

# Ugrađene funkcije

# Ugrađene matematičke funkcije

---

```
#include <math.h>
```

zaglavna (*header*)  
datoteka u kojoj su  
funkcije **deklarirane**

prototipovi u math.h

```
int abs (int x);  
long labs (long x);  
double fabs (double x);
```

Sve tri funkcije izračunavaju apsolutnu vrijednost

# Funkcije abs, fabs: zašto?

1.

```
double abs(double x) {  
    return x < 0 ? -x : x;  
}
```

ili

2.

```
int abs(int x) {  
    return x < 0 ? -x : x;  
}
```

```
int main() {  
    int x;  
    x = abs(-5);  
    printf("%d\n", x);  
    return 0;  
}
```

- Hoće li se u oba slučaja (definicija funkcije 1. ili 2.) dobiti isti rezultat?
- U kojem slučaju se obavlja manji broj konverzija tipova podataka?

# Funkcije abs, fabs: zašto?

1.

```
double abs(double x) {  
    return x < 0 ? -x : x;  
}
```

ili

2.

```
int abs(int x) {  
    return x < 0 ? -x : x;  
}
```

```
int main() {  
    double x;  
    x = abs(-5.2);  
    printf("%f\n", x);  
    return 0;  
}
```

- Hoće li se u oba slučaja (definicija funkcije 1. ili 2.) dobiti isti rezultat?

# Funkcije abs, fabs: česte pogreške

- Napisati program koji ispisuje vrijednosti  $x$  i  $|x|$  za  $x$  u intervalu od -2 do 2 s korakom 0.5

```
#include <stdio.h>
#include <math.h>
int main() {
    double x, absX;
    for (x = -2.; x <= 2.; x+=0.5) {
        absX = abs(x);
        printf("%4.1f %4.1f\n", x, absX);
    }
    return 0;
}
```

-2.0	2.0
-1.5	1.0
-1.0	1.0
-0.5	0.0
0.0	0.0
0.5	0.0
1.0	1.0
1.5	1.0
2.0	2.0

Ispravan rezultat će se dobiti korištenjem funkcije fabs

# Ugrađene matematičke funkcije

---

```
#include <math.h>
```

<code>double sin (double x);</code>	<i>sin x</i>
<code>double cos (double x);</code>	<i>cos x</i>
<code>double tan (double x);</code>	<i>tan x</i>
<code>double asin (double x);</code>	<i>arcsin x</i>
<code>double acos (double x);</code>	<i>arccos x</i>
<code>double atan (double x);</code>	<i>arctan x</i>

# Ugrađene matematičke funkcije

---

```
#include <math.h>
```

<code>double sinh (double x);</code>	<i>sh x</i>
<code>double cosh (double x);</code>	<i>ch x</i>
<code>double tanh (double x);</code>	<i>th x</i>
<code>double exp (double x);</code>	$e^x$
<code>double log (double x);</code>	$\ln x$
<code>double log10 (double x);</code>	$\log x$
<code>double pow (double x, double y);</code>	$x^y$
<code>double sqrt (double x);</code>	$\sqrt{x}$

# Ugrađene matematičke funkcije

---

```
#include <math.h>
```

```
double fmod (double x, double y);
```

```
double ceil (double x);
```

```
double floor (double x);
```

fmod: ostatak dijeljenja  $x / y$

ceil: najmanji cijeli broj koji je veći ili jednak  $x$

floor: najveći cijeli broj koji je manji ili jednak  $x$

- Primjeri:

<code>fmod(3.82, 0.7)</code>	<code>→</code>	<code>0.32</code>
<code>ceil(-5.2)</code>	<code>→</code>	<code>-5.0</code>
<code>ceil(5.001)</code>	<code>→</code>	<code>6.0</code>
<code>floor(-5.2)</code>	<code>→</code>	<code>-6.0</code>
<code>floor(5.999)</code>	<code>→</code>	<code>5.0</code>

# Ugrađene posebne funkcije iz <stdlib.h>

---

```
#include <stdlib.h>
```

```
void exit (int status);
```

`exit (x)` trenutno prekida izvođenje programa i pozivajućem programu (operacijskom sustavu) vraća vrijednost `x`

izvršavanje `exit (x)`; u funkciji `main` je ekvivalentno s izvršavanjem `return x`; u funkciji `main`

# Funkcija exit

```
#include <stdio.h>
#include <stdlib.h>
int fun(int x) {
    return 1;
}

int main() {
    int i;
    i = fun(5);
    printf("%d\n", i);
    return 0;
}
```

Ispis na zaslon: 1

Op. sustavu vraća se 0

```
#include <stdio.h>
#include <stdlib.h>
int fun(int x) {
    exit(1);
}

int main() {
    int i;
    i = fun(5);
    printf("%d\n", i);
    return 0;
}
```

Ispis na zaslon: ništa

Op. sustavu vraća se 1

# Ugrađene posebne funkcije iz <stdlib.h>

---

## Generiranje pseudoslučajnih brojeva

```
#include <stdlib.h>
```

```
int rand (void);
```

- generira pseudoslučajni cijeli broj iz intervala [0, **RAND\_MAX**]
- **RAND\_MAX** je simbolička konstanta iz [stdlib.h](#). Iznos konstante ovisi o arhitekturi i prevodiocu: u VS C prevodiocu **RAND\_MAX=32767**
- svakim novim pozivom funkcije `rand()` dobije se novi pseudoslučajni broj - uzastopnim pozivanjem funkcije dobiva se niz pseudoslučajnih brojeva

Primjer: na zaslon ispisati niz od 10 pseudoslučajnih brojeva iz intervala [0, 32767] (uz VS C prevodilac)

---

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int i;
    for (i = 0; i < 10; i++) {
        printf("%5d\n", rand());
    }
    return 0;
}
```

41  
18467  
6334  
26500  
19169  
15724  
11478  
29358  
26962  
24464

Ponovljenim izvršavanjem istog programa dobit će se **isti niz!**

# Kako dobiti različite nizove pseudoslučajnih brojeva?

---

