MAKROUTASITÁSOK ALKALMAZÁSA MAGASABB SZINTŰ NYELV KITERJESZTÉSÉRE

Hunya Péter

Szeged, 1974.

T a r t a l o m

Bevezetés

Az 1950-es évek második felétől kezdődően mindinkább előtérbe kerültek a programozási tevékenység megkönnyitésével kapcsolatos feladatok, amelyek szoros összefüggésben vannak a programozási nyelvekkel és a programozás automatizálásával.

Az eljárásra orientált programozási nyelveknek - a legfontosabb alkalmazási területek igényeinek megfelelően - három fő csoportja alakult ki.* Noha valamennyi programozási nyelv kidolgozásánál bizonyos foku univerzelitásra törekedtek, mégis jól megkülönböztethető az elsősorban numerikus számitásokat szolgáló nyelvek csoportja /ALGOL 60, FORTRAN és néhány belőlük származtatott nyelv/ az adatfeldolgozási célokra készült nyelvektől /COBOL stb./; a harmadik csoportot a szimbólumsorozatok feldolgozására orientált nyelvek alkotják /IPL-V, LISP stb./.

Valamennyi nyelv --függetlenül attól, hogy melyik csoporthoz tartozik --többé-kevésbé alkalmas minden számitógépes feladat megoldására, és a feladatok sem sorolhatók mereven az egyik vagy másik kategóriához. Ennélfogva egymás után alakultak ki a módositott nyelvek /pl.FORMULA-ALGOL,LISP-ALGOL stb/, amelyek előrehaladást jelentettek a csoportok között mesterségesen meghuzott határvonalak átlépése felé. Fontos megjegyeznünk, hogy az ALGOL 68 [25] - az igen általános definiciós lehetőségek bevezetésével - egy ujabb fejlődési irányt myitott ezen a területen.

---

* BOLLIET [5] a nyelveknek csak két csoportját különbözteti meg:
  a/ a numerikus számitásokra, valamint
  b/ a szimbólumsorozatok feldolgozására orientált nyelveket.

A jelenlegi --gyakorlatban is használatos-- programozási nyelvek többsége azonban nem éri el a programozási gyakorlathoz kivánatos általánossági szintet, ami nehézségekhez vezet. Ezek áthidalására a gyakorlatban alkalmazott nyelveknél arra törekedtek, hogy a gépi reprezentációkat alkalmassá tegyék olyan utasitások /un. makroutasitások/ használatára, amelyek lehetővé teszik az esetleg különböző nyelveken irt --a rendszer könyvtárához tartozó-- programok aktivizálását is. Felvetődik az a gondolat, hogy vajon nem használhatók-e fel az ilyen makroutasitások egy adott nyelv meghatározott implementációjában, a lehetőségek szokásosnál lényegesen szélesebb körü bővitésére? Ebben az esetben ugyanis egy közös forditóprogram felhasználásával, különböző programkönyvtárak mellett egy alapnyelv különféle feladatokra orientált változatait kaphatnánk meg.

A jelen dolgozatban a magasabb szintü programozási nyelvek makroutasitásainak felhasználási lehetőségeivel foglalkozunk, rámutatva a legfontosabb gyakorlati alkalmazásokra. Részletesen ismertetjük a MINSZK-22 számitógép INZSENYER autokódjának kiterjesztését.

1. A célkitüzés általános megfogalmazása és feltételei

1.1. Célunk a magasabb szintü programozási nyelveknél alkalmazott makroutasitásokban rejlő lehetőségek elemzése, különös tekintettel arra a körülményre, hogy bizonyos esetekben lehetővé teszik a szabványostól jelentősen eltérő feladatok megoldását is. A dolgozat alapját az a munka képezte, amelyet a JATE Kibernetikai Laboratóriumában 1968 végén és 1969 elején végeztünk, az INZSENYER [3] programozási nyelv /a továbbiakban: AKI - AvtoKod Inzsenyer/ és transzlátora fejlesztésével kapcsolatban. Ebben feladatként tüztük ki a nyelv szemantikai és szintaktikus lehetőségeinek jelentős bővitését, anélkül, hogy a forditóprogramot meg kellene változtatni. A megoldáshoz az INZSENYER könyvtárrendszerét, illetőleg azt a lehetőséget használtuk fel, hogy annak programjai makroutasitásokkal aktivizálhatók a forrásprogramokban. Ezek a makroutasitások -- mint általában a magasabb szintü nyelvek makroutasitásai /beleértve az ALGOL 60, a PL/1 stb. eljárásait is, [6], [7]/, amelyek szintén makroutasitások -- bizonyos kereteken belül lehetővé teszik a nyelvben megengedett müveletek kiterjesztését [4], ami egyuttal többnyire a szintaxis bővitését is jelenti. Emellett assembly szintü nyelvekben a makrokat deklarációs tevékenység ellátására is alkalmazzák [10].

A bővités lehetőségei attól függenek, hogy a makrodefinicióban

- milyen operanduszokon,
- milyen müveletek

végrehajtását jelölhetjük ki. Amennyiben az utóbbiak olyanok, hogy operanduszaikként a tárgyprogram elemei /utasitások stb./ is szerepelhetnek és emellett a müveletek köre is elég tág, bizonyos esetekben "korlátlan" fejlesztési lehetőséget biztositanak.

Nem kell ugyanis befejezettnek tekinteni a transzlációs tevékenységet a tárgyprogram kialakulásával, hiszen a futási idő alatt is folytathatjuk az általában többmenetes forditási eljárás [5] kiegészitésének tekintve a forrásprogramban a programozó által definiált lépéseket.

Ez lehetővé teszi, hogy akár az egész tárgyprogramot átformáljuk. Természetesen erre általában nincs szükség; elegendő a tárgyprogram egyes részeit, elemeit módositani. A szóban forgó müvelet segitségével, a meglévő jelölésrendszer felhasználása révén pl. olyan operációkat és operanduszokat /kategóriákat, fogalmakat, szintaktikus változókat/ vezethetünk be, amelyek eredetileg nem szerepeltek a forrásnyelvben; azaz kibővithetjük a nyelv objektumainak körét és egyes kifejezések jelentését, anélkül, hogy meg kellene változtatnunk az objektumokra való hivatkozási formákat, az irásmódot vagy a transzlátort.

A forditás folytathatóságához azonban még nem elegendő, ha a makroutasitások a tárgyprogram utasitásaihoz, cimeihez való hozzáférhetőséget, a müveletek tekintetében pedig univerzalitást biztositanak. Ezenkivül meg kell tudnunk találni a forrásprogramban szereplő azon müveleteknek, operanduszoknak stb. a tárgyprogramban megfelelő müveleti kódokat, hivatkozási cimeket, utasitásokat és utasitáscsoportokat, amelyeket a továbbforditás során másra kell kicserélni. Ez tulajdonképpen a nyelvre és a transzlátorra vonatkozó követelmény. A folytathatóságnak tehát szükséges feltétele, hogy a forrásprogram módositandó részének elemei, ill. azok gépi megfelelői egyértelmüen felismerhetők legyenek a tárgyprogramban. Feltételezve a müveletek univerzalitását, ez egyben elégséges feltétel is. Az alábbiakban részletesen foglalkozunk a felismerhetőség itt használt fogalmával.

1.2. Általában nem várhatjuk, hogy a forrásprogram karakterenként visszaállitható legyen a tárgyprogramból; más szóval nem várható a legtágabb lehetőség a továbbforditásra. A karakterenkénti visszaállitás nyilvánvalóan nem lehetséges már az azonositókra vonatkozóan sem, minthogy azoknak a transzlátor által előállitott transzformációja több-egyértelmü.

Továbbá: a programok felirásánál meg van engedve bizonyos redundancia is, tehát egyes információk "elvesznek" a fordítás folyamán /pl. az ALGOL 60-ban a commentek, az üres utasitások, az eljárások hivásánál a paraméter-elválasztó jelek, a redundáns zárójelek stb.; hasonlóképpen az AKI-ban a commentek, a zárójelek, az értékadó utasitások sorozatában a COMPUTE-ok; pl. a

... COMPUTE A1=K1* COMPUTE A2=K2* ...

sorozat -- ahol a * az utasitás-végjel -- mindig helyettesithető

... COMPUTE A1=K1 A2=K2*...

sorozattal, ahol A1, A2 változókat, K1, K2 pedig kifejezéseket jelölnek/. A fentiek mellett még számos olyan példát lehetne felsorolni, melyek arra késztetnek, hogy más uton közelitsük meg e problémát.

1.3. Az utófordítás és a felismerhetőség kérdésével -- minthogy a forrásprogramok karakterenkénti visszaállitása általában nem lehetséges -- nem a program egészére, hanem egyes elemeire vonatkozóan kell foglalkoznunk. Továbbá, meg kell vizsgálnunk, hogy milyen programokkal lehetséges ezeket a tevékenységeket elvégeztetni, és hogy ezeknek milyen feltételeket kell kielégiteniök.

Tekintettel arra, hogy az utófordítást a forrásprogram makroutasitásaival kivánjuk megoldani, a megfelelő programnak a tárgyprogram részének kell lennie. Azt is meg kell követelnünk, hogy ez a program -- a forrásprogramtól függetlenül -- egyértelmüen meghatározott legyen, ellenkező esetben ugyanis minden ujabb forrásprogram uj makrodefiniciót követelne.

Az utóforditási tevékenység célját sem teljes általánosságban fogjuk megvizsgálni, csupán egy adott szintaktikus kategória jelentésváltozására korlátozottan.

Az utófordítást végző program két alapvető részre tagolható.

Az első rész a FELISMERŐ. A FELISMERŐ feladata azoknak a tárgyprogrambeli elemeknek a felkutatása és kijelölése, amelyek módositásra szorulnak abból a célból, hogy a megfelelő forrásprogram-elemek a kivánt jelentést kapják. Ez utóbbi feladatot az utófordităst végző program második alapvető része, az un. MÓDOSITÓ látja el. A módositást nem vizsgáljuk általánosan, minthogy erősen függ a mindenkori megoldandó feladattól. A felismeréssel viszont részletesebben foglalkozunk.

Ha ugyanis a felismerési feladat megoldható, a felkutatott jelsorozatok nyilvánvalóan megváltoztathatók. Ezután a kérdés csupán arra redukálódik, hogy milyen módositással érhető el a kivánt jelentés.

1.4. Tekintsünk egy s_f forrásnyelvi szintaktikus kategóriát. Jelölje

$$s_f^1, \ldots, s_f^k$$

ennek a kategóriának az F forrásprogrambeli előfordulásait, /a megjelenési alaktól eltekintve/. A fordităs folyamán a fordităprogram az i-edik előforduláshoz hozzárendeli az s_t^i tárgynyelvi szintaktikus kategóriát /ez lehet az "üres" kategória is, pl. kommentárok esetében/. A fordităprogram az s_t^i -hez általában az F-től függően adja meg azokat a különböző C_j^i tárgynyelvi jelsorozatokat /$j=1,\ldots,m_i$/, amelyeket n_{ij} számu /$n_{ij} \geq 1$/ helyen kell beépiteni a T tárgyprogramba. Legyen C_{jh}^i a C_j^i jelsorozat h-adikként beépitett példánya /$1 \leq h \leq n_{ij}$/. Jelölje g^i a C_{jh}^i elemek /$i=1,\ldots,k$; $j=1,\ldots,m_i$; $h=1,\ldots,n_{ij}$/, $\bar{g}^i$ pedig a $C_1^i, \ldots, C_{m_i}^i$ elemek halmazát. A $C_{j1}^i, \ldots, C_{jn_{ij}}^i$ jelsorozatok alakra megegyeznek C_j^i-vel, egymástól csak a beépités helyében különböznek. Az s_t^i kategória különböző alternativáinak a $\bar{g}^i$ részhalmazai felelnek meg. Ezek a részhalmazok azokat a különböző jelsorozatokat tartalmazzák, amilyen formában beépitettük az egyes alternativákat.

Ezt a megfeleltetést nem részletezzük a további hivatkozásoknál. Az "s_t^i egy C_j^i alaku előfordulása" kifejezést fogjuk használni és ezt ugy értjük, hogy "s_t^i valamely alternatívájának /alternatíváinak/ -- az emlitett megfeleltetésben megengedett -- egy C_j^i alaku előfordulása". Ezt a kifejezést $s_t^i \sim C_j^i$-vel fogjuk jelölni.

Az s_f^i képe a tárgyprogramban -- a bevezetett jelölések mellett -- g^i lesz. Tehát a tárgyprogram-beli kép felismerése g^i meghatározásával ekvivalens.

Ezután tételezzük fel, hogy a forrásprogramok egy halmazához tartozó bármely F forrásprogramban valamely s_f-re teljesülnek a következő feltételek: /$F \in \varphi$ /:

a/ s_f bármely előfordulása megjelenik a tárgyprogramban is, azaz $g^i \neq \emptyset$ /i=1,...k/;

b/ s_t valamennyi forrásprogrambeli előfordulásához létezik a tárgyprogramnak olyan z^i szelete, amely g^i összes elemét tartalmazza, és minden benne előforduló C_j^i alaku, s_t^i-hez tartozó jelsorozat hozzátartozik az s_f^i képéhez, vagyis eleme g^i-nek:

$$\forall i \, (\exists z^i \, ((g^i \prec z^i) \wedge ((C_j^i \sim s_t^i \, \varepsilon \, z^i \mid 1 \leq j \leq m_i) \Rightarrow (C_j^i \in g^i)))) .$$

/$g^i \prec z^i$ itt azt jelöli, hogy z^i részsorozatként tartalmazza a g^i elemeit, $C_j^i \sim s_t^i \, \varepsilon \, z^i$ pedig azt, hogy a $C_j^i \sim s_t^i$ részsorozata z^i-nek./

Ha e feltételek teljesülése mellett a FELISMERŐ program tartalmaz olyan P eljárást, amely bármely $F \in \varphi$-re meg tudja adni k-t, g^i-t, s_t^i-t és z^i-t /i=1,...,k/, a közöttük lévő megfeleltetéssel együtt, azaz

c/ $\exists$(P∈FELISMERŐ){ P (T) = (k, $\bar{g}^1$, s_t^1, z^1, ..., $\bar{g}^k$, s_t^k, z^k)}

akkor az alábbi lépésekből álló KERESŐ algoritmus alkalmas az s_f^1, ..., s_f^k szintaktikus egységek egyedenkénti felismerésére a φ -hez tartozó forrásprogramokban:

1/ P (T) végrehajtása,

2/ az s_t^i szintaktikus kategóriához tartozó C_j^i alaku jelsorozatok meghatározása z^i-ben $\bar{g}^i$ összes C_j^i elemére az i = 1,...,k értékek mellett.

