Pap Gáborné Szlávi Péter Zsakó László

Módszeres programozás: Szövegfeldolgozás

µlógia 14

ELTE Informatikai Kar

7. bővített kiadás

Készült az *NJSzT* gondozásában
200 példányban
Felelős kiadó: Dr. Kozma László
Sorozatszerkesztő: Szlávi Péter - Zsakó László

Tartalomjegyzék

Előszó

Kötetünk a *µlógia* sorozat 14. tagja, de valamiben első is. Nevezetesen ez az *első* olyan kötet, amely egy *szilánk*ból (a *µlógia Szilánkok 3.* kötetéből) formálódott egy teljesebb anyaggá. Több elődjénél, amelynek mindössze annyi volt célja, mint a szilánkokénak általában, hogy egy évfolyam egy tárgyának egyetlen előadását írásba foglalja. A maga korlátaival együtt a célt elérte: megismertette az Olvasót a legfontosabb fogalmakkal, egy jellegzetes látásmóddal, a téma egyetlen megközelítési módjával, kifejtetlenül hagyott problémákkal ... s föl lehetett készülni ez alapján a vizsgára.

E kötet a fentieken túllép: a témát sokoldalúbban és mélyebben elemzi; oly módon tárgyalja a szövegfeldolgozási problémakört, hogy az bővebb olvasóközönség számára legyen hasznos olvasmány; s nem utolsósorban számos kérdést világosabban, érthetőbben részletez, mint a korábbi kötet.

A "filozófia" most sem változott: minden feladatcsoport kiinduló pontja, vezérlő gondolata az az adattípus, amelyre a feldolgozás vonatkozik. Így a *szövegtípus*(ok) sajátosságainak elemzésével jutunk a legkülönfélébb feladatok megoldásához ...

A sorozat szerkesztői

Bevezetés

A szövegtípusok[1] kiemelten fontosak a típusok között, amint a következőkben látható lesz. Minden program működésének *kezdete* és/vagy *vége* az *emberrel történő kommunikáció*, amelyet karakterek megjelenítésére, befogadására képes berendezés közvetít. Tehát az információ kezdeti és végstádiuma ilyen típusú adatban realizálódik.

A program megírásakor a program írója végső fokon egy olyan részfeladatra is koncentrál, amely az *ember-gép kommunikációt* oldja meg. Ez az a speciális feladat, amelynek során a program gondoskodik az információnak *a szöveg és a belső ábrázolás közötti* ide-oda *konverziójáról*. Ezen **konverziós** feladatok egy részét a programozási nyelvek elfedik a program írója elöl, biztosítva az egyszerű közvetlen beolvasását és kiírását a pl. egész, valós stb. típusú adatoknak; másokét nem. A megoldottak esetén sajnos legtöbbször a típusellenőrzés lehetőségét is elvonja a programozótól, s hibás típusú adat beírásakor a program futása –hibakóddal és egy általában angol nyelvű hibaüzenettel– megszakad. Az *összetett struktúrájú* (rekord, tömb stb.), vagy –ugyan nem összetett, de a program írója által frissen létrehozott– *felsorolás* típusú adatok konvertálása sem megoldott automatikusan a nyelvekben[2]. Ugyanez a konverziós probléma jelentkezik pl. az adatbáziskezelés 'formázott' beolvasásnál, megjelenítésénél[3] is, csak talán aprólékosabban leküzdhető problémaként. Ebben az esetben a képernyő-klaviatúra kettőse –mint egy interaktívan működő, karakter alapú input-output *berendezés– hardvertől függően egy többféleképpen is elképzelhető típus konkrét objektuma. Néha egyszerűen karakterek „lineáris" sorozata*, vagy *sorok „szakadatlan" egymásutánja*, máskor a képernyőt *karaktermátrix*ként szemlélhetjük, megint máskor *sorok **formázott** rekord*jaként.[4] A konverziós rutinok megtervezése az esetenként furcsa

---

[1] Értsd ez alatt a karakter bázisú típusokat!

[2] A szövegfélék különleges státuszát jelzi az a tény is, hogy a háttértárolóra, vagy háttértárolóról problémamentesen elvégezhetők a be-, illetve kiviteli műveletek. Ennek magyarázata: háttértárra nem szövegesre konvertáltan, hanem belső ábrázolásának megfelelően kerülnek az adatok.

[3] A „formázott" megjelenítés alatt azt értjük, hogy egy előre kitervelt „képernyőséma, -terv" által a képernyő kijelölt tartományaiban jelennek meg „rendezetten" az információk, illetve ezek közül csak bizonyosak férhetők hozzá módosítási, beírási céllal.

[4] Érdemes észrevenni, hogy pl. a Borland és más haladó rendszerek felhasználói felületének *ablaktechnikája* a fenti formázott input/output-nak egy 3-dimenziós kiterjesztését hordozza magában. Mivel a képernyő csak 2-dimenziós ábrázolást támogat, ezért többfajta leképezést is alkalmazhatunk a 2D-re való áttérésre:
1. takarással érzékeltetni a 3. dimenziót, a mélységet;
2. időben eltolva, egymásutánisággá transzformálni a hiányzó vetületet.

szövegtípus *reprezentálását* és *implementálását* jelenti. Tehát –mint oly sokszor már– jellegét tekintve most is egy **típusspecifikációs feladat** áll előttünk.

A szöveggel[5] kapcsolatban persze nemcsak konverziós feladatok merülnek föl gyakorta, hanem sok más, természetükben, céljukban eltérő is. Ilyen típusfeladat az ún. **szűrés**, amely a szöveg bizonyos fajta jelektől való megfosztását jelenti. Természetesen a szűrés legtöbb esetben olyan jelek kiszűrésére irányul, amelyek a szöveg értelmét nem, legfeljebb *formáját* befolyásolják[6]. Ehhez a feladathoz eredménye alapján hasonlatosnak tűnik az ún. **tömörítés** feladat, amelynek valóban „eredménye" a kisebb terjedelem, de egy nem is elhanyagolható többletkritériummal, hogy ti. információ nem veszhet el. Tehát elvégezhető az inverze is: visszanyerhető az eredeti szöveg is. (Ezt nem tehetjük meg a szűrés utáni szöveggel.) A tömörítés során a szöveg egyedi sajátosságait is figyelembe kell venni.

Más feladatok éppenséggel abból adódnak, hogy az ember a szöveg megjelenésére, *formájára* koncentrál. Néhány tipikus **formázási** feladat:

- sor[7] margó(k)hoz igazítása, középre állítása;
- szöveg sorokra tördelése, amelynél külön megoldandó problémaként merül *föl a szófogalom beillesztése a karakter-* és a *sor*fogalmak közé[8];
- szöveg lapokra tördelése (ahol a lapnak sajátos „szerkezete", egyéni arculata van: fejléc, lábléc stb.).

A bevezető gondolatokból kiviláglik, hogy a problémák abból a speciális körülményből fakadnak, hogy az EMBER

1) speciális *berendezés*en keresztül kommunikál a géppel, amely *jel*enként teszi elérhetővé, beírhatóvá az információt;

2) az információt egy sajátos hierarchiában fogja föl, amelynek csak legalacsonyabb szintjén építőelem a *jel* (*karakter*); s efölött –domináns elemként– használja a *szó*, a *mondat* ... építőelemeket;[9]

---

[5] Még mindig nem a programozásbeli szövegtípus fogalmáról van szó, hanem köznapi értelemben használjuk a szót.

[6] Félreértés ne essék, a „legfeljebb" szó nem akarja a forma jelentőségét általában csökkenteni – mint ahogy ez a következő bekezdésből ki is derül–, hanem a szűrés szempontjából kisebb szerepére utal.

[7] Hagyományos *sor*fogalom, nem az ugyanilyen nevű *típusösszetételi mód.*

[8] A szófogalom egy tipikus kakukktojás, hisz amíg a karaktèr- és a sorfogalmakat a berendezés diktálja, addig a szóé az „emberi oldalhoz" tartozik.

[9] Hogy mennyire nem a jel a legfontosabb, mi sem bizonyítja jobban, mint az a kísérlet, amely során kiderült, hogy egy magyar szövegből egyre több és több, véletlenszerűen kiválasztott jelét elhagyva kb. 40%-os véletlenszerű „szűrés" után vált a szöveg nehezen érthetővé. (A francia esetén ugyanez kb 55%-nál tapasztalható.)

3) a megjelenő információval szemben támaszt pusztán *esztétikai* kívánalmakat is, amely információt nem hordozó „töltelékjelek" bonyolult elhelyezésével oldható meg;

és a közvetítő BERENDEZÉS

4) rákényszerít a szövegre további hierarchikusan rendezhető fogalmakat, mint a *sor*, a *lap* és a *puffer*, s a nagyobbik baj, hogy ezek nemigen *„harmonizálnak" az ember alkotta 2)–beli fogalmakkal.*[10]

5) megkívánhatja a szöveg lehető legtömörebb ábrázolását is.

Az eddigi filozófiánknak megfelelően a szövegeket mint speciális típusba tartozó objektumokat tekintjük, amelyek kezelése a típushoz rendelt tevékenységekkel valósítható meg. Megadjuk tehát a szövegféleségek típus-specifikációját; helyesebben: egy típuscsalád specifikációit. (E pontosítás azért fontos, mert szó sincs arról, hogy önmagában lezárt modulokat terveznénk, ugyanis ez nemigen lehetséges. Itt a modul csak egy csoportosítási lehetőséget kínál föl. A modulok keresztül-kasul hivatkoznak egymásra, s épp ezért tituláltuk típuscsaládnak: az egyes modulok szoros „családi kapcsolatban" állnak egymással.) A típus-specifikációt két „szinten" definiáljuk. Az elsőben a programozási nyelvekben –többé-kevésbé– szokásosként körvonalazzuk, a másikban kissé általánosítjuk ezt.

A szövegféleségeket három típuskategóriába soroljuk:

- *karakter*,
- *szöveg*[11],
- *szövegfile*.

A típusokat oly módon definiáljuk, hogy megadjuk először az „elvárásainkat" kifejező definíciós *modult*, majd -néha többféleképpen is- a megvalósító *modult*.

[10] Ütköznek azokkal, tagolási konfliktust létrehozva.(L. µlógia 12!)
[11] A továbbiakban e fogalom a *típust* jelenti.

I. A szövegtípusok szűkebb értelmezése

azaz, amit a „hagyományos nyelvek beszélnek"

1. Karaktertípus

```
ExportModul Karakter:
  Függvény
    Sorszám[1](Konstans c:Karakter): Byte[2]
    Karakter(Konstans c:Byte): Karakter
    Következő(Konstans c:Karakter): Karakter
    Előző(Konstans c:Karakter): Karakter
  Eljárás
    Olvas(Változó f:InpSzövegFile, Változó c:Karakter)
    Ír(Változó f:OutSzövegFile, Konstans c:Karakter)
  Infix Operátor
    :=(Változó c1:Karakter, Konstans c2:Karakter)
    =(Konstans c1,c2:Karakter): Logikai
    <(Konstans c1,c2:Karakter): Logikai
Modul vége.
```

Kétféle megvalósítást is elképzelünk. Az egyik a „hagyományos", azaz minden karakternek megfelel egy, valamely rögzített karakterkészletbeli *egybyte-os kód*. A másik szerint pedig a karakterekhez *változó bithosszúságú kód*ot rendelünk.

Az elsőnek érdekes speciális esetei a telexkódból származó karakterkészlet-váltó kódok. A telexkód egy ötbites kód, azaz benne maximum 32 jelet lehetne ábrázolni. Ha csak az angol ABC nagybetűit, valamint a számjegyeket kellene így kódolnunk, az is már 36 karaktert jelentene. Ennél a kódnál azt a megoldást találták ki, hogy fenntartanak két speciális kódot, a *betűváltót* és a *számváltót*. Egy szövegben a *betűváltó* utáni kódok betűknek felelnek meg, s ha érkezik egy *számváltó* karakter, azután ugyanazok a kódok, amelyek eddig betűt jelentettek, most számjegyeket fognak jelölni. Ugyanezt az ötletet használják fel nagyon sok szövegszerkesztőben a különböző *stílusú* karakterek kódolásához, így például ugyanaz a kód tartozhat az „a", „a", „*a*", „**a**", „$_a$" ... karakterekhez.

A második változat az ötletét a jól ismert Morse-féle kódból veszi, amelyben az egyes jeleknek előre lerögzített, de változó hosszúságú „bitsorozat" van megfeleltetve. Nyilván oly módon, hogy az egyes jelek a kiindulásul választott angol nyelvben nem egyforma gyakorisággal fordulnak elő, s így ha azokat a jeleket rövid

---

[1] Más néven: **Kód**, megint más néven: **Rend**.

[2] Látnivaló, hogy ez az eredménytípus nem mindig értelmes, jól definiált. A választott reprezentáció dönti el.

bitsorozattal látjuk el, amelyek gyakran fordulnak elő az angol szövegekben, akkor a szövegek kevés bittel leírhatókká válnak. A modul „kifejtését" itt sem részletezzük. Mert nemigen lehetséges úgysem precízen, hiszen olyan alapszinten kellene kifejeznünk magunkat, amelyre nem alkalmas nyelvünk. Így tehát csak *nem formálisan* jelezzük.

Reprezentáció:

Minden jelnek egy konstansként *rögzített bitsorozat* feleljen meg. E rögzítési sorrendet lehet –jobb híján– a rendezés definiálásának is tekinteni. (Hisz itt a kódból nem lehet kiindulni.)

Az alábbi táblázat összefoglalja a Magyarországon annak idején használt Morse-féle kódot.

Az abc betűi:		A számjegyek:	Az írásjelek:
a •–	n –•	1 •————	'.' •–•–•–•
á •––•–	o –––	2 ••–––	';' ––••––
b –•••	ö –––•	3 •••––	':' –––•••
c –•–•	p •––•	4 ••••–	'?' ••––••
d –••	q ––•–	5 •••••	
e •	r •–•	6 –••••	
é ••–••	s •••	7 ––•••	
f ••–•	t –	8 –––••	
g ––•	u ••–	9 ––––•	
h ••••	ü ••––	0 –––––	
i ••	v •••–		
j •–––	w •––		
k –•–	x –••–		
l •–••	y –•––		
m ––	z ––••		

A Morze-abc (a Természettudományi Kislexikonból).

2. Szövegtípus

A szövegtípus már az előbbinél színesebb lehetőségeket biztosító típus, ez persze annak a következménye, hogy az előbbire épülő összetetett struktúra. Az összetett típusok között különleges szerepet tölt be: amíg a tömb, a rekord, a halmaz (és

sokan mások) *típuskonstrukciós eszközök*, addig a szöveg egy *konkrét típus*, amely egy programozási nyelvekben viszonylag ritka típuskonstrukciós eszköz –az *absztrakt sorozat*képzés– egy alkalmazása.

Az elvárásaink minimuma:

```
ExportModul Szöveg:
  Konstans
    MaxSzövegHossz : PozEgész(???[3])[4]
  Függvény
    Hossz(Konstans s:Szöveg): Egész
    Része?(Konstans mi,minek: Szöveg): Logikai
```

Többféle elképzelés létezik a szövegtípus *szelekciós műveleteire* (programozási nyelvek általában ezek közül választanak egyet):

```
A.[5] Első(Konstans s: Szöveg): Karakter
      Elsőutániak(Konstans s: Szöveg): Szöveg
B.[6] Eleje[7]:Konstans s: Szöveg, ig: Egész): Szöveg
      Vége[8]:Konstans s: Szöveg, tól: Egész): Szöveg
      Része[9]:Konstans s: Szöveg, tól,ig: Egész): Szöveg
C.[10] Jele (Konstans s: Szöveg, dik: Egész): Karakter
         [Különleges probléma a SorVégjel kettős (Cr+Lf)
          volta. Értelmezés: ezt is egyetlen jelnek
          tekintjük, amely karakteres értéke: Cr legyen.]
```

Ezeket külön-külön használva, de akár egymással keverve (sőt az összeset megvalósítva) is definiálhatjuk a szövegtípust.

```
Infix Operátor
  +[11](Konstans s1,s2: Szöveg): Szöveg
  +(Konstans s1: Szöveg, c: Karakter): Szöveg
  +(Konstans  c: Karakter, s1: Szöveg): Szöveg
  :=(Változó s1: Szöveg, Konstans s2: Szöveg)
  :=(Változó s: Szöveg, Konstans c: Karakter)
  =(Konstans s1,s2: Szöveg): Logikai
  <(Konstans s1,s2: Szöveg): Logikai
```

[3] valamilyen, előre rögzített érték; ...a továbbiakban is elő fog fordulni ugyanilyen értelmezéssel a '???'jelölés

[4] Ez a konstans csak bizonyos ábrázolás esetén szükséges.

[5] LOGO-stílus.

[6] BASIC-stílus.

[7] Másként: **Balrész.**

[8] Másként: **Jobbrész.**

[9] Másként: **Közepe.**

[10] Pascal-stílus.

[11] Függvényszerűen is szokták definiálni: **Konkatenáció** vagy **Egymásután.**

```
Eljárás
  Olvas(Változó f: InpSzövegFile[12],Változó c: Szöveg)
    [Problematikus: mi legyen a „szöveghatár";
     szokásosan a SorVégjel.]
  Ír(Változó f: OutSzövegFile, Konstans c: Szöveg)
Modul vége.
```

A megvalósításnak most is többféle útja képzelhető el. Az első reprezentációs „alapja":

```
Típus Szöveg = Lista(Karakter))
```

[A lista ábrázolható tömbként vagy láncoltan. Ha tömbre építjük, akkor is megspórolhatjuk a hossz tárolását egy terminálló, speciális jel definiálásával, s utolsóként való beírásával (C-stílus).]

A második megvalósítás nagyon hasonlít erre. Annak kimondott „bevallása" van mögötte, hogy az előbbi –bár lehetővé teszi, de– nem túl gazdaságos, ha a *listát* láncoltan ábrázoljuk; s így gyakorlatilag a *tömb*re vezethető vissza, ami az alábbi variáns:

```
Konstans MaxSzövegHossz : PozEgész(???)
Típus Szöveg = Rekord
                 (hossz: 0..MaxSzövegHossz,
                  jel  : Tömb(1..MaxSzövegHossz:Karakter))
```

Megjegyezzük, hogy

1) itt most a hossz tárolása szükséges, ha nem a *terminálójeles* változatot definiáljuk[13], mivel nincs más mechanizmus, ami kihasználható lenne az aktuális elemszám meghatározására (ellentétben a listával);

2) a második karakteres reprezentáció (ti. a változó bithosszúság) esetén meglehetősen nehéz –bár nem lehetetlen– az i. karakter elérése, hisz a változó hossz megakadályozza a közvetlen címkiszámítást. Ez esetben implementációs szinten végülis listává „degradáltuk" a tömböt, azaz tömbként „álcáztunk" egy listát.

