Funkcije

Ko pišemo daljše programe, se pogosto zgodi, da večrat uporabimo nek podoben kos kode. Da si v takih situacijah olajšamo življenje, imamo *funkcije*. Že v zgodnjih sedemdesetih, ko se je začel razvijati jezik C so vedeli, da so funkcije zelo uporabne, zato je osnovna oblika tega jezika sestavljena iz njih. Res, funckijam se v C-ju ne da izogniti, do zdaj smo brali in pisali podatke s funkcijama `scanf` in `printf`, preverjali dolžino nizov s `strlen` in vse smo napisali v funkciji `main`.
Funkcijo si lahko predstavljamo kot neko napravo. Napravi podamo nekaj *parametrov* (včasih se uporabi tudi beseda *argumentov*), ona nekaj melje in nam potem *vrne* neko vrednost. Poleg tega, da samo vrne neko vrednost, ima lahko funkcija tudi kakšne stranske učinke. Lahko kaj izpiše z uporabo `printf`, ali pa spremeni kakšno kakšno spremenljivko. Včasih so parametri ali vrnjena vrednost nepotrebni, zato lahko napišemo tudi funkcije brez parametrov in funkcije, ki ne vrnejo ničesar.

1 Kako napišemo svojo funkcijo

Najbolj osnovna oblika funkcije je takšna, ki ne sprejme nobenega parametra in ne vrne ničesar. Če hočemo, da ta funkcija ni neuporabna, bo morala imeti kakšen stranski učinek, spodaj bo izpisala `Zdravo`.

Primer

```c
#include <stdio.h>

void funkcija() {
    // koda, ki se bo izvedla, ko pokličemo funkcijo
    printf("Zdravo\n");
}

int main() {

    funkcija(); // pokličemo funkcijo

    return 0;
}
```

- `void` - Posebna beseda, ki pove, da funkcija ne vrne ničesar.
- `funkcija` - Ime za funkcijo, kot pri spremenljivkah bi lahko tu napisali karkoli.
- `()` - Ime funkcije se more končati z (), v te oklepaje postavljamo parametre.
- `{ ... }` - Telo funkcije, v katerem je koda, ki se bo izvedla.

Kot napovedano, imajo lahko funkcije tudi kakšne parametre:

Primer

```c
#include <stdio.h>

void plusi(int n) {
    // izpišemo n plusov
    for (int i = 0; i < n; i++) {
        printf("+");
    }
    printf("\n"); // končamo z znakom za novo vrstico
}

int main() {
    plusi(12); // pokličemo funkcijo z parametrom 12

    return 0;
}
```

Zdaj smo v oklepaje za imenom spremenljivke postavili `int n`. Beseda `int` pove, da je ta parameter celo število, `n` je pa ime za parameter. Ime, ki smo si ga izbrali za parameter, uporabljamo v telesu funkcije, da dostopamo do vrednosti v parametru.
Če je naš parameter seznam, to napišemo tako:

Primer

```c
#include <stdio.h>

void izpisi_prve_tri(int seznam[]) {
    printf("%d %d %d\n", seznam[0], seznam[1], seznam[2]);
}

int main() {
    int stevila[] = {7, 8, 6, 1, 2, 6, 3};
    izpisi_prve_tri(stevila); // izpiše "7 8 6"

    return 0;
}
```

Zdaj pa še funkcije, ki vrnejo kakšno vrednost:

Primer

```c
#include <stdio.h>

int vsota(int a, int b) {
    int a_plus_b = a + b;
    return a_plus_b;
}

int main() {
    int rezultat = vsota(3, 4);
    printf("%d\n", rezultat);

    return 0;
}
```

Zdaj smo v funkcijo dodali ukaz `return`, ki pove programu, da želimo vrniti spremenljivko, ki se pojavi takoj za njim. Spodaj pa vidimo, kako dostopamo do vrednosti, ki jo vrne naša funkcija, nastavimo spremenljivko. Izraz `funkcija(3, 4)` lahko uporabimo kjerkoli, kjer bi lahko uporabili neko spremenljivko. Prav tako lahko za return-om seštevamo števila. Zgornji program bi lahko krajše napisali kot:

Primer

```c
#include <stdio.h>

int vsota(int a, int b) {
    return a + b;
}

int main() {
    printf("%d\n", vsota(3, 4));

    return 0;
}
```

2 Potek funkcije

Kot smo navajeni, se ukazi v telesu funkcije izvajajo po vrsti od zgoraj navzdol. V njej lahko uporabimo vse, kar smo se do zdaj naučili (lahko ustvarjamo nove spremenljivke, if stavke, zanke, ...). Vredno omembe je, da se funkcija konča, ko se izvede prvi `return`.

