Uvod u programiranje

- predavanja -

siječanj 2026.

20. Standardna biblioteka

Vodič „14. Prije dvadesetog predavanja”

- Očekuje se da (na razini na kojoj je obrađeno u vodiču):
  - razumijete značenje pojmova
    - standardna biblioteka,
    - datoteka zaglavlja
  - razumijete ulogu funkcija iz math.h (opisanih u tutorialu)
- Pitanja?

Standardna biblioteka

Uvod

Standardna biblioteka u programskom jeziku C

- Standardna biblioteka je skup funkcija, makro definicija i definicija tipova koja je raspoloživa u svim implementacijama programskog jezika C, neovisno o platformi (arhitekturi i operacijskom sustavu)
  - oslobađa programera potrebe za implementacijom često korištenih postupaka (npr. čitanje/pisanje na standardni ulaz i izlaz, matematičke funkcije, baratanje nizovima znakova, ...)
  - osigurava se prenosivost (portabilnost) programskog koda: korištenje standardnih funkcija omogućuje da isti (ili minimalno prilagođeni) izvorni programski kod pruža istu funkcionalnost na različitim platformama (naravno, nakon prevođenja na svakoj od dotičnih platformi)
- Standardna biblioteka (*ISO C Library*) dio je specifikacije programskog jezika C
  - ISO/IEC 9899:2011 - standard za programski jezik C

Aplikacijsko programsko sučelje

- Aplikacijsko programsko sučelje (*Application Programming Interface*) je specifikacija programskih komponenti koje se koriste za izgradnju programa
  - opis aplikacijskog programskog sučelja jezika C temelji se na sadržaju datoteka zaglavlja standardne biblioteke (*C Standard Library Header Files*)
    - pojedine implementacije programskog jezika C mogu uključivati i nestandardna zaglavlja (potreban je oprez po pitanju prenosivosti)
  - konkretna implementacija nije važna
    - npr. nije važno kako je funkcija `pow` iz `<math.h>` realizirana na operacijskom sustavu Windows, a kako na operacijskom sustavu Linux
  - važno je *sučelje (interface)*
    - npr. važan je naziv funkcije, broj i tipovi parametara, tip funkcije (dakle deklaracija ili prototip, a ne definicija funkcije), opis funkcije, eventualna ograničenja u korištenju i slično

Datoteke zaglavlja standardne biblioteke

- U nastavku će biti razmatrani dijelovi aplikacijskog programskog sučelja iz nekoliko od ukupno 29 datoteka zaglavlja standardne biblioteke
  - svaka datoteka zaglavlja standardne biblioteke obuhvaća deklaracije funkcija, makro definicije, itd. koje čine logički povezanu cjelinu

<assert.h>	<limits.h>	<stdbool.h>	<threads.h>
<complex.h>	<locale.h>	<stddef.h>	**<time.h>**
<ctype.h>	**<math.h>**	<stdint.h>	<uchar.h>
<errno.h>	<setjmp.h>	**<stdio.h>**	<wchar.h>
<fenv.h>	<signal.h>	**<stdlib.h>**	<wctype.h>
<float.h>	<stdalign.h>	<stdnoreturn.h>	
<inttypes.h>	<stdarg.h>	**<string.h>**	
<iso646.h>	<stdatomic.h>	<tgmath.h>	

Standardna biblioteka

<time.h>

Date and time handling functions

time <time.h>

```
time_t time(time_t *timer);
time_t          tip podatka za pohranu informacije o trenutku u vremenu
```

- podatak tipa `time_t` predstavlja trenutak u vremenu
  - točan oblik podatka ovisi o implementaciji
  - standardom nije definirano radi li se npr. o cijelom ili realnom broju
- u gcc implementaciji može se koristiti kao tip `signed int`
  - `time(NULL)` vraća trenutačno vrijeme izraženo u broju sekundi proteklih nakon 00:00 sati, 1. siječnja 1970-UTC (*Unix epoch*)
    - npr. 22. prosinca 2015, 11:45:30 - GMT = 1450784730 sekundi
  - ako je argument `timer` različit od NULL, tada se trenutačno vrijeme (broj sekundi proteklih od početka Unix epohe) upisuje u objekt na kojeg pokazuje `timer`

