4. Vezérl szerkezetek

Az elzekben említett program-alkotórészek fabatkát se érnek vezérl szerkezetek nélkül, amelyek alapveten döntésképessé és ciklikussá teszik a programjainkat, ezért kétféle vezérl szerkezettel találkozhatunk általában:

- **elágazások**, amelyek egy döntés eredményeképpen végrehajtanak programrészeket
- **ismétlések**, amelyek egy újra-és-újra kiértékelt döntés eredményeképpen újra-és-újra végrehajtanak programrészeket

Láthatjuk, hogy mindkét fajta vezérl szerkezet függ egy feltételtl, amely szinte kivétel nélkül egy eldöntend állítás, amelyre egyértelmen igazat vagy hamisat kell eredményül adnunk.

4.1. Egyágú elágazás (if)

Az egyik legegyszerbb vezérl szerkezet, amely a feltételtl függen végrehajtja az t követ utasítást:

Java

```
int pénz = 200;
int sörÁra = 120;
if (pénz >= sörÁra)
  System.out.println("Vegyünk egy sört és igyuk meg!");
```

Mint láthatjuk, a feltételes elágazás egy **if** kulcsszóval kezddik, amelyet két kerek zárójel közötti feltétel követ, s végül egy darab utasítással zárul. A programot futtatva minden esetben azt kapjuk, hogy "*Vegyünk egy sört és igyuk meg!*", hiszen a kiinduló feltétel szerint mindig több pénzünk van, mint amennyibe egy üveg sör kerül, de ha a *pénz* változó értékét kisebbre vesszük, mint a sör ára, akkor a feltétel már hamis lesz, ezért nem kerül kiírásra az említett szöveg. Gyakori hiba, hogy az *if* szerkezetet több utasítás is követi, amelyrl úgy gondoljuk, hogy egybe tartoznak, csak ezt a számítógéppel nem beszéltük meg:

Java

```
int pénz = 200;
int sörÁra = 120;
if (pénz >= sörÁra)
System.out.println("Vegyünk egy sört...");
System.out.println("...és igyuk meg!");
```

A várt eredmény nem marad el, hiszen kikerül a képernyre az ismert szöveg, amely abban a hitben hagy minket: a program megfelelen mködik, ám egy tesztelés nem tesztelés, nézzük meg, mi történik, ha nincs elég pénzünk sörre:

Java

```
int pénz = 100;
int sörÁra = 120;
if (pénz >= sörÁra)
System.out.println("Vegyünk egy sört...");
System.out.println("...és igyuk meg!");
```

Az eredmény egy árva *"...és igyuk meg!"*, mert sörre bizony nem volt pénzünk. A megoldás egyszer, minden *if* szerkezetet blokkutasítással kövessünk, és a blokkon belül írjuk meg a feltételtl függ programrészt – ha egy utasításról van szó, akkor is, inkább legyen hosszabb a program, mint hibás:

Java

```
int pénz = 100;
int sörÁra = 120;
if (pénz >= sörÁra)
{
  System.out.println("Vegyünk egy sört...");
  System.out.println("...és igyuk meg!");
}
```

A várakozásainknak megfelelen ez a program már nem ír ki semmit, hiszen nincs pénzünk sörre, ezért bvítsük ki a programot egy olyan feltételes szerkezettel, ahol a feltétel pont a fenti ellentéte:

Java

```
int pénz = 100;
int sörÁra = 120;
if (pénz >= sörÁra)
{
  System.out.println("Vegyünk egy sört...");
  System.out.println("...és igyuk meg!");
}
if (pénz < sörÁra)
{
  System.out.println("Sajnos nincs pénzünk sörre...");
}
```

Ebben az esetben a program kimentén a *"Sajnos nincs pénzünk sörre..."* szöveg olvasható.

4.2. Kétágú elágazás (if-else)

A hibalehetségek csökkentése a programozók elemi érdeke, ezért találták ki régesrég az **if-else** szerkezetet, amely lehetvé teszi, hogy a programunk tegyen valamit a feltétel fennállása esetén (*igaz ág*), illetve tegyen másvalamit akkor, ha a feltétel nem teljesül (*hamis ág*), így nem kell két feltételt karbantartani, s a program is rövidebb lesz:

Java

```
int pénz = 100;
int sörÁra = 120;
if (pénz >= sörÁra)
{
  System.out.println("Vegyünk egy sört...");
  System.out.println("...és igyuk meg!");
} else
{
  System.out.println("Sajnos nincs pénzünk sörre...");
}
```

Mint láthatjuk, a módosítás mindössze annyi, hogy a második *if* és a feltétel helyére egy *else* került.