A harmadik lehetőség:[14]

```
Konstans MaxSzövegHossz : PozEgész(???)
Típus Szöveg    = Rekord
                    (hossz: 0..MaxSzövegHossz,
                     tart : MunkaCím)
      MunkaCím  = Jelek'Mutató
      Jelek     = valamilyen karaktersorozat-ábrázolás
```

[12] Speciális esetben maga a klaviatúra.
[13] C-stílusú szövegábrázolás.
[14] A BASIC nyelvjárások által preferált módszer.

3. Szövegfile-típusok

A többesszámnak –ahogy a szekvenciális file-nál is volt– „csak" annyi a jelentősége, hogy élesen megkülönböztesse az inputként, illetve az outputként használandó szövegfile-okat. (Megkülönböztetendők, hiszen más műveletek tartoznak – még ha csak részben is– az egyikhez és a másikhoz.) E típusok nyelvi érdekessége, hogy ebbe beleértenek rendszerint olyan műveleteket is, amelyek már logikusak lettek volna az előbbi szövegtípus esetén is. Nevezetesen mód szokott lenni ezen összetett adatot olyan egységekben is „látni", kezelni, ami egy „makróbb" fogalomhoz, a *sor*fogalomhoz[15] kapcsolódik.

```
ExportModul InpSzövegFile:
  Eljárás
    Megnyit[16](Változó f: InpSzövegFile,
                Konstans fnév: Szöveg)
    Lezár(Változó f: InpSzövegFile)
    Olvas(Változó f: InpSzövegFile, c: Karakter)
    SorOlvas(Változó f: InpSzövegFile, s: Szöveg)
  Függvény Vége?(Változó f: InpSzövegFile): Logikai
           Hibás?(Változó f: InpSzövegFile): Logikai
Modul vége.
ExportModul OutSzövegFile:
  Eljárás
    Megnyit(Változó f: OutSzövegFile,
            Konstans fnév: Szöveg)
    Lezár(Változó f: OutSzövegFile)
    Ír(Változó f: OutSzövegFile, Konstans c: Karakter)
    SorÍr(Változó f: OutSzövegFile, Konstans s: Szöveg)
  Függvény Hibás?(Változó f: OutSzövegFile): Logikai
Modul vége.
```

Az ábrázolásukról (összevontan):

```
Konstans MaxSzövegHossz : PozEgész(???)
```

[15] A sor itt most nem az ugyanilyen nevû struktúráló eszközt jelenti.

[16] Ahogyan a file-típusnál, úgy itt is használunk algoritmusleírásbeli egyszerûsítéseket. Ha például két file-t is meg szeretnénk nyitni, s tervezéskor a file-ok nevével nem akarunk foglalkozni, akkor a

```
Megnyit(f,'elsõ'); Megnyit(g,'Második')
```

eljáráshívások helyett a

```
Nyit(f,g)
```

rövidebb írásmódot használjuk.

```
Típus SzövegFile=Rekord
                    (hossz:0..MaxSzövegHossz,
                     adat:SzekvenciálisFile(Karakter))
```
[17]

vagy a sokkal gyakoribb eset

```
SzövegFile = SzekvenciálisFile(Karakter)
```
[18]

További fogalomként vetődött föl a köznapi értelemben használt *sor* fogalma is (hogy nagyobb legyen a zavar). Így ennek „definícióját" is meg kell adni.

```
Sor = karakterek sorvégjellel lezárt sorozata
SorVégjel = Cr+Lf
Lf = Karakter(10)
Cr = Karakter(13)
```

Vegyük észre a következő –eddig kimondatlan– elvárásunkat a szövegfile-ról! A szövegfile sorfogalmat (is) használ, s ezért akár mint sorok egymásutánját (is) feldolgozhatjuk. Mivel minden sorukat sorvégjel zárja le (definíció szerint), így a file utolsó értelmes karaktere is egy *sorvégjel* kell legyen. A szövegfile-ok ezen tulajdonságát a feldolgozó programok gyakran ki is használják.

Másik problematikus helyzet: a „karakteres" Olvas művelet hogyan működjön a sor végén, azaz amit mi SorVégjelnek tituláltunk nem 1, hanem 2 jelből áll, azaz *nem fér bele a karakter* típusba. Állapodjunk meg ezért abban, hogy az Olvas művelet érzékeny erre az anomáliára, s ekkor –mondjuk– a Cr karakterrel tér vissza, bár az Lf-t is beolvasta.

[17] Itt vezetjük vissza a már definiált szekvenciális file-ok fogalmára (a leírásban most különbséget nem téve *input* és *output* file-ok között.

[18] A végét egy speciális *file-végjel* jelzi.

II. A szövegtípusok általánosabb értelmezése

Ebben a fejezetben kibővítjük az egyes szövegféleségek típusértelmezését azzal, hogy a korábban megfogalmazott „minimum" fölé képzelünk jó néhány olyan tevékenységet, ami –bár gyakran fölmerül megoldanivalóként– nem szokták a típushoz tartozónak nyilvánítani. Mi éppen ezt a kapcsolatot mondjuk ki bátran az alábbiakban.

1. Karaktertípus

Az általánosítás egyik lehetséges irányát már érintettük, amikor a *változó bithosszúságú kód*ok lehetőségét említettük. Most egy másik lehetséges utat járunk be, amit a *nemzeti jelkészletek* léte tesz szükségessé. Köztudomású, hogy ahány nyelv majdnem annyi abc van; és mindegyik egyenrangú jogokat követel magának a számítástechnikában. Márpedig a szabványosított jelkészletek angolszász eredetűek.

Az új abc-k bevezetése két problémát vet föl:

1. Új jelek *beillesztés*e, illetve más, fölösleges jelek *kihagyás*a.

2. A jelkészletbeli *rendezés* újradefiniálása.

Az első problémán nagyon könnyen keresztül lehet siklani egy kompromisszummal. A fölöslegesnek –vagy legalábbis nem nagyon fontosnak– kikiáltott jelek helyére ültessük be a kívánatos jeleket. Ezt tette az IBM is, amikor maga adott ajánlatokat nemzeti jelkészletekre. Kibővítette az ASCII készletet 256 jelre: itt helyet kaptak az angol abc kis- és nagybetűi és természetesen szokásos egyéb jelek (írásjelek, számjegyek stb.) mellett grafikus jelek, valamint néhány olyan nem angol nyelvbeli jel, mint a német umlautos ä, a „scharfes ß", a franciák tompa ékezetes č-je (accent grave e), illetve néhány –a matematikai szövegekben előszeretettel alkalmazott– görög betű. Mivel ez így –az egyes nyelvek szempontjából– teljesnek nem volt tekinthető az operációs rendszerhez adnak teljesebb nemzeti jelkészleteket is.

A második probléma komolyabban ellenáll; ugyanis az előbbi nemzeti jelkészletekben –sajnos– a *programkompatibilitás* érdekében meghagyták azokat a jeleket az eredeti helyükön, amelyek változatlanul e nyelvi készletbe is beletartoznak. Emiatt viszont az abc-szerinti rendezés mást jelent egy rendező programnak, mint amit az adott nyelv szabályai előírnának.

A megoldás: egy új <-reláció megalkotása, amelyet célszerű kellően általánosra tervezni. Alapja a következő gondolat lehet. A „beépített" és az új abc-sorrend egymáshoz rendelése egy táblázattal történhet, amelyet az új rendezés használ:

```
Változó[1] rKód: Tömb(Karakter: Byte)
Infix Operátor <(Konstans c1,c2: Karakter): Logikai
  Másként Kisebb
  Kisebb:=rKód(c1)<rKód(c2)
Operátor vége.
```

Így a magyar abc-re gondolva igaz lehet, hogy

Kód['A']<Kód['Á']<Kód['B'], annak ellenére, hogy 'B'<'Á'.

Az új betűk beillesztésével csak félig-meddig végeztünk, hiszen ezzel az ékezetes betűk „beintegrálására" találtunk gyógyírt, de még a kettős –többes– jelekre nem. Az ilyenek beillesztése megint megoldható egy másik –nem fontos– jel kihagyásával, annak helyére. Ekkor gondoskodni kell arról, hogy a karakterbeolvasó és kiíró eljárások figyeljék e kettős –többes– jeleket és azonmód helyettesítsék a neki megfeleltetett belső kóddal, illetve visszafelé ...

Példa:

'$'→'Ft', '&'→'cs' ... konvenció mellett a következő szöveg-hozzárendelés igaz: '188 Ft-os csacsi.'→'188 $-os &a&i.'

A következő eljárás a bemenetként kapott kettős vagy többes betűket (is) esetleg tartalmazó Sz szöveg első betűjét választja le, s teszi a C karakterbe. A C-ben lesz tehát az első betűnek az ABC transzformációs tömb által kódolt párja, s az Sz már e betűt nem fogja tartalmazni.

```
Változó ABC: Tömb(Karakter: Szöveg)
Eljárás ABCTranszformáció(Változó  Sz: Szöveg,
                                   C: Karakter):
  C:=min'Karakter; UjSz:=Eleje(Sz,Hossz(ABC(C)))
  Ciklus amíg UjSz≠ABC(C)[2]
    C:=Következő(C); UjSz:=Eleje(Sz,Hossz(ABC(C)))
  Ciklus vége
  Sz:=Vége(Sz,Hossz(ABC(C))+1)
Eljárás vége.
```

A többes jelekhez olyan karaktereket kell hozzárendelni a fenti eljárás alapjául szolgáló ABC-ben, amelyek kódjai a normál karakterek kódjai *előtt* vannak (vagy legalábbis *hamarabb* kell megtalálnia a kereső eljárásnak). Így például az ABC

[1] Arra való utalás, hogy dinamikusan átértelmezhető. (Bár egy nyelvre kötött, konstans.)
[2] A C≤max'Karakter feltétel elhagyható, hiszen a keresett karakter biztosan bent van a szövegben.

tömbben előbbre helyezzük az „SZ", mint az „S" betűt. (Ellenkező esetben az S-t véli fölfedezni az SZ-ben is.)

Ez a megoldás azonban nem minden esetben működik tökéletesen. Ugyanis –ha a magyar nyelvre gondolunk– létezhetnek olyan szavak, amelyekben a két egymást követő jel nem jelent kettős betűt, a beolvasó eljárás viszont ekként kezeli. (Ilyen helyzet adódhat szavak képzésekor, pl. egy „z"-re végződő szó, mint a „száraz", '-ság/-ség' képzésekor az egymás mellé kerülő „z" és „s" betűpárt „zs"-ként kódolva ábrázoljuk a memóriában; ami ha nem is mindig okoz bajt, de mindenképpen hamis.)

A rendezést még ezután is el kell végezni, de ez már könnyű az előbbiek után (hisz az abc-transzformáció után minden betű egyetlen jelből áll).

A rendezés „témakörhöz" még egy további adalék: egy újabb módszert lehet körvonalazni annak a segédcélnak a megfogalmazásával, hogy –a kompatibilitás maximális fenntartása érdekében– minden jel *kódja* maradjon az, ami korábban volt, csak a közéjük szúrást oldjuk meg új ötlet bevetésével. Az alapötlet az, hogy ha pl. az „á" az „a" és a „b" közé illik, akkor az ő kódja legyen a kettő között, vagyis mondjuk 65.5. Ez azt jelenti, hogy a kód már nem byte-os egész-szám, hanem valamiféle valós.

Például az ÁLMOS szöveget reprezentálja az eredeti kódolás szerint a következő byte-sorozat:

143[3], 76, 77, 79, 83

s kétbyte-os kóddal ugyanez így néz ki:

65, 128, 76, 0, 77, 0, 79, 0, 83, 0

Ez utóbbi kódolásnál a mellékelt szabály szerint rendeltük a byte-párokat a betűkhöz (l. az ábrát).

```
'A'→65  0
'Á'→65  128
'B'→66  0
'C'→67  0
...
'O'→79  0
'Ó'→79  128
'Ö'→79  160
'Ő'→79  192
'P'→80  0
...
```

Ha az eredeti kód hétbites lenne (ilyen volt az ASCII is a kibővítés előtt), akkor elegendő lenne csak a „különleges" jeleket kétbyte-osítani, hiszen egyértelműen visszafejthető a szöveg az alapján, hogy a kétbyte-os jelek második byte-ja≥128. (Feltéve persze, hogy a két „eredeti" betű közé legfeljebb egy új illesztendő.)

[3] A nagy Á betű a CWI szabvány szerint.
A CWI kódtábla:

	á	é	í	ó	ö	ő	ú	ü	ű
Kisbetű	160	130	161	162	148	147	163	129	150
nagybetű	143	144	141	149	153	167	151	154	152

Ezen új karaktertípus használatához meg kell adnunk olyan függvényeket, amelyek egy ily módon reprezentált szöveget teljesen „kétbyte-osított" ábrázolású karakterekre bontanak.

```
Függvény Első(Konstans Sz: Szöveg[4]): Duplakód
  Ha Hossz(Sz)>1 és Rend(Jele(Sz,2))>127
       akkor  Első:=Eleje(Sz,2)
     különben Első:=Eleje(Sz,1)+Karakter(0)
Függvény vége.

Függvény ElejérőlElhagy(Konstans Sz: Szöveg): Duplakód
  Ha Hossz(Sz)>1 és Rend(Jele(Sz,2))>127
       akkor  ElejérőlElhagy:=Vége(Sz,3)
     különben ElejérőlElhagy:=Vége(Sz,2)
Függvény vége.
```

Ezen függvények segítségével megadhatók az előző fejezetben definiált, a karaktertípuson értelmezett függvények ehhez az ábrázoláshoz is.

(Észrevehetjük, hogy a fent javasolt ábrázolása a „valós számoknak" nem más, mint egy *fixpontos* alakja: ***egészrész törtrész***. Itt mind az egészrésznek, mind a törtrésznek 1-1 byte jut.)

2. Szövegtípus

A további típusoknál föltesszük az egyszerűség kedvéért, hogy a karaktertípus szintjén a jel-kód hozzárendelés kölcsönösen egyértelmű, tehát a kettős vagy többes betűkről elfeledkezünk. ...

Az alábbi szövegtípus általánosítás célja, hogy figyelembe vegyük azokat a „makróbb" egységeit a szövegnek, amiket az ember tesz hozzá a megértés során, illetve amiket a közvetítő közeg (a berendezés) kíván meg. Így tehát bevezetjük a *szó*, a *szóelválasztó*, illetve a *sor*, *sorvég*, a *lap*, *lapvég* ... fogalmakat. (A '...' arra utal, hogy akár folytatható is ezek sora.)

A szöveg „humán" fogalmai:[5]

1) **Szó**: szóelválasztójelekkel határolt jelsorozat (maga az elválasztójel már nem része a szónak);

   Szóelválasztójelek: {' ', Tab} ∪ **Központozás** ∪ Végjelek
   Központozás: {',', ';', '.', '?', '!', ':'}
   Végjelek: {SorVégjel, LapVégjel, SzövegVégjel}

[4] Értsd: duplakódú szöveg.

[5] Az alábbiak nem akarnak precíz definíciók lenni, inkább csak körvonalazzák a fogalmat. Ajánlatos ettől függetlenül az egyes esetekben a leírtakból „kilógó" eseteket összegyűjteni.

Megvalósításuk egy lehetséges változata olvasható az alábbiakban.

```
SorVégjel: Cr + Lf  ∪ Cr
  Lf: Karakter(10)
  Cr: Karakter(13)
LapVégjel: Ff
  Ff: Karakter(12)                    {CTRL+L}
SzövegVégjel: Karakter(26)            {CTRL+Z}
```

2) **Sor**: tetszőleges hosszúságú, végjeleket nem tartalmazó jelek sorozata, amelyet *SorVégjel* zár le (a *SorVégjel* így nem része a sornak); tehát

```
Típus Sor = Szöveg [Típusinvariáns: s:Sor:
          ∀i∈[1..Hossz(s))]: Jele(s,i)∉{SorVégjel,
            LapVégjel,SzövegVégjel}]
```

3) **Lap**: tetszőleges számú sorok sorozata, amelyet *LapVégjel* zár le; azaz

```
Típus Lap = Szöveg [Típusinvariáns: l:Lap:
          ∀i∈[1..Hossz(l)]: Jele(s,i)∉{LapVégjel,
            SzövegVégjel}]
```

4) **Dokumentum**: tetszőleges számú lapok sorozata, amelyet *SzövegVégjel* zár le, azaz

```
Típus Dokumentum = Szöveg [Típusinvariáns:
          d:Dokumentum: ∀i∈[1..Hossz(d)]:
                  Jele(d,i)≠SzövegVégjel]
```

A típushoz tapadó többletműveletek ezen makróegységek kiválasztását (és egyéb, velük kapcsolatos műveleteket) hivatottak elvégezni. Megelégszünk azzal, hogy az egyes egységekből álló szöveget *sorszerű* sorozatként kezeljük (korlátozott számú sorművelet fölhasználásával)[6].

Először készítsük el a *dokumentum* típus szelekciós és konstrukciós műveleteit:

```
Típus Elválasztók=(' ',Tab,',','.','?','!',':',';',
                  SorVégjel, LapVégjel, SzövegVégjel)[7]
Konstans SzóElválasztóJelek: Halmaz(Elválasztók)
                                (' '..SorVégjel)
Eljárás ElsőSzó(Változó teljes: Sor,
                  eszó,elválasztás: Szöveg):
```

[A *teljes* szövegből, ami egy *sor*, leválasztja az *első szót* és az *első elválasztójel*-sorozatot. A leválasztott szövegrész nem szerepel a *teljes* szövegben a visszatérés után.]

