STRUKTURNI TIPOVI PODATAKA

Polja
Enumeracije
Strukture
Liste
Skupovi
Asocijatvna polja

Polja

- Kolekcija elemenata istog tipa
- Elementima polja se pristupa korišćenjem njihove pozicije (indeksa) u polju pa se zato često nazivaju i indeksne strukture
- Po definiciji to je statička struktura podataka (broj elemenata polja se ne menja u toku njegovog životnog ciklusa)

Polja u programskom jeziku C

- Polja u statičkoj zoni memorije:

```
int niz[100];
```

- Polja u dinamičkoj zoni memorije:
  - Broj elemenata polja je poznat u fazi izvršenja programa i tada se rezerviše prostor za njegove elemente u dinamičkoj zoni memorije
  - Za pristup podacima u dinamičkoj zoni memorije u programskom jeziku C koriste se **pokazivači**
    - Specijalni tipovi podataka koji pamte adrese drugih podatak u memoriji.
    - Najbitnija primena im je da pamte adrese podataka koji se kreiraju u toku izvršenja programa u dinamičkoj zoni memorije.
  - Primer:

```
int *niz, velicina;
scanf("%d", &velicina);
niz = (int*) malloc( velicina * sizeof(int) );
```

Polja u programskom jeziku C

- Brisanje polja iz dinamičke zone memorije:

```
free(niz);
```

Adresna aritmetika u programskom jeziku C

- U većini programskih jezika nad pokazivačima su definisane jedino operacije poređenja:
  - Jednakost: ==
  - Nejednakost: !=
- Jedino programski jezik C uvodi takozvanu adresnu aritmetiku pa su u ovom jeziku nad pokazivačima dozvoljene i operacije:
  - +, -, +=, -=, ++, --
- U C-u su dozvoljeni i drugi operatori poređenja (imaju smisla kad pokazivači ukazuju na elemente istog niza):
  - <, <=, >, >=

Adresna aritmetika u programskom jeziku C

- Tipovi operanada:
  - Operatori +, -, += i -= se primenjuju nad pokazivačima i podacima celobrojnog tipa
- Značenja operacija:

Izraz	Značenje
p+n	p+n*size(type)
p-n	p-n*size(type)
p+=n	P+=n*size(type)
p-=n	p-=n*size(type)
p++	p=p+size(type)
p--	P=p-size(type)

Pristup elementima niza pomoću pokazivača u programskom jeziku C

- Primenljiv i za nizove smeštene u statičkoj i za nizove smeštene u dinamičkoj zoni memorije
- I za nizove u statičkoj zoni memorije ime niza je pokazivač na njegov prvi element

```
int a[10], *pa, *pb, x, y;  // niz, pokazivaci, podaci
pa = & a[4];  // pa pokazuje na a[4], isto: pa = a + 4
x = *(pa +3)   /*  x = a[7] */
y = *pa + 3  /* y = a[4] + 3 */
*pa++; /* povecava se vrednost pokazivaca */
(*pa) ++; /* povecava se pokazivani podatak*/
pb = &a[2];/* pb pokazuje na a[2] */
if (pa < pb) … /* poredjenje pokazivaca */
```

Promena veličine niza u programskom jeziku C

- Jedino u programskom jeziku C niz u dinamičkoj zoni memorije ima osobinu dinamičke strukture podataka. Funkcijom *realloc* se može menjati veličina niza.

```
niz = (int*) malloc( velicina * sizeof(int) );
//...
realloc(niz, velicina2 * sizeof(int) );
```

Polja u programskom jeziku C++

- Nasleđene osobine iz C-a:
  - Mogu biti smeštena u statičkoj i dinamičkoj zoni memorije
- Razlika u odnosu na C:
  - Polja su statičke strukture podataka, ne može im se menjati veličina tokom životnog ciklusa

```
int niz1[50], velicina;
int* niz2;
//...
niz2 = new int[velicina];
```

Višedimenzonalna polja u programskom jeziku C ili C++

- **Dvodimenzionalna polja – matrice:**
- U statičkoj zoni memorije:

```
int matrica[10][50];
```

