OBJEKTNO PROGRAMIRANJE 1

Oznaka predmeta: OP1

Predavanje broj: 6

Nastavna jedinica: Deklaracije.

Nastavne teme:

Specifikatori; deklaracije i definicije. Povezivanje. Specifikatori memorijske oblasti. Specifikator typedef. Specifikatori const i volatile. Deklaracija asm. Specifikacije povezivanja sa drugim jezicima. Inicjializacija: objekata, agregata.

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

Literatura:

Dragan Milićev, “Objektno orijentisano programiranje na jeziku C++”, Mikro knjiga, Beograd, 2005.

Specifikatori; deklaracije i definicije

- Specifikatori su ključne reči navedene na početku deklaracije, ispred imena tipa, da bi dopunili tip (kao npr. specifikator const), ili specificirali neko drugo značenje deklaracije.
- Deklaracijom se uvodi neko ime u program. Deklaracija **nije** istovremeno i definicija u slučaju da:
  - deklariše funkciju bez navođenja njenog tela,
  - sadrži specifikator extern i ne sadrži inicijalizaciju niti telo funkcije,
  - predstavlja deklaraciju statičkog podatka člana unutar deklaracije klase,
  - predstavlja deklaraciju samo imena klase, ili da predstavlja deklaraciju tipa iza specifikatora typedef.
- Deklaracije koje nisu i definicije:

```
extern int a;
int f(int);
struct S;
```

- Definicija za objekat odvaja potreban prostor u memoriji i izvršava inicijalizaciju.
- Deklaracije koje su i definicije:

```
int a;
int f(int i) {return i+1;}
struct S { int a; int b;}; // probajte dva puta da navedete
```

Deklaracije i definicije

- Mora postojati tačno jedna definicija svakog objekta, funkcije, klase ili nabrajanja koji se koristi u programu.
- Kako je deklaracija klase sa svojim članovima i definicija, jasno je zašto se deklaracije klasa najčešće izdvajaju u fajlove-zaglavlja i uokviruju direktivama makroprocesora za uslovno prevođenje

```
struct S {int a;};
struct S {int a;};     // greška: dvostruka definicija!
```

ali:

```
#ifndef _S
#define _S
   struct S {int a;};
#endif

#ifndef _S
#define _S
   struct S {int a;};     // ispravno, jer se ovaj deo ne prevodi
#endif                    // ili koriscenjem  #pragma once
```

- Ako se ime klase koristi samo na način koji ne zahteva njenu definiciju, ne mora se definisati, već se samo može deklarisati njeno ime (class X;).

Povezivanje

- Program se sastoji iz odvojenih jedinica prevođenja, fajlova, koji se posle prevođenja povezuju.
- Svaki fajl se sastoji od deklaracija (funkcija, klasa, globalnih objekata, ...).
- Pri povezivanju se informacije o imenima iz različitih fajlova sređuju, kako bi se u nekom fajlu, u kome se koristi neko ime, obezbedilo da se to korišćenje usmeri na odgovarajuće mesto u nekom drugom fajlu, gde se nalazi definicija tog imena.
  - Primer: ako se neki objekat definiše u jednom fajlu, a koristi, uz prethodnu deklaraciju, u drugom fajlu, onda se sva mesta korišćenja u drugom fajlu moraju "usmeriti" na pravo mesto gde se nalazi definisani objekat.
- Imena koja se koriste na opisani način, koja se "vide" pri povezivanju u svim fajlovima programa, nazivaju se *eksternim* i kaže se da imaju *eksterno povezivanje* (*external linkage*).
- Eksterno povezivanje uvek imaju
  - inline funkcije,
  - statički članovi klase,
  - lokalna imena koja su eksplicitno deklarisana sa specifikatorom extern, a nisu static.

