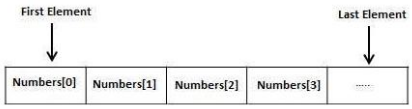
STATIČKE STRUKTURE PODATAKA

Statičke strukture podataka

- Način da se podaci grupišu i tretiraju kao celina
- Statičke strukture podataka se čuvaju u operativnoj memoriji
- Statičke su jer im se ne dodaju novi elementi, niti im se uklanjaju stari
- Dozvoljeno je samo menjati i čitati vrednost osnovnog elementa
- Pristup elementu statičke strukture je direktan i proizvoljan
- U zavisnosti od načina pristupanja elementima:
  - **Indeksirane strukture**, elementima se pristupa na osnovu njihove pozicije u statičkoj strukturi (indeksa) – nizovi, matrice, isl.
  - **Slogovi**, elementima se pristupa na osnovu njihovog naziva u strukturi (esencijalno klase minus metode; klase ćemo raditi kasnije)
- Do sada smo radili sa statičkim strukturama **skalarima**

Niz (Array)

- Struktura podataka u kojoj su podaci organizovani kao numerisana i uređena struktura
- Niz predstavlja kolekciju elemenata istog tipa
  - tip može biti bilo koji C# tip (klasa, primitivni tip, string, ...)
- Zauzima kontinuirani niz bajtova u memoriji
- Dužina niza predstavlja ukupan broj elemenata u nizu
- Svaki element niza se ponaša kao promenljiva i čuva podatak istog tipa kao i niz
- C# niz je sam po sebi klasa (Array), te sadrži i metode za rad sa nizom



Deklarisanje niza

- Deklariše se na sledeći način:

```
tip[ ] identifikator = new tip[ br_elem ];
```

  - tip određuje - kog tipa će biti elementi niza
  - identifikator - definiše naziv niza
  - [ ] je obavezna kako bi se znalo da se radi o nizu tog tipa
  - br_elem – definiše koliko će elemenata niz sadržati (mora biti ceo broj – konstanta ili neposredni operand)
  - new – operator koji ćemo učiti kasnije (alocira memoriju i inicijalizuje vrednosti)
  - količina zauzete memorije jednaka je `velicina_tipa * br_elem`
- Može i ovako, naravno:

```
tip[ ] identifikator;
identifikator = new tip[ br_elem ];
```

Definisanje niza

- Vrednosti elemenata niza se inicijalizuju tokom deklaracije (definicija) na sledeći način:

```
tip[ ] identifikator = {e1, e2, ..., ebr_elem};
tip[ ] identifikator = new tip[ ]{e1, e2, ..., ebr_elem};
```

  - tip određuje - kog tipa će biti elementi niza
  - identifikator - definiše naziv niza
  - br_elem – se izostavlja jer se broj elemenata niza izbroji iz **{ }**
  - **e_1, e_2, ..., e_{br_elem}** – vrednosti koje će se memorisati u niz
  - new – operator za inicijalizaciju i alokaciju memorije

Deklarisanje i definisanje niza primeri

- Primeri:

```
int[] intNiz = new int[3];
int[] intNiz1 = {1,2,3};
int[] intNiz2 = new int[]{1,2,3};

String[] stringNiz = new String[3];
String[] stringNiz1 = {"a","b","c"};
String[] stringNiz2 = new String[]{"a","b","c"};

float[] floatNiz = new float[3];
float[] floatNiz1 = {1.0f,2.0f,3.0f};
float[] floatNiz2 = new float[]{1.0f,2.0f,3.0f};
```

Pristupanje elementima niza

- Elementima niz se pristupa preko operatora **[ind]** gde je **ind** indeks (pozicija) elementa u nizu

  0 ≤ ind < br_elem

- Indeks niza u C# **uvek ide od 0**, a ne od 1!
- Identifikator niza predstavlja **adresu 1. elementa niza!**
- Primeri pristupanja elementima niza:

```
intNiz[0] = 12;
intNiz[1] = 2;
floatNiz[3] = floatNiz[0] * floatNiz[0];
a = intNiz[0];
```