A második lépés minden i-re pontosan g^i-t szolgáltatja. A b/ feltétel folytán ugyanis z^i-ben azok és csak azok a $C_j^i \sim s_t^i$ jelsorozatok szerepelnek, amelyek g^i-hez tartoznak és $\bar{g}^i$ a g^i-t alkotó különböző jelsorozatot tartalmazza.

A FELISMERŐ eljárásban a legnehezebb problémát a P eljárás megadása jelenti. A programoknak egy szük halmazánál viszonylag sok szintaktikus kategóriára lehet a feltételeknek eleget tevő P eljárást adni, de ezek a programok gyakorlatilag használhatatlanok lesznek, minthogy sok megkötést kell rájuk tennünk. Mintegy ellentétként a felismerendő szintaktikus kategóriákra is tehetünk megkötéseket, ami a programoknak tágabb halmazában biztositja a felismerési lehetőséget.

Tételezzük fel, hogy a P eljárás csak a $z = \bigcup_{i=1}^{k} z_i$ halmaz előállitására alkalmas, a z^i-k előállitására viszont nem. Vizsgáljuk meg, hogy az a/ és b/ feltételek mellett milyen mértékben biztositható a felismerés, megfelelően módositva a KERESŐ eljárás második lépését.

Nyilvánvaló, hogy z-re teljesül

$$g^i \prec z \qquad /i=1,...,k/,$$

ami biztositja, hogy az s_f összes előfordulásának képe beleessék a tárgyprogram vizsgálatába bevont szeletébe.

z^i-nek z-vel való helyettesitése révén azonban általában nem teljesül a b/ feltételben szereplő konjunkció második tagja. Ezért nem alkalmazható a KERESŐ eljárás második lépése sem.

Vizsgáljuk meg, hogy milyen feltétel teljesül az emlitett konjunkciós tag helyett. Állitsuk elő a $\bar{g}^i$, s_t^i párokból az alternativ szintaktikus egységek és jelsorozatok összes megengedett párositásait /i=1,...,k/. Képezzük -- valamennyi i-re -- az ily módon előálló halmazok metszetét, és tételezzük fel, hogy ez nem üres. Ehhez a következő módon jutunk el. Jelölje S_t^i az s_t^i szintaktikus kategória különböző alternativáinak halmazát. Képezzük a $\bar{g}^i \times S_t^i$ szorzathalmazt. E szorzathalmaz elemei olyan párok lesznek, melyeknek első tagja egy jelsorozat, második pedig s_t^i egyik alternativája. Ezen jelsorozat - alternativa párositások közül nem mindegyik megengedett a korábbi értelemben, ezért a megengedettek a $\bar{g}^i \times S_t^i$ szorzathalmaz egy G^i részhalmazát alkotják. Képezzük ezután a G^i /i=1,...,k/ halmazok metszetét és jelöljük G-vel. Feltételezzük, hogy ez nem üres:

$$G = \bigcap_{i=1}^{k} G^i \neq \emptyset .$$

Jelöljük a G elemeit alkotó párok első komponenseiben előforduló különböző jelsorozatokat C_r-rel, /r=1,...,m/, ezek halmazát $\bar{g}$-sal; a második komponenst képező szintaktikus egységeket pedig összefoglalóan s_t-vel. Az igy kapott $\bar{g}$ C_r elemeire és s_t-re a $C_r \sim s_t$ ugyanugy értelmezhető, mint a $\bar{g}^i$ elemeire és s_t^i-re /i=1,...,k/. Emellett G-ben pontosan azok a párok szerepelnek, amelyek megadják s_t megengedett alaku előfordulásait.

A G konstrukciójából adódóan fennáll:

$$\bar{g} \subseteq \bigcap_{i=1}^{k} \bar{g}^i ,$$

amit a következő módon láthatunk be. A $\bar{g}$ bármely C_r eleméhez /r=1,...,m/ megadható olyan eleme G-nek, és ebből adódóan minden G^i-nek is /i=1,...,k/, melynek első komponense megegyezik C_j-vel. Mivel G^i részhalmaza $\bar{g}^i \times S_t^i$-nek, ezért C_r megegyezik $\bar{g}^i$ valamely $C_{j_i}^i$ elemével /i=1,...,k/ és igy eleme a $\bigcap_{i=1}^{k} \bar{g}^i$ metszetnek is, tehát

$$C_r \in \bar{g} \Rightarrow C_r \in \bigcap_{i=1}^{k} \bar{g}^i \quad ,$$

ami igazolja állitásunkat.

Ugyancsak az előzőkből látható be -- feltételezve b/-t --, hogy $C_r \varepsilon z^i$ fennáll minden i-re és r-re /i=1,...,k; r=1,...,m/.

Tekintettel arra, hogy s_t szükebb, vagy ugyanolyan tág szintaktikus kategória, mint bármely s_t^i /i=1,...,k/, ha egy jelsorozat s_t-nek C_r alaku előfordulása, akkor ez C_r alaku előfordulása minden s_t^i-nek is. Ezért

$$C_r \sim s_t \varepsilon z^i \Rightarrow C_r \sim s_t^i \varepsilon z^i \quad . \ /r=1,...,m;\ i=1,...,k/$$

Mivel b/ alapján

$$\forall i \{ (C_j^i \sim s_t^i \in z^i \mid 1 \leqq j \leqq m_i) \Rightarrow (C_j^i \in g^i) \} \quad ,$$

és minden r-hez van olyan $1 \leqq r_i \leqq m_i$, melyre

$$C_r = C_{r_i}^i \quad , \quad /r=1,...,m;\ i=1,...,k/$$

igy

$$\forall i \{ (C_r \sim s_t^i \varepsilon z^i \mid 1 \leqq r \leqq m) \Rightarrow (C_r \in g^i) \}$$

is teljesül. Ekkor $C_r \sim s_t \varepsilon z^i \Rightarrow C_r \sim s_t^i \varepsilon z^i$ miatt fennáll

$$\forall i \{ (C_r \sim s_t \varepsilon z^i \mid 1 \leqq r \leqq m) \Rightarrow (C_r \in g^i) \} \quad .$$

Másrészről

$$(C_j \varepsilon z) \Leftrightarrow (C_j \varepsilon z^1) \vee \ldots \vee (C_j \varepsilon z^k),$$

igy azt kapjuk, hogy

$$(C_j \sim s_t \varepsilon z \mid 1 \leqq j \leqq m) \Rightarrow \{(C_j \in g^1) \vee \ldots \vee (C_j \in g^k)\},$$

amit ugy is mondhatunk, hogy

$$b'/\ (C_j \sim s_t \varepsilon z \mid 1 \leqq j \leqq m) \Rightarrow \exists i\ (C_j \in g^i)\ .$$

Szavakban megfogalmazva, azt kaptuk eredményként, hogy bármely z-beli $C_j \sim s_t$-hez van olyan g^i, amelynek a C_j eleme lesz. Ez az eredmény csak azt mondja ki, hogy van olyan g^i, amelynek $C_j \sim s_t$ eleme, de nem teszi lehetővé, hogy a g^i halmazokat egyedenként meghatározzuk, sőt arra sincs lehetőség, hogy a g^i-k egyesitését megadjuk, hiszen a vizsgálatokat mind a jelsorozatok, mind a tárgynyelvi szintaktikus kategóriák tekintetében leszükitettük.

Amennyiben a KERESŐ eljárás analógiájára, a b'/ felhasználásával, felépitünk egy KERESŐ1 eljárást, egyéb kiegészitő feltételek nélkül ez is csak részleges felismerésre lesz képes. Tekintettel arra, hogy -- amennyiben csupán z meghatározására alkalmas a P eljárás -- nem szükséges k, $\bar{g}^i$ és s_t^i ismerete, ezt nem is követeljük meg. A c/ pontbeli követelményt tehát a következővel helyettesithetjük:

$$c'/\quad \exists (P' \in \text{FELISMERŐ})\ \{P'(T) = (\bar{g}, s_t, z)\}\ .$$

Ekkor a KERESŐ1 algoritmusa a következő lesz:

1'/ P'(T) végrehajtása,

2'/ a $C_j \sim s_t$ jelsorozatok meghatározása z-ben $\bar{g}$ összes C_j elemére.

Ez az eljárás csak akkor tartalmazza eredményében egy s_f^i szintaktikus egység képét /g^i-t/ teljes egészében, ha $\bar{g}^i = \bar{g}$ és $s_t^i = s_t$ teljesülnek.

Ezen előfordulások felismerése megoldottnak tekinthető a KERESŐ1 felhasználásával, ha az utófordítás során közöttük nem kívánunk különbséget tenni. Ha ez s_f egy előfordulására nem áll fenn, akkor annak tárgyprogrambeli képét az eljárás csak részlegesen adja meg. Ezekre vonatkozóan tehát nem jutottunk kielégitő megoldáshoz.

Tekintsük most azt az esetet, amikor $\bar{g}$ és s_t üresnek adódik a metszetképzés során, de vannak olyan

$$s_f^{i1},\ s_f^{i2},\ \ldots,\ s_f^{ip} \qquad /p < k/$$

forrásprogrambeli előfordulásai s_f-nek /esetleg csak egy/, amelyekre a metszet /$\bar{g}'$ és s_t'/ már nem üres. Ekkor létezik legalább egy olyan s_f^{io}, amelyhez tartozó g^{io}-nak nincs $C_j' \sim s_t'$ alaku eleme z-ben. Tételezzük fel, hogy

$$\bar{g}^{iq} = \bar{g}',$$

és $\qquad /q=1,\ldots,p/ \qquad /1/$

$$s_t^{iq} = s_t',$$

valamint azt, hogy $s_f^{i1},\ldots,s_f^{ip}$-hez bármely más s_f^i-t hozzávéve már üres halmazhoz jutnánk. Amennyiben a FELISMERŐ tartalmaz olyan P eljárást, amely lehetőséget ad z mellett $\bar{g}'$ és s_t' megállapitására is, a KERESŐ1 eljárással pontosan az i1,i2,...ip indexü forrásprogram-elemek képét kapjuk meg.

Igy esetenként lehetőség van arra, hogy több ilyen $s_j^{i1}, s_j^{i2},\ldots, s_j^{ir}$ csoportot is kialakithassunk /P1. skalárváltozók esetében azonositóként/ és az utolsónak ismertetett módon járjunk el. A csoportonkénti felismerést követően a kapott eredményeket -- szükség esetén -- egyesithetjük.

Ha feladatunkat valamely forrásprogrambeli szintaktikus egység valamilyen A karaktersorozatból álló előfordulásainak felismerésére korlátozzuk, akkor a megkövetelt feltételek nem tulzottan szigoruak. Ekkor ugyanis a tárgyprogrambeli képek többnyire azonos elemekből állnak. Ha pedig ugyanazon szintaktikus egységhez a forrásprogramban különböző jelsorozatok tartoznak /A, ill. B/, akkor -- az esetek többségében --

$$\bar{g}'_A \cap \bar{g}'_B = 0$$

áll fenn, ami elegendő feltétel ahhoz, hogy a $\bar{g}$ és s_t üres legyen. Tehát az A alaku sorozatok képei megkülönböztethetők a B alaku sorozatok képeitől.

Az utóforditásra használt makroutasitás paraméter-listája sokszor célszerüen felhasználható a gyakorlatban a P eljárás feladatának megkönnyitésére. Ha ugyanis az s_f szintaktikus egység A alaku előfordulásának képei kielégitik az /1/ feltételt és van rá lehetőség, hogy A s_j-ként, mint aktuális paraméter szerepelhessen a makrohivásban, akkor a makroutasitás paraméter-listájából közvetlenül meghatározható mind $\bar{g}'$, mind s'_t . Nem foglalkozunk a z kijelölésével és szükitési lehetőségeivel, minthogy a gyakorlatban ez nehezen oldható meg. Két ut látszik célszerünek: vagy a teljes tárgyprogramot /az utasitásszegmentumot/, vagy annak egy -- cimkétől cimkéig -- kijelölt részét tekintjük z-nek, attól függően, hogy miképpen teljesülnek a felismerhetőség feltételei. Az utóbbi megoldásnál ugyancsak a makrohivás paraméter-listája adhat jelentős segitséget.

Előfordulnak olyan feladatok is, amelyeknél a makroutasitás konstansként tartalmazhatja a felismeréshez szükséges információkat. Ilyenkor a P eljárás funkciója megszünik, ill. a konstans kiválasztására korlátozódik.

Amennyiben az információk megtalálhatók a paraméter-listában, a makroutasitásnak olyannak kell lennie, hogy müveleti operanduszként szerepelhessen paraméter-listája is. Ugyancsak operanduszként kell tudnia kezelni a tárgyprogram elemeit is.

2. A gépi és a magasabb szintü programozás általános jellemzői, a makroutasitások feladatai

Későbbi céljaink érdekében előzetesen a gépi és a magasabb szintü programozás néhány jellemzőjével kell foglalkoznunk, hogy rávilágithassunk a magasabb szintü nyelvek makroutasitásainak jelentőségére, valamint körülirhassuk a segitségükkel megoldható feladatokat.

A gépi szintü programozás egyik legfontosabb sajátossága, hogy a programbeli utasitások operanduszoknak tekinthetők, ami annyit jelent, hogy müveleteket tudunk rajtuk végezni. Ennek a következményeként a programok a végrehajtás során módosithatják önmagukat, ami --a ciklusszervezés, a szubrutinok használata stb. révén --megteremti az automatikus programozás alapjait is. A szubrutin-technika kialakulásával a gépi müveletek köre gyakorlatilag tetszőlegesen kiterjeszthetővé vált. Ugyancsak a szubrutin-technika teremtett lehetőséget arra is, hogy igen tág értelemben kibővitsük a hivatkozható operanduszok körét /lásd később/.

Ugyanakkor a magasabb szintü nyelveknél a gépi müveletek és az operanduszok körének --makroutasitások nélkül történő-- kibővitése, vagy egyáltalán nem lehetséges, vagy csak rendkivül nehézkesen keresztül-vihető.* /Példaként hivatkozhatunk az ALGOL 60-nál az eljárások nélküli bővités nehézségeire./ Bizonyos tipusu makroutasitásokkal /pl. az ALGOL 60-ban az eljárásokkal/ lehetséges mind a gépi müveletek, mind az operanduszok körének korlátozott kiterjesztése /a

---

* Meg kell jegyeznünk, hogy nincs minden magasabb szintü nyelvben makroutasitás definiálva, sok esetben pedig csak igen speciális makrók használatát engedik meg.

lényeget tekintve éppen ez a feladatuk a magasabb szintü nyelvek forrásnyelvi makroutasitásainak/.

