Programozás 1

Vezérlési módok és
Szerkezeti ábrák

Szerkezeti ábra

- Az algoritmus tervezése során le kell írnunk valamilyen nyelvezetet használva azt, hogy a problémát milyen részproblémákra bontottuk fel, és a megoldásukat milyen módon raktuk össze. Mi erre úgynevezett szerkezeti ábrát (structure diagram, struktúradiagram) használunk.
- Minden vezérlési módhoz bevezetünk egy szerkezeti ábra jelölést.

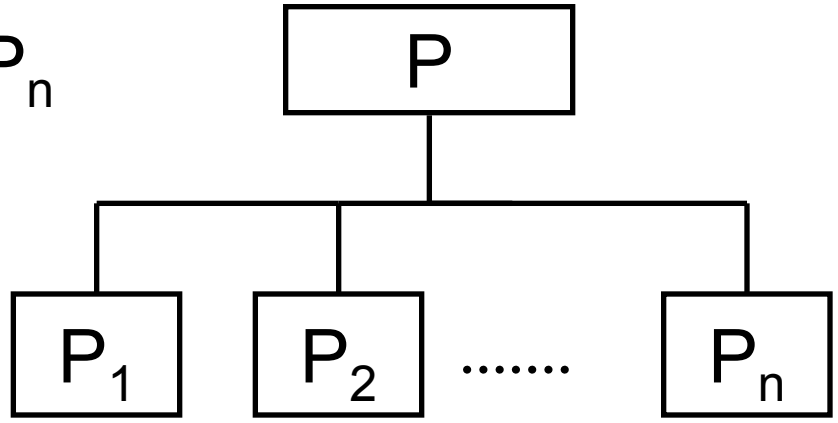
Szerkezeti ábra tulajdonságai

- A szerkezeti ábra egyszerre fejezi ki az algoritmustervezés folyamatát és a kifejlesztett algoritmust is.
- Egy részprobléma megoldását leíró szerkezeti ábrarész különálló ábrával is kifejezhető, amelynek gyökerében a részprobléma megnevezése áll.

SZEKVENCIÁLIS VEZÉRLÉS

Szekvenciális vezérlés

- A P problémát P_1, ... , P_n részproblémákra bontjuk, és ezek megoldásait ebben a sorrendben egymás után végrehajtva kapjuk P megoldását



- P_1, ... , P_n elemi műveletek, vagy részproblémák megnevezése. Utóbbi esetben a részproblémát tovább kell bontani.

Eltelt idő

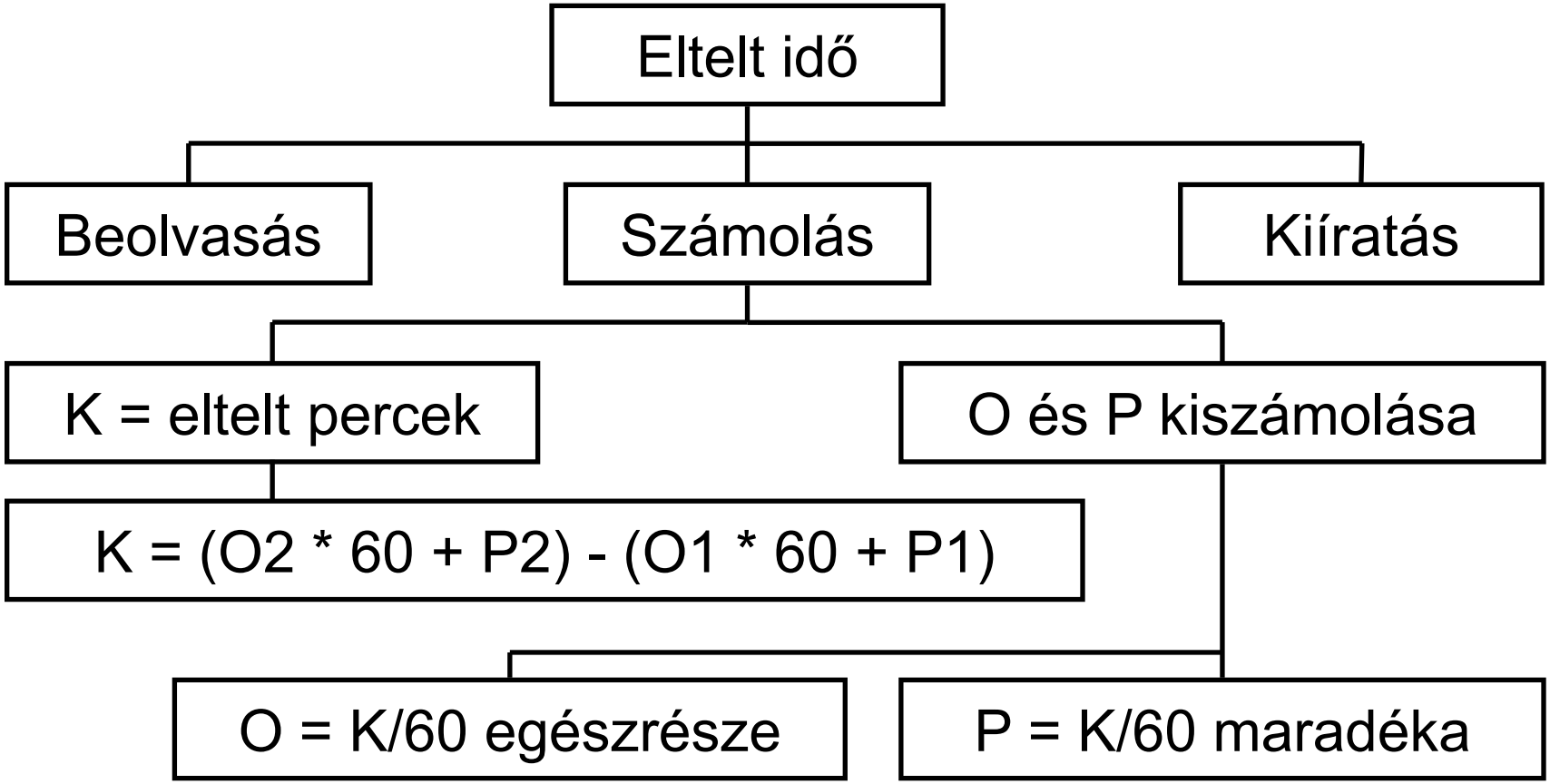
- Problémafelvetés
  - Kiszámítandó a nap két időpontja között eltelt idő.
- Specifikáció
  - A bemenő adat két időpont óra és perc formában, jelöljük ezeket O1, P1 illetve O2, P2-vel. A bemeneti feltétel:
    - (0 <= O1 < 24) és (0 <= P1 < 60)
    - (0 <= O2 < 24) és (0 <= P2 < 60)
    - (O1 < O2) vagy (O1 == O2) és (P1 <= P2)
  - A kimenő adatok O és P. A kimeneti feltétel:
    - Az első időponttól a másodikig O óra és P perc telt el

Eltelt idő

- Algoritmustervezés
  - Az eltelt idő percben kifejezve
    - (O2 * 60 + P2) - (O1 * 60 + P1)
  - Tehát O, P akkor és csak akkor megoldás, ha
    - O * 60 + P == (O2 * 60 + P2) - (O1 * 60 + P1)
    - 0 <= P < 60
  - Ez tulajdonképpen a kimeneti feltételek formális megadása, de az első feltétel már mindenképpen az algoritmustervezés fázisához kapcsolódik

Eltelt idő

- Szerkezeti ábra



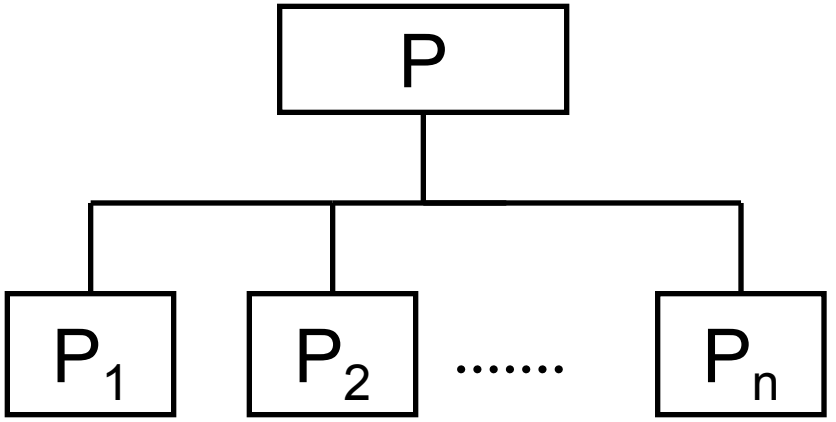
Eltelt idő

- Az Eltelt idő algoritmusban használt változók
  - O1, P1, O2, P2, O, P, K: valamennyi egész típusú, értékük tetszőleges egész szám lehet
- Az egész értékeken a következő műveleteket alkalmaztuk
  - Összeadás (+)
  - Kivonás (-)
  - Szorzás (*)
  - Osztás egészrésze (/)
  - Osztás maradéka (%)

Szekvenciális vezérlés C nyelven

- Kódolása C nyelven:

```
{
    P1;
    P2;
    .
    .
    .
    Pn;
}
```



Eltelt idő

```
/*
 * Egy nap két időpontja között mennyi idő telt el.
 * Készítette: Dévényi Károly, devenyi@inf.u-szeged.hu
 *             1997. Szeptember 26. Péntek
 */

#include <stdio.h>

main()
{
  int o1,p1;                        /* az első időpont */
  int o2,p2;                       /* a második időpont */
  int o,p;                           /* az eltelt idő */
  int k;                        /* az eltelt idő percben */
```

>>>

Eltelt idő

```
                                                /* beolvasás */
  printf("Kérem az első időpontot óra perc formában\n");
  scanf("%d %d", &o1, &p1);
  printf("Kérem a második időpontot óra perc formában\n");
  scanf("%d %d", &o2, &p2);
                                                /*  számítás */
                                        /* különbség számítás */
  k = 60 * o2 + p2 - (60 * o1 + p1);
  o = k / 60;
  p = k % 60;
                                                /*  kiíratás */
  printf("Az eltelt idő: %d óra %d perc.\n", o, p);
}
```

SZELEKCIÓS VEZÉRLÉSEK

Szelekciós vezérlések

- Szelekciós vezérléssel azt írjuk elő, hogy véges sok rögzített művelet közül, véges sok adott feltétel alapján, melyik művelet kerüljön végrehajtásra.
- Típusai:
  - Egyszerű
  - Többszörös
  - Esetkiválasztásos
  - A fenti három „egyébként” ággal

Egyszerű szelekciós vezérlés

- Egyszerű szelekció esetén egy feltétel és egy művelet van.
- Legyen F logikai kifejezés, A pedig tetszőleges művelet. Az F feltételből és az A műveletből képzett egyszerű szelekciós vezérlés a következő vezérlési előírást jelenti:

  1.) Értékeljük ki az F feltételt és folytassuk a 2.) lépéssel.

  2.) Ha F értéke igaz, akkor hajtsuk végre az A műveletet és fejezzük be az összetett művelet végrehajtását.

  3.) Egyébként, vagyis ha F értéke hamis, akkor fejezzük be az összetett művelet végrehajtását.

Egyszerű szelekciós vezérlés

- A vezérlés bővíthető úgy, hogy a 3. pontban üres művelet helyett egy B műveletet hajtunk végre.
- Legyen F logikai kifejezés, A és B pedig tetszőleges művelet. Az F feltételből és az A és B műveletből képzett egyszerű szelekciós vezérlés a következő vezérlési előírást jelenti:

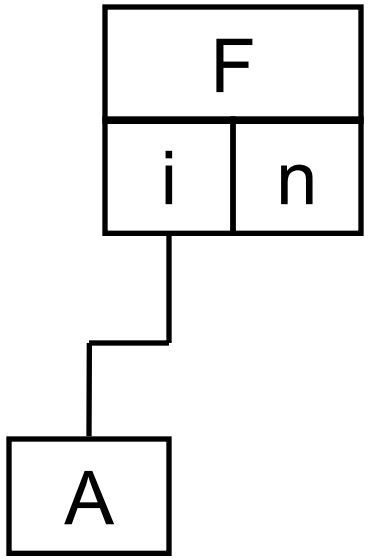
  1.) Értékeljük ki az F feltételt és folytassuk a 2.) lépéssel.

  2.) Ha F értéke igaz, akkor hajtsuk végre az A műveletet és fejezzük be az összetett művelet végrehajtását.

  3.) Egyébként, vagyis ha F értéke hamis, hajtsuk végre B-t és fejezzük be az összetett műveletet végrehajtását.

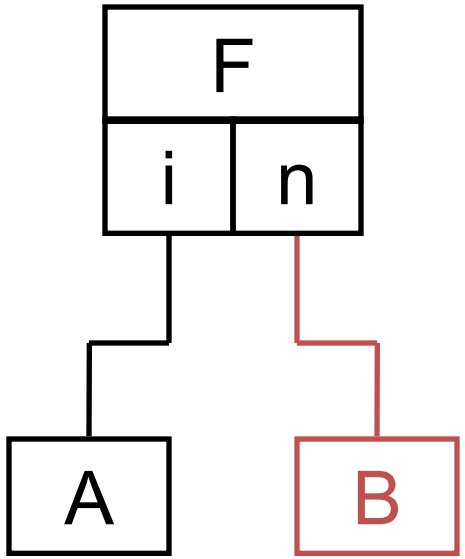
Egyszerű szelekciós vezérlés

- Az
  egyszerű szelekciós vezérlés
  szerkezeti ábrája



Egyszerű szelekciós vezérlés

- Az egyébként ággal kiegészített egyszerű szelekciós vezérlés szerkezeti ábrája



Egyszerű szelekciós vezérlés

- A vezérlés lényege:
  - Ha az F feltétel igaz, hajtsuk végre az A műveletet.
  - Ha az F feltétel hamis, hajtsuk végre a B műveletet.

Többszörös szelekciós vezérlés

- Ha a szelekciós vezérlésben a feltételek száma nagyobb, mint egy, akkor többszörös szelekcióról beszélünk.
- Ekkor minden kiválasztó feltételhez tartozik egy alternatív művelet.
- Legyenek F_i logikai kifejezések, A_i pedig tetszőleges műveletek ($1<=i<=n$).

Többszörös szelekciós vezérlés

- Az F_i kiválasztó feltételekből valamint az A_i műveletekből képzett többszörös szelekciós vezérlés a következő vezérlési előírást jelenti:

  1.) Az F_i feltételek kiértékelésével adjunk választ a következő kérdésre: Van-e olyan i (1<=i<=n), amelyre teljesül, hogy az F_i feltétel igaz és az összes F_j (1<=j<i) feltétel hamis? Folytassuk a 2.) lépéssel.

  2.) Ha van ilyen i, akkor hajtsuk végre az A_i műveletet és fejezzük be az összetett művelet végrehajtását.

  3.) Egyébként, vagyis ha minden F_i feltétel hamis, akkor fejezzük be az összetett művelet végrehajtását.

Többszörös szelekciós vezérlés

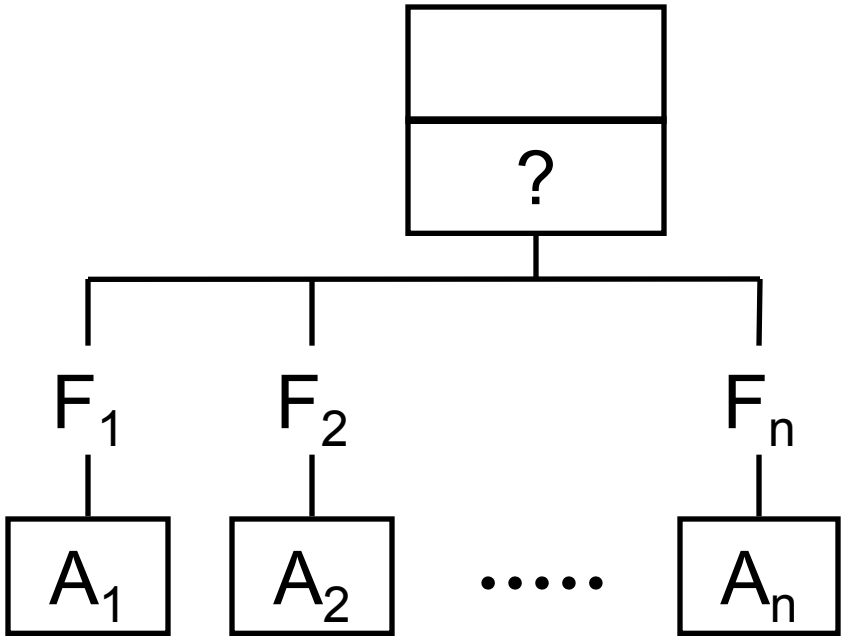
- Bővíthetjük a többszörös szelekciós vezérlést azzal, hogy a 3.) pontban ne az üres művelet, hanem egy előre megadott B tetszőleges művelet végrehajtását írjuk elő.
- Legyenek F_i logikai kifejezések, A_i és B pedig tetszőleges műveletek (1<=i<=n).

Többszörös szelekciós vezérlés

- Az F_i kiválasztó feltételekből, valamint az A_i és B műveletekből képzett többszörös szelekciós vezérlés a következő vezérlési előírást jelenti:
  1.) Az F_i feltételek kiértékelésével adjunk választ a következő kérdésre: Van-e olyan i (1<=i<=n), amelyre teljesül, hogy az F_i feltétel igaz és az összes F_j (1<=j<i) feltétel hamis? Folytassuk a 2.) lépéssel.
  2.) Ha van ilyen i, akkor hajtsuk végre az A_i műveletet és fejezzük be az összetett művelet végrehajtását.
  3.) Egyébként, vagyis ha minden F_i hamis, hajtsuk végre B-t és fejezzük be az összetett művelet végrehajtását.

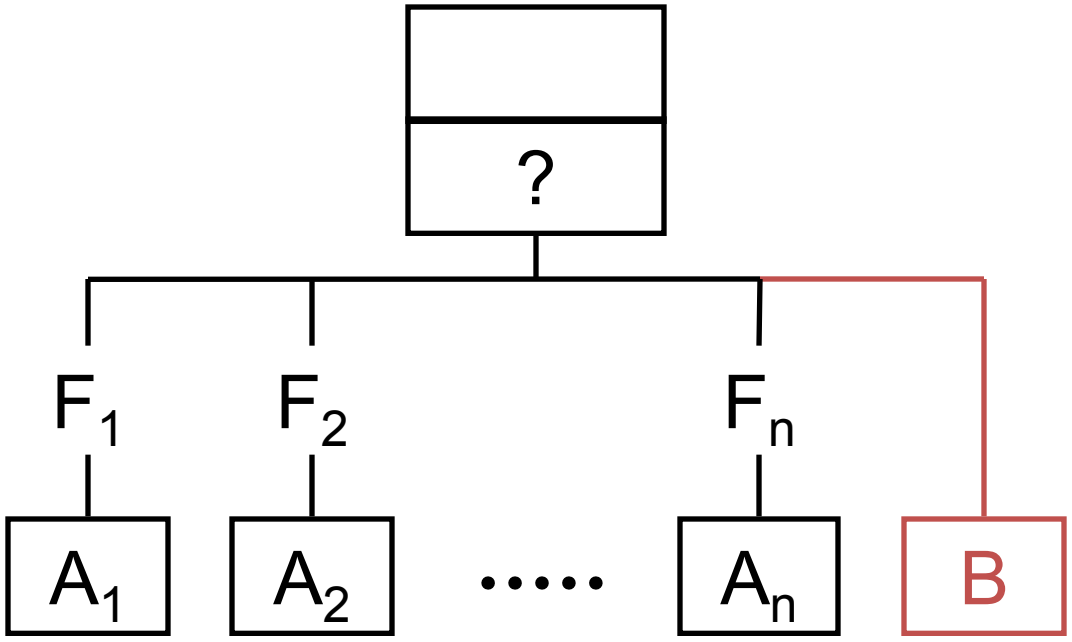
Többszörös szelekciós vezérlés

- A
  többszörös szelekciós vezérlés
  szerkezeti ábrája



Többszörös szelekciós vezérlés

- Az egyébként ággal kiegészített többszörös szelekciós vezérlés szerkezeti ábrája



Többszörös szelekciós vezérlés

- A vezérlés lényege
  - Hajtsuk végre a legelső olyan A_i műveletet, aminek az F_i feltétele igaz.
  - Ha nincs olyan F_i ami igaz, akkor hajtsuk végre a B műveletet.

