Pásztor Attila

Algoritmizálás és programozás tankönyv az emeltszintű érettségihez

1. Alapfogalmak

Ebben a fejezetben a programozás alapfogalmait tekintjük át.

1.1. Program

Lényegében a számítógép által végrehajtható algoritmus, pontosabban: adatszerkezet és algoritmus.

1.2. A program készítésének lépései

Amikor programot készítünk, akkor a program elkészítése az alábbi részfeladatokból áll:

1. **Specifikáció** = feladat meghatározása, mit kell megoldani?
2. **Tervezés** = hogyan kell a feladatot megoldani?
3. **Kódolás** = számítógépes megvalósítás, melynek eredménye a program.
4. **Tesztelés** = a program helyes működésének vizsgálata.
5. **Hibakeresés, javítás** (ha szükséges, akkor változtatás a 2.,3. pontnál).
6. **Hatékonyság** = minőség vizsgálat: gyorsabbá, kevesebb erőforrást használóvá alakítás.
7. **Dokumentálás** = Fejlesztői és felhasználói dokumentáció készítés.

A dokumentálás két elemét érdemes lenne még részekre bontani, de ez a tankönyv elsősorban az első három ponttal foglalkozik, tekintsük át, hogy mit is jelentenek ezek!

1.3. Specifikáció (a feladat specifikációja)

A **specifikáció** (meghatározás) a következő összetevőkből áll:

1. A feladat szövege.
2. A **bemenő adatok** elnevezése (hogyan hivatkozunk majd rá), értékhalmaza (milyen értékeket vehet fel).
3. A **kimenő adatok** elnevezése (amit eredményként várunk), értékhalmaza.
4. A bemenő adatokra vonatkozó **előfeltétel** (milyen megszorításnak kell megfelelnie).
5. A kimenő adatokra vonatkozó **utófeltétel**. Itt fogalmazzuk meg, hogy a bemenő adatokból hogyan „keletkeznek” a kimenő adatok.
6. Fogalmak definíciója (az előző pontokban használat fogalmak meghatározása).

A specifikáció minden pontját matematikai eszközrendszerrel illik megadni, de ebben a könyvben az egyszerűség kedvéért szövegesen fogom megadni.

1.4. Specifikáció megadása szövegesen

Az előbbiekben már említettem, hogy a feladat specifikálása matematikai eszköztárral történik, de a középiskolás szintű programozáshoz elegendőnek tartom a szöveges specifiká-

lást, viszont azt nagyon fontosnak tartom. Az elkészült program ugyanis akkor jó, ha teljesíti a specifikációt, azzal kell egybevetni.

Nézzünk egy egyszerű feladatot: Számold ki a háromszög területét, ha adott az egyik oldala és a hozzá tartozó magassága!

A szöveges specifikáció:

A feladat:	Számold ki a háromszög területét, ha adott az egyik oldala és a hozzá tartozó magassága!
Bemenet:	a (oldalhossz, valós szám), m (magasság, valós szám)
Kimenet:	t (terület, valós szám)
Előfeltétel:	a és m pozitív (ez következik abból, hogy valódi síkidom)
Utófeltétel:	t=a*m/2

Látható, hogy az utófeltétel fogalmazza meg, hogy hogyan kell elkészíteni a bemeneti adatokból a kimeneti adatokat. Ebben a feladatban nem kellett fogalom definíció.

Érdemes lesz majd megnézni, hogy a tervezés (algoritmus készítés) eredménye hogyan teljesíti a specifikációt: megengedett, hogy az elkészült program specifikációja az előfeltétel szempontjából megengedőbb, az utófeltételt tekintve szigorúbb legyen.

1.5. Összefoglalás

Ebben a fejezetben megismerkedtünk a program fogalmával és a programkészítés lépéseivel.

A programkészítés első lépésével a specifikálással foglalkoztunk, pontosabban a feladat specifikálással. Bár a programozók elsősorban **formális specifikációt** használnak, mely a specifikáció összetevőiének matematikai nyelven történő megfogalmazása, mi a szöveges specifikálással megelégedtünk.

1.6. Gyakorló feladatok

- Készítsd el az iskolában található üdítőital automata működésének szöveges specifikálását (ne törekedj a túlzott részletességre)!
- Készítsd el a palacsinta sütés szöveges specifikációját!
- Készítsd el a másodfokú egyenlet megoldó képletének szöveges specifikációját!

2. Algoritmizálás (tervezés)

Az **algoritmus** egy feladat véges számú lépésre bontott megoldása. A lépések elemi műveletek (véges sok van belőlük), melyek utasítások formájában írhatók le, tehát található hozzá olyan gép, ami végrehajtja. Az algoritmus lépései elemi tevékenységek (a végrehajtó érti, vagy meg kell tanítani rá – ez újabb algoritmus).

Az algoritmusok legfontosabb jellemzője:

1. Egyértelmű: ugyanazon bemenő adatok esetén mindig ugyanazt a választ adja.
2. Véges számú lépésből áll: leírásuk véges hosszúságú szöveggel lehetséges. Természetesen ez nem jelenti azt, hogy a futása is véges idejű, bár erre törekszünk.
3. Általános: a bemenő adatokra megfogalmazott feltételeknek eleget tevő értékekre eredményt ad.

2.1. Algoritmusok leírása

Az algoritmusokat szöveges, vagy rajzos eszközökkel adhatjuk meg. A teljesség igénye nélkül vegyünk sorra néhányat!

2.1.1. Algoritmusok szöveges leírása

Az algoritmusok szöveges leírását minden diák használta már. Válasszunk egy egyszerű matematika feladatot: Számold ki a háromszög területét, ha adott az egyik oldala és a hozzá tartozó magasság!
Jelöljük az oldalt a-val, a magasságot m-mel, a területet t-vel. Ekkor a feladat megoldása az alábbi képlet alkalmazását (kiszámítását) jelenti:
t=a*m/2
A kiszámítás menete (algoritmusa) szövegesen:

1. Szükségünk van a és m értékére (ezek az úgynevezett bemenő adatok), megkérdezzük azokat.
2. A t értéke legyen az a*m/2 műveletsor eredménye.
3. Közöljük az eredményt (ezt nevezzük kimenetnek).

Készíts algoritmust a következő számkitaláló játékra! Gondolj egy egész számot 0 és 100 között, és jelöld ezt g-vel. Aki ki akarja találni a számot, a következő módszer szerint jár el:
A játékos mond egy számot (jelöljük ezt x-szel), te pedig megmondod, hogy a gondolt szám ennél nagyobb, vagy kisebb. Ezt mindaddig ismétlitek, amíg a kérdező kitalálja a gondolt számot!
A fenti játék algoritmusa szövegesen az alábbi:

1. Gondolj egy számot (ezt nevezem g-nek)!
2. Kérdezd meg társad tippjét!
3. Az általa mondott szám legyen x.
4. Hasonlítsd össze a két számot!
5. Ha g>x, akkor mondd: „A gondolt szám nagyobb.”
6. Ha g<x, akkor mondd: „A gondolt szám kisebb.”
7. Ha g=x, akkor mondd: „Eltaláltad.”
8. Ismételd a 2. ponttól addig, amíg g=x nem teljesül!

Ez az algoritmus már szinte minden lényeges algoritmus alapelemet tartalmaz. Tanulmányaink során strukturált programokat szeretnénk írni. **Strukturált** programnak nevezzük azt a programot, mely csak az alábbi strukturált algoritmikus szerkezeteket tartalmazza:

- szekvencia,
- elágazás,
- ciklus.

A strukturált programok előnyeinek, illetve a nem strukturált programok hátrányainak ismertetése túlmutat eme könyv keretein, elégedjünk meg azzal, hogy a „jó" programok készítése érdekében mi mindig strukturáltat fogunk készíteni.