```
void srand (unsigned int seed);
```

- inicijalizira generator pseudoslučajnih brojeva. Za isti *seed* dobije se uvijek isti niz pseudoslučajnih brojeva

	seed=1	seed=2
<pre>#include &lt;stdio.h&gt;</pre>		
<pre>#include &lt;stdlib.h&gt;</pre>		
<pre>int main() {</pre>		
<pre>    int i, seed;</pre>		
<pre>    scanf ("%d", &amp;seed);</pre>		
<pre>    srand (seed);</pre>		
<pre>    for (i = 0; i &lt; 10; i++) {</pre>		
<pre>        printf ("%5d\n", rand());</pre>		
<pre>    }</pre>		
<pre>    return 0;</pre>		
<pre>}</pre>		
	41	45
	18467	29216
	6334	24198
	26500	17795
	19169	29484
	15724	19650
	11478	14590
	29358	26431
	26962	10705
	24464	18316

# Kako dobiti različite nizove pseudoslučajnih brojeva?

```
#include <time.h>           broj sekundi proteklih nakon
time_t time(NULL);         ponoći, 1. siječnja 1970. (GMT)

#include <stdio.h>          t1=1260807869   t2=1260807887
#include <stdlib.h>         1588           1647
#include <time.h>          17225          14089
int main() {               13370           7245
    int i;                 20489           27645
    srand( (unsigned)time(NULL) );
    for (i = 0; i < 10; i++) {
        printf("%5d\n", rand());
    }
    return 0;
}
```

	1588	1647
	17225	14089
	13370	7245
	20489	27645
	15147	4201
	24261	29406
	8937	32193
	28626	8703
	2696	4993
	3521	23927

# Prilagodba intervala generiranih brojeva

---

Primjer: na zaslon ispisati niz od 10 pseudoslučajnih brojeva iz intervala [1, 6]

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    int i;
    srand((unsigned)time(NULL));
    for (i = 0; i < 10; i++) {
        printf("%d\n", rand() % 6 + 1);
    }
    return 0;
}
```

6  
5  
3  
1  
2  
2  
4  
5  
6  
2

Brojevi iz intervala [a, b]:  $\text{rand}() \% (b-a+1) + a;$

# Još jedna mogućnost

Kako jednoliko preslikati cijele brojeve  $x$  iz intervala  $[a, b]$  u interval  $[c, d]$ ?

$$y = \underbrace{(x - a) / (b - a + 1)}_{[0, 1)} * (d - c + 1) + c$$

skaliranje translacija

*realno dijeljenje*

Primijenjeno na "bacanje kocke":  $[0, \text{RAND\_MAX}] \rightarrow [1, 6]$

$$(\text{float}) \text{ rand}() / (\text{RAND\_MAX} + 1) * 6 + 1$$

0- 5461 → 1	16384-21845 → 4
5462-10922 → 2	21846-27306 → 5
10923-16383 → 3	27307-32767 → 6

**Primjer:** Načiniti funkciju koja simulira bacanje kocke. Baciti kocku zadani broj puta. Ispisati frekvenciju pojavljivanja svih brojeva.

---

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int kocka () {
    float y = (float) rand() / (RAND_MAX+1) * 6 + 1;
    return (int)y;
}

int main () {
    int brojac[6] = {0};
    int i, n;
    printf ("Unesite broj bacanja kocke >");
    scanf ("%d", &n);
    srand ((unsigned) time(NULL));
```

# Primjer: Bacanje kocke, nastavak

---

```
for (i = 0; i < n; i++) {
    ++brojac[kocka()-1];
}
for (i = 0; i < 6; i++) {
    printf ("%d %5d\n", i+1, brojac[i]);
}
return 0;
}
```

Primjer izvođenja programa:

**Unesite broj bacanja kocke >1000000**

**1 166579**

**2 166446**

**3 166802**

**4 167009**

**5 166714**

**6 166450**

# Razlika između konstantnog znakovnog niza i niza znakova

- Niz znakova: jednodimenzionalno polje znakova s '`\0`':

```
#define DULJINA_NIZA 8
```

```
char ime_niza[DULJINA_NIZA + 1];
```

**polje** čiji su elementi inicijalizirani na 'I', 'v', 'a', 'n', '\0'

isto

```
char ime1[5] = {'I', 'v', 'a', 'n', '\0'};
```

```
char ime1[] = "Ivan";
```

```
char *ime2 = "Ana";
```

**pokazivač** koji pokazuje na konstantni znakovni niz "Ana"

# Razlika između konstantnog znakovnog niza i niza znakova

## ■ Primjer:

```
char *ime = "          ";  
ime[2] = 'A'; ili *(ime+2) = 'A'; /*nije dopušteno*/
```

`ime` pokazuje na konstantni znakovni niz čiji se sadržaj ne smije mijenjati

## ■ Primjer:

```
char *ime;  
char polje[3+1] = "    ";  
polje[2] = 'A'; /*dopušteno*/  
ime = polje;           ili ime = &polje[0];  
ime[2] = 'A'; ili *(ime+2) = 'A'; /*dopušteno*/
```

`ime` pokazuje na prvi element polja znakova (čiji je elemente dopušteno mijenjati)

# Ugrađene funkcije za operacije nad nizovima znakova

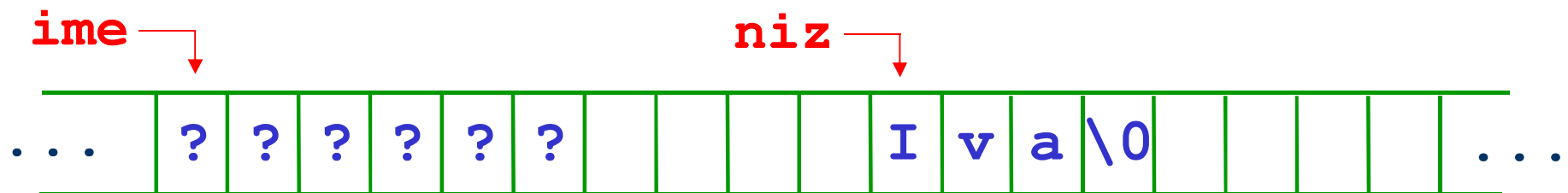
## `strcpy` `<string.h>`

- Kopiranje niza znakova (kopira src u dest, uključujući `\0`, vraća `dest`)

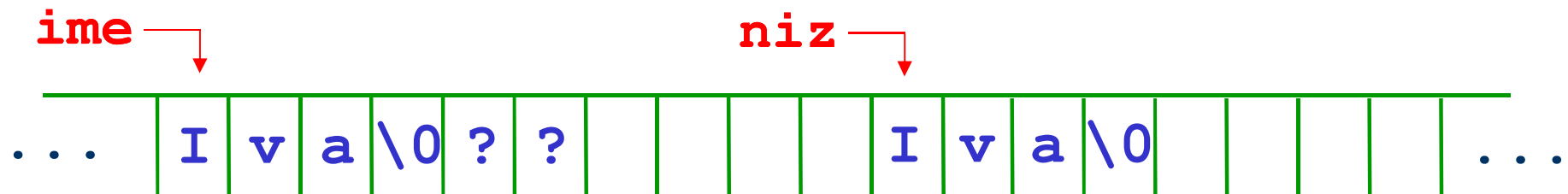
```
char *strcpy(char *dest, const char *src);
```

```
char ime[5+1];
```

```
char niz[] = "Iva";    ili char *niz = "Iva";
```



```
strcpy (ime, niz);    ili strcpy (ime, "Iva");
```



Što bi se dogodilo da je polje definirano ovako: `char ime[2]` ?

# Prvi argument funkcije `strcpy` ne smije biti konstantni znakovni niz

- **Primjer:**

```
char *ime = " ";  
char niz[] = "Iva";      ili char *niz = "Iva";  
strcpy (ime, niz);      /* nije dopušteno */
```

`ime` pokazuje na konstantni znakovni niz čiji se sadržaj ne smije mijenjati

- **Primjer:**

```
char *ime;  
char polje[3+1];  
char niz[] = "Iva";      ili char *niz = "Iva";  
strcpy (polje, niz);     /* dopušteno */  
ime = polje;             ili ime = &polje[0];  
strcpy (ime, niz);      /* dopušteno */
```

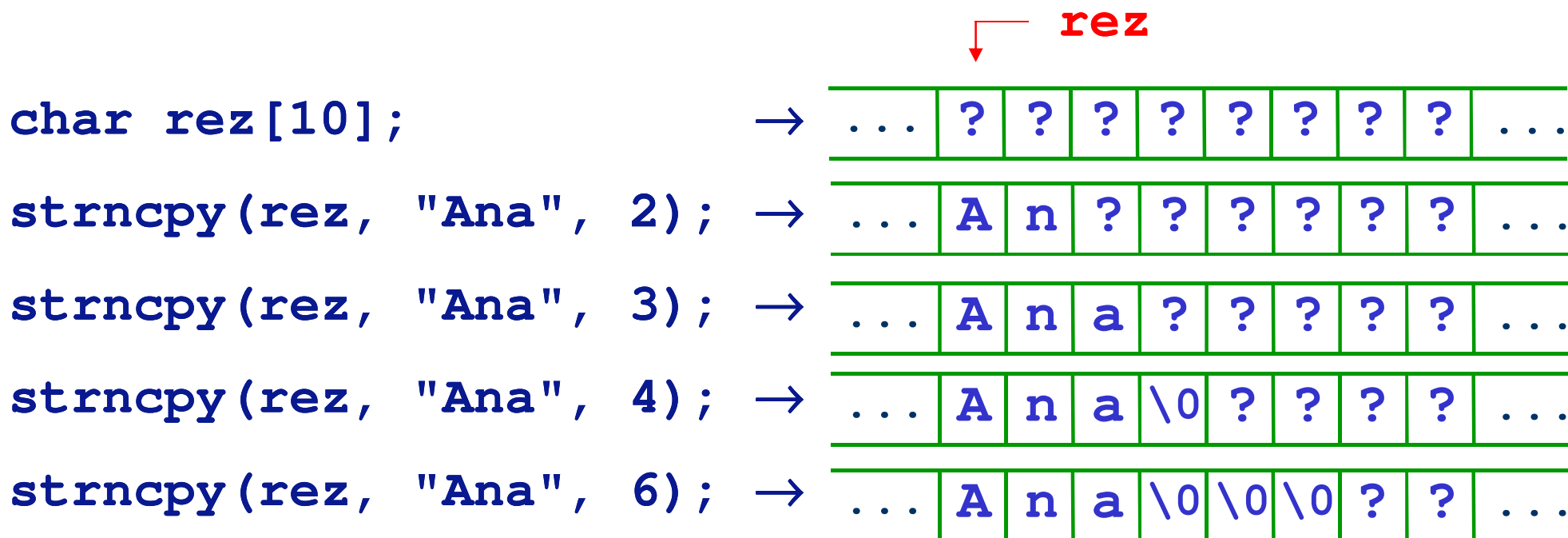
`ime` pokazuje na prvi element polja znakova (čiji je elemente dopušteno mijenjati)

# Ugrađene funkcije za operacije nad nizovima znakova

## `strncpy` `<string.h>`

- Kopiranje **dijela** niza znakova: kopira se maxlen znakova iz niza src u niz dest (to znači da se \0 možda neće uspjeti kopirati!). Ako je maxlen veći od duljine niza koji se kopira, u dest se dodaju \0 znakovi dok se ne dospije do duljine maxlen. Funkcija vraća dest.