Az összetett operanduszok definiálásánál azonban ez korántsem triviális tevékenység. Makroutasitások nélkül a program módositására egyáltalán nincs lehetőség, sőt a módositáshoz már a forrásnyelvi makroutasitások sem elegendők, ezért más jellegüek bevezetéséről kell gondoskodnunk. Ez utóbbiak alkalmazásával visszanyerhetünk olyan előnyöket, amelyekkel a gépi programozás rendelkezik, de a magasabb szintü programozás eredetileg nem biztosit. A müveletek esetében ez igen egyszerü, az operanduszoknál kevésbé nyilvánvaló, de megoldható. Végeredményben azonban az önmódositás lehetőségét csakis speciális makroutasitásokkal tudjuk biztositani. Ezek felhasználásával olyan rendszerhez juthatunk, amelyben lehetséges poszttranszlációs forditási lépések végzése, és a felhasználó szempontjából kényelmesen oldhatók meg a kiterjesztési és önmódositási feladatok.

A magasabb szintü nyelveknél emlitett hiányosságok nyilvánvalóan ugy értendők, hogy a forrásnyelvben jelentkeznek. Mivel azonban a gépi programozás előnyeit a forditók messzemenően kihasználják, igy ezek áttételes formában fellépnek e nyelvek esetén is.

Az előzőekben kifejtettek alapján elvégezhető a makrók --felhasználási szempontok szerinti-- osztályozása, figyelembe véve funkcionális jellemzőiket és rendeltetésüket. Az alábbi három csoportot különitjük el:

1/ Müveletek /operátorok/ körének kiterjesztésére szolgáló makroutasitások. /Ebben az esetben kikötjük, hogy az operanduszok halmaza nem változik. /Ezek is --mint az összes makrolehetőség-- a szintaxis kibővitéséhez vezetnek.

Az ALGOL 60 eljárásait többnyire ilyen célokra használják. A szóban forgó eljárások segitségével egy-egy program számára bővithetjük a szintaxist, mivel az eljárásokat /pl. a mátrixmüveleteket/ valamennyi program számára ujra definiálni, deklarálni kell. Bár ez bizonyos foku rugalmasságot biztosit, de egyben kényelmetlenséggel is jár, mivel a felhasználói programokban korábban alkalmazott és általánosabb érvényü eljárásokat is ujra kell irni. A PL/1, a FORTRAN és az assemblerek eljárásait, makrodefinicióit viszont már könyvtárban /az un. makrokönyvtárban/ is elhelyezhetjük, hozzáférhetővé téve azokat az összes felhasználó számára.

2/ Az operanduszok körének kiterjesztésére szolgáló makrók. Ezeknek természetesen csak az első csoporthoz tartozó makroutasitásokkal együtt van értelmük, mivel az uj operanduszok uj müveleteket is igényelhetnek. Egyes esetekben --pl. komplex változók bevezetésénél-- ez nem bonyolultabb, mint az uj müveletek bevezetése. Összetett strukturáju változók esetén azonban már lényegesen nagyobb nehézségekkel kell számolnunk, amelyek megoldása speciális /később ismertetendő/ makrokezelést követel. A második csoportba tartoznak pl. az alapnyelvben független változónevek, azonositók összekapcsolása, jelentésük megváltoztatása /megtöltésük uj tartalommal/ stb.

Az eddigi két csoport jól szemlélteti, hogy a makroutasitások alkalmazása valóban éppen azokat a tevékenységeket és lehetőségeket állitja vissza, amelyeket a magasabb szintü nyelvekre való áttérésnél a gépi kódu /vagy assembly szintü/ programozással szemben elveszitettünk.

3/ A harmadik csoportba azokat a makroutasitásokat soroljuk, amelyek segitségével maga a program is bevonható az operanduszok körébe. Ezek révén tehát magasabb szinten nyerhetjük vissza a programok önmódositási képességét.

Bár lényegében a második és a harmadik csoportot egyesithetnénk, hiszen mindkettő rendeltetése az operanduszok körének kiterjesztése, mégis célszerünek látszik most a különkezelésük, minthogy a programrészek, mint operanduszok, igen speciális esetet képeznek.

Megjegyezzük, hogy a korszerü rendszerekben mind több "tiltott operandusz" van a felhasználói programok szempontjából. /Pl. rendszerprogramok, táblázatok, védett file-ok stb./ A tiltás elengedhetetlen a többszörös hozzáférésü rendszerek megbizható müködéséhez. A tiltott részek a felhasználói programok számára praktikusan nem-létezőknek tekintendők; a kiterjesztés csak a felhasználói programokra, ill. általában a szóban forgó program hatáskörén belülre értendő. A fentiek ugyanis nemcsak a felhasználói programok makroutasitásaira vonatkoznak, hanem - természetesen - minden programéra is.

A továbbiakban a makroutasitás definiciója mellett részletesen foglalkozunk a makroutasitások osztályozásaival, hogy pontosan meghatározhassuk a céljainknak megfelelő tipusokat. Az 1. részben kitüzött feladat - a poszttranszlációs forditás - általános problémakörén belül elsősorban az operanduszok körének kibővitését biztositó lehetőségeket vizsgáljuk, ami természetesen szükségessé teszi az első csoport makróinak részletesebb áttekintését. A harmadik csoportbeli makrókkal csak vázlatosan foglalkozunk, rámutatva néhány olyan feladatra, amelyek megoldása jelentős előnyöket biztosithat, viszonylag kevés munkabefektetéssel.

3. A makroutasitás definiciója; makroutasitások osztályozása

3.1. Az előző részekben többször használtuk a makroutasitás kifejezését, anélkül azonban, hogy tartalmát pontosabban meghatároztuk volna. Az irodalomban több definiciót is találunk a makroutasitásra, amelyek többé-kevésbé fedik a gyakorlatban makroutasitásnak /vagy röviden makrónak/ nevezett programozási formákat. Nem célunk a definiciók részletes áttekintése, csupán néhányat emelünk ki közülük, kritikai összehasonlitás végett.

I.Flores [2] szerint a makroutasitás nyitott szubrutin, amely a makrofeldolgozás során épül be a programba. Ez a meghatározás azonban nyilvánvalóan szük, mivel kirekeszti az utasitássorozattal definiált operátor zárt szubrutinként való kezelésének lehetőségét. A szerző egyébként a makroutasitásoknak egy igen jól használható osztályozását is elvégezte, amely lényegesen általánosabb érvényü, mint maga a definició.

L.Bolliet [5] a makroutasitásokat konkrét példákkal mutatja be /ami ugyszintén az általános definició hiányáról tanuskodik/, megállapitva, hogy a makrók elsősorban assemblerekben használatosak, bár a későbbiekben olyanokat is ismertet, amelyek magasabb szintü nyelvekben /PL/1, ALGOL/ fordulnak elő. Végül azonban csak az assemblerekre határozza meg a fogalmat: "A pszeudo utasitások között találhatók a makroutasitások, amelyek szövegrészek dinamikus helyettesitését teszik lehetővé a forditás során." Nyilvánvaló azonban, hogy egyrészt ez a definició az assembly szintnél lényegesen szélesebb körre is vonatkozhat, másrészt - amint ezt később részleteiben is kimutatjuk - a helyettesitésnek nem kell feltétlenül a forditás idejére esnie.

Emellett Bolliet a konkrét példák között olyanokat is ismertet, amelyeknek célja kifejezetten uj változók és azokon végzett uj müveletek bevezetése /"overloading",szintaktikus makrók/.

Érdemes megemliteni Strachey [1] meghatározását is. Szerinte a következő két mozanatot kell figyelembe venni:

- makrodefinició és
- makrohivás.

Strachey-nél a makrodefinició alakja:

§ DEF, NÉV, SZÖVEG;

a makrohivásé pedig:

§ HIVÁSNÉV, ARGUMENTUM-LISTA; ,

ahol mind a hivásnév, mind az argumentumok lehetnek makrohivások. A makroszövegen belül is bárhol előfordulhat hivásnév; a szöveg egyébként tetszőleges lehet.

I.A.MacLeod [4] Brown [22] nyomán szintén megkülönbözteti a makrohivást és a makrodefiniciót. Az utóbbi makrónévből és helyettesitési szövegből áll. A makronév egyben a hivás szintaxisát is definiálja, a szöveg pedig azt a tárgykódot adja meg, amellyel a makrohivást helyettesiteni kell a makrofeldolgozás során. MacLeod definiciója lényegében ugyanazt a fogalmat takarja, mint az [5]-ben szereplő szintaktikus makro.

Az eddigi elemzésből közös vonásként kiemelhető, hogy a makroutasitásokkal, ill. használatukkal kapcsolatban három alapvető fogalmat találunk:

- a makrodefinició,
- a makrohivás és
- a makrofeldolgozás fogalmát.

A továbbiakban makrodefinición egy szöveget vagy szabályt megadó szintaktikus egységet értünk, mely azt határozza meg, hogy a makrofeldolgozás folyamán mivel kell helyettesiteni a makrohivást.

3.2. A következőkben a most megadott három alapfogalom segitségével elkészitjük a makroutasitások több szempont szerinti osztályozását, ami meg fogja könnyiteni az igen elterjedten alkalmazott, sokféle makroutasitás áttekintését.

3.2.1. Megkülönböztetünk a makrodefinició helye szerint

- programban,
- programkönyvtárban, valamint
- forditóban /processzorban: PLP [21] /

definiált makrókat.

A programban definiált makrókat a programozó a forrásnyelvi programban adja meg, és azon belül is használja fel. Ez lehetővé teszi valamennyi felhasználó számára az igénybe vett forrásnyelv keretein belüli uj makrók definiálását. Ezáltal lényegesen könnyebbé tehető egy-egy programra vonatkozóan a programozási munka.

A programkönyvtárban definiált makrókat az egy vagy több fordító /PLP/ által elérhető programkönyvtár /makrokönyvtár/ tartalmazza. Ezeket természetesen még a tárgyprogramok szerkesztése előtt kell megadni és hozzáférhetővé tenni. A makrók szóban forgó csoportja több programban és esetleg több forrásnyelven is aktivizálható.

A forditóban definiált makrók a forditóprogram könyvtárában vannak elhelyezve. Világos, hogy definiciójuknak időben meg kell előznie a transzlációt, hiszen feldolgozásuk a forditás folyamán történik. Emiatt számos különbség is jelentkezik a makrók előző csoportjával szemben, annak ellenére, hogy azok is --bizonyos értelemben-- "könyvtári" makrotipust képeznek. Lényeges különbség pl. az, hogy az utóbbiak csak egy forrásnyelv számára hozzáférhetők.

3.2.2. Osztályozhatók a makroutasitások a definició nyelve szerint. Számunkra igen fontos, hogy a definició nyelve milyen operanduszokon milyen müveleteket enged meg. Minél közelebb áll a nyelv a gépi szinthez, annál általánosabb a kezelhető operanduszok köre, viszont -- az alacsonyabb nyelvi és kifejezési szintnek megfelelően -- nehézkesebb az operanduszok kezelése. Pl. assembly nyelvekben is kezelhetünk mágnesszalag file-okat, mint egységes egészeket /másolás/, de ennek a feladatnak lényegesen egyszerübb a megoldása egy könyvtárkezelő rendszer másoló-utasitásával /bár ennek operandusza csak file lehet/.

A szóban forgó osztályozást nem az egyes konkrét nyelvek szerint végezzük, hanem annak megfelelően, hogy a definició nyelve milyen szintet képvisel. Ennek alapján

- gépi-,
- assembly- és
- magasabb szintü eljárás-orientált nyelven

megadott definiciókat különböztetünk meg. /Az "eljárás-orientált nyelv" kifejezést a [21]-ben meghatározott értelemben használjuk./ Megjegyezzük, hogy a programban definiált makrók esetében általában nem szoktak a forrásprogram nyelv-szintjénél magasabb szintü nyelvet használni. A forditók makrói természetesen alacsonyabb szintüek, mig a könyvtári makróknál az összes -- természetesen az adott rendszerben használatos -- szint megengedett.

Régebben elsősorban az alacsonyabb szintü nyelvekben alkalmaztak makroutasitásokat, de jelenleg mind jobban terjed a magasabb szintü nyelvekre épitett makroprocesszorok használata is. Az utóbbiak főleg a probléma-orientált rendszerek [23] kialakitásánál jelentősek, mivel lehetővé teszik, hogy megszabaduljanak az eljárás-orientált nyelvek merev formáitól.

3.2.3. A makroutasitásoknak -- a definiciók felhasználásával történő -- beépitése /incorporation [2] , macrogeneration [1]/, a forrásprogram feldolgozásainak különböző fázisaiban lehetséges. Ennek alapján

- forditási idő előtt,
- forditási idő alatt,
- szerkesztéskor,
- betöltéskor /load time/ és
- futási idő alatt

beépitett makroutasitásokat különböztetünk meg. Különálló makrofeldolgozásról, makroprocesszorról /vagy ahogyan egyes helyeken, pl. [1]-ben nevezik: "makrogenerátor"-ról/ csak az első esetben beszélünk /pl. MP/1 [4] /.

Természetesen nincs lehetőség az összes makrotipusnak minden időben történő feldolgozására. Nagyrészt a makrofeldolgozó határozza meg, hogy melyik tipus beépitése mikorra esik.

3.2.4. Mint már a 3.1.-ben emlitettük egyes szerzők definiciója szerint a makroutasitás nyitott szubrutin. Ugyanakkor rámutattunk arra is, hogy nemcsak nyitott, hanem zárt szubrutin formájában kezelt makroutasitások is vannak. Ez a megkülönböztetés mégis igen hasznosnak mutatkozik az osztályozásnál. A makrók ugyanis kezelhetők

- dinamikus vagy
- statikus módon.

A makrodefiniciókat az előbbi módban zárt, az utóbbiban pedig nyitott szubrutin formájában használjuk fel. A dinamikus kezelésre példaként az ALGOL 60 eljárásait, a statikusra az MP/1 makroutasitásait emlithetjük [4] .

Amennyiben statikus makrokezelésről van szó, a makroutasitások feldolgozásakor a hivások helyére a definicióban megadott jelsorozatok kerülnek, a megfelelő paraméter-helyettesités elvégzése után. Dinamikus kezelésnél a feldolgozás két lépésben valósul meg. Az elsőben végbemegy a hivás áttranszformálása olyan alakra, amelyet a program feldolgozásának következő lépcsője /pl. a szerkesztőprogram/ már értelmezni tud, de továbbra is hivás jellege marad. Lényegében önállóan /gyakran időben is elkülönülve/, másik feldolgozási lépcsőben valósul meg a definiciók alapján a megfelelő szövegrészek /zárt szubrutin formáju/ beépitése.

