ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

ПОДАТОЧНИ СТРУКТУРИ И ПРОГРАМИРАЊЕ 2023/2024

Аудиториски вежби 12

Виртуелни функции

Контрола на пристап - `protected`

До членовите кои во основната класа се декларирани како `protected`, може директно да се пристапи од изведената класа (при `public` и `protected` наследување). Тие се однесуваат како `private` членови за надворешниот свет, а како `public` за изведените класи.

Виртуелни функции

Виртуелните функции се декларираат со наведување на клучниот збор `virtual` пред декларацијата на функцијата во класата. Како виртуелни се декларираат функциите во основната класа за кои се очекува да бидат препокриени (редефинирани) од страна на изведените класи. При декларирање и имплементација на виртуелните функции во изведените класи не мора да се користи клучниот збор `virtual`. При повик на виртуелна функција преку покажувач од основната класа, се извршува онаа виртуелна функција која припаѓа на класата на покажаниот објект.

Динамичко поврзување

Механизам кој овозможува една функција да се повикува според типот на објектот, а не според типот на покажувачот, се нарекува динамичко поврзување. Одлучувањето која функција ќе се повика се прави за време на извршување на програмата, односно динамички. Виртуелната функција од основната класа не мора да се препокрие во изведените класи (тогаш важи функцијата од основната класа).

Ако во изведените класи се декларира функција со исто име како виртуелната од основната класа, но со различен број и/или тип на аргументи, тогаш таа ги сокрива сите останати функции од основната класа со исто име.

Чисти виртуелни функции

Кога виртуелната функција не е редефинирана во изведената класа, се користи верзијата од основната класа. Но, често во основната класа нема комплетна дефиниција на виртуелната функција. На пример, основната класа нема доволно солидни податоци за да се напише комплетна функција или сакаме да се осигураме дека сите изведени класи ќе ја препокријат основната функција. За овие случаи C++ поддржува чисти виртулени функции. Чиста виртуелна функција е таква функција која не мора да има дефиниција во основната класа. За да се декларира чиста виртуелна функција, се користи:

```
virtual tip_na_funkcija ime_na_funkcija(lista parametri) = 0;
```

Секоја изведена класа мора да ја препокрие чистата виртуелна функција со своја верзија.

Апстрактни класи

Класата која содржи барем една чиста виртуелна функција е апстрактна. Од неа не може да се креира објект (има една или повеќе функции кои немаат дефиниција).

Наместо тоа, апстрактната класа е основа за изведени класи.

Од апстрактна класа може да се креираат покажувачи и референци!

Забелешки:

- Конструкторот не може да биде виртуелна функција.
- Деструкторот може да биде виртуелна функција. Ако деструкторот е виртуелен, тогаш за време на извршување се определува кој деструктор ќе биде повикан.
- Деструкторот од основната класа секогаш се извршува (или како единствен или после деструкторот од изведената класа).

– Во рамки на деструкторот од изведената класа не треба да се повикува деструкторот од основната класа (тој имплицитно се повикува).

Пример:

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

class broj
{
protected:
	int vrednost;
public:
	void setVrednost(int i) { vrednost = i; }
	virtual void prikazhi() = 0;  /* funkcijata e chisto virtuelna, odnosno mora da se
prepokrie vo trite izvedeni klasi */
};

class hexBroj : public broj
{
public:
	void prikazhi() { cout << hex << vrednost << endl; }
};

class decBroj : public broj
{
public:
	void prikazhi() { cout << vrednost << endl; }
};

class octBroj : public broj
{
public:
	void prikazhi() { cout << oct << vrednost << endl; }
};

int main()
{
	decBroj d;
	hexBroj h;
	octBroj o;
	/*  broj b; greshka, ne smee da se definira objekt od apstraktna klasa */
	broj *b1 = &h; /* pokazhuvachot od osnovnata klasa broj pokazhuva na objekt od
izvedenata klasa hexBroj */
	b1->setVrednost(50);
	b1->prikazhi(); /* se povikuvaat funkcii od klasata hexBroj */
	broj &b2 = o; /* referenca od osnovnata klasa broj pokazhuva na objekt od
izvedenata klasa octBroj*/
	b2.setVrednost(50);
	b2.prikazhi();/* se povikuvaat funkcii od klasata octBroj */

