Vezérlési szerkezetek

Szelekciós utasítások: if, else, switch

If utasítás

Segítségével valamely utasítások végrehajtását valamely feltétel teljesülése esetén végezzük el. Az **if** utasítás segítségével valamely tevékenység (*utasítás*) végrehajtását egy *kifejezés* (feltétel) értékétől tehetjük függővé. Az **if** alábbi formájában az *utasítás* csak akkor hajtódik végre, ha a *kifejezés* értéke nem nulla (igaz):

```
if(kifejezés){
      utasítás
 }
```

else

Az **if-else** szerkezet használatával arra az esetre is megadhatunk egy tevékenységet, amikor a *kifejezés* (feltétel) értéke zérus (hamis):

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

else if

Ez az szerkezet egymásba ágyazott if utasítások egy gyakran használt formája, amikor az else ágakban szerepel az újabb if utasítás:

```
if (kifejezés){
    utasítás
} else if (kifejezés){
    utasítás
} else if (kifejezés){
    utasítás
} else {
    utasítás
}
```

A zárójelben lévő kifejezés egy logikai kifejezést takar. Ezt a program a szelekciós vezérlőszerkezet végrehajtásakor kiértékeli, és a kiértékelés eredménye vagy igaz vagy hamis érték. Egy logikai kifejezés logikai változókból/értékekből és logikai operátorokból állhat. A C nyelvben nincs külön

logikai típus, egész típusokban (int, char) tárolhatunk logikai értékeket: a 0 jelenti a hamisat, a nem nulla pedig az igazat (ez gyakran 1, DE fordítója válogatja). Logikai értékek keletkezhetnek relációs operátorok használatával, ezek két érték összehasonlítására használhatók.

Relációs operátorok:
a és *b* értékek: változók, konstansok, valamilyen művelet vagy függvény eredményei, vagy literálok (literálnak nevezzük a helyben definiált adatot, pl. 5, vagy 'A')

Matematikai alak	***C kifejezés***	***Jelentés***
a>b	a>b	a nagyobb mint b
a>=b	a>=b	a nagyobb vagy egyenlő mint b
a<b	a<b	a kisebb mint b
a<=b	a<=b	a kisebb vagy egyenlő mint b
a=b	a==b	a egyenlő b-vel
a≠b	a!=b	a nem egyenlő b-vel

Logikai operátorok:
a és *b* logikai értékek: változók, konstansok, valamilyen művelet vagy függvény eredményei, vagy literálok.

!a - a kifejezés értéke NEM *a*, tehát akkor lesz igaz, ha *a* hamis volt
a && b - a kifejezés értéke *a* ÉS *b*, tehát akkor lesz igaz, ha *a* és *b* is igaz volt
a || b - a kifejezés értéke *a* VAGY *b*, tehát akkor lesz igaz, ha *a* és *b* közül legalább az egyik igaz volt

***Feladat**: Írj programot, mely egy megadott egész számról eldönti, hogy az páros, vagy páratlan!*

```c
#include <stdio.h>
int main() {
    int x;

    printf("Kérek egy egész számot:");
    scanf("%d", &x);

    if (x%2 == 0)
        printf("A megadott szám páros.\n");
    else
        printf("A megadott szám páratlan.\n");

    return 0;
}
```

***Feladat**: Módosítsuk most a programot úgy, hogy két egész számot kérjen be a program majd írja ki, hogy az első szám osztható-e a másodikkal.*

***Feladat**: Próbáljuk ki, mi történik, ha a második szám 0! Javítsuk a programot!*

***Feladat**: Használjunk többszörös szelekciót!*

***Feladat**: Írjuk meg if nélkül a fenti programot.*

Feltételes kifejezések

Az **if-else** helyett **feltételes kifejezések** is használhatók. Ezzel tömörebb formában fogalmazható meg ugyanaz, sőt, egy adott helyre behelyettesítendő érték kiválasztására is használható.

```
feltétel ? művelet ha igaz : művelet ha hamis;

pl.:
(a>b) ? a : b;
```

***Feladat**: Írjuk ki egyetlen printf segítségével, hogy egy szám páros vagy páratlan-e!*

```c
#include <stdio.h>

int main() {
	int x;

	printf("Kérek egy egész számot: ");
	scanf("%d", &x);

	printf("A szám %s.\n", (x%2 == 0) ? "páros" : "páratlan");

	return 0;
}
```

