

Uvod u programiranje

- predavanja -

prosinac 2019.

12. Standardna biblioteka

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_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

```
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 $[0, \text{RAND_MAX}]$
- 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 $[0, 1]$
 $(\text{double})\text{rand}() / \text{RAND_MAX}$
 - skaliranjem: realni brojevi iz intervala $[0, b - a]$
 $(\text{double})\text{rand}() / \text{RAND_MAX} * (b - a)$
- translacijom: realni brojevi iz intervala $[a, b]$
 $(\text{double})\text{rand}() / \text{RAND_MAX} * (b - a) + a$

- naravno, moguće je dobiti najviše $\text{RAND_MAX} + 1$ 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 $[a, b]$

$(\text{int})((\text{double})\text{rand}() / (\text{RAND_MAX} + 1U) * (b - a + 1) + a)$

$a = 1, b = 6$

```
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 $(\text{double} \rightarrow \text{int})$

```
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

- trenutčno 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

```
#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);
}
```

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

izvršavanje u
operacijskom
sustavu Linux
(C shell)

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

izvršavanje u
operacijskom
sustavu Windows
(Command prompt)

Standardna biblioteka

`<math.h>`

Common mathematical functions


```
double fabs(double x);
```

- vraća apsolutnu vrijednost za x
 - **Problem:** kako izračunati apsolutnu vrijednost za tip `int` ili `long` `double` bez nepotrebnih (i potencijalno štetnih) konverzija argumenata i rezultata?
 - **Rješenje:** postoje varijante funkcije, "specijalizirane" za pojedine tipove podataka

```
int abs(int n);
```

```
<stdlib.h>
```

```
long labs(long n);
```

```
<stdlib.h>
```

```
long long llabs(long long n);
```

```
<stdlib.h>
```

```
float fabsf(float x);
```

```
<math.h>
```

```
double fabs(double x);
```

```
<math.h>
```

```
long double fabsl(long double x);
```

```
<math.h>
```

- u nastavku se neće navoditi sve varijante svake funkcije

Zašto ne makro s parametrima?

```
#include <stdio.h>

#define abs(x) ((x) < 0 ? -(x) : (x))

int main(void) {
    int a = -2;
    printf("%d", abs(++a));
    return 0;
}
```

0

```
gcc -E prog.c > prog.i
```

```
...
int a = -2;
printf("%d", ((++a) < 0 ? -(++a) : (++a)));
```

| | |
|--|-------------------|
| <code>double sin(double x);</code> | sinus x |
| <code>double cos(double x);</code> | kosinus x |
| <code>double tan(double x);</code> | tangens x |
| <code>double asin(double x);</code> | arkus sinus x |
| <code>double acos(double x);</code> | arkus kosinus x |
| <code>double atan(double x);</code> | arkus tangens x |
| <code>double atan2(double y, double x);</code> | arkus tangens y/x |

- parametri/rezultati koji se odnose na mjeru kuta izražavaju se u radijanima

```
double cosh(double x);
```

kosinus hiperbolni x

```
double sinh(double x);
```

sinus hiperbolni x

```
double tanh(double x);
```

tangens hiperbolni x

Eksponencijalne i logaritamske funkcije <math.h>

```
double exp(double x);
```

e^x

```
double log(double x);
```

$\ln x$

```
double log10(double x);
```

$\log_{10} x$

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

x^y

```
double sqrt(double x);
```

\sqrt{x}

Najbliži cijeli broj, ostatak dijeljenja

<math.h>

```
double ceil(double x);    najmanji cijeli broj koji je veći ili jednak x, [x]
double floor(double x);   najveći cijeli broj koji je manji ili jednak x, [x]

double fmod(double x, double y); ostatak dijeljenja x / y
```

■ Primjeri:

```
printf("%lf", fmod(3.82, 0.7));    0.320000

printf("%lf", ceil(-5.2));         -5.000000
printf("%lf", ceil(5.001));        6.000000

printf("%lf", floor(-5.2));        -6.000000
printf("%lf", floor(5.999));       5.0
```

Standardna biblioteka

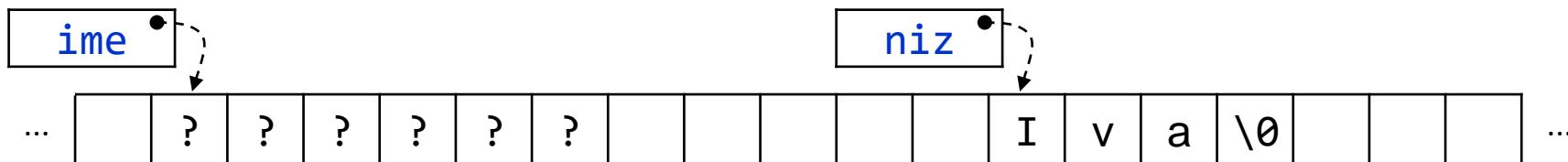
`<string.h>`

String handling functions

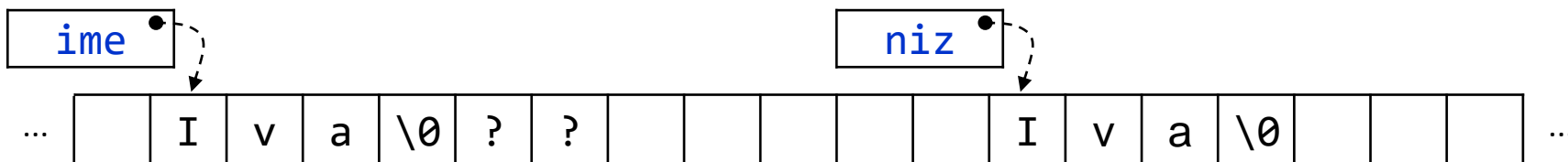
```
char *strcpy(char *s1, const char *s2);
```

- 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";
```