	d.setVrednost(20);
	d.prikazhi();
	h.setVrednost(20);
	h.prikazhi();
	o.setVrednost(20);
	o.prikazhi();
	return 0;
}
```

Повеќекратно наследување

C++ дозволува повеќекратно наследување од повеќе класи истовремено. Изведената класа наследува функционалности од сите родителски класи истовремено. Конструкторите од основните класи се повикуваат пред конструкторот од изведената класа, по редоследот во кој се декларирани (повикани).

Пример:

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

class Vozilo
{
public:
	Vozilo() { cout << "Vozilo Constructor" << endl; }
	virtual ~Vozilo() { cout << "Vozilo Destructor" << endl; }
	virtual void zabrzuvanje() const { cout << "Vozilo zabrzuvanje" << endl; }
};

class Avtomobil : virtual public Vozilo
	/* ako nasleduvanjeto e virtuelno (i vo dvete klasi, Avtomobil i Avion), togash
klasata MlazenAvtomobil kje raspolaga samo so eden primerok od chlenovite na klasata
Vozilo */
{
public:
	Avtomobil() { cout << "Avtomobil Constructor" << endl; }
	virtual ~Avtomobil() { cout << "Avtomobil Destructor" << endl; }
	virtual void vozi() const { cout << "Avtomobil vozi" << endl; }
};

class Avion : virtual public Vozilo
{
public:
	Avion() { cout << "Avion Constructor" << endl; }
	virtual ~Avion() { cout << "Avion Destructor" << endl; }
	virtual void leta() const { cout << "Avion leta" << endl; }
};

class MlazenAvtomobil : public Avtomobil, public Avion
{
public:
	MlazenAvtomobil() { cout << "MlazenAvtomobil Constructor" << endl; }
	virtual ~MlazenAvtomobil() { cout << "MlazenAvtomobil Destructor" << endl; }
	virtual void vozi() const { cout << "MlazenAvtomobil vozi" << endl; }
	virtual void leta() const { cout << "MlazenAvtomobil leta" << endl; }
};

void analizirajAvtomobil(Avtomobil *testAvto) /* se prenesuva pokazhuvach od klasata
Avtomobil */
{
	testAvto->vozi();
}

void analizirajAvion(Avion *testAvio)
{
	testAvio->leta();
}

int main()
{
	Avtomobil mojAvto;
	Avion mojAvio;
	MlazenAvtomobil myAvtoAvio;
```

```
	cout << "Testiranje na avtomobil" << endl;
	analizirajAvtomobil(&mojAvto); /* ja povikuva funkcijata vozi od klasata Avtomobil */
	analizirajAvtomobil(&myAvtoAvio); /* ja povikuva funkcijata vozi od klasata
MlazenAvtomobil */
	cout << "Testiranje na avion" << endl;
	analizirajAvion(&mojAvio); /* ja povikuva funkcijata leta od klasata Avion */
	analizirajAvion(&myAvtoAvio); /* ja povikuva funkcijata leta od klasata
MlazenAvtomobil */
	cout << endl;
	return 0;
}
```

Излез:

```
Vozilo Constructor
Avtomobil Constructor
Vozilo Constructor
Avion Constructor
Vozilo Constructor
Avtomobil Constructor
Avion Constructor
MlazenAvtomobil Constructor
Testiranje na avtomobil
Avtomobil vozi
MlazenAvtomobil vozi
Testiranje na avion
Avion leta
MlazenAvtomobil leta

MlazenAvtomobil Destructor
Avion Destructor
Avtomobil Destructor
Vozilo Destructor
Avion Destructor
Vozilo Destructor
Avtomobil Destructor
Vozilo Destructor
```

ЗАДАЧИ:

1. Да се напише класа **Registracija** која ќе содржи информации за регистрациите издадени од АМСМ. За регистрацијата се потребни следните информации: име на сопственикот, број на шасија, зафатнина на моторот, основна цена за регистрација на возило и листа од земји за кои важи регистрацијата (динамички алоцирана низа од динамички алоцирани низи од знаци). За класата да се обезбеди конструктор, сору конструктор, операторот за доделување и **функција** за пресметка на износот што треба да се плати за регистрацијата.
Од оваа класа да се изведат класи за регистрирање на автомобили (**Reg_avto**) и камиони (**Reg_kamion**).
За регистрацијата на **автомобили** дополнително се чува информација за тоа дали учествувал во сообраќајни незгоди во изминатата година. При пресметување на вкупната цена за регистрација, на основната цена се додаваат 10% доколку зафатнината е поголема од 2000 cm3, а потоа се пресметува попуст од 5% доколку автомобилот не учествувал во сообраќајни незгоди изминатата година.
За регистрацијата на **камиони** дополнително се чува староста на возилото изразена во месеци (цел број). При пресметување на вкупната цена за регистрација, основната цена се зголемува за 4% на секои 6 години старост на возилото.
Да се напише функција за споредба на вкупната цена за регистрација која треба да се плати, за да работи со регистрации од било каков тип возила. Потоа да се напише функција која за дадена низа од регистрации ќе го испечати корисникот кој платил најмалку.