Háromszögek osztályozása

- Problémafelvetés
  - Milyen háromszöget határoz meg három valós szám, mint a háromszög három oldalhosszúsága?
- Specifikáció
  - A probléma inputja
    - A,B,C valós számok
  - Outputja a következő szövegek egyike
    - ‘nem háromszög’, ‘szabályos háromszög’, ‘egyenlőszárú háromszög’, ‘egyenlőszárú derékszögű háromszög’, ‘derékszögű háromszög’, ‘egyéb háromszög’

Háromszögek osztályozása

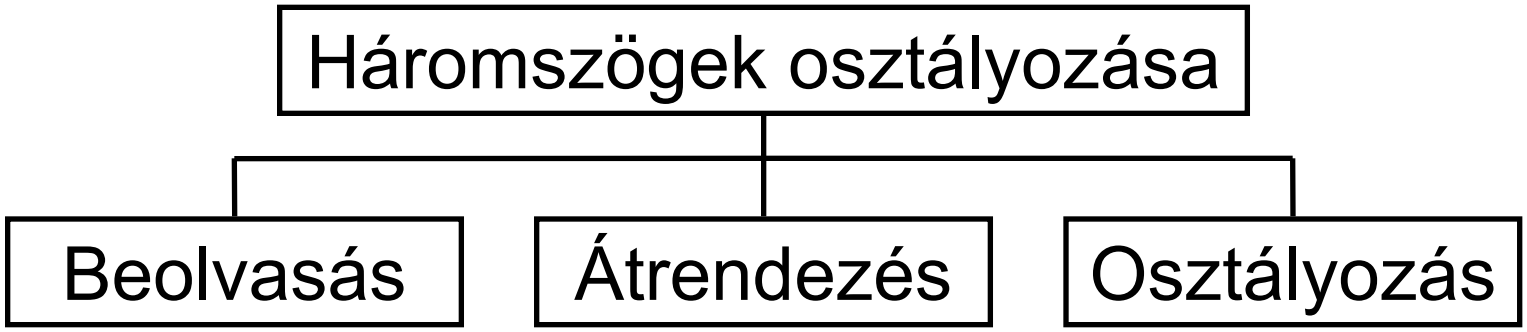
- Algoritmustervezés: fogalmazzuk meg a feltételeket magyarul és a matematika nyelvén
  - ‘nem háromszög’
  - ‘szabályos háromszög’
  - ‘egyenlőszárú háromszög’
  - ‘egyenlőszárú derékszögű háromszög’
  - ‘derékszögű háromszög’
  - ‘egyéb háromszög’

Háromszögek osztályozása

- Algoritmustervezés
  - A feltételek megfogalmazása érdekében célszerűnek látszik a bemenő adatokat úgy átrendezni, hogy az A változó tartalmazza a legnagyobb értéket.
  - Ekkor a feltételek rendre
    - NP feltétel: (A<=0) vagy (B<=0) vagy (C<=0)
    - N feltétel: A >= B + C
    - Sz feltétel: (A == B) és (B == C)
    - E feltétel: (A == B) vagy (B == C) vagy (A == C)
    - D feltétel: $A^2 = B^2 + C^2$
  - Az osztályozás eredményét közvetlenül kiírjuk.

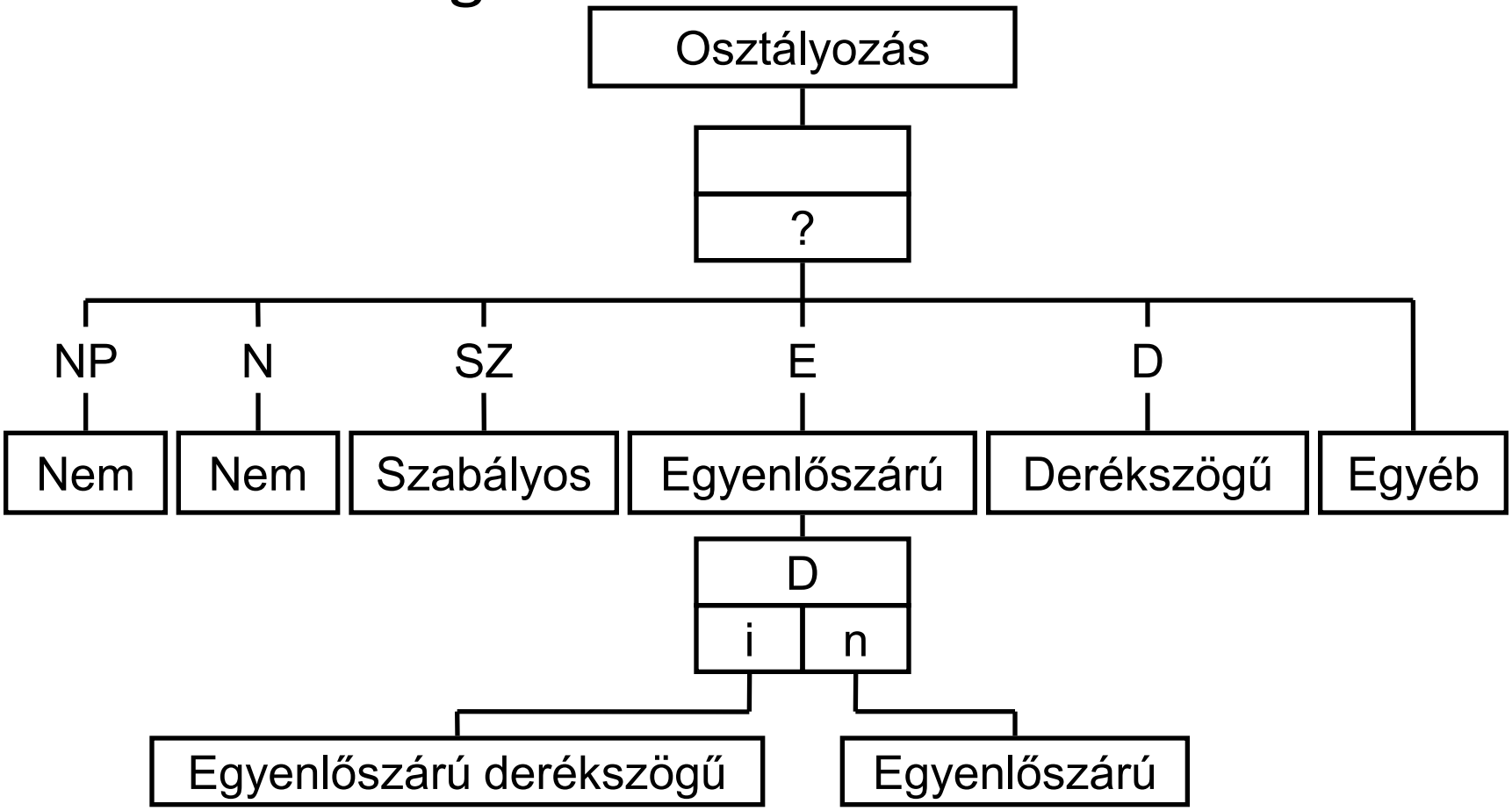
Háromszögek osztályozása

- Struktúradiagram



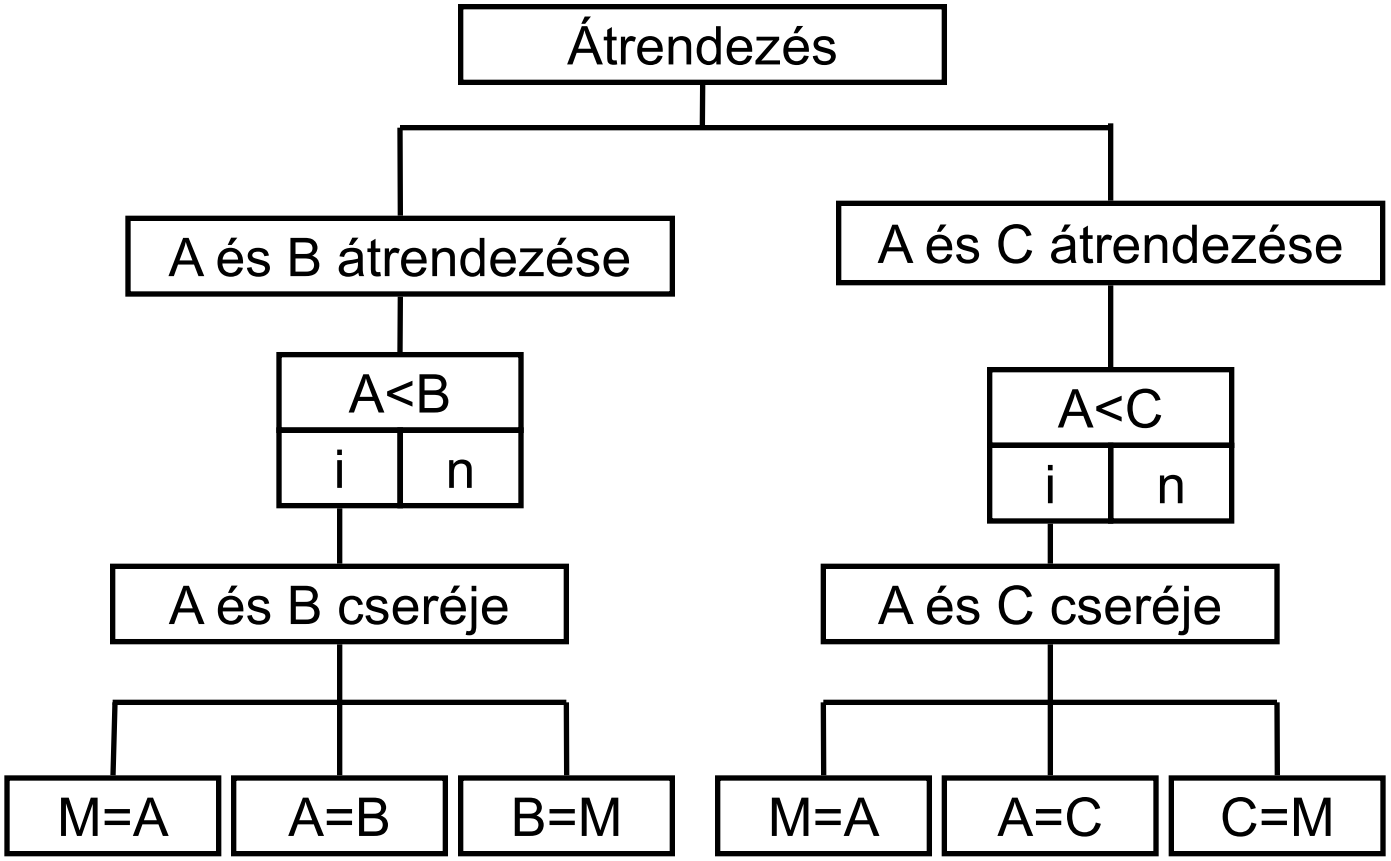
Háromszögek osztályozása

- Struktúradiagram



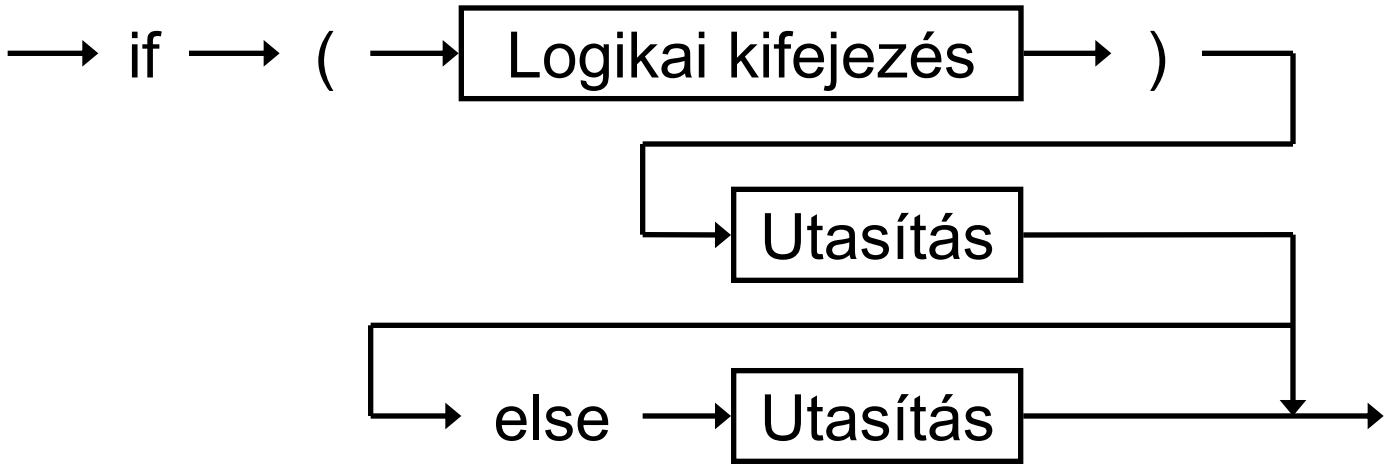
Háromszögek osztályozása

- Struktúradiagram



Az `if` utasítás

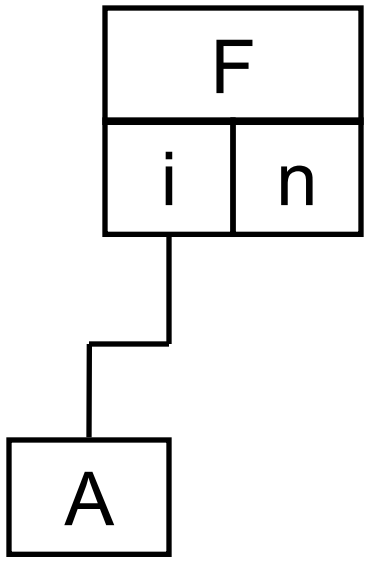
- Ha valamilyen feltétel alapján egyik vagy másik utasítást akarjuk végrehajtani
- Szintaxis



Egyszerű szelekciós vezérlés

- Az
  egyszerű szelekciós vezérlés
  szerkezeti ábrája és C
  megvalósítása

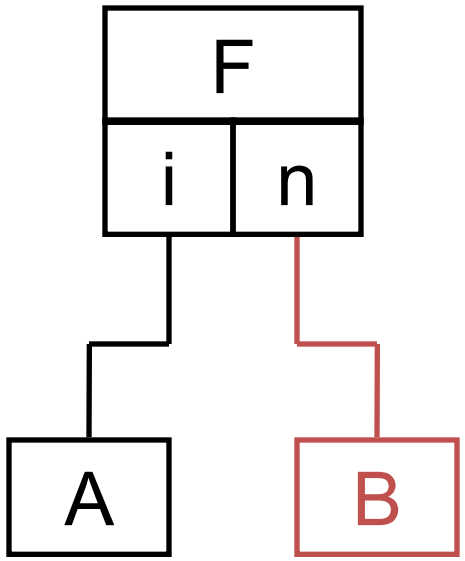
```
if ( F ) {
   A;
}
```



Egyszerű szelekciós vezérlés

- Az egyébként ággal kiegészített egyszerű szelekciós vezérlés szerkezeti ábrája és C megvalósítása

```
if ( F ) {
  A;
} else {
  B;
}
```

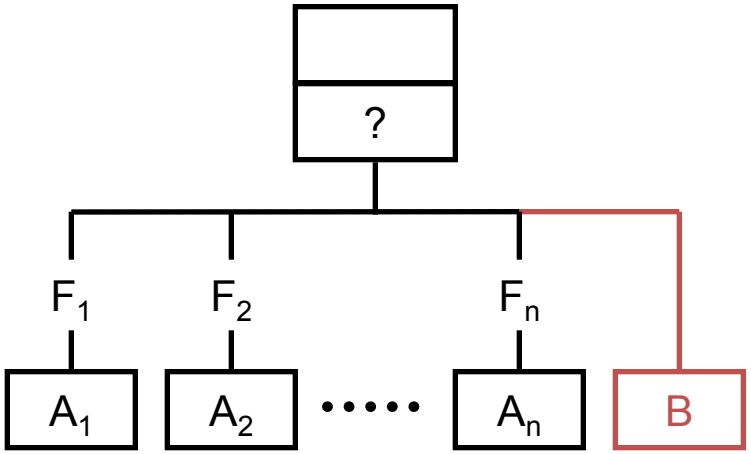


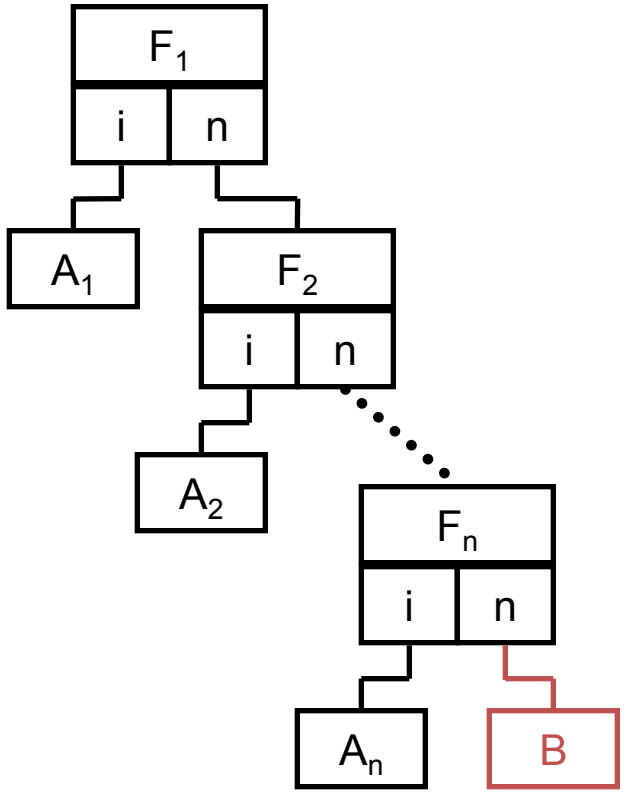
Többszörös szelekció megvalósítása

- C nyelvben nincs külön utasítás a többszörös szelekció megvalósítására, ezért az egyszerű szelekció ismételt alkalmazásával kell azt megvalósítani.
- Ez azon az összefüggésen alapszik, hogy a többszörös szelekció levezethető egyszerű szelekciók megfelelő összetételével.

Többszörös szelekció megvalósítása

- Az alábbi két ábra ugyanazt a vezérlési előírást fejezi ki.

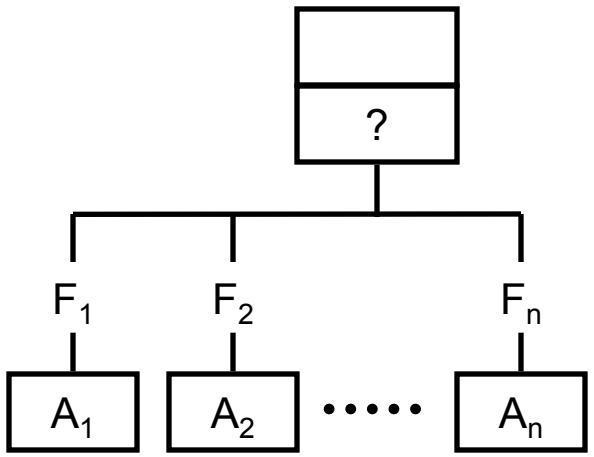




Többszörös szelekció megvalósítása

- A

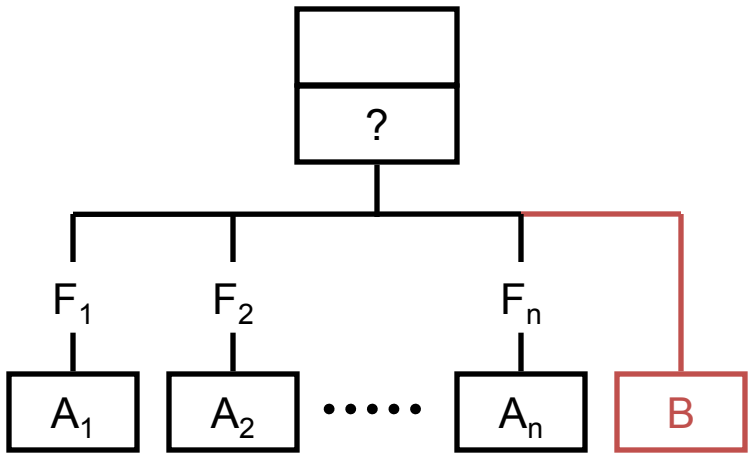
  többszörös szelekciós vezérlés szerkezeti ábrája és C megvalósítása




```
if ( F1 ) {
   A1;
} else if ( F2 )
   {
   A2;
...
} else if ( Fn )
   {
   An;
}
```

Többszörös szelekció megvalósítása

- Az egyébként ággal kiegészített többszörös szelekciós vezérlés szerkezeti ábrája és C megvalósítása




```
if ( F1 ) {
   A1;
} else if ( F2 )
   {
   A2;
...
} else if ( Fn )
   {
   An;
} else {
   B;
}
```

Háromszögek osztályozása

```
/* Milyen háromszöget határoz meg három pozitiv valós szám,
 *   mint a háromszög három oldalhosszúsága?
 *   1997. Október 13. Dévényi Károly
 */

#include <stdio.h>

main()
{
  float A, B, C;     /* a háromszög oldalhosszúságai */
  float M;           /* munkaváltozó a cseréhez      */
  printf("Kérem a három pozitív valós számot!\n");
  scanf("%f%f%f", &A, &B, &C);
```