3.2.5. Az eddig tárgyalt osztályozások mellett szokás megkülönböztetni fix és változó makroutasitásokat. Az utóbbiak a beépitési folyamatban és aktuális paraméter-listáktól függően változnak, s emiatt más és más alakban épitődnek be. A változás két helyen jelentkezhet: vagy a hivásban /pl.változó számu paraméter dinamikus kezelésénél/, vagy a hivásnak megfelelő helyettesités után beépitett makroszövegben. Az első változattal gyakran találkozunk /pl. az INZSENYER könyvtári programok hivásánál/, a másodikat -- tekintettel a kapcsolatos nehézségekre -- általában nem használják.

3.3. Most ismét áttekintjük a makroutasitások felhasználás szerinti osztályozását, miközben ujabbakkal egészitjük ki a 2. részben tárgyalt szempontokat.

3.3.1. A makroutasitások legegyszerübb alkalmazása az, ha valamely adott nyelv meglévő operanduszain olyan uj müveleteket definiálunk, amelyeket a nyelv eredetileg nem tartalmazott. Ebben az esetben a makróban leirt tevékenységek nem függenek sem a tárgy-, sem a célprogramtól /nem hivatkoznak a tárgyprogram elemeire, mint operanduszokra/.

3.3.2. Bonyolultabb feladat a nyelvhez tartozó operanduszok körének kibővitése. Itt részben deklarációs feladatok megoldásáról, részben pedig olyan müveletek bevezetéséről van szó, amelyek segitségével az uj operanduszok kezelhetők.

Az itt fellépő tevékenységeket már nem lehet olyan makroutasitásokkal elvégezni, amelyek ne hivatkoznának a tárgyprogram valamely elemére /mint operanduszra/. A hivatkozásoknál egy forrásprogramban megadott információt /a deklarációt/ kell felhasználni, vagyis a makróknak meg kell keresniök a tárgyprogramban azoknak az információknak a képeit, amelyek alapján figyelembe vehető a deklaráció. Ennélfogva a tárgyprogram elemei is operanduszokká válnak. A müveletek azonban nem a forrásprogram egységes egészként tekintett utasitásait, hanem csupán azok bizonyos elemeinek tárgyprogrambeli képét érintik, mivel a makróban kijelölt tevékenységek -- forrásnyelvi szinten tekintve -- nem a programra vonatkoznak. Hogy a tárgyprogramot felhasználjuk /esetleg módositjuk/, csupán technikai szükségmegoldás.

3.3.3. Bizonyos esetekben előnyös lehet magán a forrásprogramon is müveleteket végezni, a benne hivatkozott makrók segitségével. Ez azt jelenti, hogy a forrásnyelvi program -- bizonyos korlátok között -- önmagát módositja. Ezáltal megváltoztathatjuk pl. bizonyos müveletek értelmezését /v.ö. egy helyett többszavas pontosságu aritmetika/. Ilyenkor tulajdonképpen valamely utasitáscsoport szerepel operanduszként és ez kap uj értéket a makro végrehajtása során. További példaként emlithetjük a forrásnyelvi nyomozást. Makroutasitások segitségével meghatározott utasitáscsoportok végrehajtásának elemzése kérhető, ami /ha nem áll rendelkezésre kiegészitő információ/ inverz forditást igényel. Ezt a végrehajtás alatti forditásfolytatás egyik speciális esetének tekinthetjük.

Megemlitjük még, hogy makrók felhasználásával lehetővé válik teljesen uj változótipus bevezetése is, nevezetesen olyan változóé, melynek tipusa "utasitás" [12] . Ez az utasitástipus egyebek között lehetővé teszi, hogy egyszerübb formában irjunk fel logikailag bonyolult strukturáju programokat.

Az emlitett felhasználásoknál forrásprogram-egységek szerepelnek operanduszokként a makroutasitásokban előirt tevékenységek során, tehát az operanduszok köre kiterjesztődött magára a programra is.

A következő fejezetben részletesen foglalkozunk a funkcionális osztályozás második pontjában leirt makrókkal, ill. feladatokkal, számitásba véve a megvalósitással kapcsolatos technikai kérdéseket is.

4. Uj operanduszok bevezetése

Előzetesen az operanduszok két fő tipusát különböztetjük meg:

a/ az elsőhöz az olyan operanduszokat, változókat soroljuk, amelyeknek strukturája nem bonyolultabb, mint a nyelvben megengedett változóké, de más a jellegük /pl. logikai változó bevezetése, ha a nyelv tartalmaz aritmetikai -- egész vagy valós -- változót/;

b/ a másodikhoz bonyolultabb strukturáju változók tartoznak, amelyek elsősorban szerkezetük révén különböznek a többiektől.

Az emlitett operandusz-tipusok természetesen többnyire nem tiszta formában jelennek meg; egy második tipushoz tartozó változó pl. elemi egységként tartalmazhat első tipust is. Ennek ellenére is érdemes különbséget tenni közöttük. Az a/ tipusnál ugyanis a deklaráció és a hivatkozás eszközeiként felhasználhatók a meglévő formák, s csupán a müveletekben való kezelés igényel uj elemeket. A b/ tipusnál azonban a deklaráció és a hivatkozás már nem oldható meg a változókra vonatkozó szokásos kereteken belül, tehát nemcsak a kezelés igényel uj módszereket.

A továbbiakban az operanduszok két fajtája -- ti. a konstans, ill. a változó -- közül csak az utóbbival foglalkozunk, mivel a belső kezelés, hivatkozás szempontjából ennek van döntő szerepe.

Az uj operanduszok bevezetésével kapcsolatos kérdések tárgyalása előtt kisérletet teszünk a változó fogalmának meghatározására, és elemezni fogjuk a deklarációs, valamint a hivatkozási tevékenységet.

4.1. A változó-fogalom megközelitésének két alapvető szempontja van: a gépi reprezentáció és müveletvégrehajtás, illetőleg a formális nyelv.

Az elektronikus számitógépekben az információ legelemibb tárolási egysége a bit. Meg kell azonban jegyeznünk, hogy már a legkisebb cimezhető egységek is /szó, byte, karakter/ több bitet tartalmaznak. Egy-egy gépi szintü müvelettel információ helyezhető el a cimezhető egységekben, vagy megváltoztatható azok tartalma. Más szavakkal: a cimezhető tárolóegységekhez /szó stb./ értékeket rendelhetünk, mégpedig oly módon, hogy ezek az értékek változhatnak, átfuthatnak egy bizonyos tartományon. Ezeket a cimezhető tárolóegységeket tekintjük változóknak. Jelölésükre cimüket használjuk, a tartomány pedig, amin átfuthatnak, a bennük tárolható információk /bitkombinációk/ halmaza /részhalmaza/ lesz.

A számitógépek és a programozási technika fejlődése során hamarosan világossá vált, hogy a változóknak ez az értelmezése tulságosan szük és kényelmetlen a kitüzött feladatok megoldása szempontjából. Hasonlóan ahhoz, ahogyan a cimezhető egységeket bitek összekapcsolásával nyerték, célszerünek látszott több cimezhető egységet egyetlen hivatkozási egységgé összekapcsolni. Egyes számitógépeknél az összekapcsolás -- bizonyos esetekben -- hardware uton is megtörténhet /pl. byte - félszó - szó - dupla szó az IBM gépeknél/. Más esetekben külön utasitások alkalmasak annak megadására: hány karaktert kell egy egységnek tekinteni /pl. ilyenek a MINSZK-23 egyes utasitásai/. A MINSZK-23-on található olyan megoldás is, amelynél a karakterek egy speciális bitje /vagy bitcsoportja/ határozza meg az egy egységként kezelendő folytonos memóriaterületet.

Az emlitett hardware-segitség azonban nem jelent teljes megoldást a programozás szempontjából, mivel nem biztosit eléggé flexibilis és kényelmes keretet.

Ezzel szemben a software-eszközök alkalmasnak bizonyultak, hogy tetszőleges memóriaterületet jelöljünk ki és kapcsoljunk össze segitségükkel egyetlen hivatkozási egységgé. Gépi oldalról tekintve a kombinált módszer biztositja a legáltalánosabb lehetőségeket.

A továbbiakban gépi változónak fogjuk nevezni az egyetlen hivatkozási egységgé összekapcsolt memóriaterületeket. Itt természetesen már nem használhatjuk közvetlenül az egység "címét", mint jelölést; a változóra való hivatkozás bonyolultabbá válik, és általában software-eszközökkel valósul meg. Ezzel szemben a kiterjesztés lehetősége olyan mértékűvé bővül, hogy nem szükséges többé az operativ memóriához kapcsolni a változó fogalmát; a külső memóriaegységek elvileg egyenértékü szerepkört tölthetnek be az operativ tárral. /A hozzáférési idők jelentős eltérése miatt a gyakorlati alkalmazásokban természetesen célszerü különbséget tennünk közöttük, ami oly módon valósul meg, hogy más jellegü változókat realizálunk az operativ memória segitségével, mint a külső táréval./

A gépi változó meghatározásánál nem voltunk tekintettel arra, hogy korszerü számitógépek esetén a gépi szintü program /tárgyprogram/ előállitásakor csupán gépi szintü logikai cim-hozzárendelés történik; sőt -- szerkesztő-programokat használva -- mégcsak nem is egy lépésben [14]. A fizikai cim-hozzárendelés a lap- és szegmentum-cimzéses rendszereknél [14], a betöltés /load-time/ fázisában, ill. a futásidő /run-time/ alatt valósul meg statikus vagy dinamikus jelleggel. Egy konkrét program -- és a programozó -- szempontjából a gépi szintü logiai cimek játsszák a legfontosabb szerepet; a fizikai cimek másodlagosak a vizsgált tevékenység vonatkozásában. Emiatt a gépi változóról mondottak -- amennyiben cimekkel kapcsolatosak -- mindig gépi szintü logikai cimekre értendők.

Ha a formális nyelv oldaláról közelitünk, a változót olyan szimbólumnak kell tekintenünk, amelyhez érték rendelhető és ez az érték a program végrehajtása során változhat /lásd pl [13]/. A szimbólum /változó/ ebben az esetben kötött formáju jelsorozat, amelyhez egyértelmüen gépi változó rendelhető, emellett tartalmazza ez utóbbival való összekapcsoláshoz szükséges információkat. A továbbiakban ezt a jelsorozatot forrásnyelvi változónak nevezzük.

A forrásnyelvi és a gépi változó egymáshoz-rendelésénél két alapvető tevékenységet célszerü megkülönböztetni:

a/ először: meg kell adni egy olyan eljárást, amelynek segitségével a memóriaterület elérhető. Tekintettel arra, hogy egy programon belül kisebb eltérésekkel általában több változónál is lényegében ugyanazt az eljárást kell alkalmazni a gyakorlati munka során,az emlitett fázisban még egy megkülönböztető paraméter-listát is hozzárendelünk a változóhoz. Ezt a tevékenységet deklarációnak, a realizáló eljárást pedig deklarációs eljárásnak nevezzük.

b/ Másodszor: a változó értékének felhasználásánál vagy megváltoztatásánál aktivizálni kell a deklariációban megjelölt eljárást. Ehhez természetesen meg kell adni az eljárás végrehajtásához szükséges, esetleg még hiányzó paramétereket. Az eljárás aktivizálását a változóra való hivatkozásnak, az aktivizált eljárást pedig hivatkozási eljárásnak nevezzük. A hivatkozás a gyakorlat során valamilyen általános értelemben vett regiszterbe való töltést jelent, ill. onnan való eltárolást /esetleg bizonyos kiegészitő müveletekkel.

4.2. Mint az előző pontban megállapitottuk, a hozzárendelés két lépésben történik; az első lépésben -- amelyet a PLP végez -- a hivatkozási eljárások kijelölésére és a paraméterek egy részének rögzitésére kerül sor. Az ehhez szükséges információkat a forrásprogramnak kell tartalmaznia /most eltekintünk a tárfelosztásra vonatkozó információktól, amelyek a forditóban, ill. a rendszerben vannak adva/. Az előbbi információkat természetesen nem mindig direkt formában adjuk meg. Az eljárások kijelölhetők pl. szimbolikus nevekkel /ARRAY, RECORD stb./, egyes paramétereket /mint a tömbdeklarációk dimenziószámát/ pedig a deklaráció formája tartalmazza implicit módon.

A forrásnyelv szintjén a deklaráció határozza meg a forrásnyelvi változó jellegét, rögzitve ezzel egyrészt a hozzárendelhető információk halmazát, másrészt a megengedett hivatkozási formákat. A forditás során a PLP-hez tartozó deklarációs eljárás -- a forrásnyelvben meglévő információk alapján -- megadja a hivatkozáshoz szükséges algoritmust /specifikálja a beépitendő szubrutint, a hivatkozási eljárást/ és összeállit egy listát /az un.deklarációs listát/. Ennek első része a deklarációból nyert aktuális paramétereket, a második pedig az aktuális címtartomány meghatározásához szükséges információkat tartalmazza.

Bizonyos esetekben /pl. amikor a változók kezelése dinamikus/ a PLP csupán a forrásprogramban lévő információt adja át, és gondoskodik a fentebb emlitett tevékenységet /ill. annak egy részét/ végző eljárás /szubrutin/ beépitéséről.

A hozzárendelési folyamat második lépése a hivatkozás. A forrásprogramban található hivatkozás és a deklaráció alapján a PLP

- egy utasitássorozatot generál, amely aktivizálja a kijelölt hivatkozási eljárást és
- egy listát állit össze, amely a hivatkozás paramétereit tartalmazza /hivatkozási lista/.

A hivatkozásnál előbb az aktivizáló utasitássorozatot, majd -- a megadott lista alapján -- a kijelölt eljárást hajtjuk végre.

4.3. A fentiek alapján megállapitható, hogy mind a deklarációnál, mind a hivatkozásnál a döntő szerepet a forditás, ill. a futás idején /compile-time, ill. run-time/ aktivizált makroutasitásoknak tekinthető eljárások /algoritmusok/ játsszák. Világos, hogy valamennyi fázisban biztositanunk kell a müködésükhöz szükséges információkat. A továbbiakban -- minthogy lényegében makroutasitásokról van szó -- a hivatkozási-, ill. deklarációs eljárás kifejezések helyett a deklarációs makro /DM/, ill. a hivatkozási makro /HM/ kifejezéseket fogjuk használni.