Primjer

- Programski zadatak
  - napisati funkciju `vrijeme` koja vraća koliko je dana, sati, minuta i sekundi proteklo od početka *Unix epohe*. U glavnom programu ispisati rezultat izvršavanja funkcije
  - primjer izvršavanja programa

```
Od 00:00:00-1.1.1970(UTC) proteklo je:
   Dana: 17889
   Sati:    12
 Minuta:    42
Sekundi:    53
```

Rješenje (1. dio)

```
#include <stdio.h>
#include <time.h>

#define SEK_U_MIN 60
#define SEK_U_SAT (SEK_U_MIN * 60)
#define SEK_U_DAN (SEK_U_SAT * 24)

void vrijeme(int *dana, int *sati, int *minuta, int *sekundi) {
   time_t ukupno_sekundi;
   ukupno_sekundi = time(NULL); ili time(&ukupno_sekundi);
   *dana = ukupno_sekundi / SEK_U_DAN;
   *sati = ukupno_sekundi % SEK_U_DAN / SEK_U_SAT;
   *minuta = ukupno_sekundi % SEK_U_SAT / SEK_U_MIN;
   *sekundi = ukupno_sekundi % SEK_U_MIN;

   return;
}
```

Rješenje (2. dio)

```
int main(void) {
   int dana, sati, minuta, sekundi;
   vrijeme(&dana, &sati, &minuta, &sekundi);

   printf("Od 00:00:00-1.1.1970(UTC) proteklo je:\n");
   printf("   Dana: %5d\n", dana);
   printf("   Sati: %5d\n", sati);
   printf(" Minuta: %5d\n", minuta);
   printf("Sekundi: %5d\n", sekundi);

   return 0;
}
```

Standardna biblioteka

<stdlib.h>

General utilities

srand, rand <stdlib.h>

```
void srand(unsigned int seed);
int rand(void);
RAND_MAX                          makro
```

- funkcija `srand` postavlja početnu vrijednost (*seed*) za generator pseudoslučajnih brojeva
  - korištenje različitih početnih vrijednosti generatora garantira generiranje različitih nizova pseudoslučajnih brojeva
- funkcija `rand` pri svakom pozivu generira i vraća sljedeći broj iz niza pseudoslučajnih brojeva. Generiraju se brojevi iz intervala `[0, RAND_MAX]`
- konstanta `RAND_MAX` ovisi o implementaciji
  - za gcc, Windows: `RAND_MAX=32767`
  - za gcc, Linux: `RAND_MAX=2147483647`

Primjer

- Programski zadatak
  - na zaslon ispisati niz od 10 brojeva iz intervala [0, RAND_MAX]
    - za ispis koristiti format "%6d"
  - zahtijeva se da se pri svakom izvršavanju programa dobije drugačiji niz brojeva

  - Primjeri izvršavanja programa

```
  4065 22631 26275  2804 23973 24723 30972 21910 29178 23340
  4150  7178 31991  5859 30008 28514 13592 11336 32495 27330
  4241 13221  7900 24271 23904  7390  2436 27677  3299 19024
```

Rješenje (neispravno)

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

int main(void) {
   for (int i = 0; i < 10; ++i)
      printf("%6d", rand());
   ...
```

Isti rezultat bi se dobio da je na početku obavljeno

```
srand(1U);
```

```
    41 18467  6334 26500 19169 15724 11478 29358 26962 24464
    41 18467  6334 26500 19169 15724 11478 29358 26962 24464
    41 18467  6334 26500 19169 15724 11478 29358 26962 24464
```

- zašto se kod svakog izvršavanja programa dobije isti niz brojeva?
  - svako izvršavanje započelo je s istim početnim stanjem generatora
    - ako se generator pozivom funkcije `srand` ne inicijalizira na neku drugu početnu vrijednost, početna vrijednost mu odgovara onoj koja bi se dobila pozivom funkcije `srand` s argumentom 1U

Rješenje (ispravno)

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

int main(void) {
   srand((unsigned int)time(NULL));
   for (int i = 0; i < 10; ++i)
      printf("%6d", rand());
   ...
```

```
 14521 32629 14424 15967 21096 15200  2800 20423 11738 30795
 14609 27924  5236 10317  4677 22918 21300  6923 31567 28637
 14897 23511  4419 30736 27624  8022   284 11467 10027 11895
```