Povezivanje

- Globalna imena koja su deklarisana specifikatorom static, vide se samo u fajlu u kome su deklarisana, ona su lokalna za taj fajl, pa se u drugim fajlovima isto ime može predefinisati.
- Za ovakva imena se kaže da imaju *interno povezivanje* (*internal linkage*).
- Prema tome, ključna reč static ima, generalno, dva značenja:
    - za lokalna imena ona označava statički životni vek,
    - za globalna imena ona označava interno povezivanje.
- Na primer, ako je u nekom fajlu navedena deklaracija globalnog imena:

```
static int a=1;
```

  onda je ime **a** lokalno za taj fajl, što znači da se u drugim fajlovima ovaj objekat ne vidi, a isto ime se može koristiti za neke druge objekte, funkcije, klase itd.
- Globalno ime konstante ima interno povezivanje.
- Deklaracije iste klase mogu se nalaziti u više fajlova, uključivanjem fajla-zaglavlja sa deklaracijom klase (ne zaboravite #pragma once).
  - Isto važi i za konstante.

Povezivanje

- Jedan fajl-zaglavlje može se uključiti u proizvoljno mnogo programskih fajlova, ali u svaki samo jednom (što je obezbeđeno direktivama za uslovno prevođenje ili kao što smo koristili ranije direktivu #pragma once), može izgledati ovako:

```
#pragma once
const int N=100;
class MojString {
   //...
};
```

- Specifikatori static i extern mogu biti primenjeni na imena objekata, funkcija i bezimenih unija.
- Ne može se:
  - navesti static ili extern deklaracija formalnog argumenta,
  - članovi klase ne mogu biti deklarisani sa extern,
  - navesti static deklaracija funkcije unutar bloka.
- Ova pravila predstavljaju potreban i dovoljan skup za razumevanje tuđih i pisanje sopstvenih programa u jeziku C++.

Specifikator typedef

- Specifikatorom typedef predstavljaju mehanizam kojim se nekom izvedenom ili osnovnom tipu dodeljuje ime koje se kasnije može koristiti u deklaracijama, kako bi one bile čitljivije.
- Deklaracija koja počinje specifikatorom typedef ne uvodi objekat ili funkciju datog tipa T, već uvodi novo ime za dati tip T.
  - Novo ime za dati tip T navodi se na isti način na koji bi se deklarisao objekat datog tipa T.

  Na primer, deklaracijom:

```
typedef int CEO;
```

  dodeljuje se novo ime CEO za tip int, i ovo novo ime se može koristiti kao ime tipa (ravnopravno sa starim) u celoj oblasti važenja u kome je deklarisano:

```
int i;
CEO j;        // j je takođe tipa int
```

- Ime deklarisano u typedef deklaraciji je sinonim za dati tip (isti tip), a ne uvodi novi tip kao što to čini klasa. U prethodnom primeru promenljiva **j** je tipa int.
- U typedef deklaraciji se može deklarisati više imena tipova, na način kao i u običnoj deklaraciji:

```
typedef int   (*PF)(int), *PI, AI[10], F(int);
```

Specifikator typedef

```
typedef int   (*PF)(int), *PI, AI[10], F(int);
```

ovde je

- PF je sinonim za tip "pokazivač na funkciju koja uzima argument tipa int i vraća rezultat tipa int",
- PI je sinonim za tip "pokazivač na int",
- AI je sinonim za tip "niz od 10 elemenata tipa int",
- F je sinonim za tip "funkcija koja uzima argument tipa int i vraća rezultat tipa int".

- Deklaracije typedef se najčešće koriste za uvođenje novog imena za neki složeni izvedeni tip, kako bi deklaracije objekata tog tipa bile čitljivije.
- Za prethodni primer se, tako, mogu deklarisati npr. objekti:

```
PF pf=&f;     // pf je tipa int(*)(int)
AI a;         // a je tipa int[10]
F f;          // f je tipa int(int)
```

Specifikatori const i volatile

- Konstantni objekti imaju interno povezivanje i moraju biti inicijalizovani.
- Navođenje specifikatora volatile je zahtev prevodiocu da ne optimizuje pristupe objektu tako što će u nekim trenucima njegovu vrednost čuvati u registrima procesora, već da se za svako korišćenje objekta obraća memoriji, jer taj objekat može u nekom trenutku promeniti vrednost, na način na koji prevodilac to ne može identifikovati.
    - "Govori" kompajleru: Ne znaš kada će ova (volatile) promenljiva biti promenjena.
    - **volatile** promenljiva se **uvek** čita kada je zahtevana