Primer

```c
#include <stdio.h>

bool je_samoglasnik(char crka) {
    if (crka == 'a' || crka == 'e' || crka == 'i' || crka == 'o' || crka == 'u') {
        return true;
    }
    return false;
}

int main() {
    char c;
    scanf("%c", &c); // prebere eno črko

    if (je_samoglasnik(c)) {
        printf("To je samoglasnik!\n");
    }
    else {
        printf("To pa ni samoglasnik.\n");
    }

    return 0;
}
```

Tu funkcija vrača logično vrednost `bool`, ki je ena izmed `true` ali `false`, uporablja se jo lahko v if stavku. Deluje torej pravilno, saj če bo `crka` enaka a, e, i, o, u, se bo funkcija končala pri return-u v if stavku, sicer se bo pa nadaljevala do `return false;`.

3 Spremenljivke in funkcije

3.1 Globalne in lokalne spremenljivke

Spomnimo se, da lahko spremenljivke definiramo na dva načina. Tistim, ki so definirane na vrhu, izven funkcije rečemo *globalne spremenljivke*, tistim, ki so pa definirane v funkciji pa *lokalne spremenljivke*. Zdaj, ko vemo, kaj so funkcije, lahko povemo razliko med temi tipi spremenljivk.
Globalne spremenljivke so vidne povsod, torej v vseh funkcijah, medtem ko lahko lokalne spremenljivke uporabljamo samo v funkciji, v kateri smo jih definirali.

Primer

```c
#include <stdio.h>

// tu so globalne spremenljivke
int g = 12;

void funkcija() {
    // v tej funkciji lahko dostopamo do g
    printf("%d\n", g);
}

int main() {

    int n = 200;

    funkcija();

    // tu lahko dostopamo do g in n
    printf("%d, %d\n", g, n);

    return 0;
}
```

Seveda, če globalno spremenljivko nekje spremenimo, se bo spremenila tudi za vse druge funkcije.

3.2 Spreminjanje parametrov

Če funkciji podamo neko spremenljivko kot parameter in poskušamo to spremenljivko v funkciji spreminjati, se obnaša na dva različna načina, odvisno od tega, če je spremenljivka seznam.
Če spremenljivka ni seznam (npr. `int`, `char`) se za funkcijo ustvari kopija te spremenljivke. Če jo v funkciji spremenimo, to ne bo spremenilo izvirne spremenljivke, saj v funkciji delamo s kopijo. Če je seznam, bodo pa spremembe na seznam v funkciji vplivale na izvirni seznam, saj se ta ne bo prekopiral.
V spodnjem primeru sta prikazani obe možnosti.

Primer

```c
#include <stdio.h>

void funkcija(int stevilo, int seznam[]) {
    stevilo = 123; // ta sprememba ne vpliva na a v main-u
    seznam[0] = 123; // ta sprememba vlpliva na sez v main-u
}

int main() {
    int a = 10;
    int sez[] = {10};
    funkcija(a, sez);
    // a bo še vedno 10, medtem, ko bo seznam enak {123}

    return 0;
}
```

Sprva ne vidimo smisla, za drugačno obravnavo seznamov, a se izkaže, da je lahko ta lastnost tudi uporabna. Če bi želeli napisati funkcijo, ki sprejeme seznam in njegovo dolžino, potem pa obrne vrstni red elementov tega seznama, lahko to napišemo takole:

Primer

```c
#include <stdio.h>

void obrni(int seznam[], int dolzina) {
    // Obrnemo vrstni red prvih dolzina elementov seznama
    for (int i = 0; i < dolzina / 2; i++) {
        int vmesna = seznam[i];
        seznam[i] = seznam[dolzina-i-1];
        seznam[dolzina-i-1] = vmesna;
    }
}

int main() {
    int n;
    int seznam[1000];

    // preberemo seznam dolžine n
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &seznam[i]);
    }

    obrni(seznam, n);

    // izpišemo obrnjen seznam
    for (int i = 0; i < n; i++) {
        printf("%d\n", seznam[i]);
    }

    return 0;
}
```

Primer vhoda in izhoda

```
5
10 20 30 -3 7 1
```

```
1
7
-3
30
20
10
```

Da je postopek za obračanje seznamov res pravilen, lahko razmislite sami, named tega primera je, da vidimo, kako se pri funkcijah dela s seznami. Seznam, ki ga želimo obdelati prejmemo kot parameter in ga potem spremenimo, ničesar pa ne vrnemo.

