na vtoriot kvadar e : " << kvadari[5].Volumen() << endl;
```

- Јасен излез, бидејќи објектот `kvadari[5]` ја има вредноста на објектот `kv[1]` кој претходно беше иницијализиран со конструкторот `Kvadar(10, 20, 30)`.

```
cout << "Povrsinata na prviot kvadar e : " << kvadari[2].Povrsina() << endl;
```

- Нејасен излез, грешка, бидејќи не се иницијализирани вредностите на `kvadari[2]`.

ЗАДАЧИ:

1. На една компанија и е потребна програма со помош на која ќе се води евиденција на вработените. Притоа, за секој вработен се знаат неговите основни податоци (име, плата и работна позиција). Работна позиција може да биде една од трите: вработен, раководител или директор. На лицата кои ќе работат со програмата треба да им се овозможи да внесуваат непознат број вработени од тастатура, за потоа да се испечати списокот на вработените во опаѓачки редослед од големината на платата.

```
#include <iostream>
#include <string>
using namespace std;

enum pozicija { vraboten, rakovoditel, direktor };
class Vraboten
{
private:
	string ime;
	int plata;
	pozicija pos;
public:
	Vraboten() {}
	Vraboten(string i, int s, pozicija p) {
		ime = i;
		plata = s;
		pos = p;
	}

	~Vraboten() {}

	// Интерфејсни функции
	void postaviIme(string i);
	void postaviPlata(int p) { plata = p; }
	void postaviPozicija(pozicija p);
	string const zemiIme(void) { return ime; }
	/* Враќа константна низа од знаци, која не смее да се менува во објектот */

	int const zemiPlata(void) { return plata; }
	pozicija const zemiPozicija(void) { return pos; }
};
void Vraboten::postaviIme(string const i)
{
	ime = i;
}
void Vraboten::postaviPozicija(pozicija p)
{
	pos = p;
}
void Sort(Vraboten a[], int n);

int main() {
	Vraboten vraboteni[100];
	int plata, pos, rbr = 0;
	string ime;
	string s[] = { "Vraboten", "Rakovoditel", "Direktor" };
	cout << "Vnesete podatoci, za kraj vnesete KRAJ:" << endl;
	for (;;) {
		cout << "Ime:";
		cin >> ime;
		if (ime.compare("KRAJ") == 0) break;
		cout << "Plata:";
```

```
			cin >> plata;
			cout << "Pozicija(vraboten=0, rakovoditel=1, direktor=2):";
			cin >> pos;

			vraboteni[rbr].postaviIme(ime);
			vraboteni[rbr].postaviPlata(plata);
			vraboteni[rbr].postaviPozicija((pozicija)pos);
			rbr++;
		}
		Sort(vraboteni, rbr);
		cout << endl;
		for (int i = 0; i < rbr; i++) {
			cout << vraboteni[i].zemiIme() << " " << vraboteni[i].zemiPlata() << " " <<
s[vraboteni[i].zemiPozicija()] << endl;
		}
		return 0;

}
void Sort(Vraboten a[], int n) {
		int i, j;
		Vraboten p;
		for (i = 0; i < n - 1; i++)
			for (j = i + 1; j < n; j++)
				if (a[i].zemiPlata() < a[j].zemiPlata()) {
					p = a[j];
					a[j] = a[i];
					a[i] = p;
				}
}
```

2. Потребна е програма за работа со e-mail пораки. За секоја порака се знае кој е испраќачот, кој е примачот, темата на пораката и нејзината содржина. Пораката може да содржи најмногу 2000 знаци. Треба да се обезбеди интерфејс за прикажување на комплетна порака на екран. Програмата нуди можност за проверка на валидноста на адресата на примачот. Во самата програма се внесуваат параметрите на пораката, се проверува валидноста на адресата на примачот (постоење на симболот @ во адресата) и се печати пораката на екран.

```
#include <iostream>
#include <string.h>
using namespace std;

class EMail
{
private:
		enum { AddrLen = 100, SubLen = 200, TextLen = 1000 };
		char to[AddrLen];
		char from[AddrLen];
		char subject[SubLen];
		char text[TextLen];
public:
		EMail(char *cTo, char *cFrom, char *cSubject, char *cText)
		{
			strncpy(to, cTo, AddrLen - 1);
			strncpy(from, cFrom, AddrLen - 1);
			strncpy(subject, cSubject, SubLen - 1);
			strncpy(text, cText, TextLen - 1);
			to[AddrLen - 1] = from[AddrLen - 1] = subject[SubLen - 1] = text[TextLen -
1] = 0;
		}
		~EMail() {}
```

```
        void setTo(char const *n) {
                strncpy(to, n, AddrLen - 1);
                to[AddrLen - 1] = 0;
        }
        void setFrom(char const *n) {
                strncpy(from, n, AddrLen - 1);
                from[AddrLen - 1] = 0;
        }
        void setSubject(char const *n) {
                strncpy(subject, n, SubLen - 1);
                subject[SubLen - 1] = 0;
        }
        void setText(char const *n) {
                strncpy(text, n, TextLen - 1);
                text[TextLen - 1] = 0;
        }

        char const *getTo() { return to; }
        char const *getFrom() { return from; }
        char const *getSubject() { return subject; }
        char const *getText() { return text; }
        void print();
};

void EMail::print()
{
        cout << "To: " << getTo() << endl;
        cout << "From: " << getFrom() << endl;
        cout << "Subject: " << getSubject() << endl;
        cout << "Text: " << getText() << endl;
}

