Programabilni uređaji i objektno orjentisano programiranje

Računske vježbe 6

1. Realizovati klasu radnik koja ima četiri podatka člana i to:
   - koeficijent za platu (cijeli broj);
   - identifikacioni broj radnika (pokazivač na cijeli broj);
   - ime radnika (pokazivač na niz karaktera)
   - javni statički podatak koji će služiti za brojanje ukupnog broja radnika (objekata date klase).

   Klasa posjeduje konstruktor, destruktor i konstruktor kopije, kao i funkcije članice za pristup podacima članovima radi očitavanja i izmjene. Potrebno je realizovati i funkciju koja od dva radnika vraća ime radnika sa većim koeficijentom za platu.

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

class radnik
{
    private:
        int koef;
        int *id;
        char *ime;
    public:
         // pokazivace moramo inicijalizovati na 0 u podrazumijevanom konstruktoru
        radnik(){id=0; ime=0; broj++;}

         // standardni konstruktor sa argumentima koji ce inicijalizovati podatke clanove klase
        radnik(int, int, char *);

         // konstruktor kopije kao argument mora imati konstantnu refrencu na objekat iste klase
        radnik(const radnik &);

         // destruktor nema argumente niti rezultat
        ~radnik();

         // inspektori (geteri) - koristimo ih kada zelimo da vratimo vrijednost podatka clana
         // podaci clanovi klase su privatni i mogu im pristupati samo funkcije clanice klase!
         // da bismo u main-u odstampali podatke clanove klase neophodno je da pozovemo getere!
        int vratiKoef() {return koef;}
        int vratiId() {return *id;}
        char * vratiIme() {return ime;}

         // mutatori (seteri) - koristimo ih ako hocemo da promijenimo vrijednost podatka clana
        void promKoef(int a) {koef=a;}
        void promId(int a) {*id=a;}
        void promIme(char *a) {strcpy(ime,a);}

         // funkcija clanica koja kao argumente ima dva radnika - implicitni argument radnik (*this)
        // i eksplicitni argument radnik koji je proslijedjen u zagradama
        char * vecaPlata(radnik);

         // staticka promjenljiva je u ovom slucaju javni podatak clan klase, mogu joj pristupati
         // svi kreirani objekti klase i mijenjati njenu vrijednost (moze se shvatiti kao globalna
         // promjenljiva klase)
        static int broj;
};

// staticku promjenljivu je obavezno, van klase, inicijalizovati na neku pocetnu vrijednost !
int radnik::broj=0;
```

```
// realizacija standardnog konstruktora
// za svaki pokazivac clan klase je neophodno dinamicki zauzeti memoriju!

// nakon naredbe id=new int(b); id ce biti pokazivac koji pokazuje na memorijsko mjesto u
// kojem je smjesten jedan cijeli broj sa vrijednoscu b

// nakon naredbe ime=new char[strlen(c)]; ime ce biti pokazivac na prazno memorijsko mjesto cija
// je duzina jednaka duzini stringa c
// tek nakon zauzete memorije mozemo u ime smjestiti vrijednost stringa c pomocu funkcije strcpy

// obicne zagrade kod new zauzimaju jedno memorijsko mjesto a uglaste zagrade zauzimaju vise
// memorijskih mjesta (onoliko koliko navedemo u [])

// staticku promjenljivu inkrementiramo jer smo izvrsavanjem konstruktora kreirali novog radnika
// u mamoriji racunara

// imati na umu da podaci koef, id i ime pripadaju objektu *this koji je pozvao ovu funkciju
// ispred svakog podatka koef, id ili ime mozemo pisati (*this).koef, (*this).id ili (*this).ime
// objekat (*this) se ne navodi samo zbog kraceg zapisa
// svaka funckija clanica uvijek ima implicitni objekat *this koji je jedino moze i pozvati

radnik::radnik(int a, int b, char *c):koef(a)
{
    id=new int(b); //id=new int; *id=b;
    ime=new char[strlen(c)];
    strcpy(ime,c);
    broj++;
}

// realizacija konstruktora kopije
// cilj konstruktora kopije je da kreira novi objekat u memoriji racunara koji ce dobiti iste

// vrijednosti podataka clanova kao neki vec postojeci objekat klase radnik (taj postojeci objekat
// je proslijedjen kao argument)