>>>

Háromszögek osztályozása

```
/* A,B,C átrendezése úgy, hogy A>=B,C legyen */
if (A < B) {                                  /* A és B átrendezése */
    M = A;
    A = B;
    B = M;
}
if (A < C) {                                  /* A és C átrendezése */
    M = A;
    A = C;
    C = M;
}
/* osztályozás */
if (A <= 0 || B <= 0 || C <= 0) {
   printf(" Nem háromszög!\n");                  /* 1. alternatíva */
```

>>>

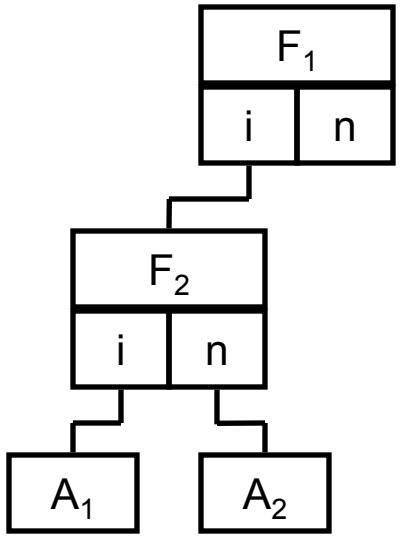
Háromszögek osztályozása

```
  } else if (A >= B + C) {
     printf(" Nem háromszög!\n");            /* 2. alternatíva */
  } else if (A == B && B == C) {
     printf(" Szabályos háromszög.\n");      /* 3. alternatíva */
  } else if (A == B || B == C || A == C) {
     if (A * A == B * B + C * C) {           /* 4. alternatíva */
        printf(" Egyenlőszárú derékszögű háromszög.\n");
     } else {
        printf(" Egyenlőszárú háromszög.\n");
     }
  } else if (A * A == B * B + C * C) {
      printf(" Derékszögű háromszög.\n");    /* 5. alternatíva */
  } else {
      printf(" Egyéb háromszög.\n");             /* egyébként */
  }  /* vége a többszörös szelekciónak */
}
```

if utasítások ismételt alkalmazása

- Az if utasítások ismételt alkalmazása esetén figyelembe kell venni, hogy a következő utasítás a mellette látható vezérlési szerkezet megvalósítása

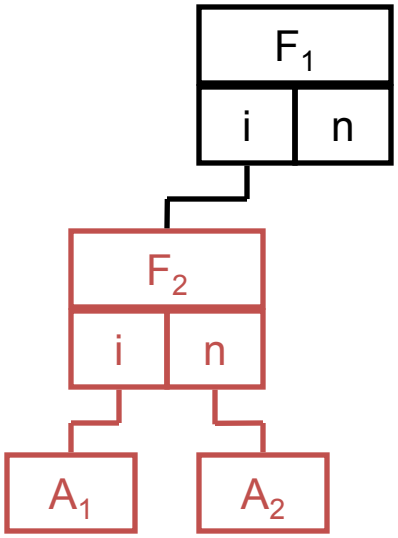
```
if ( F1 )
     if ( F2 )
          A1;
     else
          A2;
```



if utasítások ismételt alkalmazása

- Az if utasítások ismételt alkalmazása esetén figyelembe kell venni, hogy a következő utasítás a mellette látható vezérlési szerkezet megvalósítása

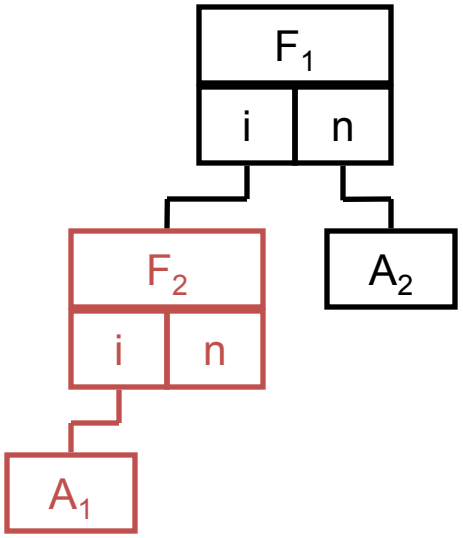
```
if ( F1 ) {
     if ( F2 )
          A1;
     else
          A2;
}
```



if utasítások ismételt alkalmazása

- Az if utasítások ismételt alkalmazása esetén figyelembe kell venni, hogy a következő utasítás a mellette látható vezérlési szerkezet megvalósítása

```
if ( F1 ) {
     if ( F2 )
          A1;
} else {
     A2;
}
```



Esetkiválasztásos szelekciós vezérlés

- A többszörös szelekció egy speciális esete az, amikor minden F_i (1<=i<=n) feltétel így szól:
  - Egy adott K kifejezés aktuális értéke eleme-e az értékek egy előre megadott H_i halmazának?
- Vagyis legyen a K egy szelektor kifejezés, H_i kiválasztó halmazok, A_i pedig tetszőleges műveletek (1<=i<=n).
- Ezekből képzett esetkiválasztásos szelekciós vezérlés a következő vezérlési előírást jelenti:

Esetkiválasztásos szelekciós vezérlés

1.) Értékeljük ki a K kifejezést és folytassuk a 2.) lépéssel.

2.) Adjunk választ a következő kérdésre: Van-e olyan i (1<=i<=n), amelyre teljesül, hogy a kiszámolt érték eleme a H_i halmaznak és nem eleme az összes H_j (1<=j<i) halmaznak? Folytassuk a 3.) lépéssel.

3.) Ha van ilyen i, akkor hajtsuk végre az A_i műveletet és fejezzük be az összetett művelet végrehajtását.

4.) Egyébként, vagyis ha a kiszámolt érték nem eleme a H_i (1<=i<=n) halmazok egyesítésének, akkor fejezzük be az összetett művelet végrehajtását.

Esetkiválasztásos szelekciós vezérlés

- Bővíthetjük az esetkiválasztásos szelekciós vezérlést azzal, hogy a 4.) pontban ne az üres művelet, hanem egy előre megadott B tetszőleges művelet végrehajtását írjuk elő.
- Legyen a K egy szelektor kifejezés, H_i kiválasztó halmazok, A_i és B pedig tetszőleges műveletek (1<=i<=n).
- Ezekből képzett esetkiválasztásos szelekciós vezérlés a következő vezérlési előírást jelenti:

Esetkiválasztásos szelekciós vezérlés

1.) Értékeljük ki a K kifejezést és folytassuk a 2.) lépéssel.

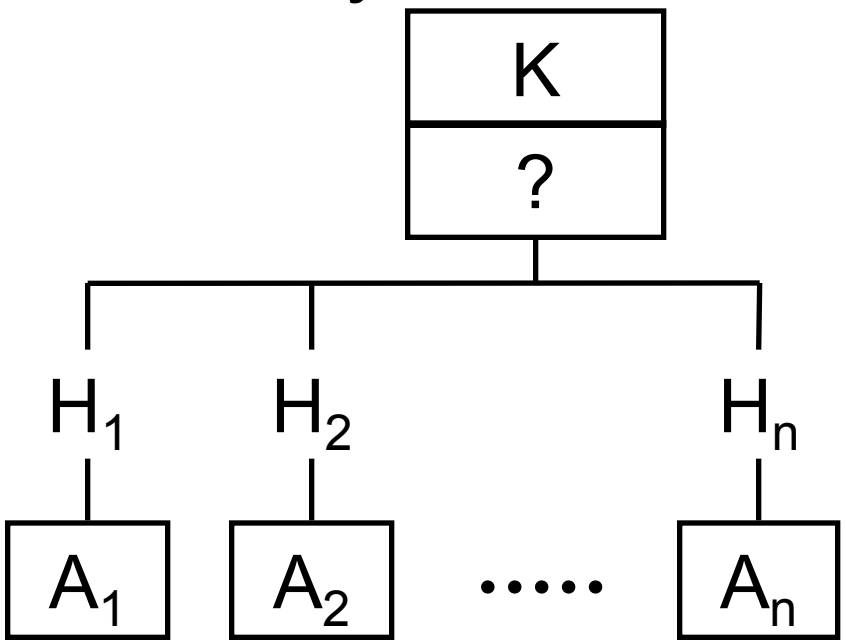
2.) Adjunk választ a következő kérdésre: Van-e olyan i (1<=i<=n), amelyre teljesül, hogy a kiszámolt érték eleme a H_i halmaznak és nem eleme az összes H_j (1<=j<i) halmaznak? Folytassuk a 3.) lépéssel.

3.) Ha van ilyen i, akkor hajtsuk végre az A_i műveletet és fejezzük be az összetett művelet végrehajtását.

4.) Egyébként, vagyis ha a kiszámolt érték nem eleme a H_i (1<=i<=n) halmazok egyesítésének, akkor hajtsuk végre a B műveletet és fejezzük be az összetett művelet végrehajtását.

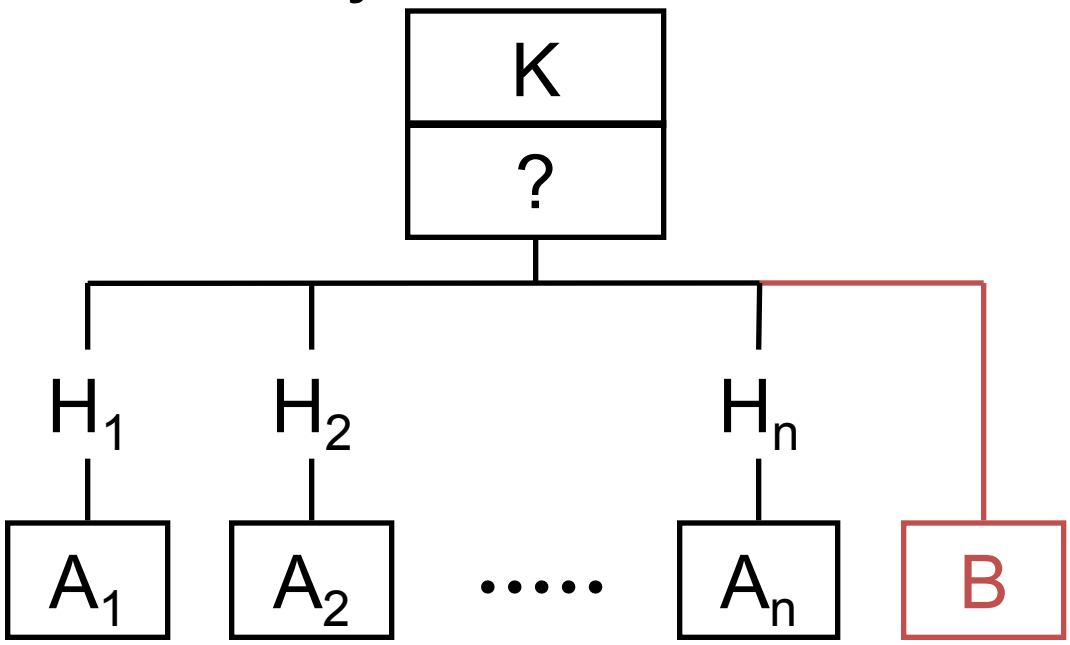
Esetkiválasztásos szelekciós vezérlés

- Az
  esetkiválasztásos szelekciós vezérlés
  szerkezeti ábrája



Esetkiválasztásos szelekciós vezérlés

- Az egyébként ággal kiegészített esetkiválasztásos szelekciós vezérlés szerkezeti ábrája



Esetkiválasztásos szelekciós vezérlés

- A vezérlés lényege
  - Értékeljük ki a K kifejezést.
  - Hajtsuk végre a legelső olyan A_i műveletet, aminek a H_i halmazában benne van a K értéke.
  - Ha K értéke nem eleme egyetlen H_i halmaznak sem, akkor hajtsuk végre a B műveletet.

Esetkiválasztásos szelekciós vezérlés

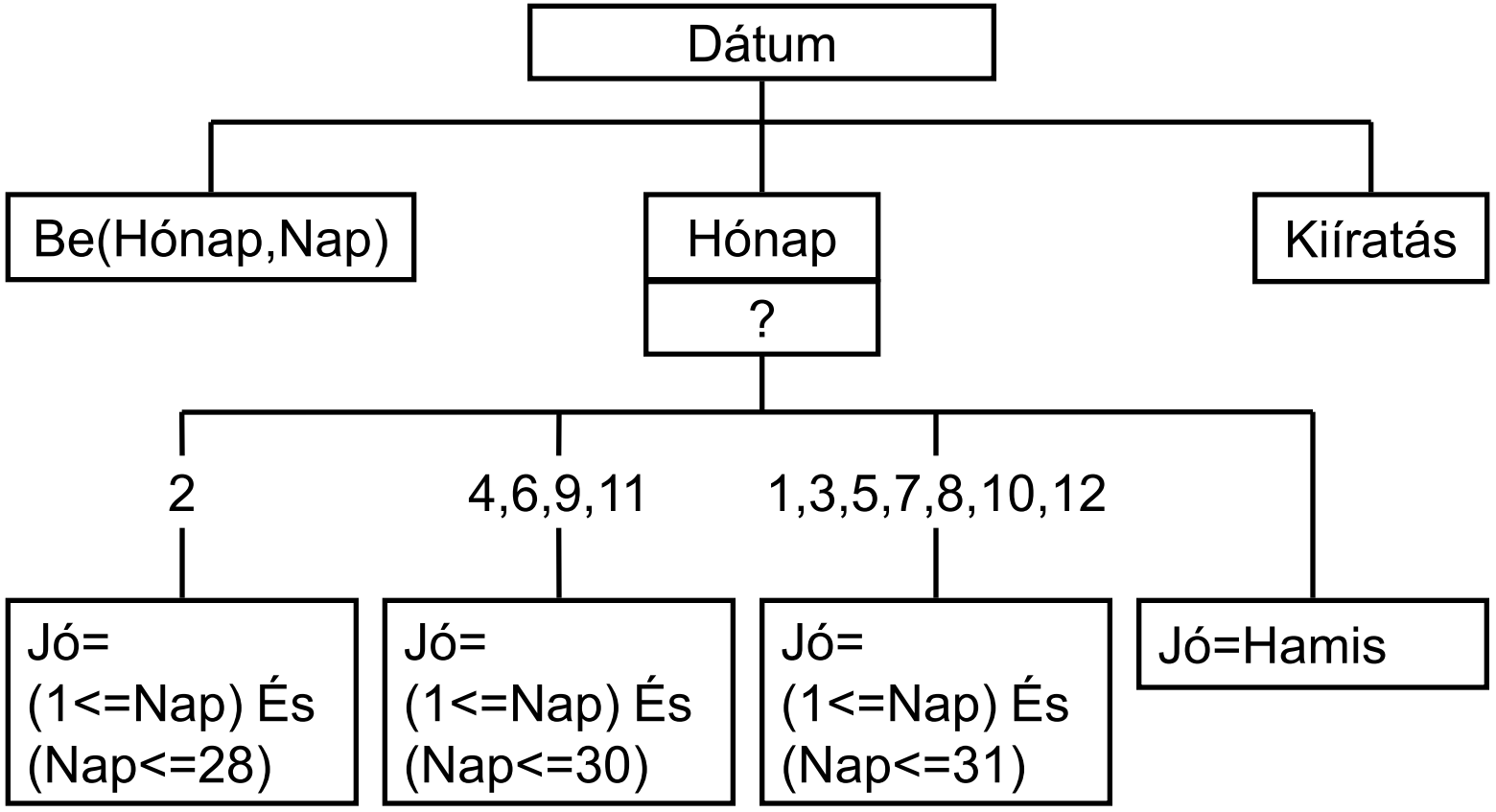
- A kiválasztó halmazok megadása az esetkiválasztásos szelekció kritikus pontja.
- Algoritmusok tervezése során minden effektív halmazmegadást használhatunk, azonban a tényleges megvalósításkor csak a választott programozási nyelv eszközeit alkalmazhatjuk.

Dátum helyessége

- Problémafelvetés
  - Eldöntendő, hogy egy dátumként megadott számpár helyes dátum-e?
- Specifikáció
  - Input
    - Egy (hónap, nap) alakban megadott dátum
  - Output
    - A dátum akkor és csak akkor helyes, ha 1 <= hónap <= 12 és a nap érték is a megfelelő intervallumba esik.

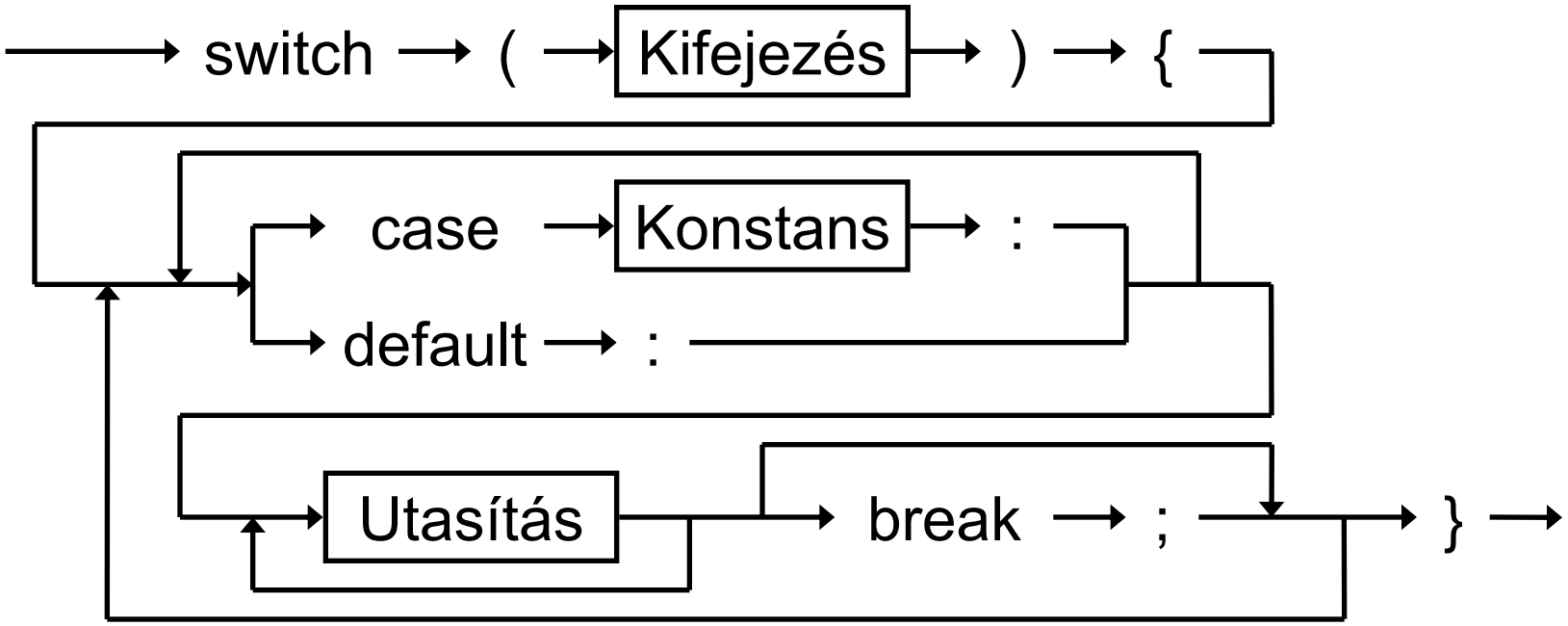
Dátum helyessége

- Algoritmustervezés



A `switch` utasítás

- Ha egy kifejezés értéke alapján többféle utasítás közül kell választanunk
- Szintaxis



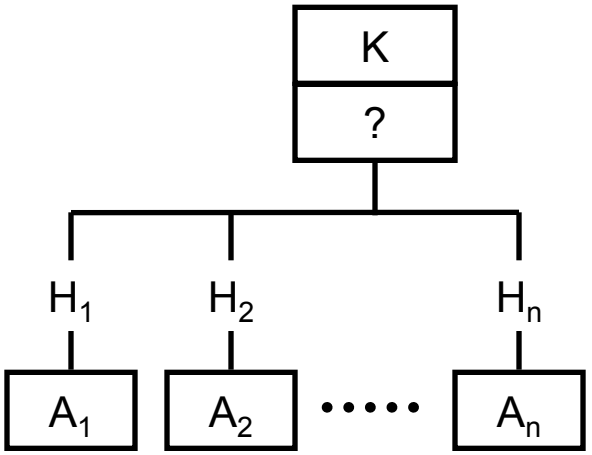
A switch utasítás

- A szelektor kifejezés és a konstansok típusának meg kell egyeznie.
- Egy konstans legfeljebb egyszer szerepelhet **`case`** mögött egy **`switch`** utasításban.
- A **`default`** kulcsszó csak egyszer szerepelhet egy **`switch`** utasításban.

