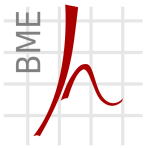
Strukturált programok – A C programnyelv elemei

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 adatok (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 Próbafeladat feladatán
- Regisztráció további szolgáltatásokra (opcionális)
  - https://accadmin.hszk.bme.hu

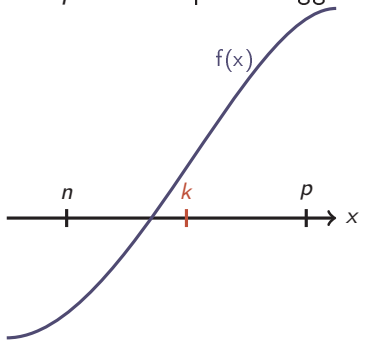
Tartalom

1. fejezet

Strukturált programozás

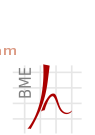
Algoritmusok

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

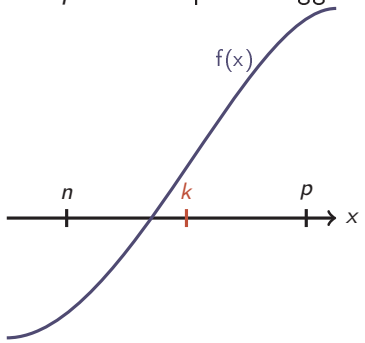



```
1  p-n < eps?
2  HA IGAZ, UGORJ 10-re
3  k ← (n+p) / 2
4  f(k) < 0?
5  HA IGAZ, UGORJ 8-ra
6  p ← k;
7  UGORJ 1-re
8  n ← k;
9  UGORJ 1-re
10 A zérushely: n
```


Algoritmusok



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




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


Strukturált vs strukturálatlan

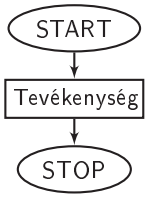
```
AMÍG p-n > eps, ISMÉTELD
  k ← (n+p) / 2
  HA f(k) > 0
    p ← k;
  EGYÉBKÉNT
    n ← k;
A zérushely: n
```

```
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
```

- 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

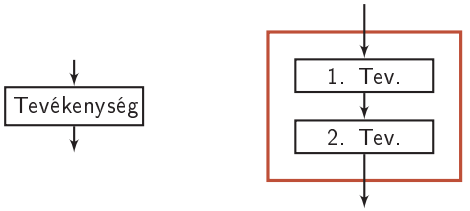

```
Elemi tev.
```

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

Strukturált programok elemei

Szekvencia

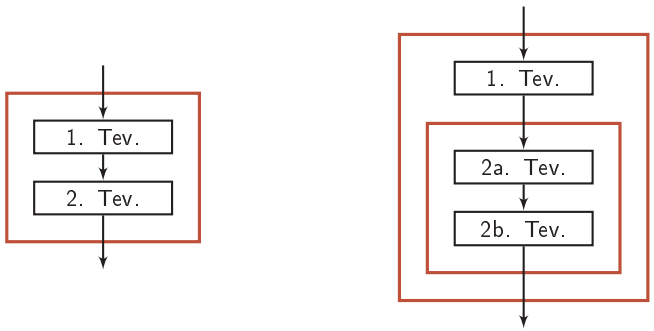
Két tevékenység egymás utáni végrehajtása adott sorrendben




```
1. Tev.
2. Tev.
```


Strukturált programok elemei

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

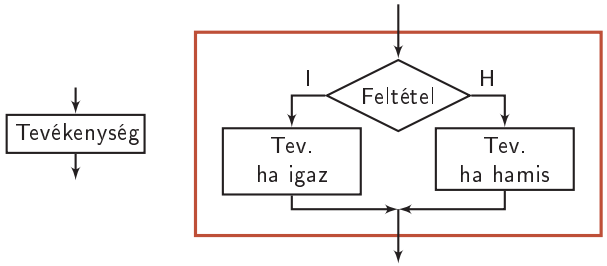


- A kifejtést folytatva a szekvencia gyakorlatilag tetszőleges hosszú (véges) tevékenységsorozatot jelenthet