  Na primer,
  - ako u okruženju ili samom programu postoji neka prekidna rutina koja može da promeni vrednost nekog objekta, i ako se ta prekidna rutina može aktivirati prekidom koji nastaje u nedefinisanom trenutku za program (npr. hardverski prekid), onda se prevodilac ne može osloniti na vrednost nekog objekta, osim ako ona nije neposredno pre korišćenja očitana iz memorije.
  - deklarisanjem takvog objekta kao volatile, prevodiocu se stavlja do znanja da se objekat može u nepoznato vreme izmeniti od strane neke prekidne rutine, i da to treba da uzme u obzir prilikom korišćenja tog objekta.

Deklaracija asm

- Deklaracija asm ima sledeći oblik:

  ```
  __asm {string_literal};
  ```

- Značenje ove deklaracije zavisi od implementacije prevodioca.
- Namena ove deklaracije je da se informacije navedene u deklaraciji proslede asembleru.
- Mnogi prevodioci nude mogućnost pisanja delova programa u asemblerskom jeziku mašine.
- Ovakvi delovi pišu se unutar bloka __asm deklaracije, po formatu koji zavisi od mašine i prevodioca.
- Asembler ugrađen u prevodioca (*built-in assembler*) prevodi dati kôd i integriše ga u program u jeziku C++, na mestu na kome se nalazi ova deklaracija.
- Prevodioci najčešće nude i mogućnost da se iz asemblerskog dela programa koriste objekti definisani spolja u jeziku C++, pozivaju funkcije itd.
- Za precizne specifikacije načina korišćenja objekata i poziva funkcija treba konsultovati dokumentaciju prevodioca.

Specifikacije povezivanja sa drugim jezicima

- Moguće je u jeziku C++ deklarisati imena tako da se omogući povezivanje delova programa koji su pisani u jeziku C++ sa delovima koji su pisani u drugim jezicima.
- Ove deklaracije imaju jedan od dva sledeća oblika:

```
extern string_literal {lista_deklaracija}
extern string_literal deklaracija
```

- String literal u deklaracijama ukazuje na način povezivanja. Tačno značenje string literala zavisi od implementacije prevodioca.
- Povezivanje sa funkcijama pisanim u jeziku C (navodi se string literal "C") kao i povezivanje sa funkcijama pisanim u jeziku C++, koje se zahteva navođenjem string literala "C++", uvek je podržano. Povezivanje "C++" se podrazumeva.

```
extern int strlen(const char *);// podrazumevano povezivanje "C++"
extern "C" {
  int strsrch(const char*, char);  // povezivanje "C"
}
```

Primer koji je validan u C-u a nije u C++-u:

```
struct template{
    int new;
    struct template* class;
};
```

Specifikacije povezivanja sa drugim jezicima

- Specifikacije povezivanja obezbeđuju da se objekti i funkcije definisani u delovima programa pisanim u nekom drugom jeziku mogu koristiti u delovima pisanim u jeziku C++, i suprotno.
- Specifikacija string literalom predstavlja zahtev za poštovanje odgovarajuće konvencije poziva funkcija i povezivanja uopšte, a ne uvek kao specifikaciju jezika, jer neki jezici imaju iste konvencije povezivanja.
  - Za konvencije treba konsultovati dokumentaciju prevodilaca (treba videti koje konvencije podržava npr. "Pascal", "FORTRAN" itd.).

Na primer:

```
integer function fortranfuntion( a, b )
        double precision a, b
        ...
end
```

```
extern "C" { int fortranfuntion_( double&, double& ); }
...
int main() {
   double a, b;
   ...
   int i = fortranfunction_( a, b );
   ...
}
```

Specifikacije povezivanja; Inicijalizacija

- Treba primetiti da specifikacija povezivanja samo zahteva od prevodioca da poštuje odgovarajuću konvenciju povezivanja, što na semantiku programa nema nikakvog značaja.

