ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

ПОДАТОЧНИ СТРУКТУРИ И ПРОГРАМИРАЊЕ 2023/2024

Аудиториски вежби 9

Copy constructor и динамичка алокација

Copy Constructor

Copy конструкторот е посебен конструктор чија задача е да направи идентична копија на објект креиран од дадена класа. Доколку copy конструкторот не е дефиниран од програмерот, по дефиниција компајлерот генерира copy конструктор за соодветната класа. Default copy конструкторот прави копија од содржината на едниот објект, бит по бит, во елементите на другиот објект. Доколку класата содржи податочни елементи за кои динамички се алоцира меморија, тогаш автоматски генерираниот copy конструктор не е сосема соодветен (ова ќе биде објаснето подоцна). Тогаш неопходно е програмерот да напише соодветен copy конструктор.

Пример:

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

//Класа и прототип copy constructor
class Person {
public:
	Person(const char *name0 = "", int age0 = 0);
	Person(const Person &p); //copy constructor
	int GetAge() { return age; }
	char *GetName() { return name; }
private:
	char name[30];
	int age;
};
Person::Person(const char *name0, int age0)
{
	strcpy(name, name0);
	age = age0;
}
//Имплементација
Person::Person(const Person &p)
{
	strcpy(name, p.name);
	age = p.age;
	cout << "ova e copy constructor\n";
}

void Pecati_God(Person p);
Person Novo();

int main() {
	cout << "Vo main:\n";
	Person lice("Matti", 20);
	cout << "Pred twin:\n";
	Person twinBrother(lice); // Person twinBrother = lice; //Експлицитно користење
	cout << "Kraj twin:\n";
	cout << "Pred Pecati_God:\n";
	Pecati_God(lice); //Имплицитно користење
	cout << "Kraj Pecati_God:\n";
	cout << "Pred Novo za tret:\n";
	Person tret = Novo(), cetvrt;
	cout << "Kraj Novo za tret:\n";
	cout << "Pred Novo za cetvrt:\n";
	cetvrt = Novo();
	cout << "Kraj Novo za cetvrt:\n";
	return 0;
}
```

```
void Pecati_God(Person p)
{
	cout << "Vo Pecati_God : \n";
	cout << "Liceto " << p.GetName() << " ima " << p.GetAge() << " godini" << endl;
}
Person Novo()
{
	cout << "Vo Novo : \n";
	Person l("Pero", 40);
	return l; // Имплицитно користење
}
```

Излез од програмата:

```
Vo main:
Pred twin:
ova e copy constructor
Kraj twin:
Pred Pecati_God:
ova e copy constructor
Vo Pecati_God :
Liceto Matti ima 20 godini
Kraj Pecati_God:
Pred Novo za tret:
Vo Novo :
ova e copy constructor
Kraj Novo za tret:
Pred Novo za cetvrt:
Vo Novo :
ova e copy constructor
Kraj Novo za cetvrt:
```

Со примерот се илустрираат следните карактеристики на copy конструкторот:

1. Дефиницијата `Person::Person(const Person &p)` означува copy конструктор. Тој секогаш има еден параметар кој е константна референца кон објект од истата класа (`const Person &p`).
2. Кодот на copy конструкторот треба да обезбедува идентична копија на објектот чија референца е влезен параметар во овој конструктор.
3. Copy конструкторот се повикува автоматски секогаш при иницијализација на податочните членови на еден објект со вредностите на податочните членови на друг објект (`Person twinBrother(lice);` или `Person twinBrother = lice;`)
4. Copy конструкторот се повикува автоматски секогаш кога се проследува објект по вредност во функција (`Pecati_God(lice);`)
5. Copy конструкторот се повикува автоматски секогаш кога функција како вредност враќа објект од класата (`Person l("Pero", 40); return l;`)

Задачи

1. За еден магацин потребна е програма за водење на евиденција на производи. За секој производ се чуваат податоци за името (максимум 20 знаци), шифрата, количината на залиха во магацин и количината која е продадена.
Да се обезбеди интерфејс за сортирање на производите. Корисникот може да одбере насока на сортирање (опаѓачки/растечки редослед) и критериум по кој ќе се врши сортирањето. Доколку не се проследат вредности за насоката и критериумот, сортирањето ќе се врши по опаѓачки редослед според бројот на производи на залиха.