// radimo iste stvari ako u standardnom konstruktoru a kao vrijednosti podataka clanova
// novokreiranog objekta smjestamo podatke clanove objekta koji je argument funkcije

// staticku promjenljivu uvecavamo jer se kreiranjem novog radnika povecao njihov ukupan broj

radnik::radnik(const radnik &a):koef(a.koef)
{
    id=new int(*a.id); //id=new int; *id=*a.id;
    ime=new char[strlen(a.ime)];
    strcpy(ime,a.ime);
    broj++;
}

// realizacija destruktora
// destruktor sluzi da izbrise sve podatke clanove objekta klase
// potrebno je brisati samo pokazivace clanove klase (ukoliko ih klasa posjeduje)
// delete id; brise vrijednost na koju pokazivac pokazuje
// id=0; brise sam pokazivac i on nakon toga ne pokazuje vise ni na sta

radnik::~radnik()
{
    delete id; id=0;
    delete ime; ime=0;
    broj--;
}
```

```
// funkcija vecaPlata() pristupa koeficijentima plate dva radnika (radnik *this i radnik a) I kao
// rezultat vraca ime radnika koji ima veci koeficijent za platu

char * radnik::vecaPlata(radnik a)
{
    if(koef>=a.koef)
        return ime;
    else
        return a.ime;
}


int main()
{
    int a,b;
    char c[20];

    cout<<"Unesete podatke za prvog radnika"<<endl;
    cin>>a>>b>>c;
    // poziv standardnog konstruktora
    radnik rad1(a,b,c);

    cout<<"Unesete podatke za drugog radnika"<<endl;
    cin>>a>>b>>c;
    // poziv standardnog konstruktora
    radnik rad2(a,b,c);

    // poziv konstruktora kopije (kreiran novi radnik rad3 sa istim vrijednostima kao rad2)
    radnik rad3(rad2);

    // poziv inspektorskih funkcija (getera) jer jedino tako mozemo pristupiti podacima clanovima
    // objekata rad1, rad2 ili rad3
    cout<<"Sada radnik 3 ima ime "<<rad3.vratiIme()<<endl;

    // poziv funkcije clanice vecaPlata() - obavezno je mora pozvati neki od kreiranih objekata
    cout<<"Vise je placen radnik "<<rad1.vecaPlata(rad2)<<endl;

    // statickoj promjenljivoj klase se pristupa navodjenjem naziva klase i opreatora dosega (::)
    // staticka promjenljiva pripada klasi a ne objektima!
    // radnik::broj u ovom trenutku ima vrijednost 3 jer se staticka promjenljiva tri puta
    // inkrementirala (pri pozivu tri konstruktora za radnike rad1, rad2 i rad3)
    cout<<"Ukupno je kreirano "<<radnik::broj<<" radnika."<<endl;

}
// Napomena: Podrazumijevani konstruktor (bez argumenata), konstruktor kopije i destruktor su
// funkcije clanice koje nam kompajler podrazumijevano kreira, za svaku klasu koju realizujemo.
// Ukoliko nemamo pokazivace kao clanove klase onda je nepotrebno kreirati pomenute funkcije jer
// ce ih kompajler sam realizovati u pozadini programa i omoguciti da ih koristimo u main funkciji.
// Ukoliko imamo pokazivace kao podatke clanove klase, predlog je da uvijek sami realizujete svoje
// verzije za sve tri pomenute funkcije jer kompajler ne vrsi dinamicku alokaciju memorije za
// pokazivace! U sljedecem primjeru pokazano je kako podrazumijevani konstruktor kopije, koji nam
// obezbjedjuje kompajler moze lose uticati na kreiranje kopija objekata klase radnik.

// radnik A(12, 1, “Marko Markovic”);
// radnik B(A);

// Ako nemamo nasu realizaciju konstruktora kopije, onda nema ni dinamicke alokacije memorije za
// podatke clanove novog objekta B, pa to znaci da ce objekat B imati iste, iskopiran pokazivac
// id kao objekat A. Ako promijenimo id objekta A promijenice se i id objekta B. To nikako ne //
// smijemo da dozvolimo! Zbog toga relizujemo nas konstrutkor kopije u kojem se dinamicki zauzima
// memorija za podatke novog objekta B i time obezbjedjujemo da je on nezavisan od memorijskih
// adresa (pokazivaca) objekta A.
```