

# Uvod u programiranje

- predavanja -

prosinac 2020.

---

## 21. Standardna biblioteka

- 2. dio -

# Standardna biblioteka

`<string.h>`

*String handling functions*

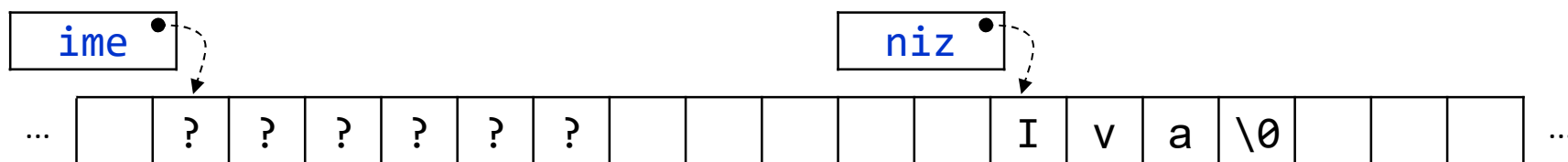
# strcpy

<string.h>

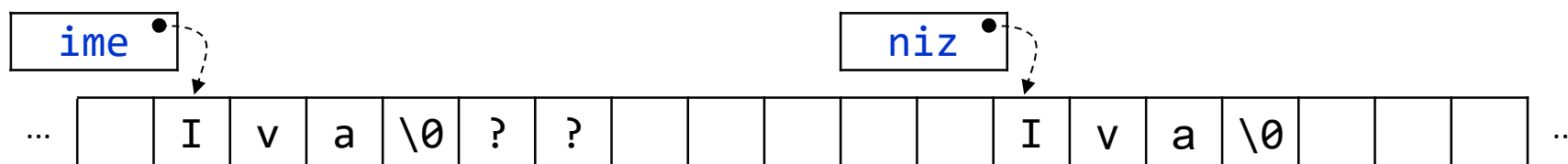
```
char *strcpy(char *s1, const char *s2);
```

- 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);
```

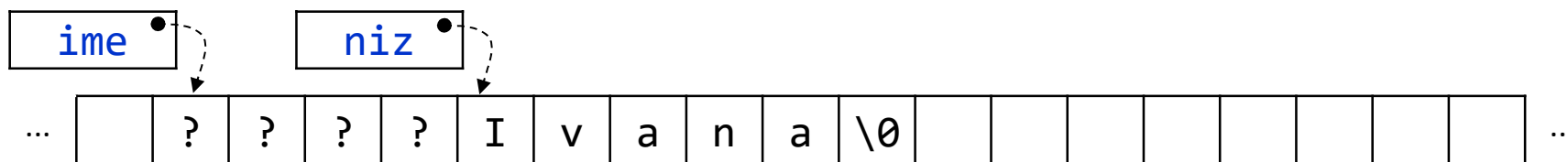


# strcpy

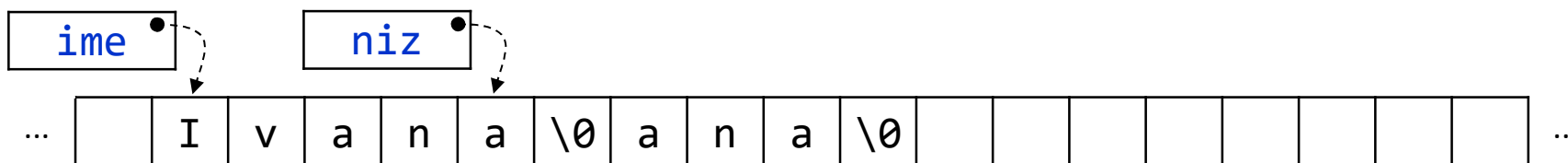
<string.h>

- 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);  
strcpy(niz1 + 7, niz2 + 8);  
printf("%s%c", niz1, '?');
```