```
#include <iostream>
#include <cstring>
#include <string>
using namespace std;
#define MAX 100
enum kriterium { imeP, sifraP, kol_prodadeni, kol_zaliha };
class Proizvod
{
private:
	char ime[20];
	int sifra, lager, prodadeni;
public:
	Proizvod(char *ii = "", int ss = 0, int ll = 0, int pp = 0)
	{
		strcpy(ime, ii);
		sifra = ss;
		lager = ll;
		prodadeni = pp;
	}
	Proizvod(const Proizvod &p)
	{
		strcpy(ime, p.ime);
		sifra = p.sifra;
		lager = p.lager;
		prodadeni = p.prodadeni;
	}
	const char* getIme() { return ime; }
	int getSifra() { return sifra; }
	int getLager() { return lager; }
	int getKolicina() { return prodadeni; }

	void setIme(char * ii) { strcpy(ime, ii); }
	void setSifra(int ss) { sifra = ss; }
	void setLager(int ll) { lager = ll; }
	void setKolicina(int pp) { prodadeni = pp; }
	void promeni(Proizvod p)
	{
		setIme(p.getIme());
		setSifra(p.getSifra());
		setLager(p.getLager());
		setKolicina(p.getKolicina());
	}
	void pecati()
	{
		cout << "Proizvod:" << ime << ",sifra: " << sifra << ", na lager: ";
		cout << lager << ", prodadeni: " << prodadeni << endl;
	}
	~Proizvod() {}
};
void sortirajImeUp(Proizvod a[], int br_proizvodi)
{
	Proizvod pom;
	for (int i = 0; i < br_proizvodi - 1; i++)
	{
		for (int j = i + 1; j < br_proizvodi; j++)
		{
			if (strcmp(a[i].getIme(), a[j].getIme()) > 0)
			{
				pom.promeni(a[i]);
				a[i].promeni(a[j]);
				a[j].promeni(pom);
			}
		}
	}
}
```

```
void sortirajImeDown(Proizvod a[], int br_proizvodi)
{
	Proizvod pom;
	for (int i = 0; i < br_proizvodi - 1; i++)
	{
		for (int j = i + 1; j < br_proizvodi; j++)
		{
			if (strcmp(a[i].getIme(), a[j].getIme()) < 0)
			{
				pom.promeni(a[i]);
				a[i].promeni(a[j]);
				a[j].promeni(pom);
			}
		}
	}
}
void sortirajSifraUp(Proizvod a[], int br_proizvodi)
{
	Proizvod pom;
	for (int i = 0; i < br_proizvodi - 1; i++)
	{
		for (int j = i + 1; j < br_proizvodi; j++)
		{
			if (a[i].getSifra() > a[j].getSifra())
			{
				pom.promeni(a[i]);
				a[i].promeni(a[j]);
				a[j].promeni(pom);
			}
		}
	}
}
void sortirajSifraDown(Proizvod a[], int br_proizvodi)
{
	Proizvod pom;
	for (int i = 0; i < br_proizvodi - 1; i++)
	{
		for (int j = i + 1; j < br_proizvodi; j++)
		{
			if (a[i].getSifra()<a[j].getSifra())
			{
				pom.promeni(a[i]);
				a[i].promeni(a[j]);
				a[j].promeni(pom);
			}
		}
	}
}
void sortirajZalihaUp(Proizvod a[], int br_proizvodi)
{
	Proizvod pom;
	for (int i = 0; i < br_proizvodi - 1; i++)
	{
		for (int j = i + 1; j < br_proizvodi; j++)
		{
			if (a[i].getLager()>a[j].getLager())
			{
				pom.promeni(a[i]);
				a[i].promeni(a[j]);
				a[j].promeni(pom);
			}
		}
	}
}
```

```
void sortirajZalihaDown(Proizvod a[], int br_proizvodi)
{
	Proizvod pom;
	for (int i = 0; i < br_proizvodi - 1; i++)
	{
		for (int j = i + 1; j < br_proizvodi; j++)
		{
			if (a[i].getLager()<a[j].getLager())
			{
				pom.promeni(a[i]);
				a[i].promeni(a[j]);
				a[j].promeni(pom);
			}
		}
	}
}
void sortirajProdadeniUp(Proizvod a[], int br_proizvodi)
{
	Proizvod pom;
	for (int i = 0; i < br_proizvodi - 1; i++)
	{
		for (int j = i + 1; j < br_proizvodi; j++)
		{
			if (a[i].getKolicina()>a[j].getKolicina())
			{
				pom.promeni(a[i]);
				a[i].promeni(a[j]);
				a[j].promeni(pom);
			}
		}
	}
}
void sortirajProdadeniDown(Proizvod a[], int br_proizvodi)
{
	Proizvod pom;
	for (int i = 0; i < br_proizvodi - 1; i++)
	{
		for (int j = i + 1; j < br_proizvodi; j++)
		{
			if (a[i].getKolicina()<a[j].getKolicina())
			{
				pom.promeni(a[i]);
				a[i].promeni(a[j]);
				a[j].promeni(pom);
			}
		}
	}
}