[6] Szelekciós műveletek: *ElsőSzó, ElsőSor, ElsőLap...*
Konstrukciós műveletek: *SorVégére, LapVégére, DokumentumVégére...*

[7] Bár elképzelhető lenne olyan reprezentáció is, amely lehetővé tenné ezek megváltoztatását, újradefiniálását.

```
  eszó:=''; Köv:=Eleje(teljes,1)
  Ciklus amíg Köv∉Szóelválasztójelek[8]
    eszó:+Köv; teljes:=Vége(teljes,2)
    Köv:=Eleje(teljes,1)
  Ciklus vége
  elválasztás:=''; Köv:=Eleje(teljes,1)
  Ciklus amíg Hossz(teljes)>0 és Köv∈Szóelválasztójelek
    elválasztás:+Köv; teljes:=Vége(teljes,2)
    Ha Hossz(teljes)>0 akkor Köv:=Eleje(teljes,1)
  Ciklus vége
Eljárás vége.
```

```
Eljárás SorVégére(Változó teljes: Sor,
                  Konstans szó,elválasztás: Szöveg):
    [A teljes szöveg végére, ami egy sor, hozzáteszi a szót és az elválasztást]
  teljes:=teljes+szó+elválasztás
Eljárás vége.
```

```
Eljárás ElsőSor(Változó teljes: Lap, esor: Sor):[9]
    [Leválasztja a teljes szöveg, ami egy lap, első sorát: vagyis annyi szót –a
     követő elválasztójeleivel együtt– átmásol az esorba, amennyi az első Sor-
     Végjelig tart.]
  esor:=''; Köv:=Eleje(teljes,1)
  Ciklus amíg Köv≠SorVégjel
    esor:+Köv; teljes:=Vége(teljes,2)
    Köv:=Eleje(teljes,1)
  Ciklus vége
  teljes:=Vége(teljes,2)
Eljárás vége.
```

```
Eljárás LapVégére(Változó teljes: Lap,
                  Konstans usor: Sor):
    [A teljes szöveg végéhez, ami egy lap, hozzáilleszti az usor sort és egy
     SorVégjelet.]
  teljes:=teljes+usor+SorVégjel
Eljárás vége.
```

```
Eljárás ElsőLap(Változó teljes: Dokumentum, elap: Lap):
    [Leválasztja a teljes szöveg, ami egy dokumentum, első lapját: vagyis annyi
     sort –a követő SorVégjellel együtt– átmásol az elapba, amennyi az első
     lapvégjelig tart.]
  elap:=''; Köv:=Eleje(teljes,1)
  Ciklus amíg Köv≠LapVégjel
    elap:+Köv; teljes:=Vége(teljes,2)
    Köv:=Eleje(teljes,1)
  Ciklus vége
  teljes:=Vége(teljes,2)
Eljárás vége.
```

[8] Kiválasztás tétel, mivel garantált, hogy valamilyen elválasztó jel van a végén.

[9] Itt az **ElsőSzó** eljárással ellentétben nem kell visszaadni az elválasztójelet, mert itt ez most egyértelmű.

```
Eljárás DokumentumVégére(Változó teljes: Dokumentum,
                         Konstans l: Lap):
```
[A *teljes* szöveg végére, ami egy *dokumentum*, hozzáteszi az *l* lapot és egy lapvégjelet.]
```
  teljes:=teljes+l+LapVégjel
Eljárás vége.
```

A következő eljárások egy *képernyő sor*nyi szöveggel végeznek műveleteket.

```
  Típus KépSor(Konstans db:Egész) = Szöveg
        [Típusinvariáns: ks:KépSor:
             Hossz(ks)≤db és ∀i∈[1..Hossz(ks)]:
             Jele(ks,i)∉Végjelek]

Eljárás ElsőKépsor(Változó s: Sor, ks: KépSor(db)):
```
[Levágja a tetszőleges hosszúságú *s* sor első, maximum *db* hosszúságú részét úgy, hogy a szavai teljes egészükben beleférjenek *ks* KépSorba.]
```
  ks:=''
  Ciklus
    ElsőSzó(s,szó,elválasztás); vége:=Igaz
    Elágazás
      Hossz(ks+szó+elválasztás)≤db esetén
          ks:+szó+elválasztás
          vége:=Hamis
      Hossz(ks+szó)≤db és
                Jele(elválasztás,1)=' ' esetén
          ks:+szó
          s:=elválasztás+s
      Hossz(ks+szó)<db esetén
          befér:=db-Hossz(ks+szó)
          ks:+Eleje(elválasztás,befér)
          s:=Vége(elválasztás,befér+1)+s
      egyéb esetben
          s:=szó+elválasztás+s
    Elágazás vége
  amíg Hossz(s)>0 és nem vége
  Ciklus vége
Eljárás vége.

Eljárás BalrólLevág(Változó ks: KépSor):
```
[A szöveget, ami egy KépSor, átalakítja úgy, hogy balról a szóközöket elhagyja.]
```
  Ciklus amíg Hossz(ks)≥1 és Jele(ks,1)=' '
    ks:=Vége(ks,2)
  Ciklus vége
Eljárás vége.

Eljárás JobbrólLevág(Változó ks: KépSor):
```
[A szöveget, ami egy KépSor, átalakítja úgy, hogy jobbról a szóközöket elhagyja]

```
  Ciklus amíg Hossz(ks)≥1 és Jele(ks,Hossz(ks))=' '
    ks:=Eleje(ks,Hossz(ks)-1)
  Ciklus vége
Eljárás vége.
```

A margóhoz igazító eljárások nem törődnek a szöveg szélein levő szóközökkel, az igazításnál ezeket is normál karakterekként veszik figyelembe. Ha ezek megtartására nincs szükség, akkor az igazítás előtt alkalmazni kell a BalrólLevág és a JobbrólLevág műveleteket. A továbbiakban az *ks* KépSort jelent (azaz legfeljebb *szél*nyi szélességűt sort, amelyre igaz, hogy SorVégjel nélküli).

```
Eljárás JobbraIgazít(Változó ks: KépSor(szél)):
    [A szöveget, ami egy KépSor, átalakítja úgy, hogy balról annyi szóközzel
     egészíti ki, hogy a sor utolsó karaktere a szél szélességű tartomány jobbol-
     dalán legyen.[10]]
  Ciklus amíg Hossz(ks)<szél
    ks:=' '+ks
  Ciklus vége
Eljárás vége.

Eljárás KözépreIgazít(Változó ks: KépSor(szél)):
    [A szöveget, ami egy KépSor, átalakítja úgy, hogy balról és jobbról annyi szó-
     közzel egészíti ki, hogy az a szél szélességű tartományban középre essék.]
  Ciklus amíg Hossz(ks)<szél-1
    ks:=' '+ks+' '
  Ciklus vége
  Ha Hossz(ks)<szél akkor ks:=ks+' '
Eljárás vége.

Eljárás MargóhozIgazít(Változó ks: KépSor(szél)):
    [A szöveget, ami egy KépSor, a bal és a jobb margóhoz igazítja: a szavak
     között többlet („puha") szóközöket helyez el egyenletesen.]
  kell:=szél-Hossz(ks); hely:=Szószámlálás(ks)-1
  kellplusz:=kell MOD hely; kell:=kell DIV hely
  ElsőSzó(ks,t,elv); t:+elv
  Ciklus i=1-től hely-ig      [„szavanként"]
    Ha i≤kellplusz akkor      t:+Szóközök(kell+1)[11]
                  különben    t:+Szóközök(kell)
    ElsőSzó(ks,szó,elv); t:+szó+elv
  Ciklus vége
  ks:=t
Eljárás vége.
```

[10] Lehet arra is gondolni, hogy ezen *nem eredeti* szóközöket megkülönböztethető kóddal kell ellátni, az utólagos fölismerhetőség érdekében. Hívhatjuk ezeket *puha szóköz*öknek.

[11] Szóközök(X) az X darab ún. „puha" szóközből álló szövegkonstanst jelöli.

3. „Lineáris" szövegfile-típusok[12]

A szövegfile-típusokhoz is definiálhatjuk a szövegtípus strukturáló műveleteit, némi átalakítással. A file-t mint *dokumentum*ot tároló adatszerkezetet tekintjük (s mint ilyen: nem lehet üres; ez az aktuális *típusinvariáns*). Vizsgáljuk meg először a tágabb értelemben vett input szövegfile-t!

```
Típus Elválasztók=(' ',Tab,',','.','?','!',':',';',
                   SorVégjel, LapVégjel, SzövegVégjel)
Konstans SzóElválasztójelek: Halmaz(Elválasztók)
                                    (' '..SzövegVégjel)
Változó c: Karakter [az előreolvasott jel]
Eljárás Nyitás[Olvasásra](Változó f: InpSzövegfile,
                          Konstans fnév: Szöveg):
  Megnyit(f,név); Olvas(f,c)[13] [el reolvasás]
Eljárás vége.
Eljárás Zárás[Olvasásra](Változó f: InpSzövegfile):
  Lezár(f)
Eljárás vége.
Függvény Vége?(Változó f: InpSzövegfile): Logikai
  Vége?:=c=SzövegVégjel
Függvény vége.
Eljárás SzóOlvasás[14](Változó f: InpSzövegfile,
                                 szó,elválasztás: Szöveg):
    [Az elválasztójelek átlépésével olvassa a file első szavát;
     Ef: nem Vége?(f)]
  szó:=''; elválasztás:=''
  Ha c∉SzóElválasztójelek akkor
    Ciklus
      szó:+c;  Olvas(f,c)
    amíg [elválasztójel garantáltan van] c∉SzóElválasztójelek
    Ciklus vége
  Elágazás vége
  Ciklus
    elválasztás:+c;  Olvas(f,c)
  amíg nem Vége?(f) és c∈SzóElválasztójelek
  Ciklus vége
Eljárás vége.
```

[12] A „lineáris" jelző utalás arra, hogy vannak nem lineáris szövegek is: hipertextek.

[13] Előreolvasást kell alkalmaznunk, mivel olvasás nélkül el sem érhető az első jele. Így a szűkebb értelmezésű szövegfile műveleteket „ki kell terjesztenünk" erre a szövegfile-ra. A Megnyit garantáltan elvégezhető a *típusinvariáns* miatt.

[14] Rokona az előbbi *ElsőSzó* eljárásnak.

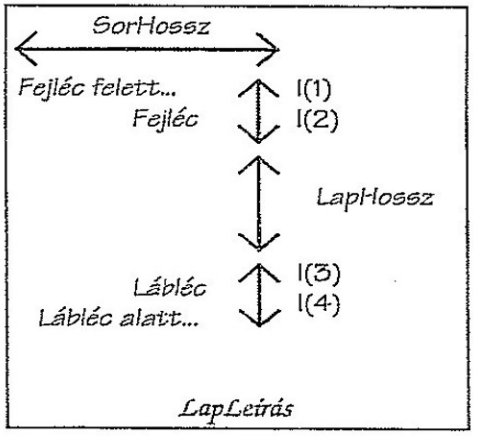
```
Eljárás SorOlvasás(Változó f: InpSzövegfile,
                           s: Sor):
    [SorVégjelig olvassa a sor karaktereit;
     Ef: nem Vége?(f)]
  s:=''
  Ciklus amíg [SorVégjel garantáltan van] c≠SorVégjel
    s:+c;  Olvas(f,c)
  Ciklus vége
  Ha nem Vége?(f) akkor Olvas(f,c)
Eljárás vége.

Eljárás LapOlvasás(Változó f: InpSzövegfile,
                           l: Lap):
    [LapVégjelig olvassa a lap karaktereit;
     Ef: nem Vége?(f)]
  l:=''
  Ciklus amíg [LapVégjel garantáltan van] c≠LapVégjel
    l:+c;  Olvas(f,c)
  Ciklus vége
  Ha nem Vége?(f) akkor Olvas(f,c)  [előreolvasás]
Eljárás vége.
```

Az input szövegfile párja, az output szövegfile következik. A tágabb értelmezésű output szövegfile SzóÍrás, SorÍrás, LapÍrás eljárásai *hasonló* szerkezetűek: karakterenként ki kell írniuk a paraméterként kapott szövegeket, külön Nyitás, illetve Zárás eljárásra pedig nincs is szükség.

A tágabb értelmezésű output szövegfile-nak van egy másik változata is, a képernyőn vagy nyomtatón megjelenő forma[15]. Ezen esetben ugyanis a nyomtatott szöveg formátumhoz megszokott hétköznapi sor-, lap-, s esetleg hasábfogalmakat kell megvalósítani. (Az eljárások a paraméterek mellett globális adatokat is fognak használni.)

```
Konstans[16]
  SorHossz    =80
  LapHossz    =64
  FejlécHossz= 4
  LáblécHossz= 4
Típus
  LapLeírás=Tömb(1..4: Egész)[17]
  Sor       =Tömb(1..SorHossz:
                          Karakter)
```



[15] Nevezzük ezt *KépSzövegfile*-nak, amelynek precízebb megadása tartalmazza a „formátum" leírását, s magát a megjelenítendő output szövegfile-t.

[16] Tekinthetnénk ezeket akár az OutSzövegfile generic-paramétereiként is.

[17] A lapleírás tartalmazza a lapon a fejlécsor előtti, a fejléc és a szöveg közötti, a szöveg és a lábléc közötti, valamint a lábléc utáni üres sorok számát.

```
Változó
  s        : Sor
  sorszám ,
  lapszám : Egész
  fejléc   : Tömb(1..FejlécHossz: Sor)
  lábléc   : Tömb(1..LáblécHossz: Sor)
  l        : LapLeírás
Eljárás Nyitás[Írásra](Változó f: KépSzövegfile,
                       Konstans fnév: Szöveg):
  Megnyit(f,fnév); s:=''; sorszám:=0; lapszám:=0
Eljárás vége.
Eljárás SzóÍrás(Változó f: KépSzövegfile,
                Konstans szó,elválasztó: Szöveg):
  Ha Hossz(s+szó+elválasztó)>SorHossz akkor
       SorÍrás(f,s);  s:=szó+elválasztó
    különben
       s:+szó+elválasztó
  Elágazás vége
Eljárás vége.
```

A SzóÍrás eljárás, mivel a kiírandó paramétere egy szöveg, egyben az elválasztójelek írására is felhasználható. A SorÍrás eljárásban felhasználjuk a szűkebb értelemben vett, „normál szövegfile-ba" írás műveletét (SorÍr).

```
Eljárás SorÍrás(Változó f: KépSzövegfile,
                Konstans s: Sor):
  Ha sorszám=0
    akkor lapszám:+1
          ÜresSorok(l(1))
          Ciklus i=1-től FejlécHossz-ig
            SorÍr(f,fejléc(i))
          Ciklus vége
          ÜresSorok(l(2))
  Elágazás vége
  SorÍr(f,s);  sorszám:+1
  Ha sorszám=LapHossz
    akkor ÜresSorok(l(3))
          Ciklus i=1-től LáblécHossz-ig
            SorÍr(f,lábléc(i))
          Ciklus vége
          ÜresSorok(l(4))
          sorszám:=0
  Elágazás vége
Eljárás vége.
Eljárás Lapdobás(Változó f: KépSzövegfile):
  sorszám:=0
Eljárás vége.
```

```
Eljárás Zárás(Változó f: KépSzövegfile):
  SorÍrás(f,s)
  Ha LapHossz>sorszám akkor
    ÜresSorok(LapHossz-sorszám)
    ÜresSorok(l(3))
    Ciklus i=1-től LáblécHossz-ig
      SorÍr(f,lábléc(i))
    Ciklus vége
    ÜresSorok(l(4))
  Elágazás vége
  Zár(f)
Eljárás vége.
```

Az általánosításnál abból a kérdésből indulunk ki, hogy mik azok a transzformációk, amelyek egy *teljes* szövegfile-ra vonatkozhatnak. Ilyen a *hozzáfűz*, *tartalomjegyzék-*, *indexkészítés*, illetve a *másolás*nak *formázott* variációja. Az utóbbiak igényelnek csak némi magyarázatot.

A tartalomjegyzék-készítés azon alapszik, hogy valamilyen módon fölismerhető a szöveg *struktúrá*ja, s ennek, címekre korlátozódó kilistázását kell elvégezni.

Az indexkészítés egyszerűen egy különálló szövegben felsorolt *kulcs-szav*ak előfordulási helyeinek kigyűjtését jelenti.

A formázott másolás pedig a file egy, ún. *stílus*leírásnak megfelelő átmásolását.

Adalékok a fentiek megvalósításához:

1. A kiinduló file maga a nyers, formázatlan szöveg.
2. Ebben lehetnek fejezetek, alfejezetek, ..., amelyek címzésére, fejlécére stb.-re vonatkozhatnak formai elvárások (betűtípus, igazítás).
3. Ezeket az elvárásokat kell megadni a stílusleírásban, ami lehet akár egy szövegben tárolva. Ebben rögzített szintaxisú *stílusutasítások*ban sorolhatók föl az egyes globális, formai jellemzők.

Példa: *stílusleírás-(szöveg)file*

```
BalMargó:4<sorvégjel>
JobbMargó:76<sorvégjel>
LapHossz:78<sorvégjel>
Fejléc:'Ez a fejléc szövege.'<sorvégjel>
Lábléc:'Ez meg egy lábléc. Lapsorszám:'#[18]<sorvégjel>
FejezetKezdet:'%%'[19]
AlFejezetKezdet:'%'[20]
FejezetcímIgazítás:Középre
...
```

(A **vastagon** szedettek a kulcsszavak. A többiek a paraméterek.)

Egy fenti stílus szerint „fölépített" szövegben a fejezetkezdeteket ilyesfajta módon képzelhetjük el:

```
...
%%0. Bevezetés
...
%%1. A szöveg típusok szűkebb értelmezése
...
%%%1.1. A karaktertípus reprezentálása, implementálása
...
%%2.A szövegtípusok általánosabb ...
...
```

Ennek értelme többek között: a szövegfeldolgozó program mindig tudja, hogy a szöveg milyen „hierarchiai szintjénél" tart.

4. A tartalomjegyzék a fenti többletinformációkkal már elkészíthető. Legfeljebb a tartalomjegyzék *formá*jának, hollétének lerögzítése nyitott. Ezt is egy, a fentihez hasonló *stílusleírás*ban lehet lefektetni.

Példa:

```
Hol:Elején
FejezetKezdet: '%%' [Azonosítás.]
AlFejezetKezdet: '%' [Azonosítás.]
FejezetCímForma: FejezetSzám FejezetCím: Bold #: Jobbra
AlFejezetCímForma: Beljebb: 2
  [A többi megegyezik a fejezetével.]
```

5. Az indexkészítéshez szükséges *kulcsszó*felsorolást is egy sajátos szintaxisú szöveg tartalmazhatja.

[18] A lapsorszám jele.
[19] A szövegben ilyen jel jelzi a fejezet kezdetét.
[20] Az alfejezetek kezdetén a fejezetkezdet jeléhez még ilyen jel társul. Mégpedig annyi, „ahányadikszeres" alfejezet. A hierarchia így tükröződik.

