Zapiski s priprav

Vhod in izhod

ACM priprave na tekmovanja

Različica z dne 17. oktober 2024

Kazalo

1 Osnovna struktura programa

Programi za svoje delovanje potrebujejo način za komunikacijo z uporabnikom. Kompleksnejši programi v ta namen uporabljajo ekran, miško in tipkovnico, pri tekmovalnem programiranju pa najpogosteje uporabljamo najpreprostejši način za komunikacijo: pisanje in branje s *standardnega vhoda in izhoda*. Običajno to pomeni, da se nam ob zagonu programa odpre okno, kamor lahko pišemo programu in kamor program izpisuje stvari. Ko želimo, da naš program kaj izpiše, uporabimo *funkcijo* `printf`. Poglejmo si enostaven primer.

```c
#include<stdio.h>

int main(){
    printf("Hello World!\n");
    return 0;
}
```

Funkciji `printf` v dvojnih narekovajih damo besedilo ali števila, ki jih želimo izpisati. Na koncu tega besedila napišemo `\n`, ki označuje, da mora program na tem mestu iti v novo vrstico. To je pomembno vključiti predvsem, če funkcijo `printf` uporabimo večkrat zaporedoma, saj bi bilo sicer celotno besedilo izpisano v eni vrstici.

Posvetimo se tudi splošni obliki zgornje kode, saj vsebuje ključne elemente, ki jih mora vsebovati vsak program. Prva vrstica, `#include <stdio.h>`, pove programu, da bomo uporabljali funkcije za vhod in izhod, konkretno `printf` in kasneje `scanf`. Če te vrstice nebi napisali, bi ob poskusu izvajanja kode sistem javil napako, in trdil, da funkcije `printf` ne pozna.

Besedilo `int main()` računalniku pove, da bodo sledili zaviti oklepaji (to so oklepaji, ki izgledajo {takole}), znotraj katerih bo glavno telo naše kode. Zaenkrat bomo vso našo kodo napisali med te zavite oklepaje, ko pa bomo spoznali sezname in kasneje funkcije, bomo nekaj kode vnesli tudi drugam. Koda v `main` je organizirana v vrstice, ki se morajo končati s podpičjem `;`. Ko se bo program izvedel, se bodo zaporedoma od zgoraj navzdol izvedle vse vrstice, dokler ne pridemo do zadnje vrstice, ki se mora začeti z ukazom `return`. Temu ukazu sledi številka — ta pove, če se je med izvajanjem programa zgodila kakšna napaka. Če je številka enaka 0, se je program končal brez napak, drugim številkam pa pravimo *kode napake*. Te so uporabne predvsem zato, da lahko uporabnik programerju le z eno številko pove, kakšna napaka se je v programu zgodila. Mi bomo v prihodnje večinoma pisali programe, katerih uporabniki bomo sami, zato bomo vedno uporabili kodo 0.

V program lahko dodamo *komentarje*. To je besedilo, ki je sicer napisano v kodi programa, a ne vpliva na njegov potek, ker računalnik komentarjev ne izvede. Zaradi tega lahko komentarji vsebujejo tudi besedilo v naravnem jeziku (slovenščini), ki programerjem razlaga pomen kode poleg komentarja. Na voljo imamo dve vrsti komentarjev:

- Če na začetku vrstice napišemo dve poševnici, `//`, s tem dobimo komentar, ki prikriva besedilo v tej vrstici. Če se ta komentar pojavi sredi vrstice, bo prikril vse besedilo od tam naprej do konca vrstice.
- Če v besedilu zapišemo poševnico in zvezdico, `/*`, s tem dobimo komentar, ki prikriva vso besedilo do vključno prve pojavitve nasprotnega simbola, `*/`.

```c
#include<stdio.h>
int main(){ //komentar do konca vrstice
    printf /*komentar znotraj vrstice*/("Zivjo svet!\n");
    /*
      komentar
      čez
      več
      vrstic
    */
    return 0;
}
```

2 Branje podatkov

Programu lahko sporočimo različne podatke, program pa mora te podatke nekam shraniti, preden jih lahko obravnava. Mestu, kamor podatke shranimo, pravimo *spremenljivka*, saj lahko te podatke med tekom programa spreminjamo. Vsem spremenljivkam v programu damo ime, s katerim se na njih sklicujemo, ter *podatkovni tip*, ki pove, kakšni podatki so v spremenljivki shranjeni (npr. besedilo, številka, …). V spodnjem primeru ustvarimo eno spremenljivko, ki jo imenujemo `tvoje_ime`, njen tip pa je „besedilo dolžine največ 50“.

```c
#include <stdio.h>

int main(){
    char tvoje_ime[50];
    printf("Kako ti je ime?\n");
    scanf("%s", tvoje_ime);
    printf("Zivjo, %s!\n", tvoje_ime);
    return 0;
}
```

Tudi `scanf` je funkcija, ki ji podamo dva ali več *parametrov*. Prvi parameter mora vedno biti niz znotraj narekovajev, ki opisuje, kakšnega tipa so podatki, ki naj jih funkcija prebere. Ta opis podamo s *formatnikom*, v zgornjem primeru `%s` računalniku pove, da bo program prebral eno besedo. Preostali parametri povedo, v katero spremenljivko naj funkcija shrani prebrane podatke. Če imamo v nizu več formatnikov, moramo podati eno spremenljivko za vsak formatnik.

Do zdaj smo funkciji `printf` podali samo točno določeno besedilo, ki smo ga želeli izpisati. Izpisujemo pa lahko tudi spremenljivke, kot smo to naredili v tem zadnjem primeru. Znotraj besedila dodamo formatnike na mesta, kjer želimo, da so spremenljivke, potem pa izven narekovajev naštejemo imena spremenljivk, ki jih želimo izpisati.

V zgornjem primeru smo brali in izpisali niz besedila, kar pa je pravzaprav zahtevnejše od branja in pisanja števil. Za delo z nizi potrebujemo kompleksnejše ukaze, ki jih bomo spoznali kasneje, zato se bomo do nadaljnjega omejili na delo s (celimi) števili. Tem pripada tip `int` (angl. *integer*) ter formatnik `%d`, kakor vidimo v naslednjem primeru.

```c
#include<stdio.h>

int main(){
    int razred;
    printf("Kateri razred si?\n");
    scanf("%d", &razred);
    printf("%d. razred je najboljši.\n", razred);
    return 0;
}
```

Pri branju števil imamo le eno dodatno zahtevo kot pri branju nizov — pred ime spremenljivke moramo zapisati znak `&`. Razlog za tem bomo spoznali, ko bomo obravnavali kazalce, za sedaj pa to vzemimo kot zahtevo postopka. Pri številih tudi ne povemo direktno največje dolžine, kakor smo to naredili pri nizih, saj je največja velikost določena že vnaprej. Tip `int` lahko shrani pozitivna in negativna števila velikosti največ 2 milijardi.