

# Objektno orijentisano programiranje

Rad sa listama

# Podsećanje

Jednostruko spregnuta lista je dinamička struktura podataka koja se sastoji od skupa čvorova koji predstavljaju njene elemente i koji su povezani pokazivačima u jednom smeru. Svaki čvor liste je strukturna promenljiva koja ima dva polja: prvo polje koje čuva podatak i drugo polje koje predstavlja pokazivač na sledeći čvor liste.

Lista koju ćemo koristiti pri rešavanju zadatka je implementirana kao generička klasa i iz tog razloga u njoj možemo čuvati podatke bilo kog tipa.

Prilikom rešavanja narednih zadataka podrazumevamo da klasu **List** dobijamo već napravljenu i da samo pozivamo njene metode.

# Metode klase List

Metoda koja vraća broj elemenata u listi.

```
32      int size() const { return noEl; }
```

Metoda za proveru da li je lista prazna.

```
34      bool empty() const { return head == NULL ? 1 : 0; }
```

Metoda za dodavanje elementa u listu. Prvi parametar je pozicija na koju se element dodaje, drugi parametar je element koji se dodaje.

```
36      bool add(int, const T&);
```

# Metode klase List

Metoda za brisanje elementa iz liste. Parametar je pozicija elementa koji se briše.

```
38      bool remove(int);
```

Metoda za iščitavanje elementa iz liste. Prvi parametar je pozicija elementa koji se čita, drugi parametar je referenca na promenljivu u koju se smešta pročitana vrednost.

```
40      bool read(int, T&) const;
```

Metoda za brisanje svih elemenata iz liste.

```
42      void clear();
```

# Primer 1

1. Napisati klasu **Osoba** koja ima polja ime i prezime. Iz klase **Osoba** izvesti klasu **Student** koja ima dodatno polje za broj indeksa. Iz klase **Osoba** izvesti i klasu **Profesor** koja dodatno ima i naziv predmeta koji predaje. Napisati klasu **Učionica** koja ima listu osoba koji su u njoj prisutne. Učionica ima svoj naziv i maksimalan broj osoba koje mogu da stanu u nju. Napisati kratak test program.

(videti tekst zadatka u folderu 1)

# Rešenje: 1.1. Kreirati klasu Osoba - polja

Napomena: polja ime i prezime su tipa DinString, moramo uključiti dinstring.hpp



```
main.cpp X  osoba.hpp X
1  #ifndef OSOBA_DEF
2  #define OSOBA_DEF
3
4  #include "dinstring.hpp"
5
6  class Osoba {
7      protected:
8          DinString ime, prezime;
9      public:
```

## Rešenje: 1.2. Kreirati klasu Osoba - konstruktori

- Potrebno je kreirati konstruktor bez parametara, konstruktor sa parametrima i konstruktor kopije.
- Preporuka je da se odmah nakon kreiranja konstruktora i ostalih metoda u fajlu *osoba.hpp* izvrši njihovo testiranje u fajlu *main.cpp* (kreiranjem objekta ili pozivom metode u *main* funkciji).
- Na taj način smanjujemo mogućnost od gomilanja grešaka.

# Rešenje: 1.2. Kreirati klasu Osoba - konstruktori

- Na isti način kreiramo i ostale potrebne konstruktore i odmah ih pozivamo u *main* funkciji.
- *osoba.hpp*:

```
public:
    ///metode
    Osoba() {}
    Osoba(const DinString &i, const DinString &p) : ime(i), prezime(p) {}
    Osoba(const Osoba &osoba) : ime(osoba.ime), prezime(osoba.prezime) {}
```

- *main.cpp*:

```
int main()
{
    cout << "Testiranje klase Osoba" << endl << endl;

    Osoba o;
    Osoba o1 ("Petar", "Petrovic");
    Osoba o2(o1);
}
```



# Rešenje: 1.3. Kreirati klasu Osoba - virtuelna metoda za ispis

- Napomena:

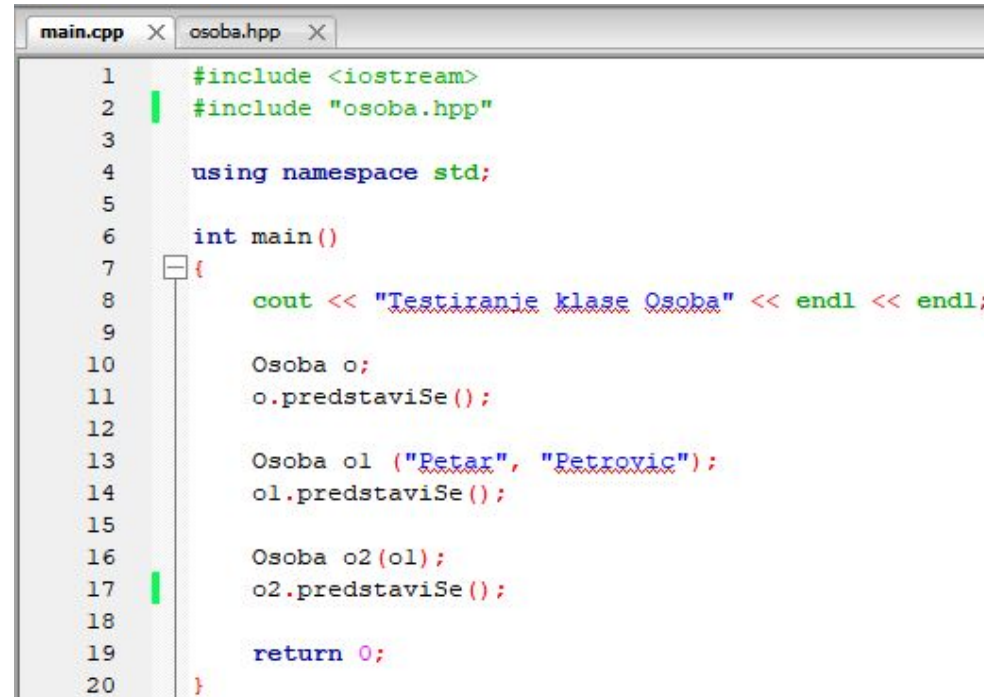
Metoda se proglašava virtuelnom u roditeljskoj klasi tako što se ispred tipa metode navede rezervisana reč **virtual**