Niz – inicijalizacija

- Niz se može inicijalizovati elemenat po elemenat (što je dobar i jedini pristup kada se ne znaju unapred svi elementi niza
- Idealno je koristiti **for** iteraciju kada se zna tačan broj elemenata niza
- Niz se može inicijalizovati na sledeći način:

```
for (i = 0; i < br_elem; i++)
  nizInt[ i ] = ei;
```

- **for** iteracija se koristi i za prolaz kroz sve elemente niza (ili jednog dela niza)

Unos elemenata niza sa tastature

- Mora se unositi svaki element zasebno
- Primer unosa vrednosti elemenata niza sa standardnog ulaza (pretpostavlja se niz celih brojeva od 5 elemenata):

```
for (i = 0; i < br_elem; i++)
  nizInt[ i ] = int.Parse(Console.ReadLine());
```


Ispis elemenata niza

- Mora se ispisivati svaki element zasebno
- Primer ispisa vrednosti elemenata niza na standardni izlaz (pretpostavlja se niz celih brojeva od 5 elemenata):

```
for (i = 0; i < br_elem; i++)
  Console.WriteLine(nizInt[ i ] );
```

Delimično popunjeni niz

- Kako je niz statička struktura, potrebno je unapred odrediti količinu memorije koja će se zauzeti
- Međutim, nije obavezno popuniti vrednost svih polja, tj. ne moraju se sva polja iskoristiti
- Npr. napravi se niz sa 10 polja, ali se od korisnika očekuje da unese koliko elemenata niz ima (maksimalno do 10) i da onda popuni vrednosti elem. niza za taj broj

```
int max_br_elem = 10;
int br_elem = 1;
int[] nizInt = new int[10];
Console.WriteLine("Unesite broj elemenata niza (maks. 10).");
br_elem = int.Parse(Console.ReadLine());
for (i = 0; i < br_elem; i++)
      nizInt[i] = int.Parse(Console.ReadLine());
```

- Kako proširiti zadatak sa proverom unosa?

Niz – primer

- **ZADATAK:**
  Implementirati program za računanje sume vrednosti elemenata niza prirodnih brojeva koji sadrži maksimum 50 elemenata. Program prihvata od korisnika broj elemenata ($0 < N \leq 50$) i vrednost svakog pojedinačnog elementa.

- **VEŽBA:**
  Proširiti prethodni zadatak tako da računa poziciju maksimalne vrednosti niza. Korisniku se prikazuje na kojoj poziciji u nizu se nalazi maksimalna vrednost, kao i sama vrednost.

Šta ako ima više istih vrednosti koje su maksimalne?

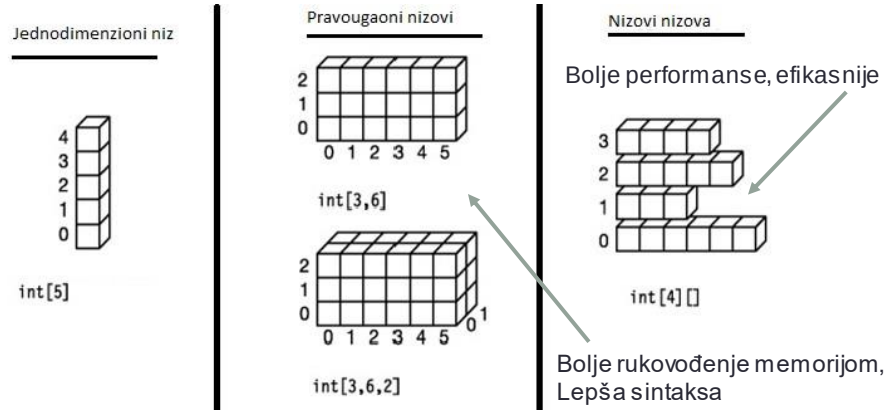
Niz – zadatak

```
/*************************************************************************************************
* Primer program za računanje sume vrednosti elemenata niza prirodnih brojeva koji sadrži maksimum 50 elemenata.
* Program prihvata od korisnika broj elemenata (0 < N ≤ 50) i vrednost svakog pojedinačnog elementa.
*************************************************************************************************/
            static void Main(string[] args) {
             const int MAXELNIZA = 50;
             int N = 1, i, suma = 0;
             int[] NizPrirordnihBrojeva = new int[MAXELNIZA];
             Console.WriteLine("Program za racunanje sume elemenata niza N prirodnih brojeva.");
             Console.Write("Unesite N:\t"); // prihvati broi elemenata niza
             N = Convert.ToInt32(Console.ReadLine());
             for (i = 0; i < N; i++)
             { // prihvati elemente niza od korisnika
                 Console.WriteLine("Unesite " + (i + 1) + ". element niza:\t");
                 NizPrirordnihBrojeva[i] = Convert.ToInt32(Console.ReadLine());
             }
             for (i = 0; i < N; i++) // izracunaj sumu elemenata niza
                 suma += NizPrirordnihBrojeva[i];
             Console.WriteLine("Suma elemenata niza izosi " + suma + ".");
             Console.ReadKey();
         }
```