A programok alapelemei szövegesen

Most ismertetem, hogy az algoritmusok szöveges leírása során milyen elemeket szeretnék felhasználni.

Egymásutániság (szekvencia)

Az utasítások egymás utáni végrehajtása.

Elágazás (szelekció) többféle lehet:

- Egy műveletsort csak egy feltétel teljesülésekor kell végrehajtani (ilyen van a példában). Szokták ezt **feltételes utasítás**nak is nevezni.
- Két műveletsor közül az egyiket akkor kell végrehajtani amikor a feltétel igaz, míg a másikat akkor, ha a feltétel hamis. Például: Ha süt a Nap strandra megyünk, ha nem, akkor moziba. Ezt nevezzük **kétágú elágazás**nak, esetleg röviden **elágazás-nak**. Az előző típus is felfogható úgy, hogy hamis feltétel esetén nem tartalmaz utasítást az elágazás (pontosabban üres utasítást tartalmaz).
- Kettőnél több műveletsor közül kell választani, általában egy kifejezés kiértékelése alapján. Például: Ha a Balaton vízmélysége kisebb, mint 1 méter, akkor pancsolunk benne, ha a vízmélység 1 és 2 m közötti, akkor úszunk, ha mélyebb, mint két méter, akkor nem megyünk be. Ezt nevezik **többágú**, vagy **sokágú** elágazásnak. Gyakran az ilyen többágú elágazásnak van „egyébként" része, mely akkor hajtódik végre, ha egyik feltétel sem igaz. Megállapodás kérdése, hogy a többágú elágazás végrehajtása során igaz feltételt találva tovább folytatjuk-e a feltételek kiértékelését, vagy nem.

Ismétlés (iteráció)

Egy feltétel fennállásáig, vagy megadott számszor ismételjük a felsorolt utasításokat, amiket **ciklusmag**nak nevezünk. A ciklusmag nem más, mint a többször végrehajtandó utasítások sorozata.

Az ismétlésnek három típusát különböztetjük meg:

Elöltesztelő ciklus:

A feltétel vizsgálata a ciklusmag előtt található, a ciklusmag csak akkor hajtódik végre, ha a feltétel igaz. Például: Amíg van pénzed, vásárolj! Ha nincs pénzed, akkor egyszer sem vásárolsz (praktikus megoldás).

Hátultesztelő ciklus:
A feltétel vizsgálata a ciklusmag után található, ezért a ciklusmag egyszer biztosan végrehajtódik. Például: Ismételd a tapsolást, amíg nem fáj a kezed! Pontosabban: Tapsolj, és ismételd amíg nem fáj a kezed! (Egyszer mindenképpen tapsolsz.)

Számláló ciklus:
A ciklusmag végrehajtása előre ismert mennyiségben.
Például: Ismételd 5-ször: Vágj egy szelet kenyeret!
Másik példa: Ismételd 1-től 5 „darab”-ig: Süss ki egy palacsintát! (Először a darab=1, kisütsz 1 palacsintát, a darabszám 2 lesz, kisütsz még 1 palacsintát…). Az ismétlésszám megadásánál olyan mennyiséget használhatsz, amelyiknél a „következő” fogalma értelmezett. Lehet tehát egész szám például, de nem lehet valós szám.

Láthattuk, hogy a mondatokkal történő leírás kellemesen használható, de hátránya, hogy hosszadalmas, erősen nyelvfüggő és nem mindig egyértelmű.

Kérdések, feladatok

- Miért jó a szöveges algoritmus leírás?
- Mi a hátránya az algoritmusok szöveges leírásának?
- Milyen „alapelemeket” használunk az algoritmusokban?
- Készítsd el az iskolában található üdítőital automata működésének szöveges algoritmusát (ne törekedj a túlzott részletességre)!
- Készítsd el a palacsintasütés szöveges algoritmusát!
- Készítsd el a másodfokú egyenlet megoldó képletének szöveges algoritmusát!

2.1.2. Algoritmusok leírása mondatszerű elemekkel

A módszer lényege, hogy az algoritmusokban használatos alapelemekre szabványos szövegelemeket vezetünk be. A mondatszerű leírás alapszerkezetei megegyeznek a strukturált programok alapszerkezeteivel, kiegészítve néhány „kényelmi” elemmel.

Már itt érdemes megemlíteni két fogalmat, melyek elsősorban a programozási nyelvekhez kötődnek, de már az algoritmus leírásnál is értelmezhetők:
A **szintaxis** a formai szabályok gyűjteménye (hogyan kell leírni).
A **szemantika** a tartalmi helyességet leíró szabályok összessége (mi a jelentése, mit kell értenünk alatta).

A legfontosabb alapelemek:

A program
A legfontosabb egysége, a program az alábbi felépítésű:

```
Program:
    utasítások
Program vége.
```

Megjegyzések:
Program után esetleg megadható programnév is a későbbi felidézést elősegítendő.

Mi is az a változó, és az azonosító?
Amikor matematika órán a feladatok általános megoldását keressük, akkor konkrét értékek helyett egy rövid elnevezést használunk (többnyire egy, vagy néhány betűből állót), ami emlékeztet a jelentésére.

Például a kör kerületképlete:

K=2*r*PI

Az algoritmusokban a **változó**k a megadott típusú értékek (adatok) tárolására használt memóriatartományok elnevezései. A változó betűkből, számokból és néhány írásjelből álló elnevezése az **azonosító**. Ezen azonosítók segítségével lehet lekérdezni és módosítni a változók értékét. Köznapi nyelven fogalmazva: az adatokat olyan fiókos szekrényben tároljuk, ahol a fiókok tartalmára névvel lehet hivatkozni. Természetesen az algoritmus más elemeihez is lehet azonosítót rendelni (program, programegység,…).
Vegyük észre, hogy a változók értéke az algoritmus során megváltozhat (valóban, a nevében is benne van). **Állandó**nak (konstansnak) hívjuk azt az értéket, amit az algoritmusban felhasználhatunk, de nem változtathatjuk meg. A kör kerületét meghatározó képletben ilyen a PI.

Értékadás

```
Azonosító := kifejezés
```

Kifejezés: Minden, ami kiszámítható, lehet akár konkrét érték is.

A változó neve (azonosítója) jelentheti a változó értékét, és annak helyét (címét) is.Vegyük például az **x:=x+1** értékadó utasítást!
A fenti kifejezés jobb oldalán x értékéhez hozzáadunk egyet (tehát x, az x változó értékét jelenti, majd az eredményt eltároljuk x változóban (tehát x az x változó helyét, azaz címét jelenti).

Példa:

```
a := x+y-3
```

Az **a := x+y-3** értékadó utasítás azt jelenti, hogy kiszámoljuk az **x+y-3** kifejezés értékét, és ez az érték kerül az **a** változóba.
A kettőspont nélküli egyenlőségjel az algoritmus leírásnál nem jelent értékadást, csak összehasonlítást. Az **a=x+y-3** utasítás eredménye egy logikai érték lesz (igaz, vagy hamis), annak függvényében, hogy az a,x,y változók pillanatnyi értékét behelyettesítve a kifejezésbe az egyenlőség fennáll-e, vagy sem.

Adatbevitel

```
Be: azonosítók [feltételek]
```

Az azonosítókat vesszővel választjuk el, például: Be: a,b
A feltételek tartalmazzák az előfeltétel elemeit.

Adatkivitel

```
Ki: kifejezések [formátum megkötések]
```

A kifejezéseket vesszővel választjuk el egymástól. Egy kifejezés lehet konkrét érték (szöveg is), változó, kiértékelhető kifejezés.
Szöveget idézőjelek között adhatunk meg, a kiírandó adatokat vesszővel választjuk el, például: Ki: „Eredmény”, x
Idézőjelek helyett használhatunk aposztrófokat is.
Például: Ki: ’Eredmény:’,x
A formátum megkötések tartalmazzák a megjelenésre vonatkozó előírásokat.

