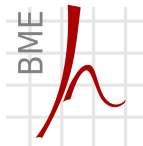


Strukturált programok – A C programnyelv elemei

A programozás alapjai I.



Hálózati Rendszerek és Szolgáltatások Tanszék
Farkas Balázs, Fiala Péter, Vitéz András, Zsóka Zoltán

2019. szeptember 16.

Adminisztrációs feladatok

Aki még nem tette meg...

- BME címtáras azonosító lekérése, jelszó beállítása
 - Szükség van rá a laborgépek használatához
 - <https://login.bme.hu/admin>
- Moodle bejelentkezés és kipróbálás
 - A tárggyal kapcsolatos adatokat (jelenlét, pontok) itt találhatók
 - Felhasználónév a neptun kód
 - Először az elfelejtett jelszó linket használjuk
 - <https://moodle.hit.bme.hu>
 - Próbáljuk ki a feladatleadást a laborkurzus Próba feladat feladatán
- Regisztráció további szolgáltatásokra (opcionális)
 - <https://accadmin.hszk.bme.hu>

Tartalom

1 Strukturált programozás

- Bevezetés
- Definíció
- Strukturált programok elemei
- Strukturált programozás tétele
- A struktogram

2 Strukturált programozás C-ben

- Szekvencia
- Választás
- Elöltesztelő ciklus
- Alkalmazás

3 Egyéb strukturált elemek

- Elöltesztelő másként
- Hátultesztelő ciklus
- Egész értéken alapuló választás

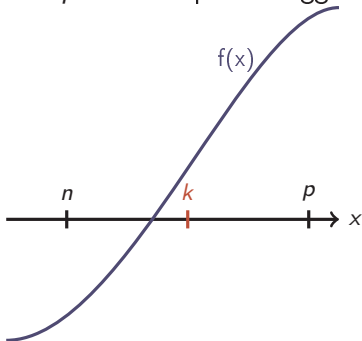
1. fejezet

Strukturált programozás

Algoritmusok

Ismétlés gyakorlatról:

Keressük az $f(x)$ monoton növekvő függvény zérushelyét n és p között ϵ pontossággal.

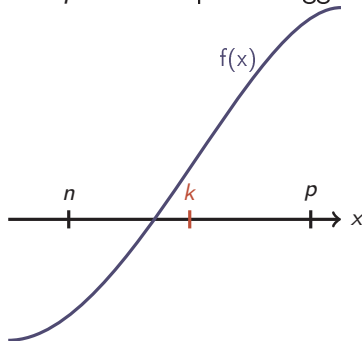


```
1  p-n < eps?  
2  HA IGAZ, UGORJ 10-re  
3  k ← (n+p) / 2  
4  f(k) < 0?  
5  HA IGAZ, UGORJ 8-ra  
6  p ← k;  
7  UGORJ 1-re  
8  n ← k;  
9  UGORJ 1-re  
10 A zérushely: n
```

Algoritmusok

Ismétlés gyakorlatról:

Keressük az $f(x)$ monoton növekvő függvény zérushelyét n és p között ϵ pontossággal.



```
AMÍG  $p - n > \epsilon$ , ISMÉTELD  
   $k \leftarrow (n + p) / 2$   
  HA  $f(k) > 0$   
     $p \leftarrow k$ ;  
  EGYÉBKÉNT  
     $n \leftarrow k$ ;  
A zérushely:  $n$ 
```

Strukturált vs strukturálatlan

```
AMÍG  $p - n > \text{eps}$ , ISMÉTELD  
   $k \leftarrow (n + p) / 2$   
  HA  $f(k) > 0$   
     $p \leftarrow k$ ;  
  EGYÉBKÉNT  
     $n \leftarrow k$ ;  
A zérushely:  $n$ 
```

```
1   $p - n < \text{eps}$ ?  
2  HA IGAZ, UGORJ 10-re  
3   $k \leftarrow (n + p) / 2$   
4   $f(k) < 0$ ?  
5  HA IGAZ, UGORJ 8-ra  
6   $p \leftarrow k$ ;  
7  UGORJ 1-re  
8   $n \leftarrow k$ ;  
9  UGORJ 1-re  
10 A zérushely:  $n$ 
```