Strukturált programok elemei

Igazságértéken alapuló választás

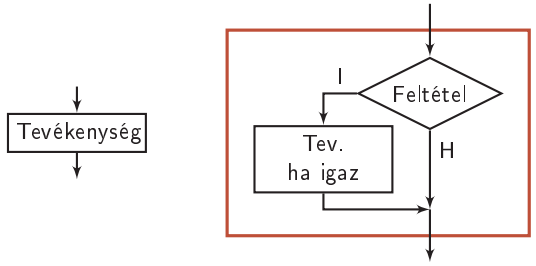
Két tevékenység alternatív végrehajtása egy feltétel igazságértékének megfelelően




```
HA Feltétel
  Tev. ha igaz
EGYÉBKÉNT
  Tev. ha hamis
```

Strukturált programok elemei

- Gyakran az egyik ág üres

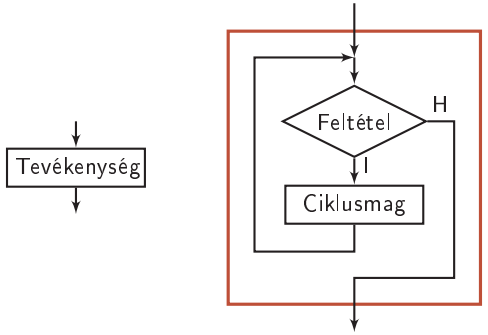



```
HA Feltétel
  Tev. ha igaz
```


Strukturált programok elemei

Elöltesztelő ciklus

Tevékenység ismétlése mindaddig, míg egy feltétel teljesül




```
AMÍG Felt., ISMÉTELD
  Ciklusmag
```

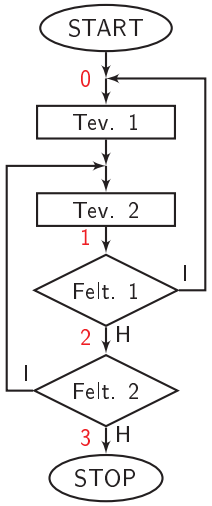

Strukturált programok elemei

Strukturált programozás tétele

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

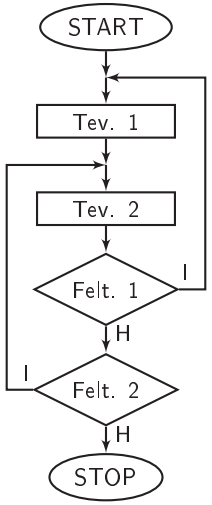
alkalmazásával MINDEN algoritmus megfogalmazható.

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

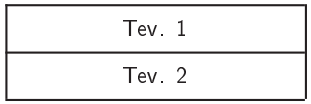
A struktogram

- A folyamatábra
  - a strukturáltalan programok leírási eszköze
  - azonnal kódolható belőle strukturálatlan program (HA IGAZ, UGORJ)
  - a strukturált elemek (főleg a ciklusok) sokszor nehezen ismerhetőek 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

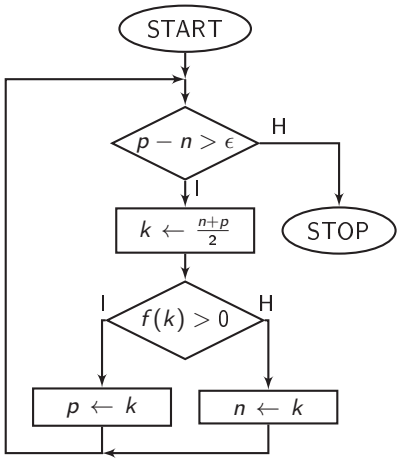


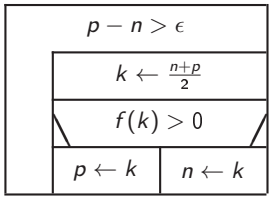
  - Elöltesztelő ciklus

  - 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

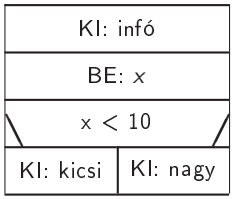
Szekvencia megfogalmazása egymás után kiadott utasításokkal

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

```
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)!