Áttérünk annak vizsgálatára, hogy az egyes fázisokban milyen makroutasitások alkalmazására van szükség. A tipusok megállapitásánál a 3.2. pontban megadott szempontokat vesszük alapul.

4.3.1. A deklarációk két alapvető tipusát különböztetjük meg: a statikusan és a dinamikusan kezelt deklarációkat.

4.3.1.1.Tekintsük először a statikusan kezelt deklarációk esetét. Itt a felsorolt tevékenységeket általában a PLP-ben lévő eljárás /makro/ végzi, de ez nem szükségszerü megoldás. Csupán az a lényeges, hogy a szükséges eszközöket a deklarációs fázis az első hivatkozást megelőzően biztositsa. /Ez utóbbi természetesen nemcsak a statikusan, hanem a dinamikusan kezelt deklarációk esetére is érvényes./

a/ Célszerüségi szempontok azt diktálják, hogy compilerben vagy könyvtárban definiált makrókat alkalmazzunk, minthogy a felhasználók szempontjából sokkal inkább a kényelmetlenséget, mintsem a szabadságot fokozná, ha a deklarációs makrók definicióját a programban kellene megadni.

b/ Továbbá: annak ellenére, hogy a definició nyelve elvileg bármilyen eljárásra orientált nyelv lehet /esetleg magasabb szintü nyelv is/, a gyakorlatban mégis erre a célra inkább a gépi vagy assembly szintü nyelveket használjuk, mivel a deklarációs makrók, tevékenységük során, gépi szintü operanduszokat /gépi logikai cim, jelzőbit stb./ is kezelnek. A rájuk vonatkozó müveletek leirása pedig a legtöbb magasabb szintü /nem kifejezetten rendszerprogramok irására orientált/ nyelven lehetetlen, vagy legalábbis kényelmetlen.

c/ Végül fontos jellemzőként emlithetjük meg, hogy csak ritkán találkozunk olyan megoldással, amelynél nyitott szubrutinformában, statikusan kezelt makrók szerepelnek. Általában a zárt szubrutinformáju kezelés az elterjedt. Emellett szól -- egyebek között -- a többszöri felhasználásnál már lényegesen kisebb memóriaigény.

Az eddig felsorolt jellemzők is lényegesek céljaink szempontjából, a döntő azonban a DM beépitésének és végrehajtásának időpontja. Statikusan kezelt deklarációknál a DM végrehajtása a forditási idő alatt, a szerkesztés és a futás megkezdése előtt történik meg. Ennek megfelelően /hacsak eleve nincs beépitve/ a beépitésnek is legkésőbb erre a szakaszra kell esnie.

4.3.1.2. A deklarációk dinamikus kezelése esetén -- a fenti szempontokat tekintve -- nincs sok eltérés az előzőkhöz képest, csak a végrehajtás időpontja különbözik lényegesen. A dinamikus kezelésnek mindig a futás idejére kell esnie, aminek

megfelelően a beépitési ütem is ekkorra tehető. Itt jegyezzük meg, hogy a statikus kezelést a dinamikus kezelés speciális esetének tekinthetjük, ti. annak, amelyben

- vagy egyszer kerül sor a DM végrehajtására,
- vagy változatlanok maradnak a megismételt végrehajtások során azok a paraméterek, amelyek alapján a DM müködik.

Ez a körülmény rámutat arra, hogy nem szükségszerüen áll fenn különbség a beépités és a végrehajtás időpontjában, de gyakorlati aspektusból felesleges lenne bonyolultabb eszközt használni, amikor egyszerübb is rendelkezésre áll. A statikus kezelés további előnye, hogy futási időben nem jelent terhelést a programra nézve.

4.3.2. A változóra való hivatkozásnál nincs értelme a fenti megkülönböztetésnek; a hivatkozás itt mindig dinamikusan kezelendő. A hivatkozási makrók általában a következőképpen jellemezhetők:

- compilerben vagy könyvtárban,
- gépi vagy assembly szintü nyelvben definiált,
- zárt szubrutinként kezelendő olyan makroutasitások, melyek
- beépitésére legkésőbb a futási idő alatt /interprter technikánál/ kerül sor és
- futásidő alatt hajtódnak végre.

4.3.3. Az előző két pontban kifejtettekből az alábbi következtetések vonhatók le:

- mind a deklaráció, mind a hivatkozás lehetséges oly módon, hogy nem a compiler-ben megadott eljárásokat /makrókat/ használjuk fel bennük, hanem compiler-től függetlenül /pl. könyvtárban/ definiáltakat;

- a szóban forgó makrók beépitése és végrehajtása a futás fázisában is megtörténhet;
- ezáltal lehetővé válik, hogy a compiler-ben eredetileg figyelembe vett, szintaktikusan rögzitett változók mellett ujabbakat definiáljunk, éspedig ugy, hogy ez csupán a compiler-hez kapcsolódó könyvtárt érintse, magát a compilert ne;
- ehhez természetesen arra van szükség, hogy az előző pontokban felsorolt követelményeket kielégitő makroutasitásokat lehessen hivni a forrásprogramban, továbbá hogy
- az igényelt utófordítási tevékenység elvégezhető legyen. Az utóbbival bővebben foglalkozunk a további réazekben.

4.4. Elsőként a definiált uj változók deklarációjával kapcsolatos problémákat tárgyaljuk. A forrásprogramban deklaráció gyanánt a DM hivását használjuk; a deklarációs tevékenység elvégzéséhez szükséges információkat a hozzá tartozó aktuális paraméter-listában adjuk meg.

4.4.1. A deklarált változót mindenek előtt azonositóval kell ellátnunk. Az azonositó -- forrásnyelvi szinten -- általában valamilyen, betüvel kezdődő betükből és számjegyekből álló jelsorozat. A DM müködésbe lépésekor többnyire már nem ez az információ áll rendelkezésünkre, hanem a forditás során előálló képe. Ez utóbbi -- skalárváltozók esetén -- az operativ memória valamely cimezhető egységének cime; az egység eredetileg a változó értékének tárolására szolgált. A fordító által előállitott kép /általános esetben/ egy változóra történő hivatkozás lesz; ennek speciális esete az imént emlitett skalárváltozó is. Megjegyezzük, hogy a gyakorlatban az uj változó azonositására elegendő a skalárváltozó azonositóját használni. Elvileg megengedhető lenne összetett változó alkalmazása is, de ez nem jár semmi különösebb előnnyel. Emiatt nem is foglalkozunk ezzel az esettel.

Egy azonositónak, a forrásprogram bármely helyén szerepeljen is, a tárgyprogramban mindenütt ugyanaz a cim felel meg. /Más jelölésszintet bevezető blokkszerkezet esetén ez a jelölés hatáskörében érvényes./ A tárgyprogramban azonban előfordulhatnak olyan részek is, amelyekben cimként értelmezhetően ugyanezt a cimértéket találjuk, tehát a hozzárendelés nem egy-egy-értelmü. Emiatt a kérdéses azonositó identifikálása a cim szerint nem terjedhet ki a tárgyprogram egészére, hanem csak azokban a részekben lehetséges, amelyekben

- vagy a program ismert strukturája,
- vagy pedig valamely más információ segitségével biztositva van, hogy a vizsgált cim a kérdéses azonositó képe.

Az azonositó képét azért kell megkeresni, mert a hivatkozások helyén biztositani kell a HM-et aktivizáló olyan sorozat kialakitását, amely a megfelelő deklarációnál kialakitott deklarációs listára /DL/ való hivatkozást is tartalmazza.

A DL-re való hivatkozás legegyszerübb megoldása, ha az átdeklarált változóhoz eredetileg rendelt gépi változót használjuk fel a lista /DL/ kezdetére mutató poiterként /DLP -- deklarációs lista pointer/. Általában elégséges valamely eredetileg skalárváltozónak nyilvánitott változót alkalmazni pointerként, mivel ennek kapacitása elegendő egy lista kezdetére mutató cim tárolásához, s a vizsgált kérdéskörben szerepe csupán erre korlátozódik. A felhasznált változónak természetesen "legálisnak" kell lennie a compiler számára, azaz a szabályoknak megfelelően deklarálnunk kell /kivéve azt az esetet, amikor a skalárváltozók -- mint pl. az INZSENYER-ben -- nem esnek deklarációs kötelezettség alá/.

Az utóforditás számára tehát az "átdeklarált" változó azonositója egy skalárváltozó cime, melynek tartalma a DL kezdetére mutató cim. A DLP-t a DM tölti ki.

4.4.2. A deklaráció folytán kialakitott DL-nek egyes részeit attól függően különböztetjük meg, hogy milyen jellegü információt tartalmaznak /1.ábra/./A DL összeállitását a DM végzi, a felhivásban megadott paraméterértékek alapján./

- Az első rész -- amely üres is lehet -- a tárolóegység--kijelölést tartalmazza, amennyiben a gépi változót valamely külső tárolóban kell elhelyezni. Itt egyaránt szerepelhetnek logikai és fizikai tárolóegység-kijelölések. /A felsorolásban a sorrendnek nincs lényeges szerepe./

- A második rész az operativ memóriában jelöl ki területet, amelyre szükség van még akkor is, ha a bevezetett uj változóhoz valamilyen háttértárolót rendeltünk hozzá, ti. éppen az ennek kezeléséhez szükséges pufferterület számára. A kijelölés megtörténhet pl. oly módon, hogy a DM a deklaráció többi paraméterének értékét felhasználva meghatározza és automatikusan kijelöli az operativ tár valamely szabad területét, a tárgyprogramban vagy az operációs rendszerben található információ alapján. Ha eredetileg nincs ilyen információ, akkor ezt is a DM hivásban kell megadni. Amennyiben szegmentum-cimzéses rendszerről van szó, akkor uj szegmentumot foglalhatunk le a deklarálandó változóra; ilyenkor egyébként nincs szükség az igényelt memóriaterület méretének megkötésére.

- A lista harmadik része a változó strukturájára vonatkozó adatokat és a HM specifikációját tartalmazza. /Az utóbbira egyes esetekben nincs szükség, mivel a hivott DM-hez egyértelmüen hozzá lehet rendelve valamely HM./ Ez a rész adja meg pl. strukturált változók esetében a record mezőinek elnevezését, jellemzőit stb. A deklarált változónak -- strukturált esetben -- valamely más változó is része lehet. Az erre vonatkozó információkat egy részlista tartalmazza, amely a deklarációban elsődlegesként kezelt változóhoz tartozó alaplista része.

Mivel tárgyprogramban dolgozunk, azonositóként itt is egy gépi változó cimét használjuk, ti. azt, amelyet az azonositóhoz a kompilálás rendelt hozzá. Amennyiben a változónak csak a strukturában elfoglalt helyzete lényeges /ha ti. már ez is kijelöli a hivatkozásoknál/, akkor az azonositójára nincs is szükség. Ezért bizonyos esetekben az azonositó rész elmaradhat a részlistából.

Ha a másodkagos változó szerkezete bonyolultabb és emellett önálló deklarációval is rendelkezik, akkor a DL-ben szereplő paraméterek nem közvetlenül a jellemzőket adják meg,hanem a teljes leirásra, listára /DL_1/ mutató skalárváltozó cimét /DLP cime/. Ez a cim, átdeklarált változó esetén, az azonositásra használt cimmel egyezik meg. Ebből következően összetett strukturák kialakitása is lehetővé válik.

Ha egy egyszerü vagy összetett strukturáju változó valamely másik változó egy elemeként szerepel, azonositásához meg kell határozni azt a nagyobb egységet, amelyhez tartozik és ennek DL-je szerint a benne elfoglalt helyét, továbbá saját jellegét és szerkezetét. Ennélfogva az ilyen változóra csakis a nagyobb egység megjelölésével lehet hivatkozni.

A DL elhelyezésével kapcsolatban két lehetőséget emlitünk. Zárt makrokezelés mellett a DM hivásánál elhelyezett aktuális paraméter-lista helyére kerülhet a DL, amely tulajdonképpen a paraméter-lista átdolgozott formája. Ez a megoldás azonban csak statikus deklarációkezelést tesz lehetővé, ismételt végrehajtásnál ugyanis a paraméter-lista már értelmezhetetlen a DM számára. Ezért az ilyen esetben gondoskodnunk kell arról, hogy a kérdéses hivás még akkor se aktivizálja ténylegesen a DM-et, ha a program végrehajtása során ismételten sor kerülne rá. A másik lehetőség az, hogy a listát az operativ memória e változó számára kijelölt területén helyezzük el. Bár ezáltal a memóriaigény megnövekszik, viszont lehetőség nyilik a dinamikus kezelésre.

4.4.3. A HM-nek az átdeklarált változóhoz való hozzárendelése szorosan összefügg a hivatkozással /ekkor kell ugyanis aktivizálni a HM-et/. A hivatkozásnak két alapesetét különböztetjük meg a DM tevékenységének meghatározásával kapcsolatban.

Az egyik, amikor az uj változóra valamilyen speciálisan e célra készült makroutasitás segitségével, esetleg valamilyen müvelettel kombináltan kivatkozunk. Ilyenkor a makrohivás keretein belül csupán szemantikai szabályok irják elő a korrekt hivatkozási módot. A hivatkozás helyességének ellenőrzésére a forditás során nincs közvetlen lehetőség. Futás közben azonban maga az aktivizált HM ellenőrizheti a deklaráció és a hivatkozás kompatibilitását /pl. a DL eljárás specifikáló részében elhelyezett kulcsszó segitségével/. Ezt a makroutasitást -- a hivatkozás révén -- maga a programozó jelöli ki, tehát a változóhoz az eljárás hozzárendelése "off-line" módon /emberi közremüködéssel/ valósul meg; a DM-nek ilyenkor semmi teendője sincs a hivatkozásokkal.

Skalárváltozók esetén, amennyiben a kérdéses változóra az eredeti hivatkozási formák valamelyikével hivatkozunk, egyszerüen egy memóriarekesz cimét kapjuk. Ha eredetileg is eljárás segitségével realizálódott a gépi változó /összetett változók esetében/, akkor a tárgyprogramban a hivatkozásnál egy eljárást aktivizáló utasitást találunk /természetesen a megfelelő paraméter-listával/. A tárgyprogram hivatkozási módja egyik esetben sem felel meg az átdeklarált változónak, mivel az eredeti helyett, egy ujonnan definiált HM aktivizálására lenne szükség. Ekkor a DM-nek kell gondoskodnia arról, hogy a tárgyprogramban eredetileg található hivás egy ujjal helyettesitődjék. Ez megköveteli a tárgyprogram egészének átfésülését abból a célból, hogy meghatározzuk a kérdéses változóra való hivatkozások helyeit. Itt döntővé válik a felismerhetőség kérdése, amelyet mint a fordithatóság feltételét vizsgáltunk 1.-ben /lásd ott/.

