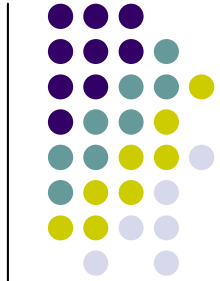
Članska funkcija



- U C-u (i ostalim proceduralnim jezicima) podaci i funkcije su u programu odvojeni
- Nije problem samo po sebi, osim u slučajevima koji su slični našem!
  - Funkcije koje smo definirali za rad sa strukturom DinamickoPolje su tijesno povezane s tom strukturom
  - Pitanje je kako dodatno naznačiti tu povezanost!
- Sintaksno – deklaraciju funkcije stavljamo **unutar** deklaracije strukture
  - Time funkcija postaje član strukture
- Terminološka promjena
  - Element strukture (varijabla deklarirana kao dio strukture) postaje **članska varijabla** strukture

Primjer članske funkcije

```
struct StrukturaC {
  int NekiPodatak;
};
void Funkcija(StrukturaC *s){};
```

```
struct StrukturaCPP {
  int NekiPodatak;
  void Funkcija() {}
};
```

```
int main(int argc, char* argv[])
{
   StrukturaC objC;
   Funkcija (&objC);

   StrukturaCPP objCPP;
   objCPP.Funkcija();

   return 0;
}
```

Struktura proširena članskim funkcijama

```
struct DinamickoPolje
{
// Članske funkcije
   int   Inicijaliziraj(int inMaxBrojElem);
   void  Izbrisi();
   int   PostaviNovuVelicinu(int NoviBrojElem);

   void PostaviElement(int Indeks, int Vrijednost);
   int  DohvatiElement(int Indeks);
   int  DodajElementNaKraj(int Vrijednost);

   int  BrojElemenata();

// Članske varijable
   int *_Podaci;

   int _BrojElem; // koliko stvarno ima elemenata u polju
   int _MaxBrojElemenata; // maksimalni raspoloživi prostor
};
```

S članskim funkcijama

- Podaci
- P04_DinamickoPolje_Cpp1_clanske_funkcije
- Pokreni program

- P03

Obična i članska funkcija

```
int Inicijaliziraj(struct DinamickoPolje
    *Polje, int MaxBrojElem)
{

    Polje->BrojElem = 0;
    Polje->MaxBrojElemenata = MaxBrojElem;




    Polje->Podaci = (int *)
    malloc(MaxBrojElem * sizeof(int));

    if( Polje->Podaci == NULL )
          return 1;
    else
          return 0;
}
```

```
int DinamickoPolje::Inicijaliziraj(int
    inMaxBrojElem)
{
    // Implicitno korištenje this
    _BrojElem = 0;
    _MaxBrojElemenata = inMaxBrojElem;

    // ili eksplicitno
    this->_BrojElem = 0;
    this->_MaxBrojElemenata = inMaxBrojElem;

    _Podaci = (int *) malloc(_MaxBrojElemenata *
    sizeof(int));

    if( _Podaci == NULL )
          return 1;
    else
          return 0;
}
```

Klasa (class) – Struktura (struct)

(vrijedi samo za C++)

Klasa (class) i struktura (struct) – ista funkcionalnost.

Jedina razlika:
članovi u klasi su po defaultu privatni; članovi u strukturi su po defaultu javni

Operator dosega :: (1.put)

- Koristi se za definiranje člana klase izvan deklaracije klase.

```
int DinamickoPolje::Inicijaliziraj(int inMaxBrojElem)
{
    this->_BrojElem = 0;
    this->_MaxBrojElemenata = inMaxBrojElem;

    _Podaci = (int *) malloc(_MaxBrojElemenata *
    sizeof(int));

    if( _Podaci == NULL ) return 1; else return 0;
}
```

Prednosti korištenja članskih funkcija

- Riješili smo nezgrapnost stalnog prenošenja pokazivača na strukturu kao parametra
  - Doduše, “problem” je samo prividno riješen – ipak se mora naznačiti za koju varijablu (instancu strukture) se poziva funkcija
- Riješili smo problem sukoba imena
  - Možemo imati funkcije istog imena, sve dok su one elementi različitih struktura

Prednosti korištenja članskih funkcija (2)

- Poziv funkcija koje su dio strukture se po sintaksi jasno razlikuju od poziva običnih funkcija:

```
instancaStrukture.imeFunkcije(parametri)
```

- Funkcije koje djeluju nad podacima u strukturi imaju izravnu vezu s konceptom strukture budući da su njen sastavni dio

Kamo je nestao pokazivač DinamickoPolje * ?

- Kako pozvana funkcija zna za koju je varijablu strukture pozvana ?
- Funkciji se **implicitno** prenosi pokazivač `this`
  - `this` je pokazivač koji pokazuje na varijablu strukture za koju je članska funkcija pozvana
  - "lokalna varijabla" za čiju se "deklaraciju" i inicijalizaciju brine prevoditelj (compiler)

```
int DinamickoPolje::Inicijaliziraj(int inMaxBrojElem)
   {
   // implicitno korištenje this
   _BrojElem = 0;

   // ili eksplicitno
   this->_BrojElem = 0;
   ...
}
```

Operator dosega (2.put)

```
int DinamickoPolje::Inicijaliziraj
  (int inMaxBrojElem){…}
```

- `::` = *scope resolution* operator (operator određivanja dosega)
- U C++ : *scope* (doseg) = mjera vidljivosti identifikatora (varijable, funkcije, klase) u programu
- U našem primjeru je bitan jer njime kod implementacije funkcije `Inicijaliziraj` (a i svake druge članske funkcije) naznačavamo njenu pripadnost klasi `DinamickoPolje`