■ Strukturált program

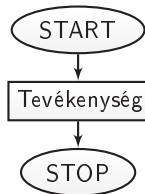
- könnyen karbantartható
- komplex vezérlés
- magas szintű

■ Strukturálatlan program

- spagettikód
- egyszerű vezérlés
- „hardverszint”

Strukturált programok elemei

- Minden strukturált program az alábbi egyszerű sémát követi:

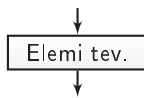
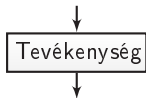


- A program struktúráját Tevékenység **belső szerkezete** határozza meg.
- Tevékenység lehet
 - Elemi tevékenység
 - Szekvencia
 - Ciklus
 - Választás

Strukturált programok elemei

Elemi tevékenység

ami nem szorul további kifejtésre



Elemi tev.

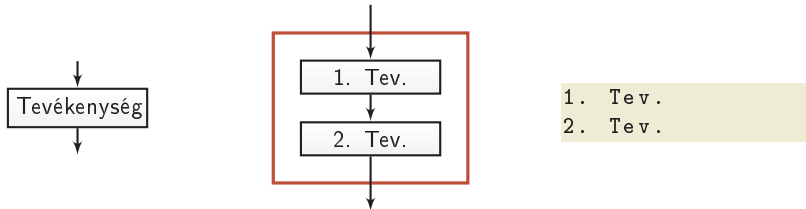
- Az üres tevékenység (Ne csinálj semmit) is elemi tevékenység



Strukturált programok elemei

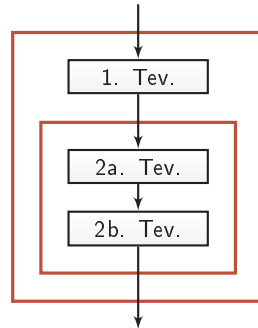
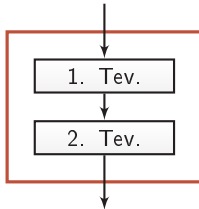
Szekvencia

Két tevékenység egymás utáni végrehajtása adott sorrendben



Strukturált programok elemei

- A szekvencia minden eleme maga is tevékenység, így természetesen kifejthető szekvenciaként

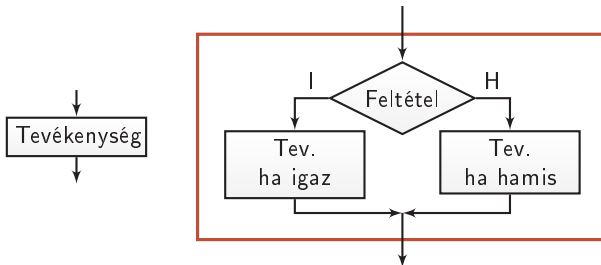


- A kifejtést folytatva a szekvencia gyakorlatilag tetszőleges hosszú (véges) tevékenységsorozatot jelenthet

Strukturált programok elemei

Igazságértéken alapuló választás

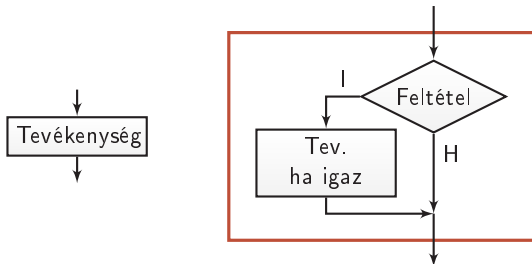
Két tevékenység alternatív végrehajtása egy feltétel igazságértékének megfelelően