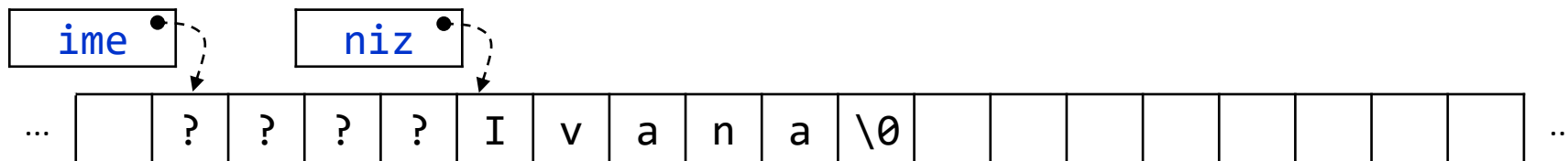


```
strcpy(ime, niz);
```

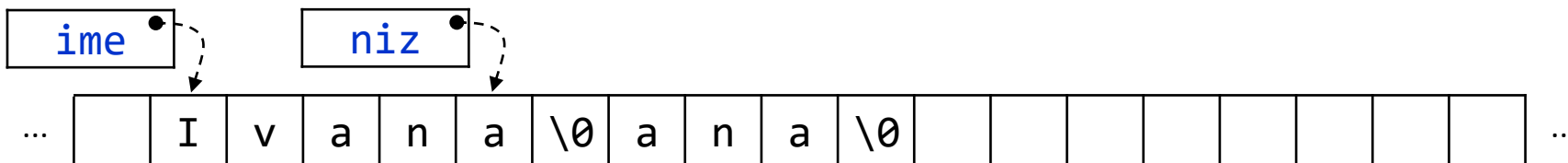


- važno je da je za s1 rezervirano dovoljno mjesta

```
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;`
- 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?

```
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);
```

rez

... [?] [?] [?] [?] [?] [?] [?] [?] ...

... [A] [n] [?] [?] [?] [?] [?] [?] ...

... [A] [n] [a] [?] [?] [?] [?] [?] ...

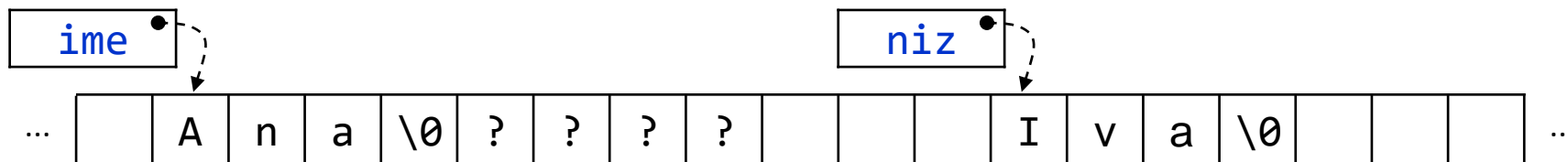
... [A] [n] [a] [\0] [?] [?] [?] [?] ...

... [A] [n] [a] [\0] [\0] [\0] [?] [?] ...

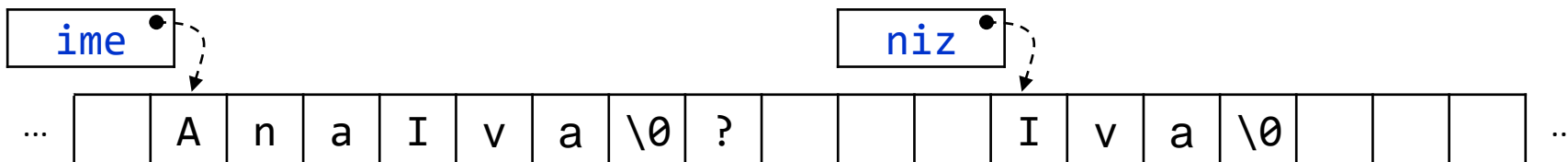
```
char *strcat(char *s1, const char *s2);
```

- 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);
```



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

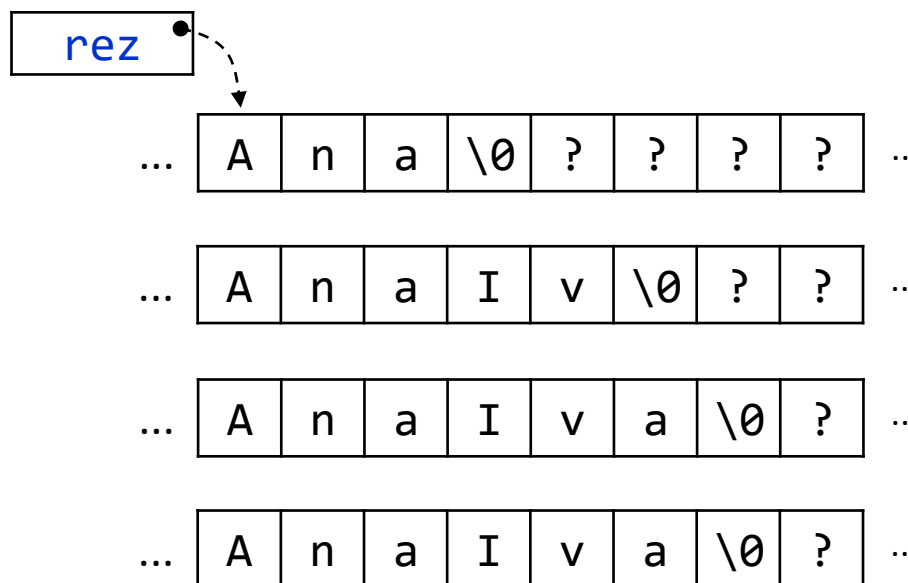
- ne više od n znakova niza znakova s2 kopira na kraj niza znakova s1 i novi niz terminira znakom '\0'
- drugim riječima, niz znakova ili dio niza znakova s2 do najviše duljine n nadovezuje na niz s1
- funkcija vraća s1

```
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

```
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

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 oznakom 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 > kiselo↵  
1. niz je veci
```

```
1. niz > kiselo↵  
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;
}
```

```
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 znak 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));
```

e, jedna duga u vodi se stvara,