- Dodati proveru korektnosti unetih vrednosti
- Da li se može koristiti manji broj **for** iteracija?

Matrice i višedimenzioni nizovi

- C# podržava rad sa multidimenzionalnim nizovima, ali i sa nizom nizova
- Te strukture su veoma slične ali ipak pomalo različite – koristite ih prema upotrebi



Matrice i višedimenzioni nizovi

- Matrica je višedimenzioni („pravougaoni“) niz, deklariše se na sledeći način:

```
tip[,] identifikator = new tip[dimenzija1, dimenzija2];
```

  - tip određuje - kog tipa će biti elementi matrice
  - identifikator - definiše naziv matrice
  - dimenzijaN – definiše koliko će nizova sadržati ta dimenzija sadržati matrice
  - new – operator za inicijalizaciju i alokaciju memorije
  - ukupan broj elemenata matrice je **dimenzija1 * dimenzija2 * … * dimenzijaN**

- Može i ovako, naravno (na primeru 2D niza):

```
tip[ , ] identifikator;
identifikator = new tip[br_vrsta,br_kolona];
```

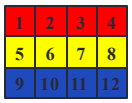
```
int[][] array2 = new int[4, 2]; // ERROR
int[][] array3 = new int[4][2]; // ERROR

int[,] array = new int[4, 2]; // ISPRAVNO ☺
```


… Matrica – deklarisanje

- 2D niz smatramo najvažnijim, te ćemo u nastavku govoriti o broju vrsta i kolona
- Kao i kod niza, broj vrsta i kolona mora biti ili **konstanta** ili **neposredni operand**
- Količina zauzete memorije jednaka je

```
velicina_tipa * br_vrsta * br_kolona
```

- Matrica zauzima kontinuiranu količinu memorije na sledeći način:



Matrica – definisanje ...

- Vrednosti elemenata matrice se mogu inicijalizovati tokom deklaracije (definicija) na sledeći način:

```
tip[,] identifikator = new tip [,]{{e11, ..., e1br_kol},
                                   {e21, ..., e2br_kol},
                                   ... ,
                                   {ebr_vrst1, ..., ebr_vrstbr_kol}};
```

  - tip određuje - kog tipa će biti elementi niza
  - identifikator - definiše naziv niza
  - br_vrst – broj vrsta se izostavlja jer se izbroji iz broja unutrašnjih **{}**
  - br_kol - broj kolona se izostavlja jer se izbroji broj elemenata
  - **e_{11}, e_{12}, ..., $e_{br\_vrstbr\_kol}$** – vrednosti koje će se memorisati u matricu
  - new tip [,] - operator za inicijalizaciju i alokaciju memorije - nije neophodno dok je inicijalizacija u istoj naredbi kao deklracija