```
Legyen x egész
KI: infó
BE: x
HA x < 10
  KI: kicsi
EGYÉBKÉNT
  KI: nagy
```

```c
#include <stdio.h>
int main()
{
  int x;
  printf("Adjon meg egy számot: ");
  scanf("%d", &x);
  if (x < 10)          /* feltétel */
    printf("kicsi"); /*igaz ág*/
  else
    printf("nagy");  /*hamis ág*/
  return 0;
}
```

```
Adjon meg egy számot:  5
kicsi
```


Választás – az `if` utasítás

Az `if` utasítás szintaxisa

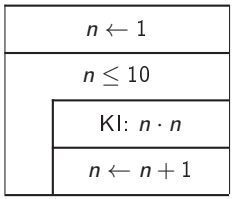
```
if (<feltétel kifejezés>) <utasítás ha igaz>
[ else <utasítás ha hamis> ]opt
```

```c
if (x < 10)          /* feltétel */
  printf("kicsi"); /* igaz ág */
else
  printf("nagy"); /* hamis ág */
```

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

Elöltesztelő ciklus C-ben – a while utasítás

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




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

```c
#include <stdio.h>
int main()
{
  int n;
  n = 1; /* inicializálás */
  while (n <= 10) /* feltétel */
  {
    printf("%d ", n*n);/* Kiírás */
    n = n+1;          /* növelés */
  }
  return 0;
}
```

```
1 4 9 16 25 36 49 64 81 100
```


Elöltesztelő ciklus – a while utasítás

A while utasítás szintaxisa

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

```c
while (n <= 10)
{
  printf("%d ", n*n);
  n = n+1;
}
```

- C-ben utasítás mindig helyettesíthető blokkal.

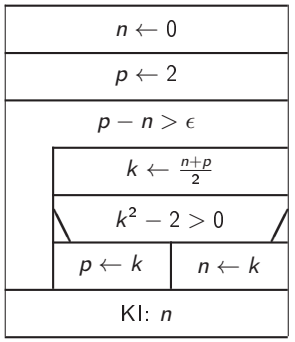
Komplex alkalmazás

- 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)

```c
double a;          /* a valós szám */
a = 2.0;           /* értékadás */
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!

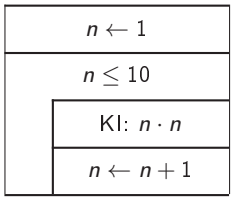



```c
#include <stdio.h>

int main()
{
  double n = 0.0, p = 2.0;
  while (p-n > 0.001)
  {
    double k = (n+p)/2.0;
    if (k*k-2.0 > 0.0)
      p = k;
    else
      n = k;
  }
  printf("A gyökhely: %f", n);

  return 0;
}
```

3. fejezet

Egyéb strukturált elemek

Elöltesztelő ciklus C-ben – a `for` utasítás



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

Az

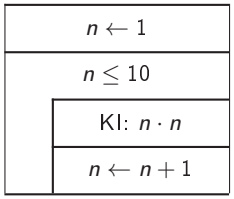
- Inicializálás
- Amíg Feltétel IGAZ
  - Tevékenység
  - Léptetés

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<=10, egyesével
  KI: n*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 }
```

```
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
```

Szorzótábla

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

- 10 sort kell kiírnunk (`row = 1, 2, 3, ...10`)
- Minden sorban
  - 10 oszlopba írunk (`col = 1, 2, 3, ...10`)
  - Minden oszlopban
    - Kiírjuk `row*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  }
```


Szorzótábla

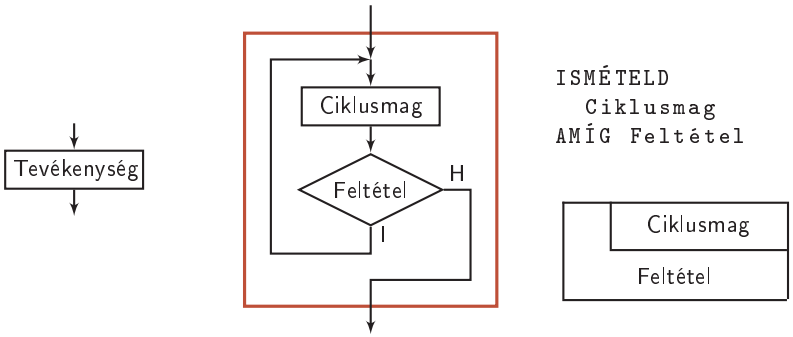
- Ne sajnáljunk blokkba zárni akár egyetlen utasítást is, ha ez követhetőbbé teszi a kódot!

```
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 }
```


