Branje in pisanje 2

1 Scanf

Funkcija `scanf` lahko bere različne vrste podatkov:

- `%s` - beseda (vsi znaki razen praznih znakov (glej spodaj)) (`char`)
- `%c` - en (poljuben) znak (`char`)
- `%d` - število med -2^{31} in $2^{31} - 1$ (`int`)
- `%lld` - število med -2^{63} in $2^{63} - 1$ (`long long`)
- ...

Prazni znaki (whitespace) so znaki, ki jih ne vidimo:

- `\n` nova vrstica
- `\t` zamik
- presledek

S formatnikom `%s` funkcija `scanf` bere do prvega takšnega znaka. Prebere tudi vse take znake, ki sledijo, a jih ne shrani. Naslednjič, ko jo pokličemo, bere od prvega nepraznega znaka naprej.

Primer

```c
#include<stdio.h>

int main(){
        char a[50], b[50];
        scanf("%s", a);
        scanf("%s", b);
        printf("%s %s\n", a, b);
        return 0;
}
```

Primer vhoda in izhoda

```
Hello

World!
```

```
Hello World!
```

2 Branje do konca vrstice

Lahko določimo, da se `scanf` ne bo ustavil pri prvem praznem znaku, temveč šele pri koncu vrstice (ali kje drugje).

Primer

```c
#include<stdio.h>

int main(){
        char a[50];
        scanf("%[^\n]", a); //ali scanf("%[^\n]%*c") in brez getchar()
        getchar();
        printf("%s\n", a);
        return 0;
}
```

Primer vhoda in izhoda

```
Beremo do konca vrstice.
```

```
Beremo do konca vrstice.
```

`[...]` - med oklepaje pišemo navodila, kakšne znake lahko bere

- `[a-z]` - beri male črke angleške abecede
- `[a-zA-Z0-9]` - beri male in velike črke in številke

`[^...]` - beri vse do znakov, ki sledijo strešici

- `[^\n]` - beri vse do `\n`

Pogoste napake

Za razliko od `%s` funkcija `scanf` s `[^\n]` prebere vse do `\n`, tega pa ne prebere in se ustavi pred njim. Ko funkcijo pokličemo naslednjič, začne tam, kjer je nazadnje ostala, kar je v tem primeru točno pred znakom `\n`. Če jo torej ponovno pokličemo s parametrom `[^\n]`, se ne bo nikamor premaknila, saj je pred njo znak za novo vrstico.

`%c` - prebere en znak (`char`), ki je lahko karkoli - črka, številka, prazen znak...
`%*c` - prebere en znak, a ga ne shrani
Funkcija `getchar` dela podobno, vzame en znak in ga ne shrani.

Za razliko od tega s formatnikom `%s` preberemo vse prazne znake med dvema nepraznima nizoma in jih ne shranimo. Tudi, če bodo prazni znaki pred besedo, jih bo program ignoriral in poiskal prvi neprazen znak.

3 Branje do konca vhoda

Če vemo točno, koliko besed/številk/vrstic bomo imeli na vhodu, jih lahko preberemo s for zanko. Kako preberemo neznano količino podatkov na vhodu, tako da preberemo vse?
Če napišemo `while(true)` ali `while(1)`, bomo sicer prebrali vse, a se program ne bo nikoli ustavil. Namesto tega lahko napišemo:

Primer

```c
#include<stdio.h>

int main(){
        int a;
        while(scanf("%d", &a) != EOF){
                printf("%d\n", a*a); //izpisujemo kvadrate prebranega števila
        }
        return 0;
}
```

Primer vhoda in izhoda

```
1 2 3 4 5
```

```
1
4
9
16
25
```

Funkcija `scanf` vrne število formatnikov, ki jih je uspešno prebrala (če ji kot parameter podamo samo `%d`, bo vrnila 1, če je uspešno prebrala število, sicer pa 0). `EOF` (End of File) vrne, če na vhodu ni ničesar več za prebrati. Tedaj se bo zanka ustavila. Če programu vhodne podatke podajamo iz datoteke, se to zgodi avtomatsko ob koncu datoteke, če pa mu podatke podajamo na roko, konec vhoda sporočimo s `Ctrl+D (Linux in MacOS)` ali `Ctrl+Z` (Windows).