HA Feltétel
Tev. ha igaz
EGYÉBKÉNT
Tev. ha hamis

Strukturált programok elemei

- Gyakran az egyik ág üres

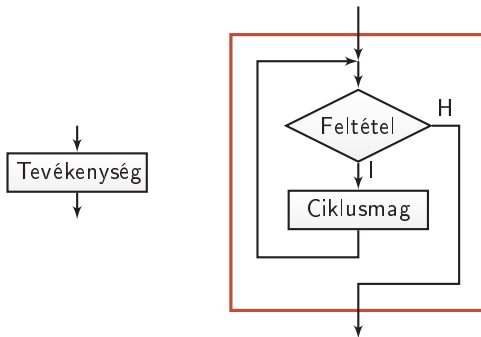


HA Feltétel
Tev. ha igaz

Strukturált programok elemei

Elöltesztelő ciklus

Tevékenység ismétlése mindaddig, míg egy feltétel teljesül



AMÍG Felt., ISMÉTELD
Ciklusmag

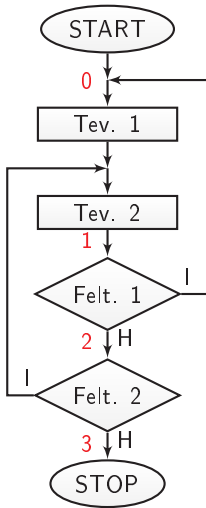
Strukturált programok elemei

Strukturált programozás tétele

- Elemi tevékenység,
- szekvencia,
- választás és
- ciklus

alkalmazásával **MINDEN** algoritmus megfogalmazható.

Strukturált programozás tétele – bizonyítás

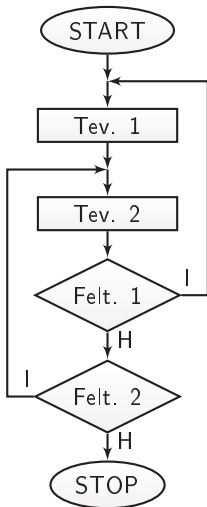


```

állapot ← 0
AMÍG állapot nem 3, ISMÉTELD
  HA állapot = 0
    Tev. 1
    Tev. 2
    állapot ← 1
  HA állapot = 1
    HA Felt. 1
      állapot ← 0
    EGYÉBKÉNT
      állapot ← 2
  HA állapot = 2
    HA Felt. 2
      Tev. 2
      állapot ← 1
    EGYÉBKÉNT
      állapot ← 3

```


Strukturált programozás tétele – bizonyítás



Most, hogy tudjuk, hogy mindig lehet, gondolkodhatunk egyszerűbb strukturált megfelelőn is:

```
Tev. 1
Tev. 2
AMÍG (Felt. 1 VAGY Felt. 2.), ISM.
    HA Felt. 1
        Tev. 1
    Tev. 2
```

Mi a továbbiakban **eleve** strukturált szerkezetben fogalmazzuk meg algoritmusainkat

A struktogram

- A folyamatábra
 - a strukturáltalan programok leírási eszköze
 - azonnal kódolható belőle strukturálatlan program (HA IGAZ, UGORJ)
 - a strukturált elemek (főleg a ciklusok) sokszor nehezen ismerhetők fel benne
- A struktogram
 - a strukturált programok ábrázolási eszköze
 - csak strukturált program írható le vele
 - könnyen kódolható belőle strukturált program

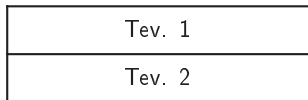
A struktogram

- A program egy téglalap



- további téglalapokra bontható az alábbi szerkezeti elemekkel

- Szekvencia



- Elöltesztelő ciklus

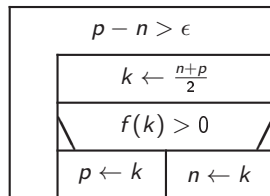
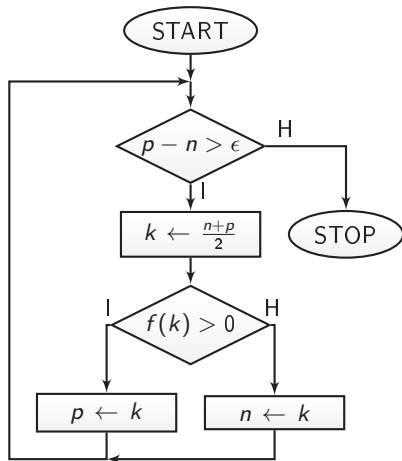