- za većinu namjena je ovo prihvatljivo rješenje, ali treba uzeti u obzir da i u ovom rješenju, ako se program pokrene tri puta unutar iste sekunde, dobit će se tri jednaka niza brojeva
  - inače, treba primijeniti bolju rezoluciju vremena ili neki drugi izvor početne vrijednosti za generator pseudoslučajnih brojeva

Primjer

- Programski zadatak
  - napisati funkciju `baciKocku` koja pri svakom pozivu vraća sljedeći pseudoslučajni broj iz intervala [1, 6]. Funkcija `baciKocku` treba na prikladan način inicijalizirati generator (postaviti početnu vrijednost generatora) tako da se izbjegne dobivanje istog rezultata pri višekratnom izvršavanju programa. U alternativnom rješenju neka za inicijalizaciju generatora bude zadužen *pozivajući program*
  - U glavnom programu kocku baciti zadani broj puta i ispisati frekvencije ishoda bacanja kocke
  - Primjeri izvršavanja programa

```
Broj bacanja > 10↵
1     1
2     1
3     4
4     1
5     2
6     1
```

```
Broj bacanja > 10↵
1     2
2     3
3     0
4     1
5     1
6     3
```

```
Broj bacanja > 1000000↵
1 166627
2 166725
3 166802
4 166843
5 166445
6 166558
```

Rješenje (1. dio)

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

int baciKocku(void) {
   static _Bool inicijaliziran = 0;
   if (!inicijaliziran) {
      srand((unsigned int)time(NULL));
      inicijaliziran = 1;
   }

   return rand() % 6 + 1;
}
```

Alternativno: ako pozivajući program mora inicijalizirati generator

```
int baciKocku(void) {
   return rand() % 6 + 1;
}
```

- Zašto bi bilo pogrešno pri svakom pozivu funkcije `baciKocku` izvršiti naredbu

```
srand((unsigned int)time(NULL));
```

Rješenje (2. dio)

```
int main(void) {
   int n, brojac[6] = {0};

   printf ("Broj bacanja > ");
   scanf ("%d", &n);

   for (int i = 0; i < n; ++i) {
      ++brojac[baciKocku() - 1];
   }

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

   return 0;
}
```

Alternativno: ako pozivajući program mora inicijalizirati generator

```
srand((unsigned int)time(NULL));
```

- uočiti: ako se program izvrši više puta unutar iste sekunde, pri svakom izvršavanju će se dobiti isti rezultat

Prilagodba intervala generiranih brojeva

- kako dobiti **realne** brojeve iz intervala [a, b]?
  - realni brojevi iz intervala `[0, 1]`

    ```
    (double)rand() / RAND_MAX
    ```

  - skaliranjem: realni brojevi iz intervala `[0, b - a]`

    ```
    (double)rand() / RAND_MAX * (b - a)
    ```

- translacijom: realni brojevi iz intervala `[a, b]`

  ```
  (double)rand() / RAND_MAX * (b - a) + a
  ```

- naravno, moguće je dobiti najviše RAND_MAX + 1 različitih realnih brojeva iz intervala [a, b]

Prilagodba intervala generiranih brojeva

- kako dobiti **cijele** brojeve iz intervala `[a, b]`?

