



Polje kao argument funkcije

SIU1 Funkcije – ishod 3

Kada funkciji nije dovoljan jedan podatak

1D polje, 2D polje i vektor kao argument funkcije

- funkciji ne moramo slati svaki element zasebno – možemo joj predati cijelu zbirku podataka.
- Što funkcija treba dobiti? Kako pristupa elementima? Koji rezultat treba vratiti? Jednodimenzionalno polje je temeljni primjer, vector se uspoređuje s njim, a dvodimenzionalno polje uvodi se problemski kao prirodno proširenje za tablične podatke.
- 1) ideja zbirke kao argumenta + čitanje i projektiranje kratkih funkcija;
- 2) obrada elemenata, veličina i različiti tipovi zadataka;
- 3) složeniji problemski zadaci, 2D podaci i izbor prikladne strukture;
- 4) rekapitulacija i 15-minutna provjera ishoda.

Operativni ishodi

- objasniti zašto je u nekim problemima funkciji potrebno predati zbirku podataka, a ne niz pojedinačnih parametara;
- prepoznati jednodimenzionalno polje ili vektor kao ulazni podatak funkcije;
- odrediti što uz polje funkcija još mora znati (npr. broj elemenata) kako bi ga ispravno obradila;
- čitati kratku funkciju koja prima polje ili vektor te predvidjeti njezin rezultat za mali primjer;
- uočiti osnovnu razliku između predavanja klasičnog polja i vektora funkciji na razini uporabe;
- prepoznati dvodimenzionalno polje kao prikladan zapis podataka organiziranih u retke i stupce;
- predložiti ulazne podatke i povratnu vrijednost funkcije za jednostavan problem nad 1D ili 2D zbirkom;
- obrazložiti izbor: pojedinačni parametri, polje, vektor ili dvodimenzionalno polje.

PROBLEM → KOJI PODACI? → KAKO IH ORGANIZIRATI? → ŠTO
FUNKCIJA PRIMA? → ŠTO VRAĆA? → TEST

- Problem sadrži više podataka iste vrste,
- kako ih predstaviti
- oblikovati funkciju.

1. Problemski uvod – zašto zbirka kao argument?

- „Natjecatelj je na pet zadataka osvojio 12, 7, 18, 5 i 14 bodova. Želimo funkciju koja vraća najveći rezultat.”
- Koji su ulazni podaci funkcije?
- Bi li pet zasebnih parametara bilo moguće? Bi li bilo dobro rješenje?
- Što kad bi imali 50 rezultata?
- Što je u tim podacima zajedničko?
- Tek nakon zaključka prikazati usporedbu:
- **int najveći(int a, int b, int c, int d, int e); // moguće, ali nepraktično**

int najveći(int a[], int n); // zbirka + broj elemenata

- Ključni zaključak: **polje omogućuje da funkciji predamo skup elemenata istog tipa kroz jedan podatak**. Obično moramo dati i informaciju koliko elemenata treba obraditi.

Razumijevanje funkcije koja prima jednodimenzionalno polje

- ```
int najveci(int a[], int n)
{
 int m = a[0];

 for (int i = 1; i < n; i++)
 if (a[i] > m)
 m = a[i];

 return m;
}
```

Koji su parametri funkcije i što svaki predstavlja?

Koji je izlaz funkcije?

Zašto obrada počinje s `a[0]`?

Koje elemente funkcija promatra za `n = 5`?

Što bi se dogodilo kada bismo pogrešno poslali `n = 3` za polje od pet elemenata?

Ručno pratiti primjer: `a = {12, 7, 18, 5, 14}`, `n = 5` → rezultat 18.