- osoba.hpp

```
virtual void predstaviSe() const {  
    cout << "Osoba: ";  
    if (ime == "" && prezime == "")  
        cout << "x" << endl;  
    else  
        cout << "Zovem se " << ime << " " << prezime << "." << endl ;  
}
```

# Rešenje: 1.3. Kreirati klasu Osoba - virtualna metoda za ispis

- Testiranje metode *predstaviSe()*
- *main.cpp*



```
1  #include <iostream>
2  #include "osoba.hpp"
3
4  using namespace std;
5
6  int main()
7  {
8      cout << "Testiranje klase Osoba" << endl << endl;
9
10     Osoba o;
11     o.predstaviSe();
12
13     Osoba o1 ("Petar", "Petrovic");
14     o1.predstaviSe();
15
16     Osoba o2(o1);
17     o2.predstaviSe();
18
19     return 0;
20 }
```

## Rešenje: 2. Kreirati klasu Student

- Kreirati *student.hpp* i u njemu pisati kod za klasu **Student**.
- Klasa **Student** nasleđuje klasu **Osoba** i proširuje je poljem brojIndeksa.
- U klasi **Student** potrebno je implementirati konstruktor bez parametara, konstruktor sa parametrima, konstruktor kopije i preklopiti metodu za ispis.

# Rešenje: 2.1. Kreirati klasu Student

- Poželjno je nakon svake implementirane metode istu pozvati u *main.cpp*.
- Ne zaboraviti uključiti *osoba.hpp* jer se tu nalazi klasa koju nasleđujemo.

```
main.cpp x osoba.hpp x student.hpp x
1  #ifndef STUDENT_HPP_INCLUDED
2  #define STUDENT_HPP_INCLUDED
3
4  #include "osoba.hpp"
5
6  class Student : public Osoba {
7      protected:
8          int brojIndeksa;
9
10     public:
11         Student() : Osoba(), brojIndeksa(0) {}
12         Student(const DinString &i, const DinString &p, int bri) : Osoba(i, p), brojIndeksa(bri) {}
13         Student(const Student& s) : Osoba((Osoba)s), brojIndeksa(s.brojIndeksa) {}
14
15         void predstaviSe() const {
16             cout << "Student: ";
17             if (ime == "" && prezime == "")
18                 cout << "x" << endl;
19             else
20                 cout << "Zovem se " << ime << " " << prezime << ", a broj mog indeksa je " << brojIndeksa << "." << endl;
21         }
22     };
23
24 #endif // STUDENT_HPP_INCLUDED
```

## Rešenje: 2.1. Kreirati klasu Student - testiranje u main.cpp

Napomena: Da bismo mogli testirati klasu **Student** moramo uključiti *student.hpp* u *main.cpp*-u. Pošto je u *student.hpp*-u uključen *osoba.hpp*, možemo iz *main.cpp*-a izbrisati *osoba.hpp*.

```
main.cpp x osoba.hpp x student.hpp x
1  #include <iostream>
2  #include "student.hpp"
3
4  using namespace std;
5
6  int main()
```

```
20  cout << "Testiranje klase Student" << endl << endl;
21
22  Student s;
23  s.predstaviSe();
24
25  Student s1("Marko", "Markovic", 12345);
26  s1.predstaviSe();
27
28  Student s2(s1);
29  s2.predstaviSe();
```

## Rešenje: 3. Kreirati klasu Profesor

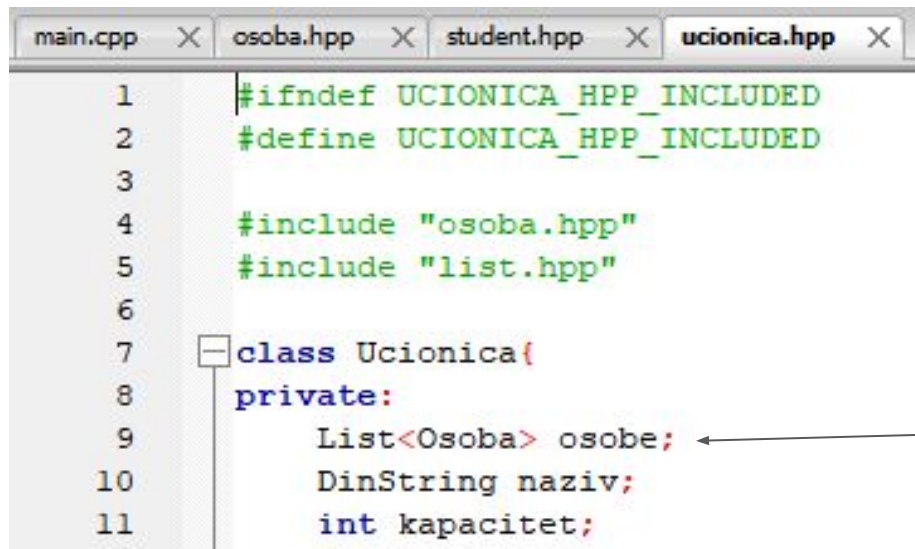
- Kreirati *profesor.hpp* i u njemu pisati kod za klasu **Profesor**.
- Klasa **Profesor** nasleđuje klasu **Osoba** i proširuje je poljem predmet
- Uraditi korake kao za klasu **Student**.
- Rešenje se nalazi u folderu 1, u fajlu *profesor.hpp*.
- Uključiti *profesor.hpp* u *main.cpp*-u i testirati ga kao što je urađeno za klasu **Student**.