```
char *strncpy(char *dest,  
              const char *src,  
              size_t maxlen);
```



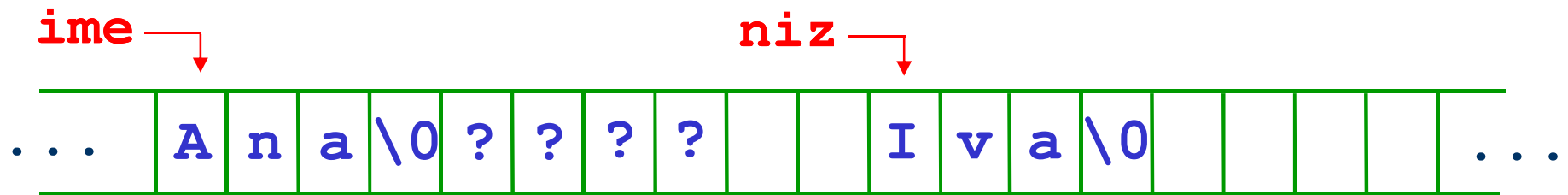
# Ugrađene funkcije za operacije nad nizovima znakova

## `strcat` `<string.h>`

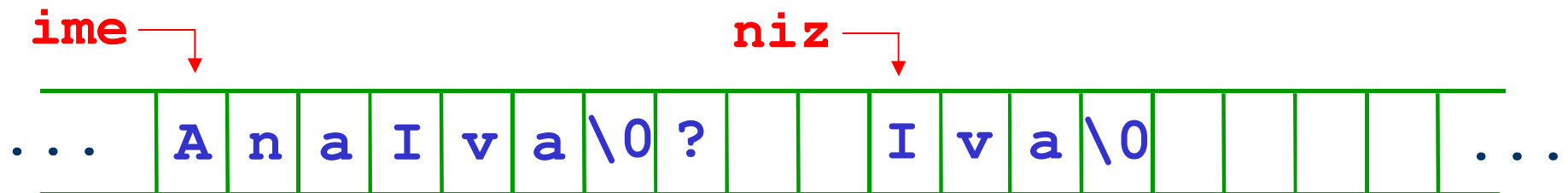
- Konkatenacija (nadovezivanje) nizova znakova: na kraj niza `dest` dodaje (kopira) sve znakove iz niza `src` i `\0`. Funkcija vraća `dest`.

```
char *strcat(char *dest, const char *src);
```

```
char ime[7+1];  
char niz[] = "Iva";  
strcpy(ime, "Ana");
```



```
strcat(ime, niz);
```



# Ugrađene funkcije za operacije nad nizovima znakova

## `strlen` `<string.h>`

---

- Duljina niza: vraća broj znakova u nizu. Ne broji `\0`.

```
size_t strlen(const char *s);
```

```
char niz[] = "Pero";
```

```
char *p = "Ana";
```

```
char polje[3] = {'I', 'v', 'a'};
```

```
strlen(niz);    → 4
```

```
strlen(p);     → 3
```

```
strlen(polje); → nije poznato gdje se u memoriji nalazi \0
```

# Ugrađene funkcije za operacije nad nizovima znakova

## `strcmp` `<string.h>`

---

- Usporedba nizova: leksikografski uspoređuje nizove `s1` i `s2`.
  - vraća 0 ako su nizovi jednaki
  - vraća cijeli broj  $< 0$  ako je `s1 < s2`
  - vraća cijeli broj  $> 0$  ako je `s1 > s2`

```
int strcmp(const char *s1, const char *s2);
```

```
strcmp("abcd", "abrd");    → -1
```

```
strcmp("abc", "abcd");    → -1
```

```
strcmp("abcd", "abc");    → 1
```

```
strcmp("abcd", "abcc");   → 1
```

```
strcmp("aBc", "abc");     → -1
```

```
strcmp("aBc", "aBc");    → 0
```

# Ugrađene funkcije za operacije nad nizovima znakova

## `strncmp` `<string.h>`

---

- Usporedba nizova: leksikografski uspoređuje **najviše** `maxlen` znakova u nizovima `s1` i `s2`.
  - vraća 0 ako su podnizovi jednaki
  - vraća cijeli broj  $< 0$  ako je podniz `s1`  $<$  podniz `s2`
  - vraća cijeli broj  $> 0$  ako je podniz `s1`  $>$  podniz `s2`

```
int strncmp(const char *s1,  
            const char *s2,  
            size_t maxlen);
```

```
strncmp("abcd", "abrd", 2);    → 0
```

```
strncmp("abc", "abcd", 5);     → -1
```

```
strncmp("abcd", "abc", 4);     → 1
```

```
strncmp("bcd", "abc", 1);      → 1
```

# Ugrađene funkcije za operacije nad nizovima znakova

## `strchr` `<string.h>`

---

- Traženje znaka unutar niza: vraća **pokazivač** na prvi znak vrijednosti `c` unutar niza znakova `s`. Ako takav znak ne postoji, vraća `NULL`.

```
char *strchr(const char *s, int c);
```

```
char niz[] = "Pero";
```

```
char *p = "Ana-Marija";
```

```
printf("%c", *strchr(niz, 'r')); → r
```

```
printf("%s", strchr(p, 'a')); → a-Marija
```

```
strchr(p, 'Z'); → vraća NULL
```

# Ugrađene funkcije za operacije nad nizovima znakova

## `strstr` `<string.h>`

---

- Traženje podniza unutar niza: u nizu `s1` pronalazi prvi podniz koji je jednak nizu `s2` i vraća pokazivač na prvi znak tog podniza. Ako odgovarajući podniz ne postoji, vraća `NULL`.