```
rand() % (b - a + 1) + a
```

ILI

  - realni brojevi iz intervala `[0, 1)`

```
(double)rand() / (RAND_MAX + 1U)
```

  - skaliranjem: realni brojevi iz intervala `[0, b - a + 1)`

```
(double)rand() / (RAND_MAX + 1U) * (b - a + 1)
```

  - translacijom: realni brojevi iz intervala `[a, b + 1)`

```
(double)rand() / (RAND_MAX + 1U) * (b - a + 1) + a
```

- odsijecanjem decimala: cijeli brojevi iz intervala `[a, b]`

```
(int)((double)rand() / (RAND_MAX + 1U) * (b - a + 1) + a)
```

Primjer (primjena na bacanje kocke)

- cijeli brojevi iz intervala `[a, b]`

```
(int)((double)rand() / (RAND_MAX + 1U) * (b - a + 1) + a)
a = 1, b = 6
```

```
int baciKocku(void) {
   static _Bool inicijaliziran = 0;
   if (!inicijaliziran) {
      srand((unsigned int)time(NULL));
      inicijaliziran = 1;
   }
   return (double)rand() / (RAND_MAX + 1U) * 6 + 1;
}
```

- `cast` operator `(int)` u ovom slučaju nije potreban jer će se obaviti implicitna konverzija rezultata funkcije (`double→int`)

exit <stdlib.h>

```
void exit(int status);
EXIT_FAILURE  makro: može se koristiti kao status neplaniranog završetka programa
EXIT_SUCCESS  makro: može se koristiti kao status planiranog završetka programa
```

- trenutačno prekida daljnje izvršavanje programa i pozivajućem programu (operacijskom sustavu) vraća cjelobrojnu vrijednost `status`
- poziv funkcije `exit(status)` u funkciji `main` ima jednaki efekt kao izvršavanje naredbe `return status;`
- vrijednosti statusa **nisu** standardizirane
  - često korištena konvencija: nula predstavlja uspješan, a druge vrijednosti neuspješan završetak programa, pri čemu različite vrijednosti mogu imati različite interpretacije pogreške

Primjer

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

int dijeli(int a, int b) {
   if (b == 0) {
      exit(10);
   }
   return a / b;
}

int main(void) {
   int a, b;
   scanf("%d %d", &a, &b);
   printf("%d", dijeli(a, b));

   return 0;
   ili return EXIT_SUCCESS;
   ili exit(0);
   ili exit(EXIT_SUCCESS);
}
```

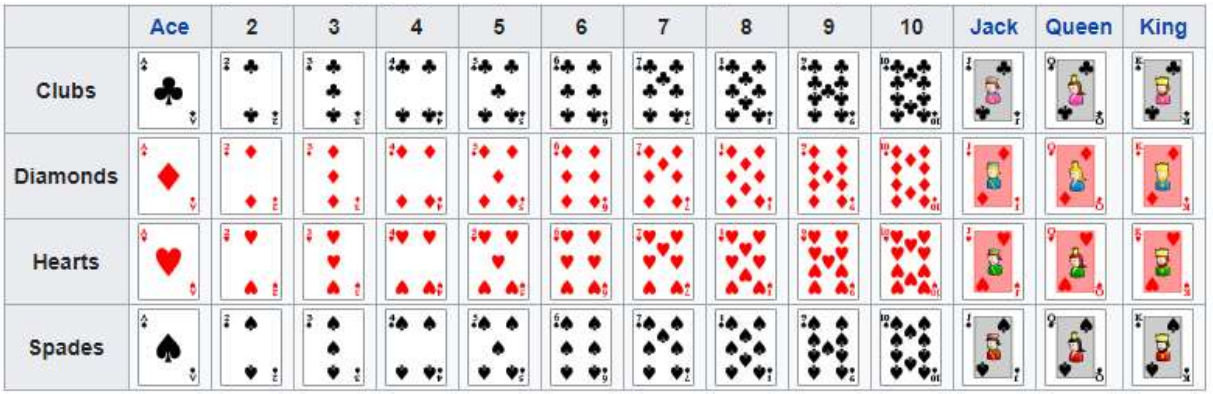
izvršavanje u operacijskom sustavu Linux (C shell)

```
$ prog
7 2
3
$ echo $status
0
$ prog
7 0
$ echo $status
10
```

izvršavanje u operacijskom sustavu Windows (Command prompt)

```
c:> prog.exe
7 2
3
c:> echo %errorlevel%
0
c:> prog.exe
7 0
c:> echo %errorlevel%
10
```

Zadatak



Napisati program koji

- generira standardni špil od 52 karte (4 boje * 13 karata) i ispiše generirane karte,
- promiješa špil i ispiše promiješane karte.