- Választás



A struktogram

- Gyökhelykeresés folyamatábrával, struktogrammal és strukturált pszeudokóddal



```

AMÍG p-n > eps, ISMÉTELD
  k ← (n+p) / 2
HA f(k) > 0
  p ← k;
EGYÉBKÉNT
  n ← k;
  
```

2. fejezet

Strukturált programozás C-ben

Szekvencia C-ben

Szekvencia megfogalmazása egymás után kiadott utasításokkal

```
1 /* football.c -- szurkolóprogram */
2 #include <stdio.h>
3 int main()
4 {
5     printf("Szódásüveget"); /* nincs újsor */
6     printf(" a bírónak,\n"); /* itt van */
7     printf("hajrá, Fradi!");
8     return 0;
9 }
```

[link](#)

```
Szódásüveget a bírónak,
hajrá, Fradi!
```

Választás C-ben – az if utasítás

Írjunk programot, mely a bekért egész számról eldönti, hogy az kicsi (< 10) vagy nagy (≥ 10)!

KI: infó	
BE: x	
x < 10	
KI: kicsi	KI: nagy

Legyen x egész

KI: infó

BE: x

HA x < 10

KI: kicsi

EGYÉBKÉNT

KI: nagy

```

1  #include <stdio.h>
2  int main()
3  {
4      int x;
5      printf("Adjon meg egy számot: ");
6      scanf("%d", &x);
7      if (x < 10)           /* feltétel */
8          printf("kicsi"); /*igaz ág*/
9      else
10         printf("nagy");  /*hamis ág*/
11     return 0;
12 }
```

[link](#)

Adjon meg egy számot: 5
kicsi

Választás – az if utasítás

Az if utasítás szintaxisa

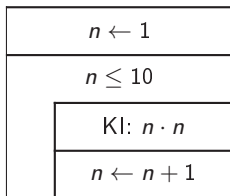
```
if (<feltétel kifejezés>) <utasítás ha igaz>  
[ else <utasítás ha hamis> ]opt
```

```
1 if (x < 10)           /* feltétel */  
2   printf("kicsi"); /* igaz ág */  
3 else  
4   printf("nagy");  /* hamis ág */
```

```
1 if (a < 0)           /* abszolútérték képzése */  
2   a = -a;  
3 /* nincs hamis ág */
```


Előtesztelő ciklus C-ben – a while utasítás

Írjuk ki 1-től 10-ig az egész számok négyzeteit!



Legyen n egész

$n \leftarrow 1$

AMÍG $n \leq 10$

KI: $n \cdot n$

$n \leftarrow n + 1$

```

1  #include <stdio.h>
2  int main()
3  {
4      int n;
5      n = 1; /* inicializálás */
6      while (n <= 10) /* feltétel */
7      {
8          printf("%d ", n*n); /* Kiírás */
9          n = n+1;             /* növelés */
10     }
11     return 0;
12 }
```

[link](#)

1 4 9 16 25 36 49 64 81 100

Elöltesztelő ciklus – a while utasítás

A while utasítás szintaxisa

```
while (<feltétel kifejezés>) <utasítás>
```

```
1 while (n <= 10)
2 {
3     printf("%d ", n*n);
4     n = n+1;
5 }
```

- C-ben utasítás mindig helyettesíthető blokkal.

