



Mérnöki programozás 5

Szerkesztette: dr.Vass Péter Tamás

Vezérlési szerkezetek

A vezérlési szerkezetek kiemelten fontos részei a C programozási nyelvnek. Alkalmazásukkal határozhatjuk meg a program utasításainak végrehajtási sorrendjét.

A vezérlési szerkezeteket felépítő szintaktikai elemek:

- kulcsszavak (pl. if, else, while stb.),
- utasításblokkot határoló kapcsos zárójelpárok { },
- a vezérléshez kapcsolódó feltételeket és egyéb műveleteket tartalmazó zárójel párok (),
- utasításlezáró jel ;

A vezérlési szerkezetek alkalmazásának két fő célja:

- feltételes elágazások és
- ciklusok megvalósítása a programban.

Vezérlési szerkezetek

Feltételes elágazások megvalósítására szolgáló vezérlési szerkezetek

I. Utasítás, vagy utasítások feltételes végrehajtása

`if (kifejezés) utasítás`

vagy

```
if (kifejezés){  
    utasítások  
}
```

Csak a kifejezésben megfogalmazott feltétel teljesülése esetén kerül végrehajtásra az utasítás (ill. az utasításblokkba foglalt utasítások). Egyébként pedig a program végrehajtása átugorja az utasítást, ill. utasításblokkot, és a szerkezet utáni első utasítás kiértékelése következik.

Igaznak bizonyul a feltétel, ha a kifejezés értéke nem nulla. Csak akkor hamis a feltétel, ha a kifejezés értéke nulla.

Vezérlési szerkezetek

Példák:

```
if (a > b) x++;
```

```
if (a > b){  
    x++;  
    y = pow(2, x);  
}
```

2. Kétirányú elágazás megvalósítása

```
if (kifejezés)  
    utasítás1  
else  
    utasítás2
```

A kifejezésben megfogalmazott feltétel teljesülése esetén az if ághoz tartozó utasítás vagy utasításblokk kerül végrehajtásra.

Ellenkező esetben pedig az else ághoz tartozó utasítás vagy utasításblokk tartalma valósul meg.

Vezérlési szerkezetek

Példák:

```
if (a > b)  
    x++;
```

```
else  
    x--;
```

```
if (a > b){  
    x++;  
    y = pow(2, x);  
}  
else{  
    x--;  
    y = pow(2, x);  
}
```

Vezérlési szerkezetek

3. Többirányú elágazás megvalósítása

```
if (kifejezés1)
    utasítás1
else if (kifejezés2)
    utasítás2
.....
else
    utasításn
```

Ha az **if** ághoz tartozó kifejezésben megfogalmazott feltétel igaz, akkor az **if** ág utasítása vagy utasításblokkja kerül végrehajtásra.

Ha nem teljesül az **if** ág feltétele, akkor a soron következő **else if** ág feltétele kerül kiértékelésre.

Egymás után több **else if** ág is beiktatható a szerkezetbe. Mindegyik saját feltétellel rendelkezik.

Ha valamelyik feltétel teljesül, akkor a hozzá tartozó ág utasítása, ill. utasításai végrehajtódnak, és a többi ág már nem kerül kiértékelésre.

Ha egyik feltétel sem teljesül, akkor az **else** ág tartalmát hajtja végre a program.

Vezérlési szerkezetek

Példa:

```
if (x < 0)
    x++;
else if (x > 0)
    x--;
else
    ;
```

Az önmagában álló utasításlezáró jel egy *üres utasítást* jelent. Ilyenkor semmilyen művelet nem kerül végrehajtásra az adott ágon. Ha több utasítást tartalmaz valamelyik ág, akkor utasításblokkba kell foglalni { }.

Vezérlési szerkezetek

4. Többirányú elágazás switch – case szerkezettel

```
switch (kifejezés){  
    case konstans kifejezés1:  
        utasítások1  
    case konstans kifejezés2:  
        utasítások2  
    .....  
    default:  
        utasításokn  
}
```

A **switch – case** vezérlési szerkezet olyankor használható, amikor a **switch** kulcsszó utáni kifejezés értékétől függően akarunk más – más utasításokat végrehajtani. Az egyes **case** címkék utáni konstans kifejezések értékeivel hasonlítja össze a program a **switch** kifejezésének értékét. Ha egyezést talál, akkor az adott **case** ághoz tartozó utasítások kerülnek végrehajtásra. Ha egyik **case** címkéhez tartozó konstans kifejezés értéke sem bizonyul egyenlőnek a **switch** kifejezés értékével, akkor a **default** címkével azonosított ág utasításai kerülnek végrehajtásra.