Naziv karte se sastoji od

- oznake boje (suit) iz skupa znakova CDHS (Clubs – tref, Diamonds - karo, Hearts - herc, Spades - pik) te
- vrijednosti karte iz skupa znakova A23456789TJQK (slova označavaju Ace, Ten, Jack, Queen, King).

Napisati, i u glavnom programu upotrijebiti, funkciju za miješanje karata ***shuffleDeck*** kojoj se predaju informacije o polju - špilu kojeg treba promiješati. U funkciji je za potrebno implementirati ***Fisher–Yates algoritam***:

- početi od zadnjeg elementa i zamijeniti ga s nasumično odabranim elementom iz cijelog preostalog polja (uključujući samoga sebe).
- smanjiti raspon na prethodne elemente i ponavljati dok se ne dođe do prvog elementa.

Mogući ispis:

```
Generirani spil:
A.C 2.C 3.C 4.C 5.C 6.C 7.C 8.C 9.C T.C J.C Q.C K.C
A.D 2.D 3.D 4.D 5.D 6.D 7.D 8.D 9.D T.D J.D Q.D K.D
A.H 2.H 3.H 4.H 5.H 6.H 7.H 8.H 9.H T.H J.H Q.H K.H
A.S 2.S 3.S 4.S 5.S 6.S 7.S 8.S 9.S T.S J.S Q.S K.S
Promijesani spil:
K.S 5.C 3.S T.C 6.S 2.H 9.C 9.H 2.C J.H 2.S J.C 3.C
K.C Q.S K.H J.D A.D 4.C 5.H 6.H 6.C 7.D 3.D 4.D K.D
5.D 8.D 7.H 8.C T.S 3.H T.H A.C Q.D 2.D Q.H 8.S 6.D
7.S Q.C A.H 4.S 5.S A.S T.D 4.H 9.D 8.H 7.C 9.S J.S
```

Standardna biblioteka

<string.h>

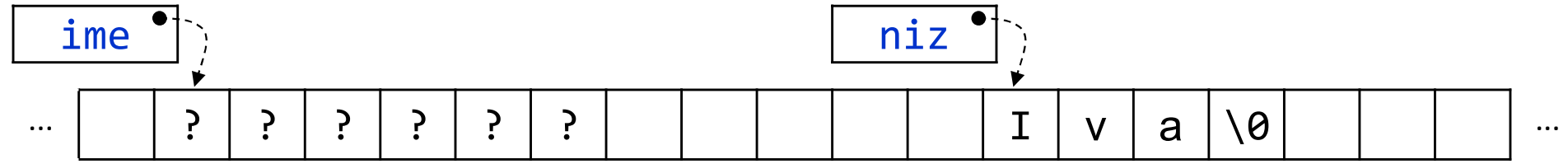
String handling functions

strcpy <string.h>

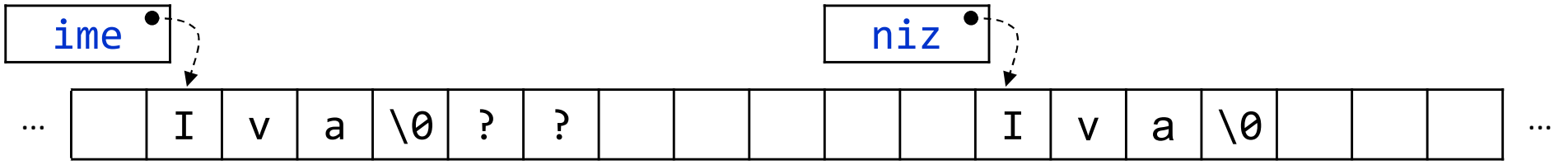
```
char *strcpy(char *s1, const char *s2);
```