Megjegyzés

```
[megjegyzések]
```

Az algoritmusban elhelyezhetők magyarázó szövegek. A megjegyzéseket szögletes zárójelek közé fogjuk írni:
[ez itt egy megjegyzés]

Szekvencia

Egymás után végrehajtandó utasítások. Az utasításokat külön sorban adjuk meg, ha egy sorban több utasítás szerepel, akkor kettősponttal választjuk el őket.

Például:

```
Utasítás1
Utasítás2
.
.
UtasításN
```

Vagy:

```
Utasítás1 : Utasítás2 : … UtasításN
```

Elágazás

Feltételes utasítás

```
Ha feltétel akkor utasítások igaz esetén
Elágazás vége
```

Az előző feltételes utasítás rövidebben:

```
Ha feltétel akkor utasítások igaz esetén
```

Példa:

```
Ha i>0 akkor Ki: 2*i
```

Jelentése: Ha i pozitív, akkor kiírja a kétszeresét.

A feltétel nem más, mint egy logikai kifejezés (tetszőleges matematikai reláció, logikai értéket adó kifejezés, logikai művelet), melynek értéke igaz, vagy hamis lehet. Például:

Reláció:	Algoritmus leírásban használható más jelölés	Jelentés:
<,>,=		Kisebb, nagyobb, egyenlő
≠, ≤, ≥	<>, <=, >=	Nem egyenlő, kisebb v. egyenlő, nagyobb v. egyenlő
∈, ∉, ⊃, ⊇		Eleme, nem eleme, része, valódi része
Műveletek:		
∧, ∨, ¬	és, vagy, nem	Logikai és, megengedő vagy, tagadás

Példa:

```
Ha i>0 és i<=N akkor Ki: 2*i
```

Jelentése: ha i pozitív, és nem nagyobb, mint N, akkor kiírja i kétszeresét.

Kétágú elágazás

```
Ha feltétel
      akkor utasítások igaz esetén
      különben utasítások hamis esetén
Elágazás vége
```

Példa:

```
Ha i>=0
     akkor Ki: i
     különben Ki: -i
Elágazás vége
```

Jelentése: Ha i nem negatív, akkor önmagát írja ki, ha i negatív, akkor az ellentettjét (ez i abszolút értéke).

Vegyük észre, hogy itt már a tagolt (oszlopszerű) írásnak is jelentősége van: segíti az értelmezést. Mindig alkalmazzunk oszlopszerű írásmódot az elágazásoknál és később tárgyalandó ismétléses szerkezeteknél!
Bármelyik ágban lehet üres utasítás (bár akkor célszerű az egyszerű feltételes utasítást alkalmazni)
Az elágazás vége elhagyható, ha az írás tördelése megfelelő (az akkor és a különben ágat bentebb kezdtük).

Sokágú elágazás

Elképzelhető, hogy az elágazás ágain újabb elágazások keletkeznek, a leggyakrabban a különben ágon kell újabb elágazást kezdeni:

Nézzünk egy ilyen esetet:

```
Ha feltétel_1
     akkor utasítások_1
     különben ha feltétel_2
                akkor utasítások_2
                különben utasítások_3
Elágazás vége
```

Példa:

```
Ha i<0
     akkor Ki: 'Negatív.'
     különben ha i=0
                 akkor Ki: 'Nulla.'
                 különben Ki: 'Pozitív.'
Elágazás vége
```

Az elágazás vége elhagyható, ha az oszlopos írásmód miatt egyértelmű.

A feltételek és utasítások számozása csupán azt jelzi, hogy azok különbözőek (általában). Ebben az esetben az egyes utasításágak egyenrangúak, de a feltételeket a megadott sorrendben kell végrehajtani. Szokás ezt így is írni:

```
Ha feltétel_1  akkor utasítások_1
               különben ha feltétel_2 akkor utasítások_2
                                      különben utasítások_3
Elágazás vége
```

Amennyiben a sokirányú elágazás feltételeit tetszőleges sorrendben hajthatjuk végre (akár párhuzamosan is), azt így írjuk le:

```
Elágazás
     feltétel_1 esetén utasítások_1
     feltétel_2 esetén utasítások_2
     .
     .
     feltétel_N esetén utasítások_N
     egyéb      esetén utasítások
Elágazás vége
```

Ebben az algoritmusban az igaz feltételhez tartozó utasításokat kell végrehajtani, ha több igaz is van, akkor bármelyiket, s amennyiben minden feltétel hamis, akkor az egyéb utáni utasításokat.
Látható, hogy az ilyen elágazás nemdeterminisztikus (előre meg nem határozható) algoritmust eredményezne, amit nem szeretnénk, ezért általában ezt az áttekinthetőbb írásmódot használjuk a korábbi értelmezéssel: számít a kiértékelés sorrendje.

Példa:

```
Elágazás
     i>0   esetén    Ki: 'Pozitív'
     i=0   esetén    Ki: 'Nulla.'
     egyéb esetén    Ki: 'Negatív.'
Elágazás vége
```

Az elágazás végét még oszlopos írásmód mellett sem szokás elhagyni.

Ismétlő szerkezetek (ciklusok)

Elöltesztelő ciklus:

```
Ciklus amíg ciklusfeltétel
    ciklusmag utasításai
Ciklus vége
```

A ciklusfeltétel bármilyen logikai kifejezés lehet (amely rendelkezik logikai értékkel), tehát eldönthető, hogy igaz-e vagy hamis. A ciklusmag utasításai mindaddig ismétlődnek, amíg a ciklusfeltétel igaz. Figyeljünk arra, hogy a ciklusmag utasításai között lennie kell olyannak, amely hatással van a ciklusfeltételre, különben soha nem lépne ki a ciklusból, vagy soha sem hajtaná végre a ciklusmagot.

Példa:

```
a:=1
Ciklus amíg a<=10
    Ki: a
    a:=a+1
Ciklus vége
```

Megjegyzés: A matematikában használatos ≤ jel helyett <= -t írtam, és így teszem ezt a továbbiakban is, ezzel is jelezve, hogy a kódolásnál majd így kell írni. Természetesen a szokásos relációjelek is használhatók a mondatszerű leírásnál.

Hátultesztelő ciklus:

```
Ciklus
    Ciklusmag utasításai
amíg ciklusfeltétel
Ciklus vége
```

A ciklusmag utasításai egyszer biztosan végrehajtódnak. A ciklusmag akkor ismétlődik meg, ha a ciklusfeltétel igaz (pascal nyelvben másként lesz).

Példa:

```
a:=1
Ciklus
    Ki: a
    a:=a+1
amíg a<=10
Ciklus vége
```

Jelentése: Kiírja az 1 és 10 közötti egész számokat.

A „ciklus vége” szöveget gyakran elhagyjuk, mert látszik. Vegyük észre, hogy amennyiben az eddig említett két példában az első értékadó utasításban a értékét nem 1-re, hanem 100-

ra állítjuk, akkor az elöl tesztelő ciklus magja nem fut le, míg a hátul tesztelő ciklus magja egyszer lefut.

Számláló ciklus:

```
Ciklus ciklusváltozó:=kezdőérték[1]-től végérték-ig lépésköz-évelciklusmag utasításai
Ciklus vége
```

A ciklusváltózó értéke a kezdőérték lesz. Ha ez nem nagyobb, mint a végérték, akkor végrehajtódik a ciklusmag. A ciklusváltozó értékének növelése automatikusan megtörténik a ciklusmag utasításai után, és az új megnövelt értéket vizsgálja meg, hogy kell-e még ismételni. Amennyiben a lépésköz értéke nincs megadva, akkor a növekedés mértéke 1 lesz, egyébként „lépésköz”-nyit nő. Amikor a ciklusváltozó nagyobb lesz a végértéknél, akkor az algoritmus a „ciklus vége” utáni utasítással folytatódik.