niz1: Tko je android

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

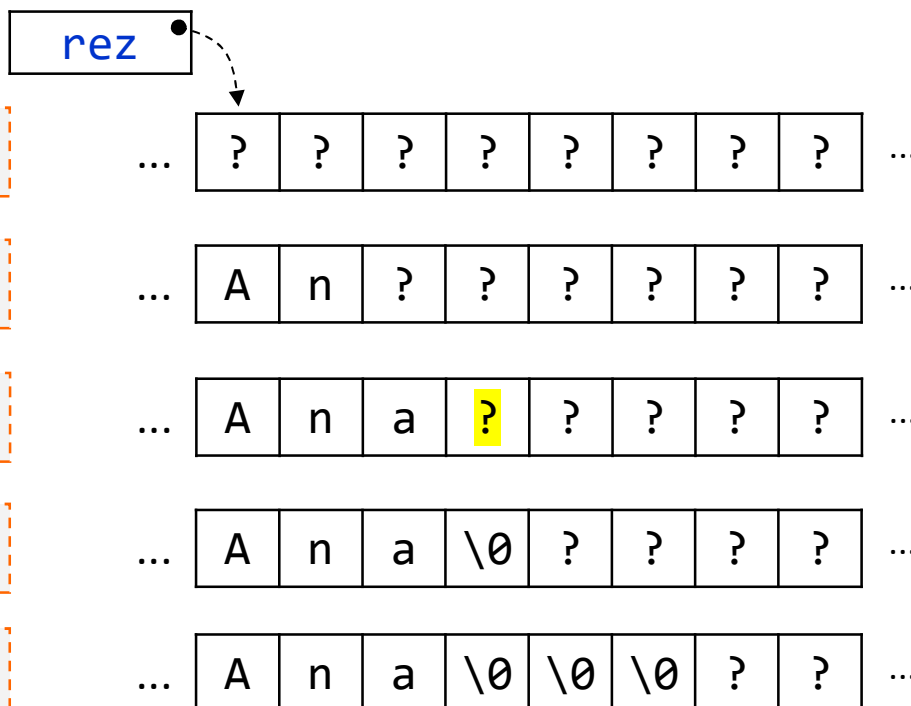
```
char rez[7 + 1];
```

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

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

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

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



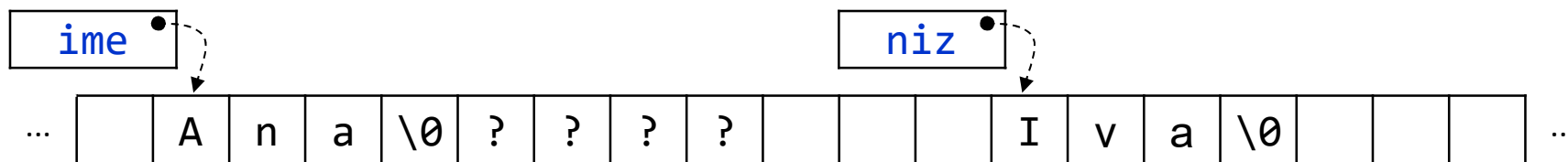
# strcat

<string.h>

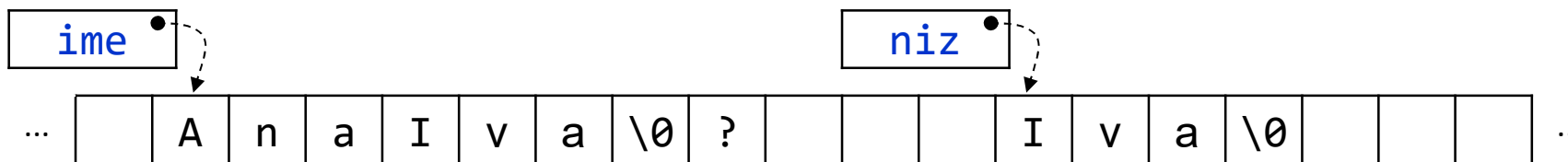
```
char *strcat(char *s1, const char *s2);
```

- niz znakova s2 kopira na kraj niza znakova s1 (nadovezivanje ili konkatencija 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);
```

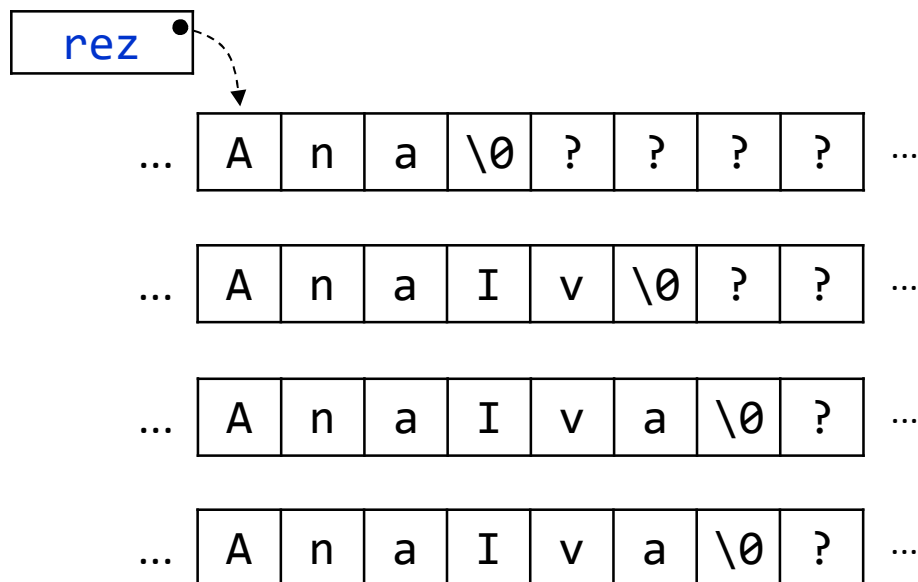
- ne više od n znakova niza znakova s2 kopira na kraj niza znakova s1 i novi niz terminira znakom '\0'
- drugim riječima, niz znakova ili dio niza znakova s2 do najviše duljine n nadovezuje na niz s1
- funkcija vraća s1

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

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

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

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