Példa: *Indexleírás-(szöveg)file*

```
Formátum:KulcsSzóFejezetSzám:Jobbra
'számítástechnika'<sorvégjel>
'struktúramegfeleltetéskonfliktusa'<sorvégjel>
'struktúramegfeleltetés'?'konfliktus'<sorvégjel>
...
```

(A kulcsszavak tartalmazhatnak szóközt, sőt bármilyen –nem vég– jelet, ha aposztrofok közé zárjuk. Ilyen szövegkonstansok között pedig néhány jelentéssel bíró speciális jel lehet; mint pl. a '?', melynek jelentése 'egy tetszőleges nem elválasztójel'.)

```
Eljárás
  Hozzáfűz(Változó  ehhez:OutSzövegFile,
           Konstans ezt:InpSzövegFile)
    [   ...   ]
  Tartalomjegyzék(Konstans ezt:InpSzövegFile,
                           stílus:Szöveg,
                  Változó  ebbe:OutSzövegFile)
    [   ...   ]
  IndexKészítés(Konstans ezt:InpSzövegFile,
                         kulcs:Szöveg,
                  Változó  ebbe:OutSzövegFile)
    [   ...   ]
  FormázvaMásol(Konstans ezt:InpSzövegFile,
                         stílus:Szöveg,
                  Változó  ebbe:OutSzövegFile)
    [   ...   ]
```

III. Szövegtípusok speciális feladatai

Ebben a fejezetben találunk majd olyan algoritmusokat is, amelyek valójában nem csak szövegekre alkalmazhatók. Ezek olyan gyakori tevékenységeket írnak le, amelyek minden olyan adattömeg esetén alkalmazhatók, *amik byte-onként is szemlélhetők* [1]. (Az persze csak nézőpont kérdése, hogy az egyes byte-ok tartalmát úgy látjuk, mint jelek kódjait, vagy ilyen jelentést nem is tulajdonítva nekik, önmagukban.) Megnézzük, hogy miként lehet egy karakteres (byte-os) adattömegből a felesleges *információt elhagyni*, az adattömeget *sűrűbben, tömörebben* tárolni, *formázni, transzformálni*, illetve hogyan lehet egy *részsorozatát* a keresési tételből származtathatónál *ügyesebben megtalálni*. ...

E fejezetben általában a következő deklaráció minden feladatnál érvényes:

```
Változó X:InpSzövegfile [nem üres!]
        Z:OutSzövegfile
        c:Karakter
```

1. Szűrés

A szűrés szövegeknél leggyakrabban előforduló feladata a *szóközök kihagyása* a szövegből. Előfordul szinte minden olyan beolvasási feladatnál, ahol a felhasználó viszonylag szabad formátumban válaszolhat a program kérdéseire, de a szóközök nem játszanak szerepet a válaszban. A szűrés egyben klasszikus *adatfeldolgozási* feladat, így nem meglepő a struktúra megfeleltetésen alapuló megoldása. A megoldást szövegfile-okra adjuk meg.

Típus

Bemenet = **File**(Karakter)	Kimenet = **File**(Karakter)
Karakter = szóköz ∪ egyéb karakter	Karakter = üres ∪ egyéb karakter

```
Eljárás Szűrés:
  Nyit(X,Z)
  Ciklus amíg nem Vége?(X)
    Olvas(X,c)
    Ha c≠' ' akkor Ír(Z,c)
  Ciklus vége
  Zár(X,Z)
Eljárás vége.
```

[1] Érdemes azon is elmélázni, hogy az itt vázolt algoritmusokat nem lehet-e általánosítani olyan adatszerkezetekre is, amelyeket később *absztrakt sorozat* típusú objektumnak fogunk definiálni.

Bonyolultabb a helyzet abban az esetben, amikor a szóközöknek elválasztó funkciójuk van, ekkor ugyanis nem lehet minden szóközt elhagyni, hanem az egyes szóközcsoportokból egy szóközt meg kell hagyni. Érdemes megvizsgálni e feladat többféle megoldását, mert tanulságos példája a típusmegfeleltetés és a programbonyolultság összefüggésének. Az egyes megoldásokban a típusmegfeleltetést más-más ötlet alapján végezzük el.

1. megoldás: az eredményfile-ba vagy egy szóközt kell írni, vagy egy nem szóköz karaktert.

Típus

Bemenet = **File**(Elem)	Kimenet = **File**(Elem)
Elem = szóközcsoport ∪ egyéb kar.	Elem = szóköz ∪ egyéb kar.

A szóközcsoport végének felismeréséhez előreolvasásra[2] van szükség, valamint emiatt meg kell oldani a file-típusétól eltérő filevége kezelést (akkor van a file-nak vége, ha már nincs lehetőség előreolvasásra).

```
Változó
  k         :Karakter        [az előreolvasott karakter]
  filevége:Logikai           [az új Vége? fv. megvalósításához]

Eljárás Szűrés:
  Nyitás(X);  Nyit(Z)
  Ciklus amíg nem filevége
    Elemolvasás(X,c);  Ír(Z,c)[3]
  Ciklus vége
  Zárás(X); Zár(Z)
Eljárás vége.

Eljárás Nyitás(Változó X:InpSzövegfile):
  Nyit(X);  Olvas(X,k);  filevége:=Vége?(X)
Eljárás vége.

Eljárás Elemolvasás(Változó X:InpSzövegfile,
                                c:Karakter):
  c:=k
  Ha Vége?(X) akkor  filevége:=Igaz
                     k:=Karkter(0)
```

[2] A fejezet további részében feltételezzük, hogy nem üres file-t kell feldolgoznunk.
[3] A kimeneti elem mindkét esetben egy karakter, tehát azonnal kiírható.
[4] Kétféle értelmezést definiálhatunk:

```
        különben [előreolvasás:]
                 Ha c=' ' akkor
                    Ciklus
                      Olvas(X,k)
                    amíg nem Vége?(f) és k=' '
                    Ciklus vége
                    filevége:=k=' '
                    Ha filevége akkor k:=Karkter(0)
                 különben
                    Olvas(X,k)
                 Elágazás vége
  Elágazás vége
Eljárás vége.

Eljárás Zárás(Változó X:InpSzövegfile):
  Zár(X)
  Ha k≠Karakter(0) akkor Ír(Z,k)
Eljárás vége.
```

2. megoldás: a bemenő file-ban szóközökből, illetve nem szóköz karakterekből álló csoportok vannak.

Típus

Bemenet = File(Csoport)	Kimenet = File(Csoport)
Csoport = szóközcsoport ∪ egyéb csoport	Csoport = szóköz ∪ egyéb csoport

A szóközcsoport és az egyéb csoport végének felismeréséhez is előreolvasásra van szükség, továbbá tudni kell, hogy milyen jelek csoportját dolgozzuk éppen fel, s a kiírás sem egyszerű. Ebben a megoldásban akkor vagyunk file végén, ha a csoportolvasás nem egy másik csoport első karakterénél ért véget, hanem a file végénél. Feltesszük, hogy a file nem üres.

```
Változó
  k         :Karakter        [az előreolvasott karakter]
  szóköz,
  filevége:Logikai
  csoport :Szöveg

Eljárás Szűrés:
  Nyitás(X); Nyit(Z)
  Ciklus amíg nem filevége
    Csoportolvasás(X,csoport); Csoportírás(Z,csoport)
  Ciklus vége
  Zárás(X); Zár(Z)
Eljárás vége.

Eljárás Nyitás(Változó X:InpSzövegfile):
  Nyit(X);  Olvas(X,k)
  szóköz:=(k=' '); filevége:=Vége?(X)
Eljárás vége.
```

```
Eljárás Csoportolvasás(Változó X:InpSzövegfile,
                                csoport:Szöveg):
  csoport:=k
  Ha nem Vége?(X)
    akkor   Olvas(X,k)
            Ciklus amíg szóköz=(k=' ') és nem Vége?(X)
              csoport:+k; Olvas(X,k)
            Ciklus vége
            Ha szóköz=(k=' ') akkor filevége:=Igaz
                                    k:=Karakter(0)
            szóköz:=nem szóköz
   különben filevége:=Igaz; k:=Karakter(0)
  Elágazás vége
Eljárás vége.

Eljárás Csoportírás(Változó Z:OutSzövegfile,
                    Konstans csoport:Szöveg):
  Ha Jele(csoport,1)=' '  akkor     Ír(Z,' ')
                          különben   Ír(Z,csoport)
Eljárás vége.

Eljárás Zárás(Változó X:InpSzövegfile):
  Zár(X)
  Ha k≠Karakter(0) akkor Ír(Z,k)
Eljárás vége.
```

3. megoldás: az eredményfile-ba kikerülő nem szóköz karakterek elé vagy kell írni szóköz karaktert, vagy nem.

Típus

Bemenet = **File**(Csoport)	Kimenet = **File**(Csoport)
Csoport = Szóközcsoport ∪ egyéb karakter	Csoport = szóköz + egyéb karakter
Szóközcsoport = szóközök + egyéb karakter	∪ egyéb karakter

Ebben a megoldásban minden csoportot egy nem szóköz karakter zár le, tehát nem kell előreolvasást alkalmazni. A csoportok vagy egyetlen elemből állnak (nem szóköz karakter), vagy a karaktert megelőző szóközök létét kell még tárolni.

```
Változó vanSzóköz:Logikai

Eljárás Szűrés:
  Nyit(X,Z)
  Ciklus amíg nem Vége?(X)
    Csoportolvasás(X,k,vanSzóköz)
    Csoportírás(Z,k,vanSzóköz)
  Ciklus vége
  Zár(X,Z)
Eljárás vége.
```

A **Csoportolvasás** eljárásban most kihasználhatjuk, hogy a szövegfile-ok –definíció szerint– mindig olyanok, hogy a végükön egy sorvége karakter (tehát nem szóköz) van.

```
Eljárás Csoportolvasás(Változó X:InpSzövegfile,
                                k:Karakter,
                                vanSzóköz:Logikai):
  Olvas(X,k);  vanSzóköz:=(k=' ')
  Ciklus amíg k=' '
    Olvas(X,k)
  Ciklus vége
Eljárás vége.

Eljárás Csoportírás(Változó Z:OutSzövegfile,
                    Konstans k:Karakter,
                             vanSzóköz:Logikai):
  Ha vanSzóköz akkor Ír(Z,' ')
  Ír(Z,k)
Eljárás vége.
```

Megállapítható e három megoldás alapján, hogy az első ránézésre legkevésbé természetes definícióhoz, a harmadik változathoz tartozik a legegyszerűbb megoldás.

2. Tömörítés

Elsőként olyan sűrítési lehetőségeket tárgyalunk, amelyek nemigen vonatkozhatnak nem karakterekből álló byte-sorozatokra. Ugyanis arra építjük őket, hogy az *ember* bizonyos speciális karaktereket sajátos módon értelmez, használ.

2.1. TAB karakterek alkalmazása

Hosszú szóközintervallumok TAB-bal helyettesítése[4] egy gyakori megoldás, amely még az írógép korszakból öröklődött. Lényege, hogy a sorban egy adott pozíció eléréséhez szükséges szóközöket helyettesít egyetlen karakter, a TAB. Egyik megoldandó feladat az ilyen szóközcsoportok helyettesítése TAB karakterrel, a másik pedig –képernyőre vagy nyomtatóra íráskor– a TAB karakterek helyettesítése megfelelő számú szóközzel.

Kezdjük az első feladattal, a TAB-ok elhelyezésével! A megoldást itt is –mint az eddigiekben– a jó struktúra megfeleltetés alapján kapjuk:

Típus

Bemenet = **File**(Csoport)	Kimenet = **File**(Csoport)
Csoport = nem szóköz ∪ szóközök TAB-pozícióig ∪ szóközök betűig	Csoport = nem szóköz ∪ TAB ∪ szóközök betűig

A struktúra megfeleltetés ezek alapján egyszerűen elvégezhető. Ezeket tekintve egységnek, s használva az előreolvasás technikáját, a következő megoldás adódik:

```
Típus
  FajtaTíp = (NemSzóköz,Tabjel,Szóközök)
Változó
  k,kar    : Karakter[az előreolvasott karakter]
  filevége: Logikai
  oszlop   : Egész          [az előreolvasott karakter helye]
  db       : Egész
  fajta    : FajtaTíp
Eljárás Tömörítés:
  Nyitás(X); Nyit(Z)
  Ciklus amíg nem filevége
    Csoportolvasás(X,fajta,kar,db)
    Csoportírás(Z,fajta,kar,db)
  Ciklus vége
  Zárás(X); Zár(Z)
Eljárás vége.

Eljárás Nyitás(Változó X:InpSzövegfile):
  Nyit(X); Olvas(X,k); oszlop:=1; filevége:=Vége?(X)
Eljárás vége.

Eljárás Zárás(Változó X:InpSzövegfile):
  Zár(X)
  Ha k≠Karakter(0) [volt előreolvasás] akkor Ír(Z,k)
Eljárás vége.
```

A **Csoportolvasás** eljárásnak el kell döntenie, hogy melyik esetről is van szó a lehetséges háromból. Ennek „kódját" el kell helyeznie a *fajta* változóban. Az első esetben a *kar* változóba kell tenni a karaktert, a másodikban nem kell semmi egyebet tenni, a harmadikban pedig a *db* változóba kell tenni a szóközök darabszámát. Az első és a többi eset szétválasztása megtörténhet az előre beolvasott karakter alapján. Az eljárásnak természetesen gondoskodni kell az újabb előreolvasásról. A harmadik eset alapján válik világossá, hogy miért van szükség az előreolvasásra: azt, hogy egy szóközsorozatot betű zár le, csak úgy tudjuk eldönteni, ha már beolvassuk a betűt is. A beolvasás feladata az is, hogy az aktuális oszlop sorszámát mindig növelje, illetve sor végén az 1 értéket adja.

```
Eljárás Csoportolvasás(Változó X:InpSzövegfile,
                                fajta:FajtaTíp,
                                kar:Karakter,
                                db:Egész):
  kar:=k
  Elágazás
   Vége?(X) esetén filevége:=Igaz
                  k:=Karakter(0) [nincs előreolvasás]
    k≠' '   esetén Betűolvasás(X,fajta,kar); [db:=1]
    egyéb  esetben Szóközcsoportolvasás(X,fajta,kar,db)
  Elágazás vége
Eljárás vége.
```

A **Betűolvasás** eljárás az előreolvasáson és a fajta állításán kívül az aktuális karakter alapján megadja a következő karakter oszlopsorszámát.

```
Eljárás Betűolvasás(Változó X:InpSzövegfile,
                           fajta:FajtaTíp,
                    Konstans kar:Karakter):
  fajta:=NemSzóköz; Olvas(X,k)
  Ha kar=SorVégjel akkor  oszlop:=1
                  különben oszlop:+1
Eljárás vége.
```

A **Szóközcsoportolvasás** eljárás biztosan nem lép át SorVégjelet, sőt nem juthat el a file végéig sem (hiszen minden sor végét, még az utolsóét is SorVégjel zár le), csupán azt kell figyelnie, hogy a második, vagy a harmadik esetről van-e szó.

```
Eljárás Szóközcsoportolvasás(Változó   X:InpSzövegfile,
                                       fajta:FajtaTip,
                                Konstans kar:Karakter,
                                Változó  db:Egész):
  db:=0
  Ciklus amíg k=' ' és nem TABpozíció(oszlop)
    db:+1;  Olvas(X,k);  oszlop:+1 [előreolvasás]
  Ciklus vége
  Ha k=' ' akkor fajta:=Tabjel különben fajta:=Szóközök
Eljárás vége.
```

A **Csoportírás** már egyszerű, csupán egy fajta szerinti elágazást tartalmaz.

```
Eljárás Csoportírás(Változó  Z:OutSzövegfile,
                    Konstans fajta:FajtaTíp,
                             kar:Karakter,
                             db:Egész):
```

```
  Elágazás
    fajta=NemSzóköz esetén   Ír(Z,kar)
    fajta=Tabjel    esetén   Ír(Z,TAB)
    fajta=Szóközök  esetén   Ciklus I=1-től db-ig
                               Ír(Z,' ')
                             Ciklus vége
  Elágazás vége
Eljárás vége.
```

Nézzük most a másik feladatot, a TAB karaktereket helyettesítsük a megfelelő számú szóközzel!

A megoldásban figyelni kell, hogy a kiírásban melyik oszlopnál tartunk. Ha TAB karakter következik, akkor a következő tabulációs pozícióig szóközöket kell írni, különben pedig az aktuális karaktert. A megoldást a **másolás programozási tétel**re vezetjük vissza, csak arra kell ügyelnünk, hogy egyes karakterek helyett adott darabszámú szóközt kell írni.

Típus

Bemenet = **File**(Jel)	Kimenet = **File**(Csoport)
Jel = TAB ∪ egyéb	Csoport = szóközök TAB-ig ∪ egyéb

```
Eljárás TABtalanítás:
  Nyit(X); Nyitás(Z)
  Ciklus amíg nem Vége?(X)
    Olvas(X,kar); Csoportírás(Z,kar)
  Ciklus vége
  Zár(X,Z)
Eljárás vége.
```

A **Nyitás** az eredményfile-ba kikerülő következő karakter helyét 1-re állítja.

```
Eljárás Nyitás(Változó Z:OutSzövegfile):
  Nyit(Z); oszlop:=1
Eljárás vége.

Eljárás Csoportírás(Változó   Z:OutSzövegfile,
                    Konstans kar:Karakter):
  Ha kar=TAB akkor  SzóközökÍrása(Z)
            különben Betűírás(Z,kar)
Eljárás vége.
```

Szóközöket addig kell írni, amíg egy TAB-pozícióig el nem érünk.

```
Eljárás SzóközökÍrása(Változó Z:OutSzövegfile):
  Ciklus
    Ír(Z,' '); oszlop:+1
  amíg nem TABpozíció(oszlop)
  Ciklus vége
Eljárás vége.
```

A **Betűírásnak** a sorvég karakter írása után az **oszlop** változót 1-re kell állítania!

```
Eljárás Betűírás(Változó Z:OutSzövegfile,
                 Konstans kar:Karakter):
  Ír(Z,kar)
  Ha kar=SorVégjel akkor oszlop:=1 különben oszlop:+1
Eljárás vége.
```

2.2. TOKEN-ek alkalmazása

Tetszőleges jelismétlődések TOKEN-es helyettesítésének lényege, hogy ha valamilyen karakterből sok van egymás mellett, akkor azt kódoljuk a karakterrel és annak darabszámával! A számkódolás felismeréséhez be kell vezetni még egy speciális karaktert (∉X!), s így a hosszú karaktersorozatokat egy hármassal ábrázolhatjuk.

Példa:

...xaaaaaaaaay... → ...x⊕∂aẏ...

⊕: speciális, más célra nem használható jel
∂: darabszámnyi kódú karakter (most éppen a Karakter(9))

Típus

Bemenet = **File**(Csoport)	Kimenet = **File**(Csoport)
Csoport = **Szöveg** [Típusinvariáns: s:Szöveg azonos betűk sorozata és Hossz(s)<256 és ⊕∉s]	Csoport = **Szöveg** ∪ Kódolt jel Kódolt jel = ⊕ + ∂ + **Karakter**

A megoldásban előreolvasást kell alkalmazni, mivel az azonos karakterekből álló sorozat végét egy tőlük eltérő karakter jelzi.