## Rešenje: 4. Kreirati klasu Ucionica

- Kreirati *ucionica.hpp* i u njemu pisati kod za klasu **Ucionica**.
- Učionica ima svoj naziv, broj osoba koje mogu da stanu u nju i listu osoba.
- Za učionicu je potrebno implementirati konstruktor sa parametrima, metodu za dodavanje osobe u učionicu kao i metodu za ispis osoba u učionici.
- Uključiti *list.hpp* i *osoba.hpp* u *ucionica.hpp*-u.

# Rešenje: 4.1. Ucionica - polja

- Klasa **Ucionica** sadrži polje osobe koje predstavlja listu osoba.
- Pošto je **List** generička klasa potrebno je unutar <> napisati tip podataka koji će se čuvati u listi.



```
1  #ifndef UCIONICA_HPP_INCLUDED
2  #define UCIONICA_HPP_INCLUDED
3
4  #include "osoba.hpp"
5  #include "list.hpp"
6
7  class Ucionica{
8  private:
9      List<Osoba> osobe;
10     DinString naziv;
11     int kapacitet;
```

Na ovaj način kreiramo listu koja u sebi može da sadrži osobe



# Rešenje: 4.1. Učionica - metoda za dodavanje osoba u učionicu

- Dodavanje osobe u učionicu je moguće samo ako u učionici ima mesta.
- Trenutni broj elemenata liste (broj osoba u listi) dobijamo pozivom metode *size()* koja je implementirana u klasi **List**.
- Indeksi u listi koju koristimo kreću od 1.
- Prilikom dodavanja osobe u učionicu koristimo metodu *add* iz klase **List**. Kao prvi parametar ova metoda prima poziciju na koju dodajemo - u ovom slučaju dodajemo na kraj liste što je trenutni broj osoba u listi +1. Drugi parametar metode *add* je ono šta dodajemo u listu, u našem slučaju osoba koju smo prosledili kao parametar u metodi.

```
bool dodaj(const Osoba& os){  
    if(osobe.size() < kapacitet){  
        return osobe.add(osobe.size() + 1, os);  
    }  
  
    return false;  
}
```

Metoda *add* kao povratnu vrednost vraća *bool* koji nam govori da li je uspešno dodavanje.

# Rešenje: 4.1. Ucionica - metoda za dodavanje osoba u učionicu

- Nakon što implementiramo metodu preporuka je da je odmah testiramo u *main.cpp*-u.
- Da bismo mogli testirati metode iz *ucionica.hpp*-a potrebno ga je uključiti u *main.cpp*.

```
cout << endl << endl;  
cout << "Testiranje klase Ucionica" << endl << endl;
```

```
Ucionica u("FTN-U1", 5);  
u.ispis();
```

```
if(u.dodaj(pl)) {  
    cout << "Uspesno dodavanje." << endl;  
} else {  
    cout << "Neuspesno dodavanje." << endl;  
}
```

Metoda dodaj kao povratnu vrednost vraća bool koji nam govori da li je uspešno dodavanje, iz tog razloga imamo if else kako bismo mogli da pokrijemo oba slučaja.

Neuspešno dodavanje možemo testirati ako pokušamo da dodamo 6 osoba u učionicu u koju staje 5 osoba.

# Rešenje: 4.1. Ucionica - metoda za ispis osoba u učionici

- Ukoliko je učionica prazna (broj elemenata u listi je 0) ispisujemo poruku: "Ucionica je prazna"
- Ukoliko u učionici ima osoba, iteriramo for petljom i iščitavamo svaku osobu i nad njom pozivamo metodu *predstaviSe()*

```
void ispis() const {  
    if(osobe.size() == 0) {  
        cout << endl << "Ucionica je prazna." << endl;  
    } else {  
        Osoba o;  
        cout << endl << "U ucionici su:" << endl;  
  
        for (int i = 1; i <= osobe.size(); i++){  
            osobe.read(i, o);  
            o.predstaviSe();  
        }  
    }  
}
```

Za iščitavanje iz liste koristimo metodu *read* čiji je prvi parametar pozicija sa koje iščitavamo, a drugi parametar promenljiva u koju smeštamo pročitanu vrednost.

# Rešenje: 4.1. Ucionica - metoda za ispis osoba u učionici

- Nakon što implementiramo metodu preporuka je da je odmah testiramo u *main.cpp*-u

87

`u.ispis();`

# Zadatak 1-1.

Izmeniti zadatak 1 tako da se u učionici ne čuvaju objekti tipa **Osoba**, već pokazivači na osobe.

# Rešenje - Osoba, Student i Profesor

- Klase **Osoba**, **Student** i **Profesor** su identične kao u prošlom zadatku
- Ne zaboraviti dodati fajlove *dinstring.hpp*, *dinstring.cpp* i *list.hpp* u projekat

# Rešenje - Ucionica - polja

- Klasa **Ucionica** sada kao polje sadrži listu pokazivača na osobe
- *osoba.hpp*

```
class Ucionica{  
private:  
    List<Osoba*> osobe; //promena  
    DinString naziv;  
    int kapacitet;
```

# Rešenje - Ucionica - metoda za dodavanje osobe

Pošto radimo sa pokazivačima, prilikom dodavanja u listu potrebno je da stavimo &. Time dodajemo adresu objekta klase **Osoba** u listu.

Napomena: Neophodno je da izbacimo const koje se nalazilo ispred parametra metode.

```
bool dodaj(Osoba& os){  
    if(osobe.size() < kapacitet){  
        return osobe.add(osobe.size() + 1, &os); // promena  
    }  
  
    return false;  
}
```



# Rešenje - Ucionica - metoda za ispis osoba u učionici

```
void ispis() const {  
    if(osobe.size() == 0) {  
        cout << endl << "Ucionica je prazna." << endl;  
    } else {  
        Osoba* o; // promena  
        cout << endl << "U ucionici su:" << endl;  
  
        for (int i = 1; i <= osobe.size(); i++){  
            osobe.read(i, o);  
            o -> predstaviSe(); //promena  
        }  
    }  
}
```

Iz liste ćemo iščitavati pokazivače na osobe, iz tog razloga pravimo lokalnu promenljivu koja je pokazivač na osobu.