## Problemski zadatak – prag

- Funkcija treba prebrojiti koliko je rezultata strogo većih od zadanog praga. Za {12, 7, 18, 5, 14} i prag 10 očekivani rezultat je 3.

|                        |                                                                     |
|------------------------|---------------------------------------------------------------------|
| Što funkcija prima?    | polje rezultata, broj elemenata, prag                               |
| Što funkcija vraća?    | broj elemenata većih od praga                                       |
| Kako obrađuje podatke? | prolazi element po element i povećava brojač kada je element > prag |
| Kako provjeravamo?     | mali primjer s očekivanim rezultatom                                |

jedno moguće rješenje:

```
int iznadPraga(int a[], int n, int
prag)
{
 int brojac = 0;

 for (int i = 0; i < n; i++)
 if (a[i] > prag)
 brojac++;

 return brojac;
}
```

Naglasak: zašto funkcija ima baš tri parametra i zašto vraća jedan cijeli broj.

## S vektorom

- ```
int iznadPraga(const vector<int>& v, int prag)
{
    int brojac = 0;

    for (int x : v)
        if (x > prag)
            brojac++;

    return brojac;
}
```

- funkcija dobiva vektor bez nepotrebnog kopiranja,
- `const` pokazuje da ga ova funkcija ne namjerava mijenjati.
- Važnija je razlika da vektor zna svoju veličinu pa poseban parametar `n` nije potreban.
- Pitanje za dublje razumijevanje: „Koji podatak postoji u prvoj funkciji, a nestao je iz druge – i zašto?”

Dvodimenzionalno polje – kada podaci imaju retke i stupce

- rezultati triju učenika na četiri zadatka:

	Z1	Z2	Z3	Z4
U0	10	15	8	12
U1	20	5	17	9
U2	7	14	11	18

Problem: funkcija za zadanog učenika vraća ukupan broj bodova. Prije koda učenici zaključuju da funkcija mora znati tablicu i koji redak obrađuje; broj stupaca je u ovom primjeru poznat.

```
int zbrojRetka(int bodovi[][4], int redak)
{
    int zbroj = 0;

    for (int j = 0; j < 4; j++)
        zbroj += bodovi[redak][j];

    return zbroj;
}
```

`bodovi[i][j]` označava element u retku `i` i stupcu `j`.

Provjera: `zbrojRetka(bodovi, 1) → 51`.

Mini izazov – odaberi prikladan način prijenosa

Problem	Zaglavlje funkcije	Obrazloženje
Program ima točno tri duljine stranica trokuta, a funkcija vraća opseg.		
Program obrađuje rezultate svih učenika u razredu, a njihov se broj može mijenjati.		
Program čuva temperature za 7 dana, po 4 mjerenja svakog dana.		

razumno rješenje prihvatljivo je ako ga se može argumentirati.

Cilj je povezati strukturu podataka sa strukturom problema.

glavni cilj: razviti sposobnost da učenik iz problema prepozna zbirku podataka kao argument funkcije i smisleno odredi njezino sučelje

- Polje ili vektor možemo predati funkciji kao zbirku podataka istog tipa.
 - Kod klasičnog 1D polja funkciji često treba i broj elemenata n .
 - vektor **zna** svoju veličinu (`size()`).
 - 2D polje koristimo kada podatke prirodno promatramo kroz retke i stupce.
 - Prije pisanja funkcije pitamo: ŠTO PRIMA? ŠTO VRAĆA? KAKO ĆEMO TESTIRATI?



Polje kao argument funkcije – dio 2.

SIU1 Funkcije – ishod 3

Operativni ishodi sata

- prepoznati zajednički algoritamski obrazac prolaska kroz elemente polja;
- predvidjeti rezultat funkcije koja zbraja ili prebrojava odabrane elemente;
- objasniti kako se tijekom prolaska određuju minimum ili maksimum;
- razlikovati funkciju koja traži vrijednost od funkcije koja obrađuje sve elemente;
- prepoznati kada funkcija samo vraća rezultat, a kada mijenja sadržaj proslijeđenog polja;
- predložiti prikladan vlastiti test, uključujući rubni ili nezgodni slučaj.

- isti temeljni obrazac:
 - PROĐI KROZ ELEMENTE -> PROMATRAJ TRENUTNI ELEMENT -> AŽURIRAJ REZULTAT / DONESI ODLUKU -> VRATI REZULTAT
-
- stalno postavljati pitanja:
 - Što pamtimo dok prolazimo poljem?
 - Kada tu vrijednost mijenjamo?
 - Što funkcija na kraju vraća?