Példa:

```
Ciklus i:=1-től 10-ig
    Ki: i, i*i
Ciklus vége
```

Jelentése: Kiírja az 1 és 10 közötti egész számokat és négyzetüket.

Példa 1-től eltérő lépésközre:

```
Ciklus i:=1-től 10-ig 2-esével
    Ki: i, i*i
Ciklus vége
```

Jelentése: Kiírja az 1 és tíz közötti páratlan számokat és négyzetüket (a ciklusváltozó rendre: 1,3,5,7,9 lesz, majd 11, amikor a ciklusmag már nem hajtódik végre).

A ciklusváltozó kezdőértéke lehet nagyobb a végértéknél, ilyenkor a ciklusmag lefutása után a ciklusváltozó csökkenni fog 1-gyel (vagy a megadott értékkel), és az ismétlődés addig tart, amíg a ciklusváltozó kisebb nem lesz a végértéknél:

```
Ciklus i:=10-től 1-ig -3-asával
    Ki: i, i*i
Ciklus vége
```

Jelentése: Kiírja a 10 és 1 közötti számokat és négyzetüket hármasával haladva (a ciklusváltozó rendre: 10,7,4,1, lesz, majd -2, amikor a ciklusmag már nem hajtódik végre).

A számlálós ciklusnál a ciklusváltozó egész szám volt a példákban. A változók típusával majd a későbbiekben foglalkozunk, de fontos már itt megemlíteni, hogy számlálós ciklus ciklusváltozója olyan típusú lehet, ahol értelmezett a „következő”, „előző” fogalma. Ezért

[1] A tradícionális mondatszerű leírás egyenlőségjelet használ, és a kódolás megkönnyítése miatt mindig a legyen egyenlő jelét fogom használni, mert jobban sugalja, hogy a ciklusváltozó felveszi a kezdőértéket.

például valós szám nem lehet ciklusváltozó, de például az ABC betűi igen (értelmes az a ciklusfeltétel, hogy Ciklus i:=’A’-tól ’G’-ig).

Eljárások, függvények, operátorok

Az algoritmusok készítésének egyik legfontosabb alapelve, hogy a feladatot részfeladatokra bontjuk. Ezen részfeladatokat is algoritmussal írjuk le. A részfeladatokra bontást addig folytatjuk, amíg egy-egy ilyen részfeladat megoldása már kellően egyszerűvé válik. A „valami egyszerűt csináló” algoritmusokat **eljárás**oknak hívjuk. Érdemes eljárásként megfogalmazni azokat az algoritmus részeket, amiket egy egységként szeretnénk kezelni, többször is végre kell hajtani az algoritmus során. Az eljárások leírása pedig a következő:

```
Eljárás Eljárásnév:
    utasítások
Eljárás vége.
```

Egy eljárás meghívható a nevének leírásával, mintha utasítás lenne:

```
Utasítások_1
Eljárásnév
Utasítások_2
```

A programozási feladatok többsége alapvetően felírható az alábbi módon:

```
Program:
    Adatbeolvasás
    Feldolgozás
    Eredménykijelzés
Program vége.
```

Programunk három eljárást már bizonyára tartalmaz.

Nézzünk egy egyszerű példát, az átlagszámítást!
Az átlagolandó értékeket egy sorozatba fogjuk beolvasni, majd átlagot számítunk, és utána kiírjuk azt:

```
Specifikáció:
Feladat: Számoljuk ki N db szám átlagát!
Bemenet: N (adatok száma, egész),
         A (N elemű sorozat, elemei valós számok)
Kimenet: ATL (átlag, valós szám)
Előfeltétel: N>1 (ekkor van értelme az átlagnak)
Utófeltétel: ATL=(A(1)+A(2)+…A(N))/N

Algoritmus:

Program:
    Sorozatbeolvasás
    Átlagszámítás
    Átlagkiírás
Program vége.
```

```
Eljárás Sorozatbeolvasás:
    Be: N
    Ciklus i:=1-től N-ig
         Be: A(i)
    Ciklus vége
Eljárás vége

Eljárás Átlagszámítás:
    ATL:=0
    Ciklus i:=1-től N-ig
         ATL:=ATL+A(i)
    Ciklus vége
    ATL:=ATL/N
Eljárás vége

Eljárás Átlagkiírás:
    Ki: ATL
Eljárás vége.
```

Megjegyzés: Az átlag számításhoz nem akartam a ∑ operátort használni. Az A jelű sorozat i-edik elemére való hivatkozás a matematikában: A_i. Az algoritmusok leírásánál A(i)-t használunk, de én elfogadhatónak tartom az A[i] írásmódot is, mely a pascal kódolást segíti. A program beolvassa a sorozat elemszámát és magát a sorozatot, az ATL nevű változóban összegzi a sorozat elemit, majd elosztja az elemek számával, s végül kiírja az átlagot.
A program sajnos rossz: nem felel meg a specifikációnak, mert nem ellenőrzi, hogy a sorozat elemszámának megfelelő értéket adtunk-e meg. Írjuk át az algoritmusokat, és most törekedjünk a „felhasználóbarát” kezelésre, valamint a változók deklarációjára is!

A megváltoztatott algoritmusok:

```
Program:
    Változó: N: egész
             A(1..N: valós)    [így adom meg a sorozatot]
             ATL: valós
    Sorozatbeolvasás
    Átlagszámítás
    Átlagkiírás
Program vége.

Eljárás Sorozatbeolvasás:
    N:=0
    Ciklus amíg N<1
         Ki: 'Adja meg a sorozat elemszámát (N):'
         Be: N
    Ciklus vége
    Ciklus i:=1-től N-ig
         Ki: 'Adja meg a sorozat ',i,'. elemét:'
         Be: A(i)
    Ciklus vége
Eljárás vége

Eljárás Átlagkiírás:
    Ki: 'A sorozat elemeinek átlaga: ',ATL
Eljárás vége.
```

Látható, hogy minden eljárás használ változókat. Jó lenne megadni, hogy egy-egy eljárás milyen változókat használ, pontosabban milyen változókon keresztül kommunikál a „külvilággal”. Ezt nevezzük **paraméter**ezésnek, melynek elve a következő:
Az eljárás neve után zárójelben vesszővel elválasztva felsoroljuk azokat a változókat, amiket az eljárás felhasznál, vagy előállít (bemenő és kimenő paraméterek), megadva azok típusát (értékhalmazát) is. Az adattípusokról majd a következő fejezetben lesz szó, itt most elégedjünk meg azzal, hogy feltüntetjük, hogy milyen alaphalmazból valók az értékei.
A paraméterek azonosítója előtt megadjuk, hogy **változó**-e, vagy **konstans**. A változó értéke a paraméteren belül módosulhat, és a módosítás az eljáráson kívül is megmarad, A nem módosuló paraméterek elé írhatjuk, hogy konstans. Lehetnek olyan paraméterek, amelyek az eljárás során keletkeznek, természetesen azok is változók lesznek. Az eljárásokon belül használhatunk olyan változókat, amelyek csak az adott eljárásban használatosak (az eljáráson kívül nem látszanak), ezeket az eljáráson belül kell deklarálni. A paraméterekkel rendelkező eljárások meghívásakor zárójelben fel kell sorolni a paramétereket)!