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

class Registracija
{
protected:
	string ime;
	int shasija;
	int zafatnina;
	float osCena;
	char **zemji;
	int brZemji;
public:
	Registracija(string i = "", int s = 0, int z = 0, float oc = 0, char ** zem =
NULL, int brZ = 0)
	{
		ime = i;
		shasija = s;
		zafatnina = z;
		osCena = oc;
		brZemji = brZ;
		zemji = new char*[brZemji];
		for (int i = 0; i < brZemji; i++)
		{
			zemji[i] = new char[strlen(zem[i]) + 1];
			strcpy(zemji[i], zem[i]);
		}
	}
	Registracija(const Registracija &r)
	{
		ime = r.ime;
		shasija = r.shasija;
		zafatnina = r.zafatnina;
		osCena = r.osCena;
		brZemji = r.brZemji;
		zemji = new char*[brZemji];
		for (int i = 0; i < brZemji; i++)
		{
			zemji[i] = new char[strlen(r.zemji[i]) + 1];
			strcpy(zemji[i], r.zemji[i]);
		}
	}
	Registracija & operator=(Registracija &r)
	{
		for (int i = 0; i < brZemji; i++)
			delete[] zemji[i];
		delete[] zemji;

		zemji = new char*[r.brZemji];
		for (int i = 0; i < r.brZemji; i++)
		{
			zemji[i] = new char[strlen(r.zemji[i]) + 1];
			strcpy(zemji[i], r.zemji[i]);
		}
		brZemji = r.brZemji;
		ime = r.ime;
		shasija = r.shasija;
		zafatnina = r.zafatnina;
		osCena = r.osCena;
		return *this;
	}

	virtual ~Registracija()
```

```
	{
		for (int i = 0; i < brZemji; i++)
			delete[] zemji[i];
		delete[] zemji;
	}
	virtual float iznos() = 0;
	friend void najmala(Registracija **r, int br); /* friend za da mozhe da go
pechatime imeto vo funkcijava */
};

class Reg_avto : public Registracija
{
	bool dali;
public:
	Reg_avto(string i = "", int  s = 0, int  z = 0, float  oc = 0, char  **  zem =
NULL, int  brZ = 0, bool d = 0) :Registracija(i, s, z, oc, zem, brZ), dali(d) {}
	Reg_avto(Registracija &r, bool d = 0) :Registracija(r), dali(d) {}
	Reg_avto(Reg_avto &ra) :Registracija(ra), dali(ra.dali) {}

	float iznos()
	{
		float vkupno = osCena;
		if (zafatnina > 2000)
			vkupno += (10 * vkupno) / 100;
		if (dali == 0)
			vkupno -= (5 * vkupno) / 100;
		return vkupno;
	}
};

class Reg_kamion : public Registracija
{
	int starost;
public:
	Reg_kamion(string i = "", int  s = 0, int  z = 0, float  oc = 0, char  **  zem =
NULL, int  brZ = 0, int st = 0) :Registracija(i, s, z, oc, zem, brZ), starost(st) {}
	Reg_kamion(Registracija &r, int st = 0) :Registracija(r), starost(st) {}
	Reg_kamion(Reg_kamion &rk) :Registracija(rk), starost(rk.starost) {}
	float iznos()
	{
		float vkupno = osCena;
		int kolku = starost / 12 / 6;
		vkupno += (osCena * 4) / 100 * kolku;
		return vkupno;
	}
};
bool sporedi(Registracija *r1, Registracija *r2)
{
	if (r1->iznos() < r2->iznos())
		return 1;
	else
		return 0;
}
void najmala(Registracija **r, int br)
{
	Registracija *min = r[0];
	for (int i = 1; i < br; i++)
	{
		if (sporedi(r[i], min))
			min = r[i];
	}
	cout << endl << min->ime << endl;
}