```
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  $s1 > s2$

<code>strcmp("abcd", "abrd");</code>	<code>cijeli broj manji od 0</code>
<code>strcmp("abc", "abcd");</code>	<code>cijeli broj manji od 0</code>
<code>strcmp("abcd", "abc");</code>	<code>cijeli broj veći od 0</code>
<code>strcmp("abcd", "abcc");</code>	<code>cijeli broj veći od 0</code>
<code>strcmp("aBc", "abc");</code>	<code>cijeli broj manji od 0</code>
<code>strcmp("aBc", "aBc");</code>	<code>0</code>

# Primjer

- Programski zadatak
  - napisati definiciju funkcije `mystrcmp` koja obavlja isto što i funkcija `strcmp` iz `<string.h>`
  - u glavnom programu učitati dva niza znakova koji zajedno s eventualno učitanim znakom novog retka sigurno neće biti dulji od 20 znakova. Pomoću funkcije `mystrcmp` usporediti nizove i ispisati odgovarajuću poruku
- Primjeri izvršavanja programa

```
1. niz > kisik↵  
2. niz > kisel↵  
1. niz je veci
```

```
1. niz > kisel↵  
2. niz > kisik↵  
2. niz je veci
```

```
1. niz > kisik↵  
2. niz > kisik↵  
nizovi su jednaki
```

## Rješenje (1. dio)

```
...  
int mystrcmp(const char *s1, const char *s2) {  
    while (*s1 == *s2 && *s1 != '\0' && *s2 != '\0') {  
        ++s1;  
        ++s2;  
    }  
    return *s1 - *s2;  
}
```

Može se ispustiti

ili

```
int mystrcmp(const char *s1, const char *s2) {  
    for (; *s1 == *s2 && *s1 != '\0'; ++s1, ++s2);  
    return *s1 - *s2;  
}
```

## Rješenje (2. dio)

```
#define MAXNIZ 20

int main(void) {
    char niz1[MAXNIZ + 1];
    char niz2[MAXNIZ + 1];
    printf("1. niz > ");
    fgets(niz1, MAXNIZ + 1, stdin);
    printf("2. niz > ");
    fgets(niz2, MAXNIZ + 1, stdin);

    int rez = mystrcmp(niz1, niz2);
    if (rez == 0) {
        printf("nizovi su jednaki");
    } else if (rez > 0) {
        printf("1. niz je veci");
    } else {
        printf("2. niz je veci");
    }
    return 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

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

- traženje prve pojave zadanog znaka c unutar niza s
- funkcija vraća
  - pokazivač na prvi znak unutar niza znakova koji ima vrijednost c
  - ako takav znak u nizu ne postoji, vraća NULL

```
char niz[] = "San Antonio";  
char *p = "New Orleans";  
printf("%c", *strchr(niz, 'a'));  
printf("%s", strchr(p, 'e'));  
printf("%s", strchr(p + 2, 'e'));  
strchr(p, 'R');
```

a  
ew Orleans  
eans  
vraća NULL



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

- traženje zadnje pojave zadanog znaka c unutar niza s
- funkcija vraća
  - pokazivač na zadnji znak unutar niza znakova koji ima vrijednost c
  - ako takav znak u nizu ne postoji, vraća NULL

```
char niz[] = "San Antonio";  
char *p = "New Orleans";  
printf("%c", *strrchr(niz, 'n'));  
printf("%s", strrchr(p, 'e'));  
  
strrchr(p + 1, 'N');  
strrchr(p, 'R');
```

n

eans

vraća NULL

vraća NULL

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

- traženje prve pojave podniza u s1 koji je jednak nizu s2 (znak terminatora niza s2 se ne uzima u obzir kod usporedbe)
- funkcija vraća
  - pokazivač na prvi znak pronađenog podniza
  - ako takav podniz u nizu s1 ne postoji, vraća NULL

```
char niz[] = "Nigdar ni tak bilo da ni nekak bilo";  
char *p = "tak";  
printf("%s", strstr(niz, p));           tak bilo da ni nekak bilo  
printf("%s", strstr(niz, "ni"));        ni tak bilo da ni nekak bilo  
printf("%s", strstr(strstr(niz, "ni") + 1, "ni")); ni nekak bilo  
strstr(niz, "Tak");                     vraća NULL
```

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

- traženje u nizu s1 prve pojave bilo kojeg od znakova navedenih u nizu s2
- funkcija vraća
  - pokazivač na prvi pronađeni znak
  - ako niti jedan od znakova iz s2 ne postoji u nizu s1, vraća NULL