Esetkiv. szelekció megvalósítása

- Az

  esetkiválasztásos szelekciós vezérlés C megvalósítása



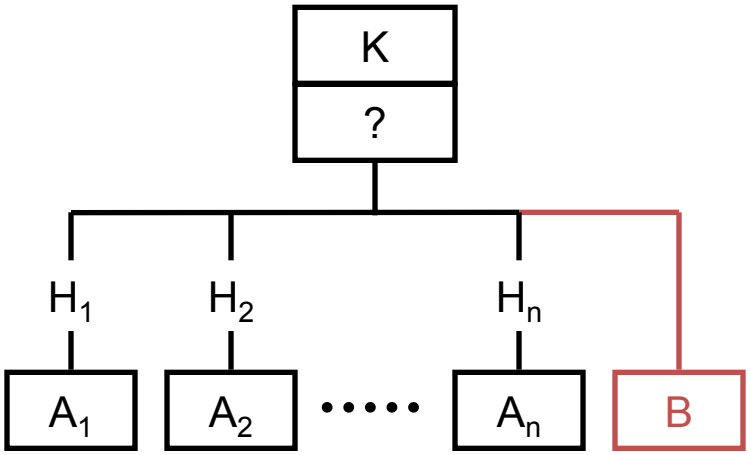

```
switch (K) {
   case H1 :
        A1;
        break;
   ...
   case Hn :
        An;
        break;



}
```

Esetkiv. szelekció megvalósítása

- Az egyébként ággal kiegészített esetkiválasztásos szelekciós vezérlés C megvalósítása




```
switch (K) {
   case H1 :
        A1;
        break;
   ...
   case Hn :
        An;
        break;
   default :
        B;
        break;
}
```

A switch utasítás tulajdonságai

- C-ben a H_1, ..., H_n halmazok csak egyeleműek lehetnek, amelyeket így egyszerűen az elem megadásával jelölhetünk.
- A hatékonyabb kódolás érdekében azonban kihasználhatjuk a **`switch`** utasítás tulajdonságát:
  - A K kifejezés értékétől csak az függ, hogy melyik helyen kezdjük el végrehajtani a **`switch`** magját.
  - Ha a végrehajtás elkezdődik, akkor onnantól kezdve az első **`break`** utasításig, vagy a **`switch`** végéig sorban hajtódnak végre az utasítások.

A switch utasítás tulajdonságai

- Tegyük fel, hogy a H_1 halmaz elemei
  - $x_1, x_2, \ldots, x_m$
- Ezt C-ben a

```
case x1:
case x2:
...
case xm:
          A1;
          break;
```

kódrészlettel tudjuk lekódolni.

A switch utasítás tulajdonságai

- A **`default`** ág olyan, mintha a K kifejezés összes lehetséges de a **`switch`**-ben fel nem sorolt értékét megadnánk egy-egy **`case`** ággal.
- Így a **`default`** ág a **`switch`** magján belül tetszőleges helyen lehet, de csak akkor kezdődik itt a vezérlés, ha a kifejezés aktuális értéke egyetlen **`case`**-ben sem szerepel, beleértve a **`default`** után megadott **`case`**-eket is.
- Éppen ezért a **`default`** ágban is mindig használjunk **`break`** utasítást.

Logikai adattípus C nyelven

- Az algoritmustervezés során használtunk egy „Jó” nevű változót, amely logikai értéket tárolt.
- A C nyelvben nincs logikai típus, de azért logikai értékek persze keletkeznek. Ezeket az értékeket eltárolhatjuk egy **`int`** változóban.
- Ha egy logikai értéket egy **`int`** típusú változóba tettünk, akkor a logikai hamis érték tárolása után az **`int`** típusú változó értéke 0, az igaz érték tárolása után pedig nem 0. (Sok megvalósításban 1, de ezt nem használhatjuk ki, ha gépfüggetlen programot szeretnénk.)

Dátum helyessége

```
/* Eldöntendő, hogy egy dátumként megadott számpár
   helyes dátum-e?
 *   1997. Október 4.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

main()
{
  int Honap, Nap;
  int Jo;                          /* a Boolean érték tárolására */

  printf("Kérem a dátumot (hónap, nap) !\n");
  scanf("%d%d", &Honap, &Nap);

  switch (Honap) {
    case 2:
      Jo = (1 <= Nap && Nap <= 28);
      break;
```

>>>

Dátum helyessége

```
    case 4:
    case 6:
    case 9:
    case 11:
      Jo = (1 <= Nap && Nap <= 30);
      break;
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
      Jo = (1 <= Nap && Nap <= 31);
      break;
```

>>>

Dátum helyessége

```
    default:
      Jo = 0;
      break;
  } /* switch */
                                                    /* Kiíratás  */
  printf("A dátum ");
  if (!Jo) {                              /* Ezt másképpen szokták */
    printf("nem ");
  }
  printf("helyes.\n");
}
```

Feltételes kifejezés

- A feltételes operátor a C nyelv egyetlen háromoperandusú művelete. A K&R könyv feltételes kifejezésnek említi.

```
kif1 ? kif2 : kif3
```

- Először a kif1 kerül kiértékelésre, ha ez
  - Igaz (nem 0), a kifejezés értéke kif2 lesz
  - Hamis (0), a kifejezés értéke kif3 lesz

Feltételes kifejezés

- Az előző programban a kiíratás ez volt:

```
printf("A dátum ");
if (!Jo) {
  printf("nem ");
}
printf("helyes.\n");
```

Feltételes kifejezés

- Ez lerövidíthető:

```
printf("A dátum ");
printf(Jo ? " " : " nem ");
printf("helyes.\n");
```

vagy

```
printf(Jo ?
    "A dátum helyes.\n" :
    "A dátum nem helyes.\n");
```

Feltételes kifejezés

- Illesszük be prioritási sorba az **=** és a **?:** műveleteket!
  - prefix művelet (prefix -, !)
  - multiplikatív műveletek (*, /, %)
  - additív műveletek (+, -)
  - kisebb-nagyobb relációs műveletek (<=, >=, <, >)
  - egyenlő-nem egyenlő relációs műveletek (==, !=)
  - logikai 'és' művelet (&&)
  - logikai 'vagy' művelet (||)
  - feltételes művelet (? :)
  - értékadó művelet (=)

Feltételes kifejezés

- Így a következő programrészlet

```
if (a > b) {
    z = a;
} else {
     z = b;
}
```

átírható így:

```
z = (a > b) ? a : b;
```

vagy akár:

```
z = a > b ? a : b;
```

ISMÉTLÉSES VEZÉRLÉSEK

Ismétléses vezérlések

- Ismétléses vezérlésen olyan vezérlési előírást értünk, amely adott műveletnek adott feltétel szerinti ismételt végrehajtását írja elő.
- Az ismétlési feltétel szerint öt formáját különböztetjük meg az ismétléses vezérléseknek
  1. Kezdőfeltételes
  2. Végfeltételes
  3. Számlálásos
  4. Hurok
  5. Diszkrét

Ismétléses vezérlések

- Az algoritmustervezés során a leginkább megfelelő ismétléses vezérlési formát használjuk, függetlenül attól, hogy a megvalósításra használt programozási nyelvben közvetlenül megvalósítható-e ez a vezérlési mód.
- Ismétléses vezérlés képzését ciklusszervezésnek is nevezik, így az ismétlésben szereplő műveletet ciklusmagnak hívjuk.

Kezdőfeltételes ismétléses vezérlés

- Legyen F logikai kifejezés, M pedig tetszőleges művelet.
- Az F ismétlési feltételből és az M műveletből (a ciklusmagból) képzett kezdőfeltételes ismétléses vezérlés a következő vezérlési előírást jelenti

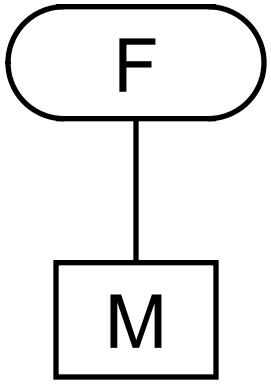
  1.) Értékeljük ki az F feltételt és folytassuk a 2.) lépéssel.

  2.) Ha F értéke hamis, akkor az ismétlés és ezzel együtt az összetett művelet végrehajtása befejeződött.

  3.) Egyébként, vagyis ha az F értéke igaz, akkor hajtsuk végre az M műveletet, majd folytassuk az 1.) lépéssel.

Kezdőfeltételes ismétléses vezérlés

- A kezdőfeltételes ismétléses vezérlés szerkezeti ábrája



Kezdőfeltételes ismétléses vezérlés

- Az algoritmus tervezésekor a ciklusmag olyan részprobléma megoldása is lehet, aminek a megoldását a tervezés adott stádiumában még nem ismerjük. Ekkor a ciklusmag helyére a részprobléma megnevezését írjuk, majd ehhez kapcsoljuk a tervezés során a ciklusmagot megfogalmazó szerkezeti ábrarészt.

Kezdőfeltételes ismétléses vezérlés

- Ha az F értéke hamis, az összetett művelet végrehajtása befejeződik anélkül, hogy az M művelet egyszer is végrehajtásra kerülne
- Ha az F értéke igaz, és az M művelet nincs hatással az F feltételre, akkor F igaz is marad, tehát az összetett művelet végrehajtása nem tud befejeződni. Ilyenkor végtelen ciklus végrehajtását írtuk elő.
- Fontos tehát, hogy az M művelet hatással legyen az F feltételre.

Minimax program

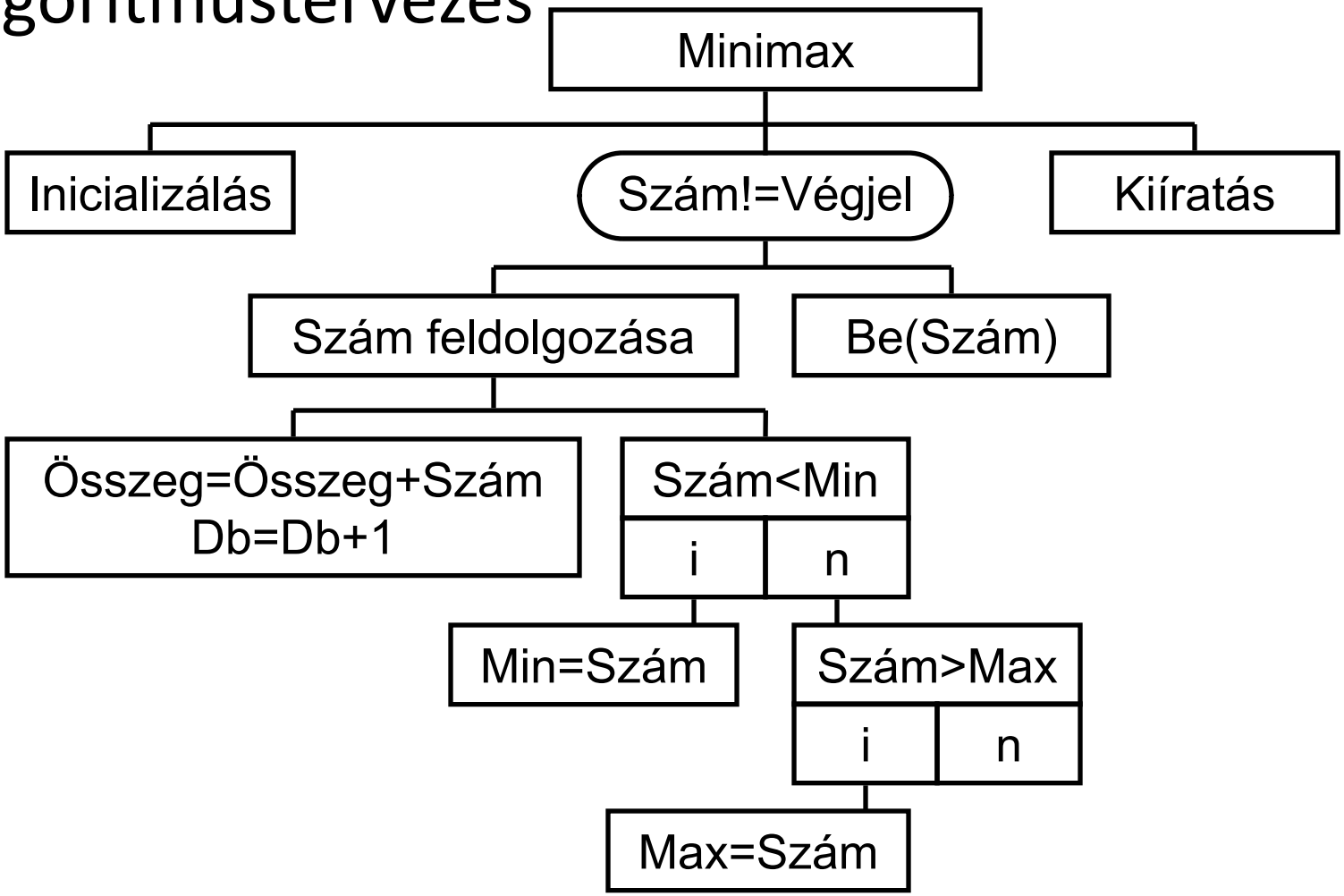
- Problémafelvetés
  - Határozzuk meg egy valós számsorozat legkisebb és legnagyobb elemét, valamint a sorozat átlagát!
- Specifikáció
  - A probléma inputja a valós számsorozat.
  - Az input számsorozat végét egy végjel fogja jelezni, amit a felhasználó ad meg inputként, nyilván a számsorozat előtt.
  - Az output a sorozat legkisebb és legnagyobb eleme, valamint az átlaga.

Minimax program

- Algoritmustervezés
  - Elsőre talán az tűnne a legegyszerűbb megoldásnak, ha beolvasnánk az összes számot, majd ezek között keresgélnénk. Ez a megoldás egy összetett adatszerkezetet (tömböt) igényelne.
  - Ha viszont végiggondoljuk, a „következő” elem beolvasásakor elegendő az eddigi sorozatból csak a lényeges információkat tárolni:
    - a legkisebb elem értékét,
    - a legnagyobb elem értékét,
    - az elemek összegét,
    - és az elemek darabszámát.

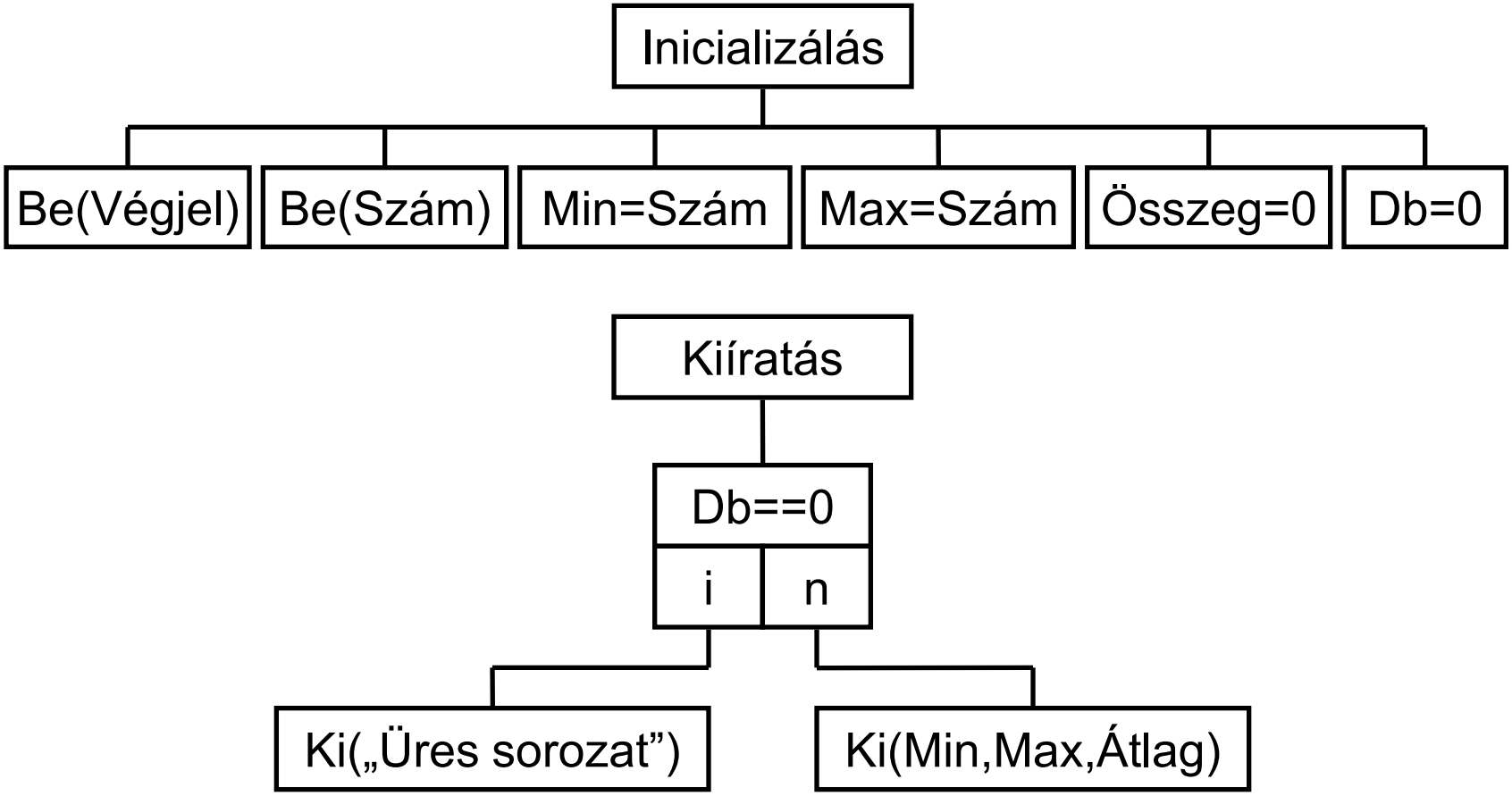
Minimax program

- Algoritmustervezés



Minimax program

- Algoritmustervezés



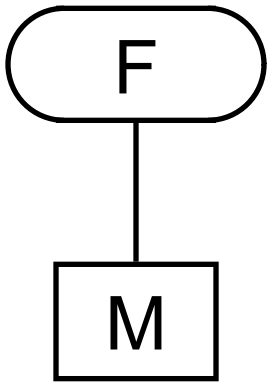
A `while` utasítás

- Ha valamilyen műveletet mindaddig végre kell hajtani, amíg egy feltétel igaz. Ha a feltétel kezdetben hamis volt, a műveletet egyszer sem hajtjuk végre. (A feltétel ellenőrzése a művelet előtt történik.)
- Szintaxis

A kezdőfeltételes vez. megvalósítása

- A kezdőfeltételes ismétléses vezérlés szerkezeti ábrája és C megvalósítása

```
while ( F ) {
     M;
}
```



- Az M utasítás tetszőleges, összetett utasítás lehet.

A C értékadó műveletei

- Mielőtt elhamarkodottan lekódolnánk a szerkezeti ábrán található programot, megismerkedünk a C nyelv azon műveleteivel, amelyek használata tömörré teszi a programok kódját.

Inkrementáló és dekrementáló műv.

- A C nyelv tartalmaz két operátort, amelyekkel változók inkrementálhatók és dekrementálhatók.
  - A **++** inkrementáló operátor az operandus értékét 1-el növeli
  - A **--** dekrementáló operátor az operandus értékét 1-el csökkenti
- A **++** és a **--** egyaránt használható
  - prefix operátorként (**++i**, **--i**)
  - postfix operátorként (**i++**, **i--**)

Inkrementáló és dekrementáló műv.

- Az **`i`** változó a prefix és postfix használat esetén is pontosan eggyel nő (csökken)
- A különbség a kétféle használat között az, hogy
  - prefix művelet esetén a **`++i`** (**`--i`**) kifejezés értéke az **`i`** változó új, azaz eggyel megnövelt (csökkentett) értéke
  - postfix esetben az **`i++`** (**`i--`**) kifejezés értéke az **`i`** eredeti értéke
- Tehát ha nem csak a művelet inkrementáló (dekrementáló) tulajdonságát, hanem a kifejezés értékét is felhasználjuk, akkor a pre- és postfix használat között különbség van.

Inkrementáló és dekrementáló műv.

- Például, ha i értéke 5, akkor

```
x = i++;
```

az x-et 5-re állítja, de

```
x = ++i;
```

x-et 6-ra állítja. **i** értéke mindkét esetben 6 lesz.

- Ezek az operátorok csak változókra (l-value) alkalmazhatók; az olyan kifejezés, mint

```
(i + j)++
```

nem megengedett!

Értékadó műveletek és kifejezések

- Az olyan kifejezések, mint például

```
i = i + 2
```

  a **+=** értékadó operátor segítségével az

```
i += 2
```

  tömörített alakban is írhatók.
- A C-ben a legtöbb operátornak megvan az

  op=

  alakú megfelelője, ahol az op egy műveleti szimbólum. Az eddig megismertek közül op lehet:

  + - * / %

Értékadó műveletek és kifejezések

- Ha e1 és e2 kifejezés, akkor

  `e1 op= e2`

  jelentése

  `e1 = (e1) op (e2)`

- Ügyeljünk az **`e2`** körüli zárójelekre:

```
x *= y + 1
```

jelentése

```
x = x * (y + 1)
```

nem pedig

```
x = (x * y) + 1
```

Értékadó műveletek és kifejezések

- Illesszük be a prioritási sorba a **++** és **−−** valamint az értékadó műveleteket!
  - a egyoperandusú műveletek (prefix -, ++, --, !)
  - a multiplikatív műveletek (*, /, %)
  - az additív műveletek (+, -)
  - a kisebb-nagyobb relációs műveletek (<=, >=, <, >)
  - az egyenlő-nem egyenlő relációs műveletek (==, !=)
  - a logikai 'és' művelet (&&)
  - a logikai 'vagy' művelet (||)
  - a feltételes művelet (? :)
  - értékadó művelet (=, +=, -=, *=, /=, %=)

Értékadó műveletek és kifejezések

- Az értékadó műveletek természetesen mind jobb-asszociatívak.
- Óvatosan és csak a céljának megfelelően szabad használni ezeket a műveleteket. Nézzünk néhány példát!