`int a[] = {7, 12, 4, 12, 9};`
što tijekom prolaska treba pamtit?

Problem	Što pamtimo?	Očekivani rezultat
Zbroj svih elemenata	tekući zbroj	44
Koliko je elemenata > 8?	brojač	3
Najveći element	trenutni maksimum	12

mijenja se podatak koji pamtimo i pravilo po kojem ga ažuriramo.

Zbrajanje i prebrojavanje

- ```
int zbroj(int a[], int n)
{
 int s = 0;
 for (int i = 0; i < n; i++)
 s += a[i];
 return s;
}
```
- ```
int brojVecih(int a[], int n, int granica)
{
    int brojac = 0;
    for (int i = 0; i < n; i++)
        if (a[i] > granica)
            brojac++;
    return brojac;
}
```

Koja je uloga varijable s, a koja varijable brojac?

Zašto obje počinju s 0?

Za $a = \{7, 12, 4, 12, 9\}$, što vraća `brojVecih(a, 5, 8)`?

Što bi se promijenilo kada bismo tražili elemente $\geq$ granica?

Minimum i maksimum - važnost početne vrijednosti

- Problemski kontekst: funkcija treba vratiti najnižu temperaturu iz n mjerenja. Predložiti pogrešan početak `double najmanja = 0;` i pitati radi li za mjerenja {18.5, 20.1, 17.9, 19.0}.
- Očekivani zaključak: ne radi - vratila bi 0, iako 0 nije element polja.

```
double minimum(double a[], int n)
```

```
{  
    double m = a[0];  
    for (int i = 1; i < n; i++)  
        if (a[i] < m)  
            m = a[i];  
    return m;  
}
```

Pretraživanje - mora li funkcija uvijek proći cijelo polje?

- Problem: provjeri postoji li u polju zadana vrijednost.

- ```
bool postoji(int a[], int n, int trazeni)
{
 for (int i = 0; i < n; i++)
 if (a[i] == trazeni)
 return true;

 return false;
}
```

Zašto je bool prirodan povratni tip?

Kada funkcija može završiti prije posljednjeg elementa?

Gdje mora stajati `return false` i zašto ne smije biti unutar petlje nakon prvog neuspjelog uspoređivanja?

Za  $a = \{7, 12, 4, 12, 9\}$ , koliko elemenata mora pregledati pri traženju 4? A pri traženju 6?

učinkovitost: ponekad je rezultat poznat prije nego što obradimo sve elemente.

# Izmjena elemenata - kada rezultat ostaje u polju?

- Problemski kontekst: senzor ponekad daje negativnu vrijednost, a u pojednostavljenom sustavu takva se očitavanja zamjenjuju nulom.
- `void ukloniNegativne(int a[], int n)`

```
{
 for (int i = 0; i < n; i++)
 if (a[i] < 0)
 a[i] = 0;
}
```
- Primjer: {-2, 5, -1, 8} -> {0, 5, 0, 8}
- što funkcija treba vratiti ako je cilj promijeniti podatke?

funkcija ne vraća izračunatu vrijednost, nego mijenja sadržaj proslijeđenog polja.

Što je ovdje 'rezultat' rada funkcije?

Zašto povratni tip može biti void?

Kako bismo testirali da je funkcija doista promijenila podatke?

razlikovati rezultat obrade od izmijenjenog sadržaja polja

## Mini problemski izazov - odstupanje senzora

- AI sustav prati mjerenja senzora.  
Poznata je referentna vrijednost R.  
Potrebno je odrediti koliko  
mjerenja odstupa od R za više od 5  
jedinica.
- Primjer: mjerenja {20, 27, 15, 23,  
31}, R = 22 -> odstupaju 15 i 31 ->  
rezultat 2.

predložiti:

- parametre funkcije;
- povratni tip;

što funkcija pamti tijekom prolaska?

uvjet kojim prepoznaje odstupanje?

jedan vlastiti rubni test.