Da bismo mogli da pozovemo metodu *predstaviSe* neophodno je da dereferenciramo pokazivač na objekat klase **Osoba**.  
Drugi način da ovo uradimo bio bi: `(*o).predstaviSe();`

# Poređenje zadatka 1 i zadatka 1.1

Prilikom pokretanja rešenja zadatka 1 i zadatka 1.1 uočljiva je razlika u onome šta se prikazuje prilikom pozivanja metode *ispis()* u klasi **Ucionica**

U zadatku 1 radimo bez pokazivača. U tom slučaju kada pozovemo metodu za ispis osoba u učionici pozove nam se metoda *predstaviSe()* iz klase **Osoba**, bez obzira da li se radi o osobi, studentu ili profesoru. Prilikom instanciranja učionice mi radimo sa listom objekata klase **Osoba** i samim tim jedino šta može da se sačuva u listi je osoba. U slučaju da u listu dodamo objekat klase **Student** ili objekat klase **Profesor** biće dodat samo njihov roditeljski deo odnosno osoba.

Ako radimo sa pokazivačima nama se u listi sačuvaju samo adrese objekata. Preko sačuvane adrese imamo pristup pravom objektu (osobi, studentu ili profesoru), tako da se prilikom ispisa poziva metoda *predstaviSe()* iz odgovarajuće klase.

## Zadatak 2

Napisati apstraktnu klasu **Artikal** koja sadrži apstraktnu metodu za izračunavanje vrednosti artikla (vraća double).

Iz klase **Artikal** izvesti klasu **Lek**, koja sadrži polja: jkl (long), naziv (DinString), jedinicznaCena (double) i kolicina (int). Implementirati konstruktore i preklopiti operator za ispis.

Napisati klasu **EvidencijaLekova**, koja sadrži polja: nazivApoteke (DinString), datum (DinString) i lekovi (List<Lek>). U klasi implementirati: konstruktor bez parametara, metodu za unos novog leka u evidenciju, metodu za dodavanje leka u evidenciju, metodu za brisanje leka iz evidencije, metodu za sortiranje evidencije po vrednosti leka i operator za ispis.

(videti tekst zadatka u folderu 2)

# Rešenje - klasa Artikal

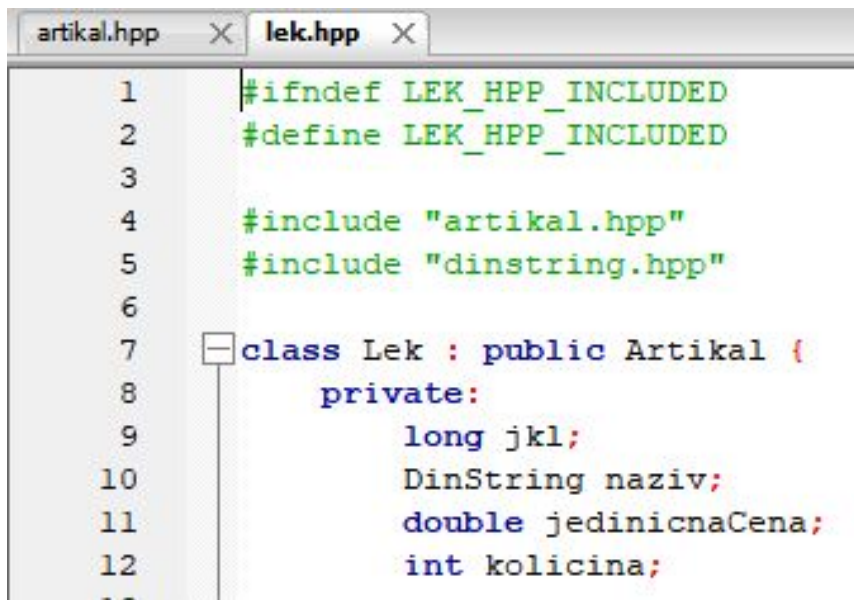
Kreiramo apstraktnu klasu **Artikal** koja ima jednu apstraktnu metodu *getVrednost*

```
1  #ifndef ARTIKAL_HPP_INCLUDED
2  #define ARTIKAL_HPP_INCLUDED
3
4  #include <iostream>
5  using namespace std;
6
7  class Artikal{
8      public:
9          virtual double getVrednost() const = 0;
10 };
11
12 #endif // ARTIKAL_HPP_INCLUDED
```

Napomena: Pošto je klasa **Artikal** apstraktna, ne pokušavati napraviti objekat ove klase jer će to prouzrokovati grešku!

# Rešenje - klasa Lek

- Dodamo fajl *lek.hpp* u kojem pišemo klasu **Lek**
- Klasa **Lek** nasleđuje klasu **Artikal**



The screenshot shows a code editor with two tabs: 'artikal.hpp' and 'lek.hpp'. The 'lek.hpp' tab is active, displaying the following C++ code:

```
1  #ifndef LEK_HPP_INCLUDED
2  #define LEK_HPP_INCLUDED
3
4  #include "artikal.hpp"
5  #include "dinstring.hpp"
6
7  class Lek : public Artikal {
8      private:
9          long jkl;
10         DinString naziv;
11         double jedinicznaCena;
12         int kolicina;
```

# Rešenje - klasa Lek

Klasa **Lek** pored konstruktora mora da sadrži i implementaciju metode *getVrednost*

```
double getVrednost() const {  
    return jedinicznaCena * kolicina;  
}
```