Strukturált programok elemei

Hátultesztelő ciklus

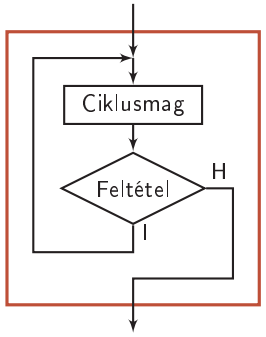
Tevékenység ismétlése mindaddig, míg egy feltétel teljesül

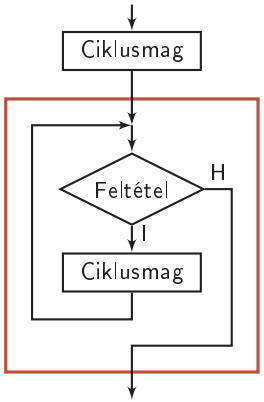



```
ISMÉTELD
  Ciklusmag
AMÍG Feltétel
```


Strukturált programok elemei

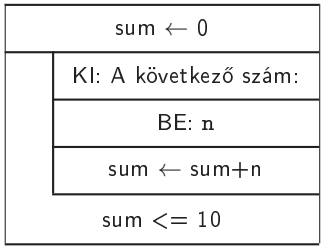
- Visszavezethető szekvenciára és elöltesztelő ciklusra





Hátultesztelő ciklus – a do utasítás

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




```
sum ← 0
ISMÉTELD
  KI: Infó
  BE: n
  sum ← sum+n
AMÍG sum ≤ 10
```

```c
#include <stdio.h>
int main()
{
  int sum = 0, n;
  do
  {
    printf("A következő szám: ");
    scanf("%d", &n);
    sum = sum+n;
  }
  while (sum <= 10);
  return 0;
}
```


Hátultesztelő ciklus – a do utasítás

A do utasítás szintaxisa

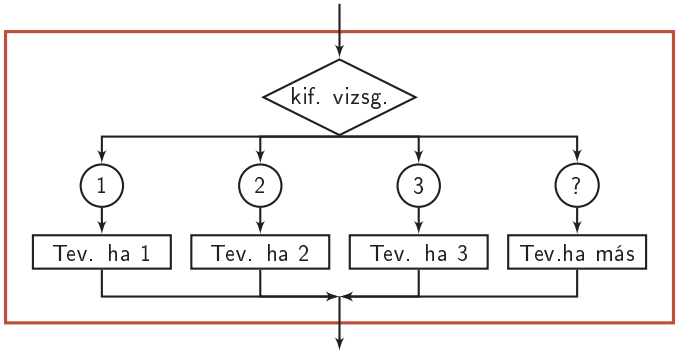
```
do <utasítás> while (<feltétel kifejezés>);
```

```c
do
{
  printf("A következő szám: ");
  scanf("%d", &n);
  sum = sum+n;
}
while (sum <= 10);
```


Strukturált programok elemei

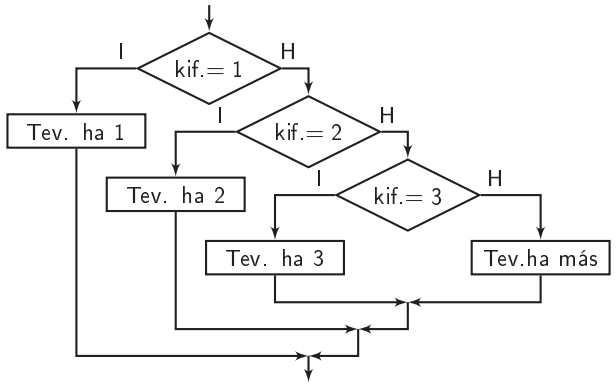
Egész értéken alapuló választás

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!

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

Egész értéken alapuló választás – a switch utasítás

- Rendeljünk szöveges értékelést számmal kifejezett jegyekhez!

```c
#include <stdio.h>
int main() {
  int n;
  printf("Adja meg a jegyet: ");
  scanf("%d", &n);
  switch (n)
  {
    case 1: printf("elégtelen"); break;
    case 2: printf("elégséges"); break;
    case 3: printf("közepes"); break;
    case 4: printf("jó"); break;
    case 5: printf("jeles"); break;
    default: printf("baj van");
  }
  return 0;
}
```


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
}
```

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


Egész értéken alapuló választás – a switch utasítás

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

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

```
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 }
```

```
Adja meg a jegyet:  2
átment
```