***Feladat**: Írjuk ki egyetlen printf használatával, hogy egy szám osztója-e egy másiknak.*

A feltételes kifejezéseket egymásba is ágyazhatjuk!

```c
#include <stdio.h>

int main () {
  int x;

  printf("Kérek egy egész számot: ");
  scanf("%d", &x);
  printf("Kérek egy másik számot: ");
  scanf("%d", &y);

  printf("Osztoja-e %d-nek %d?. %s\n", x, y,
    (y==0) ? "A kerdes ertelmetlen!" :
    ((x%y==0) ? "Igen, osztója." : "Nem, nem osztója.")
  );
}
```

Esetkiválasztásos szelekciós vezérlés

A **switch** utasítás többirányú programelágaztatást tesz lehetővé olyan esetekben, amikor egy egész kifejezés értékét több konstans értékkel kell összehasonlítanuk. Az utasítás álatlános alakja:

```c
switch ( kifejezés ) {
    case címke : muveletek; break;
    case címke : muveletek; break;
        ...
    case címke : muveletek; break;
    default : muveletek; break;
}
```

A **switch** utasítás először kiértékeli a *kifejezést*, majd átadja a vezértlést arra a **case** címkére (esetre), amelyben a címke értéke megegyezik a kiértékelt *kifejezés* értékével - a futás ettől a ponttól folytatódik. Amennyiben egyik **case** sem egyezik meg a *kifejezés* értékével, a program futása a **default** címkével megjelölt utasítással folytatódik. Ha nem használunk **default** címkét, akkor a vezérlés a **switch** utasítás blokkját záró } utáni utasításra adódik. A **break** utasítás kiugrasztja a switchből a vezérlést, ha kihagyjuk, az a következő címkére kerül. Ez használható pl. olyankor, ha több címkéhez ugyanazokat a műveleteket kell végrehajtani.

***Feladat**: írjunk egy függvényt, ami egy "x" egész számot kap paraméterként és kiírja, hogy a hét x. napja milyen nap.*

```
void hetnapja_if (short int x) {
  if (x==1) {
    printf("Hétfo\n");
  } else if (x==2) {
    printf("Kedd\n");
  } else if (x==3) {
    printf("Szerda\n");
  } else if (x==4) {
    printf("Csütörtök\n");
  } else if (x==5) {
    printf("Péntek\n");
  } else if (x==6) {
    printf("Szombat\n");
  } else if (x==7) {
    printf("Vasárnap\n");
  } else
    printf("Hiba! x értéke legalább 1 és legfeljebb 7 lehet!\n");}
```

***Feladat**: Írjuk meg ugyanezt az fgv-t switch használatával.*

```
void hetnapja_switch (short int x) {
  switch (x) {
    case 1:
      printf("Hétfo\n");
      break;
    case 2:
      printf("Kedd\n");
      break;
    case 3:
      printf("Szerda\n");
      break;
    case 4:
      printf("Csütörtök\n");
      break;
    case 5:
      printf("Péntek\n");
      break;
    case 6:
      printf("Szombat\n");
      break;
    case 7:
      printf("Vasárnap\n");
      break;
    default:
      printf("Hiba! x értéke legalább 1 és legfeljebb 7 lehet!\n");
  }}
```

Kérdés: Hogyan kéne módosítani a függvényt akkor, ha számok helyett a napok kezdőbetűit szeretnénk használni?

Kezdőfeltételes ismétléses vezérlés

Más néven előltesztelős ciklus, vagy **while**-ciklus.
A **while** ciklus mindaddig ismétli a hozzá tartozó *utasítást* (a ciklus törzsét), amíg a vizsgált *kifejezés* (vezérlőfeltétel) értéke igaz (nem nulla). A vizsgálat mindig megelőzi az *utasítás* végrehajtását. Általános alakja:

```
while (kifejezés){
    utasítás
}
```

A **switch**-nél látott **break** utasítás ciklusokból való "kitörésre" is alkalmazható. Így írhatunk olyan (esetleg egyébként végtelen) ciklust, amelyben egy, vagy több helyen egy **if** segítségével megvizsgálunk egy feltételt, majd az alapján (esetleg néhány művelet végrehajtása után) kilépünk a ciklusból. Ezt hívjuk **hurok ismétléses vezérlésnek**.