Komplex alkalmazás

- Szekvenciával, ciklussal és választással minden megfogalmazható!
- Most már mindent tudunk, megírhatjuk a gyökhelykeresés programját C-ben!
- Új elem: a valós számok tárolására alkalmas `double` típus (később részletezzük)

```
1 double a;           /* a valós szám */
2 a = 2.0;            /* értékadás */
3 printf("%f", a);    /* kiírás */
```

Zérushely keresése

Keressük az

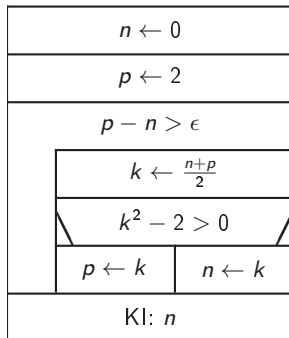
$$f(x) = x^2 - 2$$

függvény gyökhelyét

$n = 0$ és $p = 2$ között

$\epsilon = 0,001$

pontossággal!



```

1  #include <stdio.h>
2
3  int main()
4  {
5      double n = 0.0, p = 2.0;
6      while (p-n > 0.001)
7      {
8          double k = (n+p)/2.0;
9          if (k*k-2.0 > 0.0)
10             p = k;
11          else
12             n = k;
13      }
14      printf("A gyökhely: %f", n);
15
16      return 0;
17  }
```

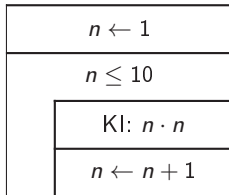
[link](#)

3. fejezet

Egyéb strukturált elemek

Elöltesztelő ciklus C-ben – a for utasítás

Írjuk ki 1-től 10-ig az egész számok négyzeteit!



Az

- Inicializálás
- Amíg Feltétel IGAZ
 - Tevékenység
 - Léptetés

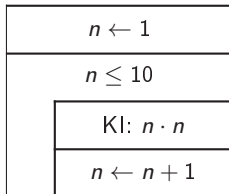
struktúra annyira gyakori a programozásban, hogy külön utasítással egyszerűsítjük alkalmazását.

```

Legyen n egész
n ← 1
AMÍG n <= 10
  KI: n*n
  n ← n+1
  
```

Elöltesztelő ciklus C-ben – a for utasítás

Írjuk ki 1-től 10-ig az egész számok négyzeteit!



Legyen n egész

$n=1$ -től, AMÍG $n \leq 10$, egyesével

KI: $n \cdot n$

```

1  #include <stdio.h>
2  int main()
3  {
4      int n;
5      for (n = 1; n <= 10; n = n+1)
6          printf("%d ", n*n);
7      return 0;
8  }
```

[link](#)

1 4 9 16 25 36 49 64 81 100

Elöltesztelő ciklus – a for utasítás

A for utasítás szintaxisa

```
for (<inic kif>; <felt kif>; <utótev kif>) <utasítás>
```

```
1 for (n = 1; n <= 10; n = n+1)
2   printf("%d ", n*n);
```

- Utótevékenység az utasítás végrehajtása után történik meg.

n: 11

1 4 9 16 25 36 49 64 81 100

Szorótábla

Írjuk ki a $10 \cdot 10$ -es szorzótáblát!

- 10 sort kell kiírnunk ($\text{row} = 1, 2, 3, \dots, 10$)
- Minden sorban
 - 10 oszlopba írunk ($\text{col} = 1, 2, 3, \dots, 10$)
 - Minden oszlopban
 - Kiírjuk $\text{row} \cdot \text{col}$ értékét
 - Majd új sort kell kezdenünk

```
1 int row;
2 for (row = 1; row <= 10; row=row+1)
3 {
4     int col;        /* blokk elején deklaráció */
5     for (col = 1; col <= 10; col=col+1)
6         printf("%4d", row*col); /* kiírás 4 szélesen */
7     printf("\n"); /* már nincs benne a col for-ban */
8 }
```

[link](#)

Szorótábla

- Ne sajnáljunk blokkba zárni akár egyetlen utasítást is, ha ez követhetőbbé teszi a kódot!