- U dinamičkoj zoni memorije:
  - Svaka vrsta poseban niz (ne moraju biti iste dužine)
  - Potreban niz pokazivača na vrste
  - Potreban i pokazivač na pokazivače na vrste

```
int n,m;
int** matrica;
matrica = new int*[m];
for (int i=0; i<m; i++)
      matrica[i] = new int[n];
```

Višedimenzonalna polja u programskom jeziku C ili C++

- **Dvodimenzionalna polja – matrice:**
- Brisanje polja iz dinamičke zone memorije

```
for (int i=0; i<m; i++)
      delete[] matrica[i];
delete[] matrica;
```

Polja u programskim jezicima Java i C#

- Referentni tip podataka
  - Smeštena uvek u dinamičkoj zoni memorije
  - Pristupa im se korišćenjem reference
  - To su statičke strukture podataka (ne može se menjati veličina tokom životnog ciklusa)

- Deklaracija promenljive tipa polja

Java:

```
<tip>[] <ime_polja>
```

ili

```
<tip> <ime_polja>[]
```

Primer:

```
int [] niz1;
int niz2[];
```

C#:

```
<tip>[] <ime_polja>
```

Primer:

```
int[] niz;
```

- Kreiranje polja

```
<ime_polja> = new <tip>[<velicina>];
```

Primer:

```
niz = new int[50];
```

Nizovi objekata u programskim jezicima Java i C#

- Deklaracija promenljive

```
<ime_klase> [] <ime_polja>;
```

Primer:

```
MyClass [] niz;
```

- Kreiranje polja

```
<ime_polja> = new <ime_klase>[<velicina>];
```

Primer:

```
niz = new MyClass[50];
```

- Kreiranje objekata

```
<ime_polja>[<pozicija>] = new <ime_klase>();
```

Primer:

```
for (int i=0; i<50; i++)
        niz[i] = new MyClass();
```

Atributi polja u programskim jezicima Java i C#

- U programskom jeziku Java polja sardže atribut **length** koji predstavlja veličinu polja.

Primer:

```
int [] niz = new int[n];
for (int i=0; i<niz.length; i++)
      niz[i]=i;
```

- Analogno, u programskom jeziku C# polja sardže svojstvo **Length** koji predstavlja veličinu polja.

Primer:

```
int [] niz = new int[n];
for (int i=0; i<niz.Length; i++)
      niz[i]=i;
```

Višedimenzonalna polja u programskom jeziku Java

- **Dvodimenzionalna polja – matrice:**
  - Dvodimenzionalnog polja se uvek definišu kao nizovi nizova:

```
<tip> [][] <ime_polja>;
```

Primer:

```
int [][] matrica;
```

  - Kreiranje pravougaonog dvodimenzionalnog polja

```
<ime_polja> = new <tip>[<brojVrsta>][<brojKolona>]
```

Primer:

```
matrica = new int[10][10];
```

  - Kreiranje "nazubljenog" polja

Primer:

```
trougaona_matrica = new int[3][];
for (int i=0; i<3; i++)
        trougaona_matrica[i] = new int[i+1];
```

Višedimenzonalna polja u programskom jeziku C#

- **Dvodimenzionalna polja – matrice:**
  - Definicija pravougaonih vi[edimenzionalnih polja:

```
<tip> [,] <ime_polja>;
```

Primer:

```
int [,] matrica;
```

  - Kreiranje pravougaonog dvodimenzionalnog polja

```
<ime_polja> = new <tip>[<brojVrsta>,<brojKolona>]
```

Primer:

```
matrica = new int[10,10];
//…
matrica[i,j] = 5;
```

  - Kreiranje “nazubljenog” višedimenzionalnog polja je isto kao u Javi.

Enumeracije

- Enumeracije – nabrojivi tip podataka
- Promenljiva tipa enumeracije može uzeti jednu od vrednosti navedenih u definiciji tipa.