Primer

```c
#include<stdio.h>

int main() {
    int a, b, c;
    int r = scanf("%d%d%d", &a, &b, &c);
    printf("%d\n", r);
    return 0;
};
```

Primer vhoda in izhoda

```
5 8 100
```

```
3
```

Primer vhoda in izhoda

```
5 8 miha
```

```
2
```

4 Branje in pisanje v in iz niza

`scanf` uporabljamo za branje s standardnega vhoda, `printf` pa za pisanje na standardni izhod. Namesto tega lahko beremo in pišemo tudi drugače, npr. v in iz nizov. Za to uporabljamo funkciji `sscanf` in `sprintf`, ki delata podobno kot `scanf` in `printf`.

Primer

```c
#include<stdio.h>

int main(){
        int n;
        char a[10], b;
        char text[]="Slovenska 157 a";
        sscanf(text, "%s%d %c", a, &n, &b);
        sprintf(text, "%c %d %s", b, n, a);
        printf("%s\n", text);
        return 0;
}
```

Primer vhoda in izhoda

```
a 157 Slovenska
```

Pogoste napake

Medtem ko `%s` praznih znakov ne shrani, jih `%c` obravnava tako kot vse ostale. Če pogeldamo prejšnji primer vidimo, da je v funkciji `scanf` pred `%c` presledek. Ker so med posameznimi podatki, ki jih želimo prebrati, presledki, bi `%c` pobral presledek, ne pa črke, ki mu sledi. Če med posamezne formatnike postavimo presledek, ta načeloma pobere prazne znake do naslednjega drugačnega znaka, vendar to v splošnem ni dobra praksa.

Funkciji `sscanf` podamo tri parametre (ali več):

1. ime niza, iz katerega naj bere
2. tip podatka, ki naj ga prebere (niz, število...)
3. kam naj ta podatek zapiše (ime spremenljivke)

Funkciji `sprintf` prav tako podamo tri parametre (ali več):

1. ime niza, kamor naj piše
2. tip podatka, ki naj ga zapiše
3. kaj naj zapiše (ime spremenljivke ali podatek sam)

5 Branje in pisanje v in iz datoteke

Podobno kot v niz lahko pišemo in beremo tudi v in iz datotek s funkcijama `fscanf` in `fprintf`.

Primer

```c
#include<stdio.h>

int main(){
        int a;
        FILE *fr, *fw;
        fr = fopen("in.txt", "r");
        fw = fopen("out.txt", "w");
        while(fscanf(fr, "%d", &a) != EOF){
                fprintf(fw, "%d\n", a*a);
        } //iz datoteke fr preberemo vsa števila in v fw izpišemo njihove kvadrate
        fclose(fr);
        fclose(fw);
        return 0;
}
```

Primer vhoda in izhoda

in.txt:

```
1 2 3 4 5 6 7
```

out.txt:

```
1
4
9
16
25
36
49
```

(Ta program ne bere s standardega vhoda in ne piše ne standardni izhod.)

`FILE` - tip podatka `fopen` - funkcija, s katero odpremo datoteko, podamo ji ime datoteke, ki jo želimo odpreti in način `"r"` (*read* - za branje) ali `"w"` (*write* - za pisanje).
Datoteke, ki jih odpiramo za branje, morajo že prej obstajati, sicer se bo program sesul.

Za datoteke, v katere pišemo, ni nujno, da že obstajajo. Če še ne obstajajo, bo program ustvaril novo datoteko s tem imenom. Če že obstajajo, program ne bo pisal na konec te datoteke, temveč bo pobrisal vso prejšnjo vsebino. Če želimo obstoječi datoteki dodajati vsebino, moramo uporabiti način `"a"` (*append* - pripenjanje).
`fclose` - funkcija, ki zapre odprto datoteko, podamo ji ime datoteke