Vezérlési szerkezetek

```
#include <stdio.h>
main(){
    int n;
    printf("Adjon meg egy 10-nel kisebb pozitív számot: ");
    scanf("%d", &n);
    if (n>10 || n<1){
        printf("Helytelen adat!\n");
        return -1;
    }
    switch (n){
        case 1:
            printf("A megadott szám 1.\n");
            break;
        case 2:
            printf("A megadott szám 2.\n");
            break;
        case 3:
            printf("A megadott szám 3.\n");
            break;
        default:
            printf("A megadott szám nagyobb, mint 3.\n");
            break;
    }
    printf("A program vége.\n");
}
```

Vezérlési szerkezetek

4. Többirányú elágazás switch – case szerkezettel

Az egyes **case** címkék utasítása, ill. utasításai után alkalmazott **break** utasítások azt eredményezik, hogy amennyiben egy adott konstans kifejezés értéke egyenlőnek bizonyul a **switch** kifejezésének értékével, akkor a megfelelő **case** címke ágához tartozó utasítás(ok) végrehajtása után a további **case** címkék konstans kifejezéseire vonatkozóan már ne végezzen vizsgálatokat a program.

A **break** utasítása nélkül, minden egyes **case** címkéhez rendelt konstans kifejezés értékét összehasonlítja a program a **switch** kifejezés értékével.

Természetesen csak annak az ágnak az utasításai kerülnek végrehajtásra, amire a kifejezések egyenlősége teljesül.

A **case** címkékhez tartozó utasításokat nem kell blokkba foglalni kapcsolószerűjelekkel, de a **:** kötelező a címke kifejezése után.

Vezérlési szerkezetek

5. Előtesztelő while ciklus

```
while (kifejezés)  
    utasítás
```

vagy

```
while (kifejezés){  
    utasítások  
}
```

Az előzőekben már megismertük!

6. Háultesztelő do - while ciklus

```
do  
    utasítás  
while (kifejezés);
```

vagy

```
do {  
    utasítások  
} while (kifejezés);
```

Az előzőekben már megismertük! Ne felejtsük el kitenni a ;-t a while kifejezése után!

Vezérlési szerkezetek

7. for ciklus

```
for (inicializáló_kif; feltétel; léptető_kif)  
    utasítás
```

vagy

```
for (inicializáló_kif; feltétel; léptető_kif){  
    utasítások  
}
```

A **for** után következő inicializáló kifejezésben kezdő értéket rendelünk hozzá a ciklusváltozóhoz (pl. $i = 0$).

A következő kifejezés a ciklusváltozó lehetséges felső vagy alsó értékhatárára vonatkozó feltételt tartalmaz (pl. $i \leq 8$).

A léptető kifejezéssel növelhetjük vagy csökkenthetjük a ciklusváltozó értékét a ciklus mag minden egyes végrehajtása után (pl. $i=i+2$).

Minden for ciklus megvalósítható while ciklussal, de fordítva ez nem igaz.

Vezérlési szerkezetek

break utasítás

A **break** utasítást nem csak a **switch – case** szerkezetnél lehet alkalmazni, hanem a ciklusok esetében is.

Ha a **break** utasítást valamilyen feltétel teljesüléséhez kötve alkalmazzuk egy ciklus magon belül, akkor lehetővé válik a kiugrás a ciklusból mielőtt még a ciklus feltétel szerinti ismétlődés be nem fejeződne.

continue utasítás

A **continue** utasítás szintén alkalmazható a ciklus blokkon belül. Ezt is valamilyen belső feltétel teljesüléséhez kötve használjuk. Hatása annyiban tér el a **break**-től, hogy a **continue** nem eredményezi a ciklus idő előtti leállítását, csak az utána következő utasítások végrehajtását akadályozza meg az adott iteráción belül. A ciklus végrehajtása a következő iterációval folytatódik.

Vezérlési szerkezetek

return utasítás

A **return** utasítás befejezi az éppen végrehajtás alatt álló függvény futását.

Ezután a program vezérlése visszakerül ahhoz a függvényhez (*hívó függvény*), amelyik meghívta a már leállt függvényt (*hívott függvény*).

Ha a **main** függvényben alkalmazzuk a **return** utasítást, akkor a program futása leáll.

A **return** utasítás után kifejezés is írható, melynek értéke lesz a függvény kimeneti értéke (*visszatérési értéke*).

Természetesen a függvény definíciójában úgy kell megadni a kimeneti paraméter típusát, hogy a **return** után következő kifejezés értékének típusával összhangban legyen.

Ha nem kívánunk visszatérési értéket rendelni a függvényünkhöz, akkor definiáljuk **void** (üres) típusú kimeneti paraméterrel.