```
printf("%s", strpbrk(niz + 7, p + 1));
```

uga u vodi se stvara,

```
strpbrk(niz, "AEIOU");
```

vraća NULL

Standardna biblioteka

`<ctype.h>`

Character handling functions

| | |
|-----------------------------------|--|
| <code>int isdigit(int c);</code> | znamenka (0-9) |
| <code>int isxdigit(int c);</code> | heksadekadska znamenka (0-9, A-F, a-f) |
| <code>int isalpha(int c);</code> | slovo (A-Z ili a-z) |
| <code>int isalnum(int c);</code> | alfanumerik, slovo (A-Z ili a-z) ili znamenka(0-9) |
| <code>int isprint(int c);</code> | znak koji se može ispisati (0x20-0x7E) |
| <code>int iscntrl(int c);</code> | kontrolni znak (0x00-0x1F ili 0x7F) |
| <code>int isspace(int c);</code> | praznina (space, 0x20) ili vodoravni tabulator (0x9) |
| <code>int islower(int c);</code> | malo slovo (a-z) |
| <code>int isupper(int c);</code> | 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")

```
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

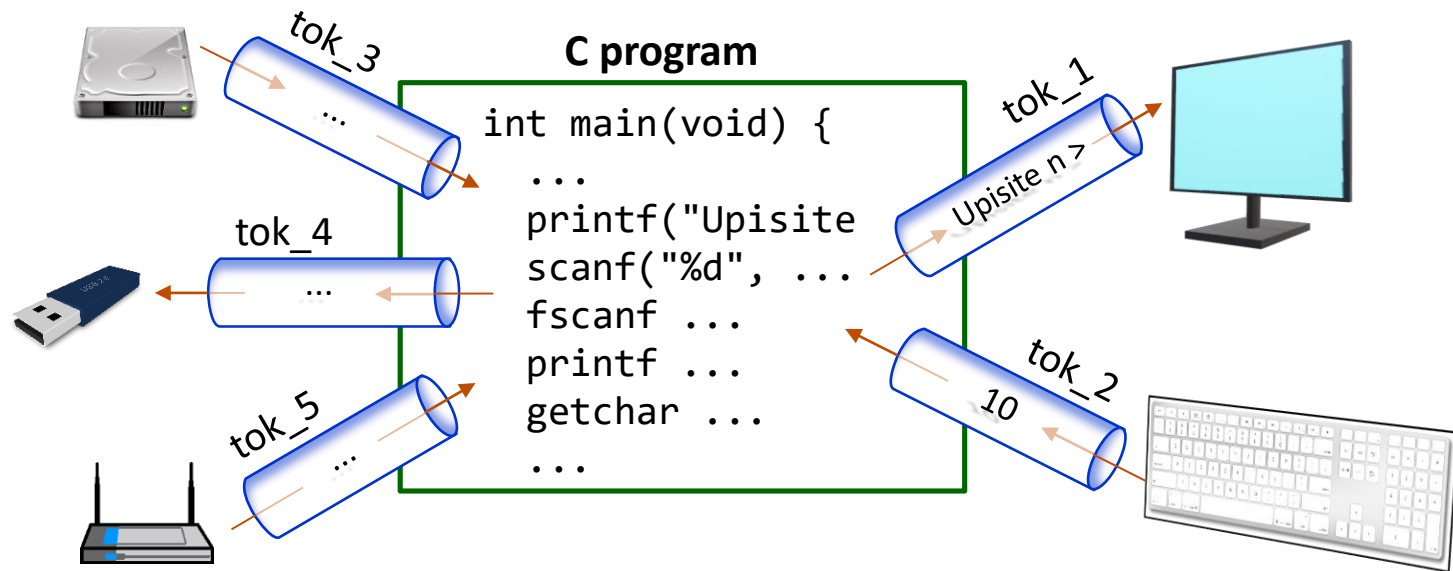
`<stdio.h>`

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)