Mind a skalár-, mind az összetett változók esetében szükséges ismernünk az átvizsgálandó tartományt, mely logikailag a cimtartomány egy vagy több szeletét tartalmazza. Az emlitett szeletekben természetesen nem fordulhat elő olyan cim /utasitás/, amely a helyettesitendő skalárváltozó cimével /szubrutin-hivó utasitással/ egyezik meg.

Ismerve a hivatkozási helyeket, még a végrehajtás előtt el kell végezni azok módositását az uj jelentéstartalomnak megfelelően. Ezt az első hivatkozás előtt, az uj változó érvénybelépésekor, tehát lényegében a deklarációval egyidőben célszerü megtenni. Amennyiben tehát utóforditás szükséges a hivatkozások átalakitása érdekében, azt is a DM feladatai közé sorolhatjuk. Az átalakitás legegyszerübb megoldását az interpretációs technika biztositja. Gondoskodnunk kell arról, hogy a hivatkozásokat tartalmazó programrészek végrehajtása egy interpreter program segitségével valósuljon meg, ami szükségessé teheti, hogy vezérlésátadó utasitásokat iktassunk be, s ez pótlólagos helyigényt jelent. Ezt valamilyen szabad terület /amely pl. a dinamikus memóriához tartozhat/ felhasználásával elégithetjük ki. Ugyanez a helyzet akkor is, ha az átalakitásnál közvetlenül a végrehajtandó utasitássorozatot épitjük be a régi hivatkozások helyett.

Az esetek többségében azonban nincs utóforditásra szükség, mivel -- amint ezt már a 3.3.2.-ben emlitettük -- az uj változók uj müveleteket is igényelnek. Az ezeket realizáló makroutasitások pedig /az uj változókkal kapcsolatban/ eleve tartalmazhatják a megfelelő hivatkozási eljárás aktivizálását /makrohivást/.

4.5. A hivatkozással kapcsolatos kérdések tárgyalásának befejezéseként röviden összefoglaljuk, hogy a DM által, ill. a müveletben közvetlenül kijelölt HM hogyan használja fel a DM tevékenysége során kialakult hivatkozási rendszert, amelynek sémáját az 1. ábra szemlélteti.

A müveleti makrót /MM/ aktivizáló utasitás mögött /2.ábra/ -- az utóforditási müveletek eredményeképpen -- olyan paraméter-listát találunk, melynek megfelelő helyein aktuális paraméterként átdeklarált összetett változó áll. Ennek tárgyprogrambeli jelentkezése: a kérdéses helyen a változó azonositójának -- compiler által előállitott képe áll, ti. egy gépi változónak -- a DLP-nek -- a cime. A változóra vonatkozó információ feldolgozása során a HM ennek a cimnek alapján azonositja a változót, tehát ennek segitségével találja meg a megfelelő DL-t. Ez a következőképpen történik: az MM-ben a DLP cimének átadásával aktivizáljuk a HM-et, mely a benne található cim alapján jut el az átdeklarált változóhoz tartozó DL-hez. A DL-ben lévő információkat ezután vagy ugyanez a HM, vagy valamely /a DL eljárás-specifikáló részben megjelölt/ "segéd-HM" dolgozza fel. A DL-hez vezető hivatkozási- és a HM-hez vezető vezérlésátadási-láncot a 2. ábra szemlélteti.

5. Makroutasitások alkalmazása az INZSENYER-autokód rekord-rendszerének /REC/ kialakitásában

5.1. Információfeldolgozási feladatok megoldásakor általában valemilyen objektumokhoz /egységekhez/ tartozó, egymással összefüggő "információcsomagokat" kell kezelnünk. Ilyen egységek lehetnek pl. a fizetési jegyzéken szereplő dolgozók valamely bérszámfejtési programban, vagy az összetett változók egy forditóprogramban stb. A szóbanforgó egységekhez többnyire heterogén, de meghatározott szerkezettel és hierarchikus felépitéssel rendelkező információhalmaz tartozik. Az ilyen jellegü információhalmazok kezelésének megkönnyitésére vezették be a rekord, ill. a struktura fogalmát [24].

Rekordon általában különböző tipusu elemekből, un. mezőkből felépülő összetett változót értünk. A mezők vagy tetszőleges tipusu egyszerü változók, vagy ezekből felépitett tömbök, vagy ismét rekordok lehetnek. /Egyes implementációkban, természetesen korlátozásokkal, illetve általánosabb fogalmakkal is találkozunk az alkalmazott mezőkre vonatkozóan./ Az azonos jellegü egységekhez tartozó azonos szerkezetü rekordokat egységes rekord-osztályba foglalják össze; ez az implementációkban un. file formában jelenik meg. A file a rekord-osztály elemeit tartalmazza és a saját szerkezetét leiró, valamint az azonositási és hozzáférési /irást, olvasást engedélyező, ill. tiltó/ információkat magukban foglaló részekből -- file feje /header label/, ill. vége /trailer label/ -- áll.

A rekord-osztály és a file között lényeges különbséget találunk. Az előbbi ugyanis rendezetlen, az utóbbi viszont rendezett, vagy részben rendezett rekord-halmaz. A rendezetlenség itt azt jelenti, hogy az egyes rekordokhoz tetszőleges sorrendben férhetünk hozzá, tehát a feldolgozás menetét nem köti a tárolási, vagy valamely egyéb sorrend.

A rendezettség jelentése ezzel szemben az, hogy valamennyi rekordra nézve szigoruan meg van határozva -- elsősorban a fizikai elhelyezésből adódóan -- az őt megelőző, ill. követő rekord, függetlenül azoktól a relációktól, amelyeket a rekordokban található referencia-tipusu mezők [24] adnak meg. Rendezett rekord-halmaz esetén egy lépésben mindig csak a rendezésben soronkövetkező /esetleg a megelőző/ rekordhoz férhetünk hozzá. A referencia-mezők előirhatnak egy vagy több logikai kapcsolatot is, ezek azonban csak a rendezés által megszabott uton -- általában több lépésben -- valósulhatnak meg. A rendezett rekord-osztály elemeihez való hozzáférési módot szekvenciális /soros/ hozzáférésnek nevezzük. A gyakorlatban többnyire ez fordul elő, mivel viszonylag egyszerü kezelést tesz lehetővé és a soros hozzáférésü tárolók /lyukkártya, mágnesszalag/ esetén nemcsak a leggazdaságosabb, hanem általában az egyetlen lehetséges megoldás.

A szekvenciális hozzáférés mellett egyéb hozzáférési módok is ismeretesek, nevezetesen

- a direkt,
- a környezet közvetlen elérésén alapuló,
- az indirekt és
- a fa-strukturát felhasználó

hozzáférési módok [2] .

Direkt hozzáférésről akkor beszélünk, ha a keresett rekord helye a keresés alapját képező jellemzőkből /keresési kulcs, referencia-mező/ valamilyen függvény segitségével közvetlenül megállapitható, maga a rekord pedig egy lépésben kiolvasható. /Ebben az esetben tulajdonképpen a file is rendezetlen rekord-halmaznak tekinthető. A további esetekben azonban file-on -- az előbbi értelmezésnek megfelelően -- rendezett, vagy részben rendezett rekord-halmazt értünk./

A környezet közvetlen elérésén alapuló hozzáférési módról akkor beszélünk, ha direkt eljárással a keresett rekordnak csak a környezetét tudjuk meghatározni, majd e környezeten belül magát a rekordot soros kereséssel érjük el.

Indirekt hozzáférési módról akkor van szó, ha a keresési kulcs segitségével csupán egy pointerhez jutunk el /pl. soros keresést használva/, amely a kérdéses rekordot tartalmazó rész--file-ra mutat. Ebben a rész-file-ban azután a már emlitett hozzáférési módok valamelyike alkalmazható. Az indirekt hozzáférési mód lehet többlépcsős is.

A fa-sturkturát felhasználó hozzáférési módnál általában bináris fákat alkalmaznak /ezekre ugyanis bármely fa visszavezethető [15]/. Ez a hozzáférési mód az indirekt elérés egy többlépcsős változatának is tekinthető, amelyben minden egyes lépésnél a bal- vagy a jobboldali ág által kijelölt rész-filehoz jutunk.

Megjegyezzük, hogy egy önmagában tekintett file nem sorolható mereven egyik kategóriához sem. Hogy adott esetben melyik kategóriához tartozónak tekintjük, az -- egyebek között -- attól is függ, hogy mi a keresési kulcs. Előfordulhat pl., hogy valamely file egy keresési kulcsra vonatkozóan direkt hozzáférésü, egy másikra vonatkozóan viszont soros hozzáférésü. Azt, hogy a felhasználó számára hogyan jelenik meg a file, e mellett az alkalmazott software szintje határozza meg. Ebben a vonatkozásban igen nagy mértékü lehet a változatosság: pl. soros hozzáférésü file-ra direkt elérésü kezelőrendszer is épülhet.

A gyakorlatban természetesen mindig számolni kell az elérés rugalmassága és a hatékonyság közötti ellentmondással. Pl. dolgozhatunk mágnesszalagos tároló esetén is direkt hozzáféréssel, de ebben az esetben -- az általánosan alkalmazott mágnesszalag-rendszerek mellett -- igen nagy hozzáférési időket kapunk.

A hozzáférési mód kiválasztásakor célszerü figyelembe venni egyrészt, hogy milyen hardware-eszközök állnak rendelkezésre, másrészt, hogy milyen müveleteket kivánunk végezni a file-on. Az egyes hozzáférési módok esetén ugyanis a különböző file-kezelési tevékenységek igen eltérő hatékonysággal valósithatók meg [15]. /A továbbiakban csak a soros és a direkt hozzáférési módokkal foglalkozunk, mivel az ismertetendő REC-rendszerben ezek realizálhatók./

A rekord-osztályokkal kapcsolatban un. alaptevékenységeket különböztetünk meg, amelyek felhasználásával a többi művelet előállitható. Ezek:

- rekord beépitése a rekord-osztályba,
- rekord törlése a rekord-osztályból és
- rekord kiválasztása a rekord-osztályból /megadott szempont alapján/.

Egy rekord-osztály generálása a rekordok egymás utáni előállitásával történik. Az időben egymás után keletkező rekordokat ugy illesztjük az osztály logikai szerkezetébe, hogy a /már meglévő/ kapcsolódó rekordok referencia-tipusu mezőit megfelelő értékre állitjuk be. Ez a tevékenység azonban csak a rekord-osztályon belüli elhelyezést oldja meg, de független a file hozzáférési módjától. A referencia-változók értékének beállitásakor azonban már ismernünk kell a beiktatott rekordnak a file-ban elfoglalt helyét is. Ezért a file-ban való elhelyezésnek időben meg kell előznie a rekord-osztályhoz való illesztést. Szekvenciális hozzáférésü file generálásakor az ujabb elem közvetlenül a legutoljára beiktatott elem után kerül, tehát a rekordok az időbeli beérkezés sorrendjébe rendezve, folytonosan helyezkednek el a file-ban. A feldolgozási folyamatban ezt a sorrendet megváltoztathatjuk /pl. egy rendezési művelettel/, de ekkor tulajdonképpen egy uj file-t állitunk elő, amelyre ugyancsak az időbeli beérkezés szerinti rendezettség a jellemző, csakhogy ez már megegyezik a megadott szempont szerinti rendezettséggel.

Átrendezéskor természetesen gondoskodnunk kell a referencia-mezők módositásáról is.

Direkt hozzáférésü file-oknál, amennyiben adva van a kulcs, a beépités a kulcs által kijelölt helyre történik, függetlenül attól, hogy volt-e már értékes információ a kérdéses helyen. Amennyiben a kulcs értéke nincs eleve megadva, a rekordhoz egy "üres" helyhez tartozó kulcs-értéket rendelünk és ezt használjuk fel a kapcsolódó rekordok referencia-mezőinek kitöltésénél. Ez az eljárás természetesen feltételezi, hogy az üres helyeket valamilyen nyilvántartási rendszerben számon tartjuk. Emiatt egy file rekordjainak nem feltétlenül folytonosan kell elhelyezkedniök a file számára kijelölt tárolóterületen belül. /A nyilvántartást megvalósithatjuk pl. az üres helyek láncolt listárafüzésével, vagy egy speciális "üres" rekord felhasználásával, esetleg az emlitett eljárások együttes alkalmazásával./

A rekordok törlése szintén igen egyszerüen oldható meg, mivel ez a file vonatkozásában csak a szabad helyek nyilvántartásba vételét jelenti. Emellett "nullázni" kell a kérdéses rekordra vonatkozó referencia-mezőket és változókat is. Egy referencia-tipusu változó nullázása azt jelenti, hogy egy speciális értéket /pl. "NULL" [24] / rendelünk hozzá, amely azt mutatja, hogy a változó nem jelöl ki semmilyen rekordot.

Szekvenciális hozzáférésü file-nál a törlés nem oldható meg közvetlenül, minthogy ez megszüntetné az elhelyezés folytonosságát. Ez a körülmény, valamint a beiktatásnak az a tulajdonsága, hogy beirni csak az utolsóként elhelyezett rekord után lehet, szükségessé teszi az input- /I-/ és az output- /O-/ file fogalmának bevezetését. Direkt elérés esetén, ahol nehézség nélkül megoldható mind a tetszőleges helyre való beirás, mind a tetszőleges helyről való törlés, ezekre a fogalmakra nincs szükség.

Előfordul azonban, hogy más szempontok miatt akkor is megkülönböztetjük az I-, ill. az O-file-okat.

Az olyan file-t nevezzük output file-nak, amelybe csak rekordok beépitése végezhető el. Ilyenkor a következő rekord kérése azt eredményezi, hogy a file soronlévő -- szabad -- helye válik hozzáférhetővé; az ennek megfelelő rekord ekkor még definiálatlan és csak a megfelelő értékadás után válik meghatározottá. A legközelebbi rekord-kérés /rekord-váltás/ hatására azután ez a rekord válik hozzáférhetetlenné, és ismét egy definiálatlan rekordhoz jutunk. Ahhoz azonban, hogy egy file-ban a soron következő rekordot kérhessük, valamilyen módon lehetővé kell tenni az első rekord elérését. Erre egy speciális müvelet,az un. file-nyitás szolgál. /A file-nyitás az emlitett tevékenység mellett még a kezelőrendszerrel kapcsolatos adminisztrativ teendőket is végez./ Az output file-okkal kapcsolatos másik /kiegészitő/ müvelet a file lezárása. Ennek az a hatása, hogy megszünik a file output jellege, tehát további beiktatások nem végezhetők el. Egyes esetekben a file lezárása még egy további eredménnyel is jár: a file jellege inputra változik [16] .