Írjuk át ennek szellemében az algoritmust!

```
Program:
    Változó: N: egész
             A(1..N: valós)    [így adom meg a sorozatot]
             ATL: valós
    Sorozatbeolvasás(N,A)
    Átlagszámítás(N,A,ATL)
    Átlagkiírás(ATL)
Program vége.

Eljárás Sorozatbeolvasás(változó N:egész,változó A(1..N:valós)):
    N:=0
    Ciklus amíg N<1
         Ki: 'Adja meg a sorozat elemszámát (N):'
         Be: N
    Ciklus vége
    Ciklus i:=1-től N-ig
         Ki: 'Adja meg a sorozat ',i,'. elemét:'
         Be: A(i)
    Ciklus vége
Eljárás vége.

Eljárás Átlagszámítás(konstans N:egész, konstans A(1..N:valós)),
                      változó ATL: valós):
    Változó összeg: valós
     Összeg:=0
    Ciklus i:=1-től N-ig
         összeg:=összeg+A(i)
    Ciklus vége
    ATL:=összeg/N
Eljárás vége

Eljárás Átlagkiírás(konstans: ATL: valós):
    Ki: 'A sorozat elemeinek átlaga: ',ATL
Eljárás vége.
```

Az átlagot kiszámító eljárásban használtam egy helyi (lokális) változót (őszintén szólva fölöslegesen), hogy megmutathassam használatát. Az összeg változó csak az adott eljárásban létezik, másutt nem.
Vegyük észre, hogy minden eljárásban úgy neveztem el a változókat, ahogyan azokat a programban meghatároztam. Ez azonban nem kötelező, sőt néha kellemetlen is. Az eljárások fejlécében (az eljárásnév utáni zárójelben) felsorolt azonosítókat **formális paraméterek**nek nevezzük. Formálisak azért, mert az eljárás algoritmusának szövegében ezeket az elnevezéseket fogjuk használni. Az eljárás meghívásakor ezen paraméterek helyére az eljárás meghívásakor felsorolt paraméter értékek kerülnek, amiket **aktuális paraméterek**nek hívunk. A formális és aktuális paraméterek megfeleltetése a felsorolás sorrendjében történik, ezért pontosan annyi (és azonos típusú) formális és aktuális paramétert kell megadni! Az eljárás algoritmusát tehát a formális paraméterek segítségével írjuk le, míg az eljárás meghívásakor adjuk meg a végrehajtáskor használatos – aktuális – paramétereket.
Az eljárások továbbra is használhatnak olyan változókat, amelyek nem szerepelnek a paraméterek között. Ezeket a változókat azonban az eljárás fejléce alatt meg kell adni, és természetesen a korábbiakban (például a főprogramban) már deklarált változók is használhatók!

Az eljárás leírása formális paraméterekkel:

```
Eljárás Eljárásnév(formális paraméterek):
    utasítások
Eljárás vége.
```

Az eljárás meghívása aktuális paraméterekkel:

```
Eljárásnév(aktuális paraméterek)
```

A fentiek szellemében átírt program az alábbi:

```
Program:
    Változó: N: egész
             A(1..N: valós)    [így adom meg a sorozatot]
             ATL: valós
    Sorozatbeolvasás(N,A)
    Átlagszámítás(N,A,ATL)
    Átlagkiírás(ATL)
Program vége.

Eljárás Sorozatbeolvasás(változó DB:egész,
                         változó Sorozat(1..N: valós)):
    DB:=0
    Ciklus amíg DB<1
         Ki: 'Adja meg a sorozat elemszámát:'
         Be: DB
    Ciklus vége
    Ciklus i:=1-től DB-ig
         Be: Sorozat(i)
    Ciklus vége
Eljárás vége.
```

```
Eljárás Átlagszámítás(konstans DB:egész,
          konstans Sorozat(1..N: valós), változó ÁTLAG:valós):
Változó összeg: valós
     összeg:=0
    Ciklus i:=1-től DB-ig
         összeg:=összeg+Sorozat[i]
    Ciklus vége
    ÁTLAG:=összeg/DB
Eljárás vége.


Eljárás Átlagkiírás(konstans: ÉRTÉK: valós):
    Ki: 'A sorozat elemeinek átlaga:',ÉRTÉK
Eljárás vége.
```

Mint látható, még arra sem törekedtem, hogy az egyes eljárások ugyanazon tartalomra megegyező azonosítót használjanak.

A változók használatának következményei:

1. A programban megadott változók és konstansok minden eljárásban használhatók, módosíthatók. Az egyszerűbb feladatokat általában ilyen a főprogramban deklarált változókkal, úgynevezett **globális válozók**kal oldjuk meg

2. Az eljárásban megadott változók az adott eljárásban használhatók, valamint az eljárásból meghívott eljárásokban. Szoktuk ezeket a változókat **lokális változók**nak nevezni.

3. Ha egy eljárás megad (deklarál) egy az előzőekben már használt azonosítót, akkor ebben az eljárásban (és az ebből meghívottakban) a korábbi változó értéke nem lesz látható. Természetesen az eljárásból kilépve ismét látszani fog a változó korábbi értéke.

4. Az eljárásokat utasításként használhatjuk programunkban.

Jó lenne az eljárások mintájára olyan programrészeket is írni, amelyek úgy viselkednek, mint a matematikai függvények. A függvények kifejezésekben szerepelhetnek, például: X:=**SIN(3*Y)**. Természetesen lehetőségünk van ilyen algoritmusokat készíteni.

A függvény megadásának módja:

```
Függvény Függvénynév(formális paraméterek):függvényérték típusa
    utasítások
    Függvénynév := kifejezés
Függvény vége
```

Az eljárástól való egyik fontos eltérés, hogy a függvény már nem csupán a paraméterekkel kommunikál az őt meghívó program részlettel, hanem rendelkezik **visszatérési érték**kel, s ennek típusát a függvény fejlécében meg kell adni.
A visszatérési érték általában csak valamilyen egyszerű típusból származhat, de az algoritmusok leírásakor ezzel még nem törődünk.

A függvény természetesen paraméterként megadott változót is módosíthat. Gyakori, hogy a visszatérési érték egy sikeresség jelzés, míg a tényleges hatás a paramétereken keresztül történik.
A függvény végrehajtásakor a függvény nevének értéket adó utolsó utasítás lesz a függvény visszatérési értéke. Ha ezt elfelejtjük, akkor a visszatérési érték meghatározatlan. Természetesen a függvény programszövegében több helyen is szerepelhet a függvénynévre vonatkozó értékadás, a végrehajtás szempontjából utolsó határozza meg a visszatérési értéket.

A függvény használata egy kifejezésben:

```
… Függvénynév(aktuális paraméterek) …
```

Minden egyéb tulajdonság, amit az eljárásoknál felsorolunk, az a függvényre is igaz.
Írjuk át a programot úgy, hogy az átlag kiszámítása függvénnyel történjen!

```
Program:
    Változó: N: egész
           A(1..N: valós)      [így adom meg a sorozatot]
           ATL: valós
    Sorozatbeolvasás(N,A)
    ATL:=Átlag(N,A)
    Átlagkiírás(ATL)
Program vége.

Eljárás Sorozatbeolvasás(változó DB:egész,
                         változó Sorozat(1..N): valós):
    DB:=0
    Ciklus amíg DB<1
         Ki: 'Adja meg a sorozat elemszámát:'
         Be: DB
    Ciklus vége
    Ciklus i:=1-től DB-ig
         Be: Sorozat[i]
    Ciklus vége
Eljárás vége.

Függvény Átlag(konstans DB:egész,
               konstans Sorozat(1..N:valós)): valós
Változó összeg: valós
     összeg:=0
    Ciklus i:=1-től DB-ig
         összeg:=összeg+Sorozat(i)
    Ciklus vége
    Átlag=összeg/DB
Függvény vége.

Eljárás Átlagkiírás(konstans: ÉRTÉK: valós):
    Ki: 'A sorozat elemeinek átlaga:',ÉRTÉK
Eljárás vége.
```

Mivel a függvény kifejezésben használható, ezért magában a kiírásban is. A program így írható át (csak a változó részeket adom meg):

```
Program:
     Változó: N: egész
              A(1..N: valós)    [így adom meg a sorozatot]
     Sorozatbeolvasás(N,A)
     Átlagkiírás(N,A)
Program vége.

Eljárás Átlagkiírás(konstans: DB: egész, Sorozat(1..DB): valós):
     Ki: 'A sorozat elemeinek átlaga:', Átlag(DB,A)
Eljárás vége.
```