Mogući potpis nakon rasprave:

```
int brojOdstupanja(int a[], int n, int referentna);
```

Za cjelobrojne podatke uvjet se može zapisati kao  
``abs(a[i] - referentna) > 5``.

Vrijednost točno 5 ne broji se jer zadatak kaže 'više od 5'.

- ZBRAJANJE: pamtimo tekući zbroj.
  - PREBROJAVANJE: pamtimo brojač i povećavamo ga kada je uvjet ispunjen.
  - MIN/MAX: početna vrijednost može biti prvi element polja.
  - PRETRAŽIVANJE: kada pronađemo traženo, ponekad možemo odmah završiti.
  - IZMJENA: funkcija može promijeniti elemente polja; rezultat ne mora biti povratna vrijednost.
- 
- Uvijek prije pokretanja smisli mali test i očekivani rezultat.



# Polje kao argument funkcije – dio 3.

SIU1 Funkcije – ishod 3

# Jednodimenzionalno polje, vektor i dvodimenzionalno polje kao argumenti funkcije

- povezati dosadašnje obrasce
- Razlikovati dvije uloge funkcije:
  - obraditi zbirku i vratiti jednu izračunatu vrijednost ili
  - promijeniti same proslijeđene podatke.
- prošireno na vektor i 2D polje
- NAKON POZIVA FUNKCIJE – GDJE SE NALAZI REZULTAT?
- Ako funkcija računa zbroj, minimum, broj elemenata ili postoji/ne postoji, rezultat je obično povratna vrijednost.
- Ako funkcija ispravlja, normalizira ili mijenja elemente, rezultat može biti novo stanje proslijeđene zbirke.

# Operativni ishodi

- razlikovati funkciju koja vraća rezultat od funkcije koja mijenja elemente zbirke;
- predvidjeti povratnu vrijednost i stanje polja nakon poziva funkcije;
- objasniti učinak predaje vektora po vrijednosti i po referenci;
- čitati jednostavnu funkciju koja obrađuje redak ili stupac 2D polja;
- za novi problem odrediti potrebne parametre, povratnu vrijednost i treba li funkcija mijenjati podatke;
- osmisliti test s očekivanim rezultatom.

## Aktivacija – dvije funkcije, dva mjesta rezultata

- `int maksimum(int a[], int n)`

```
{
 int m = a[0];
 for (int i = 1; i < n; i++)
 if (a[i] > m) m = a[i];
 return m;
}
```

Za `a = {4, -2, 9, -5, 3}` : `maksimum(a,5)` vraća 9 i ne mijenja polje;

nakon `postaviNule(a,5)` polje je `{4,0,9,0,3}`, a funkcija ne vraća vrijednost.

```
void postaviNule(int a[], int n)
```

```
{
 for (int i = 0; i < n; i++)
 if (a[i] < 0) a[i] = 0;
}
```

Pitanje: Zašto bi bilo loše da funkcija koja samo broji negativna očitavanja usput mijenja ta očitavanja?

to nije dio njezine zadaće i izgubili bismo izvorne podatke.

# Vector – vrijednost ili referenca?

Usporediti dvije gotovo jednake funkcije:

```
void povecaj1(vector<int> v)
```

```
{
 for (int& x : v) x++;
}
```

```
void povecaj2(vector<int>& v)
```

```
{
 for (int& x : v) x++;
}
```

Za `v = {2,5,8}` : nakon `povecaj1(v)` izvorni v ostaje `{2,5,8}` jer funkcija **mijenja kopiju**.

Nakon `povecaj2(v)` izvorni v postaje `{3,6,9}` jer **referenca omogućuje izmjenu izvornika**.

Za funkciju koja samo čita podatke potpis  
`int zbroj(const vector<int>& v);` .

Nema kopiranja, ali ni dopuštene izmjene.