```
char niz[] = "Gle, jedna duga u vodi se stvara,";  
char *p = "aeiou";  
printf("%s", strpbrk(niz, p));  
printf("%s", strpbrk(niz + 7, p + 1));  
strpbrk(niz, "AEIOU");
```

e, jedna duga u vodi se stvara,  
uga u vodi se stvara,  
vraća NULL

# Standardna biblioteka

`<ctype.h>`

*Character handling functions*

## Funkcije za klasifikaciju znakova

<ctype.h>

```
int isdigit(int c);   znamenka (0-9)
int isxdigit(int c);  heksadekadska znamenka (0-9, A-F, a-f)
int isalpha(int c);   slovo (A-Z ili a-z)
int isalnum(int c);   alfanumerik, 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 (space, 0x20) ili vodoravni tabulator (0x9)
int islower(int c);   malo slovo (a-z)
int isupper(int c);   veliko slovo (A-Z)
```

- ako zadani znak pripada određenom razredu znakova
  - funkcija vraća cijeli broj različit od nule ("istina")
- inače
  - funkcija vraća cijeli broj nula ("laž")

## Funkcije za konverziju znakova

<ctype.h>

```
int toupper(int c);  
int tolower(int c);
```

- toupper: ako je zadani znak malo slovo, vraća odgovarajuće veliko slovo, inače vraća zadani znak
- tolower: ako je zadani znak veliko slovo, vraća odgovarajuće malo slovo, inače vraća zadani znak