- **Inicijalizacija**
- U deklaraciji se može navesti inicijalizacija identifikatora koji se deklariše.
- Ako je deklarisani objekat pripadnik neke klase koja ima konstruktor, to znači da se nailaskom na deklaraciju sa inicijalizacijom, koja je zbog toga i definicija, kreira taj objekat, odnosno poziva njegov konstruktor sa odgovarajućim argumentima.
  - Inače, ako nije u pitanju objekat klase, onda se samo objekat postavlja na početnu vrednost nailaskom programa na njegovu deklaraciju sa inicijalizacijom.
    - Negde se deklaracija bez eksplicitne inicijalizacije realizuje tako da se postavlja vrednost "0" a negde ostavlja zatečeni sadržaj u memoriji koja je dodeljena (int i;)
- U jeziku C++ postoji nekoliko stilova pisanja inicijalizatora.

Inicijalizacija objekata

- Objekti se, generalno, mogu inicijalizovati na jedan od sledeća tri načina:

```
T ime = izraz;
T ime(izraz);

T ime = {lista_izraza};
```

primenjuje se za inicijalizaciju struktura podataka i nizova.

- U većini slučajeva, efekat inicijalizacije je isti za prva dva načina. Zbog toga je pitanje koji će se od ova dva načina upotrebiti, najčešće samo pitanje stila.
- Prvi način se najčešće upotrebljava za inicijalizaciju objekata ugrađenih tipova i nekih jednostavnijih korisničkih tipova.
  Na primer:

```
int i=6, j=0;
Complex c = Complex(0.3, -0.1);
char *s="Zdravo!";
```

- U drugom redu objekat c se inicijalizuje privremenim objektom koji je kreiran direktnim pozivom konstruktora klase complex iza znaka dodele (=).
- Ovaj stil uglavnom asocira na semantiku da se "objekat samo postavlja na početnu vrednost koja je navedena u izrazu, a da u toku inicijalizacije nema nekih sporednih operacija potrebnih pri kreiranju objekta".

Inicijalizacija objekata

- Drugi način se koristi uglavnom za kreiranje objekata složenijih korisničkih klasa, koje imaju konstruktore u kojima se obavljaju složenije operacije:

```
string p("Zdravo!");
Osoba otac("Petar Petrovic", 40);
```

- Prethodni primeri mogli su biti napisani i ovako, što za ugrađene tipove ima isti efekat, kao najšće i za korisničke tipove:

```
int i(6), j(0);
Complex c(0.3, -0.1);
char *s("Zdravo!");

string p="Zdravo!";
Osoba otac = Osoba("Petar Petrovic", 40);
```

- Automatske, registarske, statičke i eksterne promenljive mogu se inicijalizovati proizvoljnim izrazom koji u sebi sadrži konstante i prethodno deklarisane promenljive i funkcije.
- Deklaracija sa inicijalizacijom je definicija, pa se izraz kojim se inicijalizuju ovakve promenljive izračunava u toku izvršavanja, a ne u toku prevođenja programa.

Inicijalizacija objekata

- Inicijalizacija globalnih objekata ide redom:
  - globalni objekti
  - statički globalni objekti
    - Inicijalizacija svih statičkih globalnih objekata u programskom fajlu vrši se pre korišćenja bilo koje funkcije ili objekta iz istog fajla.
    - To može biti pre poziva funkcije main, ali i u bilo kom trenutku pre korišćenja bilo koje funkcije ili objekta iz istog fajla.
- Statički objekti za koje nije naveden inicijalizator se obavezno inicijalizuju na 0 konvertovanu u odgovarajući tip.
- Destruktori za inicijalizovane statičke objekte klasa se pozivaju po povratku iz funkcije main, kao i pri pozivu funkcije exit.
  - Redosled poziva ovih destruktora je obratan od redosleda inicijalizacije.
    - poziv funkcije abort ne uključuje poziv ovih destruktora.
- Početna vrednost za automatske objekte koji nemaju inicijalizatore nije definisana.
- Prazne zagrade ne predstavljaju inicijalizator, pa sledeća deklaracija ne deklariše objekat, već funkciju koja nema argumente:

```
int x();
```