bool checkMail(char *c)
{
        int vk = 0;
        while (*c != 0)
                if ('@' == *c++)
                        vk++;
        return (1 == vk);
}

int main()
{
        char to[200], from[300], subject[400], text[4000];
        cout << "Vnesi DO kogo e porakata" << endl;
        cin >> to;
        if (checkMail(to)) {
                cout << "Vnesi OD kogo e porakata" << endl;
                cin >> from;
                cout << "Vnesi naslov" << endl;
                cin.ignore(1);
                cin.getline(subject, 400);
                cout << "Vnesi sodrzina" << endl;
                cin.getline(text, 4000);
                EMail poraka(to, from, subject, text);
                cout << "Isprateno:" << endl;
                poraka.print();
        }
        else
        {
                cout << "Pogresna adresa za isprakjanje" << endl;
        }
        return 0;
}
```

3. Да се напише **класа Let** во која се чуваат информации за: шифра на летот (цел број), град на полетување (низа од макс. 30 знаци), град на слетување (низа од макс. 30 знаци) и шифра на пилотот кој управува со летот (цел број). Да се напишат потребните **конструктори** и **методи** (не постои лет, ниту пилот со негативен број за шифра). Да се напише **метода pechati**() која го печати летот во формат: <shifra_let>: <poletuva_od>, <sletuva_vo>.
Да се напише **класа Pilot** која чува информации за: шифра на пилотот, име на пилотот (низа од макс. 30 знаци), плата која ја зема за еден лет (реален број).
Да се напише **класа AvioKompanija** која чува информации за: летови кои ги нуди компанијата (низа од макс. 15 летови), број на летови (цел број), име на компанијата (низа од макс. 30 знаци).
За класите Pilot и AvioKompanija да се напишат потребните **конструктори** и **методи**.
Во класата AvioKompanija да се напише **метода proveri**(int shifra), која проверува дали некој од летовите кои ги нуди компанијата бил изведен од пилот со шифра shifra и го враќа индексот на летот во низата, или -1 доколку нема таков лет. Да се напише **метода plata**(Pilot *piloti, int n), која прима низа од n пилоти и треба да ја испечати плата која секој од пилотите во низата треба да ја земе. Во низата со пилоти може да има пилоти кои немаат изведено ниту еден лет.

```
#include <iostream>
#include <string>

using namespace std;

class Pilot {
private:
	int shifraPilot;
	string ime;
	int plataLet;

public:
	Pilot(int sP = 0, string i = "", int pL = 0) {
		if (sP >= 0)
			shifraPilot = sP;
		else
			shifraPilot = 0;
		ime = i;
		plataLet = pL;
	}

	int getShifra() {
		return shifraPilot;
	}

	int getPlata() {
		return plataLet;
	}

	string getIme() {
		return ime;
	}
};

class Let {
private:
	int shifraLet;
	string gradOd;
	string gradDo;
	int shifraPilot;

public:
```

```
	Let(int sL = 0, string gOd = "", string gDo = "", int sP = 0) {
		if (sL >= 0)
			shifraLet = sL;
		else
			shifraLet = 0;
		gradOd = gOd;
		gradDo = gDo;
		shifraPilot = sP;
	}

	void pechati() {
		cout << "Let so shifra " << shifraLet << ": " << gradOd << ", " << gradDo <<
endl;
	}

	int getShifraPilot() {
		return shifraPilot;
	}
};

class AvioKompanija {

private:
	Let letovi[15];
	int brojLetovi;
	string imeKompanija;

public:
	AvioKompanija(Let l[] = {}, int bL = 0, string iK = "") {
		brojLetovi = bL;
		for (int i = 0; i < brojLetovi; i++)
			letovi[i] = l[i];
		imeKompanija = iK;
	}

	string getIme() {
		return imeKompanija;
	}

	int proveri(int shifra) {
		for (int i = 0; i < brojLetovi; i++) {
			if (letovi[i].getShifraPilot() == shifra)
				return i;
		}
		return -1;
	}

	void plata(Pilot piloti[], int n) {
		for (int i = 0; i < n; i++) {
			int plata = 0;
			for (int j = 0; j < brojLetovi; j++) {
				if (piloti[i].getShifra() == letovi[j].getShifraPilot()) {
					plata += piloti[i].getPlata();
				}
			}
			cout << "Pilotot " << piloti[i].getIme() << " ima plata od " << plata
<< " denari" << endl;
		}
	}
};



int main()
```

```
{
	Pilot piloti[3] = { Pilot(1, "Pilot1", 18000), Pilot(2, "Pilot2", 25000), Pilot(3, 
"Pilot3", 30000) };
	Let letovi[5] = { Let(1, "Skopje", "Malme", 3), Let(2, "Skopje", "Paris", 1), 
Let(3, "Rim", "Madrid", 3),
		Let(4, "Belgrad", "Venecija", 2), Let(5, "Skopje", "berlin", 3) };
	AvioKompanija kompanija(letovi, 5, "WizzAir");

	if (kompanija.proveri(3) == -1)
		cout << "Ne postoi takov pilot vo aviokompanijata " << kompanija.getIme() << 
endl;
	else
		cout << "Vo aviokompanijata " << kompanija.getIme() << " postoi pilot so 
shifra 3" << endl;

	kompanija.plata(piloti, 3);

}
```