Az operátorok (műveletek) olyan függvények, melyeket a szokásostól eltérően használunk: az operátor nevét (a műveleti jelet) nem a paraméterek elé, hanem azok közé írjuk, gondoljuk például az összeadás (+) operátorra. Ilyen operátorokat is készíthetünk, de a középiskolás gyakorlatban nem lesz rá szükségünk. Csak a teljesség kedvéért szerepeljen itt egy példa, amely megadja a minimum operátor algoritmusát:

```
Művelet min(konstans a,b: valós): valós
     Másként a min b
     Ha a>b
          Akkor min:=b
          Különben min:=a
     Elágazás vége
Művelet vége.
```

A „Másként” kulcsszó után látható a művelet használatának „helyesírása”.
Eddigiekben az eljárások és függvények paraméterei változók és konstansok voltak. Paraméter azonban lehet eljárás, vagy függvény is, például:

```
Eljárás Eljárásnév(konstans x:valós, Függvény fgv(valós):valós)
     Ki: x, fgv(x)
Eljárás vége.
```

Ez az eljárás egy függvény értéktáblázatának elkészítésére jó.

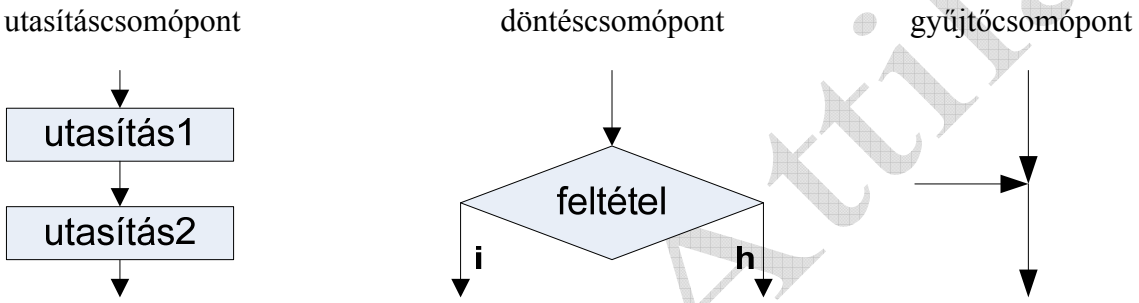
Kérdések, feladatok

- Miért jó a mondatszerű algoritmus leírás?
- Sorold fel a mondatszerű leírás alapelemeit!
- Készítsd el a másodfokú egyenlet megoldó képletének algoritmusát mondatszerű leírással! Használj benne függvényt, vagy eljárást is!
- Készíts algoritmust, amelyik beolvas egy számot, és ha az nem negatív, akkor a négyzetgyökét írja ki, ellenkező esetben pedig a négyzetét! (Használható a Négyzetgyök(X) függvény a négyzetgyök kiszámítására!
- Készíts algoritmust, amelyik beolvas egy pozitív számot, és kiírja a 3-mal osztható számokat 1 és a beolvasott szám között!
- Készíts algoritmust, amelyik beolvas egy pozitív számot, és kiírja, hogy prím-e! (A prímszámnak pontosan két osztója van: az 1, és önmaga. Ne használj nevezetes algoritmust!)
- Készíts algoritmust, amelyik beolvas egy számot (jelöljük N-nel), és ezt követően beolvas N db számot, s végül kiírja a beolvasott N db szám közül a legkisebbet!

2.1.3 Algoritmusok rajzos ábrázolása

Folyamatábra

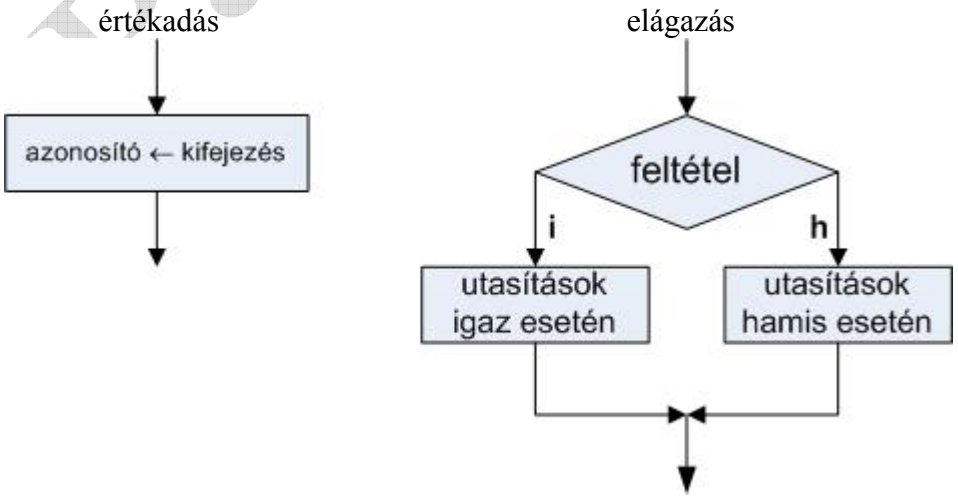
A folyamatábra a programot gráfként írja le. A programgráf egy irányított gráf, amely csomópontokból és az őket összekötő élekből áll. Egyetlen induló éllel, és egyetlen befejező éllel rendelkezik, az induló élből bármely csomópont elérhető, s bármely csomópontból el lehet jutni a befejező élhez.
A folyamatábra az alábbi háromféle csomópontot tartalmazhatja (ezek a folyamatábra alapelemei):



Utasításcsomópont: végighaladva rajta végre kell hajtani a beleírt utasítást.
Döntéscsomópont: a benne lévő feltétel igaz értéke esetén az i betűvel jelölt élen, hamis értéke esetén a h betűvel jelölt élen kell továbbhaladni.
Gyűjtőcsomopont: a befutó élek (kettő, vagy több) az egyetlen kimenő élben folytatódnak
Szokás a csomópontot kis karikával jelölni.

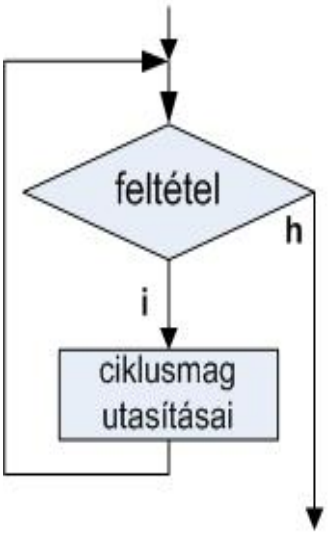
Ez a leírás jól használható az algoritmusok végrehajtásának követésére, de súlyos hátránya, hogy alapelemei nem strukturált szerkezetek, így könnyen készülhet vele nem strukturált program, amit szeretnénk elkerülni.