***Feladat**: írjunk egy programot, ami kiírja 1-től 10-ig számokat.*

```
#include <stdio.h>

int main() {
  int i;

  i=1;
  while (i<=10) {
    printf("%d\n", i++);
  }

  return 0;
}
```

***Feladat**: Írjunk olyan programot, ami addig kér be számokat a billentyűzetről, amíg a beírt szám nem 0! (0 az adott végjel)*

***Feladat**: Módosítsuk a programot úgy, hogy végeredményként írja ki a beírt számok összegét!*

Végfeltételes ismétléses vezérlés

Más néven hátultesztelős ciklus, vagy **do-while**-ciklus.
A **do-while** utasításban a ciklus törzsét képező utasítás végrehajtása után kerül sor a tesztelésre. Így a ciklus törzse legalább egyszer mindig végrehajtódik. Általános alakja:

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

A **do-while** ciklus futása során mindig először végrehajtódik az *utasítás* és ezt követően értékelődik ki a *kifejezés*. Amennyiben a *kifejezés* értéke igaz (nem nulla), akkor új iteráció kezdődik (azaz újra lefut a ciklus), míg hamis (0) érték esetén a ciklus befejezi működését.

***Feladat**: Írjunk egy olyan programot 'do-while' ciklus segítségével, ami 0 végjelig kér be számokat, majd kírja azok összegét. A ciklusban ne szerepeljen a 'break' utasítás.*

```
int main() {
  int x;
  int osszeg=0;

  do {
    printf("Kérek egy számot (kilépéshez: 0):");
    scanf("%d", &x);
    osszeg+=x;
  } while (x!=0);

  printf("A számok összege: %d\n", osszeg);
}
```

Miért tehetjük meg, hogy a végjelet is feldolgozzuk?

Számlálásos ismétléses vezérlés

Más nevén **for** ciklus.

A **for** utasítást általában akkor használjuk, ha a ciklusmagban megadott utasítást adott számszor kívánjuk végrehajtani. Általános alakja:

```
for (kezdoérték_kifejezés ; feltétel_kifejezés ; lépteto_kifejezés){
      utasítás
}
```

A ciklusban van egy változó (a ciklusváltozó, vagy számláló), amit egy kezdőértékből kiindulva, folyamatosan növelünk vagy csökkentünk egy végértékig, és minden ilyen körben végrehajtunk néhány műveletet. A műveletekben a ciklusváltozó aktuális értéke is felhasználható.

A *kezdőérték_kifejezés*-sel állítjuk be a ciklusváltozó kezdőértékét. A *léptető_kifejezés*-sel növeljük vagy csökkentjük a ciklusváltozó értékét tetszés szerint. A *feltétel_kifejezés*-t pedig minden egyes iterációban ellenőrizzük. A ciklus addig fut amíg ez a feltétel teljesül. Mivel a feltételünk akkor nem fog már teljesülni, amikor a ciklusváltozó elér egy bizonyos értéket, ezért jól befolyásolható, hogy a ciklus hányszor fusson le.

```
A++;     =>     A = A + 1;   (avagy A += 1;)
A--;     =>     A = A - 1;   (avagy A -= 1;)

Postfix és prefix alakban is használhatóak, jelentésük kicsit különböző:

B = A++;     =>     1. B = A;
                    2. A = A + 1;
Tehát B A korábbi értékét kapja.

B = ++A;     =>     1. A = A + 1;
                    2. B = A;
Tehát B A megnövelt értékét kapja meg.

Ugyanez érvényes a -- operátorra is.
```

***Feladat**: Írjunk egy programot, ami összeszorozza 1-10-ig a számokat.*

```
int main() {
  int i;
  int szorzat;

  for (i=1, szorzat=1; i<=10; ++i) {
    szorzat*=i;
  }

  printf("A számok szorzata: %d\n", szorzat);
}
```

Feladat: Hogyan nézne ki ugyanez a program while ciklussal?

Feladat: *Módosítsuk a for ciklust úgy, hogy csak minden 3-mal osztható számot szorozzon össze!*

Feladat*: Próbáljuk ki mit csinál az alábbi for ciklus:*

```
int i,j,out;
 for (i=1, j=100, out=0; i<=10; i++, j--)
    out+=i*j;
```

Feladat*: Módosítsuk a ciklusmagot úgy, hogy egy printf segítségével kiírjuk az i,j és out aktuális értékét.*