void sort(Proizvod a[], int br_proizvodi, int krit = kol_zaliha, bool nasoka = 1)
{
	switch (krit)
	{
	case imeP:
		if (nasoka)
			sortirajImeUp(a, br_proizvodi);
		else
			sortirajImeDown(a, br_proizvodi);
		break;
	case sifraP:
		if (nasoka)
			sortirajSifraUp(a, br_proizvodi);
		else
			sortirajSifraDown(a, br_proizvodi);
		break;
```

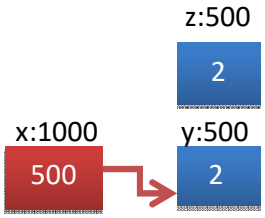
```
	case kol_zaliha:
		if (nasoka)
			sortirajZalihaUp(a, br_proizvodi);
		else
			sortirajZalihaDown(a, br_proizvodi);
		break;
	case kol_prodadeni:
		if (nasoka)
			sortirajProdadeniUp(a, br_proizvodi);
		else
			sortirajProdadeniDown(a, br_proizvodi);
		break;
	}
}

int main()
{
	Proizvod niza[MAX];
	int sifra, lager, prodadeni, rbr = 0;
	char ime[MAX];

	for (int g = 0; g < 5; g++)
	{
		cout << "Vnesi ime, sifra, na lager i prodadeni:" << endl;
		cin >> ime >> sifra >> lager >> prodadeni;
		niza[rbr].setIme(ime);
		niza[rbr].setSifra(sifra);
		niza[rbr].setLager(lager);
		niza[rbr].setKolicina(prodadeni);
		rbr++;
		cin.getline(ime, MAX);
	}
	cout << "Kraj na vnesuvanjeto:" << endl;
	cout << "Spored sto ke se sortira?\n Mozni opcii:  ";
	cout << " imeP | sifraP | kol_prodadeni | kol_zaliha" << endl;
	string kriteriumi[4] = { "imeP", "sifraP", "kol_prodadeni", "kol_zaliha" };
	string k;
	cin >> k;
	int kr = 3;

	for (int z = 0; z < 4; z++) {
		if (k == kriteriumi[z])
		{
			kr = z;
			break;
		}
	}
	cout << "Dali ke sortirate vo rastecki redosled? Y|N" << endl;
	char c;
	cin >> c;
	bool n;
	if (c == 'Y' || c == 'y')
		n = true;
	else
		n = false;
	sort(niza, rbr, kr, n);
	for (int z = 0; z < rbr; z++)
		niza[z].pecati();
	return 0;
}
```

Покажувачи и референци (повторување)

Променлива е именувана мемориска локација.
Покажувач е променлива која како вредност содржи адреса на друга променлива.
Референца е ново име за иста променлива, односно за иста мемориска локација.

Пример:

```
int y = 2, *x = &y, &z = y;
cout << &y << ' ' << y << endl;                  //500 2
cout << &z << ' ' << z << endl;                  //500 2
cout << &x << ' ' << x << '\t' << *x << endl; //1000 500 2
```



Алоцирање на меморија (повторување)

Статичко алоцирање – меморија за променливите се алоцира за време на **компајлирање** на програмата и таа е фиксна.
Динамичко алоцирање – меморија се алоцира за време на **извршување** на програмата. Динамичката алокација на меморија обезбедува флексибилност во додавање и бришење на објекти. Вистинското значење на покажувачите доаѓа до израз при алоцирање на **неименуван мемориски простор** за време на извршување на програмата. Тогаш името на покажувачот е единствен начин за пристапување до таа меморија.

Динамичка алокација на меморија (повторување)

Во **C** меморијата динамички се алоцира со функцијата **`malloc`**, додека истата се деалоцира со функцијата **`free`** (се наоѓаат во библиотеката `malloc.h`).
Во **C++** меморијата динамички се алоцира со функцијата `new`, додека истата се деалоцира со функцијата `delete`.

Пример:

```
int *pn = new int;  /* динамичка алокација на меморија за една целобројна променлива */
```

- `new` наоѓа слободен мемориски блок со специфицираната големина и ја враќа неговата почетна адреса.
- `int` кажува колку меморија да се алоцира

```
delete pn;
```

- `delete` ја враќа алоцираната меморија назад во слободниот мемориски простор.

После `delete`, ослободената меморија може повторно да се алоцира, додека покажувачот кој претходно покажувал кон неа, не е уништен и може повторно да се користи.
Користете `delete` заради ефикасно програмирање!

Динамичка алокација на низи

Динамичката алокација најчесто се користи за резервирање на меморија за големи множества на податоцим како низи, структури, итн.

Пример:

```
int brojElementi;
cin >> brojElementi;
int *niza = new int[brojElementi];
delete[] niza;

char boja[] = "bela";
char *pBoja = new char[strlen(boja) + 1];  /* +1 заради '\0' терминаторот */
strcpy(pBoja, boja);
```

Задачи

2. Да се напише класа за опис на ученици. За секој ученик се чува името како динамички алоцирано поле од знаци, просекот како децимален број и школска година како цел број. Класата треба да ги содржи следните функции:

- Конструктори
- get() и set() методи
- Зголемување на запишаната школска година за еден.
- Печатење на ученик со сите негови податоци.

Потоа треба да се креира класа паралелка која во себе содржи динамички алоцирано поле од ученици, како и број на елементи во полето. Класата треба да ги содржи следните функции:

- Конструктори
- Функција за зголемување на запишаната школска година за еден за сите ученици во полето.
- Функција за додавање на еден ученик во полето од ученици
- Функција за печатење на сите ученици во класата
- Функција која ги печати само оние ученици кои имаат просек 5.0.

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

class Uchenik
{
	char *ime;
	float prosek;
	int godina;
public:
	Uchenik()
	{
		ime = new char[1];
		strcpy(ime, "");
		prosek = 0.0;
		godina = 0;
	}

	Uchenik(char *i, float p, int g) :prosek(p), godina(g)
	{
		ime = new char[strlen(i) + 1];
		strcpy(ime, i);
	}
	Uchenik(const Uchenik &u) :prosek(u.prosek), godina(u.godina)
	{
		ime = new char[strlen(u.ime) + 1];
		strcpy(ime, u.ime);
	}
	~Uchenik() { delete[] ime; }
	const char *getIme() { return ime; }
	float getProsek() { return prosek; }
	int getGodina() { return godina; }
	void setIme(char *i)
	{
		delete[] ime;
		/*не заборавајте најпрво да ја ослободите меморијата која ја зафаќа име,
		а потоа да алоцирате нова меморија, онолку колку што зафаќа аргументот i*/
		ime = new char[strlen(i) + 1];
		strcpy(ime, i);
	}
	void setProsek(float p) { prosek = p; }
	void setGodina(int g) { godina = g; }
```