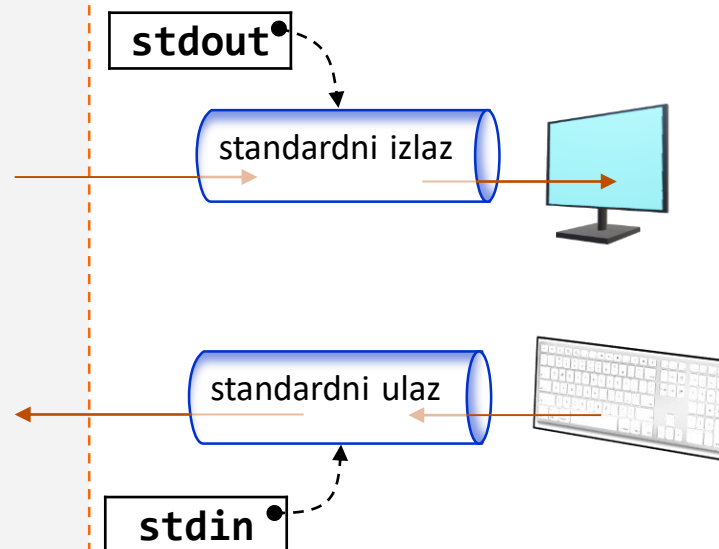
Primjer

```
#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>

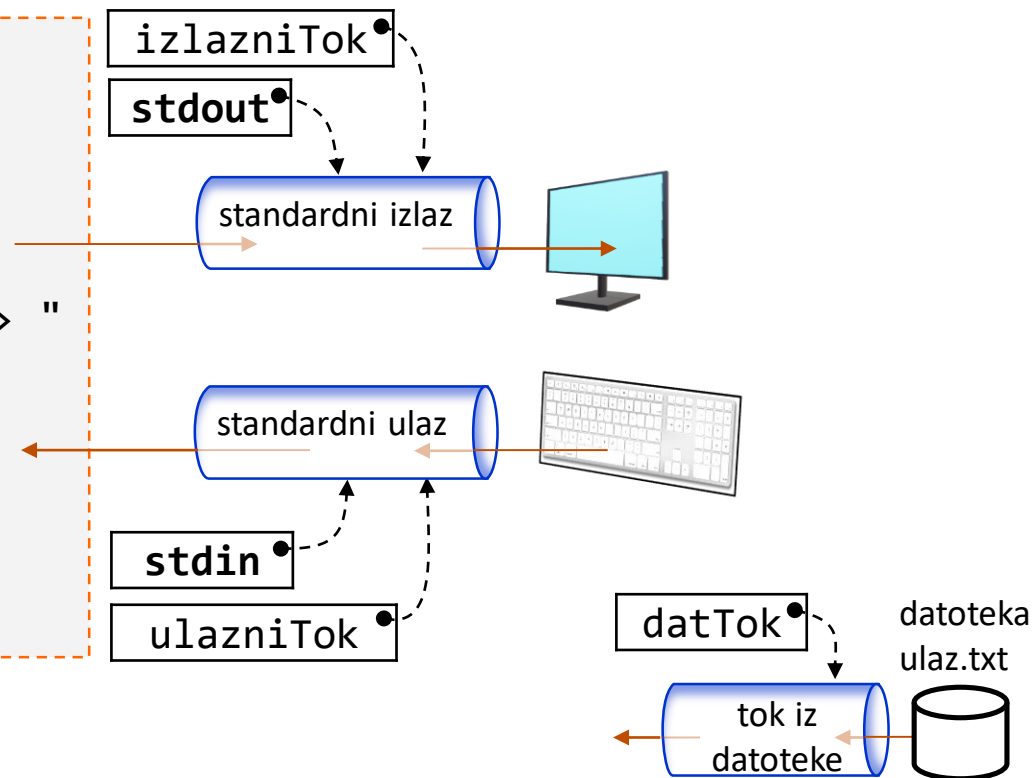
- printf zadani sadržaj uvijek piše na *standardni izlaz*
 - fprintf ima **dodatni parametar**: pokazivač na *tok* u kojeg treba ispisati zadani sadržaj (može biti pokazivač na *standardni izlaz* ili na neki drugi *tok*)
 - fprintf(**stdout**, "format", ...) ≡ printf("format", ...)
- scanf sadržaj uvijek 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 ili zadanog ulaznog *toka*
 - `getchar() ≡ 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 pročitano 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*

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

- vraćanje (*push back*) jednog upravo pročitano 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

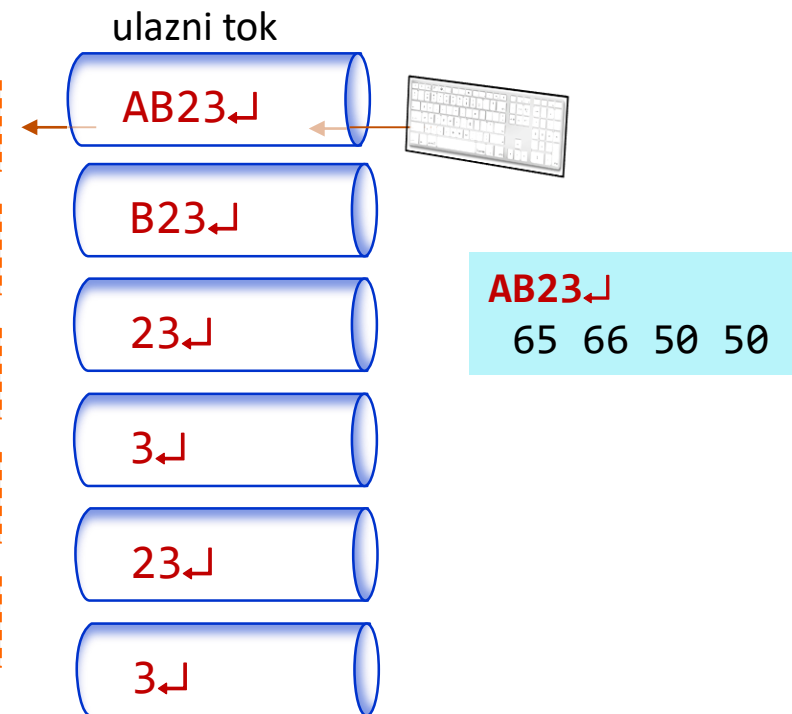
```
printf("%3d", c = getchar()); 65
```

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

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

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

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



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

- ispis jednog znaka na standardni izlaz ili zadani izlazni *tok*
 - `putchar(c) ≡ fputc(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

```
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 uspjelo, funkcija vraća vrijednost parametra s
 - ako čitanje nije uspjelo (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 nepročitan 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 ili zadani izlazni *tok*
 - `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;
}
```

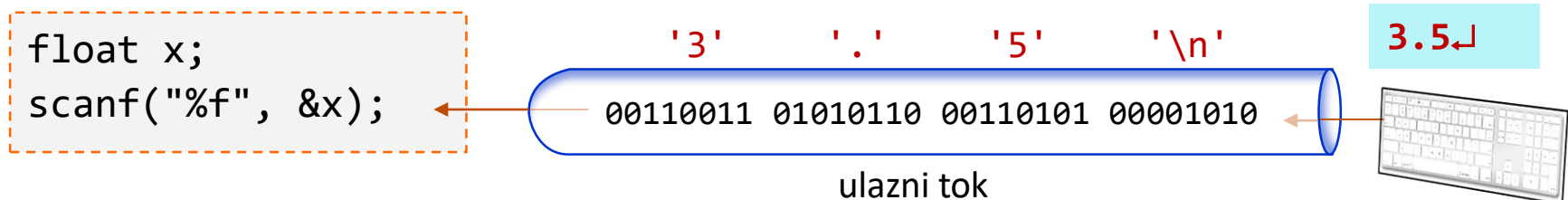
- Za vježbu analizirati:
 - koji bi se rezultat dobio kada bi se makro MAXNIZ promijenio na 10?
 - zašto se program neće zaustaviti ako se upiše redak **nigdje KRAJA↵**, a zaustavit će se ako se umjesto tog retka upiše redak **ima KRAJA↵**