4 Rekurzija

Rekurzija je način pisanja programa na način, da funkcija kliče samo sebe. Pogosto lahko zanko nadomestimo z rekurzijo, če nam je tak način programiranja ljubši.
Recimo, da želimo sešteti vsa števila med 1 in nekim n. Z zanko bi to naredili tako, da si pripravimo neko srpemenljivko *vsota* in potem napišemo for zanko od 1 do n in v *vsota* seštejemo vsa ta števila. Pri rekurziji je pa pristop drugačen.

Primer

```
int vsota(int n) {
    // osnovni pogoj
    if (n <= 0) {
        return 0;
    }

    // korak
    return n + vsota(n - 1);
}
```

Najprej napišemo *osnovni pogoj*, ki za n-je manjše ali enake 0 vrne 0. Če ne bi napisali osnovnega pogoja bi šli rekurzivni klici funkcije v neskončnost, `vsota(3)` bi poklicala `vsota(2)`, ki pokliče `vsota(1)`, potem `vsota(0)`, `vsota(-1)`, `vsota(-2)`, ... Po osnovnem pogoju moramo narediti *korak*, ki izračuna `vsota(n)` s pomočjo vrednost funkcije v nekem manjšem številu, v našem primeru kar `vsota(n - 1)`. O rekurziji bi lahko veliko povedali in je za reševanje raznih problemov zelo uporabna, a smo se odločili, da se letos o njej ne bomo poglabljali.

5 Funkcije že vključene v C

5.1 Funkcije iz <stdio.h>

- `sprintf` deluje podobno kot `printf`, le da sprejme niz kot prvi parameter, naslednji parametri so pa enaki kot v `printf`. Namesto da bi izpisala na standardni izhod, ta funkcija izpiše v niz, ki je bil dan kot prvi parameter.
- `sscanf` deluje podobno kot `scanf`, le da sprejme niz kot prvi parameter, naslednji parametri so pa enaki kot v `scanf`. Namesto da bi prebrala iz standardnega vhoda, ta funkcija bere iz niza, ki je bil dan kot prvi parameter.

Primer

```c
#include <stdio.h> // ko napišemo to vrstico,
                   // dobimo dostop do sprintf in sscanf

int main() {
    int n;
    scanf("%d", &n);

    char n_niz[100];
    sprintf(n_niz, "%d", n);
    // zdaj smo spremenili n v niz

    int nazaj_n;
    sscanf(n_niz, "%d", &nazaj_n);
    // zdaj smo iz niza nazaj prebrali nazaj_n
}
```

Če bi zgornji program združili s funkcijo za obračanje seznamov (niz je seznam char-ov) bi dobili precej preprosto rešitev naloge števila v ogledalu (https://putka-rtk.acm.si/tasks/t/addrev/).

5.2 Funkcije iz <string.h>

- `strcpy` kopira nize. Sprejme dva niza in drugega prepiše v prvega.
- `strcat` stika nize, enega za drugim. Sprejme dva niza in drugi niz prepiše na konec prvega tako, da prvi postane oba niza staknjena skupaj.
- `strcmp` primerja nize. Sprejme dva niza, če sta niza enaka vrne število 0, če sta pa različna pa vrne število različno od 0.
- `strlen` že poznamo, sprejme en niz in vrne njegovo dolžino.

Primer

```c
#include <string.h>

int main() {

    char niz1[] = "jabolko";
    char niz2[] = "pomaranca";
    char oba_skupaj[100];

    strcpy(oba_skupaj, niz1); // oba_skupaj je zdaj "jabolko"
    strcat(oba_skupaj, " in "); // oba_skupaj je zdaj "jabolko in "
    strcat(oba_skupaj, niz2); // oba_skupaj je zdaj "jabolko in pomaranca"

    return 0;
}
```