```
char niz[] = "GLE, jedna duga u vodi se stvara,";  
for (char *p = niz; *p != '\0'; ++p) {  
    if (isupper(*p)) {  
        printf("%c", tolower(*p));  
    } else {  
        printf("%c", toupper(*p));  
    }  
}
```

gLE, JEDNA DUGA U VODI SE STVARA,

# Standardna biblioteka

`<stdio.h>`

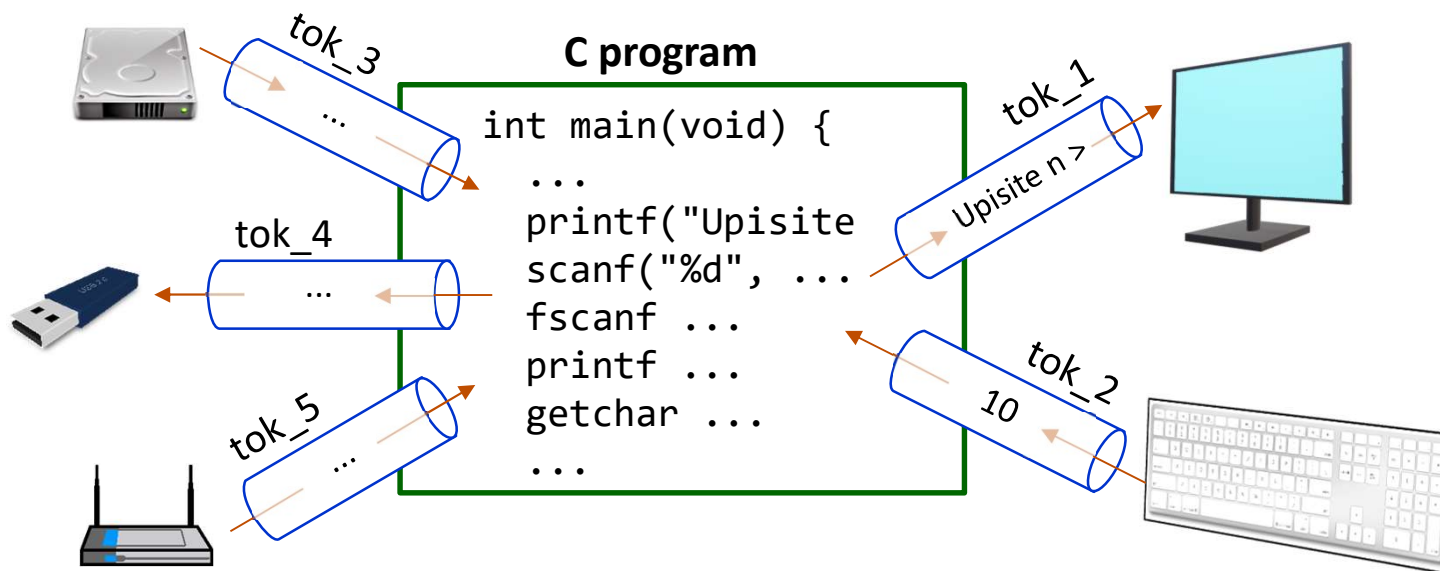
*Input and Output*

## *Tok (stream)*

- aplikacijsko programsko sučelje za obavljanje ulaznih i izlaznih operacija (čitanje i pisanje podataka) temelji se na pojmu *tok (stream)*
- *tok* je apstraktni pojam: kanal ili veza programa s jednim od izvora podataka (*input source*) ili odredištem podataka (*output sink*)
  - program se može povezati (kolokvijalno: *otvoriti ulazni tok iz, otvoriti izlazni tok prema*) npr. sa sljedećim izvorima ili odredištima podataka
    - tipkovnica
    - zaslon
    - datoteka
    - računalna mreža (*network socket*)
    - drugi programi
  - dakle, čitanje iz izvora podataka obavlja se čitanjem iz ulaznog toka koji povezuje program i dotični izvor podataka (npr. datoteku)
    - simetrično vrijedi i za pisanje u odredište podataka



## Tok (stream)



- velika prednost korištenja mehanizama *toka* je u tome što programera (uglavnom) oslobađa brige o tome koji se konkretni izvor ili odredište podataka, na kojem ulazno/izlaznom uređaju, koristi za čitanje ili pisanje podataka. Aplikacijsko programsko sučelje omogućuje
  - otvaranje i zatvaranje *toka* iz/prema nekom izvoru/odredištu podataka
  - korištenje funkcija za čitanje i pisanje (i ostalih operacija) na način koji (u principu) ne ovisi o vrsti izvora/odredišta podataka za koji je *tok* otvoren

## Tok u programskom jeziku C

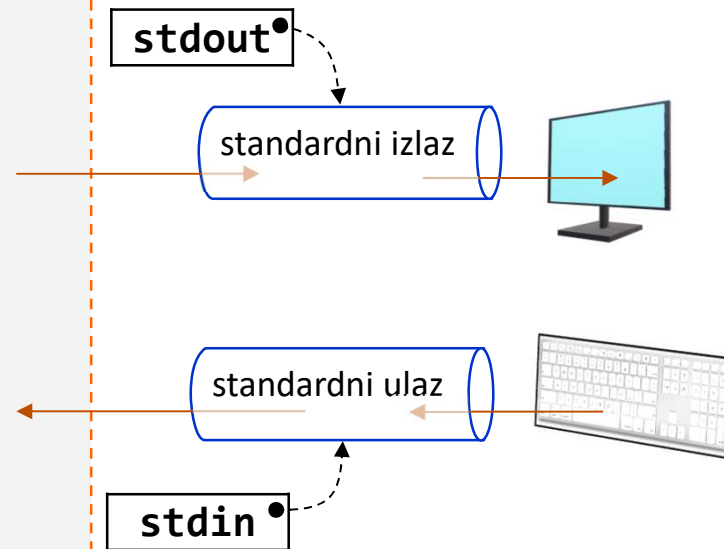
- U programskom jeziku C, *tok* je objekt tipa FILE
  - tip FILE je definiran u `<stdio.h>`. Konkretna implementacija tipa nije propisana standardom i za programera je potpuno nevažna
    - u programskom sučelju će se koristiti *pokazivači* na objekte tipa FILE
    - naziv tipa (pogrešno) asocira na datoteku (*file*) iz povijesnih razloga
  - za sada će se koristiti samo dva toka
    - *tok* iz tipkovnice (standardni ulaz) i *tok* prema zaslonu (standardni izlaz)
    - oba *toka* otvaraju se automatski, odmah po pokretanju programa
    - tokovi prema drugim izvorima/odredištima podataka (npr. datotekama) bit će objašnjeni kasnije
  - sljedeće globalne (eksterne) varijable tipa pokazivača na FILE mogu se koristiti u svim modulima s uključenom datotekom zaglavlja `<stdio.h>`
    - `stdin` (pokazivač na *tok* standardni ulaz)
    - `stdout` (pokazivač na *tok* standardni izlaz)

# Primjer

```
#include <stdio.h>
int main(void) {
    ...
    printf("Upis n > ");

    ...