Értékadó műveletek és kifejezések

- Mi lesz a kiírt érték?

```
int i = 7;
printf("%d\n", i-- * i++);
```

- Nálunk ez 49, de lehetne 42 vagy 56 is.
- Az ANSI szabvány szerint az aritmetikai részkifejezések kiértékelésének sorrendje tetszőleges.
- Tehát i értékét csak akkor inkrementáljuk, ha i sehol máshol nem szerepel az egész kifejezésben.

Minimax program

```
/* Határozzuk meg egy valós számsorozat legkisebb
 * és legnagyobb elemét, valamint a sorozat átlagát!
 * 1997. Október 25.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

main()
{
  double Vegjel, Szam, Osszeg, Min, Max, Atlag;
  int Db;                          /* az összegzett elemek száma */
  printf("Ez a program valós számsorozat minimális,\n");
  printf("maximális elemét és átlagát számolja.\n");
  printf("Az input sorozatot végjel zárja.\n");
  printf("Kérem a végjelet!\n");              /* inicializálás */
  scanf("%lf", &Vegjel);
```

>>>

Minimax program

```
printf("Kérem az input számsorozatot!\n");
printf("? ");
scanf("%lf", &Szam);
Min = Max = Szam;
Osszeg = 0.0;
Db = 0;
while (Szam != Vegjel) {                    /* a ciklus kezdete */
    Osszeg += Szam;                             /* összegzés */
    Db++;                                   /* számláló növelés */
    if (Szam < Min) {                       /* min-max számítás */
        Min = Szam;
    } else if (Szam > Max) {
        Max = Szam;
    }
                                /* a következő szám beolvasása */
    printf("? ");
    scanf("%lf", &Szam);
}                                               /* a ciklus vége */
```

>>>

Minimax program

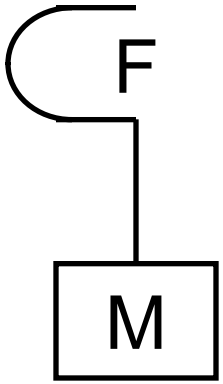
```
  if (Db == 0) {
      printf("Üres számsorozat érkezett.\n");
  } else {
      Atlag = Osszeg / Db;
      printf("Minimum  = %10.3f Maximum= %10.3f\n", Min, Max);
      printf("Az átlag = %10.3f\n", Atlag);
  }
}
```

Végfeltételes ismétléses vezérlés

- Az F ismétlési feltételből és M műveletből (ciklusmagból) képzett végfeltételes ismétléses vezérlés a következő vezérlési előírást jelenti
  1.) Hajtsuk végre az M műveletet majd folytassuk a 2.) lépéssel.
  2.) Értékeljük ki az F feltételt és folytassuk a 3.) lépéssel.
  3.) Ha F értéke igaz, akkor az ismétléses vezérlés és ezzel együtt az összetett művelet végrehajtása befejeződött.
  4.) Egyébként, vagyis ha az F értéke hamis, akkor folytassuk az 1.) lépéssel.

Végfeltételes ismétléses vezérlés

- A végfeltételes ismétléses vezérlés szerkezeti ábrája



Végfeltételes ismétléses vezérlés

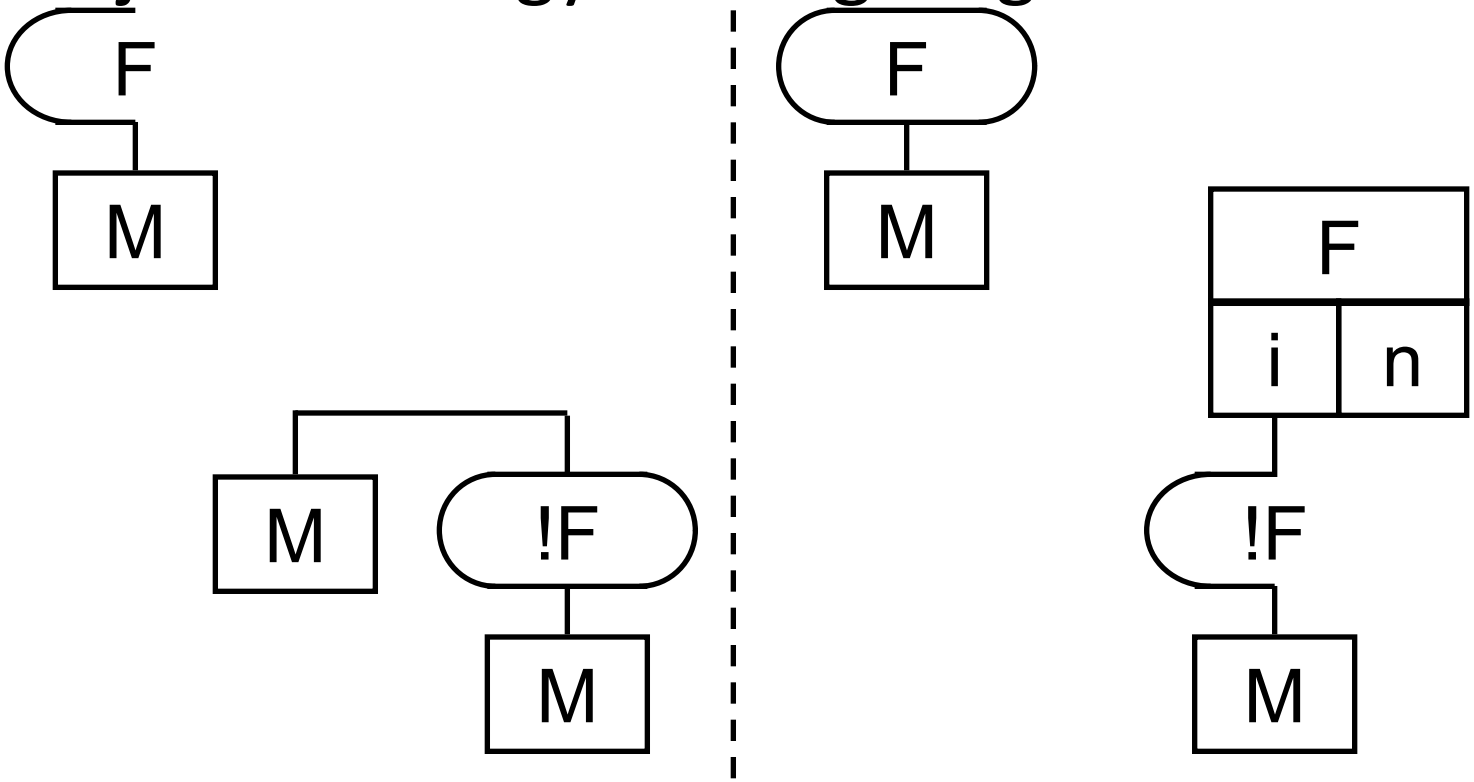
- Látható, hogy a végfeltételes ismétléses vezérlés alapvetően abban különbözik a kezdőfeltételes ismétléses vezérléstől, hogy a ciklusmag legalább egyszer végrehajtódik.

Végfeltételes ismétléses vezérlés

- Ha az M művelet nincs hatással az F feltételre, akkor
  - Ha az F értéke igaz, igaz is marad, így az M művelet egyszer került végrehajtásra és az összetett művelet végrehajtása befejeződik
  - Ha az F értéke hamis, hamis is marad, tehát az összetett művelet végrehajtása nem tud befejeződni. Ilyenkor végtelen ciklus végrehajtását írtuk elő
- Fontos tehát, hogy az M művelet hatással legyen az F feltételre.

Ismétléses vezérlések kapcsolata

- A kezdő és végfeltételes ismétléses vezérlések kifejezhetőek egymás segítségével.



Ismétléses vezérlések kapcsolata

- Az algoritmus tervezésekor előfordulhat, hogy mind a kezdőfeltételes ismétléses vezérlés, mind a végfeltételes ismétléses vezérlés alkalmasnak látszik a probléma megoldására.
- Ilyenkor érdemes megvizsgálni, hogy az F feltétel szükséges feltétele-e az M művelet végrehajtásának?
- Ha igen, akkor a kezdőfeltételes ismétléses vezérlést kell választani.

Szinusz

- Problémafelvetés
  - Szinusz(x) közelítő értékének kiszámítása
- Specifikáció
  - Input
    - X valós szám
  - Outputja
    - sin(X)
  - Nem hívhatjuk meg a C standard sin(x) függvényét

Szinusz

- Algoritmustervezés
  - Ismeretes, hogy sin(x) értéke az
    - $x - x^3/3! + x^5/5! - \ldots + (-1)^{(i-1)} * x^{(2i-1)}/(2i-1)! + \ldots$

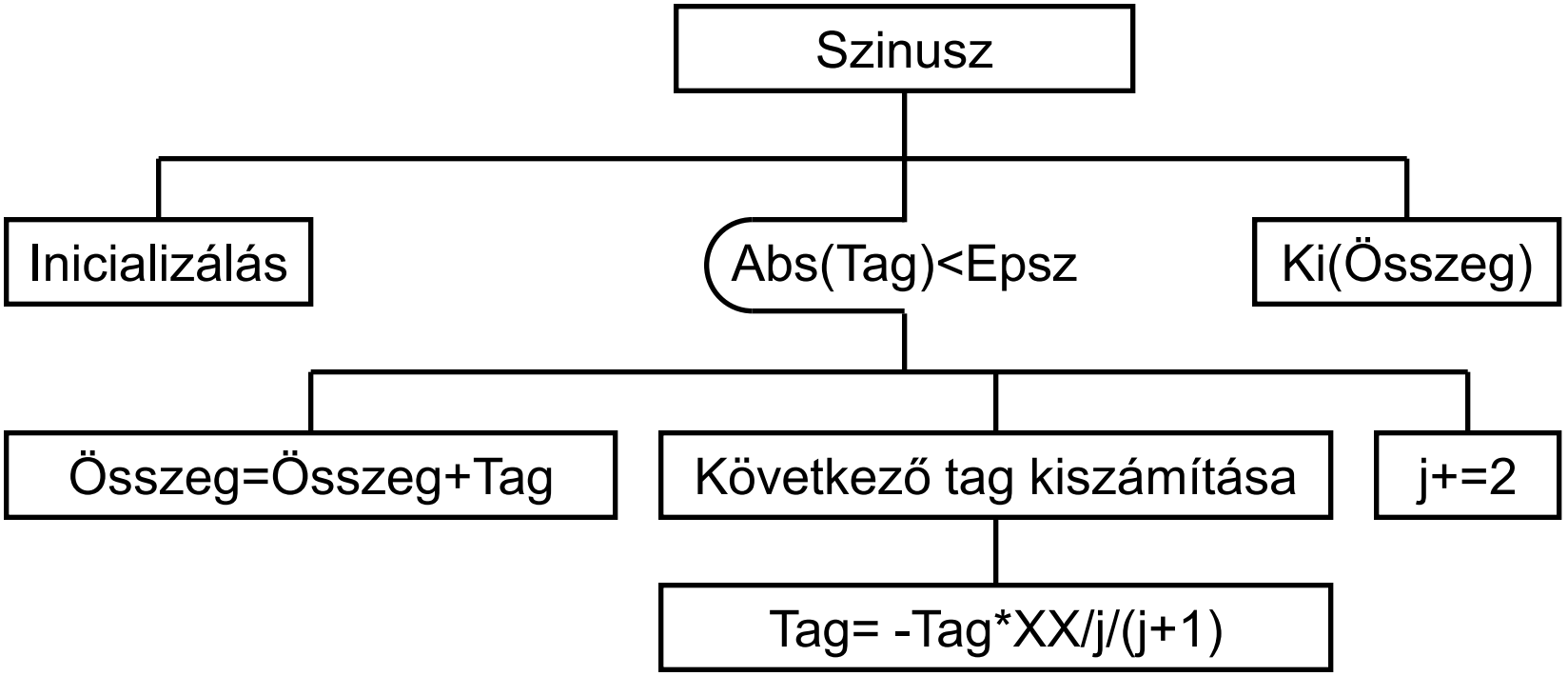
    végtelen sor összege.
  - Ezen végtelen sor kezdőszeletének összegével közelítjük sin(x) értékét, úgy, hogy az összegzés befejeződik, ha az utolsó tag abszolút értéke kisebb, mint a konstansként megadott epsz pontosság.

Szinusz

- Algoritmustervezés
  - Nyilvánvaló, hogy nem célszerű a számolást úgy szervezni, hogy i minden értékére külön kiszámoljuk a tagot, hiszen az i. tag kiszámolható az (i-1)-ik tagból.
  - A tag számlálójának és nevezőjének külön számolása egyébként is pontatlanná tenné a számítást, mert mindkettő, különösen a nevező rohamosan növekedik i függvényében.

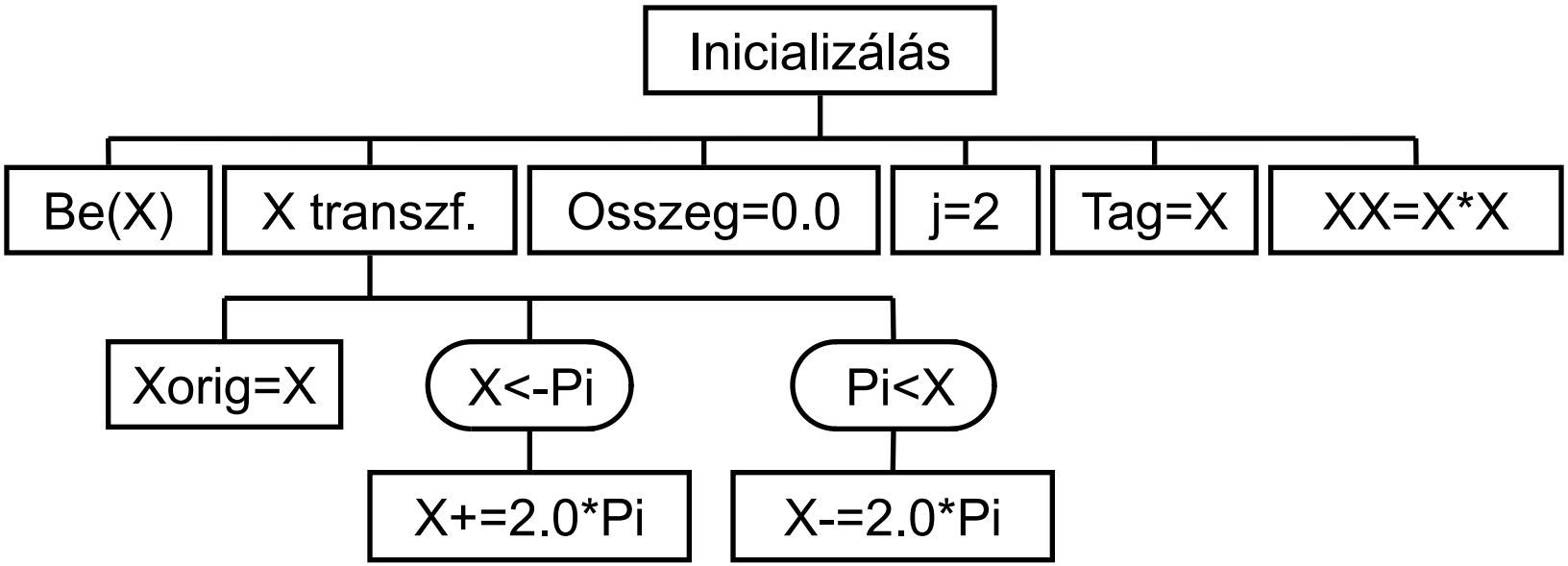
Szinusz

- Struktúradiagram



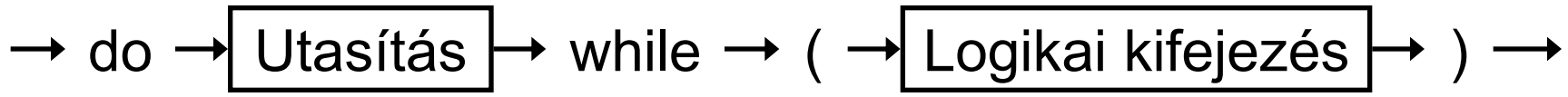
Szinusz

- A **`float`** típus tulajdonságai miatt nem érdemes abszolút értékben nagy X értékekkel számolni. Kihasználjuk, hogy a sin(x) függvény periodikus.



A `do while` utasítás

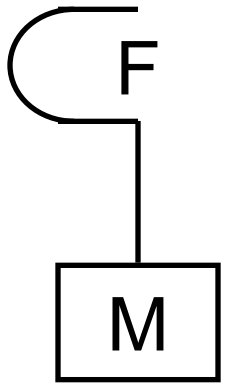
- Ha valamilyen műveletet mindaddig végre kell hajtani, amíg egy feltétel igaz. A műveletet legalább egyszer végrehajtjuk akkor is, ha a feltétel kezdetben hamis volt. (A feltétel ellenőrzése a művelet után történik.)
- Szintaxis



A végfeltételes vez. megvalósítása

- A végfeltételes ismétléses vezérlés szerkezeti ábrája és C megvalósítása

```
do {
     M;
} while ( !F )
```



- Az M utasítás tetszőleges, összetett utasítás lehet.

Szinusz

```
/* sin(x) közelítő értékének kiszámítása a beépített sin(x)
 * függvény alkalmazása nélkül.
 * X értékét transzformáljuk a (-Pi,Pi) intervallumba.
 * 1997. Október 25.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */
#include <stdio.h>
#include <math.h>

#define EPSZ    1e-10                   /* a közelítés pontossága */

main()
{
  double Osszeg;     /* a végtelen sor kezdőösszege */
  double Tag;        /* a végtelen sor aktuális tagja */
  double X;          /* argumentum */
  double Xorig;      /* az argumentum értékének megőrzése*/
  double XX;         /* sqr(x) */
  int j;             /* a nevező kiszámításához */
```

>>>

Szinusz

```
printf("Kérem sin(x)-hez az argumentumot\n");
scanf("%lg%*[^\n]", &X);
getchar();                                  /* ez most már ReadLn */
Xorig = X;

while (X < -M_PI) {                         /* transzformálás */
    X += 2 * M_PI;
}
while (M_PI < X) {
    X -= 2 * M_PI;
}

Osszeg = 0.0;
j = 2;                                      /* inicializálás */

Tag = X;
XX = X * X;
```

Szinusz

```
  do {                                                /* ciklus kezdete */
    Osszeg += Tag;
    Tag = -(Tag * XX / j / (j + 1));                   /* következő tag */
    j += 2;
  } while (fabs(Tag) >= EPSZ);     /* végfeltétel, ciklus vége */
  printf("sin(%8.5f)= %13.10f\n", Xorig, Osszeg);
}
```

Függvények előnyei

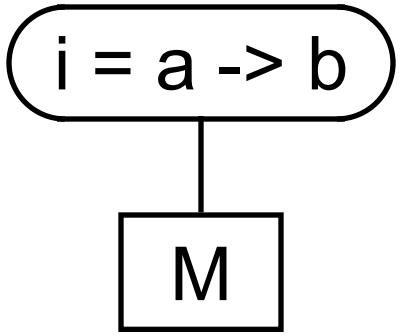
- Függvények használata programozás során a következő előnyöket biztosítja
  - Többszörös felhasználás. Hasonló részproblémák megoldására elég egy függvényt készíteni és a különböző adatokra végrehajtatni a részalgoritmust. Így a program méretét csökkenteni lehet.
  - Memória igény csökkentése. Az függvények lokális változói számára csak az függvény végrehajtása idejére foglalódik memória.
  - Függvények használatával a program áttekinthetőbb lesz. A tervezés során a részproblémák függvénnyel történő megoldása lehetővé teszi a figyelem lokalizálását.
  - Függvények alkalmazása megkönnyíti a bizonyítást, a program tesztelését, a hibakeresést, a javítást és a program módosítását.