```
int scanf(const char *format, ...);  
int fscanf(FILE *stream, const char *format, ...);
```

- `scanf(format, ...) ≡ fscanf(stdin, format, ...)`
 - čitanje znakova iz standardnog ulaza ili zadanog ulaznog *toka*, u skladu s formatom kojim se definiraju
 - dopušteni oblik ulaza
 - vrste *konverzija* koje nad ulazom treba obaviti da bi se dobile vrijednosti točno određenog tipa
 - vrijednosti dobivene *konverzijom* znakova s ulaza redom se upisuju na mjesta određena preostalim argumentima (pokazivačima na objekte)
 - funkcija vraća broj vrijednosti koje je uspjela upisati u objekte na koje pokazuju argumenti
- Opisane su tek najvažnije mogućnosti funkcije `scanf`.
 - Detaljniji opis funkcije `scanf` može se pronaći u gotovo svakom C priručniku.

Zašto se govori o *konverziji*?

- funkcija iz ulaznog toka čita **znakove** (bajtove od kojih svaki sadrži po jednu ASCII vrijednost znaka)



- obavlja konverziju **pročitanih znakova** u podatak odgovarajućeg tipa (tip podatka određen je konverzijskom specifikacijom). U konkretnom slučaju, realni broj standardne preciznosti, IEEE 754
 - 01000000 01100000 00000000 00000000
- rezultat dobiven konverzijom upisuje na mjesto u memoriji na koje pokazuje argument `&x`, dakle u varijablu `x`
- budući da se konverzija obavlja prema specifikacijama iz formata
 - `scanf`, `printf` i slične funkcije se nazivaju funkcije za formatirani ulaz/izlaz

Format za funkciju scanf

- format za funkciju scanf može sadržavati
 - konverzijske specifikacije
 - npr. %d, %f
 - bijele praznine, odnosno *bjeline* (*white-space*)
 - znak praznine (*space*, ASCII 32₁₀) ili vodoravni tabulator (*horizontal tab*, ASCII 11₁₀)
 - ostale (*non-white-space*) znakove

Konverzijska specifikacija

- Opći oblik `%[*][širina][modifikator]specifikator`

| opcionalni modifikator | specifikator | konverzija u tip |
|---|--------------|---------------------------|
| | c | znak |
| h - short l - long ll - long long | d | cijeli broj |
| | u | cijeli broj bez predznaka |
| | o | oktalni cijeli broj |
| | x | heksadekadski cijeli broj |
| l - double L - long double | f | realni broj |
| | s, [...] | niz znakova |

- npr. `%hd`: konverzijska specifikacija za konverziju u short `int`
- konverzijske specifikacije nalažu sljedeće
 - preskoči sve eventualne bjeline na ulazu (osim za specifikator `c` i `[...]`)
 - čitaj dok god znakovi na ulazu odgovaraju specifikatoru

Primjer

```
short i; unsigned int j; int k;  
float x; double y;  
char c; char niz[20];  
scanf("%hd%x%d", &i, &j, &k);  
scanf("%f%lf", &x, &y);  
scanf("%c%s", &c, niz);
```

....12...
..F2
512.-7.5e5..2.
...Iva...

- %hd preskače 4 praznine, čita znakove 12, pretvara ih u short 12
- %x preskače 5 praznina, novi red, 2 praznine, čita znakove F2, pretvara ih u unsigned int 242
- %d preskače novi red, čita znakove 512 i pretvara ih u int 512
- %f preskače prazninu, čita znakove -7.5e5 i pretvara ih u float $-7.5 \cdot 10^5$
- %lf preskače prazninu, čita znakove .2 i pretvara ih u double 0.2
- %c ne preskače ništa, čita prazninu i upisuje njezinu ASCII vrijednost u varijablu c
- %s preskače novi red, 3 praznine, čita znakove Iva, upisuje ih u polje niz i dodaje '\0'
- dvije praznine i znak za novi red ostaju na ulazu, nepročitani

Specifikator [*znakovi*]

- konverzijske specifikacije %[*znakovi*] i %[[^]*znakovi*] slične su specifikaciji %s po tome što čitaju znakove s ulaza i pohranjuju ih u niz znakova kojeg na kraju terminiraju znakom '\0', uz sljedeće razlike:
 - ne preskaču bjeline na ulazu
 - %[*znakovi*] čita sve znakove s ulaza koji se nalaze na popisu znakova
 - %[[^]*znakovi*] čita sve znakove s ulaza koji se **ne** nalaze na popisu znakova
- primjeri specifikacija
 - %[abcdefABCDEF0123456789]
 - učitava znakove dok god se na ulazu nalaze heksadekadske znamenke
 - %[·\n]
 - učitava znakove dok god se na ulazu nalaze praznine ili oznake novog retka
 - %[[^]\n!]
 - učitava znakove dok god se na ulazu ne pojavi oznaka novog retka ili znak uskličnik

Primjer

```
char niz1[40], niz2[40];
char niz3[40], niz4[40];

scanf("%[ABCDEF0123456789]", niz1);
scanf("%[^\\n!]", niz2);
scanf("%[.!\n]", niz3);
scanf("%[^.]", niz4);
```

C00Fa...!..
.....
.Mali.auto.
dobro.voji..Narocito.
u.gradu.

- `%[ABCDEF0123456789]` čita znakove C00F, upisuje u `niz1` i terminira `niz1`
- `%[^\\n!]` čita znakove a..., upisuje u `niz2` i terminira `niz2`
- `%[.!\n]` čita znakove !.., upisuje u `niz3` i terminira `niz3`
- `%[^.]` čita znakove Mali.auto, upisuje u `niz4` i terminira `niz4`
- znakovi .Narocito.u.gradu ostaju na ulazu, nepročitani

Konverzijska specifikacija

`%[*][širina][modifikator]specifikator`

- opcionalna širina (pozitivni cijeli broj)
 - čitaj najviše zadani broj znakova

```
int i, j;
```

```
float x;
```

```
scanf("%3d%5d", &i, &j);
```

```
scanf("%5f", &x);
```

12345...3.1415926↵

- `%3d` čita znakove 123 i pretvara ih u `int` 123
- `%5d` čita znakove 45 i pretvara ih u `int` 45
- `%5f` preskače 3 praznine, čita znakove 3.141 i pretvara ih u `float` 3.141
- znakovi 5926↵ ostaju na ulazu, nepročitani

Konverzijska specifikacija

%[*][širina][modifikator]specifikator

- instrukcija funkciji scanf da treba suspendirati konverziju i pridruživanje. Znakovi će se pročitati u skladu s ostatkom navedene specifikacije, ali se konverzija neće obaviti
 - to znači da za dotičnu specifikaciju ne treba navesti pripadni pokazivač
 - korisno ako neke znakove na ulazu treba samo preskočiti

