Pogojni stavki: 2. del

1 Nizanje pogojev

Pogosto se srečamo s problemi, kjer je za rešitev potrebno upoštevati več kot en pogoj. V takih primerih želimo združiti več pogojnih stavkov tako, da se nek del kode izvede, če velja prvi pogoj, drugi del kode pa, če prvi pogoj ne velja, velja pa drugi pogoj. Z gnezdenjem stavkov lahko to v kodo vključimo na naslednji način:

```
if (prvi pogoj) {
    // koda, ki se izvede, če velja prvi pogoj
} else {
    if (drugi pogoj) {
        // koda, ki se izvede, če prvi pogoj ne velja,
        // velja pa drugi pogoj
    } else {
        // koda, ki se izvede, če ne veljata ne prvi ne drugi pogoj
    }
}
```

V takem primeru nam je na voljo bližnjica `else if`. Zgornja koda deluje popolnoma enako kot spodnja:

```
if (prvi pogoj) {
    // koda, ki se izvede, če velja prvi pogoj
} else if (drugi pogoj) {
    // koda, ki se izvede, če prvi pogoj ne velja,
    // velja pa drugi pogoj
} else {
    // koda, ki se izvede, če ne veljata ne prvi ne drugi pogoj
}
```

Prednost te bližnjice je, da je naša koda krajša in bolj razumljiva. Stavke `else if` lahko tudi verižimo; enemu `if` stavku lahko sledi poljubno mnogo stavkov `else if`. Pri tem bo računalnik pogoje preverjal po vrsti. Pri prvem veljavnem pogoju se bo ustavil in izvedel kodo v pripadajočih zavitih oklepajih, za čimer ne bo več preverjal pogojev, temveč bo izvajanje nadaljeval za zaključkom vseh nanizanih stavkov.
Kakor nam v osnovnem `if` stavku ni bilo treba pisati dela z `else`, če ga nismo potrebovali, nam ga tudi pri uporabi `else if` ni treba.

Primer

Naslednji program prebere število in pove, če je večje, manjše ali enako 0.

```
#include <stdio.h>

int main() {
    int stevilo;
    scanf("%d", &stevilo);
    if (stevilo < 0) {
        printf("Stevilo je manjse od nic.\n");
    } else if (stevilo == 0) {
        printf("Stevilo je enako 0.\n");
    } else if (stevilo > 0) {
        printf("Stevilo je vecje od 0.\n");
    }
    return 0;
}
```

Primer vhoda in izhoda

```
3
```

```
Stevilo je vecje od 0.
```

Primer

Pogoji se preverjajo po vrsti; izvede se samo koda pri prvem veljavnem pogoju, ne glede na to, koliko pogojev za njim je tudi veljavnih.

```
#include <stdio.h>

int main() {
    int a = 7;
    if (a < 3) {
        printf("a je manjsi od 3.\n");
    } else if (a == 7) {
        printf("a je 7.\n");
    } else if (a == 4) {
        printf("a je 4.\n");
    } else if (a > 1) {
        printf("a je vecji od 1.\n");
    } else {
        printf("Nic od nastetega ne velja.\n");
    }
    return 0;
}
```

Program izpiše le eno stvar - `a je 7.`, kljub temu, da velja tudi `a > 1`.

2 Logični vezniki

Osnovni operatorji za primerjavo pogosto niso dovolj, da izrazimo željen pogoj. Če na primer želimo pogledati, ali je neko število med dvema drugima, tega ne moramo narediti samo z eno primerjavo. Pogoje združujemo s t.i. *logičnimi vezniki*, ki jim včasih pravimo tudi *logični operatorji*. Poznamo tri osnovne veznike:

- `and` oz. `&&` združi dva pogoja tako, da združeni pogoj velja samo v primeru, da veljata oba hkrati.
- `or` oz. `||` združi dva pogoja tako, da združeni pogoj velja v primeru, da velja katerikoli od dveh, ali da veljata oba
- `not` oz. `!` sprejme samo en pogoj. Nov pogoj velja samo takrat, ko originalni pogoj ne velja.

Logična veznika `&&` (*in*) ter `||` (*ali*) lahko predstavimo z resničnostno tabelo.

Tabela 1: Resničnostna tabela za in

Leva vrednost	Desna vrednost	Vrednost veznika
resnica	resnica	resnica
resnica	neresnica	neresnica
neresnica	resnica	neresnica
neresnica	neresnica	neresnica

Tabela 2: Resničnostna tabela za ali

Leva vrednost	Desna vrednost	Vrednost veznika
resnica	resnica	resnica
resnica	neresnica	resnica
neresnica	resnica	resnica
neresnica	neresnica	neresnica

Primer

Če želimo preveriti, ali je dano število med dvema drugima, uporabimo logični veznik `&&`.

```c
#include <stdio.h>

int main() {
    int n;
    scanf("%d", &n);
    if (3 < n && n < 9) {
        printf("n je med 3 in 9.\n");
    }
    return 0;
}
```

Primer vhoda in izhoda

```
5
```

```
n je med 3 in 9.
```

Pogoste napake

V matematiki pogosto napišemo dvojno primerjavo: `a < b < c`. Če nekaj podobnega napišemo v C++ program, se bo le-ta sicer zagnal, vendar ne bo deloval pravilno. Kaj pričakujemo, da se zgodi, če v tako primerjavo zapišemo `3 < 2 < 1`? Kaj pa se dejansko zgodi?

Pri kombiniranju pogojev bodimo previdni glede pravil prednosti. Zanikanje (veznik `not` oz. `!`) ima namreč prednost pred primerjalnimi vezniki (`<`, `==`, ...), ter drugima ločnima veznikoma. Če v takem primeru uporabljamo zanikanje, moramo zanikan izraz postaviti v oklepaje.

Primer

Zanikanje se lahko v večini primerov zapiše tudi na krajši način.

Izraz	Ekvivalenten izraz	Komentar
`!(a == b)`	`a != b`	Če sta `a` in `b` enaka, bo prvi izraz veljaven.
`!(a < b)`	`a >= b`	
`!(a <= b)`	`a > b`	

Primer

Napišimo program, ki preveri, ali je uporabnik vpisal prestopno leto. Leto je prestopno, če je deljivo s 4; razen, če je hkrati deljivo s 100. Izjema so leta, deljiva s 400, ki so prestopna kljub temu, da so deljiva s 100.
Kako te pogoje zapišemo v program? Opazimo, da so leta, deljiva s 400, prestopna ne glede na druga pogoja. Če leto ni deljivo s 400, potem mora biti deljivo s 4 in ne sme biti deljivo s 100. Povedano krajše; leto mora biti deljivo s 400, ali pa s 4 in ne hkrati s 100. Tak pogoj lahko zapišemo z logičnimi vezniki.

```
#include <stdio.h>

int main() {
    int leto;
    scanf("%d", &leto);
    if (leto % 400 == 0 || (leto % 4 == 0 && !(leto % 100 == 0))) {
        printf("Leto je prestopno.\n");
    } else {
        printf("Leto ni prestopno.\n");
    }
    return 0;
}
```