4.3. Többágú elágazások

Sok esetben szükség van többágú elágazásra, mivel a való életben is gyakran kell egy bekövetkezett tényre több lehetség közül választani, gondoljunk csak arra az esetre, amikor sört akarunk venni a sarki kisboltban, de nincs mindig tele a pénztálcánk. Ekkor az éppen aktuális anyagi lehetségeink határozzák meg, hogy milyen sört tudunk megvásárolni.

4.3.1. Egymásba ágyazott if-else

Egymásba ágyazott kétágú elágazásokból tudunk építeni többágú elágazást, ekkor a hamis ágba újabb elágazást tudunk tenni, amely sok ág esetén kellen átláthatatanná teheti a programunkat:

Java

```
int pénz = 100;
if (pénz >= 500)
{
  System.out.println("Igyunk egy Guinness sört!");
} else
{
  if (pénz >= 300)
  {
    System.out.println("Igyunk egy Leffe sört!");
  } else
  {
    if (pénz >= 100)
    {
      System.out.println("Igyunk egy Soproni sört!");
    } else
    {
      System.out.println("Sajnos nincs pénzünk sörre... :(");
    }
  }
}
System.out.println("Menjünk haza...");
```

A program tartalmaz több - egymásba ágyazott - *if-else* szerkezetet, amelyeknél látnunk kell egy döntési sorozatot. Ha a fenti feltételek szerint 100 forintból akarunk sört inni, akkor a kiértékeljük a *pénz >= 500* feltételt, amely nyilvánvalóan hamis lesz, ezért nem tudunk Guinness sört inni, kénytelenek vagyunk a hamis ágon továbblépni. Az els hamis ágban egy újabb feltételt kell kiértékelni: *pénz >= 300*, amely szintén hamis lesz, ezért Leffe sört sem tudunk inni. Továbbmegyünk a második hamis ágon, ahol meglátjuk az utolsó feltételt, amelynek már megfelel a pénzmagunk, így tudunk inni egy Sopronit. Észre kell vennünk, hogy bármelyik feltétel teljesülése esetén - a hamis ág kihagyása miatt - a program végrehajtása az els hamis ág után folytatódik (*menjünk haza...*).

4.3.2. Egymásba ágyazott if-else-if

Az egymásba ágyazott *if-else* szerkezetek mindig átírhatóak egy *if-else-if* szerkezetre, amely sokkal áttekinthetbb, mivel látszólag mellzi az egymásba ágyazást:

Java

```
int pénz = 100;
if (pénz >= 500)
{
  System.out.println("Igyunk egy Guinness sört!");
} else if (pénz >= 300)
{
  System.out.println("Igyunk egy Leffe sört!");
} else if (pénz >= 100)
{
  System.out.println("Igyunk egy Soproni sört!");
} else
{
  System.out.println("Sajnos nincs pénzünk sörre... :(");
}
System.out.println("Menjünk haza...");
```

Észre kell vennünk, hogy az *else* utasítások után lespóroltunk egy blokk utasítást, mivel a blokkban csak egy *if* utasítás volt. Ezzel az apró módosítással átláthatóbbá tettük a program mködését, hiszen fentrl lefelé olvashatjuk a feltételeket és a feltételekhez tartozó utasításokat, illetve az utolsó *else* után azt az utasítást, amely akkor hajtódik végre, ha egyik feltétel se teljesült. Ez az utolsó *else* ág egyébként elhagyható, ha nincs rá szükségünk.

4.3.3. Kapcsoló (switch-case-default)

Ha primitív típus meghatározott értéke alapján szeretnénk különféle dolgokat végezni, akkor egy hosszabb *if-else-if* szerkezet helyett használjuk inkább a **s witch** utasítást:

Java

```
int pénz = 300;
int sörÁra = 120;
int sörökSzáma = pénz / sörÁra;
switch (sörökSzáma)
{
  case 0:
    System.out.println("Nincs sör, nincs mit inni.");
    break;
  case 1:
    System.out.println("Egy sör nem sör.");
    break;
  case 2:
    System.out.println("Két sör fél egészség.");
    break;
  case 3:
    System.out.println("Három sör jó kezdés.");
    break;
  case 4:
    System.out.println("Négy sör jó, két sör rossz.");
    break;
  default:
    System.out.println("Sok sör soha nem árt.");
}
```

Ahogy a példában látjuk, döntés a *sörökSzáma* változó értékétl függ, ha a változó értéke megegyezik valamelyik *case*ág értékével, akkor az az ág végrehajtásra kerül. A várt mködés szerint a program az alábbit írja ki:

Eredmény

```
Két sör fél egészség.
```

Ha egyik ág se hajtódna végre, akkor a *default* ágra kerül a vezérlés, ha nincs *default* ág, akkor a program végrehajtása a *switch* utasítás végétl folytatódik. A *switch* mködési sajátossága, hogy az els egyezségtl kezdve az összes utasítást végrehajtja a blokk végéig, ezért minden ágat egy *break* utasítás zár, amely hatására a program végrehajtása a *switch* utasítás után folytatódik. Gyakori programozói hiba a *break* elhagyása, amely okán másképp mködik a programunk, próbáljuk ki a fenti programot *brake* utasítások nélkül:

Eredmény break nélkül

```
Két sör fél egészség.
Három sör jó kezdés.
Négy sör jó, két sör rossz.
Sok sör soha nem árt.
```

Mivel két sörre volt pénzünk, a két sört tartalmazó ágtól kezdve az összes utasítás végrehajtódik.

4.4.Ciklusok

Sok esetben szükséges egy-egy programrészletet megismételni, gondoljunk csak arra, ha bemegyünk egy kocsmába egy ezressel és addig szeretnénk sörözni, amíg el nem fogy a pénzünk.

4.4.1. Ell tesztel ciklus (while)

Hasznos dolog még a sörözés eltt megállapítani, hogy van-e még pénzünk egy sörre:

Java

```
int pénz = 1000;
int sörÁra = 400;
while (pénz > sörÁra)
{
  pénz -= sörÁra; // Vonjuk le a sör árát a pénzünkbl
  System.out.println("Kérjünk egy korsó sört, majd igyuk meg.");
}
```

A végrehajtás során a belépési feltételünk kiértékelésre kerül, csak akkor hajtódik végre a ciklus magja, ha a feltétel igaz – vagyis van pénzünk legalább egy sörre. A ciklusmag végrehajtása után a feltétel újra és újra kiértékeldik, és a ciklus magja újra és újra végrehajtódik. Gyakori programozási hiba, hogy a ciklusmagban nem változtatjuk a feltételben szerepl változók értékét: ekkor végtelen ciklust kapunk, amely – ha ideje engedné – a ciklusmagot végtelen esetben hajtaná végre.

4.4.2. Hátul tesztel ciklus (do-while)

A hátul tesztel ciklus mködése hasonlít az ell tesztel ciklus mködésére, egyedüli különbség, hogy a feltétel kiértékelése az els ciklusmag lefutása **után** történik meg, tehát a hátul tesztel ciklus egyszer **mindenképpen** végrehajtja a ciklusmagot:

Java

```
int pénz = 1000;
int sörÁra = 400;
do
{
  System.out.println("Kérjünk egy korsó sört, majd igyuk meg.");
  pénz -= sörÁra; // Vonjuk le a sör árát a pénzünkbl
}
while (pénz > sörÁra);
```

A hátul tesztel ciklus tipikus felhasználási területe a beolvasott érték alapján történ döntés, a beolvasást a ciklus addig ismétli, amíg a várt értéket sikerül beolvasni.

4.4.3. Számláló Ciklus (for)

Számláló ciklust olyan esetben használjuk, amikor konkrétan tudjuk (avagy sejtjük), hogy hányszor fog ismétldni egy feladat. Ha például három kör sört szeretnénk inni, akkor:

Java

```
for (int kör = 1; kör <= 3; kör++)
{
  System.out.println("A(z) " + kör +". sör rendelése, majd elfogyasztása.");
}
```

Természetesen elfordulhat, hogy még nem ittuk meg a sörünket, ekkor ki tudunk maradni a körbl, ha a *continue* utasítást használjuk:

Java

```
for (int kör = 1; kör <= 3; kör++)
{
  if (vanMégSörAKorsónkban) continue;

  System.out.println("A(z) " + kör +". sör rendelése, majd elfogyasztása.");
}
```

Ha a *vanMégSörAKorsónkban* változó értéke igaz, akkor az adott körbl kimaradunk.

Lehetségünk van arra is, hogy kiszálljunk a ciklusból:

Java

```
for (int kör = 1; kör <= 3; kör++)
{
  if (elégVoltMára) break;

  System.out.println("A(z) " + kör +". sör rendelése, majd elfogyasztása.");
}
```

A *break* hatására a kikerülünk a ciklusból, így kifizethetjük a megivott söreinket és elbúcsúzhatunk a haveroktól... 🙂

4.4.4. Léptet ciklus (for each)