Preostali problemi

- I dalje imamo problem mogućeg izravnog mijenjanja sadržaja varijabli u strukturi
  - Utječe na kasnije ponašanje članskih funkcija (najčešće s nedefiniranim posljedicama!)
- Rješava se uvođenjem **prava pristupa**
  - Korištenjem ključnih riječi **`private`** i **`public`** graditelj strukture može odrediti koji njen dio će biti dostupan svima, a koji dio će biti dostupan samo članskim funkcijama strukture

Javni dostupni i privatni dijelovi strukture

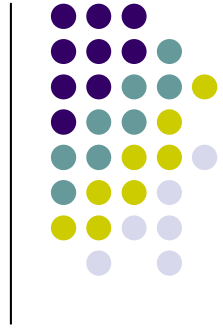
```
struct PrimjerStrukture {
  public:
    int _JavnoDostupnaVarijabla;
    void  JavnoDostupnaClanskaFunkcija(int
      Parametar);
  private:
    int _PrivatnaVarijabla;
    float PrivatnaClanskaFunkcija();
};
```

```
struct PrimjerStrukture {
public:
int _JavnoDostupnaVarijabla;
void     JavnoDostupnaClanskaFunkcija(int Parametar);
private:
int _PrivatnaVarijabla;
float PrivatnaClanskaFunkcija();
};
```

- Može se ograničiti skup mjesta (lokacija) u programu s kojih se može pristupiti privatnim dijelovima strukture
- Primjer “pristupa”:

```
PrimjerStrukture  var;
var._PrivatnaVarijabla = 0;
```

- Prevoditelj (compiler) će kod prevođenja programa javiti pogrešku!



```
class DinamickoPolje
{
public:
   int Inicijaliziraj(int inMaxBrojElem);
   void Izbrisi();
   int PostaviNovuVelicinu(int NoviBrojElem);

   void PostaviElement(int Indeks, int Vrijednost);
   int DohvatiElement(int Indeks);
   int DodajElementNaKraj(int Vrijednost);

   int BrojElemenata();

private:
   int *_Podaci;

   int_BrojElem; // koliko stvarno ima elemenata u polju
   int MaxBrojElemenata; // koliki je maksimalni
   raspolozivi prostor
};
```

- Privatnim dijelovima strukture se može pristupiti samo unutar definicije (tijela) članske funkcije
- I članske funkcije mogu biti privatne!
- Napomena:
  - Postoji i **`protected`** pravo pristupa
  - Važno je kod uspostavljanja hijerarhije struktura putem nasljeđivanja

U sljedećem nastavku...

- Konačno – klase !

Klasa

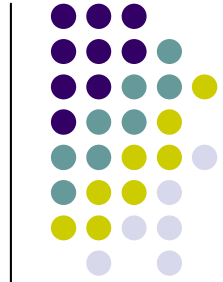
- Uvodimo ključnu riječ *class*
  - Klasa poopćava pojam strukture iz C-a
- Koja je razlika ?
  - Sa stanovišta C++ prevodioca primarno sintaksna – kod strukture je sve podrazumijevano *public* dok je kod razreda sve podrazumijevano *private*
  - Konceptualna razlika:
    - Struktura je (ipak) namijenjena modeliranju skupa jednostavnih podataka nad kojima ostali dijelovi programa direktno operiraju
    - Razred kao primarni koncept OO paradigme (prisutan u svim OO jezicima!) je namijenjen modeliranju (predstavljanju) koncepata iz područja problema koji rješavamo (DinamickoPolje, Stog, HashFile, ...) koji imaju složeno ponašanje realizirano preko skupa članskih funkcija
- Primjer: P05_DinamickoPolje_Cpp2_prava_pristupa

Malo teorije

- Dva osnovna elementa OO paradigme: **apstrakcija** i **enkapsulacija** (učahurivanje)
- Apstrakcija – razredi / objekti predstavljaju koncepte iz domene problema koji rješavamo
  - Razred DinamickoPolje je apstrakcija koncepta dinamičkog polja s dobro definiranim karakteristikama
  - Nije li to isto i struktura u C-u?
    - Struktura predstavlja agregatni skup podataka nad kojima operiraju “vanjski” elementi programa (funkcije)
    - Vanjske funkcije nisu dio strukture – zbog toga struktura nije “potpuna” jer je za razumijevanje

- Enkapsulacija – “niti jedan dio složenog sustava ne bi smio ovisiti o unutrašnjim detaljima drugog dijela”
  - “unutrašnji detalji” – korisnika razreda DinamickoPolje uopće ne bi trebalo zanimati kako je realizirana njegova funkcionalnost
    - Napomena: koncept dinamičkog polja je vrlo jednostavan i praktički implicira način implementacije (dinamička alokacija memorije za elemente polja), ali kod složenijih razreda to ni u kojem slučaju nije tako!
      - Čak i kod dinamičkog polja možemo imati varijabilnost u implementaciji
      - Npr., ako je rukovanje memorijom “skriveno” od korisnika razreda, može se izgraditi vlastiti *memory-manager* (ne koristimo *malloc* i *realloc*, već vlastitu implementaciju –

Javno sučelje razreda



- Sve što je u razredu deklarirano kao *public* dio je javnog sučelja razreda
- Javno sučelje razreda predstavlja “prozor u svijet” kroz koji razred komunicira s ostalim dijelovima programa
  - Definiranjem članskih varijabli kao *private*, razred “skriva” svoje stanje
- Primjer enkapsulacije u stvarnom životu:
  - DVD player – javno sučelje čini par kabela i daljinski upravljač
    - “Običnog” korisnika u biti ne zanima **kako** DVD player