Deklarisanje i definisanje - matrice primeri

```
// deklarisan, neinicijalizovan, nealociran 3D niz
int[,,] x3;
// deklarisan, neinicijalizovan, alociran 2D niz
int[,] x2 = new int[2, 100];

// 2D niz - definisanje
int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
// Identican 2D niz sa jasno definisanim dimenzijama.
int[,] array2Da = new int[4, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
// 2D niz stringova
string[,] array2Db = new string[3, 2] { { "one", "two" }, { "three", "four" },
                                        { "five", "six" } };

// 3D Niz.
float[,,] array3D = new float[,,] { { { 1f, 2f, 3f }, { 4f, 5f, 0.6f } },
                                    { { 7.66f, 8.90f, 9f }, { 10f, 11.87f, 12f } }};
// Identican 3D niz sa jasno definisanim dimenzijama.
int[,,] array3Da = new int[2, 2, 3] { { { 1, 2, 3 }, { 4, 5, 6 } },
                                      { { 7, 8, 9 }, { 10, 11, 12 } } };
```

Pristupanje elementima matrice

- Elementima matrice se pristupa preko operatora **[ind_vrste , ind_kolone]**
  gde su **ind_vrste** i **ind_kolone** indeksi (pozicije) elementa u matrici

  0 ≤ indeks_vrste < br_vrsta
  0 ≤ indeks_kolone < br_kolona
- Identifikator matrice predstavlja **adresu 1. elem. matrice!**
- Primeri pristupanja elementima niza:

```
intMatrica[0, 0] = 12;
intMatrica[1, 0] = 2;
floatMatrica[0, 3] = floatMatrica[0, 0] * floatMatrica[1, 0];
```


Matrica – inicijalizacija

- Kao i niz i matrica se može inicijalizovati elemenat po elemenat
- Idealno je koristiti (dve) **for** iteracije kada se zna tačan broj elemenata matrice
- Matrica se može inicijalizovati na sledeći način:

```
for (i = 0; i < br_vrst; i++)
    for (j = 0; j < br_kolona; j++)
        intMatrica[ i , j ] = eij;
```

- **Dve for** iteracije se koriste i za prolaz kroz sve elemente matrice (ili jednog njenog dela)

Niz nizova (Jagged Array) – deklaracija I inicijalizacija...

- Vrednosti elemenata niza nizova se najčešće ne inicijalizuju tokom deklaracije, jer nije poznata dimenzija unutrašnjih nizova
- Inicijalizacija se vrši nakon deklaracije
- Deklaraciju vršimo na sledeći način:

```
tip[][] identifikator = new tip [br_vrst][];
```

  - tip određuje - kog tipa će biti elementi niza
  - identifikator - definiše naziv niza
  - br_vrst – prva dimenzija

- Inicijalizacija se nakon toga vrši u nekoj od petlji (najčešće FOR petlji)

```
int[][] myJagArray = new int[3][];
for (int i = 0; i < myJagArray.Length; i++)
{
    myJagArray[i] = new int[i + 2];
}
```


Deklarisanje i definisanje niza nizova - primeri

```
int[][] nizNizova = new int[5][]; // deklarisan niz nizova

// pomalo neintuitivan zapis; ali svaki niz u nizu moze da ima razlicite
dimenzije te se stoga pojedinacno inicijalizuje

var dim1 = nizNizova.GetLength(0);
for (int i = 0; i < dim1; i++)
{
  nizNizova[i] = new int[i+1];
}
for (int i = 0; i < dim1; i++)
{
  var dim2 = nizNizova[i].GetLength(0);
  for (int j = 0; j < dim2; j++)
  {
    nizNizova[i][j] = j;
  }
}
```

Može i ovako!

```
int[][] jaggedArray3 =
{
    new int[] {1,3,5,7,9},
    new int[] {0,2,4,6},
    new int[] {11,22}
};
```

Pristupanje elementima niza nizova

- Elementima se pristupa preko operatora **[ind_vrste][ind_kolone]** gde su **ind_vrste** i **ind_kolone** indeksi (pozicije) elementa u nizu nizova