Számlálásos ismétléses vezérlés

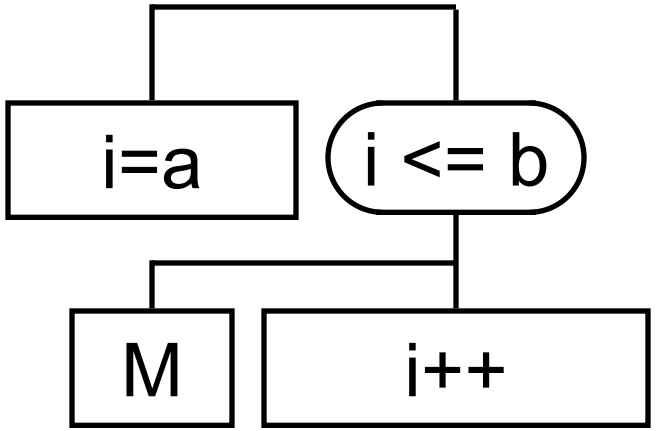
- Számlálásos ismétléses vezérlésről beszélünk, ha olyan ismétlést írunk elő, amely szerint a ciklusmagot végre kell hajtani egy változó minden olyan értékére (növekvő vagy csökkenő sorrendben), amely egy adott intervallumba esik.
- Legyen
  - a és b valamely egész érték
  - i egész típusú változó
  - M tetszőleges művelet

Növekvő számlálásos vezérlés

- Szerkezeti ábra



- A növekvő számlálásos ismétléses vezérlés a következő vezérlési előírást jelenti

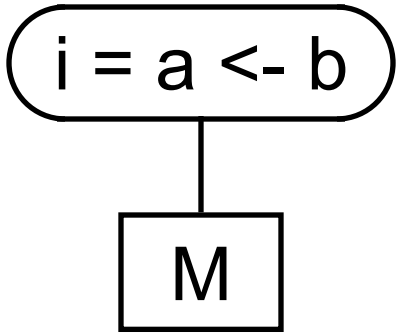


Növekvő számlálásos vezérlés

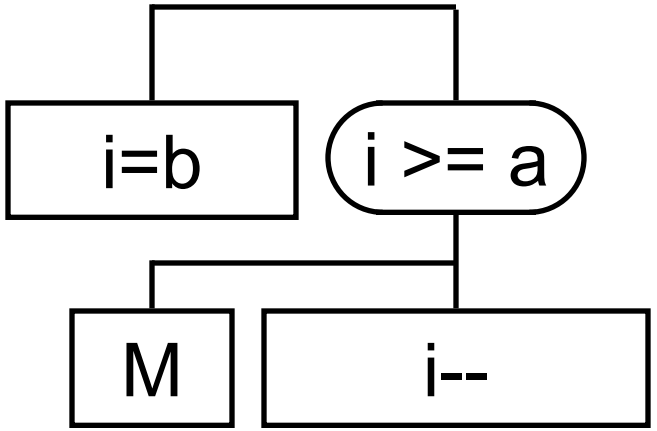
- Szokásos elnevezések
  - i változó, a ciklusváltozó
  - a kifejezés, az ismétlés kezdőértéke
  - b kifejezés, az ismétlés végértéke
  - M utasítás (művelet), a ciklusmag
  - i = a -> b az ismétlési előírás
- Fontos, hogy az M művelet nem lehet hatással az ismétlési előírás egyik elemére sem.

Csökkenő számlálásos vezérlés

- Szerkezeti ábra



- A csökkenő számlálásos ismétléses vezérlés a következő vezérlési előírást jelenti



Csökkenő számlálásos vezérlés

- Szokásos elnevezések
  - i változó, a ciklusváltozó
  - b kifejezés, az ismétlés kezdőértéke
  - a kifejezés, az ismétlés végértéke
  - M utasítás (művelet), a ciklusmag
  - i = a <- b az ismétlési előírás
- Fontos, hogy az M művelet nem lehet hatással az ismétlési előírás egyik elemére sem.

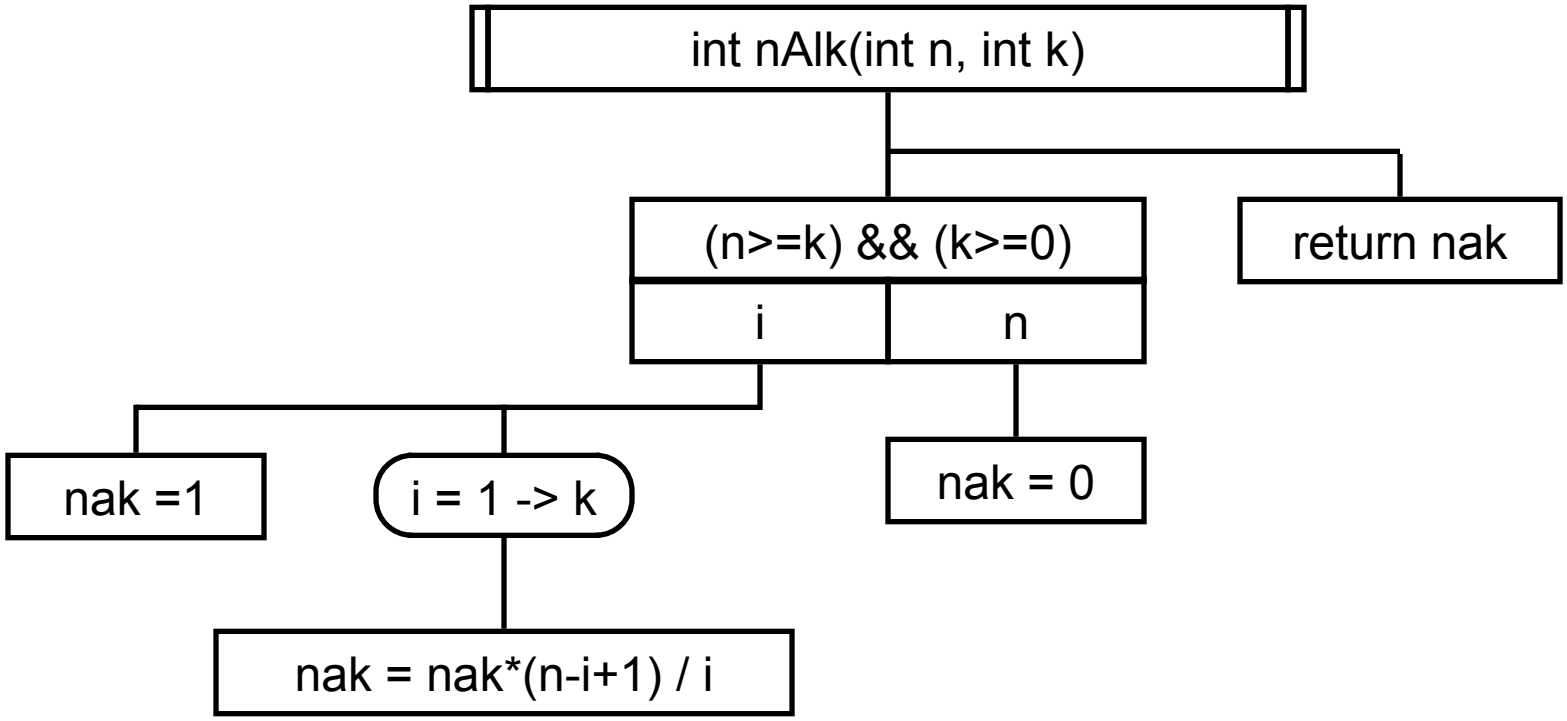
n alatt k

- Problémafelvetés
  - n alatt k kiszámítása
- Specifikáció
  - Input
    - n és k nemnegatív egész számok
  - Output
    - n alatt k értéke

n alatt k

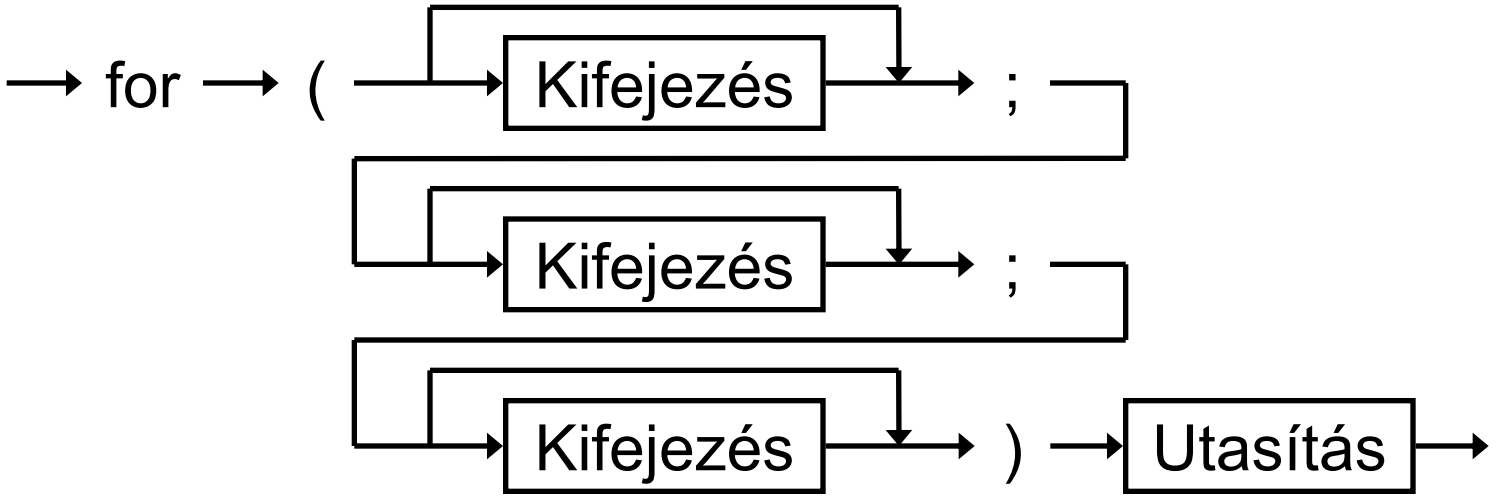
- Algoritmustervezés
  - Készítsünk az n alatt k kiszámolására egy függvénydeklarációt
  - Ismeretes, hogy n alatt k = n!/(k!*(n-k)!). Nyilvánvaló, hogy nem célszerű a számolást úgy szervezni, hogy a faktoriálisokat egyenként kiszámoljuk, hiszen ezek nagy értékek lehetnek és így kiléphetünk az Integer értékkészletből.
  - A nevező egyik tényezőjével egyszerűsítve egy törtet kapunk, amelynek a számlálója is és a nevezője is azonos számú tényezőből álló szorzat. Ez indokolja a számlálásos ismétléses vezérlés alkalmazását.

n alatt k


int nAlk(int n, int k)
(n>=k) && (k>=0)
return nak
i
n
nak =1
i = 1 -> k
nak = 0
nak = nak*(n-i+1) / i

A `for` utasítás

- Ha valamilyen műveletet sorban több értékére is végre kell hajtani, akkor ezt érdemes használni.
- Szintaxis



A `for` utasítás

- A C-ben a **`for`** utasítás általános alakja így néz ki:

```
for (kif1; kif2; kif3) utasítás
```

ami egyenértékű a

```
kif1;
while (kif2) {
     utasítás
     kif3;
}
```

alakkal.

A `for` utasítás

- Többnyire *kif1* és *kif3* értékadás vagy függvényhívás, *kif2* pedig relációs kifejezés.
- A három kifejezés bármelyike elhagyható, de a pontosvesszőknek meg kell maradniuk.
- Ha *kif1* vagy *kif3* marad el, akkor a ; egyszerűen elmarad a kifejtésből. Ha a *kif2* vizsgálat nem szerepel, akkor állandóan igaznak tekintjük, így

```
for (;;) { /* ... */ }
```

végtelen ciklus, amelyből más módon kell kiugrani (pl. **`return`** vagy **`break`** révén).

A , művelet

- Előfordulhat, hogy a ciklus előkészítése nem egy kifejezés kiértékeléséből áll, illetve minden egyes ciklusmenetben több adminisztratív lépés is lehet.
- A **`for`** utasításban ekkor alkalmazhatjuk a `,` műveletet, (sequential expression) amely végülis szekvenciális vezérlést ír elő.

A , művelet

- A

```
for (kif11, kif12, kif13; kif2;
        kif31, kif32, kif33) {
            utasítás
   }
```

egyenértékű a

```
kif11; kif12; kif13;
while (kif2) {
    utasítás
    kif31; kif32; kif33;
}
```

utasítással.

A , művelet

- A , műveletet nyilvánvalóan a prioritási sor aljára kell helyeznünk
  - a egyoperandusú műveletek (-, ++, --, !, ~)
  - a multiplikatív műveletek (*, /, %)
  - az additív műveletek (+, -)
  - bitléptetés (<<, >>)
  - a kisebb-nagyobb relációs műveletek (<=, >=, <, >)
  - az egyenlő-nem egyenlő relációs műveletek (==, !=)
  - bitenkénti 'és' művelet (&)
  - bitenkénti 'kizáró vagy' művelet (^)
  - bitenkénti 'vagy' művelet (|)
  - a logikai 'és' művelet (&&)
  - a logikai 'vagy' művelet (||)
  - a feltételes művelet (? :)
  - értékadó művelet (=, +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)
  - szekvencia művelet (,)

Ábécé

- A feladat: írassuk ki felváltva az ábécé kis és nagybetűit
- Egy karakter tárolására C-ben a **`char`** típus használható.

Ábécé

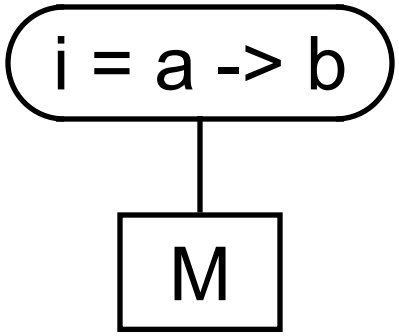
```
/* Az ábécé kis- és nagybetűinek kiíratása összefésülve.
 * A for ciklusban a , műveletet alkalmazzuk.
 * 1997. November 7.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */
#include <stdio.h>

main()
{
  char cha;                          /* az ábécé kisbetűinek */
  char chA;                          /* az ábécé nagybetűinek */

  for ( cha = 'a', chA = 'A'; cha <= 'z'; cha++, chA++ ) {
      printf ( "%c%c", cha, chA );
  }
  printf ( "\n" );
}
```

A for utasítás

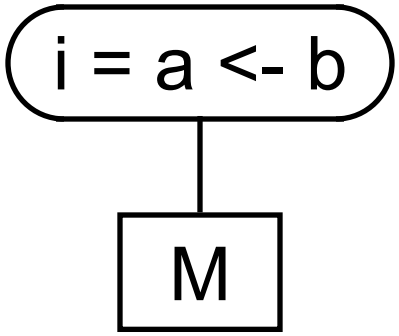
- A növekvő számlálásos ismétléses vezérlés szerkezeti ábrája és megvalósítása C-ben




```
for (i = a; i <= b; ++i) {
    M;
}
```

A for utasítás

- A csökkenő számlálásos ismétléses vezérlés szerkezeti ábrája és megvalósítása C-ben




```
for (i = b; i >= a; --i) {
    M;
}
```

n alatt k

```
int nAlk(int n, int k)
{ /* n alatt k értéke nem rekurzív függvénnyel */
  int i, nak;
  if (n >= k && k >= 0) {          /* input adatok jók-e? */
    nak = 1;                            /* inicializálás */
    for (i = 1; i <= k; i++) {                /* ciklus */
      nak = nak * (n - i + 1) / i;
    }
  } else {
      nak = 0;
  }
  return nak;
}
```

Hurok ismétléses vezérlés

- Az ismétléses vezérlésnek azt a módját, amikor a ciklusmag ismétlését a ciklusmagon belül vezéreljük, úgy, hogy ha adott feltétel, a kijárati feltétel teljesül, akkor a ciklusmag és ezzel együtt az összetett művelet végrehajtása befejeződik, hurok ismétléses vezérlésnek nevezzük.
- A ciklusmagban több kijárati feltételt is megadhatunk.
- Legyenek F_i (1<=i<=n) logikai kifejezések, K_i és M_i (0<=i<=n) pedig tetszőleges, (esetleg üres) műveletek.

Hurok ismétléses vezérlés

- Az F_i kijárati feltételekből, a K_i kijárati műveletekből és az M_i műveletekből képzett hurok ismétléses vezérlés a következő vezérlési előírást jelenti:

  1.) A ciklusmag egységei szekvenciális vezérlést képeznek a felírásuk sorrendjében.

  2.) Az ismétléses vezérlés a ciklusmag első egységének végrehajtásával kezdődik.

Hurok ismétléses vezérlés

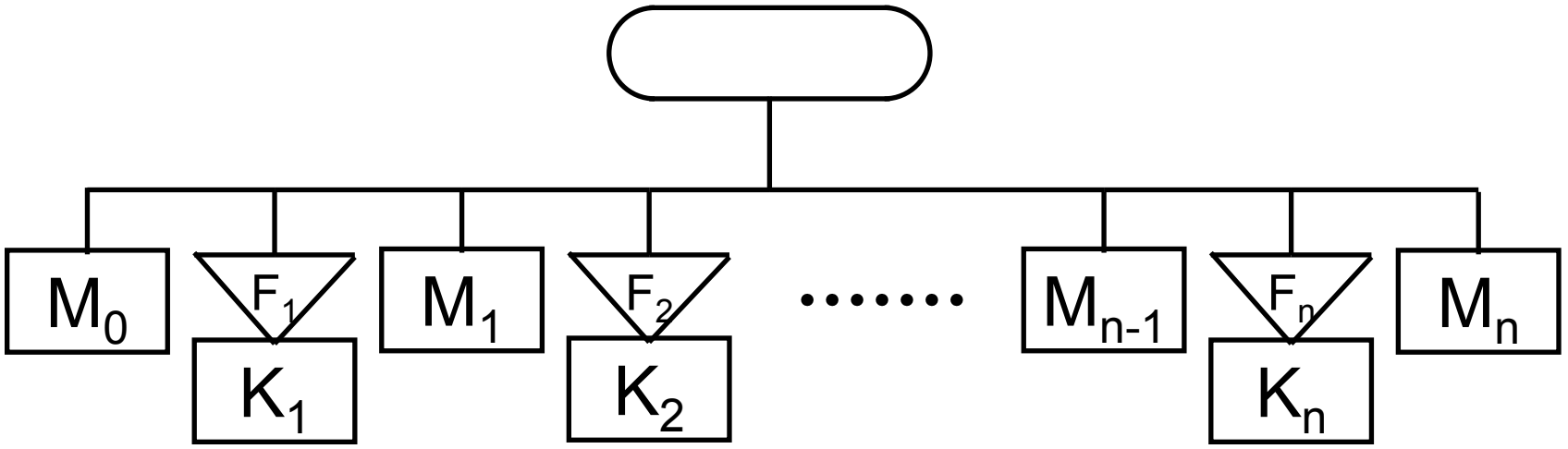
3.) A ciklusmag i-dik egységének végrehajtása azt jelenti, hogy

- Ha az művelet, akkor végrehajtódik a művelet, és a ciklusmag következő egysége kap vezérlést.
- Ha az kijárat és a kijárati feltétel igaz, akkor a K_i kijárati művelet végrehajtódik és a hurok ismétléses vezérlés végrehajtása véget ér
- Ha az kijárat és a kijárati feltétel hamis, akkor a kijárat hatástalan és a ciklusmag következő egysége kap vezérlést.

4.) Ha a ciklusmag utolsó egységének végrehajtása után, az az előírás, hogy a következő egység kap vezérlést, akkor a ciklusmag első egysége kap vezérlést.