Vezérlési szerkezetek

```
#include <stdio.h>
```

```
main(){
```

```
/*A program beolvas egy sort, és megszámlolja a  
karaktereket*/
```

```
    int nc=0;
```

```
    char c;
```

```
    printf("Gepeljen be egy sort majd nyomjon \  
enter-t!\n");
```

```
    do{
```

```
        c = getchar();
```

```
        nc++;
```

```
        putchar(c);
```

```
    }while ( c != '\n');
```

```
    printf("A karakterek szama: %d\n", --nc);
```

```
}
```

Vezérlési szerkezetek

```
#include <stdio.h>
#include <ctype.h>

main(){
    int nc=0;
    char c;

    printf("Gepeljen be egy sort majd nyomjon \
enter-t!\n");
    do{
        c = getchar();
        if (isspace(c))
            break;
        nc++;
        putchar(c);
    }while ( c != '\n');
    printf("\nAz elso ures karakter elotti \
karakterek szama: %d\n", nc);
}
```


Vezérlési szerkezetek

```
#include <stdio.h>
#include <ctype.h>

main(){
    int nc=0;
    char c;

    printf("Gepeljen be egy sort majd nyomjon "\
           "enter-t!\n");
    do{
        c = getchar();
        if (isspace(c))
            continue;
        nc++;
        putchar(c);
    }while ( c != '\n');
    printf("\nA nem ures karakterek "\
           "szama: %d\n", nc);
}
```

Vezérlési szerkezetek

Magyarázat

A `getchar()` és a `putchar()` függvények a szabványos bemeneti és kiviteli függvénykönyvtár részei (hasonlóan a `printf` és a `scanf` függvényekhez).

```
c = getchar();
```

A `getchar` függvény beolvas egy karaktert a parancssorból, és a karakter kódjával tér vissza. A karakter kódját a fenti utasítás szerint a `c` változóban tároljuk el, amit előzőleg `char` típusal deklaráltunk.

```
putchar(c);
```

A `putchar` függvénynek egyetlen bemeneti paramétere van, amelyben egy karakter kódját képes fogadni. A feladata, hogy a megadott kódú karaktert megjelenítse a parancssorban.

Vezérlési szerkezetek

Magyarázat

`isspace(c)`

Az `isspace` is egy szabványos könyvtári függvény, amelynek deklarációját a `ctype.h` fejlécfájl tartalmazza.

Ez a fejlécfájl tartalmazza a karakteres vizsgálatokra szolgáló függvények deklarációit.

A karakteres vizsgálatokra használható könyvtári függvényeknek egyetlen karakter típusú bemeneti paramétere van. Ebben fogadják annak a karakternek a kódját, amelyre nézve valamilyen feltétel teljesülését vizsgálják.

Ha a feltétel nem teljesül, akkor 0, azaz logikai hamis értékkel térnek vissza.

Ha a feltétel teljesül, akkor nem 0, azaz logikai szempontból igaznak minősülő értéket adnak a kimenetükön.

Vezérlési szerkezetek

Magyarázat

`isspace(c)`

A fenti függvényhíváskor az `isspace` függvény a `c` változóban tárolt karakter kódját kapja meg a bemenetén, és megvizsgálja, hogy a karakter megegyezik-e valamelyik üres karakterrel (szóköz, horizontális vagy vertikális tabulátor, újsor, lapdobás).

Ha egyezést talál, akkor nem 0, azaz igaz logikai értékkel tér vissza a hívó függvényhez (ami most maga a `main` függvény).

Ha nem talál egyezést, azaz a vizsgált karakter nem üres karakter, akkor 0 (logikai hamis) lesz a visszatérési értéke.

A példákban a függvényhívás egy `if` szerkezet feltételében helyezkedik el. Ez azt eredményezi, hogy az `if` –hez tartozó ág utasításai csak akkor kerülnek végrehajtásra, ha a vizsgált karakter üres karakternek minősül.

Vezérlési szerkezetek

Néhány karaktervizsgáló könyvtári függvény

<code>islower(c)</code>	értéke igaz, ha c egy kisbetű
<code>isupper(c)</code>	értéke igaz, ha c egy nagybetű
<code>isalpha(c)</code>	értéke igaz, ha c egy betű
<code>isdigit(c)</code>	értéke igaz, ha c egy decimális számjegy
<code>isprint(c)</code>	értéke igaz, ha c egy nyomtatható karakter, beleértve a szóközt is
<code>ispunct(c)</code>	értéke igaz, ha c egy nyomtatható karakter, de nem betű, számjegy, vagy szóköz
<code>isctrl(c)</code>	értéke igaz, ha c egy vezérlőkarakter

A ctype.h-ban deklarált karakter átalakító függvények

<code>int tolower(c)</code>	a betűt képviselő c karaktert kisbetűvé alakítja, és a kisbetű karakterkódjával tér vissza
<code>int toupper(c)</code>	a betűt képviselő c karaktert nagybetűvé alakítja, és a nagybetű karakterkódjával tér vissza