Takođe, u klasi **Lek** potrebno je preklopiti operator za ispis

```
friend ostream& operator<<(ostream& out, const Lek& lek) {  
    out << "LEK [ jkl: " << lek.jkl << ", n: " << lek.naziv << ", jc: " << lek.jedinicznaCena << ", kol: " << lek.kolicina << " ]";  
    return out;  
}
```

Nakon implementacije svake metode obavezno je istestirati u *main.cpp*-u

# Rešenje - testiranje klase Lek

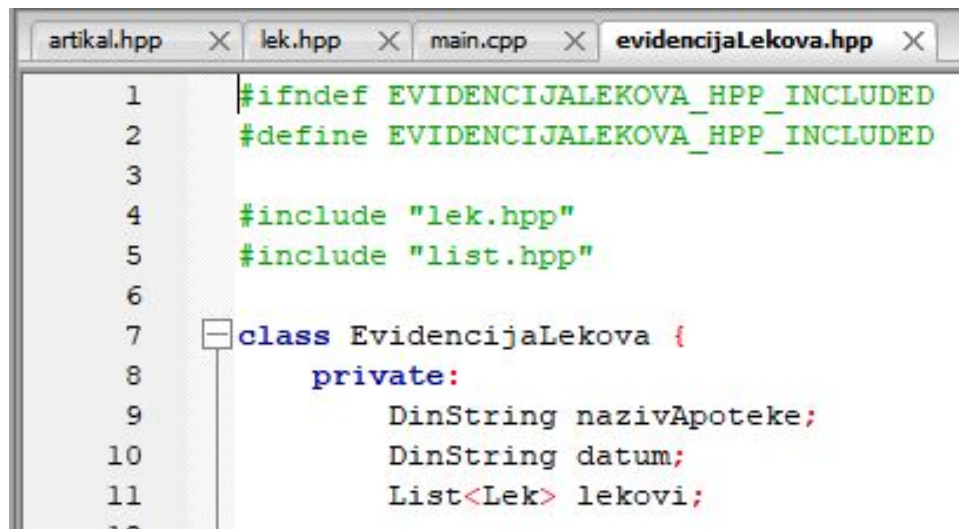
```
int main()
{
    cout << "Testiranje klase Lek" << endl << endl;

    DinString d("Pressing");
    Lek lek1, lek2(2, "Aspirin", 22.33, 10), lek3(3, d, 34.56, 20), lek4(lek3);

    cout << lek1 << endl;
    cout << lek2 << endl;
    cout << lek3 << endl;
    cout << lek4 << endl;
}
```

# Rešenje - klasa EvidencijaLekova

Evidencija lekova sadrži listu objekata klase **Lek**



```
artikal.hpp x lek.hpp x main.cpp x evidencijaLekova.hpp x
1  #ifndef EVIDENCIJALEKOVA_HPP_INCLUDED
2  #define EVIDENCIJALEKOVA_HPP_INCLUDED
3
4  #include "lek.hpp"
5  #include "list.hpp"
6
7  class EvidencijaLekova {
8      private:
9          DinString nazivApoteke;
10         DinString datum;
11         List<Lek> lekovi;
```



# Rešenje - EvidencijaLekova operator ispisa

```
friend ostream& operator<<(ostream& out, const EvidencijaLekova& evidencija) {
    out << endl << "*****" << endl;
    out << "Naziv apoteke: " << evidencija.nazivApoteke << " datum: " << evidencija.datum << endl;

    if(evidencija.lekovi.empty()){
        out << "Nema lekova u apoteci." << endl;
    }
    else {
        /// 1. nacin
        /// out << evidencija.lekovi;

        /// 2. nacin
        Lek lek;

        for(int i = 1; i <= evidencija.lekovi.size(); i++){
            evidencija.lekovi.read(i, lek);
            out << lek << endl;
        }
    }

    out << "*****" << endl << endl;
    return out;
}
```

Jedan način je pozivanje operatora ispisa koji je implementiran u klasi **List**

Drugi način je iščitavanje svakog elementa iz liste, a zatim pozivanje operatora ispisa nad njim.

# Rešenje - klasa EvidencijaLekova

Metoda za kreiranje leka na osnovu prosleđenih parametara i dodavanje kreiranog leka u listu lekova. Lek se dodaje na poslednje mesto u listi. Za dodavanje u listu koristimo metodu *add* iz klase **List**.

```
bool dodajLek(long jkl, const char naziv[], int jedinicznaCena, int kolicina) {  
    Lek lek(jkl, naziv, jedinicznaCena, kolicina);  
    return lekovi.add(lekovi.size() + 1, lek);  
}
```

Metoda za dodavanje postojećeg leka u listu lekova. Postojeći lek se šalje po referenci. Lek se dodaje na prvo mesto u listi. Za dodavanje u listu koristimo metodu *add* iz klase **List**

```
bool dodajLek(const Lek& lek) {  
    return lekovi.add(1, lek);  
}
```

# Rešenje - testiranje klase EvidencijaLekova

```
EvidencijaLekova e;  
|cout << "Dodavanje leka preko parametara" << endl;  
  
// učitavanje parametara  
long jkl;  
char naziv[100];  
int jedinicnaCena, kolicina;  
  
cout << "Unesite jedinstveni ključ leka (sifru leka): ";  
cin >> jkl;  
  
cout << "Unesite naziv leka: ";  
cin >> naziv;  
  
cout << "Unesite jedinичnu cenu leka: ";  
cin >> jedinicnaCena;  
  
cout << "Unesite kolicinu leka: ";  
cin >> kolicina;  
  
cout << endl;  
if(e.dodajLek(jkl, naziv, jedinicnaCena, kolicina)){  
    cout << "Lek uspesno dodat." << endl;  
    cout << e;  
} else {  
    cout << "Greska prilikom dodavanja leka." << endl;  
}
```

