Zapiski s priprav

Vhod in izhod: dopolnitev

ACM priprave na tekmovanja

Različica z dne 19. oktober 2024

Kazalo

1 Dodatni formatniki za `scanf`

Poleg že znanih formatnikov `%d`, `%lld` in `%s` poznamo tudi druge, ki so lahko uporabni v različnih situacijah. Seznam pogosto uporabnih formatnikov je v tabeli 1. V tabeli se pojavi izraz *prazen znak*, angl. *whitespace* — to je znak, ki zasede prostor v spominu (in kateremu pripada ASCII koda), vendar se na zaslonu ne pojavi. Poznamo tri take znake, to se presledek, tabulator `\t` in znak za novo vrstico `\n`. Formatniki `%d`, `%lld` in `%s` preberejo, a ignorirajo, prazne znake, ki se pojavijo pred vsebino, ki jo formatnik dejansko shrani. Če npr. v spodnji primer programa vpišemo vhod

```
Hello

␣␣␣World!
```

bo program prebral le ti dve besedi, in ne praznih znakov med njima.

formatnik	opis
`%%`	en znak `%`
`%d`	število tipa `int`
`%lld`	število tipa `long long`
`%c`	poljuben znak (lahko tudi prazen znak)
`%s`	zaporedje nepraznih znakov
`%[...]`	zaporedje znakov, ki se pojavijo med oglatimi oklepaji
`%[^...]`	zaporedje znakov, ki se ne pojavijo med oglatimi oklepaji

Tabela 1: Različni formatniki za `scanf` in `printf`

```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;
}
```

Zadnja formatnika v tabeli, `%[...]` in `%[^...]` sta bolj zapletena od ostalih. Uporabljamo ju tako, da namesto tropičja v oglate oklepaje zapišemo seznam znakov, ki so dovoljeni v prebranem besedilu. Tako lahko npr. enostavno ločimo besedo, sestavljeno iz črk in številk na niz in na število, ki ga lahko takoj shranimo v `int`:

```c
#include <stdio.h>

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

Če zgornjemu programu kot vhod podamo `aacbb012`, bo izpisal

```
Niz: aacbb, stevilo: 12
```

Če želimo dovoliti vse male črke abecede, lahko namesto seznama vseh črk zapišemo tudi `%[a-z]`, podobno lahko za velike črke uporabimo `%[A-Z]`, za števke pa `%[0-9]`. Formatnik bo v zadnjem primeru še vedno prebral niz znakov, in ga ne bo avtomatsko pretvoril v `int`. Tudi te obsege lahko kombiniramo, formatnik `%[a-z2-7B]` npr. prebere niz, sestavljen iz malih črk angleške abecede, števk med 2 in 7 ter velike črke B.

Formatnik `%[^...]` deluje podobno kot formatnik `%[...]`, le da prebere vse znake, razen tistih, ki smo jih zapisali namesto tropičja. Ta formatnik je uporaben pri branju niza do konca vrstice, česar s formatnikom `%s` ne moremo narediti, če nimamo podanega števila besed v vrstici. Primer je prikazan v spodnjem programu, kjer uporabimo tudi formatnik `%*c`. Ta deluje enako kot formatnik `%c`, le da znaka, ki ga prebere, nikjer ne shrani, zato zanj tudi ne podamo spremenljivke. Ta formatnik je potreben, ker `%[^\n]` ne prebere znaka za novo vrstico, ki ga „počistimo“ z `%*c`. Če bi želeli prebrati dve zaporedni vrstici in nebi uporabili `%*c`, nebi druga uporaba formatnika `%[^\n]` shranila ničesar, saj bi nemudoma naletela na znak za novo vrstico, in zaključila branje.

```c
#include<stdio.h>

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

2 Neznano število podatkov

Če vemo točno, koliko besed, številk oz. vrstic bomo imeli na vhodu, jih lahko preberemo z zanko. Kaj pa, če ne vemo, koliko podatkov bo na vhodu, kot v nalogi Neznane vsote? Če podatke beremo v neskončni zanki, bomo sicer prebrali vse, a se program ne bo nikoli ustavil. V naslednjem primeru je nakazano, kako ta problem rešimo:

```c
#include<stdio.h>

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

Funkcija `scanf` vrne število formatnikov, ki jih je uspešno prebrala (če ji kot parameter npr. podamo samo `%d`, bo vrnila 1, če je uspešno prebrala število, sicer pa 0). Posebno vrednost `EOF` (End of File) pa 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).

3 Uporaba nizov ali datotek za vhodno-izhodni sistem

Včasih si lahko pri procesiranju podatkov pomagamo s tem, da dano besedilo prvo shranimo v niz, preden ga pretvorimo v drugo obliko, ali iz njega izluščimo podatke. Pri tem si lahko pomagamo s funkcijama `sscanf` in `ssprintf`, ki delujeta podobno kot poznana `scanf` in `printf`, vendar namesto branja iz oz. pisanja na standardni vhod oz. izhod delujeta z nizi v spominu. Niz, s katerega beremo oz. na katerega pišemo, podamo kot prvi argument v ti funkciji, kakor v spodnjem primeru.

```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;
}
```

Podobno lahko s pomočjo funkcij `fscanf` ter `fprintf` beremo iz in pišemo v datoteke na računalniku. Datoteke predstavimo s posebnim tipom `FILE`, pri katerem moramo poleg imena spremenljivke napisati tudi zvezdico (zakaj je temu tako, spoznamo v enem od kasnejših poglavij). Da se povežemo s pravo datoteko, uporabimo funkcijo `fopen`, ki ji podamo dva argumenta. Prvi argument je ime datoteke, ki jo želimo uporabljati, drug argument pa je možnost odpiranja. Za naše potrebe sta dovolj dve možnosti, `"r"`, ki pove, da bomo datoteko odprli v načinu za branje, ter `"w"`, ki pove, da bomo datoteko odprli v načinu za pisanje. Slednja možnost tudi ustvari datoteko z danim imenom, če ta ne obstaja, oziroma izbriše vso besedilo v tej datoteki, če datoteka že ima vsebino. Primer uporabe funkcij `fscanf` in `fprintf` je prikazan spodaj. Opazimo, da se na koncu programa pojavi tudi funkcija `fclose`. Ta funkcija operacijskemu sistemu pove, da smo z branjem oz. pisanjem zaključili, in da lahko spremembe na tej točki shrani v dejansko datoteko (brez klica te funkcije shranjevanje sprememb ni zagotovljeno). Funkcijo moramo nujno poklicati za vsako datoteko, ki smo jo odprli z `fopen`.

```
#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;
}
```