int main()
```

```
{
	char *s[2];
	s[0] = "drz1";
	s[1] = "drz2";
	Registracija *r[2];
	Reg_avto ra("Ime1", 2500, 3000, 1000, s, 2, 0);
	Reg_kamion rk("Ime2", 5000, 4000, 7000, s, 2, 100);
	r[0] = &ra;
	r[1] = &rk;
	najmala(r, 2);
}
```

2. Да се креира хиерархија на класи за водење евиденција за служби за достава. За секоја служба е познато името, брзина на достава на пакети (km/h) и капацитетот на достава по пратка. Службата за достава може да биде пакетна или приватна и овие класи да се изведат од основната. Да се напише функција за пресметка на капацитетот на службата и функција за пресметка на времето за достава на пакет (која добива како аргумент колку километри треба да се поминат за достава).
За пакетна служба дополнително се чува податок за тоа дали службата е во Скопје и дополнително дали има дополнителен багажник за големи пратки. За овој тип на служби, доколку не се во Скопје, капацитетот се поставува на вредност 0. Додека пак, доколку имаат багажник, брзината на движење се намалува за 15%, а капацитетот се зголемува за 10%.

За приватна служба дополнително се чува податок за дали доставата ќе биде изведена со помош на мотор и годината на производство на возилото за достава. За приватната служба, доколку доставата е со мотор, брзината на достава се зголемува за 60% но капацитетот се намалува за 50%. Дополнително, со секои 5 години старост на возилото, брзината се намалува за 10km/h.

Во секоја од класите да се дефинираат конструктори со подразбрани (default). Да се дефинираат и методи за печатење на сите информации за соодветните класи (во изведените класи да се повика и искористи методот за печатење на основната класа). Исто така за секоја класа да се дефинира функција vremeDostava(float km) со чија помош се враќа колку точно време ќе е потребно да се достави даден пакет кој е оддалечен km километри за достава. Надвор од класите да се напише функција najbrzo() која добива три аргументи: низа од покажувачи кон служба, должина на низата и број на километри кои треба да се извозат. Функцијата najbrzo() треба да ги испечати податоците на службата која најбрзо ќе го достави пакетот (независно дали е пакетна или приватна) а притоа има најмалку 10 капацитет. Доколку има повеќе служби што најбрзо ќе го достават пакетот, да се отпечатат податоците за сите.

```
#include <iostream>
#include <string.h>
#include <string>
#include <cmath>
using namespace std;
bool compare_float(float x, float y, float epsilon = 0.01f){
   if(fabs(x - y) < epsilon)
      return true; //they are same
      return false; //they are not same
}
class sluzba{
    private:
    string ime;
    float brzina;
    int kapacitet;
    public:
    sluzba(string i="",float b=0.0, int c=0){
        ime = i;
        brzina=b;
        kapacitet=c;
```

```
    }
    virtual float getKapacitet(){return kapacitet;}
    virtual float vremeDostava(int km){return km/brzina;}
    float getBrzina(){return brzina;}
    virtual void pecati(){
    cout << "ime : "<< ime << " brzina: "<< brzina << " km/h Kapacitet"<<getKapacitet()<<endl;
    }
};
class Paketna: public sluzba{
private:
bool skopje;
bool extra;
public:
Paketna(string i="",float b=0.0, int c=0,bool s=true,bool e=false):sluzba(i,b,c){
    skopje=s;
    extra=e;
}
 float getKapacitet(){if(!skopje){return 0;}
if(extra) return sluzba::getKapacitet()*1.1; else return sluzba::getKapacitet();
}
 float vremeDostava(int km){
    if(!extra){
        return sluzba::vremeDostava(km);
    }else{
      return  km/(getBrzina()*0.85);
    }
 }
 void pecati(){
    sluzba::pecati();
    cout << "Od skopje: "<< skopje << " ekstra bagaz: "<<extra<<endl;
 }
};
class Privatna: public sluzba{
    private:
    bool motor;
    int godina;
    public:
    Privatna(string i="",float b=0.0, int c=0,bool m=false,int g=2010):sluzba(i,b,c){
        motor=m;
        godina=g;
    }
float getKapacitet(){
    if(motor){return sluzba::getKapacitet()*0.5;}
    else{
      return  sluzba::getKapacitet();
    }
}
float vremeDostava(int km){
    float b=sluzba::getBrzina();
    if(motor){b=b*1.6;}
    int namali=((2023-godina)/5*10);
    b-=namali;
    return km/b;
}
 void pecati(){
    sluzba::pecati();
    cout << "Motor opcija: "<< motor << " Godina vozilo: "<<godina<<endl;
 }
};
void najbrzo(sluzba **niza, int n, int km){
   float vremeDostava = 0.0;
   for(int i = 0 ;i<n ;i++){
    if(niza[i]->getKapacitet()>=10){
        vremeDostava=niza[i]->vremeDostava(km);
        break;
    }
   }
    for(int i = 0 ; i < n; i++){
```

```
        if(niza[i]->getKapacitet()>=10&&vremeDostava>niza[i]->vremeDostava(km)){
          vremeDostava=niza[i]->vremeDostava(km);
        }
    }
    for(int i=0;i<n;i++){

        if(compare_float(vremeDostava,niza[i]->vremeDostava(km)) &&
                         niza[i]->getKapacitet()>=10)
         {
        niza[i]->pecati();
        }

    }
}
int main()
{
    sluzba *lista[4];
    Paketna P1("A123", 10, 15, true,false);
    Paketna P2("B123", 60, 20, true, true);
    Privatna PR1("V123", 50, 25,false,2021);
    Privatna PR2("V456", 80, 20,true,2015);

    lista[0] = &P1; lista[1] = &P2; lista[2] = &PR1; lista[3]=&PR2;
    najbrzo(lista,4,18);
    return 0;
}
```