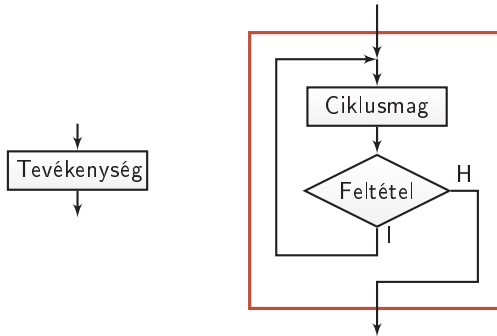
```
1  int row;  
2  for (row = 1; row <= 10; row=row+1)  
3  {  
4      int col;      /* blokk elején deklaráció */  
5      for (col = 1; col <= 10; col=col+1)  
6      {  
7          printf("%4d", row*col); /* kiírás 4 szélesen */  
8      }  
9      printf("\n");  
10 }
```

[link](#)

Strukturált programok elemei

Háttesztelő ciklus

Tevékenység ismétlése mindaddig, míg egy feltétel teljesül



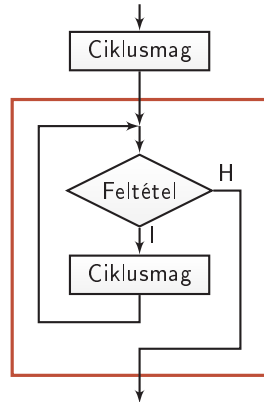
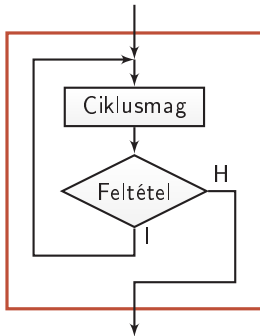
ISMÉTELD

Ciklusmag
AMÍG Feltétel



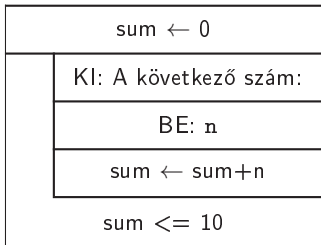
Strukturált programok elemei

■ Visszavezethető szekvenciára és előltesztelő ciklusra



Hátulatesztelő ciklus – a do utasítás

Olvassunk be pozitív egész számokat! Akkor hagyjuk abba, ha az összeg meghaladta a 10-et!



```
sum  $\leftarrow$  0
ISMÉTELD
    KI: Infó
    BE: n
    sum  $\leftarrow$  sum+n
AMÍG sum  $\leq$  10
```

```

1  #include <stdio.h>
2  int main()
3  {
4      int sum = 0, n;
5      do
6      {
7          printf("A következő szám: ");
8          scanf("%d", &n);
9          sum = sum+n;
10     }
11     while (sum <= 10);
12     return 0;
13 }
```

[link](#)

Háttesztelő ciklus – a do utasítás

A do utasítás szintaxisa

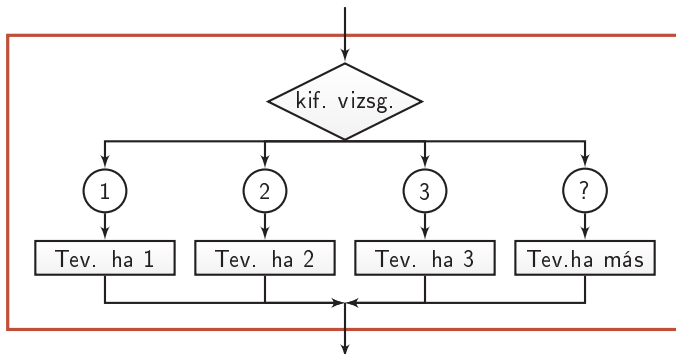
```
do <utasítás> while (<feltétel kifejezés>);
```

```
1 do
2 {
3     printf("A következő szám: ");
4     scanf("%d", &n);
5     sum = sum+n;
6 }
7 while (sum <= 10);
```