Input-file-nak az olyan file-t nevezzük, amelynél a nyitás és a váltás az első, ill. a soron következő rekordot teszi hozzáférhetővé /kiolvashatóvá/ és a hivatkozások a következő váltásig erre a rekordra vonatkoznak. Az input-file-okban levő rekordokat csak kiolvasni lehet; ujak beiktatása vagy változtatás nem lehetséges. Vannak olyan rendszerek, amelyekben lehetőség van egy file input jellegének megváltoztatására oly módon, hogy egy folytatási utasitás segitségével visszaállitjuk azt az állapotot, amely a file lezárását megelőzte.

Input-file-oknál definiáltak olyan müveletet is, amely nem a soron következő, hanem a jelenlegitől meghatározott távolságra lévő rekordot szolgáltatja. Ez természetesen, az egyszerü váltási utasitás ciklikus megismételtetése, a közbeeső rekordok átlapolásával.

Mind a beiktatási, mind a törlési müvelet elvégeztetésénél valamilyen módon hivatkozni kell arra a rekordra, amelyre a kérdéses müvelet vonatkozik. A hivatkozásra természetesen más müveletek elvégzése céljából is szükség van.

A file-t ill. rekord-osztályt olyan összetett változónak lehet tekinteni, mely maga is összetett változókból /ti. az osztály azonos szerkezetü rekordjaiból/ áll. A rekordok mezőkre tagolódnak; a mezők szintén lehetnek összetettek. A rekord-osztály elemeinek strukturája azonos; e struktura leirását a rekord-osztály deklarációja tartalmazza. A gyakorlatban azonban nem a rekord-osztály, hanem a file meghatározását kell megadnunk, és ebbe -- az előbb emlitettek mellett -- a file jellegére /I/O/, a külső tárolóeszközre, a központi tárban elhelyezkedő ill. oda benyuló pufferterületre stb-re vonatkozó információkat is bele kell foglalni. Hogy pontosan milyen, és mennyire részletezett kiegészitő információt szükséges megadni, az a felhasználandó operációs rendszer adatkezelő részének szintjétől is függ, minthogy az emlitett információk jelentős részét az az operációs rendszer automatikusan is szolgáltathatja.

A hivatkozásnak olyannak kell lennie, hogy lehetővé tegye a file-ban szereplő legkisebb egység -- az egyszerü változókból álló mező -- elérését is. A file rekordjainak közös nevét és szerkezetét megadó deklarációban az egyes mezőkhöz azonositókat rendelünk. Ezután a mezőkre ezekkel az azonositókkal hivatkozhatunk, természetesen a nagyobb egység /a rekord/ megjelölésével. A hierarchiában való elhelyezkedés minden mezőhöz hozzárendel a név mellett egy sorszámot is, amely szintén lehet hivatkozási eszköz. A sorszám segitségével könnyen megoldható pl. egy rekord több mezőjére való ciklikus hivatkozás /a névvel való azonositáskor minden ciklus-lépést külön fel kell sorolni/.

Valamely mezőre történő hivatkozásnál a file egy konkrét rekordját is meg kell jelölni; ennek elmaradása esetleg ugy értelmezendő, hogy a hivatkozást tartalmazó müveletet a file összes rekordján végre kell hajtani.

Az olyan rendszerekben, amelyekben egyes rekordok önállóan, a rekord-osztályoktól függetlenül is előfordulhatnak, a rájuk való hivatkozás egyértelmü, és nem kell külön kijelölő részt tartalmaznia. Ezért az ilyen rendszerekben a file és a hozzá tartozó rekord deklarációja külön is válhat, de a file deklarációjában meg kell nevezni azt a rekordot, amely a file elemeit irja le. Ez a rekord egyben a file aktuális /hivatkozott/ rekordját is tartalmazza, mégpedig oly módon, hogy amennyiben a file valamely elemét kérjük, az ebbe való áttöltést, ill. innen való eltárolást eredményez. Mindaddig, amig nem történik ujabb hivatkozás a szóban forgó file-ra, nincs kapcsolat közte és az emlitett rekord között. Ezért a file elemeit leiró rekord egyidejüleg több azonos szerkezetü file-hoz is tartozhat. Itt a rekord-osztálynak az a legkisebb eleme, amelyre hivatkozni lehet, a rekord, és ennek is csak az áttöltése történhet a kapcsoló rekordba, vagy onnan. Az egyes mezőkre csak ezen keresztül hivatkozhatunk. A kivánt rekordok kiválasztása tekintetében azonban nincs eltérés attól az esettől, amikor a file-ok és a rekordok között egy-egy értelmü a megfeleltetés -- ezért csak az utóbbi esetet elemezzük részletesebben.

Mivel ebben az esetben a file-hoz egy-egy értelmüen van hozzá rendelve az őt alkotó rekordok strukturája, nem szükséges a file-t és a benne szereplő rekordokat külön azonositóval ellátni. Igy a file neve egyuttal a file valamely rekordját is jelöli, attól függően, hogy milyen müveletben szerepel. Szekvenciális hozzáférés esetén ez a név mindig azt a rekordot jelöli, melyet a nyitási, váltási, zárási és folytatási müveletek addig végrehajtott sorozata meghatároz.

Direkt hozzáférésnél -- tekintettel arra, hogy bármely rekord egy lépésben is elérhető -- bizonyos kiegészitő információkra van szükség. Egy számitásba jöhető lehetőség az lenne, hogy megadjuk valamely mező értékét és az ezzel való egyenlőség alapján hivatkozunk az értéknek megfelelő rekordra. Ezáltal azonban általában nem egyetlen rekordot, hanem egy rekord részosztályt jelölünk ki. Megjegyezzük, hogy ezt a hivatkozási módot egyértelmüvé lehet tenni azáltal, ha a részosztály elsőnek megtalált elemét tekintjük kiválasztottnak. Ezzel az esettel azonban nem foglalkozunk részletesebben, mivel a megvalósitott rendszernél kizártuk. Egy másik lehetőség az, hogy a kijelölést referencia-változó segitségével végezzük el. A referencia-változó olyan értékek halmazán futhat át, amelyek segitségével a hivatkozási makro /eljárás/ már képes meghatározni a hivatkozott rekord helyzetét. A szóban forgó értékhalmaz pl. a természetes számok halmazának egy véges rész-halmaza lehet; minden természetes szám, amely e halmazban szerepel, egy-egy értelmüen hozzá van rendelve valamely potenciális rekordhoz /ti. egy rekord számára fenntartott helyhez/, de a feldolgozás során még nem definiált rekordokhoz tartozó természetes számokkal való hivatkozásnak csak valamely uj rekord beépitésekor van értelme. Uj rekord beépitésénél nem szükséges a referencia-értéket "kivülről" megadni, ezt automatikusan is elvégezheti egy beépitő eljárás azzal, hogy valamely meghatározott üres helyet jelöli ki a rekord számára. A megfelelő referencia-érték ilyen esetekben történő használatához meg kell engednünk, hogy egy referencia-változó értékül kapjon egy fenti módon beiktatott rekordot /vagy általában bármely rekordot/. Tartalmi szempontból ez azt jelenti, hogy a referencia-változó értéke a kérdéses rekordhoz /esetenként automatikusan/ hozzárendelt referencia-érték lesz. Amennyiben a referencia-értékek halmaza az adott nyelvben tipusként szerepel, akkor lehetőség van arra, hogy a programozó maga irányitsa az elhelyezést /controlled memory allocation, [17]/.

A programozó által vezérelt hozzárendelésnek számos előnye van. Ezek közé tartozik az, hogy a programozó a feldolgozás /esetleg változó/ céljainak megfelelően irányithatja a rekordok elhelyezését. Ugyanakkor fellép az a kényelmetlenség, hogy az elhelyezésnél olyankor is definiálni kell a referencia-változó értékét, amikor semmiféle kikötés nincs a rekordok elhelyezkedésére. Amennyiben a programozó számára a referencia-változó értéke hozzáférhetetlen, az automatikus elhelyezés /automatic memory allocation/ megkönnyiti a munkát, viszont nincs lehetőség speciális igények érvényesitésére.

A harmadiknak megjelölt alaptevékenység -- valamely rekord kiválasztása -- egyrészt a hozzáférési rendszertől, függ, másrészt attól, hogy minek alapján végezzük a keresést. Szekvenciális hozzáférésnél ez utóbbi nem jön számitásba; mindig csak a következő rekordot választhatjuk ki, és amennyiben ez nem felel meg a keresési kritériumnak, utána soron következő rekord kiválasztását kérjük s.i.t. Direkt hozzáférés esetén /ha a kiválasztást valamely referencia-érték alapján kérjük/ a hivatkozást egyszerüen a referencia-konstans /vagy az ezt tartalmazó változó/ felhasználásával -- keresés nélkül -- végezzük. Ha a kiválasztás szempontja más, ez az ut nem járható. Ilyenkor a referencia-értékek segitségével ugyanazt az eljárást kell alkalmaznunk, mint a szekvenciális elérésnél.

5.2. Az előzőleg tárgyalt elvek és módszerek felhasználásával egy olyan makro-utasitás-csomagot dolgoztunk ki, melynek segitségével kibővithető az INZSENYER autokódban kezelhető operanduszok halmaza. Ebben a nyelvben eredetileg egész és valós tipusu változók, valamint azonos tipusu egyszerü változókból felépitett, maximálisan kétdimenziós tömbök elemeinek kezelésére volt lehetőség. Számos megoldandó probléma azonban más operanduszokat is követelt.

Az emlitett tipusokon tulmenően szükség volt idézetek /string tipus/, a tömbök mellett pedig összetett, különböző tipusu mezőket tartalmazó, tehát az eredetiektől strukturájukban is eltérő változók kezelésére is. Mindezek bevezetését ugy kellett megoldani, hogy az ne igényeljen változtatást magán a forditóprogramon, minthogy ez aránytalanul sok munkát követelt volna. Számitásba véve, hogy a nyelv eredetileg is lehetővé tette a felhasználó által definiálható könyvtári /nem strandard/ programok hivását a

LIBRARY PROGRAM N /P_1, P_2, ..., P_k/ *

formáju utasitás révén /lásd 1.sz. melléklet/, azt a megoldást választottuk, hogy az uj operanduszok bevezetését az előző fejezetekben ismertetett általános érvényü módszerek segitségével végezzük el.

A definiált könyvtári program olyan makró-definicióként fogható fel, amelynek eredményeként

- könyvtárban definiált,
- gépi nyelven megadott

makrokhoz jutunk. Ezek beépitése

- a forditási idő alatt,
- zárt szubrutin formájában történik.

Ugyanazon makro beépitése a paraméterlista függvényében más-más módon valósul meg; a különbség a felhivó utasitássorozatban jelentkezik. /A hivás gépi reprezentációját a 3. ábra szemlélteti./

A felsorolt tulajdonságok kielégitik a 3.3.2. pontban megjelölt és a 4.3.1., továbbá a 4.3.2. pontokban megfogalmazott felhasználás /az operanduszok körének bővitése/ követelményeit.

5.2.1. Azok a változók, amelyekkel az eredeti nyelvet kibővitettük, az 5.1. pontban elemzett rekord, rekord-osztály és file fogalmak speciális esetei, egyes vonatkozásokban az általánosság lényeges megszoritásával.

A következőkben olyan rekordok kezeléséről lesz szó, amelyek mezői az eredetileg a nyelvben használatos egész és valós tipusu egyszerü változók mellett string-tipusuak /karaktersorozatok/ is lehetnek. Az első két tipushoz tartozó mezők egy-egy gépi szót foglalnak el a MINSZK-22 jelü gépen szokásos ábrázolásmódban /a gépi szó hossza itt 37 bit/. Az információk kezelésére szolgáló string tipusu mezők hossza tetszőleges lehet; egy mezőre vonatkozóan ezek rögzitett számu egymást követő memóriarekeszt foglalnak el. Egy-egy szó -- a szokásos ábrázolás mellett -- 6 karaktert tartalmaz. Ha a tárolt információ karaktereinek száma 6-nak nem egész számu többszöröse, a fennmaradó részek automatikusan a szóköz kódjával töltődnek ki. Az ily módon kapott karaktersorozatot egységes egésznek tekintjük; ennek részein nem végezhetők közvetlenül müveletek /közvetett müveletvégzésre természetesen van lehetőség/.

Nem engedtük meg a tömbök, ill. rekordok mezőként való szerepeltetését, bár ennek nincs elvi akadálya.

Mind a rekordokra, mind pedig a mezőkre azonositókkal hivatkozhatunk. Mezőre történő hivatkozásnál meg kell jelölni, hogy melyik rekordhoz tartozik. Ugyanazt az azonositót több rekord mezőinek jelölésére is felhasználhatjuk, azzal a megkötéssel, hogy a jelölt mezőnek mindegyik rekordban azonos tipusunak, string tipus esetén pedig azonos hosszuságunak kell lennie. A mezőre történő hivatkozás egy másik módja az, hogy a rekord megnevezése mellett a kérdéses mező sorszámát adjuk meg /amelyet a rekordban elfoglalt helye határoz meg/. A rekordok és a mezők azonositására használt jelölés formailag megegyezik az AKI skaláris változóinak jelölésével.

Mivel a rekordok és a rekord-osztályok között egy-egyértelmü megfelelés áll fenn, ezért a rekordok azonositói egyuttal a megfelelő rekord-osztályokat is jelölik, és deklarációjuk is egyben történik. A rekord-osztályokat közvetlen hozzáférésü file-okkal realizáltuk, tekintettel arra, hogy a MINSZK-22 mágnesszalagos tárolói ezt lehetővé teszik /ti. a mágnesszalagon is külön cimezhető minden szó, és bármelyik -- legfeljebb 8K szónyi -- szelet irása /olvasása egy lépésben elvégezhető, függetlenül a megelőző mágnesszalag I/O müveletektől/. Minden file-hoz hozzátartozik

- egy mágnesszalag /pontosabban annak valamely szavától kezdődő szelete; a szelet hossza nincs korlátozva/ és
- egy pufferterület a központi tárolóban /4. ábra/, amely az egymáshoz közeleső rekordok elérésének meggyorsitását és a mágnesszalagos I/O müveletek számának csökkentését szolgálja.

Közvetlen hozzáférési módnak megfelelően az I- és O-file-ok megkülönböztetése nem szükséges. Ez a hozzárendelési mód lehetővé teszi, hogy ugyanazon a mágnesszalagon több file-t is elhelyezhessünk. Az egy szalag kapacitását meghaladó méretü file-ok kezelésére a REC-ben nincs lehetőség.