```
char *strstr(const char *s1, const char *s2);
```

```
char niz[] = "Nigdar ni tak bilo da ni nekak bilo";  
printf("%s", strstr(niz, "ni"));
```

→ ni tak bilo da ni nekak bilo

```
strstr(niz, "Tak"); → vraća NULL
```

# Funkcije nad znakom

<ctype.h>

- Pretvorba u veliko slovo

```
int toupper(int ch);
```

```
printf("%c", toupper('r')); → R
```

```
printf("%c", toupper('R')); → R
```

```
printf("%c", toupper('3')); → 3
```

- Pretvorba u malo slovo

```
int tolower(int ch);
```

```
printf("%c", tolower('R')); → r
```

```
printf("%c", tolower('r')); → r
```

```
printf("%c", tolower('3')); → 3
```

# Macro nad znakom

`<ctype.h>`

└─ logička vrijednost

`int isdigit(int c);` znamenka (0–9)

U `<ctype.h>` je definiran macro

```
#define isdigit(c) ((c) >= '0' && (c) <= '9')
```

`int isalpha(int c);` slovo (**A–Z** ili **a–z**)

`int isalnum(int c);` slovo (**A–Z** ili **a–z**) ili znamenka (0–9)

`int isprint(int c);` znak koji se može ispisati (**0x20–0x7E**)

`int iscntrl(int c);` kontrolni znak (**0x00–0x1F** ili **0x7F**)

`int isspace(int c);` praznina

`int islower(int c);` slovo (**a–z**)

`int isupper(int c);` slovo (**A–Z**)

# Učitavanje i ispis podataka

---

Učitavanje:

`getchar`

`scanf`

`gets`

Ispis:

`putchar`

`printf`

`puts`

# getchar

<stdio.h>

---

- Protip funkcije:  
`int getchar(void);`
- Učitava jedan znak.
- Uspješno pročitani znak pretvara u cijeli broj.
- Ako pročita znak koji odgovara kraju datoteke (na DOS-u je to 0x1A ili ^Z, na Unix-u je to 0x04 ili ^D), tada vraća macro vrijednost EOF.
- Ako se pri čitanju dogodi pogreška, vraća EOF.
- macro EOF (end-of-file) je definiran u <stdio.h>

# Primjer: čitati i ispisivati znak po znak

IOGetchar

```
#include <stdio.h>

int main() {
    char c;
    while(1) {
        c = getchar();
        printf("%d ", c);
        if (c == EOF) break;
    }
    return 0;
}
```

Rezultat izvođenja programa:

```
aAB↵
97 65 66 10 /↵
47 10 <Ctrl+Z>↵
-1
```

# putchar

<stdio.h>

- Prototip funkcije:

```
int putchar(int ch);
```

- Ispisuje jedan znak.
- Vraća vrijednost uspješno ispisanog znaka ili vraća EOF ukoliko ispis znaka nije uspio.

```
#include <stdio.h>
int main() {
    int i;
    for (i = 'A'; i <= 'Z'; i++)
        putchar(i);
    return 0;
}
```

→ **ABCDEFGHIJKLMNOPQRSTUVWXYZ**

**Primjer:** Učitati i ispisati niz znakova, pri čemu sva mala slova treba ispisati kao velika. Znakove učitavati dok se ne učitava znak '\n'

---

📁 IOPutchar

```
#include <stdio.h>
#include <ctype.h>
int main() {
    char slovo[80+1];
    int i, n;

    /* ucitavanje retka teksta */
    for (i = 0; (slovo[i] = getchar()) != '\n'; i++);

    /* broj procitanih znakova */
    n = i;

    /* ispis retka, ali velikim slovima */
    for(i = 0; i < n; ++i)
        putchar(toupper(slovo[i]));

    return 0;
}
```

# Izvođenje programa

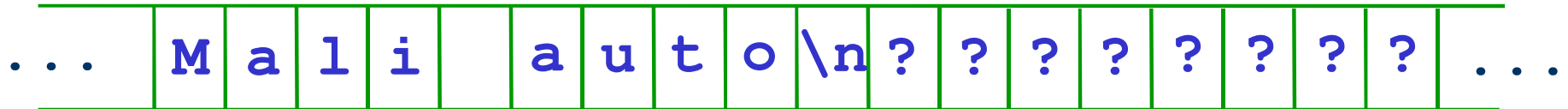
---

Ulazni niz podataka:

**Mali auto**↵

Što će se kod učitavanja desiti sa znakom '\n'

**slovo** ↘



U polje je upisan i znak \n, ali brojač `i` nakon završetka petlje ima vrijednost 9, tako da se znak \n neće ispisati u drugoj petlji

**MALI AUTO**

**Primjer:** Učitati i ispisati niz znakova, pri čemu sva mala slova treba ispisati kao velika. Znakove učitavati dok se ne učitava EOF

```
for(i = 0; (slovo[i] = getchar()) != EOF; i++);  
n = i;  
for(i = 0; i < n; ++i)  
    putchar(toupper(slovo[i]));
```

```
Mali↵  
auto↵  
vozi↵  
<Ctrl+Z>↵
```

vrijednost n će biti 15

slovo ↘

...	M	a	l	i	\n	a	u	t	o	\n	v	o	z	i	\n	EOF	?	?	...
-----	---	---	---	---	----	---	---	---	---	----	---	---	---	---	----	-----	---	---	-----

```
MALI  
AUTO  
VOZI
```

Prva tri znaka ' \n ' učitana su u niz i uzrokuju prijelaz u novi red kod ispisa. EOF se neće ispisati jer je  $n == 15$

# gets, puts

<stdio.h>

- Učitava znakove u niz `s` dok ne učitava `'\n'` ili znak koji odgovara kraju datoteke (`^Z` ili `^D`). Tada učitani znak `'\n'` ili učitano oznaku kraja datoteke zamijeni sa znakom `'\0'`. Vraća `s` ukoliko je učitavanje uspješno. Ukoliko pri čitanju nastupi pogreška ili se kao **prvi** znak pročita oznaka kraja datoteke, vraća `NULL`

```
char *gets(char *s);
```

- Ispisuje niz znakova `s` i `'\n'`. Vraća nenegativni cijeli broj u slučaju kada ispis uspije, inače vraća `EOF`

```
int puts(const char *s);
```

**Primjer:** uzastopno učitavati i ispisivati retke teksta (redak sigurno nije dulji od 80 znakova) dok se ne upiše redak u kojem se pojavljuje tekst DOSTA

---

```
#include <stdio.h>
#include <string.h>
int main() {
    char redak[80+1];
    while (strstr(gets(redak), "DOSTA") == NULL)
        puts(redak);
    return 0;
}
```

Izvođenje programa:

```
Now is the time↵
Now is the time
for all good men↵
for all good men
sada je DOSTA pisanja↵
```

# scanf

<stdio.h>

- Prototip funkcije:

```
int scanf(const char *format, arg_1, arg_2, ..., arg_n);
```

- čita iz "standardne ulazne jedinice" (tipkovnica) u skladu sa zadanim formatom, te obavlja konverziju pročitanih znakova u podatke. Konverzija se obavlja na temelju konverzijskih specifikacija koje su dio formata, a rezultati se pohranjuju na lokacije na koje pokazuju argumenti `arg_1, ..., arg_n`
- argumenti `arg_1, ..., arg_n` su pokazivači i moraju odgovarati konverzijskim specifikacijama po broju, redoslijedu i tipu
- funkcija vraća broj uspješno pročitanih podataka ili EOF ukoliko se pri čitanju dogodi pogreška

**Opisane su tek najvažnije mogućnosti funkcije `scanf`.**

**Detaljniji opis funkcije `scanf` može se pronaći u gotovo svakom C priručniku.**

# Konverzijske specifikacije za `scanf`

---

`% [širina] [modifikator] tip`

## Tip

<b>d</b>	cijeli broj s predznakom (dekadski)
<b>u</b>	cijeli broj bez predznaka (dekadski)
<b>o</b>	cijeli broj bez predznaka (oktalni)
<b>x</b>	cijeli broj bez predznaka (heksadekadski)
<b>e, f, g</b>	broj s pomičnim zarezom, sa ili bez eksponenta (u <code>scanf</code> nema razlike među tipovima <b>e, f, g</b> )
<b>c</b>	jedan znak
<b>s</b>	niz znakova

# Konverzijske specifikacije za `scanf`

Modifikator (zadaje se opcionalno)

- h** uz cjelobrojni tip (d, o, x, u): konverzija u `short int`
- l** uz cjelobrojni tip (d, o, x, u): konverzija u `long int`  
uz realni tip (e, f, g): konverzija u `double`

```
short i, j; int k; long m;
```

```
float x; double y, z, w;
```

```
scanf("%hd %hx %d %ld %f %le %lg %lf",  
      &i, &j, &k, &m, &x, &y, &z, &w);  
printf("%d %d %d %d %f %f %f %f",  
       i, j, k, m, x, y, z, w);
```

```
1 2a 3 4↵  
5.1 6.1 7.1 8.1↵
```

```
1 42 3 4 5.100000 6.100000 7.100000 8.100000
```

# Konverzijske specifikacije za `scanf`

Širina (zadaje se opcionalno)

- najveći broj znakova koje je dopušteno pročitati uz dotičnu konverzijsku specifikaciju

Primjer:

```
Ana912 3 4567↵
```

```
char ime[20];
```

```
int i, j;
```

```
scanf("%s %d %d", ime, &i, &j);
```

```
ime → "Ana912"      i → 3      j → 4567
```

```
scanf("%8s %3d %2d", ime, &i, &j);
```

```
ime → "Ana912"      i → 3      j → 45
```

```
scanf("%4s %3d %2d", ime, &i, &j);
```

```
ime → "Ana9"        i → 12     j → 3
```

# Format za `scanf`

---

- u svom jednostavnijem obliku, format se sastoji od konverzijskih specifikacija i bjelina (*white-space characters: blank, tab, newline*).
- jedna ili više bjelina u formatu znači: preskoči 0 ili više bjelina na ulazu, dok ne dođeš do znaka koji nije bjelina
- konverzijska specifikacija znači:
  - u slučaju tipova `d`, `u`, `o`, `x`, `e`, `f`, `g`, `s`, prvo preskoči sve bjeline na ulazu, a zatim pročitaj grupu znakova koji se mogu pretvoriti u odgovarajući podatak. Ukoliko se niti prvi pročitani znak ne može pretvoriti u traženi podatak, funkcija prestaje s čitanjem ulaza i vraća broj podataka koje je do tada uspješno pročitala i obavila konverziju
  - u slučaju tipa `c` pročitaj znak s ulaza (to može biti i bjelina)

# Primjer:

```
int i, j;
float x, y;
scanf("%d %d %f %f", &i, &j, &x, &y);
printf("%d %d %f %f", i, j, x, y);
```

```
38 -15 5.51↵
+151↵
```

Preskače bjeline s ulaza (zbog %d). Čita znakove 38 (čita dok god ulaz odgovara specifikaciji %d). 38 pretvara u `int` kojeg upisuje na adresu `&i`. Preskače bjeline (zbog bjeline u formatu). -15 pretvara u `int`, upisuje na `&j`. Preskače bjeline. Čita znakove 5.51 i pretvara ih u `float`, upisuje na `&x`. Preskače bjeline. Čita znakove +151, pretvara u `float` i upisuje na `&y`. Ostatak ulaza ostaje nepročitan (npr. sljedeći `getchar()` bi pročitao znak `\n`).

```
38 -15 5.510000 151.000000
```

# Primjer:

```
int i, j;
float x, y, z, w;
scanf("%d%d %f %f %f %d", &i, &j, &x, &y, &z, &w);
printf("%d %d %f %f %f %f", i, j, x, y, z, w);
```

38↵

-15.012 24+25 7.8↵

Preskače bjeline s ulaza. Čita znakove 38, pretvara u `int` kojeg upisuje na adresu `&i`. Preskače bjeline. -15 pretvara u `int`, upisuje na `&j` (točka nije pročitana jer ne može biti dio cijelog broja). Čita točku i znakove 012, pretvara u `float`, upisuje na `&x`. Preskače bjeline. Čita znakove 24 (+ nije pročitano), pretvara u `float` i upisuje na `&y`. Čita znakove +25, pretvara u `float` i upisuje na `&z`. Preskače bjeline. Čita znak 7, pretvara u `int`, zapisuje na adresu `&w`. Ostatak ulaza ostaje nepročitano (npr. sljedeći `getchar()` bi pročitao znak '.').