```
int i; float x;  
scanf("%*#[#]%d", &i);  
scanf("%*[\n#C]%f", &x);
```

```
#####150·C↵  
#####9.65·paskala↵
```

- %*#[#] čita znakove ##### i zanemaruje ih
- %d čita znakove 150 i pretvara ih u int 150
- %*[\n#C] čita znakove ·C##### i zanemaruje ih
- %f čita znakove 9.65 i pretvara ih u float 9.65
- znakovi ·paskala↵ ostaju na ulazu, nepročítani

Bjeline i ostali znakovi navedeni u formatu

- Bjelina u formatu
 - instrukcija za funkciju `scanf` da treba preskočiti sve sljedeće bjeline na ulazu (*space*, *horizontal tab* i *newline*) dok god se ne pojavi neki *non-white-space* znak
- Ostali (*non-white-space*) znakovi u formatu
 - svaki *non-white-space* znak zahtijeva da se na ulazu pojavi upravo takav znak

Napomena uz konverzijsku specifikaciju %c

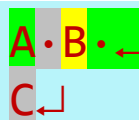
- specifikacija %c ne preskače bjeline na ulazu. Stoga, ako se namjerava učitati samo znak koji nije bjelina, potrebno je prvo pomoću bjeline navedene u formatu preskočiti bjeline na ulazu

```
char c1, c2, c3;  
scanf("%c%c%c", &c1, &c2, &c3);
```

- %c čita znak A i upisuje ga u c1
- %c čita znak praznine i upisuje ga u c2
- %c čita znak B i upisuje ga u c3

```
char c1, c2, c3;  
scanf("·%c·%c·%c", &c1, &c2, &c3);
```

- u formatu nema što preskočiti na ulazu
- %c čita znak A i upisuje ga u c1
- u formatu preskače znak · na ulazu
- %c čita znak B i upisuje ga u c2
- u formatu preskače znakove ·↵ na ulazu
- %c čita znak C i upisuje ga u c3



Napomena uz konverzijsku specifikaciju %s

- specifikacija %s prestaje učitavati niz znakova kada naiđe na prvu bjelinu na ulazu

```
char niz1[40], niz2[40], niz3[40];  
scanf("%20s", niz1);  
scanf("%s", niz2);  
scanf("%3s", niz3);
```



..Ana·Marija↵
...Ivana·↵

- %20s preskače znakove ··, čita znakove Ana, upisuje ih u niz1 i terminira niz
- %s preskače znak ·, čita znakove Marija, upisuje ih u niz2 i terminira niz
- %3s preskače znakove ↵···, čita znakove Iva, upisuje ih u niz3 i terminira niz
- znakovi na ·↵ ostaju na ulazu, nepročitani

Napomena uz konverzijsku specifikaciju %s

- niz znakova koji sadrži bjeline može se pročitati pomoću %[...]

```
char niz1[40], niz2[40];  
scanf("%[^\\n]*c", niz1);  
scanf("%[^\\n]*c", niz2);
```



..Ana.Marija.
...Ivana.

- %[^\n] čita znakove ..Ana.Marija, upisuje ih u niz1 i terminira ga. Zaustavlja se ispred \n.
- %*c preskače znak \n
- %[^\n] čita znakove ...Ivana., upisuje ih u niz2 i terminira ga. Zaustavlja se ispred \n.
- %*c preskače znak \n

Prijevremeni prekid izvršavanja funkcije

- ako znak na ulazu ne odgovara konverzijskoj specifikaciji, funkcija se prijevremeno prekida. Ostatak ulaza, uključujući znak koji nije odgovarao konverzijskoj specifikaciji, ostaje nepročitano.

```
int n, rez; float x, y;  
rez = scanf("%f%d%f", &x, &n, &y);
```

..1.5..62.50↵

- %f preskače .., čita znakove 1.5, pretvara u float 1.5, upisuje u x
- %d preskače znakove .., čita znakove 62, pretvara u int 62, upisuje u n
- %f čita znakove .50, pretvara u float 0.5, upisuje u y
- znak ↵ ostaju na ulazu, nepročitano. Funkcija je vratila 3 \Rightarrow rez = 3

```
int n, rez; float x, y;  
rez = scanf("%f%d%f", &x, &n, &y);
```

..1.5..62.50↵

- %f preskače .., čita znakove 1.5, pretvara u float 1.5, upisuje u x
- %d preskače znakove .., točka nije u skladu s %d, vraća ju u stdin
- znakovi .62.50↵ ostaju na ulazu, nepročitani. Funkcija je vratila 1 \Rightarrow rez = 1

Primjeri

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

```
....38↵  
...-15.012.....↵  
..24+5e2...-8↵
```

- %d preskače , čita znakove 38, pretvara u int 38, upisuje u i
- %d preskače znakove ↵... , čita znakove -15, pretvara u int -15, upisuje u j
- • u formatu bi preskočila bjeline kad bi ih bilo na ulazu. Ne radi ništa.
- %f bi preskočio bjeline kad bi ih bilo na ulazu. Čita znakove .012, pretvara u float 0.012, upisuje u x
- • u formatu preskače↵..
- %f bi preskočio bjeline kad bi ih bilo na ulazu. Čita znakove 24, pretvara u float 24.0, upisuje u y
- • u formatu bi preskočila bjeline kad bi ih bilo na ulazu. Ne radi ništa.
- %f bi preskočio bjeline kad bi ih bilo. Čita +5e2, pretvara u float 500.0, upisuje u z
- • u formatu preskače ...
- znakovi -8↵ ostaju na ulazu, nepročitani. Funkcija vraća 5

Primjeri

```
int d, m, g, rez;  
rez = scanf("Datum:%d.%d.%d", &d, &m, &g);
```

Datum: 15.1.2017.↵

- Datum: preskače Datum:
- %d preskače . , čita 15, pretvara i upisuje u d
- . iz formata preskače . na ulazu ... itd. Na kraju, znakovi .↵ ostaju na ulazu, nepročitani

```
rez = scanf("Datum:%d.%d.%d", &d, &m, &g);
```

Datum: : 15.1.2017.↵

- Datum: ne odgovara znakovima na ulazu. Funkcija se prijevremeno prekida, vraća 0
- znakovi : 15.1.2017. ostaju na ulazu, nepročitani
- za popravak bi bilo dovoljno u formatu iza Datum staviti jednu prazninu

```
rez = scanf("Datum: %d.%d.%d", &d, &m, &g);
```

Datum: 15.1.2017.↵

- . u formatu bi preskočila bjeline kad bi ih bilo na ulazu. Ne radi ništa.
- svi cijeli brojevi će se uspjeti pročitati. Funkcija vraća 3
- na kraju, znakovi .↵ ostaju na ulazu, nepročitani