  0 ≤ indeks_vrste < br_vrsta
  0 ≤ indeks_kolone < br_kolona

- Identifikator niza nizova predstavlja **adresu 1. elem. matrice!**
- Primeri pristupanja elementima niza:

```
intMatrica[0][0] = 12;
intMatrica[1][0] = 2;
floatMatrica[0][3] = floatMatrica[0][0] * floatMatrica[1][0];
```


Niz nizova – inicijalizacija

- Kao i niz i niz nizova se može inicijalizovati elemenat po elemenat
- Idealno je koristiti (dve) **for** iteracije kada se zna tačan broj elemenata matrice
- Matrica se može inicijalizovati na sledeći način:

```
for (i = 0; i < br_vrst; i++)
    for (j = 0; j < br_kolona; j++)
        intMatrica[ i ][ j ] = eij;
```

- **Dve for** iteracije se koriste i za prolaz kroz sve elemente matrice (ili jednog njenog dela)

Matrica – primer

- **ZADATAK:**
  Implementirati program za računanje srednje vrednosti elemenata matrice prirodnih brojeva koji sadrži maksimum 10 vrsta i 20 kolona. Program prihvata od korisnika broj vrsta i kolona i vrednost svakog pojedinačnog elementa.

- **VEŽBA:**
  Proširiti prethodni zadatak tako da računa sumu elemenata svake pojedinačne vrste. Korisniku se prikazuje par vrsta – suma njenih elemenata.

Matrica – zadatak

```
/*********************************************************************************************************
* Primer za računanje srednje vrednosti elemenata matrice prirodnih brojeva koji sadrži maksimum 10 vrsta i 20
* kolona. Program prihvata od korisnika broj vrsta i kolona i vrednost svakog pojedinačnog elementa.
*********************************************************************************************************/
static void Main(string[] args){
            const int MAXVRSTA = 10;
            const int MAXKOLONA = 20;
            int Nvr = 1, Nkol = 1, i, j;
            float srvr = 0;
            int[,] MatricaPrirordnihBrojeva = new int[MAXVRSTA, MAXKOLONA];
            Console.WriteLine("Program za racunanje srednje vrednosti elemenata matrice prirodnih
brojeva.\n\n");
            Console.WriteLine("Unesite broj vrsta:\t"); // prihvati broj vrsta i kolona
            Nvr = int.Parse(Console.ReadLine());
            Console.WriteLine("Unesite broj kolona:\t");
            Nkol = int.Parse(Console.ReadLine());
            for (i = 0; i < Nvr; i++) // prihvati elemente matrice od korisnika
                for (j = 0; j < Nkol; j++)
                {
                    Console.WriteLine("Unesite element matrice na koordinati [" + (i + 1) + "][" + (j + 1)
+ "]:\t");
                    MatricaPrirordnihBrojeva[i,j] = int.Parse(Console.ReadLine());
                }
            for (i = 0; i < Nvr; i++) // izracunaj sumu elemenata matrice
                for (j = 0; j < Nkol; j++)
                    srvr += MatricaPrirordnihBrojeva[i,j];
            srvr /= (Nvr * Nkol); // izracunaj srednju vrednost elemenata matrice
            Console.WriteLine("\n\nSrednja vrednost elemenata matrice iznosi " + srvr + ".");
              Console.ReadKey();
          }
```

Strukture podataka – zadaci

1. Napisati program koji pronalazi minimalan i maksimalan element u matrici. Korisniku omogućiti da unese proizvoljan broj vrsta i kolona (do 20), odnosno da unese sve elemente te matrice.
   Dodatno: *Probajte isto i sa 3D matricom*

2. Napisati program koji omogućuje korisniku da unese proizvoljnu matricu (kao u zadatku 1) a potom mu tu matricu prikažite i omogućite da zameni bilo koje dve kolone ili bilo koje dve vrste.

3. Napisati program koji omogućuje korisniku da uneste proizvoljnu kvadratnu matricu, a potom pronađite srednju vrednost i sumu obe dijagonale matrice.

4. Napisati program kojim korisnik unosi ograničen (N) broj studenata, a potom prikazuje sve unete studente. Svaki Student je određen imenom i prezimenom, smerom, godinom i prosecnom ocenom.