Inicijalizacija agregata

- Agregat (*aggregate*) je niz, ili objekat klase koja nema konstruktore, ni privatne ni zaštićene članove, nema osnovnu klasu i nema virtuelne funkcije.
- Ovakve klase predstavljaju samo strukture podataka namenjene za prosto skladištenje podataka, bez složenijih operacija, pa su najčešće deklarisane kao struct, a ne kao class.
- Agregati se mogu inicijalizovati na sledeći način:

```
T ime={lista_izraza};
```

- Lista izraza sadrži izraze razdvojene zarezom (,).
- Svaki izraz inicijalizuje jedan element agregata. Ako je agregat:
  - niz, elementi se inicijalizuju redom, po rastućem indeksu.
  - klasa, članovi se inicijalizuju po redosledu kojim su deklarisani u klasi.

```
const int c=5;
int a[5]={32, 12, -1, 0, c+7}; // a[0]=32, ..., a[4]=12
struct S {int a,b;};
S ss={1,4};                    // ss.a=1, ss.b=4
```

- Ako agregat sadrži podagregate (višedimenzioni niz; član klase je objekat druge klase koja je agregat), pravilo se primenjuje rekurzivno, tako što se unutar vitičastih ({}) zagrada navode ugnežđene liste izraza unutar vitičastih zagrada.

Inicijalizacija agregata

```
int a[4][3]={ {1,2,3},     // a[0][0]=1, ... , a[0][2]=3
              {2,3,4},     // ...
              {0,0,0},     // ...
              {1,0,-1}     // a[3][0]=1, ... , a[3][2]=-1
            };
```

- Ako inicijalizator ima manje izraza nego što agregat ima elemenata, preostali elementi se inicijalizuju na vrednost 0 konvertovanu u odgovarajući tip.
- Ako je agregat klasa onda se on može inicijalizovati i objektom.
- Ako agreagat sadrži podagregate, onda se ugnježdene vitičaste zagrade mogu izostaviti prema sledećem pravilu.
  - Ako lista izraza, koja inicijalizuje agregat ili neki podagregat (rekurzivno), počinje vitičastom zagradom, onda se elementi agregata inicijalizuju odgovarajućim izrazima u posmatranoj listi izraza; pri tom, broj izraza ne sme biti veći od broja elemenata.
  - Ako lista izraza ne počinje vitičastom zagradom, onda se za inicijalizaciju elemenata uzima tačno onoliko izraza koliko ima elemenata; preostali izrazi se ostavljaju za inicijalizaciju ostalih elemenata.
- Pored toga, ako se dimenzije niza u deklaraciji izostave, uzeće se da niz ima onoliko elemenata, koliko ima izraza za inicijalizaciju.

Inicijalizacija agregata

Na primer:

```
int a[]={1,2,3,4};      // niz a ima 4 elementa;
int b[4][3]={1,2,3, 2,3,4, 0,0,0, 1,0,-1};
                        // b[0][0] je 1, ..., b[1][0] je 2 itd.
int c[4][3]={{1},{2},{3},{4}};
                        // c[0] je {1,0,0}, c[1] je {2,0,0},
                        // c[2] je {3,0,0}, c[3] je {4,0,0}
struct Osoba {
  char *ime;
  int godine;
};

Osoba niz[2] = {
                   "Petar Petrovic", 40,
                   "Milka Petrovic", 35
                };
```

- Unija koja nema konstruktor može se inicijalizovati ili objektom istog tipa, ili inicijalizatorom u vitičastim zagradama koji inicijalizuje samo prvi član unije.
- Nizovi znakova koji se incijalizuju string literalima ne moraju imati specificiranu dimenziju u deklaraciji; za dimenziju se uzima dužina string literala.
    - I za ovakve nizove važi pravilo da dimenzija niza ne sme biti manja od broja elemenata u inicijalizatoru.

```
char s[]="Zdravo!";   // s ima 8 elemenata, 7 "slova" i '\0'
char p[7]="Zdravo!";  // greška!
```