Hurok ismétléses vezérlés

- Szerkezeti ábra

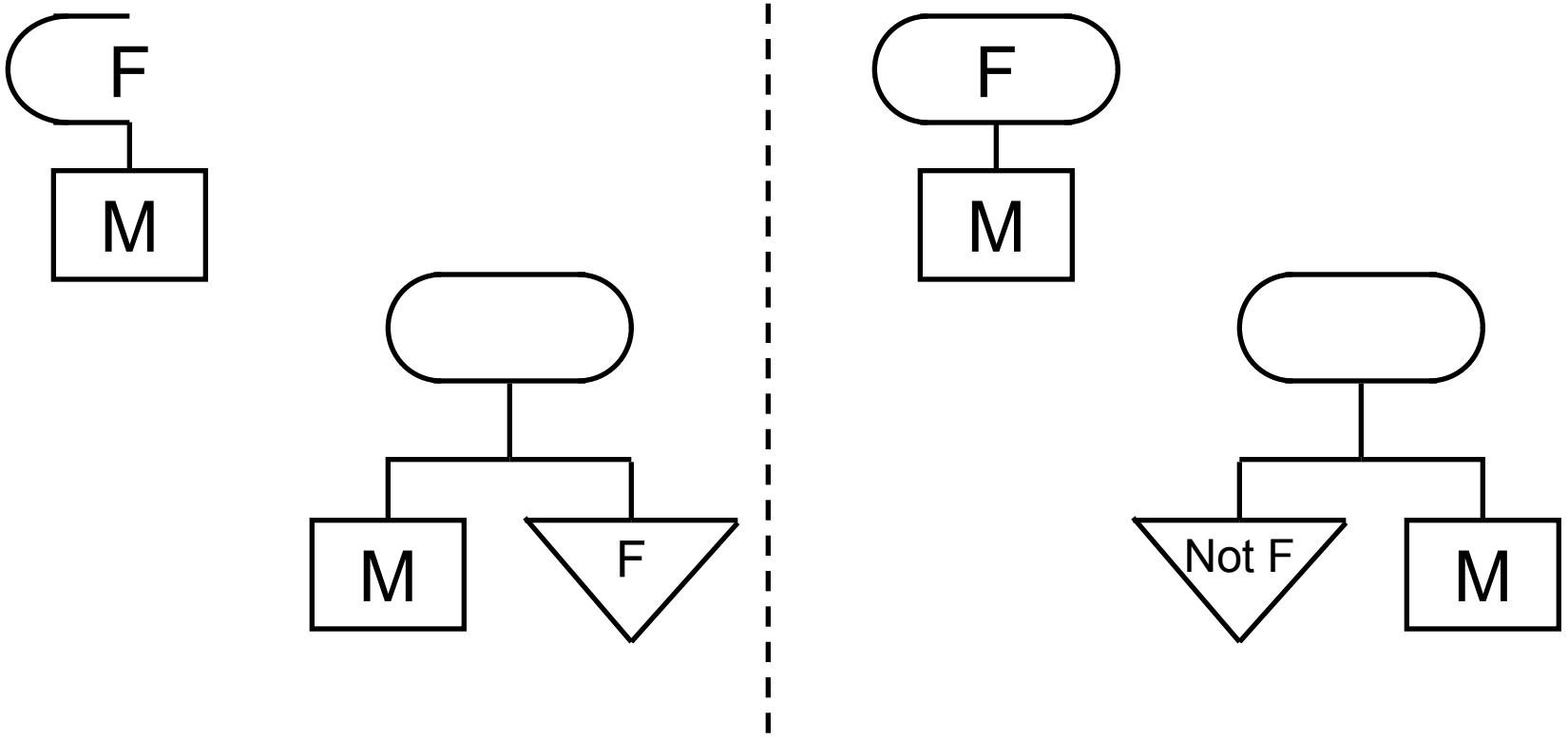


Hurok ismétléses vezérlés

- Ha egyetlen M_i művelet sincs hatással egyetlen feltételre sem, akkor
  - Vagy van olyan F_j, amelynek értéke igaz, és így az összetett művelet végrehajtása befejeződik mielőtt minden M_i művelet egyszer is végrehajtásra kerülne.
  - Vagy az összes F_j értéke hamis, és mivel feltettük, hogy az M_i műveleteknek nincs hatása az F_j feltételekre, ezért az összes F_j értéke hamis is marad, tehát az összetett művelet végrehajtása nem tud befejeződni. Ilyenkor végtelen ciklus végrehajtását írtuk elő.
- Fontos tehát, hogy legyen olyan M_i, amelyik valamelyik kijárati feltételre hat.

Hurok ismétléses vezérlés

- A kezdő- és végfeltételes ismétléses vezérlések speciális esetei a hurok ismétléses vezérlésnek.



Legnagyobb közös osztó

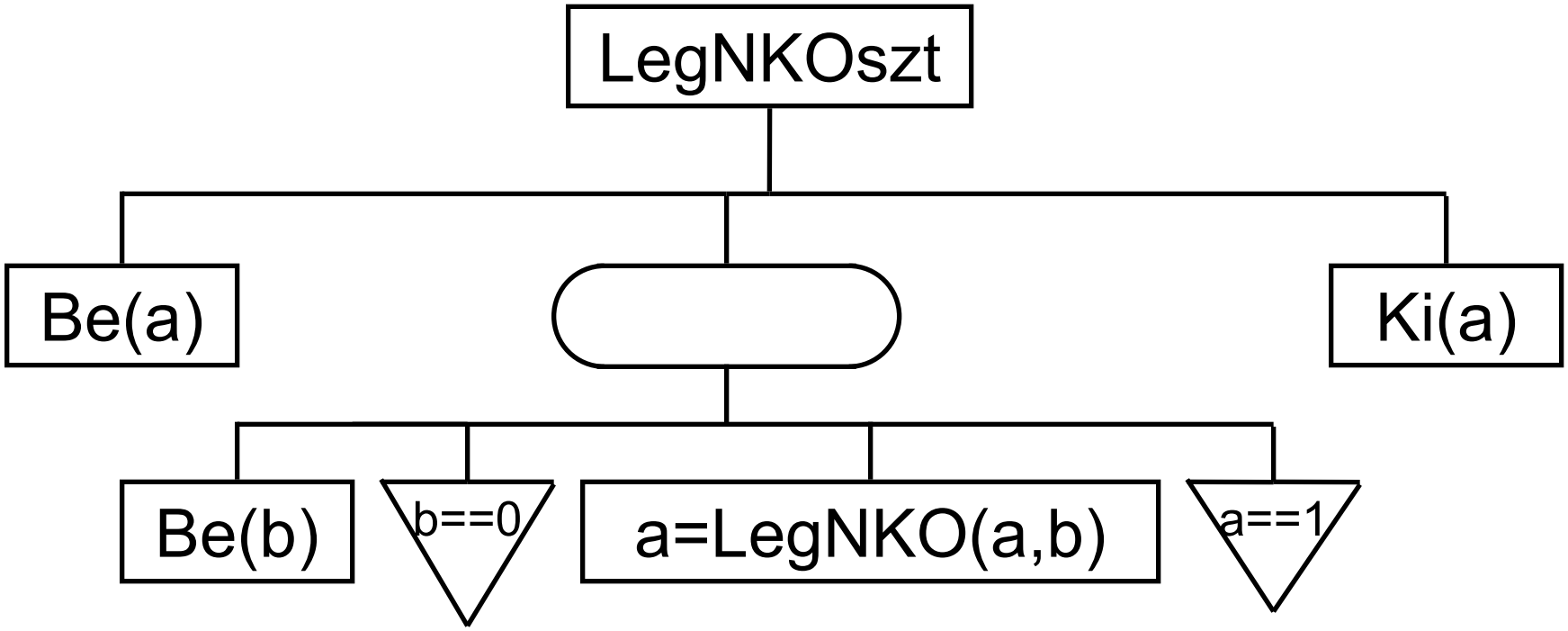
- Problémafelvetés:
  - Pozitív egész számok legnagyobb közös osztójának meghatározása
- Specifikáció:
  - Input
    - Számsorozat melyet a 0 szám zár
  - Output
    - A számok legnagyobb közös osztója

Legnagyobb közös osztó

- Algoritmustervezés:
  - Az algoritmus lényege egy olyan ismétléses vezérlés, amely ciklusmagjának egyszeri végrehajtása kiszámítja a már feldolgozott input számsor és a beolvasott következő szám legnagyobb közös osztóját.
  - Látható, hogy a ciklusmag ismétlését célszerű a ciklusmagban vezérelni, mert az ismétlés befejeződhet úgy, hogy
    - Végére értünk az inputnak
    - A feldolgozott sorozat legnagyobb közös osztója 1

Legnagyobb közös osztó

- Szerkezeti ábra



Legnagyobb közös osztó

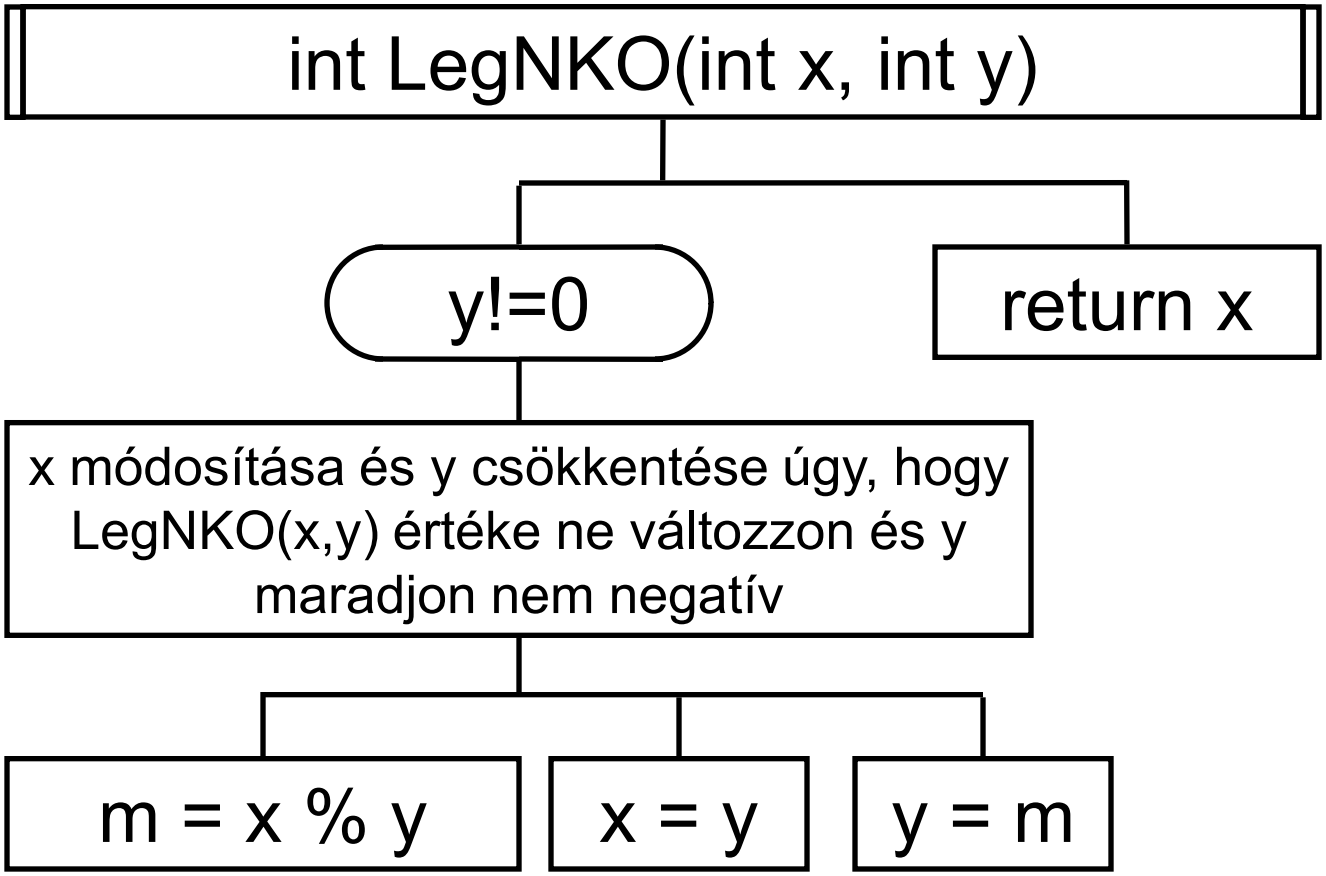
- Problémafelvetés
  - Legnagyobb közös osztó kiszámítása
- Specifikáció
  - Függvényműveletet készítünk
  - Input
    - x és y nemnegatív egész számok (bemenő argumentumok)
  - Output
    - Az LegNKO függvényművelet int típusú eredményt szolgáltat
  - Tehát
    - int LegNKO(int x, int y)

Legnagyobb közös osztó

- Algoritmustervezés
  - Euklidesz algoritmusát valósítjuk meg.
  - Mivel LegNKO(x,0)=x, így a következő algoritmus a probléma megoldását adja.
  - A ciklusmag helyes megoldása biztosítja, hogy az ismétlés véges sok lépésben befejeződik.

Legnagyobb közös osztó

- Szerkezeti ábra



Legnagyobb közös osztó

- A ciklusmag kivitelezéséhez vegyük figyelembe:

  LegNKO(x,y) = LegNKO(y,x % y), ha y>0
- Ugyanis a % és / műveletek definíciója miatt

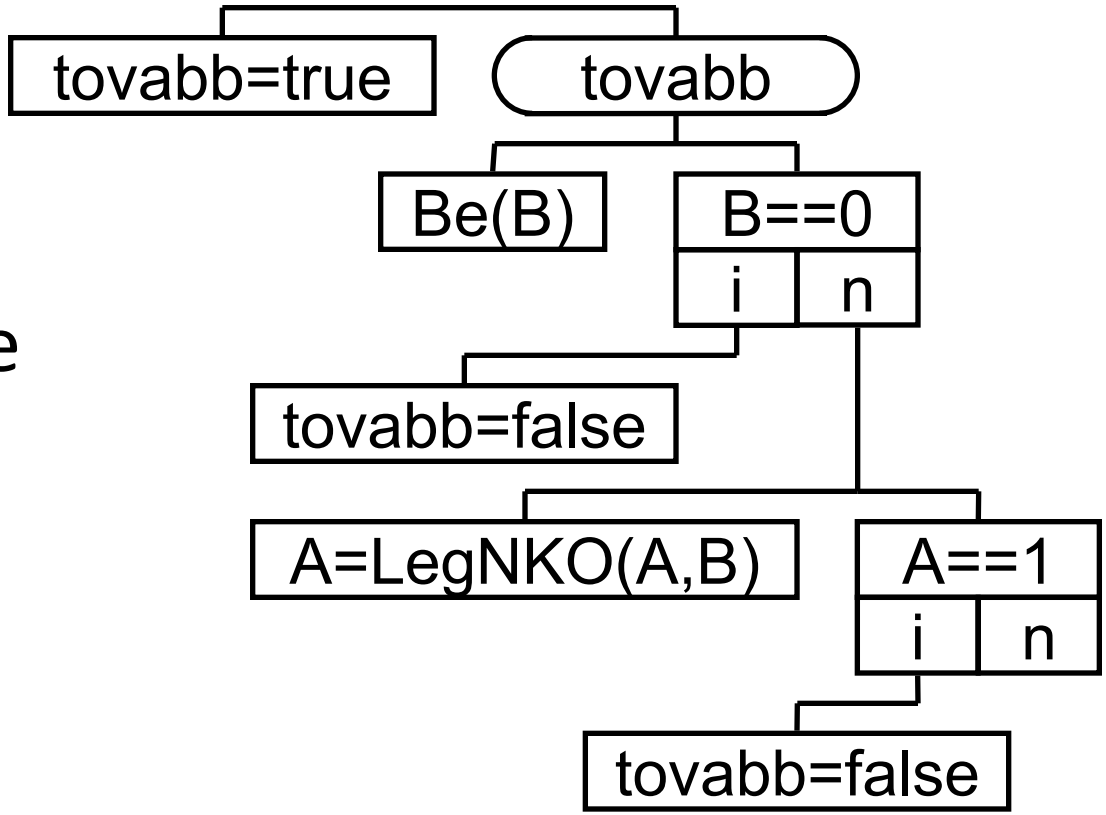
  x = y * (x / y) + x % y
- Tehát,
  - Ha d osztója x-nek és y-nak, akkor osztója (x % y)-nak is
  - Ha d osztója y-nak és (x % y)-nak, akkor x-nek is,
  - Továbbá 0 <= (x % y) < y
- Ha a ciklusmagot így finomítjuk, akkor a megfogalmazott feltételek teljesülnek.

Hurok vezérlés megvalósítása #1

- A C nyelvben nincs olyan vezérlési forma, amellyel közvetlenül megvalósíthatnánk a hurok ismétléses vezérlést, de a kezdőfeltételes ismétléses vezérlés felhasználásával megtehetjük.
- A megvalósítás lényege, hogy választunk egy logikai változót (legyen az a **`tovabb`**), ez lesz az ismétlés feltétele és a ciklusmagban a feltételes kijáratokat úgy alakítjuk át, hogy
  - Ha a feltétel igaz, akkor a **`tovabb`** hamis értéket kap
  - Egyébként végrehajtódik a ciklusmag további része

Hurok vezérlés megvalósítása #1

- A LNKOszt algoritmusban szereplő hurok ciklus tehát a következőképpen valósítható meg



Legnagyobb közös osztó #1

```
/* Pozitív egész számok legnagyobb közös osztójának meghatározása.
 * A hurok ismétléses vezérlés megvalósítása break utasítással.
 * 1997. November 7.  Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2006. Augusztus 8.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>
```

Legnagyobb közös osztó #1

```
int LegNKO(int x, int y)
{ /* x és y legnagyobb közös osztójának meghatározása
   * Euklidesz algoritmusával.
   */
  int m;
  while (y != 0) {
      m = x % y;
      x = y;
      y = m;
  }
  return x;
}
```

>>>

Legnagyobb közös osztó #1

```
main()
{
  int a, b;
  int tovabb; /* logikai változó a ciklus megvalósításához */
  printf("A program pozitív egész számok legnagyobb\n");
  printf("közös osztóját számítja.\n");
  printf("Kérem a számok sorozatát, amit 0 zár!\n");
  printf("? ");
  scanf("%d%*[^\n]", &a);
  getchar();
```

>>>

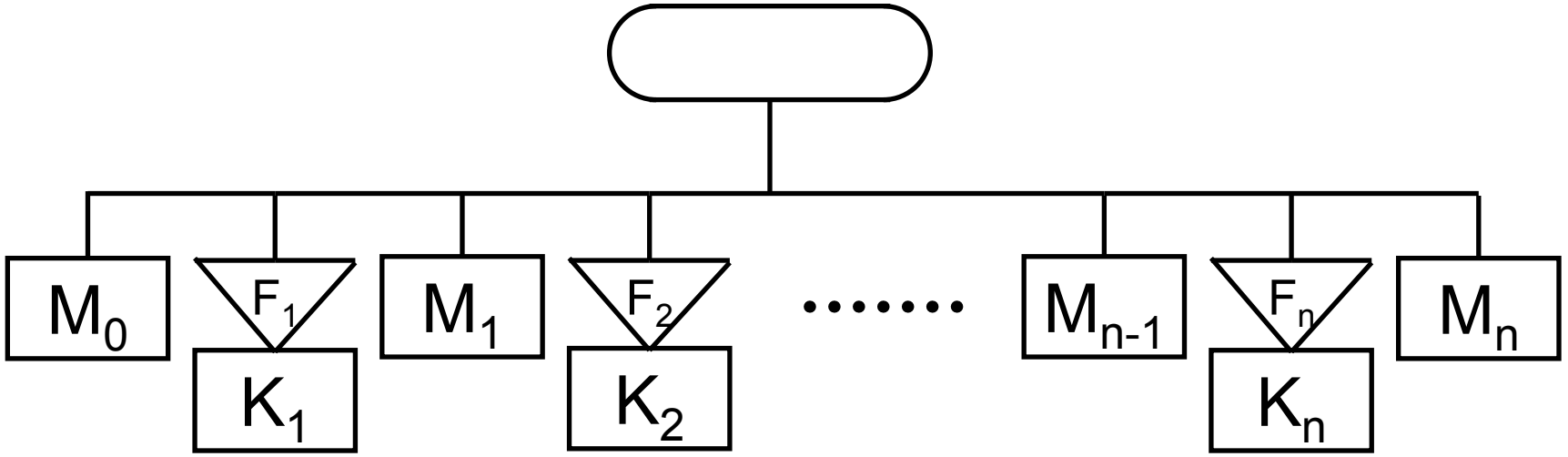
Legnagyobb közös osztó #1

```
  tovabb = !0;
  while (tovabb) {                  /*  a hurok ciklus kezdete */
      printf("? ");
      scanf("%d%*[^\n]", &b);
      getchar();
      if (b == 0) {                          /* első kijárat */
          tovabb = 0;
      } else {
          a = LegNKO(a, b);
          if (a == 1) {                    /* második kijárat */
              tovabb = 0;
          }
      }
  }                                      /* a hurok ciklus vége */
  printf(" A számok legnagyobb közös osztója: %d\n",a);
}
```

Hurok vezérlés megvalósítása #2

- A hurok ismétléses vezérlés második megvalósítása a C nyelv break utasítását használja.
- A **`break`** utasítás „megtöri” az aktuális ismétléses (vagy mint láttuk, esetkiválasztásos szelekciós) vezérlést, és a vezérlési szerkezet utáni első utasításnál folytatja a programot.

Hurok vezérlés megvalósítása #2




```
while (1) {
     M0;
     if (F1) {
          K1; break;
     }
     M1;
     ...
     if (Fn) {
          Kn; break;
     }
     Mn;
}
```

Hurok vezérlés megvalósítása #2

- A `while (1) {}`

  végtelen ciklus helyett alkalmazhatjuk a

  `for (;;) {}`

  végtelen ciklus utasítást is.