    scanf("%d", &n);
    ...
}
```



- `stdin` i `stdout` su globalne varijable
- inicijaliziraju se automatski u trenutku pokretanja programa
- funkcije `printf` i `scanf` po definiciji koriste tokove podataka standardni izlaz i standardni ulaz na koje pokazuju `stdin` i `stdout`

## Varijante funkcija u <stdio.h>

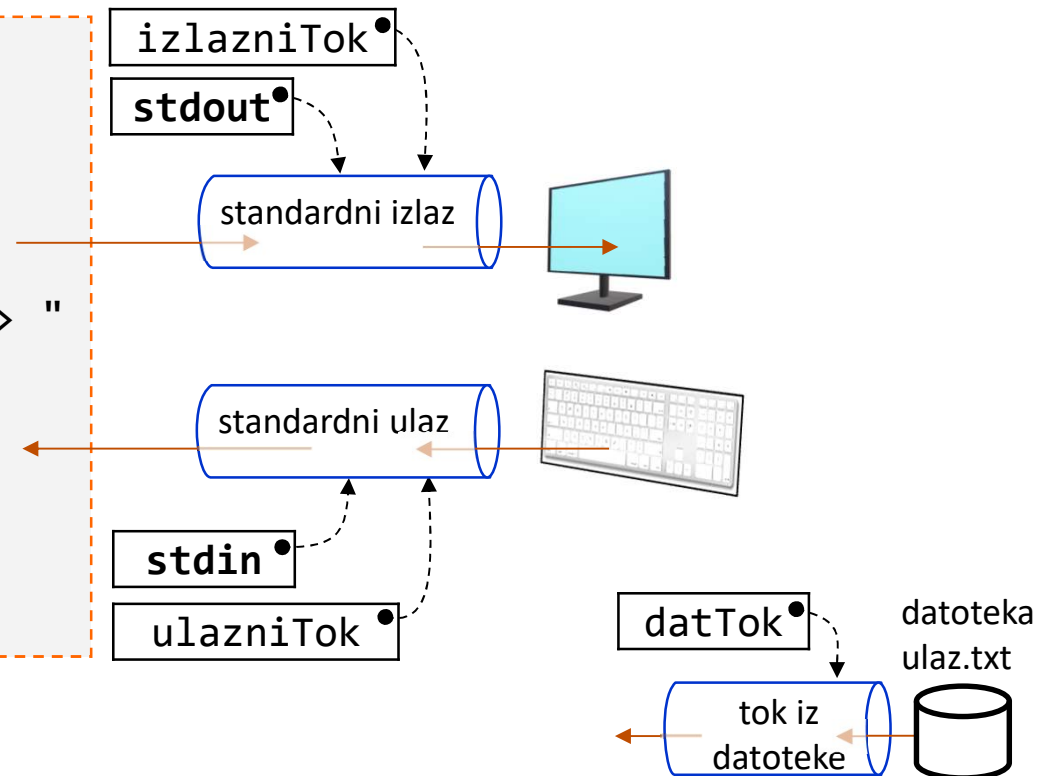
- printf zadani sadržaj uvijek piše na *standardni izlaz*
  - fprintf ima **dodatni parametar**: pokazivač na *tok* u kojeg treba ispisati zadani sadržaj (može biti pokazivač na *standardni izlaz* ili na neki drugi *tok*)
  - fprintf(**stdout**, "format", ...) ≡ printf("format", ...)
- scanf sadržaj uvijek čita iz *standardnog ulaza*
  - fscanf ima **dodatni parametar**: pokazivač na *tok* iz kojeg treba čitati sadržaj (može biti pokazivač na *standardni ulaz* ili na neki drugi *tok*)
  - fscanf(**stdin**, "format", ...) ≡ scanf("format", ...)
- i ostale funkcije iz <stdio.h> često se pojavljuju "u paru"
  - jedna funkcija koja po definiciji čita/piše na standardni ulaz/izlaz
  - jedna funkcija kojoj se pokazivač na *tok* zadaje kao argument/parametar

# Primjer

```
#include <stdio.h>

int main(void) {
    FILE *izlazniTok = NULL;
    izlazniTok = stdout;
    fprintf(izlazniTok, "Upis n > ")

    FILE *ulazniTok = NULL;
    ulazniTok = stdin;
    fscanf(ulazniTok, "%d", &n);
    ...
}
```



- zgodno je da se jednostavnom zamjenom vrijednosti pokazivača može promijeniti izvor ili odredište podataka
  - npr. u varijablu `ulazniTok` upisati pokazivač na *tok* iz datoteke "ulaz.txt"
  - kako otvoriti *tok* (osim standardnog ulaza i izlaza) i dobiti pokazivač na *tok* npr. iz datoteke `ulaz.txt` bit će objašnjeno u poglavlju o datotekama



