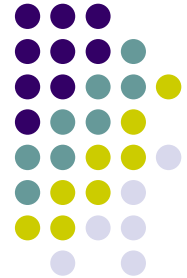
Poboljšanja:

- Primjena funkcija **`get`** i **`set`** za dohvaćanje i postavljanje vrijednosti elemenata u polju
- Dodavanje podatka o iskorištenosti alociranog prostora u strukturu **`DinamickoPolje`**
- Definiranje dodatnih funkcija za rad sa strukturom **`DinamickoPolje`**

Proširenje strukture DinamickoPolje

```
struct DinamickoPolje
{
  int *Podaci;
  int BrojElem;
     // stvarni broj elemenata u polju
  int MaxBrojElemenata;
     // maksimalno raspoloživi prostor
};
```

Funkcije potrebne za rad s dinamičkim poljem



Inicijaliziraj

Izbrisi

PostaviNovuVelicinu

PostaviElement

DodajElementNaKraj

DohvatiElement

BrojElemenata

Prototipovi funkcija

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

void Izbrisi (struct DinamickoPolje *Polje);

int  PostaviNovuVelicinu (
   struct DinamickoPolje *Polje, int NoviBrojElem);

void PostaviElement (
   struct DinamickoPolje *Polje, int Ind, int Vrijednost);

int DodajElementNaKraj (struct DinamickoPolje *Polje, int
   Vrijednost);

int DohvatiElement (struct DinamickoPolje *Polje, int
   Indeks);

int BrojElemenata (struct DinamickoPolje *Polje);
```

Problem dojave pogreške

- Funkciju **`DohvatiElement()`** smo deklarirali kao

```
int DohvatiElement (struct
  DinamickoPolje *Polje, int Indeks);
```

- A što ukoliko se zatraži vrijednost elementa polja za nepostojeći indeks ?
  - Potrebno je nekako naznačiti pogrešku !

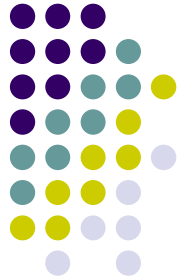
Signaliziranje pogreške preko povratnog argumenta funkcije

- Novi prototip:

```
int DohvatiElement2 (
   struct DinamickoPolje *Polje,
   int Indeks,
   int *DohvacenaVrijednost);
```

- Preko povratnog parametra (*return*) signaliziramo pogrešku (ako je bude)
- Preko call by reference vraćamo dohvaćenu vrijednost
- Alternativno: Izvesti naredbu *exit*() pa neka programer koji koristi DinamickoPolje popravi pogrešku u svojem kodu

Primjer P02



- P02_DinamickoPolje_C_get_set

- P01 DinamickoPolje_C_osnovna_impl
- Podaci (ulazna datoteka)

Naše DinamickoPolje se može koristiti samo za cjelobrojne podatke (int)

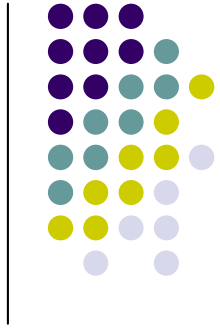
- Što ukoliko imamo datoteku u kojoj su zapisani *float* podaci ?
- Klasično C rješenje – korištenje *typedef*-a

```
typedef int TipDinamickogPolja;

struct DinamickoPolje
{
   TipDinamickogPolja *Podaci;
   int BrojElem;
     // koliko stvarno ima elemenata u polju
   int MaxBrojElemenata;
     // koliki je maksimalni raspolozivi prostor
};
```

Rješenje za neki drugi tip podatka u dinamičkom polju

- Bolje (i složenije) rješenje – koristimo void * kao tip podataka u polju
  - Programer kod iskorištavanja dinamičkog polja mora koristiti *cast* operatore!



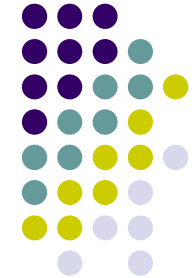
- Primjer: P03_DinamickoPolje_C_typedef
- P03_DinamickoPolje_C_void (nema primjera)
- *Napomena*: u C++ se efikasno i programski “čisto” rješava pomoću *template*-a (predložaka)
- template <class identifikator> deklaracija_funkcije;

```
template <class MojTip>
mojTip getMax (MojTip a, MojTip b) {
  return (a>b?a:b);
}
.....
int x,y;
getMax <int> (x,y);
```

Nedostaci sa stajališta dizajna programa (1/2)

- Svakoj funkciji moramo prenositi pokazivač na strukturu
  - Ovo je primarno sintaktički “nedostatak”
  - C (ali i Pascal, Fortran, Basic) programeri s takvom paradigmom “žive” već 20-ak godina
- Pozivi funkcija iz biblioteke su isprepleteni s pozivima drugih funkcija i sintaktički izgledaju isto
  - Smanjuje se preglednost i čitljivost programa

Nedostaci sa stajališta dizajna programa (2/2)

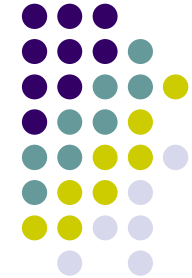


- Nema izravne i na prvi pogled vidljive povezanosti deklarirane strukture i funkcija koje nad njom operiraju
  - Sve ove funkcije dobivaju na mjestu prvog argumenta pokazivač `struct DinamickoPolje *`, ali će korisnici i sami definirati mnoge funkcije koje dobivaju takav pokazivač

Problem mogućeg sukoba naziva (engl. *name clash)*

- Što ukoliko je u nekoj drugoj biblioteci definirana funkcija s istim imenom ?
- Ako koristimo hrvatske nazive za varijable i funkcije, ovo nije toliki problem
- Kod korištenja engleskih termina (kod izrade biblioteka funkcija bitan zahtjev ukoliko se želi omogućiti međunarodno korištenje razvijene biblioteke) je situacija znatno nepovoljnija (Nazivi *initialize*, *setSize*, *cleanup* ili *delete* su standardni nazivi)

Problem izravne promjene vrijednosti u strukturi

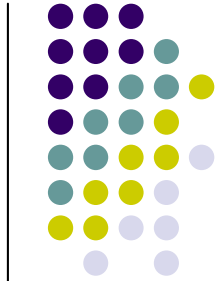


- Što ukoliko netko izravno promijeni vrijednost podatka u strukturi!
  - Definirane funkcije se oslanjaju na činjenicu da samo one izravno operiraju nad podacima u strukturi !
    - Situacija: korisnik izravno promijeni podatak BrojElem
      - Podaci u polju se nisu promijenili ali će se funkcije drugačije ponašati
    - Čak i ukoliko u dokumentaciji eksplicitno piše da se to ne smije raditi, prije ili kasnije će netko to i napraviti (greškom, neznanjem, lijenošću – tako mu je “lakše”)

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;
}
```