# obrada retka i izmjena stupca

- AI sustav bilježi očitavanja triju senzora u četiri vremenska trenutka:

|    | t0 | t1 | t2 | t3 |
|----|----|----|----|----|
| S0 | 12 | 14 | -2 | 15 |
| S1 | 8  | 11 | 10 | 13 |
| S2 | -1 | 9  | 7  | 16 |

Funkcija koja vraća maksimum zadanog retka:

```
int maksimumRetka(int a[][4], int redak)
{
 int m = a[redak][0];
 for (int j = 1; j < 4; j++)
 if (a[redak][j] > m) m = a[redak][j];
 return m;
}
```

`maksimumRetka(a,0)` vraća 15; tablica ostaje nepromijenjena.

Funkcija koja mijenja zadani stupac:

```
void ispraviStupac(int a[][4], int brojRedaka, int stupac)
{
 for (int i = 0; i < brojRedaka; i++)
 if (a[i][stupac] < 0)
 a[i][stupac] = 0;
}
```

Nakon `ispraviStupac(a,3,2)` vrijednost -2 postaje 0. Ostali elementi ostaju isti.

Zašto prva funkcija treba parametar `redak`, a druga `stupac`? Koja funkcija vraća vrijednost, a koja mijenja strukturu? Koji bi test dokazao da se ne mijenjaju elementi izvan zadanog stupca?

# Problemska sinteza – AI senzorska mreža

- Ne pisati tijelo funkcije. Za svaki zahtjev odrediti parametre, povratni tip, mijenja li funkcija podatke i jedan test.

| Zahtjev                                                                | Mogući potpis                                                            | Učinak         | Zašto?                               |
|------------------------------------------------------------------------|--------------------------------------------------------------------------|----------------|--------------------------------------|
| Koliko je vrijednosti u vektoru veće od zadane granice?                | <code>int brojVecih(const vector&lt;int&gt;&amp; v, int granica);</code> | vraća int      | Podatke samo čita i prebrojava.      |
| Sva negativna mjerenja u vektoru zamijeni nulom.                       | <code>void ispravi(vector&lt;int&gt;&amp; v);</code>                     | mijenja vector | Rezultat je novo stanje iste zbirke. |
| U tablici senzori × vrijeme vrati najmanje očitavanje zadanog senzora. | <code>int minimumRetka(int a[][4], int redak);</code>                    | vraća int      | Obrađuje jedan redak bez izmjene.    |

Funkcija dobiva `vector<int>` s brojem riješenih zadataka po danima. Sve negativne vrijednosti treba zamijeniti nulom. Treba li vratiti novi vektor ili može promijeniti postojeći?

Napiši mogući potpis i obrazloži jednom rečenicom.

Mogući odgovor:

```
void ispravi(vector<int>& v);
```

Može promijeniti postojeći vektor jer je proslijeđen po referenci (adresi); povratna vrijednost nije potrebna.

Prihvatiti i dobro obrazloženo rješenje koje vraća novu kopiju ako učenik jasno razumije razliku.

- Za svaku funkciju treba moći odgovoriti:
- 1. ŠTO PRIMA?  
2. ŠTO VRAĆA?  
3. MIJENJA LI PROSLIJEĐENE PODATKE?  
4. KAKO ĆU PROVJERITI DA RADI?
- Bitna sintaksa. Važno je čitati funkciju, predvidjeti njezin učinak i iz problemskog zahtjeva odrediti prikladne parametre i rezultat.
- Funkcija može obraditi zbirku i VRATITI jednu vrijednost.
- Funkcija može MIJENJATI elemente prosljeđene zbirke.
- Kod klasičnog polja promjene elemenata u funkciji vide se nakon poziva.
- Kod vektora: bez `&` dobivamo kopiju; s `&` možemo mijenjati izvorni vektor.
- `const vector<T>&` koristimo kada želimo čitati bez kopiranja i bez promjene.
- Kod 2D polja često su redak ili stupac dodatni parametri.
- Prije koda: Što prima? Što vraća? Mijenja li podatke? Kako testirati?