A folyamatábra csomópontjai segítségével az alábbi módon alakíthatók ki a hiányzó strukturált programszerkezetek:

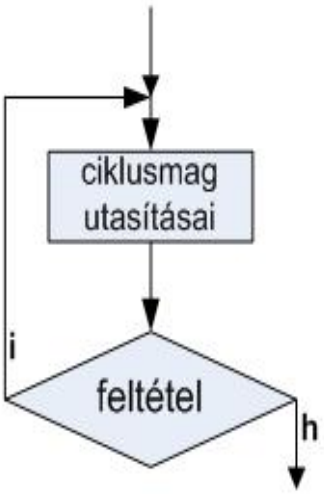


Iterációk (ciklusok):

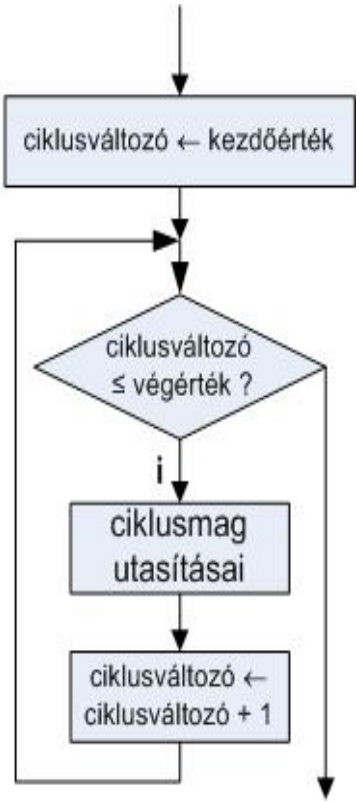
elöltesztelő ciklus



hátultesztelő ciklus

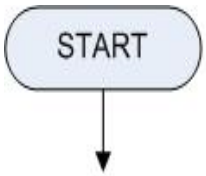


számláló ciklus

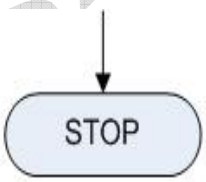


A gyakorlatban a megadott folyamatábra csomópontok kiegészülnek az alábbiakkal:

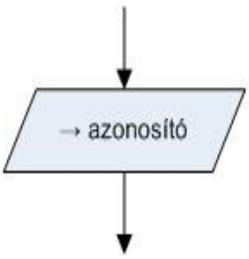
a program kezdete



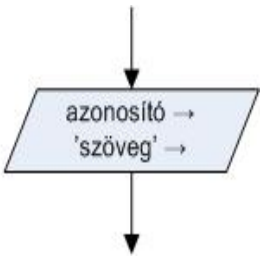
a program vége



adatbevitel

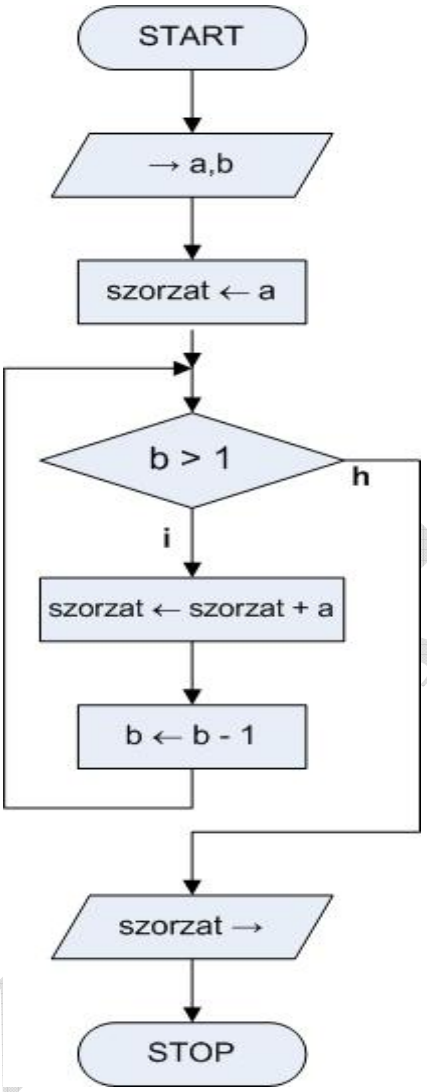


adatkivitel



A következő részben megtanuljuk, hogyan lehet egy folyamatábrával megadott algoritmust végigkövetni.

Készítsük el a pozitív egész számok szorzásának algoritmusát (a szorzást ismételt összeadásra visszavezetve)!



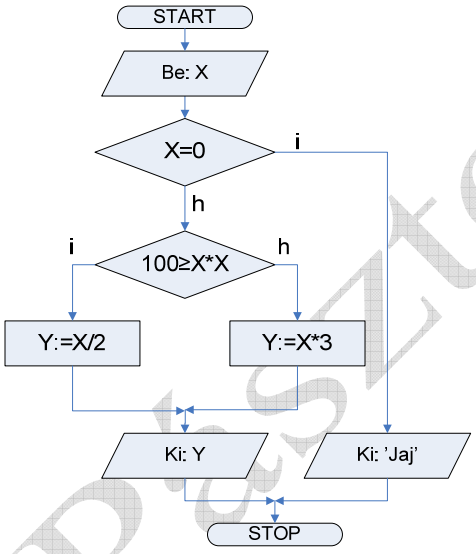
A folyamatábra kiértékeléséhez hasznos segítség lehet egy táblázat, mely a változókat ás a műveleteket tartalmazza (lépésenként kitöltve). A fenti folyamatábra értéktáblázata:

a	b	szorzat	művelet
5	3		Be: a,b
5	3	5	Szorzat := a
5	3	10	b > 1; szorzat := szorzat + a
5	2	10	b := b – 1
5	2	15	b > 1; szorzat := szorzat + a
5	1	15	b := b – 1
5	1	15	Ki: szorzat

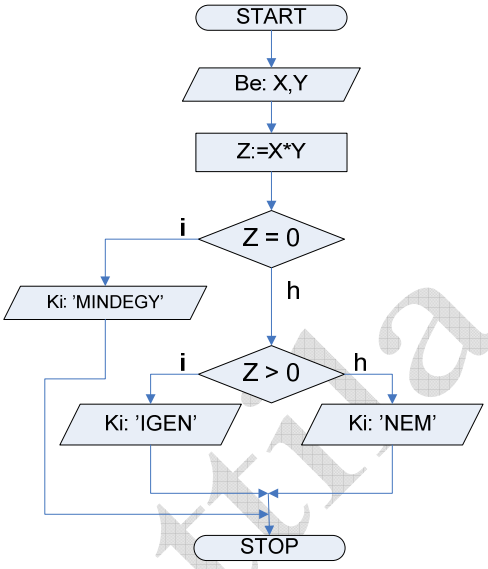
A művelet oszlopban mondatszerű elemeket használtam, de bármilyen jelölésrendszer használható, amit jól ismersz. Szabad több utasítást végrehajtásának eredményét egy sorban megjeleníteni, ha tudsz rá figyelni.

Kérdések, feladatok

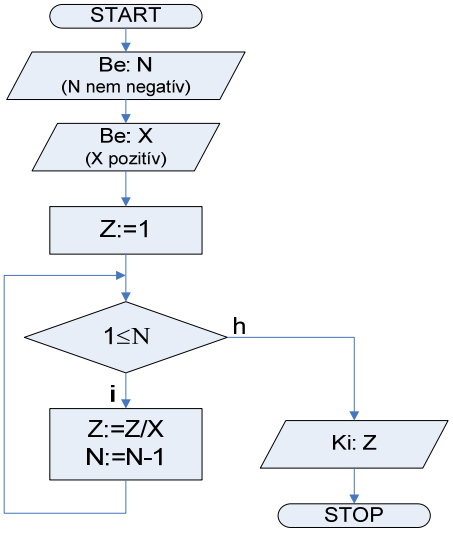
- Miért jó a rajzos algoritmus leírás?
- Hogyan ábrázolja a folyamatábra az algoritmust?
- Sorold fel a folyamatábra alapelemeit!
- Minden folyamatábra strukturált algoritmust takar?
- Készítsd el az iskolában található üdítőital automata működésének folyamatábrás algoritmusát (ne törekedj a túlzott részletességre)!
- Készítsd el a palacsinta sütés folyamatábrás algoritmusát!
- Készítsd el a másodfokú egyenlet megoldó képletének folyamatábrás algoritmusát!
- Elemezd az alábbi folyamatábrákat és válaszolj a kérdésekre!



Mit ad eredményül, ha X=0?
Mit ad eredményül, ha X=5?
Mit ad eredményül, ha X=100?
Fogalmazd meg egy mondatban, hogy mit csinál a program?



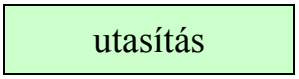
Mit ad eredményül, ha X=0 és Y=2?
Mit ad eredményül, ha X=-2 és Y=3?
Mit ad eredményül, ha X=2 és Y=4?
Fogalmazd meg egy mondatban, hogy mit csinál a program?