# Rešenje - testiranje klase EvidencijaLekova

```
cout << endl << endl;
cout << "Dodavanje vec napravljenog leka " << endl;

if(e.dodajLek(lek2)){
    cout << "Lek (" << lek2 << ") uspesno dodat." << endl;
    cout << e;
} else {
    cout << "Greska prilikom dodavanja leka (" << lek2 << ")." << endl;
}
```

# Rešenje - Evidencija Lekova

Metoda za uklanjanje leka iz evidencije. Kao parametar prima poziciju u listi leka koji se briše. Za brisanje se poziva metoda *remove* iz klase **List** koja kao parametar prima poziciju elementa koji se briše. Metoda *remove* kao povratnu vrednost vraća *true* ako je brisanje uspešno, odnosno *false* ako nije.

```
bool ukloniLek(int position){  
    return lekovi.remove(position);  
}
```

Testiranje uklanjanja leka (*main.cpp*). Preporuka je da se testira sa indeksom koji postoji u listi i sa indeksom koji ne postoji.

```
if(e.ukloniLek(1)){  
    cout << "Lek na poziciji 1 uspesno uklonjen." << endl;  
    cout << e;  
} else {  
    cout << "Greska prilikom uklanjanja leka na poziciji 1." << endl;  
}
```

# Rešenje - Evidencija Lekova

## Sortiranje liste

```
void sortirajLekove() {
```

```
    Lek l1, l2; ← Lokalne promenljive koje koristimo da u njih iščitamo lekove iz liste
```

```
    for(int i = 1; i < lekovi.size(); i++) {
```

```
        for(int j = i + 1; j <= lekovi.size(); j++) {
```

```
            lekovi.read(i, l1); ← Iščitavamo lek sa pozicije i i smeštamo ga u promenljivu l1
```

```
            lekovi.read(j, l2); ← Iščitavamo lek sa pozicije j i smeštamo ga u promenljivu l2
```

```
            if(l1.getVrednost() > l2.getVrednost()) { ← Ako je potrebno izvršiti zamenu (vrednost prvog leka veća od vrednosti drugog)
```

```
                lekovi.remove(i);
```

```
                lekovi.add(i, l2);
```

```
                lekovi.remove(j); ← Na i-tu poziciju smeštamo lek koji je bio na j-toj poziciji
```

```
                lekovi.add(j, l1); ← Potrebno je sa j-te pozicije ukloniti lek
```

```
            } ← Na j-tu poziciju smeštamo lek koji je bio na i-toj poziciji
```

```
        }
```

```
    }
```

```
}
```

```
}
```

Testirati sortiranje pozivom funkcije u *main.cpp*-u `e.sortirajLekove();`

# Rešenje - Evidencija Lekova

Kompletno rešenje zadatka nalazi se u folderu sa nazivom 2.

## Zadatak 3

Napisati klasu **Test** koja ima polja naziv (DinString) i osvojeniPoeni (int). Klasa **Test** treba da sadrži apstraktne metode *int getPoeni() const* i *bool položio() const*. U klasi implementirati konstruktore sa i bez parametara i metodu *void ispis()* koja ispisuje naziv testa i broj osvojenih poena.

Klasu **Test** nasleđuje klasa **Kolokvijum**. U klasi **Kolokvijum** implementirati: konstruktore sa i bez parametara, metode *int getPoeni() const* i *bool položio() const*.

Napisati klasu **Predmet** koja ima polja: naziv (DinString) i testovi (List<Test\*>). Implementirati sva tri konstruktora i metode: *void dodajTest(Test& t)*, *bool položio()*, *DinString getNaziv()*, *void ispis()*



## Zadatak 3

Napisati klasu **Student** koja u sebi ima dva polja predmet. U klasi implementirati konstruktor sa parametrima za dva predmeta kao i metodu *void ispis()* koja treba da ispiše informacije o svakom testu koji je student uradio iz oba predmeta, kao i naziv predmeta iz kog je osvojio više poena na testovima

(videti tekst zadatka u folderu 2)

# Rešenje - klasa Test

Napomena: Pošto je klasa **Test** apstraktna, ne pokušavati napraviti objekat ove klase jer će to prouzrokovati grešku!

```
test.hpp x
1  #ifndef TEST_HPP_INCLUDED
2  #define TEST_HPP_INCLUDED
3
4  #include <iostream>
5  using namespace std;
6  #include "dinstring.hpp"
7
8  class Test{
9      protected:
10         DinString naziv;
11         int osvojeniPoeni;
12
13     public:
14         Test(const char* d = "", int os = 0) : naziv(d), osvojeniPoeni(os) {}
15
16         virtual int getPoeni() const = 0;
17         virtual bool položio() const = 0;
18
19         void ispis() const{
20             cout << "Naziv: " << naziv << " ; Osvojeni poeni: " << osvojeniPoeni << endl;
21         }
22     };
23
24 #endif // TEST_HPP_INCLUDED
```

# Rešenje - klasa Kolokvijum

```
test.hpp x *kolokvijum.hpp x
1  #ifndef KOLOKVIJUM_HPP_INCLUDED
2  #define KOLOKVIJUM_HPP_INCLUDED
3
4  #include <cstdlib>
5  #include "test.hpp"
6
7  class Kolokvijum : public Test {
8      public:
9          Kolokvijum() : Test() {}
10         Kolokvijum(const char* n) : Test(n, rand()%25 + 1) {}
11
12         int getPoeni() const {
13             return osvojeniPoeni;
14         }
15
16         bool položio() const {
17             return (osvojeniPoeni > 12);
18         }
19     };
20
21 #endif // KOLOKVIJUM_HPP_INCLUDED
```

Napomena: Da bismo mogli iskorisiti funkciju *rand()* moramo uključiti biblioteku *cstdlib*