```
int printf(const char *format, ...);  
int fprintf(FILE *stream, const char *format, ...);
```

- `printf(format, ...) ≡ fprintf(stdout, format, ...)`
- pisanje znakova na standardni izlaz ili zadani izlazni *tok*, u skladu sa zadanim formatom
- vrijednosti argumenata navedenih iza formata formatiraju se u skladu s konverzijskim specifikacijama
 - ostali znakovi koji se nalaze u formatu ispisuju se nepromijenjeni
- funkcija vraća broj znakova koje je ispisala ili EOF ako se pri ispisu dogodi pogreška

- Opisane su tek najvažnije mogućnosti funkcije `printf`.
- Detaljniji opis funkcije `printf` može se pronaći u gotovo svakom C priručniku.

Konverzijska specifikacija

- Opći oblik %[znak][širina][.preciznost][modifikator]specifikator

| opcionalni modifikator | specifikator | konverzija iz tipa i oblik ispisa |
|---|--------------|--|
| | c | znak |
| h - short l - long ll - long long | d | cijeli broj |
| | u | cijeli broj bez predznaka |
| | o | oktalni cijeli broj |
| | x, X | heksadekadski cijeli broj (a-f ili A-F) |
| l - double L - long double | f | realni broj bez prikaza eksponenta |
| | e, E | realni broj s eksponentom, ispis e ili E |
| | g, G | realni broj s eksponentom ili bez, ispis e ili E |
| | s | niz znakova |
| | p | pokazivač |

Konverzijska specifikacija

`%[znak][širina][.preciznost][modifikator]specifikator`

- širina (pozitivni cijeli broj) određuje najmanju širinu polja za ispis
 - 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 (pozitivni cijeli broj)
 - za e , E , f : određuje broj znamenki iza decimalne točke
 - za ostale specifikatore ima drugačije značenje, ovdje se preciznost za te specifikatore neće razmatrati
 - ako se preciznost ne zada, koristi se preciznost po definiciji (npr. za e , E , f , to je šest znamenki iza decimalne točke)

Konverzijska specifikacija

%[znak][širina][.preciznost][modifikator]specifikator

- pruža dodatne mogućnosti prilagodbe ispisa, npr.
 - ispis vodećih nula
 - ispis predznaka i za pozitivne brojeve
 - lijevo pozicioniranje u polju ispisa

```
printf("%5d\n", 25);  
printf("%05d\n", 25);  
printf("%+5d\n", 25);  
printf("%-5d\n", 25);
```

```
...25↵  
00025↵  
..+25↵  
25...↵
```

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|
| 321.000000000000|0.00000012340|7654320128.000000000000|
```

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|
```

Preusmjeravanje toka *standardni izlaz*

- Standardni izlazni tok može se preusmjeriti u trenutku pokretanja programa (na razini operacijskog sustava)

```
...
int main(void) {
    ...
    printf("Upisite n > ");
    scanf("%d", &n);
    if (n < 1 || n > 46) {
        printf("Prevelik/premalen n");
        exit(1);
    }
    for (i = 1; i <= n; i = i + 1) {
        ...
        printf("%d\n", fib_i);
    }
    ...
}
```

```
C:\upro> prog > fib.txt
10
```

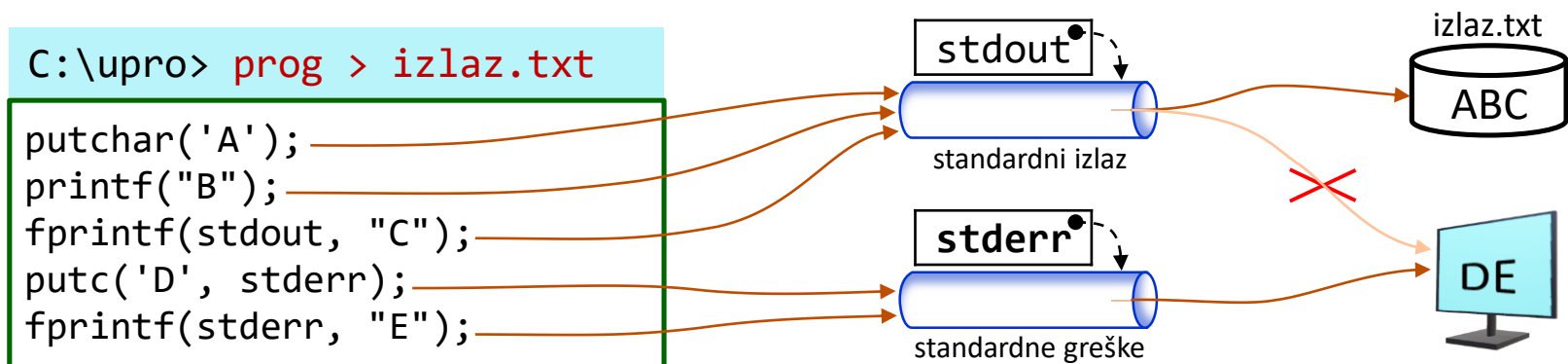
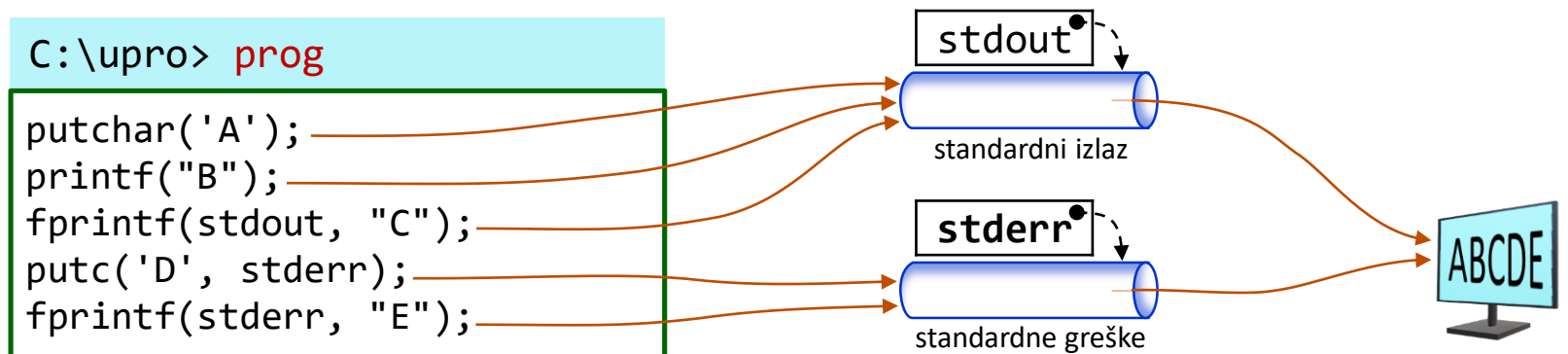
```
fib.txt Upisite n > 1
1
2
3
5
...
```