Mit ad eredményül, ha N=0 és X=4?
Mit ad eredményül, ha N=1 és X=10?
Mit ad eredményül, ha N=3 és X=2?
Fogalmazd meg egy mondatban, hogy mit csinál a program?

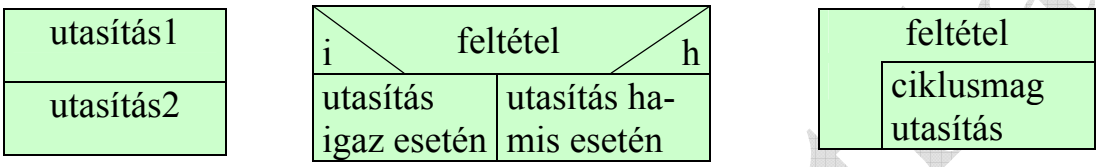
Struktogram

A folyamatábra hibáit így próbáljuk meg kiküszöbölni: a programgráfot élek nélkül ábrázoljuk, így egyetlen alapelem marad, a téglalap:



Az alapelem segítségével így építhetők fel a strukturált alapelemek:

szekvencia elágazás ciklus (elöltesztető, számláló)



Számláló ciklus esetén a feltétel az alábbi alakú:
ciklusváltozó:=kezdet-től vég-ig
például: i:=1-től N-ig

Elöltesztelő ciklus esetán a ciklusmagba lépés feltétele szerepel, például:
I<N

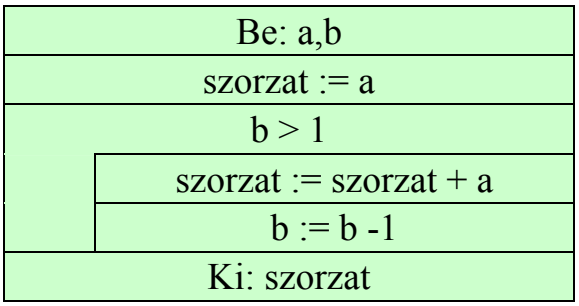
Az utasítások helyén lehet:
- egyetlen elemi utasítás,
- a három algoritmikus szerkezet valamelyike,
- eljáráshívás.

Az alábbi elemekkel szokás kiegészíteni a fenti alapelemeket:

eljárásdefinició sokágú elágazás hátultesztelő ciklus

A sokágú elágazásnál azt az ágat kell végrehajtani, amelynek igaz logikai értékű a feltétele. Csak pontosan egy feltétel lehet igaz!
Az eljárásban használt (lokális) adatokat az eljárásnév mellett soroljuk fel, típusuk megadásával.
Ez a grafikus eszköz egyértelműen csak strukturált algoritmust eredményez, így kiküszöböli a folyamatábra hibáját és megmarad a rajzos eszköz látványossága. A struktogram kiértékeléséhez is használható a folyamatábránál vázolt értéktáblázat.
Az egyes utasításokat a mondatszerű leírás segítségével, vagy a folyamatábra rajzos eszközeivel is megadhatjuk, én a mondatszerű elemeket fogom használni.

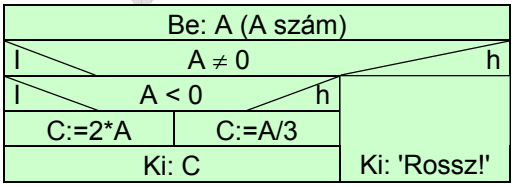
Befejezésül példaként álljon itt a pozitív egész számok szorzásának ismételt összeadásra visszavezetett algoritmusának struktogramja:



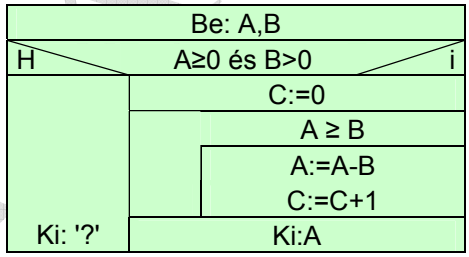
A struktogram utasításait mondatszerű elemekkel írtam le. Természetesen lehetne a folyamatábránál tanul elemeket (értékadás, be/ki) is használni.

Kérdések, feladatok

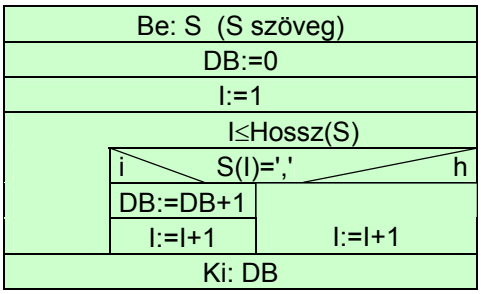
- Miért előnyösebb a struktogram, mint a folyamatábra?
- Mik a struktogram alapelemei?
- Készíts programot, amely beolvas egy számot és kiírja 2-től a megadott számig a 2-vel osztható számokat!
- Készíts programot, amely beolvas egy szöveget, és kiírja „duplázva”, azaz minden karaktert kétszer ír ki!
- Készíts programot, amely eldönti, hogy egy (pozitív) szám négyzetszám-e!
- Készíts programot, amely beolvas egy nem negatív számot (jelöljük N-nel), majd beolvas ennyi darab (N db) számot, és végül kiírja az átlagukat!
- Készíts programot, amely beolvas egy pozitív egész számot, és kiírja, tökéletes szám-e!
- Elemezd a következő struktogramokat!



Mit ad eredményül, ha A=6 ?
Mit ad eredményül, ha A=-2 ?
Mit ad eredményül, ha A=0 ?
Fogalmazd meg általánosan, hogy mit csinál a program!



Mit ad eredményül, ha A=4 és B=0 ?
Mit ad eredményül, ha A=8 és B=3 ?
Mit ad eredményül, ha A=7 és B=3 ?
Fogalmazd meg általánosan, hogy mit csinál a program!



Mit ad eredményül, ha S=’Még jőni fog, ha jőni kell’
Mit ad eredményül, ha S=’Ég a napmelegtől a kopár szik sarja’
Fogalmazd meg általánosan, hogy mit csinál a program!

2.2 Összefoglalás

Ebben a fejezetben az algoritmusok leírására koncentráltunk. Az algoritmusokat szövegesen, vagy rajzosan szokás leírni.

A szöveges leírások közül megismertük a mondatokkal történő leírás és a mondatszerű elemekkel történő leírást. Az előző előnye, hogy nem kell hozzá különleges ismeret, hátránya azonban, hogy nagyon terjedelmes, és mások számára valószínűleg nem egyértelmű, a nyelvi korlátok miatt esetleg félreérthető is.

A mondatszerű leírás előnye, hogy „szabványos" szöveges alapelemeket használ az algoritmus leírására.

A strukturált programok alapelemei (kényelmi kiegészítésekkel):

- értékadás,
- beolvasás, kiírás,
- megjegyzés,
- szekvencia (utasítások egymásutánja),
- elágazás:
    - feltétel,
    - kétágú elágazás
    - sokágú elágazs
- iteráció (ismétlés):
    - elöltesztelő ciklus,
    - hátultesztelő ciklus,
    - számláló ciklus,
- eljárás, függvény.

Rajzos algoritmus leírások közül a folyamatábrát és a struktogramot ismertük meg. A struktogram előnye, hogy alapelemi strukturáltak, míg a folyamatábra lapaelemei nem.

Megismertük az alábbi kulcsfogalmakat:

- szintaxis, szemantika,
- azonosító,
- változó, állandó,
- program,
- paraméter (formális, aktuális)

A mondatszerű leírás további előnye, hogy pascal nyelvre könnyen kódolható, ezért a továbbiakban algoritmus leírásra ezt fogjuk használni.