A speciális tulajdonságú szöveg típusának megvalósításához kihasználjuk a specialitást leíró típusinvariánst: a sorozat *azonos* elemekből áll. Ezért a sorozat mindig megadható egy karakterrel és annak darabszámával.

```
Változó
  k,kar    : Karakter
  db       : Egész
  filevége: Logikai
Eljárás Tömörítés:
  Nyitás(X); Nyit(Z)
  Ciklus amíg nem filevége
    Csoportolvasás(X,db,kar)
    Csoportírás(Z,db,kar)
  Ciklus vége
  Zár(X,Z)
Eljárás vége.
```

```
Eljárás Nyitás(Változó X:InpSzövegfile):
  Nyit(X); Olvas(X,k); filevége:=Hamis
Eljárás vége.

Eljárás Csoportolvasás(Változó X:InpSzövegfile,
                                db:Egész,
                                kar:Karakter):
  kar:=k; db:=1
  Ha Vége?(X)
      akkor filevége:=Igaz
      különben
            Olvas(X,k)
            Ciklus amíg k=kar és nem Vége?(X) és db<255
              db:+1; Olvas(X,k)
            Ciklus vége
  Elágazás vége
Eljárás vége.

Eljárás Csoportírás(Változó Z:InpSzövegfile,
                    Konstans db:Egész,
                             kar:Karakter):

  Ha db<4 akkor   Ciklus i=1-től db-ig
                    Ír(Z,kar)
                  Ciklus vége
        különben  Ír(Z,⊕+Karakter(db)+kar)
Eljárás vége.
```

Egyszerűbb az ellentett feladat megoldása, a tömörített szöveg kifejtése.

Típus

Bemenet = **File**(Elem)	Kimenet = **File**(Csoport)
Elem = Karakter ∪ Kódolt karakter	Csoport = **Szöveg** [Típusinvariáns: s:Szöveg azonos betűk sorozata és Hossz(s)<256]
Kódolt karakter = ⊕ + ∂ + karakter	

```
Eljárás Kifejtés:
  Nyit(X,Z)
  Ciklus amíg nem Vége?(X)
    Elemolvasás(X,db,kar)
    Csoportírás(Z,db,kar)
  Ciklus vége
  Zár(X,Z)
Eljárás vége.
```

A struktúramegfeleltetés úgy végezhető el könnyedén, ha például az **Elemolvasás** eljárás az elemet azonnal átalakítja „csoporttá". A **Csoport** típus itt is egyszerűen definiálható, hiszen olyan sorozatról van szó, amelynek minden eleme *azonos*.

```
Eljárás Elemolvasás(Változó X: InpSzövegfile,
                                db: Egész, kar: Karakter):
  Olvas(X,kar)
  Ha k=⊕ akkor Olvas(X,kar); db:=Egész(kar); Olvas(X,kar)
       különben db:=1
Eljárás vége.

Eljárás Csoportírás(Változó Z: OutSzövegfile,
                    Konstans db: Egész,
                                 kar: Karakter):
  Ciklus i=1-től db-ig
    Ír(Z,kar)
  Ciklus vége
Eljárás vége.
```

2.3. Huffman-kódolás

A *jelgyakoriságot figyelembevevő változó bithosszúságúra áttérés* abból az elég természetes elvárásból indul ki, hogy egy olyan szöveget, amelyben az egyes jelek igen eltérő gyakorisággal fordulnak elő, valószínűleg tömörebben lehet ábrázolni úgy, hogy a fixhosszúságú karakterkódokról *változó bithosszú*ra térünk át. A gyakoriakat rövid, a ritkákat hosszú kódokkal reprezentálva. Lássunk egy példát!

Legyen pl. a szövegben összesen nyolcféle jel: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'! A következő gyakoriságokkal:

'a' → 40,	'c' → 10,	'e' → 8,	'g' → 5,
'b' → 20,	'd' → 8,	'f' → 5,	'h' → 4.

Ez összesen 100 jel. A 8 jel nyilván ábrázolható 3 biten. (Ekkor már tömörebben ábrázoltuk, mint alaphelyzetben tettük volna, nyolcbites ASCII kódjukkal!) Vagyis a teljes szöveg tárolásához kell 100x3=300 bit.

Ha e helyett a rögzített hárombites kódok helyett változó hosszúságban 1, 2, 3 és akár több biten ábrázoljuk a jeleket, de ügyesen figyelembe véve a pillanatnyi gyakoriságokat, akkor talán még kevesebb hely is elegendő. Pl. így

'a' → 1_2,	'c' → 011_2,	'e' → 0101_2,	'g' → 00001_2,
'b' → 001_2,	'd' → 0001_2,	'f' → 0100_2,	'h' → 00000_2.

Ekkor 40x1+20x3+10x3+8x4+8x4+5x4+5x5+4x5=291 bit szükséges.

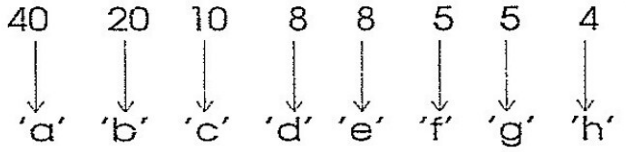
(Azon persze érdemes elmélázni, hogy miért éppen a fenti „faramuci" bitsorozat-hozzárendelést adtuk meg, és hogy tényleg jó-e! A „titok" nyitja: egyértelműen visszaállíthatónak kell lennie a tömörítés után is!) Ezután nézzük az optimális kódolás „kézi" algoritmusát a fenti példán!

A lényege egy *bináris fa* felépítése: a végül elkészült fa egyes pontjaiba az egyes jeleket képzelve és az egyes irányokhoz szisztematikusan 1-et, illetve 0-t

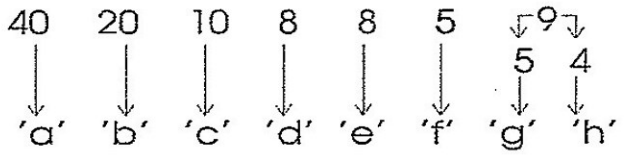
rendelve, csak végig kell haladni az egyes pontokhoz vezető éleken a gyökérből kiindulva, s föl kell jegyezni az élek 0 vagy 1 „címkéjét", s máris megkaptuk a keresett kódhozzárendelést.

A részfák képzése, majd egyesítése útján készül, azon rendező elv alapján, hogy mindig a létrejött részfa súlya a lehető legkisebb legyen a választhatók közül. Kövessük nyomon a stációkat!

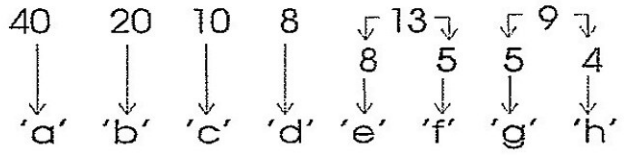
0. (kezdőhelyzet):



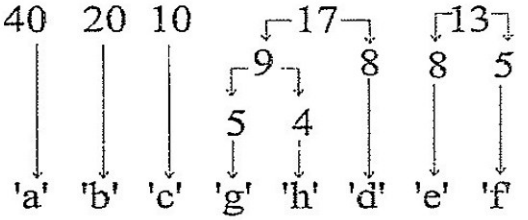
1. lépés után:



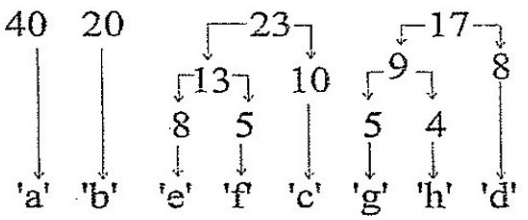
2. lépés után:



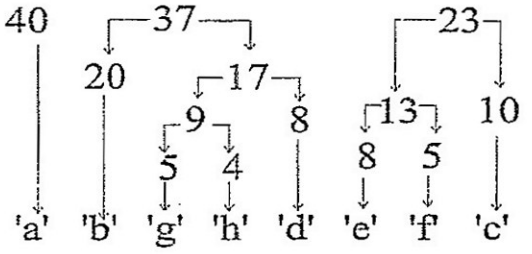
3. lépés után:



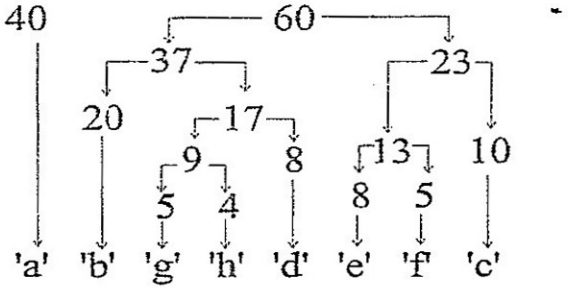
4. lépés után:



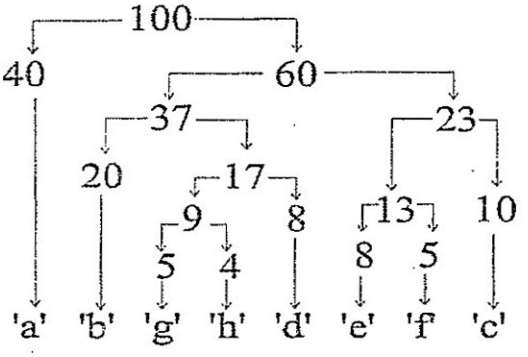
5. lépés után:



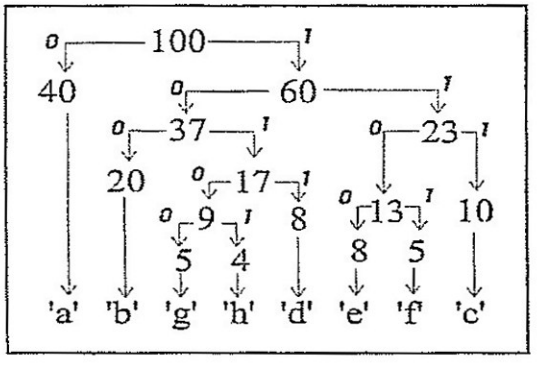
6. lépés után:



7. lépés után a bináris fa felépült:



Nincs más hátra, mint az ágak megcímkézése 0-val (minden balra lépésnél) és 1-gyel (minden jobbra lépésnél), majd leolvasni a kódokat:



S végül a kódok:

'a' → 0_2, 'c' → 111_2, 'e' → 1100_2, 'g' → 10100_2,
'b' → 100_2, 'd' → 1011_2, 'f' → 1101_2, 'h' → 10101_2.

A kijött kódok *egyértelműsége* annak egyszerű következménye, hogy a kódok csak egy, levélhez vezető út címkéiből állnak össze, és így kódok kezdő „rész-

sorozatai" kódként nem fordulhatnak elő. (Ez kellene ahhoz, hogy egy adott szituációban ne lehessen egyértelműen eldönteni, hogy kész kódnál vagy egy kódnak csak első szeleténél tartunk.)

Az így keletkezett tömörített szöveg mellé persze mellékelni kell szabványosított formában a *kódtranszformáció táblázatát* is, ami csökkenti a nyereséget. S még egy fontos utólagos megjegyzés: sajnos a betűstatisztika megszerzése érdekében –a fafelépítést megelőzően– egy többlet végigolvasásra van szükség. Tehát összesen *kétszer* olvassa a szöveget a tömörítő program.

2.4. Jelkombinációk kódolása

E fejezetben az LZW-kódolásról lesz szó.[5] Feltevése a módszernek, hogy a legnagyobb tömörítést azáltal lehet elérni, hogy a jelek helyett *jelkombinációkat* helyettesít kódokkal. Egy előre rögzített méretű kódtranszformációs táblázatot hoz létre, amelyben az „egybetűs kombinációk" kódjait előre fixálja. E táblázatbeli *sorszám* lesz maga a kód. Azaz a *kódok fix bitszámúak* lesznek, éppen annyi, amennyi a táblacímzéshez kell. (Ezért célszerű a táblaméretet kettőhatványúnak választani.) A táblázat egy eleme egy „visszamutató" sorszámból és egy „bővítő" jelből áll:

```
Konstans Méret : PozEgész(4096)
Típus KódTábla = Tömb(1..Méret: Kódoló)
      Kódoló   = Rekord
                 (eleje: 0..Méret,
                  bővítés: Karakter)
```

A tömörítés menete: minden lépésben igyekszik az éppen tömörítendő szöveg aktuális kezdő jelsorozatát a lehető legtömörebb kóddal helyettesíteni a táblázat alapján. Legrosszabb esetben az első jelnek megfelelő kód kerül kivitelre. Majd az utoljára kivitt kódú jelkombinációhoz „ragasztja", bevéve ezt egy új kombinációként. Így bővíti lépésről lépésre a táblázatot, hozva létre egyre hosszabb és hosszabb kombinációk fix hosszúságú kódjait. Ha a táblázat betelt, akkor vagy

- „lezárt" (tovább nem bővíthető) táblázattal folytatja az algoritmust, vagy
- inicializálva a táblázatot –tabula rasa-t csinálva– újrakezdi a táblafelépítést, és folytatja a tömörítést azzal a jellel, ami következett a tábla betelésekor.

Vegyük észre, hogy e módszer a következő jó tulajdonságokkal rendelkezik:

- csak egy menetet igényel,

[5] A. Lempell és J. Ziv által az IEEE Trans. Information Theory folyóirat 1977 májusi számában megjelent módszerük T. Welch által (IEEE Computer, 1984 június) módosított algoritmusának lényegét vázoljuk az alábbiakban.

- nem kell a kódtáblát sem mellékelni (nem is lehet, ha többször inicializálta), mivel a tömörítés is és a kicsomagolás is ugyanazon algoritmus alapján végezhető. (A kicsomagoláskor is ugyanúgy, dinamikusan képzendő a kódtábla.[6])

Egy példa végigkövetésével folytassuk! Az input- és az output-sorozatot a táblázatban megfordítottuk azért, hogy a feldolgozásban soron következő eleme essék a jobb oldalra.

	Input (fordított sorrendben)	*Output* (fordított sorrendben)	*Táblabeli új elem* *tartalma*	*kódolandó szöveg*	*sorszáma*
kezdetben	cbacbaba				
1. lépés	cbacba**b**⇲	(**a**)	(a), **b**	= ab	255+1=256
2. lépés	cbac**ba**⇲	(**b**)(a)	(b), **a**	= ba	255+2=257
3. lépés	cba**c**⇲	(**ba**)(b)(a)	(ab), **c**	= abc	255+3=258
4. lépés	**cba**⇲	(**c**)(ba)(b)(a)	(c), **a**	= ca	255+4=259
5. lépés	⇲	(**cba**)(c)(ba)(b)(a)			

(Az (x) szimbólum az x jelsorozat táblázatbeli címét jelenti.)

Az értékelés: az induló szöveg hossza 8 jelnyi, a tömörítés utáni csak 5. (Bár most csekélynek tűnik a nyereség, az máris érezhető, hogy hosszabb szövegek esetén jóval nagyobb lehetne, hisz még csak 3 hosszúságú „rövidítéseket" tartalmaz a kódolótábla.)

Mielőtt az algoritmus részleteibe mélyednénk, próbáljuk ki a következő igen egyszerű(nek látszó) sorozat kódolását elvégezni: 'abababac'. Ha az eddigieket szó szerint követjük, ehhez az eredményhez jutottunk: '(a)(b)(ab)(aba)(c)'. Ugyanis:

	Input (fordított sorrendben)	*Output* (fordított sorrendben)	*Táblabeli új elem* *tartalma*	*kódolandó szöveg*	*sorszáma*
kezdetben	cabababa				
1. lépés	cababa**b**⇲	(**a**)	(a), **b**	= ab	255+1=256
2. lépés	caba**ba**⇲	(**b**)(a)	(b), **a**	= ba	255+2=257
3. lépés	c**aba**⇲	(**ba**)(b)(a)	(ab), **a**	= aba	255+3=258
4. lépés	**c**⇲	(**aba**)(ba)(b)(a)	(aba), **c**	= abac	255+4=259
5. lépés	⇲	(c)(aba)(ba)(b)(a)			

Próbáljuk ugyanezen gondolatmenettel a dekódolást, azaz kódról kódra haladva, a kódtáblát használva és építve visszanyerni az eredeti jelsorozatot. Most a bemenet a kódtáblabeli megfelelő sorszámok sorozata lesz: 97 98 256 258 99 = '(a)(b)(ab)(aba)(c)'.

[6] Itt is igaz az „alaphozzáállás", hogy
1. kódtábla alapján *dekódolás*, majd
2. a dekódolt jel fölhasználásával *táblabővítés*.

	Input (fordított sorrendben)	*Output* (fordított sorrendben)	*Táblabeli új elem* *tartalma*	*kódolandó szöveg*	*sorszáma*
kezdetben	99 258 256 98 97				
1. lépés	99 258 256 98	↘ a	97, b	= ab	255+1=256
2. lépés	99 258 256	↘ ba	98, a	= ba	255+2=257
3. lépés	99 258	↘ baba	???	= ???	???
4. lépés		↘ ???baba			

A 3. lépésben hiányzik egy kód, nem lehet dekódolni! Az ok: a kódoláskor egyszerre került ez a táblába és kódként a tömörített file-ba. Megoldás ilyen esetben csak egy már ismert rövidebb szeletét lehet csak kódként fölhasználni. A részleteket nézzük meg az alábbiakban.