Zadatak: asemblerski kod u C++

```
#include <cstdio>
#include <iostream>

char format[] = "%s %s\n";
char hello[] = "Hello";
char world[] = "world";

int main( void ){
   __asm {
      mov  eax, offset world
      push eax
      mov  eax, offset hello
      push eax
      mov  eax, offset format
      push eax
      call DWORD ptr printf
      pop  eax
      pop  eax
      pop  eax
   }
   system("pause"); return(0);
}
```

Agregat

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

class Trivijalna {
  int x;
public:
  Trivijalna() { x = 0; }     //da bi se popunio agregat
  Trivijalna(int i) { x = i; }
  int getX() { return x; }
};

const int iDuzina=8;

int main(){
  Trivijalna prvi[iDuzina] = { 1, 2, 3, 4,
                       Trivijalna(50), Trivijalna(60), Trivijalna(70) };
  for(int i=0; i < iDuzina; i++)
    cout << "prvi[" << i << "].getX(): " << prvi[i].getX() << "\n";

  system("pause");
  return 0;
}
```

Mini zadaci

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

int main(){
  int a={10}, b={15};
  typedef int funkcijasadvaargumenta(int arg1, int arg2);
  funkcijasadvaargumenta funkcija_mnozenja;//probajte da komentarisete
  funkcijasadvaargumenta funkcija_sabiranja;

  cout<<a<<"*"<<b<<"="<<funkcija_mnozenja ( a, b )<<endl;
  cout<<a<<"+"<<b<<"="<<funkcija_sabiranja( a, b )<<endl;

  system("pause"); return 0;
}

int funkcija_mnozenja( int x, int y ){
   return ( x * y );
}

int funkcija_sabiranja( int x, int y ){
   return ( x + y );
}
```

Mini zadaci

```
#include <iostream>
using namespace std;
int saberi( int x, int y );
typedef int (*PF)(int, int);

int main (){
  PF pf = saberi;
  int (*obicnipf)(int,int) = saberi; //ili =&saberi;
  int a={10}, b={15};
  cout<<a<<"+"<<b<<"="<< pf(a,b) <<endl;
  cout<<a<<"+"<<b<<"="<< obicnipf(a,b) <<endl;
  cout<<a<<"+"<<b<<"="<< (*obicnipf)(a,b) <<endl;
  system("pause"); return 0;
}

int saberi( int x, int y ){
   return ( x + y );
}
```

Mini zadaci

```
//_____________________________fajl globalne.cpp _____________________
#include <iostream>
using namespace std;
#include "Automobil.h"
int gBrojPortaRMI = 1099;
Automobil dummy;
void pozdrav(){
   cout<<"pozdrav iz fajla "<<__FILE__<<endl;
}
//_____________________________ fajl gde je main _____________________
#include <iostream>
using namespace std;
#include "Automobil.h" //ovde su
extern int gBrojPortaRMI;
extern Automobil dummy;
extern void pozdrav();
namespace moj_imenik{   int max=100; }
int main (){
   cout<<"Broj porta za RMI je "<<gBrojPortaRMI<<endl;
   dummy.IspisiPodatke();
   pozdrav();
   cout<<"moj_imenik::max = "<<moj_imenik::max<<endl;
   system("pause"); return 0;}
```

Jednostavni imenik 1/6

```
//_____________________________fajl Konstante.h
#pragma once
class Konstante{
   public:
     static const int MAX_PERSONS=10;
};
//_____________________________fajl gde je main
#include <iostream>
#include <string>
using namespace std;
#include "TelephoneDictionary.h"
#include "Konstante.h"

int main(){
  TelephoneDirectory imenik;
  int size;

  cout << " Koliko osoba unosite u telefonski imenik" <<endl
       << " (od 1 do maksimalno "
       << Konstante::MAX_PERSONS <<"): ";
  cin >> size ;
```