```
C:\upro> prog > fib.txt
50
```

```
fib.txt Upisite n > Prevelik/premalen n
```

Tok standardne greške

- pored tokova standardni ulaz i standardni izlaz, u trenutku pokretanja programa automatski se otvara i tok *standardne greške*
 - slično eksternim varijablama `stdin` i `stdout`, eksterna varijabla `stderr` je tipa pokazivač na tok *standardne greške*



Korištenje toka *standardna greška*

- Ako neki sadržaj na zaslon treba ispisati bez obzira na eventualno preusmjeravanje standardnog izlaza, treba koristiti izlazni tok standardne greške

```
...
int main(void) {
    ...
    fprintf(stderr, "Upisite n > ");
    scanf("%d", &n);
    if (n < 1 || n > 46) {
        fprintf(stderr, "Prevelik/premalen n");
        exit(1);
    }
    for (i = 1; i <= n; i = i + 1) {
        ...
        printf("%d\n", a_i);
    }
    ...
}
```

```
C:\upro> prog > fib.txt
Upisite n > 10
```

fib.txt

```
1
1
2
3
5
...
```

```
C:\upro> prog > fib.txt
Upisite n > 50
Prevelik/premalen n
```

fib.txt

Preusmjeravanje toka *standardni ulaz*

- Preusmjeravanje standardnog ulaza

```
...  
int main(void) {  
    ...  
    printf("Upisite n > ");  
    scanf("%d", &n);  
    if (n < 1 || n > 46) {  
        printf("Prevelik/premalen n");  
        exit(1);  
    }  
    for (i = 1; i <= n; i = i + 1) {  
        ...  
        printf("%d\n", fib_i);  
    }  
    ...  
}
```

ulaz.txt

10

```
C:\upro> prog < ulaz.txt↵
```

```
Upisite n > 1
```

1

2

3

5

...

Preusmjeravanje toka *standardni ulaz i izlaz*

- Istovremeno preusmjeravanje standardnog ulaza i izlaza

```
...  
int main(void) {  
    ...  
    printf("Upisite n > ");  
    scanf("%d", &n);  
    if (n < 1 || n > 46) {  
        printf("Prevelik/premalen n");  
        exit(1);  
    }  
    for (i = 1; i <= n; i = i + 1) {  
        ...  
        printf("%d\n", fib_i);  
    }  
    ...  
}
```

ulaz.txt

10

```
C:\upro> prog < ulaz.txt > fib.txt ↵
```

fib.txt

```
Upisite n > 1  
1  
2  
3  
5  
...
```

```
int sscanf(const char* buffer, const char* format, ...);  
int sprintf(char *buffer, const char *format, ...);
```

- identične funkcijama fscanf i fprintf, osim po sljedećem: umjesto toka, kao izvor, odnosno destinaciju, funkcije koriste niz znakova.
 - sscanf je korisna u slučajevima kada je niz znakova koji se nalazi na ulazu potrebno formatirano pročitati više nego jednom
 - sprintf automatski terminira niz znakom '\0'. Funkcija je korisna onda kada u *niz znakova* treba upisati nešto u skladu sa zadanim formatom, a taj niz se neće ispisivati u neki tok ili će se ispisivati tek kasnije

Primjer

- Programski zadatak
 - s tipkovnice učitati niz znakova koji zajedno s eventualno učitanim oznakom novog retka ne smije biti dulji od 20 znakova
 - ako učitani niz sadrži podniz OCT ili HEX, tada od početka niza pročitati oktalni, odnosno heksadekadski broj, te ga ispisati kao dekadski broj. Ako niz ne sadrži podniz OCT ili HEX ili učitavanje broja s početka niza ne uspije, ispisati poruku "Neispravan ulaz"

```
Upisite niz > 12OCT↵
```

```
Ucitan je broj 10↵
```

```
Upisite niz > 1eM HEX↵
```

```
Ucitan je broj 30↵
```

```
Upisite niz > M1e HEX↵
```

```
Neispravan ulaz
```

```
Upisite niz > 219 hex ↵
```

```
Neispravan ulaz
```

Rješenje

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

int main(void) {
    char niz[20 + 1];
    unsigned int broj;
    int procitanoBrojeva = 0;
    printf("Upisite niz > ");
    fgets(niz, 20 + 1, stdin);
    if (strstr(niz, "OCT") != NULL)
        procitanoBrojeva = sscanf(niz, "%o", &broj);
    else if (strstr(niz, "HEX") != NULL)
        procitanoBrojeva = sscanf(niz, "%x", &broj);

    if (procitanoBrojeva == 1)
        printf("Ucitan je broj %d\n", broj);
    else
        printf("Neispravan ulaz\n");

    return 0;
}
```

Primjer

- Funkcije `sprintf` i `sscanf` ne smiju se koristiti bez razloga!
 - ako program s tipkovnice treba samo pročitati, a zatim na zaslon ispisati tri cijela broja, *vrlo loše* je napisati:

```
char ulaz[80], izlaz[80];  
int m, n, k;  
fgets(ulaz, 80, stdin);  
sscanf(ulaz, "%d %d %d", &m, &n, &k);  
sprintf(izlaz, "%d %d %d", m, n, k);  
puts(izlaz);
```

NEISPRAVNO

- umjesto:

```
int m, n, k;  
scanf("%d %d %d", &m, &n, &k);  
printf("%d %d %d", m, n, k);
```

ISPRAVNO