- niz znakova s2 (točnije: niz čiji se početak nalazi na mjestu u memoriji na kojeg pokazuje s2) kopira u niz znakova s1 (točnije: kopira u memoriju od mjesta na kojeg pokazuje s1 nadalje)
- funkcija vraća s1

```
char ime[5 + 1];
char niz[] = "Iva";
```



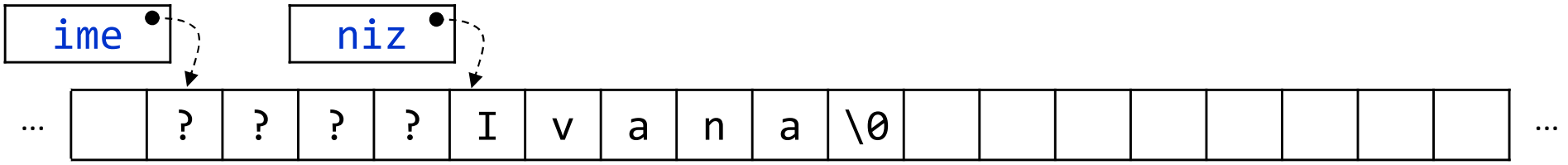

```
strcpy(ime, niz);
```



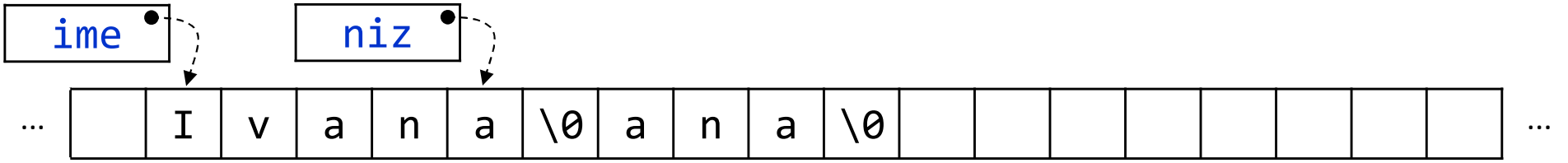
strcpy <string.h>

- važno je da je za s1 rezervirano dovoljno mjesta

```
char ime[3 + 1];
char niz[] = "Ivana";
```




```
strcpy(ime, niz);
```




```
printf("%s %s", ime, niz);
```

```
Ivana a
```

- ovo vrijedi za sve str… funkcije koje kopiraju nizove

Ključna riječ const

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

- što *općenito* znači ključna riječ `const` navedena u deklaraciji ili definiciji varijable ili parametra?
  - za pokazivač `p`: sadržaj objekta na kojeg pokazuje `p` ne može se mijenjati pomoću tog pokazivača, npr. nije dopušteno `*(p + 2) = 10;`
  - za varijablu `x`: nakon inicijalizacije, sadržaj varijable `x` se više ne smije mijenjati, npr. nije dopušteno `primBr[0] = 1;`

```
const int primBr[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
```

- konkretno, deklaracija funkcije `strcpy` garantira da se niz znakova na kojeg pokazuje `s2` sigurno neće promijeniti za vrijeme izvršavanja funkcije
  - česta su dva pogrešna tumačenje: niz znakova na koji pokazuje `s2` mora biti konstantni znakovni niz; `s2` se u funkciji ne smije mijenjati

Primjer

```
char *ime1 = "Ivana";
char ime2[] = "Marko";
strcpy(ime1, ime2);          nije dopušteno
strcpy("Darko", ime1);       nije dopušteno
strcpy(ime2, ime1);          dopušteno
strcpy(ime2, "Darko");       dopušteno
```

- Što će se ispisati izvršavanjem sljedećeg odsječka programa?

```
char niz1[] = "Tko sanja elektricne ovce?";
char *niz2 = "Hanibal pred vratima";
strcpy(niz1 + 4, "On nije android" + 5);          niz1: Tko je android
strcpy(niz1 + 7, niz2 + 8);
printf("%s%c", niz1, '?');
```