```
int getchar(void);  
int getc(FILE *stream);
```

- čitanje jednog znaka iz standardnog ulaza (`getchar`) ili zadanog ulaznog *toka* (`getc`)
  - `getchar()`  $\equiv$  `getc(stdin)`
- funkcija prvo čeka da se u ulaznom *toku* pojavi jedan ili više znakova
- zatim čita prvi po redu znak u ulaznom *toku*. Znak koji je pročitala smatra se *konzumiranim* i uklanja se iz ulaznog *toka*
  - ako je čitanje znaka iz ulaznog *toka* uspjelo, funkcija vraća ASCII vrijednost pročitanoog znaka
  - ako čitanje znaka iz ulaznog *toka* nije uspjelo ili je pročitao znak koji označava kraj datoteke, funkcija vraća cjelobrojnu vrijednost EOF
    - EOF je makro definiran u `<stdio.h>`
    - znak koji označava kraj datoteke: `0x04` (Ctrl-D) na Unix , odnosno `0x1A` (Ctrl-Z) na Windows operacijskim sustavima

# Primjer

- Programski zadatak
  - s tipkovnice čitati znak po znak (koristiti funkciju `getchar`) i za svaki znak, odmah po učitavanju znaka, na zaslon ispisati njegovu ASCII vrijednost po konverzijskoj specifikaciji `%4d`. Učitavanje znakova ponavljati sve dok se ne učitava znak koji predstavlja oznaku kraja datoteke. Tada ispisati poruku "Kraj".
  - Primjeri izvršavanja programa

```
aAB↵                               Windows
97  65  66  10/↵
47  10<Ctrl-Z>↵
Kraj
```

```
aAB↵                               Unix/Linux
97  65  66  10/↵
47  10<Ctrl-D>Kraj
```

- Oznake `<Ctrl-Z>` i `<Ctrl-D>` znače da su na tipkovnici istovremeno pritisnute tipke `Ctrl` i `Z`, odnosno `Ctrl` i `D`
- na taj se način preko tipkovnice unose kontrolni znakovi koji predstavljaju oznaku kraja datoteke, `0x1A`, odnosno `0x04`



## Rješenje

```
int c;
do {
    c = getchar();
    if (c != EOF) {
        printf("%4d", c);
    }
} while (c != EOF);
printf("Kraj");
```

```
int c;
while ((c = getchar()) != EOF) {
    printf("%4d", c);
}
printf("Kraj");
```

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

- zašto funkcija `getchar` ne pročita znak **a** istog trenutka kada je utipkan, nego tek kada je utipkan cijeli redak **aAB↵**?
  - znakovi koji se unose preko tipkovnice prvo se pohranjuju u *međuspremnik tipkovnice (buffer)*. Tek kad se na tipkovnici pritisne tipka <Enter> sadržaj međuspremnika tipkovnice se dostavlja u *tok standardni ulaz*
  - taj mehanizam se naziva *line buffering*

# ungetc

<stdio.h>

```
int ungetc(int c, FILE *stream);
```

- vraćanje (*push back*) jednog upravo pročitano­g znaka iz ulaznog toka natrag u ulazni tok
  - ako je vraćanje znaka usp­jelo, funkcija vraća vrijednost parametra *c*
  - ako vraćanje znaka nije usp­jelo, funkcija vraća EOF

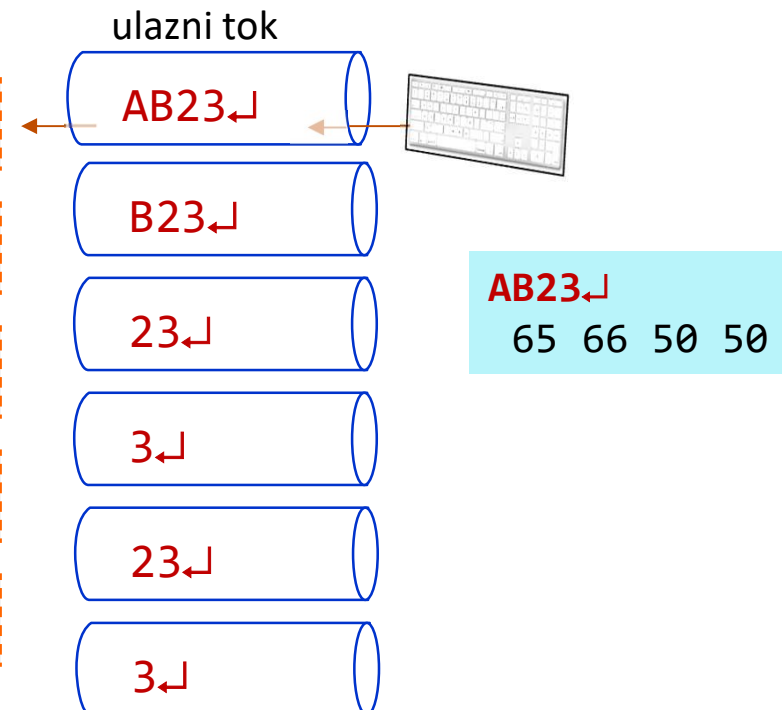
```
printf("%3d", c = getchar()); 65
```