Legnagyobb közös osztó #2

```
/* Pozitív egész számok legnagyobb közös osztójának meghatározása.
 * A hurok ismétléses vezérlés megvalósítása break utasítással.
 * 1997. November 7.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>
```

Legnagyobb közös osztó #2

```
int LegNKO(int x, int y)
{ /* x és y legnagyobb közös osztójának meghatározása
   * Euklidesz algoritmusával.
   */
  int m;
  while (y != 0) {
      m = x % y;
      x = y;
      y = m;
  }
  return x;
}
```

>>>

Legnagyobb közös osztó #2

```
main()
{
  int a, b;
  printf("A program pozitív egész számok legnagyobb\n");
  printf("közös osztóját számítja.\n");
  printf("Kérem a számok sorozatát, amit 0 zár!\n");
  printf("? ");
  scanf("%d%*[^\n]", &a);
  getchar();
```

>>>

Legnagyobb közös osztó #2

```
  while (1) {                          /*  a hurok ciklus kezdete */
      printf("? ");
      scanf("%d%*[^\n]", &b);
      getchar();
      if (b == 0) {                              /* első kijárat */
          break;
      }
      a = LegNKO(a, b);
      if (a == 1) {                            /* második kijárat */
          break;
      }
  }                                        /* a hurok ciklus vége */
  printf(" A számok legnagyobb közös osztója: %d\n",a);
}
```

break, continue

- A C nyelvben a ciklusmag folyamatos végrehajtásának megszakítására két utasítás használható:
  - break
    - Megszakítja a ciklust, a vezérlés a ciklusmag utáni első utasítással foglalkozik
  - continue
    - Megszakítja a ciklus aktuális lefutását, a vezérlés a ciklus feltételének kiértékelésével folytatódik

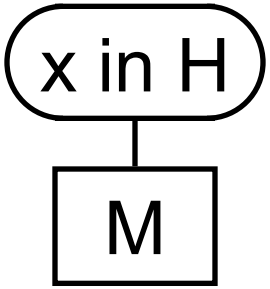
Diszkrét ismétléses vezérlés

- Diszkrét ismétléses vezérlésről beszélünk, ha azt a vezérlési előírást tesszük, hogy az x ciklusváltozó egy adott H véges halmaz minden elemét tetszőleges sorrendben felvéve hajtsuk végre az M ciklusmagot.

Diszkrét ismétléses vezérlés

- Fontos, hogy az M művelet nem lehet hatással az ismétlési előírás egyik elemére sem.
- A H halmaz számossága határozza meg tehát, hogy az M művelet hányszor hajtódik végre.
- Ha a H az üres halmaz, akkor a diszkrét ismétléses vezérlés az M művelet végrehajtása nélkül befejeződik.

Diszkrét ismétléses vezérlés



- A diszkrét ismétléses vezérlésnek nincs közvetlen megvalósítása a C nyelvben.

Diszkrét vezérlés megvalósítása

- A megvalósítás elsősorban attól függ, hogy az ismétlési feltételben megadott halmazt hogyan reprezentáljuk.
- Algoritmustervezés során szabadon használjuk a diszkrét ismétléses vezérlést, ha erre van szükség a probléma megoldásához.
- A halmaz reprezentálásáról pedig akkor döntsünk, amikor elegendő információ áll rendelkezésünkre, hogy a legmegfelelőbbet kiválaszthassuk.

ELJÁRÁSVEZÉRLÉS

Eljárásvezérlés

- Adott művelet alkalmazása adott argumentumokra, ami az argumentumok értékének pontosan meghatározott változását eredményezi
- Az eljárásvezérlés fajtái
  - Eljárásművelet
  - Függvényművelet

Eljárásművelet

- Eljárásműveleten olyan tevékenységet értünk, amelynek alkalmazása adott argumentumokra az argumentumok értékének pontosan meghatározott megváltozását eredményezi.
- Minden eljárásműveletnek rögzített számú argumentuma van, és minden argumentum rögzített adattípusú. Az argumentumok lehetnek
  - Bemenő argumentumok
  - Kimenő argumentumok
  - Be- és kimenő argumentumok

Argumentumok kezelési módjai

- Bemenő argumentum
  - Ha a művelet bármely végrehajtása nem változtatja meg az adott argumentum értékét.
- Kimenő argumentum
  - Ha a művelet hatása nem függ az adott argumentumnak a végrehajtás előtti értékétől, azonban az adott argumentum értéke a művelet hatására megváltozhat.
- Be- és kimenő argumentum
  - Ha a művelet hatása függ az adott argumentumnak a végrehajtás előtti értékétől, és az adott argumentum értéke a művelet hatására megváltozhat.

Függvényművelet

- A matematikai függvény fogalmának általánosítása.
- Ha egy részprobléma célja egy érték kiszámítása adott értékek függvényében, akkor a megoldást megadhatjuk függvényművelettel.
- A függvényművelet argumentumai ugyanúgy lehetnek kimenő és be- és kimenő módúak is, mint az eljárásműveletek esetén, tehát a függvényművelet végrehajtása az aktuális argumentumok megváltozását is eredményezheti.

Eljárás és függvény specifikációja

- Eljárásművelet specifikációja tartalmazza:
  - A művelet elnevezését
  - A formális argumentumok felsorolását
  - Mindegyik argumentum adattípusát és kezelési módját
  - A művelet hatásának leírását
- A függvényművelet specifikációja a fentieken túl tartalmazza még:
  - A függvényművelet eredménytípusát

Eljárásművelet általános jelölése

- Eljárásműveletek (általános) jelölése:

  $P(m_1\ X_1{:}T_1;\ \ldots\ ;m_n\ X_n{:}T_n)$

  ahol
  - P az eljárásművelet neve
  - X_i az i-edik formális argumentum azonosítója
  - m_i az i-edik formális argumentum kezelési módja
    - -> bemenő mód
    - <- kimenő mód
    - <-> be- és kimenő mód.
  - T_i az i-edik formális argumentum adattípusa

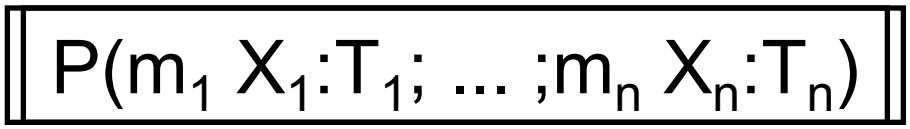
Eljárásművelet általános jelölése

- Az eljárásműveletnek adott A_1, ... ,A_n aktuális argumentumokra történő végrehajtását eljárásutasításnak (eljáráshívásnak) nevezzük
- Jelölése

  $P(A_1, \ldots, A_n)$
- Ha az i-edik argumentum módja kimenő vagy be- és kimenő, akkor az A_i aktuális argumentum csak változó lehet.

Eljárásművelet

- Algoritmustervezés során nem csak elemi (eleve definiált) eljárásműveleteket használhatunk.
- Részproblémák megoldását is kifejezhetjük olyan eljárásműveletekkel, melyek megvalósítását részprogrammal adjuk meg.
- A részprogramot különálló szerkezeti ábrával írjuk le, amelynek feje



Függvényművelet általános jelölése

- Függvényműveletek (általános) jelölése

  $F(m_1\ X_1{:}T_1; \ldots ; m_n\ X_n{:}T_n){:}T$

  ahol
  - F az függvényművelet neve
  - X_i az i-edik formális argumentum azonosítója
  - m_i az i-edik formális argumentum kezelési módja
  - T_i az i-edik formális argumentum adattípusa
  - T a függvényművelet eredménytípusa

Függvényművelet

- A fent jelölt függvényműveletnek adott A_1, ... ,A_n aktuális argumentumokra történő végrehajtását függvényhívásnak nevezzük
- Jelölése
  - $F(A_1, \ldots ,A_n)$
- Részproblémák megoldását függvényművelettel is kifejezhetjük, ekkor a szerkezeti ábra feje

$$F(m_1\ X_1{:}T_1; \ldots ; m_n\ X_n{:}T_n){:}T$$

Eljárásvezérlés megvalósítása

- Vannak olyan programozási nyelvek, ahol a függvény és eljárásműveletek meg vannak különböztetve, valamint a paraméterek módjaira sincs megkötés.
- Mivel azonban e kurzus keretében csak a C nyelvről lesz szó, a szerkezeti ábrán a továbbiakban (is) igazodni fogunk a C nyelvhez.

Eljárásvezérlés megvalósítása

- A C nyelvben lényegében csak függvényművelet van.
- C nyelvben a függvényművelet argumentumai bemenő módúak, tehát alapvetően a függvényművelet végrehajtása az aktuális argumentumok megváltozását nem eredményezheti.

Függvényművelet

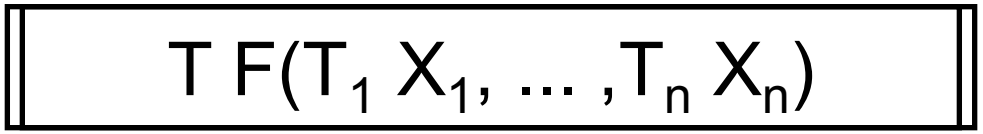
- A függvényművelet jelölésére a továbbiakban a

  `T F(T1 X1, ... ,Tn Xn)`

  formát használjuk, ahol
  - **`T`** a függvényművelet eredménytípusa
  - **`F`** a függvényművelet neve
  - **`Ti`** az **`i`**-edik formális argumentum adattípusa
  - **`Xi`** az **`i`**-edik formális argumentum azonosítója
- A zárójeleket üres paraméterlista esetén is ki kell tenni.

Függvényművelet

- A C jelölésmódhoz igazodva, a függvényművelet szerkezeti ábrájának a feje így néz ki:



- Továbbá a szerkezeti ábrában lennie kell (legalább) egy olyan **`return`** utasításnak, amely visszaadja a függvény által kiszámított értéket.

Függvényművelet

- A fent jelölt függvényműveletnek adott A1, ... ,An aktuális argumentumokra történő végrehajtását függvényhívásnak nevezzük és az

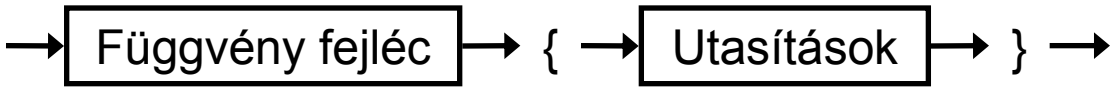
  `F(A1, ... ,An)`

  jelölést használjuk.
- A függvényhívás kifejezés.
- A zárójeleket paraméter nélküli függvény hívása esetén is ki kell tenni.

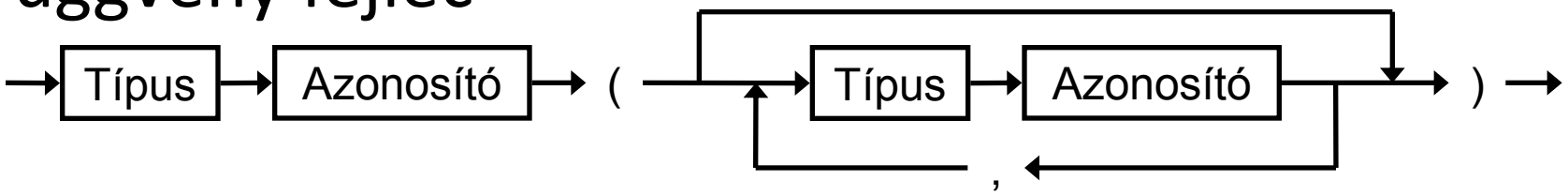
Függvények szintaxisa C-ben

- Függvény deklaráció

- Függvény definíció (egyben deklaráció is)



- Függvény fejléc



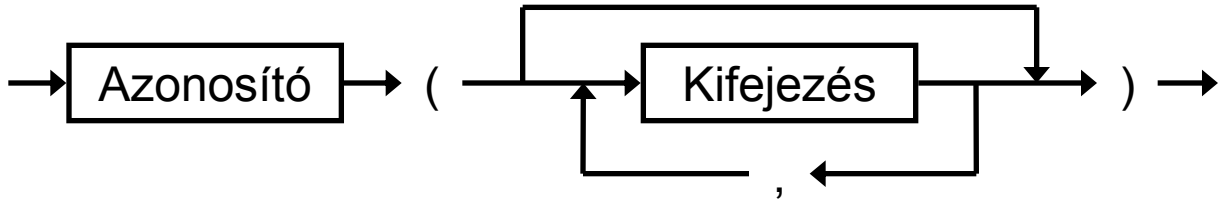
Függvények szintaxisa C-ben

- A **`return`** utasítás

- Minden függvényben szerepelnie kell legalább egy **`return`** utasításnak, amely kiszámítja a megadott kifejezés értékét, majd visszatér a függvényből.
- A hívás helyén a függvény a **`return`** által kiszámított értéket veszi fel.

Függvényhívás szintaxisa C-ben

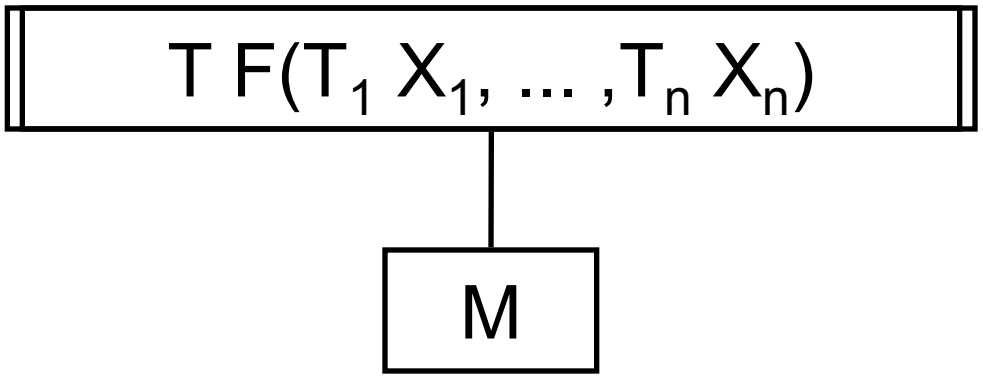
- Függvényhívás



- Természetesen egy függvénynek a híváskor pontosan annyi és olyan típusú paramétert kell átadni, amennyi és amilyen paraméterekkel deklarálva lett.

Függvény megvalósítása

- Ha a függvény szerkezeti ábrája ez:



- Akkor a függvénydefiníció C-ben:

```
T F(T1 X1, ..., Tn Xn)
{
    M;
}
```

Függvényművelet

- Például

```
float atlag(float a, float b)
{
    return (a + b)/2;
}
```

- Régen C-ben így kellett függvényt deklarálni:

```
float atlag(a,b)
    float a, float b;
{
    return (a + b)/2;
}
```

Eljárásművelet

- Ha eljárást szeretnénk készíteni, akkor a függvényműművelet eredménytípusa **`void`** és ebben az esetben nem kötelező a **`return`** utasítás, illetve ha mégis van ilyen, akkor nem adható meg utána kifejezés.

Vegyes és kimenő módú arg.

- Mint említettük, C-ben csak bemenő módú argumentumok vannak.
- De mi magunk kezelhetjük a be- és kimenő illetve kimenő módú argumentumokat pointerek segítségével.
- Az alábbiakban egy, az alaptípusokra működő megoldást mutatunk.

Vegyes és kimenő módú arg.

- Ha az i. paramétert kimenő módúnak szeretnénk, akkor a függvény deklarációjában

  `Ti Xi`

  helyett

  `Ti *Xi`

  deklarációt, a függvénytörzsben pedig

  `Xi`

  helyett mindenhol

  `(*Xi)`

  változóhivatkozást használunk.

Vegyes és kimenő módú arg.

- Továbbá a függvény meghívásakor az

  `Ai`

  paraméter helyett az

  `&Ai`

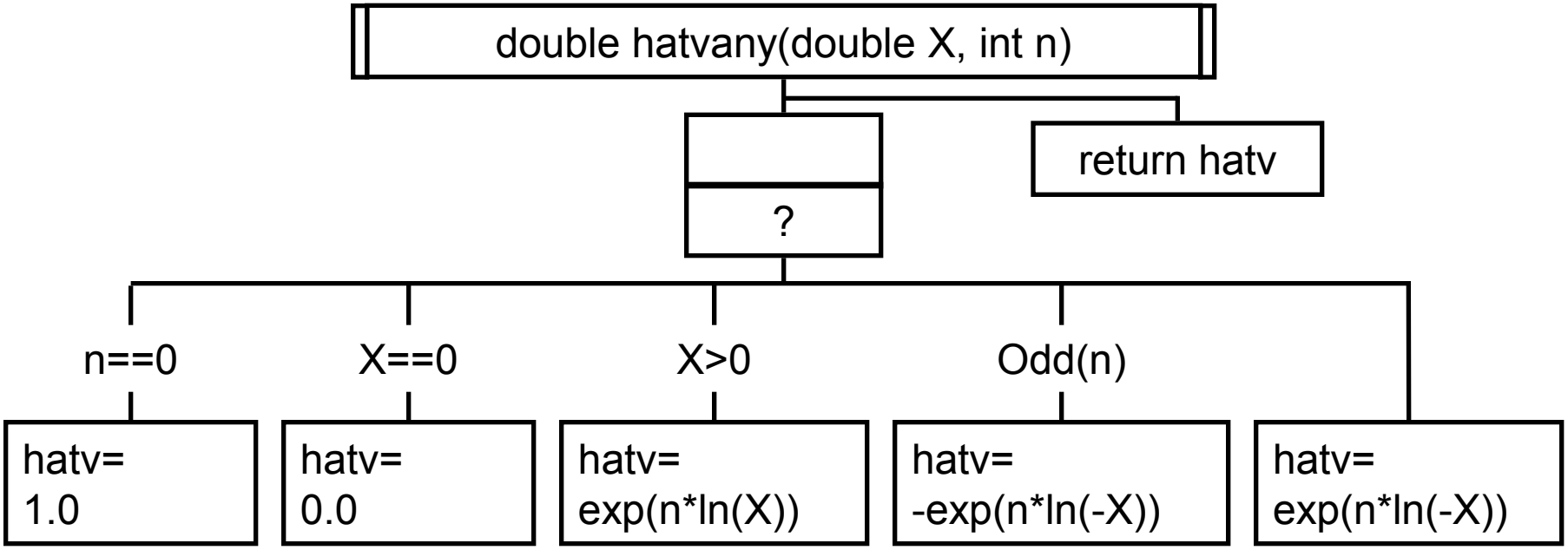
  paramétert írjuk.
- Részletesebb magyarázatot és összetettebb típusokra működő megvalósítást a pointerek megismerése után adunk.

Kamatos-kamat számítás

- Problémafelvetés
  - Számítsunk kamatos kamatot
- Specifikáció
  - Input
    - Osszeg valós, a betett összeg
    - Kamatláb egész, az éves alapkamat
    - Ev egész, az eltelt évek száma
  - Output
    - Ujosszeg a kamatos-kamattal növelt érték

Kamatos-kamat számítás

- Algoritmustervezés
  - A kamatos-kamat számításhoz készítünk egy általános hatványozó függvényt és ezt hívjuk meg a kamatos-kamat számítás alapképletével.



Bitenkénti logikai műveletek

- A C nyelvben több bitmanipulációs operátor van, ezek a **`float`** és **`double`** típusú változókra nem alkalmazhatók.
  - **`&`** bitenkénti ÉS,
  - **`|`** bitenkénti megengedő (inkluzív) VAGY,
  - **`^`** bitenkénti kizáró (exkluzív) VAGY,
  - **`<<`** bitléptetés (shift) balra,
  - **`>>`** bitléptetés (shift) jobbra,
  - **`~`** egyes komplemens (egyoperandusú).

Bitenkénti logikai műveletek

- A bitenkénti ÉS operátort gyakran használjuk valamely bithalmaz maszkolására. Például a páratlan(x) függvényt az

  ```
  ((x & 1) == 1)
  ```

  valósítja meg.

- A műveletekről később részletesen is lesz szó.

Bitenkénti logikai műveletek

- Illesszük be prioritási sorba a műveleteket!
  - a egyoperandusú műveletek (-, ++, --, !, ~)
  - a multiplikatív műveletek (*, /, %)
  - az additív műveletek (+, -)
  - bitléptetés (<<, >>)
  - a kisebb-nagyobb relációs műveletek (<=, >=, <, >)
  - az egyenlő-nem egyenlő relációs műveletek (==, !=)
  - bitenkénti 'és' művelet (&)
  - bitenkénti 'kizáró vagy' művelet (^)
  - bitenkénti 'vagy' művelet (|)
  - a logikai 'és' művelet (&&)
  - a logikai 'vagy' művelet (||)
  - a feltételes művelet (? :)
  - értékadó művelet (=, +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

Kamatos-kamat számítás

```
/* Kamatos-kamat számítás a hatványozás függvény segítségével.
 * 1997. Október 31.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

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

Kamatos-kamat számítás

```
double hatvany(double x, int n)
{ /* x n-edik hatványát kiszámító függvény */
  double hatv;
  if (n == 0) {
      hatv = 1.0;
  } else if (x == 0.0) {
      hatv =  0.0;
  } else if (x > 0.0) {
      hatv = exp(n * log(x));
  } else if (n & 1) {
      hatv = (-exp(n * log(-x)));
  } else {
      hatv = exp(n * log(-x));
  }
  return(hatv);
}
```

Kamatos-kamat számítás

```
main()
{
  double Osszeg, Ujosszeg;
  int Kamatlab, Ev;
  printf(" A kamatozó összeg ? ");
  scanf("%lg%*[^\n]", &Osszeg); getchar();
  printf(" A kamatláb ? ");
  scanf("%d%*[^\n]", &Kamatlab); getchar();
  printf(" A kamatozási évek száma ? ");
  scanf("%d%*[^\n]", &Ev); getchar();
  Ujosszeg = Osszeg * hatvany(1.0 + Kamatlab / 100.0, Ev);
  printf("A kamatos kamattal növelt összeg:");
  printf("%10.2f\n", Ujosszeg);
}
```