Az algoritmus alapja az a fölismerés, hogy a legkedvezőbb, már kódtáblában levő jelsorozat kiválasztása egy *verem* alkalmazásával oldható meg a legtermészetesebben. Valahogy így:

```
Eljárás LZW(Konstans X: InpSzövegFile,
            Változó Y: OutFile[7]):

  Konstans  Méret   : PozEgész(4096)
            ÜresJel: Karakter(0)

  Típus KódSorsz= 0..Méret
        Kódoló  = Rekord (eleje: KódSorsz,
                         bővítés: Karakter)
        [az „egyke" jeleknél az eleje-mutató 0, azaz mintha az ÜresJelre
          mutatna, azzal lenne összeláncolva]
        KódTábla    = Tömb(KódSorsz: Kódoló)
        KeresettTip= Rekord(van: Logikai,
                            melyik: KódSorsz)

  Változó minta    : Verem(Karakter)
          jel      : Karakter
          KT       : KódTábla
          leghosszabb,
          KTMax    : KódSorsz
          keresett: KeresettTip
    ... [inicializálás: file-nyitások, KT, KTMax inicializálása]
  VeremÜres(minta)
  Olvas(X,jel);  Verembe(minta,jel)
```

[7] *Méret*től függő bitszélességű elemekből álló file. A fenti 4096 esetén 12 bitet jelent. Vagyis az OutFile = File(KódSorsz)

```
  Ciklus amíg nem Vége?(X)
    leghosszabb:=KódSorszám(jel)[8];  kódolt:=minta
    keresett:=KeresettTip(Igaz,leghosszabb)
      [bővítjük a jelsorozatot, amíg lehet:]
    Ciklus amíg keresett.van és keresett.melyik≠ktMax
                  és nem Vége?(X)
      Olvas(X,jel); Verembe(minta,jel) [bővített kódolandó]
      keresett:=KódKeres(minta,KT)
      Ha keresett.van és keresett.melyik≠ktMax akkor
            leghosszabb:=keresett.melyik
            kódolt:=minta
      Elágazás Vége
    Ciklus vége [nem bővíthető: nincs mivel, vagy az utolsóval nem lehet
                  (hisz ilyen kód nincs, vagy épp a legutóbbi)]
    Ha Vége?(X) akkor          [file-vég]
      Ha minta=Dekódol(leghosszabb)
          akkor          [már végig fel van dolgozva]
                VeremÜres(minta)
        különben         [az utolsóval nem kódolható]
                VeremÜres(minta); Verembe(minta,jel)
      Elágazás vége
      Ha ker.melyik≠ktMax akkor [ilyen kód még nincs a táblában]
          ktMax: +1
          kt(ktMax):=KódTábla(leghosszabb,jel)
      Elágazás vége
    Elágazás vége
    Ír(Y,leghosszabb)
  Ciklus vége
  Ha nem ÜresVerem?(minta) akkor [az utolsó jel még kódolandó]
        Ír(Y,KódSorszám(Veremből(minta)))
  ... [file-zárások]
Eljárás vége.
```

Pontosan „veremnyi" minta keresése.

```
Függvény KódKeres(Konstans m: Verem(Karakter),
                              KT: KódTábla): KeresettTip

  Változó mjel     :Karakter
          kódolt,eredeti: Verem(Karakter)
          siker    : Logikai
          hol,honnan: KódSorsz

  Ha EgyEleműVerem(m) akkor
      KódKeres:=KeresettTip(Igaz,KódSorsz(Veremből(m)))
                        különben
      eredeti:=m;   honnan:=ktMax [a keresés kiinduló pontja]
      siker:=Hamis;   hol:=honnan [a keresés aktuális pontja]
```

[8] Egyetlen jelnek mindig van LZW-kódja.

```
      Ciklus amíg nem siker
        mjel:=Veremből(m)
        Ciklus amíg kt(hol).eleje≠0 és
                    mjel=kt(hol).bővítés
          mjel:=Veremből(m);  hol:=kt(hol).eleje
        Ciklus vége
        Ha VeremÜres?(m) és kt(hol).eleje=0 [egyszerre vége]
           és mjel=kt(hol).bővítés [első jelük is azonos]
        akkor     KódKeres:=KeresettTip(Igaz,honnan)
                  siker:=Igaz
        különben  m:=eredeti;  honnan:-1
                  hol:=honnan
                  KódKeres.van:=Hamis;  siker:=honnan=0
        Elágazás vége
      Ciklus vége
  Elágazás vége
Függvény vége.

Függvény KódSorszám(Konstans j: Karakter): KódSorsz
  KódSorszám:=KódSorsz(j)
Függvény vége.

Függvény Dekódol(Konstans kód: KódSorsz):
                                          Verem(Karakter)
  Változó i     : KódSorsz
          dekód: Verem(Karakter)
  i:=kod;  VeremÜres(dekód)
  Ciklus
    Verembe(dekód,kt(i).bővítés);  i:=kt(i).eleje
  amíg i≠0
  Ciklus vége
  Dekódol:=dekód
Függvény vége.[9]
```

3. Szövegminta keresés

A feladat a következő: adott egy s szöveg (akár lehet *inputszövegfile* is), amelyben meg kell keresni egy **sminta** szöveg első elhelyezkedésének s-beli pozícióját. A *keresés tételből* levezethető megoldás igen egyszerű (a ciklusban az *eldöntés tételét* is fölhasználtuk):

```
  ...
  siker:=Hamis;   i:=1       [ahol tartunk az s-ben]
  Ciklus amíg i≤Hossz(s)-Hossz(sminta)+1 és nem siker
    j:=1      [ahol tartunk az sminta-ban]
```

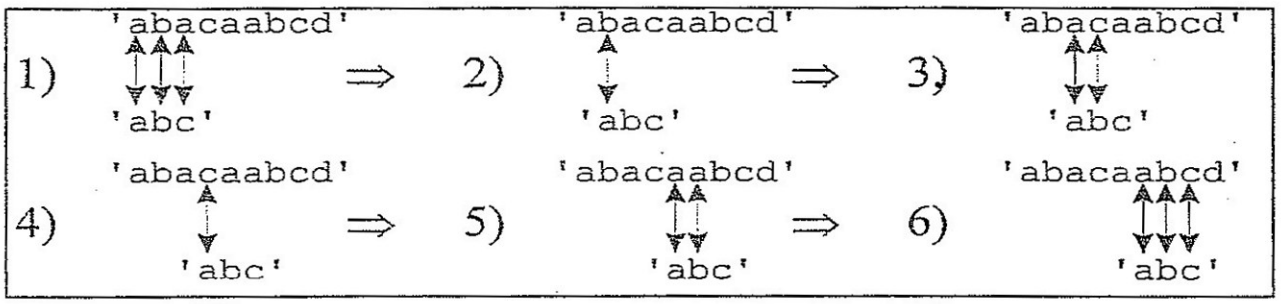
[9] A verem-műveleteket –természetesen– definiáltnak feltételezzük.

```
    Ciklus amíg j≤Hossz(sminta) és
                Jele(sminta,j)=Jele(s,i+j-1)
      j:+1
    Ciklus vége
    siker:=(j>Hossz(sminta))
    Ha nem siker akkor i:+1
  Ciklus vége
  ...
```

3.1. Eltolás a minta jelei alapján

Az egyszerűség mellett az is nyilvánvaló, hogy a fenti megoldás nem veszi igazán figyelembe a feladat (és az adatszerkezet) konkrét specialitásait. Mi most ezen jellegzetességeket fedezzük föl az alábbi példán!

Legyen az s='abacaabcd' és az sminta='abc'. A fenti „naív" algoritmus a következő lépéseket teszi:



(Jelmagyarázat: karakterhasonlítás egyezéssel, illetve ütközéssel.)

Amit az ábra láttán észre lehet venni: azt az *s*-re vonatkozó tapasztalatot, amit az algoritmus akkor szerez, amikor összehasonlítja az *s*-beli i+j. és az *sminta*-beli j. elemet, nem hasznosítja a későbbi összehasonlításoknál. Márpedig „tudhatná" pl. a 2)-lépésben, hogy az *s*-beli 2. jelnél (a 'b'-nél) nem is kezdődhet a keresett. (...) Ezeket a „tudhatná"-kat az sminta előzetes elemzése útján lehetne az algoritmusba bevenni. Arra gondolunk, hogy valamiként az algoritmus fölkészülne a sikertelen esetekre úgy, hogy „honnan lehet, *mennyivel odébb lehet a keresést folytatni* biztonságosan és hatékonyan".

Az alapötlet máris jön: vegyünk föl az *smintá*hoz egy *eltolás* vektort, amely azt adja meg, hogy az smintabeli i. karakter nem egyezésekor men*nyivel léptethető jobbra a keresett mintaszöveg*. A fenti algoritmus[10]

`'Ha nem siker akkor i:+1'`	sorát ki kellene cserélni
`'Ha nem siker akkor i:+eltolás(j)'`	a sorra.

[10] Ezzel eljutottunk a **Knuth-Morris-Pratt**-féle keresés egyik lényegi újításához.

Érzésre a fenti *smintá*hoz a következő *eltolás* vektort gyárthatnánk: 'abc' → 112. Ugyanis ha a keresett szöveg 1. jele nem egyezik meg az *s*-beli soron következő jellel, akkor még az *s*-beli azt következő jó lehet. Tehát csak 1-gyel léptethetjük. Hasonlóan, ha a 2. jelnél tapasztalunk ütközést, akkor sem tehetünk másként. (Más lenne a helyzet, ha az *sminta* első két jele megegyezne!) Ha viszont a 3. jel az első eltérő, akkor –kihasználva, hogy az előző helyeken egyezés volt és nincs *smintá*ban eddig két azonos– átléphetjük a két, már „ismert" *s*-belit.

Érdemes továbbgondolkodni a dolgon. Nem lehetne egyetlen keresési menettel az előbbinél több információt szerezni, helyesebben olyan körülményeket teremteni, hogy jobban ki tudjuk használni a szerzett ismereteket? Felhívhatja a figyelmünket az első jel vizsgálatával szerzett információ „kevéssége". Ha ez a jel nem egyezik, akkor ezt az ismeretet semmire sem lehet fölhasználni. Sajnos ez természetes, hiszen az *smintá*nak ekkorra már „alig van" olyan karaktere, amelynél kamatoztathatnánk ezt a negatív tudást (elhaladtunk mellette). Innen jöhet a következő érzésünk: a keresést nem elölről, hanem az *sminta utolsó jelénél kezdjük.* S így –képszerűen megfogalmazva– mintegy beleszaladunk ebbe a tapasztalásba, nem úgy, mint előbb, amikoris „elszaladtunk mellette".

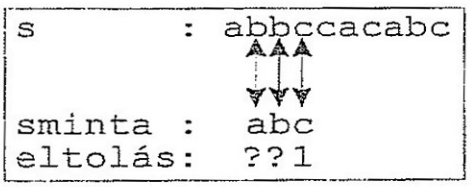
```
...
siker:=Hamis;  i:=1                [ahol tartunk az s-ben]
Ciklus amíg i≤Hossz(s)-Hossz(sminta)+1 és nem siker
  j:=Hossz(sminta)                 [ahol tartunk az sminta-ban]
  Ciklus amíg 1≤j és Jele(sminta,j)=Jele(s,i+j-1)
    j:-1
  Ciklus vége
  siker:=(j=0)
  Ha nem siker akkor i:+Eltolás(j)
Ciklus vége
...
```

Most már csak az a nagy kérdés, hogyan készíthető el az *smintá*hoz az *eltolás* vektor. Ennek a vektornak érdekes tulajdonságait derítjük föl az alábbi példák segítségével.

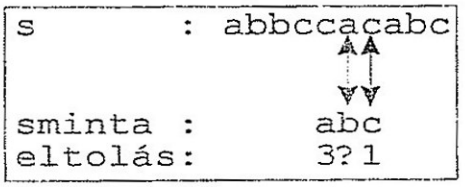
1) „Alapeset":

```
s       : abbccacabc
          ▲
          ▼
sminta  : abc
eltolás: ???
```

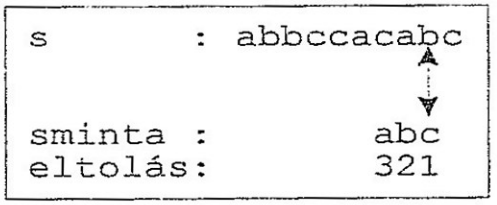
Az összehasonlításban mindjárt az első elem ütközik, nincs mit tenni, 1-gyel kell jobbra léptetni a mintát.



Az ütközés a 3. jelnél következik be, de mivel addig, a mögötte állók mind különböztek tőle, s egymástól, s rendre megegyeztek az s-beli párjaikkal, ezért átlépheti a mögöttieket: 3.



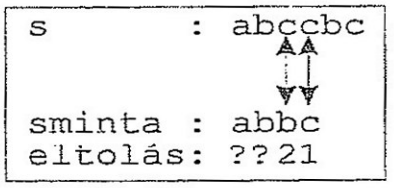
Hasonló történik a 2. jelnél, a léptetés helyes értéke: 2.

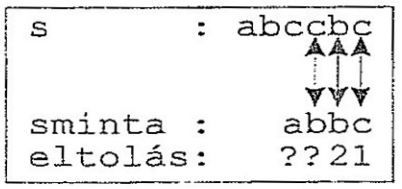


Az *eltolás*vektor kész; a léptetés utáni helyzet már ismerős ...

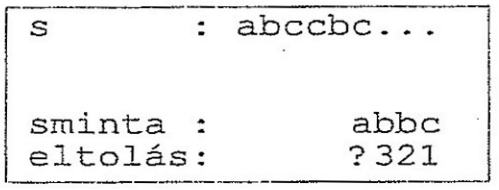
Következtetésünk: amennyiben a keresett minta *minden eleme különböző*, a léptetés értéke a *hátulról számított pozíció*val egyezik meg.

2) „Közel ismétlődő" jel a mintában:





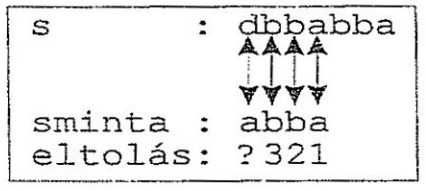
Az ütközés egy olyan helyen történik, amely jel már másodszor fordul elő ebben a menetben; mivel az ezt követő szakaszban nem ismétlődik meg vele kezdődő, most 1-hosszúságú periódus, ezért végéig elléptethetjük: 3.



A továbbiak most számunkra nemigen érdekesek; a 4. jel eltolása nyilvánvalóan 4.

Következtetés: ha van *ismétlődés* a mintában, de elég *közel* vannak egymáshoz az azonos jelek, akkor ez nem jelent különbséget az 1) esettől.

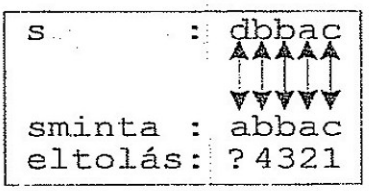
3) „Távol ismétlődő" jel a mintában:



Ütközik egy olyan karakternél, amelyik távolabb van már, mint az előző azonos a végétől; mivel nincs mi alapján kizárni az esetleges egyezést az utolsó pozíción, ezért a léptetés csak 3.

Következtetés: ha az *ismétlődő jelek távolsága nagyobb, mint az első azonos jel pozíció száma, akkor vagy az ismétlődő jel pozíció száma lesz a léptetési érték, vagy a kettejük pozíciószámának különbsége.*

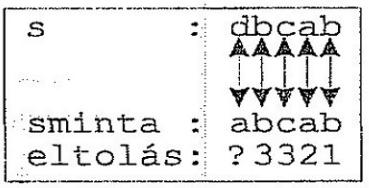
4)



Itt, bár az ismétlődés távolsága kellő ahhoz, hogy akár „részperiódus" is lehessen, mégis a pozíciószáma rendelhető hozzá lépésszámként, mivel az őt összehasonlításban megelőző nem ismétlődő jel; így az eltolásértéke: 5.

Következtetés: az eddigiek alapján állítható, hogy az eltolási vektor elemei *monoton növekedőek* (hátulról értve ezt is).

5) „Összefonódó ismétlődések"



A szituáció: összefonódó ismétlődő jelek; az összefonódás olyan, hogy ismétlődő periódusok nem kizárhatók, ezért a léptetés csak az előző azonosig történhet: 3.

Foglaljuk tehát mindezeket össze! Az *eltolási vektor*

1) hátulról monoton növekvő,

2) a (hátulról) elsőként előforduló hátulról vett sorszáma,

3) a nem először előforduló esetén vagy az előzőtől vett távolság (ha érvényes marad az 1. szabály); vagy a saját sorszáma (hátulról), ha előző (hátulról) az első ilyen, vagy ha az 1. szabály szerinti minimális távolságon túl van vele azonos, akkor a legközelebbi ilyentől vett távolsága.