```
38 -15 0.012000 24.000000 25.000000 0.000000
```

# Primjer:

12 a8↵

```
int i, j, rez;  
rez = scanf("%d%d", &i, &j);  
printf("%d %d %d", i, j, rez);
```

Preskače bjeline s ulaza. Čita znakove 12, pretvara u `int` kojeg upisuje na adresu `&i`. Preskače bjeline. Znak `a` se ne može pretvoriti u `int`, prekida učitavanje, vraća 1. Ostatak ulaza ostaje nepročitan (npr. sljedeći `getchar()` bi pročitao znak 'a').

12 -858993460 1

"smeće", jer vrijednost varijable `j` nije učitana

# Napomena uz konverzijsku specifikaciju %c

%c bjelinu prihvaća jednako kao bilo koji drugi znak

A B C↵

```
char x, y, z;
```

```
scanf("%c%c%c", &x, &y, &z);      ⇒ x=A, y= , z=B
```

```
scanf("%c %c %c", &x, &y, &z);   ⇒ x=A, y=B, z=C
```

```
scanf(" %c %c %c", &x, &y, &z); ⇒ x=A, y=B, z=C
```

A B C↵

```
scanf("%c %c %c", &x, &y, &z);   ⇒ x= , y=A, z=B
```

```
scanf(" %c %c %c", &x, &y, &z); ⇒ x=A, y=B, z=C
```

ABC↵

```
scanf(" %c %c %c", &x, &y, &z);   ⇒ x=A, y=B, z=C
```

```
scanf("%c%c%c", &x, &y, &z);     ⇒ x=A, y=B, z=C
```

# Napomena uz konverzijsku specifikaciju %s

---

%s prestaje učitavati znakove kad naiđe na prvu bjelinu

Ana Marija ↵

```
char ime1[80+1], ime2[80+1];
```

```
scanf("%s", ime1);           ⇒ ime1="Ana"
```

```
scanf("%2s", ime1);        ⇒ ime1="An"
```

```
scanf("%10s", ime1);       ⇒ ime1="Ana"
```

```
scanf("%s%s", ime1, ime2);  ⇒ ime1="Ana" ime2="Marija"
```

# Učitavanje niza znakova koji sadrži bjeline

---

Kako učitati niz koji sadrži bjeline?

Ana Marija ↵

```
char ime[80+1];
```

*/\* ne preskače bjeline na početku, učitava sve znakove dok ne dođe do znaka \n \*/*

```
scanf ("%[^\\n]", ime);      ⇒ ime=" Ana Marija "
```

*/\* ne preskače bjeline na početku, učitava sve znakove dok ne dođe do znaka \n ili učitava 10 znakova\*/*

```
scanf ("%10[^\\n]", ime);      ⇒ ime=" Ana Mari"
```

# printf

<stdio.h>

- Prototip funkcije:

```
int printf(const char *format, arg_1, arg_2, ..., arg_n);
```

- Funkcija obavlja ispis na "standardnu izlaznu jedinicu" (zaslou) u skladu sa zadanim formatom.
- Vrijednosti argumenata `arg_1, ..., arg_n`, formatiraju se u skladu s konverzijskim specifikacijama koje su dio formata.
- Ostali znakovi koji se nalaze u `format` ispisuju se nepromijenjeni
- Funkcija vraća broj uspješno ispisanih bajtova ili EOF ukoliko se pri pisanju dogodi pogreška

**Opisane su tek najvažnije mogućnosti funkcije `printf`.**

**Detaljniji opis funkcije `printf` može se pronaći u gotovo svakom C priručniku.**

# Izgled konverzijskih specifikacija kod funkcije `printf`

---

`% [znak] [širina] [.preciznost] tip`

**[znak]**

ništa	desno pozicioniranje
praznina	tiska - predznak, a umjesto + predznaka je praznina
-	lijevo pozicioniranje
+	rezultat uvijek počinje s + ili -
0	ispisuje vodeće nule
#	<b>konverzija na alternativan način:</b> ne utječe na <code>c s d i u</code> ispisuje vodeću 0 za <code>o</code> ispisuje vodeće <code>0x</code> ili <code>0X</code> za <code>x</code> ili <code>X</code> ispisuje dec. točku i kad nema decimala za <code>e E F</code> ispisuje prateće 0 za <code>g G</code>

# Konverzijske specifikacije za `printf`

## Tip

---

<b>d</b>	cijeli broj s predznakom (dekadski)
<b>u</b>	cijeli broj bez predznaka (dekadski)
<b>o</b>	cijeli broj bez predznaka (oktalni)
<b>x, X</b>	cijeli broj bez predznaka (heksadekadski), a-f ili A-F
<b>c</b>	jedan znak
<b>s</b>	niz znakova
<b>p</b>	pokazivač
	<b>brojevi s pomičnim zarezom</b> (float, double)
<b>f</b>	bez eksponenta
<b>e, E</b>	s eksponentom ( <i>scientific notation</i> ), ispisuje e ili E
<b>g, G</b>	po potrebi sa ili bez eksponenta, ispisuje e ili E

# Konverzijske specifikacije za `printf`

---

## Širina (zadaje se opcionalno)

- određuje najmanju širinu polja
- za zadanu širinu `n`, ispisat će se najmanje `n` znakova
  - ako je `n` veći od potrebne širine podatka, podatak se pozicionira desno unutar polja ispisa širine `n`, s vodećim prazninama
  - ako je podatak širi od `n`, ili ako širina nije zadana, podatak će se ispisati u širini koja je potrebna za ispis tog podatka

## Preciznost (zadaje se opcionalno)

- za tipove `e`, `E`, `f`: određuje broj znamenki iza dec. točke
- za ostale tipove, `d`, `o`, `u`, `c`, `X`, `g`, `G`, `s`: ima drugačije značenje, ovdje se preciznost za te tipove neće razmatrati
- ako se preciznost ne zada, koristi se preciznost po definiciji (npr. za `e`, `E`, `f`, to je šest znamenki iza decimalne točke)

# Primjer:

```
float x = 321.f, y = 1.234e-7f, z = 7.65432e9f;
printf("|%f|%f|%f|\n", x, y, z);
printf("|%10f|%10f|%10f|\n", x, y, z);
printf("|%10.4f|%10.4f|%10.4f|\n", x, y, z);
printf("|%.4f|%.4f|%.4f|\n", x, y, z);
printf("|%3.1f|%3.1f|%3.1f|\n", x, y, z);
printf("|%13.11f|%13.11f|%13.11f|\n", x, y, z);
```

```
|321.000000|0.000000|7654320128.000000|
|321.000000| 0.000000|7654320128.000000|
| 321.0000| 0.0000|7654320128.0000|
|321.0000|0.0000|7654320128.0000|
|321.0|0.0|7654320128.0|
|321.00000000000|0.00000012340|7654320128.00000000000|
```

# Primjer:

```
float x = 321.f, y = 1.234e-7f, z = 7.65432e9f;
printf("|%e|%e|%e|\n", x, y, z);
printf("|%15e|%15e|%15e|\n", x, y, z);
printf("|%15.2E|%15.2E|%15.2E|\n", x, y, z);
```

```
|3.210000e+002|1.234000e-007|7.654320e+009|
| 3.210000e+002| 1.234000e-007| 7.654320e+009|
|      3.21E+002|      1.23E-007|      7.65E+009|
```

# Primjer:

```
float x = 321.f, y = 1.234e-7f, z = 7.65432e9f;  
printf("|%g|%g|%g|\n", x, y, z);  
printf("|%15G|%15G|%15G|\n", x, y, z);
```

```
|321|1.234e-007|7.65432e+009|  
|          321|          1.234E-007|          7.65432E+009|
```

# Primjer:

```
char *s1 = "Ana ";  
char *s2 = " Iva";  
char *s3 = "Ana-Marija";  
printf("|%s|%s|%s|\n", s1, s2, s3);  
printf("|%12s|%12s|%12s|\n", s1, s2, s3);  
printf("|%6s|%6s|%6s|\n", s1, s2, s3);
```

```
|Ana | Iva|Ana-Marija|  
|           Ana |           Iva|   Ana-Marija|  
|  Ana  |   Iva|Ana-Marija|
```

*Zadatak:* Pročitati vrijednosti za  $mr \leq 50$  i  $ns \leq 10$ .  
 Pročitati vrijednosti članova dvodimenzionalnog realnog polja od  $mr$  redaka i  $ns$  stupaca. Ispisati pročitano polje, sume redaka i sume stupaca te ukupnu sumu u obliku:

```

                                Polje A:

      !      1      2      3      4  ...   Sumr
=====
  1  !  xxx.x  xxx.x  xxx.x  xxx.x  ...   xxx.x
-----
... !
      !
-----
Sums !  xxx.x  xxx.x  xxx.x  xxx.x  ...   xxx.x
  
```

# Rješenje u pseudokodu

---

```
čitaj mr i ns dok ne budu ispravni
učitaj realno polje od mr redaka i ns
stupaca
ispisi naslov
ponavlaj za sve retke od 1 do mr
  | izračunaj sumu retka
  | ispiši redak i sumu
  | ispiši crt
izračunaj sume stupaca i ukupnu sumu
ispisi sume stupaca i ukupnu sumu
kraj
```

# Rješenje u C-u

📁 SumeRedakaIStupaca

```
#include <stdio.h>
#define MAXR 50
#define MAXS 10
int main() {
    int mr, ns, i, j;
    float a[MAXR][MAXS], sums[MAXS], sumr, ukupno;
    /* citanje mr i ns dok ne budu ispravni */
    do {
        printf("Upisite vrijednosti za broj redaka i stupaca: ");
        scanf ("%d %d",&mr, &ns);
    } while (mr <= 0 || mr > MAXR ||
            ns <= 0 || ns > MAXS);
```

# Rješenje u C-u, nastavak

---

```
/* citanje polja od mr redaka i ns stupaca */
printf("Upisite polje po retcima\n");
for (i=0; i < mr; i++) {
    for (j=0; j < ns; j++) {
        scanf("%f", &a[i][j]);
    }
}
/* ispis naslova */
printf("                Polje A:\n\n");
printf("                !");
for (j=1; j <= ns; j++) printf("%6d", j);
printf("    Sumr\n");
for (j=1; j <= 6*(ns+1)+5; j++) printf("%c", '=');
printf("\n");
```

# Rješenje u C-u, nastavak

---

```
/* ponavlja za sve retke od 1 do mr */
for (i=0; i<mr; i++) {
    sumr = 0;
    for (j=0; j<ns; j++) {
        sumr += a[i][j];
    }
    printf("%3d !", i+1);
    for (j=0; j<ns; j++) printf(" %5.1f", a[i][j]);
    printf(" %5.1f\n", sumr);
    for (j=1; j <= 6 * (ns+1) + 5; j++) printf("-");
    printf("\n");
}
```

# Rješenje u C-u, nastavak

---

```
/* izracunaj sume stupaca i ukupnu sumu */
ukupno = 0;
for (j=0; j<ns; j++) {
    sums[j] = 0;
    for (i=0; i<mr; i++) {
        sums[j] += a[i][j];
    }
    ukupno += sums[j];
}
/* ispisi sume stupaca i ukupnu sumu */
printf("Sums!");
for (j=0; j<ns; j++) printf(" %5.1f", sums[j]);
printf(" %5.1f\n", ukupno);

return 0;
}
```

# Izvođenje programa:

Upisite vrijednosti za broj redaka i stupaca: 2 4↵

Upisite polje po retcima

1 2 3 4↵

0.1 0.2 -0.1 -0.2↵

Polje A:

	!	1	2	3	4	Sumr
=====						
1	!	1.0	2.0	3.0	4.0	10.0
-----						
2	!	0.1	0.2	-0.1	-0.2	0.0
-----						
Sums	!	1.1	2.2	2.9	3.8	10.0