# Rešenje - testiranje klase Kolokvijum

```
test.hpp x kolokvijum.hpp x main.cpp x
1  #include "student.hpp"
2  #include <ctime>
3
4  int main()
5  {
6      srand(time(NULL));
7
8      cout << "Testiranje klase Kolokvijum" << endl << endl;
9      Kolokvijum k, k1("K1"), k2("K2"), k3("K3"), k4("K4");
10
11     k.ispis();
12     cout << "Polozio: " << (k.polozio() ? "DA" : "NE") << endl << endl;
13
14     k1.ispis();
15     cout << "Polozio: " << (k1.polozio() ? "DA" : "NE") << endl << endl;
```

Pošto koristimo funkciju *rand()*, da bi broj koji nam ova funkcija vraća bio zaista slučajan broj, pozivamo funkciju *srand* i iz tog razloga moramo uključiti biblioteku *ctime*

# Rešenje - klasa Predmet

```
test.hpp x kolokvijum.hpp x main.cpp x predmet.hpp x
1  #ifndef PREDMET_HPP_INCLUDED
2  #define PREDMET_HPP_INCLUDED
3
4  #include "kolokvijum.hpp"
5  #include "list.hpp"
6
7  class Predmet {
8      private:
9          DinString naziv;
10         List<Test*> testovi;
11
12     public:
13         Predmet(const char* n = "") : naziv(n) {}
14         Predmet(const Predmet& p) : naziv(p.naziv), testovi(p.testovi) {}
```

Predmet ima listu pokazivača na testove

Testiranje napisanih konstruktora u *main.cpp*-u:

```
cout << endl << endl;
cout << "Testiranje klase Predmet" << endl;
```

```
Predmet p, p2("OP"), p3("MISS");
```

# Rešenje - klasa Predmet - metoda za ispis

```
void ispis() const {  
    cout << "*****" << endl;  
    cout << "Predmet: " << naziv << endl;  
  
    if(testovi.size() == 0) {  
        cout << endl << "Nema testova." << endl;  
    }  
    else {  
        Test* t;  
        cout << "Testovi: " << endl;  
        for (int i = 1; i <= testovi.size(); i++){  
            testovi.read(i, t);  
            t -> ispis();  
        }  
    }  
    cout << "*****" << endl;  
}
```

Iz liste ćemo iščitavati pokazivače na testove, iz tog razloga pravimo lokalnu promenljivu koja je pokazivač na test.

Iz liste iščitavamo testove. Koristimo metodu *read*. Prvi parametar ove metode je indeks u listi sa kojeg iščitavamo vrednost. Drugi parametar je promenljiva u koju smeštamo pročitanu vrednost.

Da bismo mogli da pozovemo metodu *ispis* neophodno je da dereferenciramo pokazivač. Drugi način da ovo uradimo bio bi: `(*t).ispis();`

# Rešenje - klasa Predmet - testiranje metode za ispis

Nakon što napišemo metodu potrebno je da je pozovemo u *main.cpp*-u:

```
cout << endl << endl;  
cout << "Testiranje klase Predmet" << endl;
```

```
Predmet p, p2("OP"), p3("MISS");
```

```
p.ispis();  
p2.ispis();  
p3.ispis();
```

# Rešenje - klasa Predmet - metoda položio()

Const garantuje da metoda neće napraviti izmene nad objektima koje koristi

```
bool položio() const {  
    Test* t; ←  
    for (int i = 1; i <= testovi.size(); i++){  
        testovi.read(i, t);  
        if(! t -> položio() )  
            return false;  
    }  
  
    if(getUkupniPoeni() > 55){  
        return true;  
    } else {  
        return false;  
    }  
}
```

Iz liste ćemo iščitavati pokazivače na testove, iz tog razloga pravimo lokalnu promenljivu koja je pokazivač na test.

Iz liste iščitavamo testove. Koristimo metodu *read*. Parametar ove metode je indeks u listi sa kojeg iščitavamo vrednost. Drugi parametar je promenljiva u koju smeštamo pročitane vrednosti.

Da bismo mogli da pozovemo metodu *položio* za test neophodno je da dereferenciramo pokazivač. Drugi način da pozovemo ovu metodu bi bio: `(*t).položio()`;



# Rešenje - klasa Predmet - testiranje metode polozio()

Nakon što napišemo metodu potrebno je da je pozovemo u *main.cpp*-u.

```
cout << (p2.polozio() ? "Polozio OP" : "Nije polozio OP") << endl;
```

Preostale metode klase **Predmet** implementiraju se na vrlo sličan način.  
Kompletno rešenje se nalazi u folderu sa nazivom 3.

# Rešenje - klasa Student

Da bismo proverili broj poena koje student ima iz predmeta, nad predmetom pozivamo metodu *getUkupniPoeni()*

```
test.hpp x kolokvijum.hpp x main.cpp x predmet.hpp x student.hpp x
1  #ifndef STUDENT_HPP_INCLUDED
2  #define STUDENT_HPP_INCLUDED
3
4  #include "predmet.hpp"
5
6  class Student{
7      private:
8          Predmet p1;
9          Predmet p2;
10     public:
11         Student(const Predmet& pp1, const Predmet& pp2) : p1(pp1), p2(pp2) {}
12
13         void ispis() const{
14             p1.ispis();
15             p2.ispis();
16             cout << "Student je osvojio vise poena iz predmeta: " << ((p1.getUkupniPoeni() > p2.getUkupniPoeni()) ? p1.getNaziv() : p2.getNaziv()) << endl;
17         }
18     };
19
20 #endif // STUDENT_HPP_INCLUDED
```

Istestirati klasu u *main.cpp*-u. Kompletно rešenje se nalazi u folderu sa nazivom 3.

## Zadatak 4

Napisati klasu **Component**, koja ima polja: X (double) i Y(double). U klasi implementirati: metode *double getX()* i *double getY()*, virtuelnu metodu *void printComponent()* i apstraktnu metodu *DinString getTypeName()*.