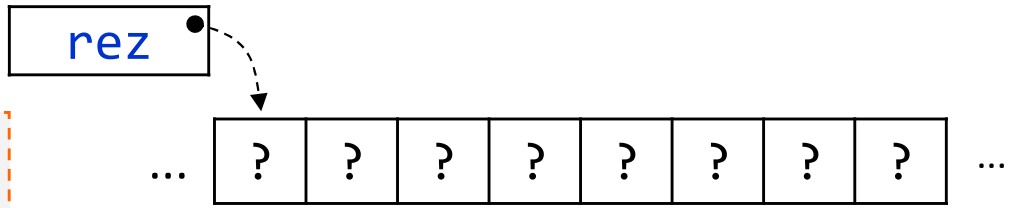
```
Tko je pred vratima?
```

strncpy <string.h>

```
char *strncpy(char *s1, const char *s2, size_t n);
```

- ne više od n znakova iz niza znakova s2 kopira u niz s1. Ako u s2 ima manje od n znakova, s1 se dopunjuje znakovima '\0' do duljine n
- funkcija vraća s1

```
char rez[7 + 1];
```




```
strncpy(rez, "Ana", 2);
```

... A n ? ? ? ? ? ? ...

```
strncpy(rez, "Ana", 3);
```

... A n a ? ? ? ? ? ...

```
strncpy(rez, "Ana", 4);
```

... A n a \0 ? ? ? ? ...

```
strncpy(rez, "Ana", 6);
```

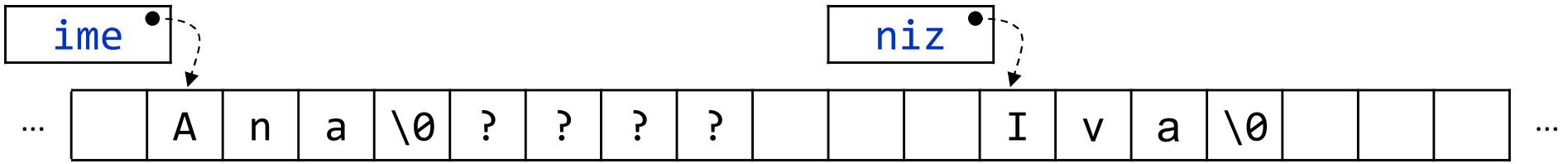
... A n a \0 \0 \0 ? ? ...

strcat <string.h>

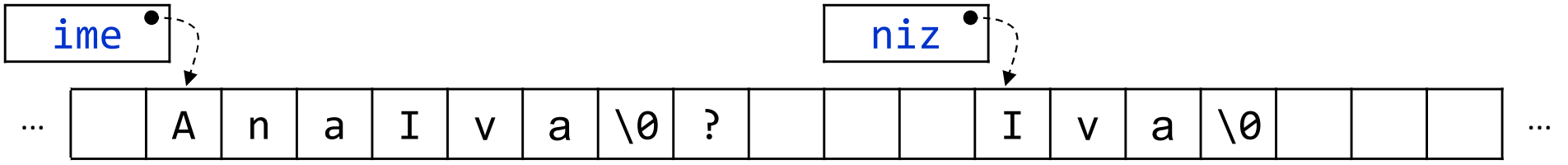
```
char *strcat(char *s1, const char *s2);
```

- niz znakova s2 kopira na kraj niza znakova s1 (nadovezivanje ili konkatenacija nizova znakova)
- funkcija vraća s1