Jednostavni imenik 2/6

```
 if (size>Konstante::MAX_PERSONS || size<1)
  {
    cout<< "NAPISAH OD 1 DO MAKSIMALNO "
    << Konstante::MAX_PERSONS << endl;
    system("pause");return 2;
  }

  for(int i=0;i<size;i++)imenik.AddPerson();

  imenik.DisplayAll();

  cout<<endl<<"Prikazi zapise kod kojih je ime vlasnika Pera"<<endl;
  imenik.DisplayNumberOfOwner("Pera");

  cout<<endl<<"Prikazi zapise kod kojih je broj telefona 456"<<endl;
  imenik.DisplayOwnerOfNumber("456");

  system("pause");
  return 0;
}
```

Jednostavni imenik 3/6

```
//_____________________________fajl TelephoneDirectory.h
#include <iostream>
#include <string>
using namespace std;
#include "Person.h"
#include "Konstante.h"
class TelephoneDirectory{
  private:
    int index;
    Person persons[Konstante::MAX_PERSONS];//nedinamicki reseno
  public:
    TelephoneDirectory();
    void AddPerson();
    void DisplayAll();
    void DisplayOwnerOfNumber(string telnumber);
    void DisplayNumberOfOwner(string owner);
    //razmislite sta je potrebno za brisanje i menjanje zapisa
};
```

Jednostavni imenik 4/6

```
//_____________________________fajl TelephoneDirectory.cpp
#include "TelephoneDictionary.h"
TelephoneDirectory::TelephoneDirectory(){
  index = -1;
}
void TelephoneDirectory::AddPerson(){
  if(index<Konstante::MAX_PERSONS-1){
    index++;
    persons[index].SetFirstName();
    persons[index].SetSecondName();
    persons[index].SetTelephone();
    persons[index].SetAddress();
  }
}
void TelephoneDirectory::DisplayAll(){
  for(int i=0; i<=index; ++i) persons[i].Display();
}
void TelephoneDirectory::DisplayOwnerOfNumber(string telnumber){
  for(int i=0; i<=index; ++i){
    if( (persons[i].GetTelephone())==telnumber) persons[i].Display();
  }
}
```

Jednostavni imenik 5/6

```
void TelephoneDirectory::DisplayNumberOfOwner(string owner){
  for(int i=0; i<=index; ++i) {
    if( (persons[i].GetFirstName())==owner) persons[i].Display();
  }
}
//_____________________________fajl Person.h
#include <iostream>
#include <string>
using namespace std;
class Person{
  private:
    char firstname[49+1];
    char secondname[49+1];
    char telephone[19+1];
    char address[49+1];
  public:
   void SetFirstName(); void SetSecondName(); void SetTelephone();
   void SetAddress(); string GetTelephone(); string GetFirstName();
   void Display();
};
```

Jednostavni imenik 6/6

```
//_____________________________fajl Person.cpp
#include "Person.h"
void Person::SetFirstName(){
  cout << " Enter first name  : " ; cin.ignore(); cin.get(firstname,50);
}
void Person::SetSecondName(){
  cout << " Enter second name : " ; cin.ignore(); cin.get(secondname,50);
}
void Person::SetTelephone(){
  cout << " Enter telephone   : " ; cin.ignore(); cin.get(telephone,20);
}
void Person::SetAddress(){
  cout << " Enter address     : " ; cin.ignore(); cin.get(address,50);
}
string Person::GetTelephone(){ return telephone; }
string Person::GetFirstName(){ return firstname; }
void Person::Display(){
  cout << " Name      : " << firstname <<" "<<secondname<< endl;
  cout << " Telephone: " << telephone  << endl;
  cout << " Address  : " << address << endl;
}
```

Izlaz

```
 Koliko osoba unosite u telefonski imenik (od 1 do maksimalno 10): 2
 Enter first name  : Pera
 Enter second name : Peric
 Enter telephone   : 123
 Enter address     : BG
 Enter first name  : Sima
 Enter second name : Simic
 Enter telephone   : 456
 Enter address     : NS
 Name      : Pera Peric
 Telephone: 123
 Address   : BG
 Name      : Sima Simic
 Telephone: 456
 Address   : NS
Prikazi zapise kod kojih je ime vlasnika Pera:
 Name      : Pera Peric
 Telephone: 123
 Address   : BG
Prikazi zapise kod kojih je broj telefona 456:
 Name      : Sima Simic
 Telephone: 456
 Address   : NS
```