Az algoritmus a következő: (*Evektor* jelöli az eltolásvektor típusát)

```
Eljárás EltolásVektor(Konstans minta: Szöveg,
                      Változó l: Evektor):
  Változó n,i: PozEgész
  n:=Hossz(minta); l(n):=1
  Ciklus i=n-1-től 1-ig -1-esével
     [ismétlődés-keresés:]
     l(i):=l(i+1)
```

```
    Ciklus amíg i+l(i)≤n és
                Jele(minta,i)≠Jele(minta,i+l(i))
      l(i):+1
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

3.2. Eltolás a szöveg jelei alapján

Az előző keresési eljárás tehát a minta minden jeléhez rendelt egy eltolás értéket, ami soha nem lehet nagyobb, mint a jel hátulról vett sorszáma. Másképpen fogalmazva: egy mintaillesztési menet után maximum annyival tudjuk jobbra tolni a mintát, amennyi jelösszehasonlítást végeztünk.

BOYER-MOORE módszere ennél nagyobb léptetést is lehetővé tesz.

Ez az eljárás nem a minta jeleihez rendel egy eltolás értéket, hanem a szövegben előforduló minden egyes jelhez, mégpedig nagyon könnyen előállítható módon. Legyen ez az érték a jel mintabeli sorszáma-1 (hátulról számolva, ha több ilyen van, akkor hátulról az elsőé), vagy a minta hossza, ha nem szerepel a jel a mintában!

Nézzünk meg egy-két példát arra, hogyan történik akkor a keresés!

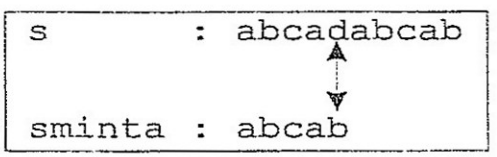
1. Példa: Legyen az

```
s : abcadabcab
sminta : abcab
```

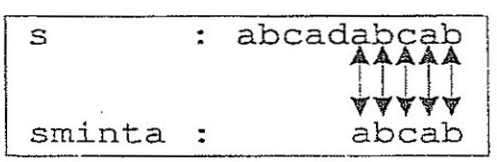
Ekkor az eltolásvektor a következő lesz:

eltolás(a)=1; eltolás(b)=0; eltolás(c)=2 és minden más jelre az értéke 5 lesz.

Az *smintát* *s*-ben balról jobbra keressük, de a hasonlítást hátulról kezdjük. Tehát:



Azt tapasztaljuk, hogy a 'd' betű nem egyezik meg a minta utolsó jelével, tehát eggyel biztosan jobbra tolhatjuk a mintát. Ennél azonban többet is észrevehetünk, nevezetesen azt, hogy a 'd' betű nem is szerepel a mintában, ennek következtében a teljes mintahosszal jobbra léphetünk. Vagyis a következő hasonlítás:



lesz, amiből a karakterenként történő hasonlítás után megállapíthatjuk, hogy a szöveg 6. karakterétől kezdve megtalálható a keresett minta.

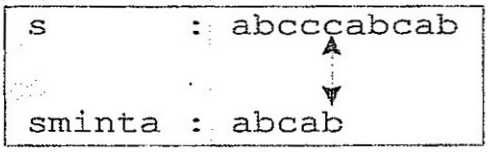
2. példa: Legyen az

```
s       : abcccabcab
sminta  : abcab
```

Mivel a minta ugyanaz, mint az előbb, az eltolásvektor is változatlan maradt, vagyis:

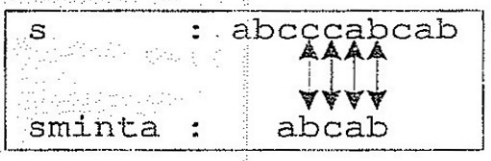
eltolás(a)=1; eltolás(b)=0; eltolás(c)=2 és minden más jelre az értéke 5.

Kezdjük megint a hasonlítást a minta végétől:



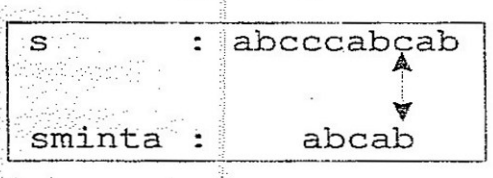
A 'c' betű nyilvánvalóan nem egyezik meg a 'b' betűvel, tehát a mintát el kell csúsztatnunk, de mennyivel? Vegyük észre, hogy eggyel hiába toljuk jobbra, mert a 'c' betű az 'a' betűvel sem egyezik meg. Kettővel elcsúsztatva már lehetséges az egyezés, hiszen így már fedésbe kerül a két 'c' betű. Célunk: úgy próbáljuk a mintát a szöveghez képest eltolni, hogy az eltérést okozó pozíción a minta feleljen meg a szöveg megfelelő jelének.

Figyeljük meg, hogy eltérés esetén eddig mindig annyival csúsztattuk jobbra a mintát, mint amennyit az *s*-ben az eltérést okozó karakterhez rendelt eltolásérték megadott. Előző példánkban az eltérést a 'd' betű okozta, az ő eltolásértéke 5 volt és ott 5-tel csúsztattuk jobbra a mintát. Most a 'c' betű okozta az eltérést, az ő eltolásértéke 2 volt és 2-vel csúsztattuk el a mintát. Folytassuk tovább a keresést!



Az eltérést most megint egy 'c' betű okozta. A hozzá rendelt eltolásérték 2, ami azt jelenti, hogy az sminta hasonlítása során már átléptünk rajta, hiszen már a negyedik hasonlításnál tartunk. Nem tudjuk tehát a mintát úgy jobbra tolni, hogy az eltérést okozó jel fedésbe kerüljön a minta megfelelő jelével.[11] Nem tehetünk ilyenkor jobbat, minthogy eggyel csúsztatjuk el a mintát. Általában ha az eltérést okozó jelhez rendelt eltolásérték kisebb, mint a megvizsgált jelek száma, akkor eltérés esetén eggyel csúsztatjuk jobbra a mintát, különben pedig az eltolásértékkel.

Most tehát eggyel toljuk el a mintát.

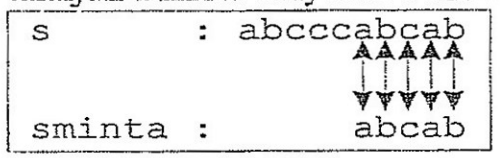


Vegyük észre azt is, hogy mielőtt még egy hasonlítást is elvégeznénk, rögtön elcsúsztathatjuk a mintát a 'c' betűhöz rendelt eltolásértékkel, hiszen így a két betű fedésbe kerül.

[11] Hisz nem a minta adottadik 'c' jeléhez tartozik az eltolásérték, hanem a 'c' jelhez, „általában".

Nem okoz problémát ez a hasonlítás előtti csúsztatás akkor sem, ha az első vizsgálandó jel nem szerepel a mintában, ilyenkor nem kerülnek fedésbe a jelek, de jókorát lépünk előre.

Példánkban a 'c' betűhöz rendelt értékkel, tehát 2-vel eltolva a mintát, megtaláljuk a minta helyét az s szövegben:



Ezek alapján a keresési algoritmus akkor a következő:

```
siker:=Hamis;  m:=Hossz(sminta)
i:=m
Ciklus amíg i≤Hossz(s) és Eltolás(Jele(s,i))≠0
  i:+Eltolás(Jele(s,i))
Ciklus vége
Ciklus amíg i≤Hossz(s) és i+Eltolás(Jele(s,i))≤Hossz(s)
        és nem siker
    [mintakeresés:]
  j:=m; siker:=(Jele(sminta,j)=Jele(s,i))
  Ciklus amíg j>1 és siker
    j:-1; i:-1
    siker:=(Jele(sminta,j)=Jele(s,i))
  Ciklus vége
  Ha nem siker akkor
    Ha Eltolás(Jele(s,i))>m-j+1
        akkor  i:+Eltolás(Jele(s,i))
      különben i:+m-j+1
    Elágazás vége
    Ciklus amíg i≤Hossz(s) és Eltolás(Jele(s,i))≠0
      i:+Eltolás(Jele(s,i))
    Ciklus vége
  Elágazás vége
Ciklus vége
[siker ⇒ a minta az s i.-nél kezdődik]
```

Nézzük, akkor hogy tudjuk meghatározni az eltolásvektort!

```
Eljárás EltolásVektor(Konstans sminta: Szöveg,
                      Változó l: Evektor):
  Változó l:Tömb(''..'z': Egész)
  Ciklus i=''-től 'z'-ig
    l(i):=Hossz(sminta)
  Ciklus vége
  Ciklus j=Hossz(sminta)-tól 1-ig -1-esével
    Ha l(Jele(sminta,j))=Hossz(sminta)
        akkor l(Jele(sminta,j)):=Hossz(sminta)-j
  Ciklus vége
Eljárás vége.
```

Belegondolva az eljárás jellegébe, megállapíthatjuk, hogy ez a keresési eljárás különösen akkor hatékony, ha az 'abc', amiből a szöveget képezzük gazdag, az sminta viszont kevés karakterből áll.

3.3. Keresés leképező függvénnyel

Az eddig vizsgált szövegkeresési eljárásoktól lényegében különbözik **Rabin-Karp** algoritmusa.

Az eljárás alapgondolata az, hogy tekintsük a keresendő, *m* hosszúságú *mintát* úgy, mint egy *d* alapú számrendszerben felírt egész számot, ahol d=a szövegben előfordulható jelek száma. (Másképpen az abc elemszáma.) Rendeljünk az abc jeleihez egy-egy kódot 0-tól folyamatosan számozva.

Használjuk az sm[i]=Kód(Jele(sminta,i)) jelölést, ekkor az *mintá*hoz rendelt egész szám az alábbi alakban írható fel:

$x=sm[1]*d^{m-1}+sm[2]*d^{m-2}+.....+sm[m-1]*d+sm[m]$

Vezessük be a h(x)=x mod q leképezőfüggvényt, ahol q egy eléggé nagy prímszám.

Ezek után a módszer abban áll, hogy számoljuk ki a leképező függvény értékét a vizsgálandó *s* szöveg minden *m* hosszúságú részszövegére is, és ha a szöveg valamely i. pozícióján a leképezőfüggvény értéke megegyezik az *mintá*hoz rendelt értékkel, akkor megtaláltuk a mintát a szövegben.

Megint használva az s[i]=Kód(Jele(s,i)) jelölést az *s* szöveg i. karakterénél kezdődő *m* hosszúságú részszöveghez rendelt egész szám a következő lesz:

$y=s[i]*d^{m-1}+s[i+1]*d^{m-2}+.....+s[i+m-1]$ és h(y)=y mod q.

Mondhatnánk, hogy valóban megspórolja ez az algoritmus a karakterenként történő hasonlítást a korábban vizsgált eljárásokhoz képest, de nem kevesebb munka kiszámolni a részszövegekhez rendelt számokat sem! Vegyük azonban észre, hogy az i+1. helyen kezdődő részszöveghez rendelt számot nem kell *m* darab hatványösszegből számolni, hanem egyszerűen képezhető az előző pozícióhoz rendelt számból az alábbiak szerint:

$y=(y-s[i]*d^{m-1})*d+s[i+m]$.

Használjuk ki a **mod** művelet számunkra most hasznos tulajdonságát, miszerint

(a + b) **mod** q = ((a **mod** q) + b) **mod** q!

Tegyük fel, hogy az ábécénket most az angol abc nagybetűi alkotják. Ekkor d=26. Kód(betű) legyen a betű ASCII kódja-65, ez akkor 0-tól folyamatosan rendeli a jelekhez az értéket.

Válasszuk a prímszámnak a q=33554393 értéket, ez elég nagy, ugyanakkor még elég kicsi ahhoz, hogy a d*q szorzás ne okozzon túlcsordulást. Ilyen választások mellett a keresési eljárás az alábbi lesz:

```
Eljárás Stringkeresés(Konstans s,sminta: Szöveg,
                      Változó siker: Logikai,
                              i:Egész):
  Konstans  q : PozEgész(33554393)
            d : PozEgész(26)
  Változó m,s1,s2,dh: PozEgész
  m:=Hossz(sminta);  dh:=1
  Ciklus i=1-től m-1-ig
    dh:=(d*dh) mod q
  Ciklus vége
  s1:=0
  Ciklus i=1-től m-ig
    s1:=(s1*d+Kód(Jele(sminta,i))) mod q
  Ciklus vége
  s2:=0
  Ciklus i=1-től m-ig
    s2:=(s2*d+Kód(Jele(s,i))) mod q
  Ciklus vége
  i:=1
  Ciklus amíg (s1≠s2) és i≤Hossz(s)-m
    s2:=(s2+d*q-Kód(Jele(s,i))*dh) mod q
    s2:=(s2*d+Kód(Jele(s,i+m))) mod q
    i:+1
  Ciklus vége
  siker:=(s1=s2)
Eljárás vége.
```

Megjegyzés: Az s2 kiszámítását a megoldásban 2 lépésre bontottuk a túlcsordulás elkerülése érdekében és az első lépésben hozzáadtunk egy nagy számot, a d*q -t, hogy az érték mindig pozitív maradjon és a *mod* helyesen dolgozzon.

A fenti megoldás sajnos nem 100 százalékos biztonsággal helyes: a *mod* művelet alkalmazása miatt ugyanis különböző számokhoz ugyanazt a maradékot rendelhetjük. Elég nagy osztó esetén ez nagyon kis valószínűséggel fordul elő. A tökéletes megoldásban a kódegyenlőség fennállása esetén meg kell vizsgálni a karakterenkénti egyenlőséget is, s csak annak teljesülése esetén szabad befejezni a keresést. Ennek a változatnak az elkészítését Olvasóinkra bízzuk.

4. Hiányos szövegminta keresése

Szövegszerkesztőkben, a DOS operációs rendszer parancsaiban és a legtöbb jól ismert programcsomagban lehetőségünk van olyan szöveg keresésére, megadá-

sára, amit nem ismerünk pontosan vagy kényelmi okokból nem kívánunk teljes hosszában kiírni.

Ilyenkor nem egy konkrét szöveget, hanem egy szövegmintát adunk meg keresésre. A mintában általában használatos ún. **joker** karakterek a '?' és a '*' jel.

Jelentésük a következő: amelyik karakterpozíción a '?' áll, azon a helyen a keresendő szövegben *bármilyen más, egyetlen* jel állhat.

Például az 'aa?bb' mintának az 'aa bb' szöveg megfelel.

A minta '*' jele pedig azt jelenti, hogy a mintának azok a szövegek felelnek meg, melyekben az első jeltől a '*'-ig pontos az egyezés, a '*' helyén pedig *tetszőleges hosszú, tetszőleges karaktersorozat* (akár az üres szöveg is) állhat.

Kétféle '*' **joker** értelmezés szokásos:

A. A '*' helyén bármilyen jelsorozat állhat és vizsgálnunk kell a mintában a '*' utáni karakterek egyezését is.

Példa:

```
s       : ab
sminta  : a*b
```

esetén megtaláljuk az első helyen a mintát,

vagy

```
s       : aaabarmibb
sminta  : a*b
```

esetén megtaláljuk a mintát, kérdés, hogy hányadik karakternél?

Mondhatjuk, hogy a teljes *s* felel meg a mintának, hiszen a-val kezdődik és b-vel végződik. Hasonlóan helyes válaszok lennének, hogy a minta a [2,9], [2,10], [3,4], [3,9], [3,10], [5,9], [5,10] helyen van. Melyiket válasszuk a sok közül? Lehetne például a válaszunk az, hogy a legrövidebbet feleltessük meg a mintának, tehát itt a [3,4]-et.

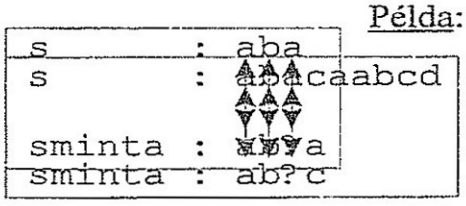
B. Másik értelmezés, hogy a mintában a '*' jel helyett bármilyen jelsorozat állhat és a '*' utáni karakterekkel már nem kell foglalkoznunk. Mi ezentúl ezt az értelmezést fogjuk használni.

A feladat tehát a következő:

Adott egy *s* szöveg (akár lehet input szövegfile is), amelyben meg kell keresni egy *sminta* szöveg első előfordulásának *s*-beli pozícióját. Az *sminta* szöveg tartalmazhat joker karaktereket is.

A megoldáshoz induljunk ki a 3.3.-ban tárgyalt legegyszerűbb, a **keresés** és **eldöntés tétel**éből levezethető algoritmusból. Nézzük meg, milyen módosítást eredményez először is a '?' joker karakter bevezetése!

Példa:



Az eredeti algoritmus a 3. karakternél történő hasonlítás után lépteti a mintát, mivel azt tapasztalja, hogy **Jele(sminta,j)≠Jele(s,i+j-1).**

A '?' joker jelentése szerint viszont a '?' helyén bármi állhat, ezért tovább kell folytatni az *sminta* karaktereinek hasonlítását. Nézzünk egy másik példát!

Az eredeti algoritmusunk hozzá sem fog a karakterek hasonlítgatásához, hiszen az *s* rövidebb, mint az *sminta*, holott általában azok a rendszerek, amik megengedik a jokerek alkalmazását, ezt a szöveget a mintához illeszkedőnek találják, mondván, hogy ha az *sminta*ban az utolsó jel a kérdőjel, akkor a helyén akár az üres karakter is állhat.

Egyszerűen megoldhatjuk akkor ezt a problémát úgy, hogy amennyiben a minta végén a '?' joker karakter van, hagyjuk el azt belőle a keresés eljárás meghívása előtt.

Nem engedik meg viszont általában azt, hogy a minta belsejében álljon üres karakter a '?' helyén. Az alábbi esetben pl. azt kell kapnunk válaszul, hogy az *sminta* nem található meg az *s*-ben.

Nézzük most milyen módosítást követel a '*' joker bevezetése! Azt kell észrevennünk, hogy ha az *sminta* és *s* hasonlításában eljutottunk az *sminta* első '*' jeléig, akkor készen vagyunk, megtaláltuk az *sminta*t. (Ugyanis szo* és szo*valami minták ugyanazok.) A ciklus belsejében tehát a *siker* értékét igazra kellene állítanunk, ha '*'-ot találtunk és a ciklusfeltételbe is bele kellene vennünk, hogy csak addig hasonlítson, amíg *siker* hamis.

Ekkor a `siker:=(j>Hossz(sminta))` értékadást is módosítanunk kellene a `siker:=(siker vagy j>Hossz(sminta))` értékadásra.

Már szinte készen vagyunk az algoritmussal, mielőtt azonban befejeznénk a megírását, vizsgáljuk meg az alábbi speciális esetet:

```
s       : valamit
sminta  : mit*
```

Ez esetben Hossz(s)=7 és Hossz(sminta)=4, s mivel az algoritmus az *sminta* utolsó illesztését az i=Hossz(s)–Hossz(sminta)+1. karaktertől kezdve végzi el, ezekre az adatokra azt fogja mondani, hogy *sminta* nem szerepel az *s*-ben, ami nem igaz (ugyanis a '*' helyén az üres szöveg is állhat). Ez hasonló probléma, mint a '?' joker esetén volt. Itt azonban az

```
s       : szo
sminta  : szo*valami
```

esetén is jelentkezik a hossz problémája.

Ezt a problémát akkor úgy tudjuk megoldani, legegyszerűbben, hogy ha van az *sminta*ban '*', akkor azt és a mögötte levő karaktereket hagyjuk el még a keresés előtt. Ekkor viszont már a '*'-gal egyáltalán nem kell törődnünk.

Mindezeket összefoglalva akkor az alábbi algoritmust kapjuk:

```
Eljárás Stringkeresés(Konstans s,sminta: Szöveg,
                      Változó siker: Logikai,
                              i: Egész):
  Ha Jele(sminta,Hossz(sminta))='?'
     akkor sminta:=Eleje(sminta,Hossz(sminta)-1)
  Ha Része?('*',sminta)
     akkor i:=1
           Ciklus amíg Jele(sminta,i)≠'*'
             i:=i+1
           Ciklus vége
           sminta:=Eleje(sminta,i-1)
  Elágazás vége
  siker:=Hamis:  i:=1            [ahol tartunk az s-ben]
  Ciklus amíg i≤Hossz(s)-Hossz(sminta)+1 és nem siker
    Van?(s,sminta,siker,i)
    Ha nem siker akkor i:+1
  Ciklus vége
Eljárás vége.
```

Az új Van? eljárás, ami tehát most már alkalmas bármilyen joker karaktert tartalmazó *sminta* megtalálására a következő:

```
Eljárás Van?(Konstans s,sminta: Szöveg,
             Változó siker: Logikai, i: Egész):
  j:=1          [ahol tartunk az sminta-ban]
  jó:=Igaz
  Ciklus amíg j≤Hossz(sminta) és jó
    jó:=(Jele(sminta,j)=Jele(s,i+j-1) vagy
         Jele(sminta,j)='?')
    j:+1
  Ciklus vége
  siker:=(j>Hossz(sminta)
Eljárás vége.
```

Hasonló módon, mint azt a jokerek nélküli szövegminta keresésnél tettük, a lineáris keresés helyett alkalmazhatjuk a sok esetben sokkal gyorsabb **KNUTH-MORRIS-PRATT** vagy **BOYER-MOORE** keresési eljárásokat, természetesen elvégezve rajtuk a megfelelő módosításokat.

Vizsgáljuk most meg azt, hogy milyen módosításra van szükségünk! Mint korábban megállapítottuk, a '*' jokerrel nem kell foglalkoznunk, hanem a mintából a keresés előtt el kell hagynunk a '*' és a mögötte levő karaktereket.

Mi a helyzet a '?' jokerrel? A **KNUTH-MORRIS-PRATT** módszer szerint a minta minden karakterpozíciójához rendeltünk egy léptetésértéket, s ha ennél a pozíciónál volt az eltérés, akkor a megfelelő léptetésértékkel toltuk jobbra a mintát és kezdtük újból a minta és a szöveg karakterenkénti hasonlítását. Kérdés akkor, hogy milyen léptetésértéket rendeljünk a '?' jel karakterpozíciójához? Mivel a '?' helyén bármi állhat, ilyenkor nem kell a mintát elcsúsztatnunk, tehát adódhat az az ötlet, hogy rendeljük hozzá a 0 léptetésértéket. Az eltérés esetén azonban az eredeti algoritmus elvégzi a csúsztatást (itt a 0-val való csúsztatást) és a minta végénél kezdi újra a hasonlítást. Ennek következtében '?' esetén végtelen ciklusba esne az algoritmusunk. Következésképpen a '?' karakterpozíciójához bármilyen értéket rendelhetünk, a keresésben úgyis meg kell vizsgálnunk, hogy a '?' okozta-e az eltérést, s ha igen, akkor folytassuk tovább a következő pozíción a hasonlítást. Javaslatunk akkor az, hogy az eltolásvektor meghatározásakor ne foglalkozzunk külön a '?' jellel, maradjon ő ugyanolyan karakter, mint a többi (így megspórolunk egy elágazást), csak a keresési eljárást módosítsuk. A módosítás lényege abban áll, hogy akkor kell továbblépnünk a hasonlításban, ha a minta és a szöveg aktuális karaktere megegyezik, vagy ha a minta aktuális karaktere a '?'. Pontosabban a **KNUTH-MORRIS-PRATT** keresésben a ciklusfeltételben szereplő

`Jele(sminta,j)=Jele(s,i+j-1)` feltételt kell kicserélni a
`Jele(sminta,j)=Jele(s,i+j-1)` **`vagy`** `Jele(sminta,j)='?'`
feltételre.

Hasonlóképpen kell eljárnunk a **BOYER-MOORE** keresésnél, ott a

`siker:=(Jele(sminta,j)=Jele(s,i))` értékadást kell lecserélni a
`siker:=(Jele(sminta,j)=Jele(s,i)` **`vagy`** `Jele(sminta,j)='?')`
értékadásra.

Vegyük észre, hogy a **RABIN-KARP** algoritmus nem alkalmas '?' joker kezelésére, hiszen ott pontosan kell hasonlítani az *sminta* karakterkódösszegét a szöveg megfelelő hosszúságú részleteinek karakterkódösszegeivel.

IV. Az Absztrakt Sorozat, mint a SzövegTípus általánosítása

A mindennapi gyakorlatban nagyon sok feladat megoldásánál találkozunk azzal az igénnyel, hogy olyan *struktúraösszeállító eszközt* alkalmazzunk, amely vészesen emlékeztet a *szövegtípus*ra, pontosabban olyan műveletek és függvények (operációk) lennének célszerűek ezek megoldásánál, amelyek megvannak minimumként a szövegtípus esetében, de hiányoznak az adott konkrét elemtípushoz. Ez a gondolat vezet bennünket el oda, hogy megkíséreljük egy új *típuskonstrukció* megalkotását, amelynek tehát kiinduló alapjául maga a szövegtípus szolgál.

Ebben a fejezetben vázoljuk ezen eszközt, és példákat adunk fölhasználási körükből. Így kerül szóba egyik klasszikus *kifejezéskiértékelő* módszer a Rutishauser-féle. Majd a *számítógépi grafika* vidékére kirándulunk ugyanilyen példaadási okból.

Legelőszöris mit értsünk a továbbiakban *absztrakt sorozat*on? A szövegtípusúakon szokásos műveletek, függvények voltak a következők: 'vedd a *baloldali részét*', 'vedd a *jobboldali részét*', vagy 'mi az *i. eleme*', 'milyen *hosszú*' stb. . Ezeket a műveletek rendeljük hozzá egy absztrakt sorozatként felépített objektumhoz is, csak éppen az elem, az egység, amelyekből összeáll, az nem a karakter, hanem egy tetszőleges *elemtípus*. Tehát az elvárásainknak kifejező exportmodul:

```
ExportModul AbsztraktSorozat(Típus ElemTip):
  Típus
    Index = reprezentációfüggő
  Eljárás
    Üres(Változó as: AbsztraktSorozat)
  Függvény
    Hossz(Konstans as: AbsztraktSorozat): Index
    BalRész(Konstans as: AbsztraktSorozat,
                      meddig: Index): AbsztraktSorozat
    JobbRész(Konstans as: AbsztraktSorozat,
                       mettől: Index): AbsztraktSorozat
    Elem(Konstans as: AbsztraktSorozat,
                   melyik: Index): ElemTip
  Operátor
    +(Konstans s1,s2: AbsztraktSorozat):
                              AbsztraktSorozat
    +(Konstans s1: AbsztraktSorozat,
                  e: ElemTip): AbsztraktSorozat
Modul vége.
```

A reprezentációról: amíg a szövegtípus esetén –bár fölvetődött– elvetettük minden karakterszintű *láncolás* lehetőségét, mivel bántóan pazarolta volna a memóriahelyet, addig egy fenti típusú objektum esetén alighanem az egyetlen ésszerű választás. (Bár a 3D grafikáról szóló részben folytonos ábrázolással definiáljuk.)

1. Az aritmetikai kifejezések kiértékelésének egy módszere

Feltételek az aritmetikai kifejezésre:

- lexikális egységekre bontva,
- csak bináris, 'köztes' jelölésű műveleteket tartalmazhat,
- teljesen zárójelezett alakban (azaz minden művelet az ő két operandusával együtt zárójelezve, attól függetlenül, hogy a szokásos precedencia ezt nem igényelné).

A módszer:

- minden lexikális egységhez egy ún. *szintszám*ot rendelünk (a későbbi szabályok szerint), ez a tulajdonképpeni *fafölépítés*;
- a legmagasabb szintszámhoz tartozó részkifejezést kiértékeljük (végrehajtjuk), és helyettesítjük eredményével, *redukáljuk a fát.*

A szintszám-hozzárendelés szabályai:

- kezdetben 0,
- '(' és a műveleti jelek növelik eggyel,
- ')' és az operandusok csökkentik eggyel,

Példa:

```
( ( ( 7 + 2 ) * 3 ) / ( 5 - 2 ) )
0 1 2 3 2 3 2 1 2 1 0 1 2 1 2 1 0
( (     9     * 3 ) / ( 5 - 2 ) )
0 1     2       1 2 1 0 1 2 1 2 1 0
(             27    / ( 5 - 2 ) )
0               1     0 1 2 1 2 1 0
(         27          /     3     )
0         1           0     1     0
                      9
                        0
```

```
Típus LexEgys=???
      LexKif=AbsztraktSorozat(LexEgys)
```

```
      Szintek=AbsztraktSorozat(Egész)
      Operátor=('(',')','+','-',...)
Függvény Kiértékel(Változó lk: LexKif): Valós
                                                    [Rutishauser]
  Változó sz           : Szintek
          op1,op2,műv  : LexEgys
          eredmény     : Valós
          hol,rkszint  : Egész          [részkifejezés-szintszám]
  SzintSzámozás(lk,sz)
  Ciklus amíg Hossz(lk)>1
    MaxSzintű(sz,hol)
    op1:=ElemÉrték(lk,hol):  op2:=ElemÉrték(lk,hol+2)
    műv:=ElemÉrték(lk,hol+1)
    rkszint:=ElemÉrték(sz,hol+1)
    eredmény:=Kiszámol(op1.azonosító,műv.azonosító,
                       op2.azonosító)
    lk:=Összerak(BalRész(lk,hol-2),eredmény,
                 JobbRész(lk,hol+4))
    sz:=Összerak(BalRész(sz,hol-2),rkszint,
                 JobbRész(sz,hol+4))
  Ciklus vége
  Kiértékel:=eredmény
Függvény vége.

Eljárás SzintSzámozás(Konstans lk: LexKif,
                      Változó sz:Szintek):
  ???
Eljárás vége.

Függvény Kiszámol(Konstans o1: Szöveg, m2: Operátor,
                  o2: Szöveg): Valós
  ???
Függvény vége.

Függvény Összerak(Konstans bal: AbsztraktSorozat,
                  elem: ElemTípus,
                  jobb: AbsztraktSorozat):
                                AbsztraktSorozat
  ???
Függvény vége.
```

2. 3D-grafika *(egy grafikus típus általánosításáról)*

2.1. Bevezető megjegyzések

A testek ábrázolása "hagyományos leírással" a következő lépésekben adható meg:

```
Konstans MaxLapszám : Egész(???)

Típus Test=Rekord
            (lapszám: Egész
             lap: Tömb(1..MaxLapszám: Lap))
```

```
Konstans MaxPontszám : Egész(???)
Típus Lap=Rekord
            (pontszám: Egész
             pontok: Tömb(1..MaxPontszám: Pont))
Típus Pont=Rekord(x,y,z: Valós)
```

Célszerű még néhány kiegészítést tenni a tömör testek ábrázolásához:

a.) Takarás megoldása konvex testek esetén:

α.) a lapot alkotó *pontok adott* ("kívülről nézve" az óra járásával ellentétes) *irányú felsorolása*, lehetővé teszi, hogy a külső oldal megkülönböztethető legyen a belső, nem láthatótól;

β.) a lap *pontjai mellett* egy *kifelé mutató normálvektor* hozzávétele;

γ.) a lap, mint sík, az *egyenlet*ével adandó meg, továbbá a *pontok már mint 2D-beli pontok* pl. az x-, y-koordinátájukkal;

b.) Takarás megoldása konkáv testek esetén:

α.) a lap *külső és belső* oldalának külön (duplán, önállóan) megadása;

β.) a lap *mindkét (irányú) normálvektor*ának megadása.

A testek definiálása jócskán egyszerűsödik, ha segítségül hívjuk az absztrakt sorozat-típuskonstrukciós eszközt. Mindössze három sor.

```
Típus Test= AbsztraktSorozat(Lap)
      Lap = AbsztraktSorozat(Pont)
      Pont= Rekord(x,y,z: Valós)
```

2.2. 3D struktúra típusdefiniálása

Először az eddig körvonalazódott testdefiníciót általánosítjuk *több test*et alkotó *struktúrá*vá. Kezdjük a reprezentációval!

```
Típus Struktúra   = AbsztraktSorozat(KonkrétTest)
      KonkrétTest= Rekord
                    (test      : AbsztraktTest
                     azonosító: Név
                     viszony:[1] Rekord
                                  (origó       : Pont
                                   tengelyszög: Rekord
                                        (x,y,z: Valós)
                                   lépték      :Rekord
                                        (x,y,z: Valós),
)
```

[1]Struktúrába illeszti az absztrakt testet.

A `viszony` *mező elhagyható*, de ekkor a konkrét testnek a *struktúrába való beillesztését meg kell előzze* a "viszonynak" megfelelő *transzformációk* elvégzése, amelyet magának a programozónak kell megszerveznie. A mező megléte vagy meg nem léte a később megírandó tevékenységek végrehajtásának miként jére lesz hatással. Alapesetben –amikor a mező megvan– a beillesztés igen egyszerű, hisz csak a paraméterek beállítását jelenti, cserében viszont a megjelenítés hosszadalmas; a másik esetben pont fordítva.

```
AbsztraktTest= AbsztraktSorozat(Lap)
     [ a test leírása saját koordinátarendszerben]
Lap= Rekord
       (tartópont : Sokszög
        attribútum: Rekord
                          (szín      : Színek
                           fény      : Fényerő
                           minta     : Minták
                           vastagság: Valós) )
Sokszög = AbsztraktSorozat(Pont)
```

Megvalósítási (implementációs) elképzeléseink:

- az **AbsztraktTest**ek között *léteznek konstans* (*előre definiált*) *objektumok*, pl. tetraéder, kocka, ..., ikozaéder;
- egy (részben kész) *táblázat* épül, amelybeli sorszámmal –mint *típuskóddal*– lehet hivatkozni a **KonkrétTest**-beli **test**-mező (**AbsztraktTest**-típusú) értékére. (Grafikus adatbázis)

A 3D grafika itt megfogalmazott elképzeléseinek betetőzéseként, összegzéseként megadjuk Struktúratípus export modulját.

```
ExportModul Struktúra:
  Típus
    Struktúra
    AbsztraktTest
    KonkrétTest
    Sokszög
    Lap
    Pont
    Index
    Név
    Léptékek
    Szögek
    Nézés
```

Struktúra, mint absztrakt sorozat

```
  Eljárás
    Üres(Változó s: Struktúra)
  Függvény
    Hossz(Konstans s: Struktúra): Egész
```

```
  BalRész(Konstans s: Struktúra,
                  meddig: Index): Struktúra
  JobbRész(Konstans s: Struktúra,
                  mettől: Index): Struktúra
  Elem(Konstans s:Struktúra, i:Egész):KonkrétTest
Operátor
  +(Konstans s1,s2: Struktúra): Struktúra
  +(Konstans s1: Struktúra,
            e: KonkrétTest): Struktúra
Függvény
  Kiválaszt(Konstans s: Struktúra, i: Index):
              KonkrétTest    [ az s struktúra i. teste]
  = s[ i]
  Kiválaszt(Konstans s: Struktúra, a: Név):
              KonkrétTest    [ az s struktúra 'a' azonosítójú teste]
  = s[ a]
```

Absztrakt test, mint absztrakt sorozat

```
Eljárás
  Üres(Konstans s: AbsztraktTest) .
Függvény
  Hossz(Konstans s: AbsztraktTest): Egész
  BalRész(Konstans s: AbsztraktTest,
                  meddig: Index): AbsztraktTest
  JobbRész(Konstans s: AbsztraktTest,
                  mettől: Index):AbsztraktTest
  Elem(Konstans s: AbsztraktTest,
               i: Egész): Lap
Operátor
  +(Konstans t1,t2:AbsztraktTest):AbsztraktTest
  +(Konstans t1: AbsztraktTest,
            l: Lap): AbsztraktTest
```

Sokszög, mint absztrakt sorozat

```
Eljárás
  Üres(Kontsans s: Sokszög)
Függvény
  Hossz(Konstans s: Sokszög): Egész
  BalRész(Konstans s: Sokszög,
                  meddig: Index): Sokszög
  JobbRész(Konstans s: Sokszög,
                  mettől: Index): Sokszög
  Elem(Konstans s: Sokszög, i: Egész): Pont
Operátor
  +(Konstans s1,s2: Sokszög): Sokszög
  +(Konstans s1: Sokszög, p: Pont): Sokszög
```

Struktúra, mint geometriai típus

```
Függvény
   Beilleszt(Konstans s:Struktúra, i: Index,
                      t:KonkrétTest):Struktúra
                    [ beillesztés s-be i. testként]
   ≡ s:+t, ha i=Hossz(s)
   Beilleszt(Konstans s:Struktúra, a:Név,
                      t:KonkrétTest):Struktúra
                [ beillesztés 'a' azonosítójú testként]
   ≡ s:+t, ha az utolsó után
   Elhagy(Konstans s: Struktúra, i: Index): Struktúra
                                  [ az s struktúrából az i. elhagyása]
   ≡ s-s[ i]
   Elhagy(Konstans s: Struktúra, a: Név): Struktúra
                    [ az s struktúrából az 'a' azonosítójú elhagyása]
   ≡ s-s[ a]
Függvény
   Nyújtás(Konstans s: Struktúra,
                    l: Léptékek): Struktúra
     [ az s[i].viszony.lépték-ek szorzása rendre l.x-szel, l.y-nal, l.z-vel
      az összes i testjének]
   Eltolás(Konstans s: Struktúra,
                    v: Pont): Struktúra
     [ az s[i].viszony.origó növelése a v-vel az összes i testjének]
   Forgatás(Konstans s: Struktúra,
                     a: Szögek): Struktúra
     [ az s[i].viszony.origó eltolása és a tengelyszögek növelése az a.x-nek,
       a.y-nak, a.z-nek megfelelően a struktúra összes i testjének]
Eljárás
   AxonometrikusDrótváz(Konstans s: Struktúra,
                                  vsík: Valós)
     [ a z=0 síkra párhuzamos vetítés, vsík távolságú síkra]
   PerspektívDrótváz(Konstans s: Struktúra,
                              szem: Nézés)
     [ a z=0 síkra középpontos vetítés a szempontból]
   AxonometrikusTömör(Konstans s: Struktúra,
                               vsík: Valós)
     [ a z=0 síkra párhuzamos vetítés -a testek egymás és öntakarásának
       figyelembevételével- a vsík távolságú síkra]
   PerspektívTömör(Konstans s: Struktúra,
                            szem: Nézés)
     [ a z=0 síkra középpontos vetítés a szempontból a testek egymás és
       öntakarásának figyelembevételével]
Modul vége.
```

Mindezzel megadtuk a **Struktúra**-típus interfészét, azaz azt a felületet, *ahogy használni lehet* (és kell is). A modul megadásával kell pontosítani a reprezentációt és az asszociált tevékenységek (reprezentációhoz illeszkedő) implementációját.

Az alábbiakban még, néhány eddig nem definiált fogalom definícióját adjuk meg, s ezzel már teljesnek mondható a reprezentáció.

```
Típus Léptékek = Rekord (x,y,z: valós)
      Szögek = Rekord (x,y,z: valós)
      Nézés = Rekord
                (honnan: Pont
                 merre : Pont        [ irányvektor]
                 kúp   : Valós       [ a kúp nyílás szöge] )
```

Irodalomjegyzék

1. Mérey A: Adatszerkezetek
SZÁMOK, 1979

2. D.Knuth: A számítógép-programozás művészete 1.
Műszaki Könyvkiadó, 1988

3. D.Stubs-N.Webre: Data Structures with Abstract Data Types and Pascal
Brooks/Cole Publishing Co., 1985

4. L.Nyhoff-S.Leestma: Data Structures and Program Design in MODULA-2
Macmillan Publishing Co., 1990

5. B.Kernighan-P.Plauger: A programozás magasiskolája
Műszaki Könyvkiadó, 1982

6. R.Sedgewick: Algorithms
Addison-Wesley Publishing Co., 1983