Enumeracije u programskim jezicima C i C++

- Enumeracija se može shvatiti i kao skup celobrojnih simboličkih konstanti.
- Definicija enumeracije:

```
enum {
        <name1> [=<value1>],
        <name2> [=<value2>],
        …
};
```

Ukoliko inicijalizaciija konstante nije navedena, podrazumeva se da prva ima vrednost 0, a vrednost svake sledeće je za 1 veća od vrednosti prethodne navedene konstante.

Enumeracije u programskim jezicima C i C++

Primer 1:

```
enum Dani { Pon=1, Uto, Sre, Cet, Pet, Sub, Ned };

enum Dani dan;
```

C:

```
    dan = Uto;
```

C++:

```
    dan = Dani::Uto;
```

Enumeracije u programskom jeziku Java

- Referentni tip podataka
- Definicija enumeracije:

```
[<pravo_pristupa>] enum <ImeEnumeracije>
{
   name_1 [(<values_1)>],
   name_2 [(<values_2>)],
  …
   name_n [(<values_n>)];
   [<definicije_metoda>]
};
```

- Tip vrednosti može biti proizvoljan (isti za sve članove)

Enumeracije u programskom jeziku Java

Primer 1:

```
enum Dani { Pon, Uto, Sre, Cet, Pet, Sub, Ned };

Dani dan;
dan = Dan.Uto;

Dani[] svi = Dani.values();
```

Enumeracije u programskom jeziku Java

Primer 2:

```
  public enum Level {
    HIGH  (3),  //calls constructor with value 3
    MEDIUM(2),  //calls constructor with value 2
    LOW   (1)   //calls constructor with value 1
    ; // semicolon needed when fields / methods follow

    private final int levelCode;
    Level(int levelCode) {
        this.levelCode = levelCode;
    }
      public int getLevelCode() {
        return this.levelCode;
    }
}
```

```
Level level = Level.HIGH;
System.out.println(level.getLevelCode());
```

Enumeracije u programskom jeziku Java

Primer 3:

```
  public enum Praznik {
    NOVA_GODINA  (1,1),
    DAN_ZENA     (8,3)
    ;

    private final int dan;
    private final int mesec;
    Level(int dan, int mesec ) {
        this.dan = dan;
        this.mesec = mesec;
    }
}
```

Enumeracije u programskom jeziku C#

- Vrednosni tip podataka
- Definicija enumeracije:

```
[<pravo_pristupa>] enum <ImeEnumeracije>
[: <osnovni_tip>]
{
   name_1 [=<value_1>],
   name_2 [=<value_2>],
  …
   name_n [=<value_n>];
};
```

<osnovni_tip> - celobrojni tip kome pripadaju navedene konstante

Enumeracije u programskom jeziku C#

Primer 1:

```
enum Dani { Pon, Uto, Sre, Cet, Pet, Sub, Ned };
//sve konstante su tipa int, vrednosti su Pon=0, Uto=1,…
```

Ili

```
enum Dani : byte { Pon=1, Uto, Sre, Cet, Pet, Sub, Ned};
//navedene konstante su tipa byte, Pon=1, Uto=2,…
```

Primer promenljive tipa enumeracije:

```
Dani dan = Dani.Pon;
```

Struktrure

- Složeni tipovi podataka sastavljeni od elemenata različitog tipa.
- Svaki član strukture ima svoje ime – ime služi za pristup članu strukture.

Strukture u programskom jeziku C

- Definicija strukture:

```
struct <ime_strukture>
{
  <tip1> <ime_1>;
  <tip2> <ime_2>;
 …
};
```

- Definicija promenljive tipa strukture:

```
struct <ime_strukture> <ime_promenljive>;
```

- Pristup članovima strukture:

```
<ime_promenljive>.<ime_clana>
<ime_pokazivaca>-><ime_clana>
```

Strukture u programskom jeziku C

- Primer:

```
struct product
{
        int weight;
        double price;
};

struct product p1, p2;
p1.weight = 2;
p1.price = 2.5;
p2 = p1;
```

Strukture u programskom jeziku C++

- Struktrure (kao i klase) služe za predstavljanje objekata u programu.
- Razlika između struktura i klasa je u podrazumevanom pravu pristupa
  - **Za klase**:
    - podrazumevano pravo pristupa članovima je **privatno**
  - **Za strukture**:
    - podrazumevano pravo pristupa članovima je **javno**

Strukture u programskom jeziku C#

- Definicija strukture:

```
[<pravo_pristupa>] struct <ImeStrukture>
{
   // defincije tipova
  // definicije atributa
   // definicije metoda
   // definicije svojstava
};
```

  - Razlike u odnosu na klase:
    - Struktura je vrednosni tip
    - Strukture se ne nasleđuju
    - U strukturi se vrednosti atributa ne inicijalizuju (osim ako nisu konstantni ili statički)

Strukture u Javi?