Napisati klasu **CheckBox** koja nasleđuje klasu **Component** koja i ima dodatna polja: pressed (bool) i statičko polje typeName (DinString). Vrednost statičkog polja tip je "CheckBox". Realizovati prazan konstruktor koji polja X i Y inicijalizuje na 0, a polje pressed na false i konstruktor sa parametrima. Definirati metodu *getTypeName()* tako da vraća vrednost polja typeName i metodu *printComponent()* da pored X i Y koordinata ispisuje i vrednost polja pressed. Implementirati i get i set metode za polje pressed...

(videti ceo tekst zadatka u folderu 2)

# Rešenje - klasa Component

```
component.hpp X
1  #ifndef COMPONENT_HPP_INCLUDED
2  #define COMPONENT_HPP_INCLUDED
3
4  #include "dinstring.hpp"
5
6  class Component {
7      protected:
8          int X;
9          int Y;
10
11      public:
12          double getX() const { return X; }
13          double getY() const { return Y; }
14
15          virtual void printComponent() const {
16              cout << "Component: X = " << X << ", Y = " << Y;
17          }
18
19          virtual DinString getTypeName() const = 0;
20      };
21
22  #endif // COMPONENT_HPP_INCLUDED
```

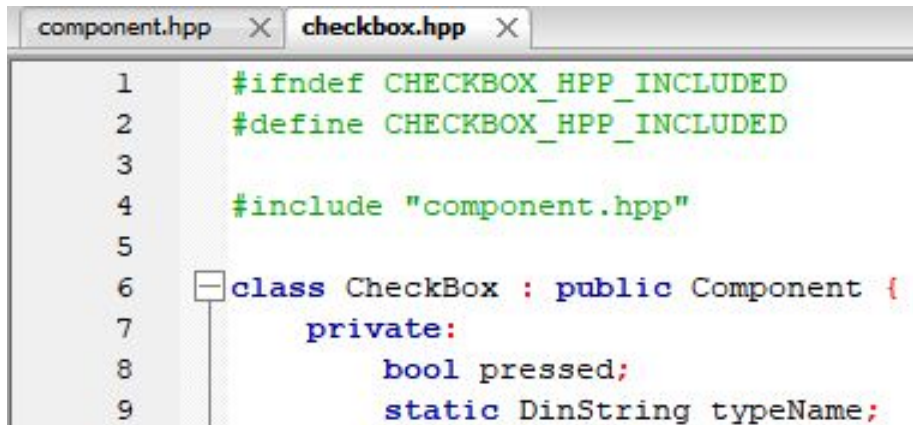
Napomena: Pošto je klasa **Component** apstraktna, ne pokušavati napraviti objekat ove klase jer će to prouzrokovati grešku!

# Rešenje - klasa Checkbox - polja

Napraviti fajl checkbox.hpp i u njemu implementirati klasu Checkbox.

Klasa **Checkbox** ima statičko polje typeName, ključna reč **static** ispred polja.

Statičko polje - zajednički podatak za sve objekte date klase. Drugim rečima, to znači da u svakom trenutku vrednost tog podatka-člana je jednaka za sve objekte date klase.



```
1  #ifndef CHECKBOX_HPP_INCLUDED
2  #define CHECKBOX_HPP_INCLUDED
3
4  #include "component.hpp"
5
6  class Checkbox : public Component {
7      private:
8          bool pressed;
9          static DinString typeName;
```

Statičko polje se inicijalizuje van klase.

```
39  DinString Checkbox::typeName = "CheckBox";
```

# Rešenje - klasa Checkbox - konstruktori

Podsećanje: u svakom objektu svake klase postoji podatak-član **this** koji jeste pokazivač na klasu kojoj taj objekat pripada i koji sadrži adresu datog objekta.

Klasa **Checkbox** ima konstruktor sa podrazumevanim vrednostima. Konstruktor sa podrazumevanim vrednostima objedinjuje konstruktor sa i bez parametara.

```
public:
    Checkbox(int X = 0, int Y = 0, int pressed = false) {
        this -> X = X;
        this -> Y = Y;
        this -> pressed = pressed;
    }
```

Pošto se parametar zove isto kao i polje, da bismo mogli izvršiti dodelu koristimo pokazivač **this**. Da u ovom slučaju nismo iskoristili pokazivač **this** već samo napisali **X=X**; parametru **X** bismo dodelili njegovu trenutnu vrednost, odnosno ne bismo ništa time postigli.

# Rešenje - klasa Checkbox - testiranje konstruktora

Implementirali smo konstruktor sa podrazumevanim vrednostima. U prvom slučaju polja će biti inicijalizovana na podrazumevane početne vrednosti. U drugom slučaju polja će biti inicijalizovana na prosleđene parametre.

```
8      int main()  
9      {  
10         CheckBox cb1;  
11         |CheckBox cb2(50, 70, true);
```

# Rešenje - klasa Checkbox - metoda print

Metoda *printComponent* je implementirana u roditeljskoj klasi. Da ne bismo morali da ponovo implementiramo deo koji ispisuje vrednosti polja koje smo nasledili iz roditeljske klase, možemo jednostavno pozvati metodu za ispis koja je implementirana u roditeljskoj klasi i nakon toga dodati ispis polja koje je dodato u ovoj klasi.

```
void printComponent() const {  
    cout << getTypeName() << endl;  
    Component::printComponent();  
    cout << " CB pressed = " << pressed;  
}
```

Ne zaboraviti pozvati metodu u *main.cpp*-u.

```
cb1.printComponent();
```



# Rešenje - klasa Label i Panel

Klasa **Label** je veoma slična klasi **Checkbox**. Rešenje se nalazi u folderu sa nazivom 4 u fajlu *label.hpp*

Klasa **Panel** je veoma slična klasi **Predmet** iz prethodnog zadatka. Rešenje se nalazi u folderu sa nazivom 4 u fajlu *panel.hpp*.

# Literatura

1. Kupusinac A. : Zbirka rešenih zadataka iz programskog jezika C++. Novi Sad: FTN, 2011.