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




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



strncat <string.h>

```
char *strncat(char *s1, const char *s2, size_t n);
```

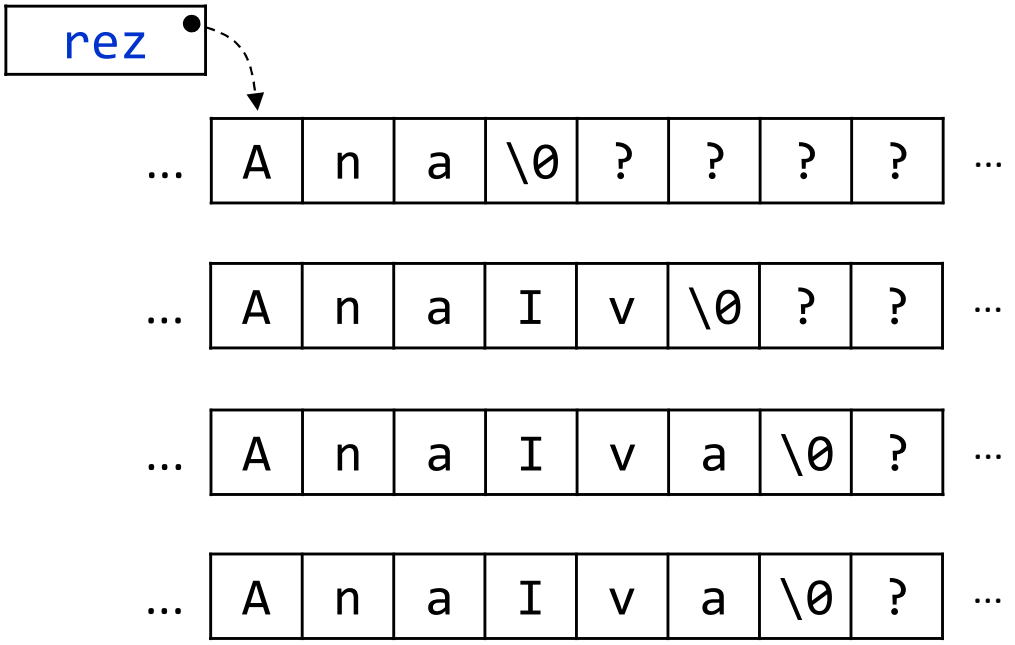
- ne više od n znakova niza znakova s2 kopira na kraj niza znakova s1 i novi niz terminira znakom '\0'
- drugim riječima, niz znakova ili dio niza znakova s2 do najviše duljine n nadovezuje na niz s1
- funkcija vraća s1

```
char rez[7 + 1];
strcpy(rez, "Ana");
```

```
strncat(rez, "Iva", 2);
```

```
strncat(rez, "Iva", 3);
```

```
strncat(rez, "Iva", 5);
```



strlen <string.h>

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

- duljina niza: vraća broj znakova u nizu s. Ne broji znak '\0'

```
char niz[] = "Pero";
char *p = "Ana";
char polje[3] = {'I', 'v', 'a'};

strlen(niz);                    4
strlen(p + 1);                  2
strlen(polje);      neizvjesno jer niz nije terminiran
                    dobije se pogrešan rezultat ili runtime pogreška
```

strcmp <string.h>

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

- leksikografska usporedba nizova znakova `s1` i `s2`.
- funkcija vraća
  - `0` ako su nizovi leksikografski jednaki
  - negativni cijeli broj ako je `s1 < s2`
  - pozitivni cijeli broj ako je `s1 > s2`

```
strcmp("abcd", "abrd");      cijeli broj manji od 0
strcmp("abc", "abcd");       cijeli broj manji od 0

strcmp("abcd", "abc");       cijeli broj veći od 0
strcmp("abcd", "abcc");      cijeli broj veći od 0
strcmp("aBc", "abc");        cijeli broj manji od 0

strcmp("aBc", "aBc");        0
```

strncmp <string.h>

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

- leksikografska usporedba nizova znakova `s1` i `s2`, najviše do duljine `n` znakova (usporedba podnizova do duljine `n`)
- funkcija vraća
  - `0` ako su (pod)nizovi leksikografski jednaki
  - negativni cijeli broj ako je (pod)niz `s1` < (pod)niz `s2`
  - pozitivni cijeli broj ako je (pod)niz `s1` > (pod)niz `s2`

```
strncmp("abcd", "abrd", 2);          0
strncmp("abc", "abcd", 5);           cijeli broj manji od 0

strncmp("abcd", "abc", 4);           cijeli broj veći od 0
strncmp("bcd", "abc", 1);            cijeli broj veći od 0
```

Prije sljedećeg predavanja

- Edgar:
  - Tutorial: **nema**
  - **20. vježbe uz predavanja**