- U programskom jeziku Java strukture **ne postoje**!!!

Liste

- Dinamičke strukture podataka – menjaju svoju veličinu tokom životnog ciklusa
- Veoma korišćene u funkcionalnim jezicima
- Svaki elementi liste sadrži:
  - Info deo – podatak
  - Vezu sa sledećim elementom (pokazivač ili referencu)

Operacije sa listama

- Dodavanje:
  - Na početak,
  - Na kraj,
  - Iza zadatog elementa.
- Brisanje:
  - Sa početka,
  - S kraja,
  - Zadatog elementa.
- Obilazak

Liste u programskim jezicima Java i C#

- Ne postoje kao ugradjeni tipovi
- Bibliotečke klase za predstavljanje listi:
  - U Javi:

    ```
    java.util.LinkedList<T>
    ```

  - U C#-u:

    ```
    System.Collections.Genric.LinkedList<T>
    ```

Liste u programskom jeziku Python

- Definicija liste:

```
Voce = [ "jabuka", "kruska", "banana"]
```

- Pristupanje elementima liste indeksiranjem:

```
Voce[1]
```

- Izdvajanje podliste:

```
Voce[1:2]
Voce[:2]
Voce[1:]
```

- Dodavanje elementa listi:

```
Voce[3]="jagoda";
```

- Heterogene liste:

```
lista = [ "jabuka", 1, [2, 3] ]
```

Skupovi

- Kolekcije “jedinstvenih” elemenata.
- Prvi put uvedeni u Pascal-u.
- Definicija skupa u Pascal-u:

```
set of <type>
```

- Primer:

```
type
    Dani = (Pon,Uto,Sre,Cet,Pet,Sub,Ned);
    SkupDana = set of Dani;
    SkupBrojeva = set of [1..10];
```

Skupovi u programskom jeziku Python

- Kreiranje skupa:

```
Voce = { "jabuka", "kruska", "banana"}
Ocene = { 1, 2, 3, 4, 5 }
```

- Operacije nad skupovima:

```
union()                    -    operator |
difference()               -    operator -
symetric_difference()      -    operator  ^
intersection()             -    operator &
                           -    operator in

add()
discard()
isdisjoint()
issubset()
issuperset()
```

Skupovi u Javi i C#-u

- Ne postoje kao ugradjeni tip, ali postoji citav niz ugradjenih klasa za predstavljanje skupova. Sve one implementiraju generički interfejs:

- U Javi:

```
java.util.Set<T>
```

U C#-u:

```
System.Collections.Genric.ISet<T>
```

Asocijativne liste (rečnici)

- Kolekcije parova (Ključ, Vrednost)
- Elementima liste se pristupa po ključu

Asocijativne liste u Python-u

- Kreiranje asocijatvne liste:

```
<ime> = { <kljuc1> : <vred1>, <kljuc2> : <vred2>,…}
```

- Primer:

```
Cenovnik={ “banana” : 120, “limun” : 150, “jabuka” : 60}
```

- Pristup elementima:
  - indeksiranjem:

```
Cenovnik[“jabuka”]
```

  - Metodom *get*:

```
Cenovnik.get(jabuka)
```

Asocijativne liste u programskim jezicima Java i C#

- Ne postoje kao ugradjeni tipovi
- Bibliotečke klase za predstavljanje asocijativnih listi:
  - U Javi:

    ```
    java.util.Dictionary<K,V>
    ```

  - U C#-u:

    ```
    System.Collections.Genric.Dictionary<K,V>
    ```