Strukturált programok elemei

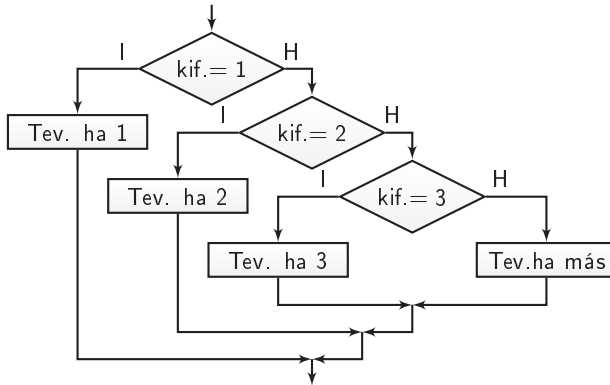
Egész értéken alapuló választás

Tevékenységek alternatív végrehajtása egy egész kifejezés értéke alapján



Strukturált programok elemei

- Megvalósítható egymásba ágyazott választásokkal



Egész értéken alapuló választás – a switch utasítás

- Rendeljünk szöveges értékelést számmal kifejezett jegyekhez!

Kl: infó					
BE: n					
$n = ?$					
\ 1	\ 2	\ 3	\ 4	\ 5	\ más
Kl: elégtelen	Kl: elégséges	Kl: közepes	Kl: jó	Kl: jeles	Kl: baj van

Egész értéken alapuló választás – a switch utasítás

- Rendeljük szöveges értékelést számmal kifejezett jegyekhez!

```
1 #include <stdio.h>
2 int main() {
3     int n;
4     printf("Adja meg a jegyet: ");
5     scanf("%d", &n);
6     switch (n)
7     {
8         case 1: printf("elégtelen"); break;
9         case 2: printf("elégséges"); break;
10        case 3: printf("közepes"); break;
11        case 4: printf("jó"); break;
12        case 5: printf("jeles"); break;
13        default: printf("baj van");
14    }
15    return 0;
16 }
```

[link](#)

Egész értéken alapuló választás – a switch utasítás

A switch utasítás szintaxisa

```
switch(<egész kifejezés>) {  
    case <konstans kif1>: <utasítás 1>  
    [case <konstans kif2>: <utasítás 2> ...]opt  
    [default: <default utasítás> ]opt  
}
```

```
1 switch (n)  
2 {  
3     case 1: printf("elégtelen"); break;  
4     case 2: printf("elégséges"); break;  
5     case 3: printf("közepes"); break;  
6     case 4: printf("jó"); break;  
7     case 5: printf("jeles"); break;  
8     default: printf("baj van");  
9 }
```

Egész értéken alapuló választás – a switch utasítás

- A `break` utasítások nem részei a szintaxisnak. Ha le hagyjuk őket, a `switch` akkor is értelmes, de nem a korábban specifikált eredményt adja:

```
1 switch (n)
2 {
3     case 1: printf("elégtelen");
4     case 2: printf("elégséges");
5     case 3: printf("közepes");
6     case 4: printf("jó");
7     case 5: printf("jeles");
8     default: printf("baj van");
9 }
```

[link](#)

```
Adja meg a jegyet: 2
elégségesközepesjójelesbaj van
```

Egész értéken alapuló választás – a switch utasítás

- A konstans kifejezések csak belépési pontok, ahonnan minden utasítást végrehajtunk az első **break**-ig vagy a blokk végéig:

```
1 switch (n)
2 {
3     case 1: printf("megbukott"); break;
4     case 2:
5     case 3:
6     case 4:
7     case 5: printf("átment"); break;
8     default: printf("baj van");
9 }
```

[link](#)

```
Adja meg a jegyet: 2
átment
```

Köszönöm a figyelmet.