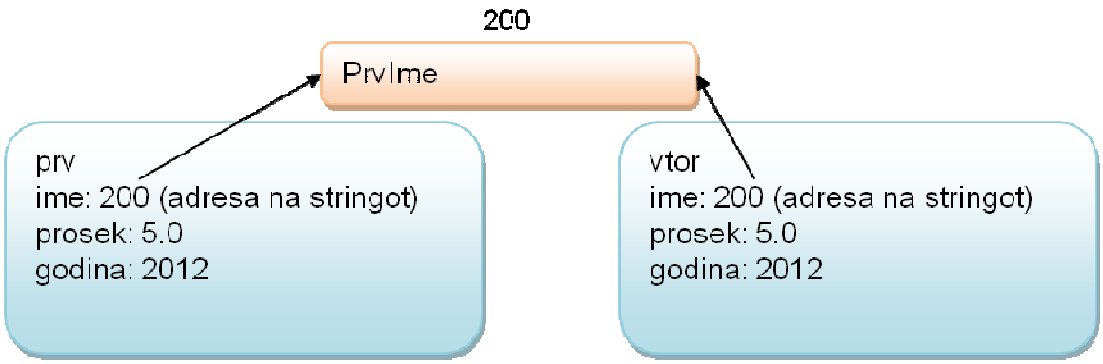
```
	void zgolemiGodina() { godina++; }
	void pechati()
	{
		cout << "Ime: " << ime << "\nProsek: " << prosek << "\n Godina: " << godina
<< endl;
	}
	void izednachi(Uchenik &u)
	/*кога ќе го преоптовариме операторот =, нема да има потреба од оваа функција*/
	{
		setIme(u.ime); /* ја повикуваме погоре дефинираната функција, која врши од
ново динамичка деалокација и алокација на меморија */
		prosek = u.prosek;
		godina = u.godina;
	}

};

class Paralelka
{
	Uchenik *klas;
	int broj;
public:
	Paralelka(Uchenik *u = NULL, int br = 0)
	{
		broj = br;
		klas = new Uchenik[broj];
		for (int i = 0; i < broj; i++)
			//klas[i] = u[i];
			/* Напомена: ако не го имаме преоптоварено операторот =, ваквото
доделување може да доведе до неправилно функционирање на програмата, затоа ќе напишеме
помошна функција за изедначување на два објекти од тип Uchenik */
			klas[i].izednachi(u[i]);
	}
	Paralelka(const Paralelka &p)
	{
		broj = p.broj;
		klas = new Uchenik[broj];
		for (int i = 0; i < broj; i++)
			klas[i].izednachi(p.klas[i]);
	}
	~Paralelka() { delete[] klas; }
	void zgolemiNaSite()
	{
		for (int i = 0; i < broj; i++)
			klas[i].zgolemiGodina();
		/*ја повикуваме функцијата од кл. Uchenik што ја зголемува годината за 1*/
	}
	void dodadiUchenik(Uchenik &u)
	{
		Uchenik *pom;
		pom = new Uchenik[broj + 1];
		/* креираме помошно поле од ученици со простор за еден ученик повеќе */
		broj += 1;
		/* вкупниот број на ученици во полето ќе се зголеми за 1 */
		for (int i = 0; i < broj - 1; i++)
			pom[i].izednachi(klas[i]);
		/* ги копираме сите тековни ученици во помошното поле */
		pom[broj - 1].izednachi(u);
		/* и го додаваме новиот ученик на последна позиција */
		delete[] klas;
		/* ослободуваме меморија за претходното поле од ученици */
		klas = pom;
		/* покажувачот да покажува кон полето со еден ученик повеќе */
	}
```

```
	void pechati()
	{
		cout << "Klasot ima " << broj << " uchenici:" << endl;
		for (int i = 0; i < broj; i++)
			klas[i].pechati();
	}
	void pechati5()
	{
		cout << "Uchenici so prosek 5.0 se slednive:" << endl;
		for (int i = 0; i < broj; i++)
			if (klas[i].getProsek() == 5.0)
				klas[i].pechati();
	}

};
int main()
{
	Uchenik u[4] = { Uchenik(), Uchenik("Prv",5.0,2012), Uchenik("Vtor",4.5,2012),
Uchenik("Tret",5.0,2011) };
	u[0].pechati();
	u[1].izednachi(u[2]);
	u[1].pechati();
	Paralelka p = Paralelka(u, 4);
	p.pechati();
	Uchenik nov = Uchenik("Nov", 5.0, 2012);
	p.dodadiUchenik(nov);
	p.pechati();
	p.zgolemiNaSite();
	p.pechati();
	p.pechati5();
}
```

Објаснување: зошто не смееме да изедначиме двајца ученици со помош на = (ако не е проптоварен како во примерот горе)?
Ако на пример во `main()` креираме двајца ученици на следниов начин:

```
Uchenik prv = Uchenik("PrvIme", 5.0, 2012), vtor;
vtor = prv;
vtor.pechati();
```

Во меморија се случува следново:



Ако после тоа ја ослободиме меморијата за името на првиот ученик и го испечатиме името на вториот, настанува грешка (затоа што мемориската локација каде што сега покажува вториот ученик е веќе ослободена)

```
delete prv.getIme();
vtor.pechati();  //печати грешно име
```

Заради тоа, користевме функција izednachi која креира сосема нова мемориска локација за вториот ученик, која не се ослободува ако го избришеме името на првиот:

```
Uchenik prv = Uchenik("PrvIme", 5.0, 2012), vtor;
vtor.izednachi(prv);
vtor.pechati();
delete prv.getIme();
```