```
printf("%3d", c = getchar()); 66
```

```
printf("%3d", c = getchar()); 50
```

```
ungetc(c, stdin);
```

```
printf("%3d", c = getchar()); 50
```



## putchar, putc

<stdio.h>

```
int putchar(int c);  
int putc(int c, FILE *stream);
```

- ispis jednog znaka na standardni izlaz (putchar) ili zadani izlazni *tok* (putc)
  - `putchar(c) ≡ putc(c, stdout)`
  - ako je ispis uspio, funkcija vraća vrijednost parametra `c`
  - ako ispis nije uspio, funkcija vraća EOF

```
for (int i = 'A'; i <= 'Z'; ++i)  
    putchar(i);
```

ABCDEFGHIJKLMNOPQRSTUVWXYZ

# fgets

<stdio.h>

```
char *fgets(char *s, int n, FILE *stream);
```

- u niz na kojeg pokazuje s čitaju se znakovi do kraja retka (ali ne više od n - 1 znak) iz zadanog ulaznog toka. Iza zadnjeg učitano znaka u niz se upisuje terminator niza '\0'.
  - ako je čitanje uspješno, funkcija vraća vrijednost parametra s
  - ako čitanje nije uspješno (zbog greške ili pokušaja čitanja znaka koji predstavlja kraj datoteke) funkcija vraća NULL
- zavisno od parametra n i duljine retka na ulazu, funkcija će pročitati oznaku novog retka
  - kako iz učitano niza ukloniti eventualno učitani znak '\n'

```
fgets(niz, n, stdin);  
char *nr = NULL;  
if ((nr = strchr(niz, '\n')) != NULL)  
    *nr = '\0';
```

# Primjer

## ■ Programski zadatak

- napisati funkciju `void citajRedak(char *niz, int n, FILE *tok);`
- u niz na kojeg pokazuje niz čitaju se svi znakovi prije kraja retka (ali ne više od  $n - 1$  znak) iz zadanog ulaznog toka te se niz terminira s `'\0'`. Znak za oznaku kraja retka, ako ga je bilo, treba ostati nepročitani u ulaznom toku. Zanimariti mogućnost pojave znaka koji označava kraj datoteke
- Napisati glavni program za testiranje funkcije
- Primjeri izvršavanja programa (za  $n = 10$ )

Duljina 10↵

Ucitan niz: |Duljina 1|↵  
Na ulazu je znak: |0|

Duljina 9↵

Ucitan niz: |Duljina 9|↵  
Na ulazu je znak: |↵  
|

D↵

Ucitan niz: |D|↵  
Na ulazu je znak: |↵  
|

↵

Ucitan niz: ||↵  
Na ulazu je znak: |↵  
|

## Rješenje (1. dio)

```
#include <stdio.h>

void citajRedak(char *niz, int n, FILE *tok) {
    int c;
    while (n > 1 && (c = getc(tok)) != '\n') {
        *niz++ = c;
        --n;
    }
    if (c == '\n') {
        ungetc(c, tok);
    }
    *niz = '\0';
}
```


## Rješenje (2. dio)

```
#include <stdio.h>
#define N 10

void citajRedak(char *niz, int n, FILE *tok);

int main(void) {
    char niz[N];
    citajRedak(niz, N, stdin);
    printf("Ucitan niz: |%s|\n", niz);
    printf("Na ulazu je znak: |%c|", getc(stdin));
    return 0;
}
```

```
int puts(const char *s);  
int fputs(const char *s, FILE *stream);
```

- ispis niza znakova na standardni izlaz (puts) ili zadani izlazni *tok* (fputs)
  - puts(s)  fputs(s, stdout)
    - puts (za razliku od fputs) nakon ispisa niza dodatno ispisuje znak za novi red
  - ako je ispis uspio, funkcija vraća nenegativni broj
  - ako ispis nije uspio, funkcija vraća EOF



# Primjer

- Programski zadatak
  - uzastopno učitavati i ispisivati retke teksta (učitati redak teksta iz standardnog ulaza, ispisati redak teksta na standardni izlaz). Učitavanje i ispis ponavljati dok god se ne upiše redak teksta u kojem se pojavljuje tekst KRAJ.
  - niti jedan redak teksta (uključujući oznaku novog retka) sigurno neće biti dulji od 20 znakova

```
The quick↵  
The quick↵  
brown fox jumps↵  
brown fox jumps↵  
nigdje KRAJA↵
```

# Rješenje

```
#include <stdio.h>
#include <string.h>
#define MAXNIZ 20
int main(void) {
    char niz[MAXNIZ + 1];
    while (strstr(fgets(niz, MAXNIZ + 1, stdin), "KRAJ") == NULL) {
        fputs(niz, stdout);
    }
    return 0;
}
```

- Za vježbu analizirati:
  - koji bi se rezultat dobio kada bi se makro MAXNIZ promijenio na 10?
  - zašto se program uz MAXNIZ=10 neće zaustaviti ako se upiše redak **nigdje KRAJA↵**, a zaustavit će se ako se umjesto tog retka upiše redak **ima KRAJA↵**