Egy file-on belül az egyes rekordok kiválasztása referencia-változók segitségével történik. Referencia-változók gyanánt az AKI egész tipusu értékeket tartalmazó változói használhatók. Emiatt lehetőség van referencia-tipusu mezők kialakitására is, mivel egy egész tipusu változó azáltal válik referencia-tipusuvá, hogy referencia tipust igénylő helyen fordul elő. De az egész tipusu mezők mégnem használhatók közvetlenül referencia-változókként, mert a megfelelő szintaktikus egység csak egyszerü változo, vagy konstans lehet.

5.2.1.1. Egy rekord strukturáját, a hozzá tartozó file adatait a közös deklarációban adjuk meg. A deklarációt egy makrohivás paraméterlistája tartalmazza. Ebben a paraméterlistában fel kell sorolnunk:

1./ a rekord /file/ nevét,
2./ a mezők számát,
3./ a file számára a központi tárolóban kijelölt pufferterület kezdetét és
4./ végét,
5./ a megfelelő mágnesszalag-egység logikai sorszámát,
6./ a mágnesszalagon elfoglalt hely kezdőcimét, valamint
7./ a file-ban lévő rekordok strukturájának leirását, amely paraméter-párokból áll.

A 3./--6./ pontokban felsorolt paraméterek adják meg a tárolásra vonatkozó információkat; a 2./ és a 7./ pontbeliek az egyes rekordokra vonatkoznak. A 7./ pontban szereplő paraméter-párok az egymás után következő mezőket specifikálják. A paraméter-párok első eleme a mező neve; a második elem a mező tipusának leirására szolgál. A második paraméter 1 vagy 0 értéke az egész, ill. a valós tipust, mig a negativ érték a string-tipust jelöli. A negativ értékek abszolut értéke egyben meghatározza a mező számára lekötött szavak számát. A rekordok és a mezők neve azonositó, mig a többi paraméter -- az AKI könyvtári programjait hivó utasitások szintaxisának megfelelően -- konstans vagy egyszerü változó lehet. Változó mezőszámu deklaráció kialakitására lehetőséget ad az a körülmény, hogy a hivások paraméter-listája tetszőleges hosszuságura választható, függetlenül a hivott program definiciójától. A forrásnyelvi hivás alapján előállitott gépi reprezentáció csak a paraméter-lista függvénye; a lista valamennyi elemének egy-egyértelmüen megfelel a gépi reprezentáció egy-egy eleme.

A deklaráció akkor lép érvénybe, ha az őt tartalmazó

utasitás végrehajtódik. Csak az első végrehajtásnak van hatása; a későbbiek során a hivás üres utasitásként viselkedik. Emiatt a deklaráció statikus jellegü.

5.2.1.2. Az egyes rekordokra történő hivatkozás a deklarációs /37/ könyvtári program tevékenységének eredményére támaszkodva valósul meg. Mivel a rekordok használata csak uj müveletekben van megengedve, a hivatkozási tevékenységet ellátó szubrutinok hivásáról nem kell külön gondoskodni; az uj müveleti makrók hivása következtében ezek beépitése automatikusan történik meg. /Az AKI forditóprogramjában a beépitést ugy végzik, hogy a program átfésülésével megállapitják: vannak-e benne beépitetlen könyvtári programra történő hivatkozások, azaz

- 31 00 N 0017 N 177_8

alaku utasitások. Ha vannak, végrehajtódik a megfelelő programok beépitése. Ez a tevékenység addig ismétlődik, mig az átfésülésnél már nem található fenti alaku utasitás./

A hivatkozás minden esetben valamilyen müveleti utasitás keretei között történik meg. A müvelettől függően ez vagy egy referencia-változóval meghatározott teljes rekordra, vagy egy ilyen rekord valamely mezőjére vonatkozik. Amennyiben a kért rekord a pufferterületen van, akkor -- a müvelet jellegétől függően -- kétféle tevékenység valósulhat meg: kiolvasás a pufferterületről egy müveleti munkamezőbe, vagy beirás a pufferterületre egy munkamezőből. A müveletvégzések egyébként a munkamezőn történnek.

Az első hivatkozásig a file pufferterületét üresnek kell tekinteni. A továbbiakban -- ha a referencia-értékkel megjelölt rekord nincs a pufferterületen -- annyi rekord beolvasása hajtódik végre ettől a rekordtól kezdődően, amennyi a pufferterületen elfér. Ezt megelőzően azonban megtörténik a puffer kiürítése, ha a benne lévő valamely elem /rekord vagy mező/ előzőleg uj értéket kapott.

/A kiürités a mágnesszalagra való visszairást jelent, mégpedig arra a helyre, ahonnan előzetesen leolvastuk a puffer tartalmát/. Üres esetben a kiürités nem eredményez semmiféle tevékenységet. Ez a kezelésmód /5.ábra/ azt eredményezi, hogy bármely rekordhoz hozzáférhetünk legfeljebb egy mágnesszalag-output és -input müvelettel. Emellett bármely rekordon végrehajtott változtatás egy pufferváltás után a mágnesszalag megfelelő helyén is megjelenik.

5.2.2. Nem valósitottuk meg a törlést a rekord-osztályokkal kapcsolatos alaptevékenységek közül; az üres helyek nyilvántartása sem történik meg automatikusan. A felhasználónak kell figyelemmel lennie arra, hogy ne szerepelhessenek definiálatlan rekord elemei olyan helyeken, ahol az értékükre lenne szükség; ezekben a müveletekben csak definiálandóként fordulhatnak elő. Valamely adott rekord elérése, vagy közvetlen referencia-érték segitségével, vagy -- ezek sorozatán átfutva -- összehasonlitásokkal valósitható meg. Az utóbbi esetben is felhasználjuk a referencia-értékeket, de csak közvetitő szerepben.

5.2.2.1. Rekordok előállitása, ill. beiktatása értékadó utasitással történhet, amelynek egyik fontos speciális esete a rekord-input utasitás. Ez az utasitásban megjelölt rekordnak a lyukszalagolvasón lévő következő adatsort adja érték gyanánt; a müvelet az egész rekordra vonatkozik. A rekord-input utasitás a rekord-néven és a konkrét rekordot kijelölő referencia-változón kivül még egy cimkét is tartalmaz. Ennek az a célja, hogy ismeretlen hosszuságu adatsor esetén maga az adatszalag vezérelhesse a beolvasó ciklust azáltal, hogy a lyukszalag-file végén erre a cimkére adja át a vezérlést. A végjelzést az a szinkron-lyuk sorozat szolgáltatja, amely a lyukszalagok végén van. Ha az input-utasitás végrehajtható, a program a következő utasitással folytatódik.

Ha a végrehajtás nem sikerül, akkor -- egy kulcstól függően, amelynek állása futás közben a vezérlőpulton változtatható -- vagy az input-utasitás megismétlése következik be /ujabb lyukszalagnak az olvasóba helyezése és továbbinditás után/, vagy a cimkével megjelölt utasitásnál folytatódik a program.
Ezáltal lehetőség nyilik több -- előre meg nem határozott számu -- tekercs beolvasására is. A beolvasandó lyukszalag formátumát a felsorolt mezőket elválasztó jelek /szóköz, kocsi-vissza, soremelés/ határozzák meg. Igen fontos, hogy a szalag formátuma kompatibilis legyen az olvasóutasitásban szereplő rekord deklarációjával, erre a programozónak kell ügyelnie.

Egyszerüsiti az adatszalag előkészitését, ha egy mező csupán "macskakörmöt" tartalmaz, ami azt jelenti, hogy ennek a mezőnek az értéke az előzőleg beolvasottal egyezik meg. A beolvasás megkezdése előtt az "előző mező" értéke definiálatlan. A "macskaköröm" használatával vigyázni kell; könnyen hibás eredményre vezethet, ha váltakozva -- más-más strukturáju rekordok beolvasása közben -- fordul elő. /Nincs ugyanis külön pufferterület a különböző rekordok számára./

Az input-utasitás alakja a következő:

LIBRARY PROGRAM 41 /P_1, P_2, P_3/ *

ahol

P_1 - a rekord/file neve,
P_2 - a referencia-változó vagy konstans,
P_3 - cimke.

5.2.2.2. A rekordok értékét nemcsak az input utasitással, hanem aritmetikai, ill. átviteli müveletekkel is változtathatjuk vagy definiálhatjuk. E funkciókat egyetlen makro, a LIBRARY PROGRAM 50 látja el, amelynek tevékenysége a paraméter-listában kijelölt müvelettől függ.

Hivásnál valamennyi esetben két rekordra hivatkozunk, amelyeknek nem kell feltétlenül ugyanahhoz a file-hoz tartozniok. A hivatkozás itt is /és a továbbiakban mindenütt/ két paraméterrel -- a rekord/file megnevezésével és egy referencia-változóval /konstanssal/ -- történik. A két rekordon kivül ki kell jelölnünk a közöttük elvégzendő müveletet, valamint fel kell sorolnunk azokat a mezőpárokat, amelyekre a kijelölt művelet vonatkozik. Ezeknél a müveleteknél ugyanis általában már nem elegendő csak a teljes rekordra hivatkozni, mivel az esetek többségében csak egyes mezőkön kell elvégeznünk azokat. Ennek figyelembe vétele mellett a LIBRARY PROGRAM 50 paraméterlistája a következő elemeket tartalmazza:

1./ a művelet első operanduszát képező rekordra /R1/ való hivatkozás,
2./ a müvelet kijelölése /0/,
3./ a külső kapcsolat jelzése,
4./ hivatkozás a második rekord operanduszra /R2/,
5./ a mezőpárok /M1i, M2i/ felsorolása.

Ha a 3./ pontban emlitett paraméter értéke nulla, a makro hatására a felsorolt mezőpárokon végrehajtódik a kijelölt művelet és az eredmény aritmetikai müveletek esetén a mezőpár második tagjában tárolódik:

$$M2i \; 0 \; M1i \Longrightarrow M2i, \qquad i=1,\ldots.k.$$

Az egyes müveleteknél természetesen csak meghatározott tipusu mezők szerepelhetnek. Az összeadásnál pl, egyaránt meg van engedve a valós és az egész mezők használata, osztásnál viszont csak valós tipusokkal lehet dolgozni. A müveletek között nemcsak aritmetikai müveletek, hanem az átvitel és a mezők cseréje is szerepelhet, amelyeknek a hatása a következő:

$$M1i \Longleftrightarrow M2i \qquad i=1,\ldots.k,$$

illetőleg

$$M1i \Longrightarrow M2i \qquad i=1,\ldots,k.$$

Az utóbbi két müvelet -- ellentétben az aritmetikai müveletekkel /összeadás, kivonás, szorzás és osztás/ a string-tipusra is értelmezve van.

Ugyanez a makró ad lehetőséget a mezők összehasonlitásának eredménye szerint történő feltételes vezérlésátadásra is. Ekkor a müveleti jel helyén az "=" , "<", ">" relációjelek egyike áll, a paraméter-lista pedig kiegészül két cimkével. Ha az összes felsorolt mezőpárra teljesül a kijelölt reláció, a vezérlés az első cimkére adódik át, ellenkező esetben a másodikra.

A 3./ pontban szereplő paraméter azt a célt szolgálja, hogy egyrészt a tárgyprogram fordítása során könnyen kezelhető strukturáju paraméter-listához juthassunk, másrészt hogy necsak rekord-elemek között végezhessünk müveleteket, hanem az egyik komponens hagyományos AKI változó is lehessen. Amennyiben a paraméter értéke /n/ nullától különböző egész, a paraméter-lista megfelelő helyén szereplő hivatkozás nem mezőre, hanem egy egyszerü változóra, vagy konstansra /string-tipus esetén pedig tömb-kezdőelemre/ vonatkozik. A kijelölt hely az Mh /n/ paraméter helye lesz, ahol

$$h = \frac{3+\operatorname{sign} n}{2} \quad .$$

Ennek a lehetőségnek a kihasználása azonban csak akkor célszerü, ha csupán egy mezőre vonatkozik hagyományos változót tartalmazó müvelet. Ellenkező esetben ugyanis minden mezőre külön müveleti makrót kellene felirni. Egy teljes rekordnak tömbbe történő áttöltését, vagy tömbből történő visszatöltését a LIBRARY PROGRAM 70 segitségével végezhetjük el [18]. Az áttöltést követően a tömbelemekre minden olyan helyen /pl. kifejezésekben/ hivatkozhatunk, amelyet az AKI eredeti szintaxisa megenged.

Ennek az az előnye, hogy gyorsabb gépi müveletvégzést tesz lehetővé, továbbá megengedi az összetettebb strukturáju kifejezések felirását. Hátránya viszont, hogy magának a feldolgozást végző algoritmusnak /programrészletnek/ kell figyelembe vennie a rekord strukturáját.

Amennyiben ez a makro paraméterekként

1./ rekord/file nevet,
2./ referencia-változót /konstanst/ és
3./ tömb-kezdőelemet

tartalmaz, az áttöltés a tömbből a rekordba valósul meg. Ha a 2./ és a 3./ paraméterek közé egy olyan ujabb paramétert iktatunk be, amelynek csak a jelenléte hordoz információt, az előálló négy paraméteres alak a rekordból a tömbbe való áttöltést eredményezi.

5.2.2.3. Az elméleti tárgyaláskor nem játszottak lényeges szerepet, de a felhasználásnál igen fontosak az output-utasitások, amelyek a különböző periféria-egységeken való megjelenitést szolgálják. A mágnesszalaggal kapcsolatban /amely ebben a rendszerben lényegében a virtuális memória egyik elemét képezi/ nem beszéltünk outputról, mivel arra vonatkozóan a programozónak nem kell output-/ill. input-/ utasitásokat kiadnia.

Csak a sornyomtató berendezésre irányitott output-utasitással foglalkozunk részletesebben, mivel a többi perifériával kapcsolatban nem merülnek fel ujabb, elvi jelentőségü kérdések.

A nyomtatást a LIBRARY PROGRAM 43 segitségével végezzük. Egy rekord tetszőleges mezőit egyetlen utasitással irhatjuk ki, az általa vezérelt formátumban.

Amennyiben több rekordhoz tartozó elemeket kell ugyanabba a sorba kinyomtatni, a soremelést vezérlő paramétert nulla értékre állitjuk; ennek hatására csak a nyomtató pufferjének töltése hajtódik végre, de sem nyomtatás, sem soremelés nem történik. Ezt követően egy ujabb utasitás adásával kiegészithetjük a puffer előző tartalmát.

A nyomtatásnál nem szükséges a rekordnak a deklarációban megadott strukturájához ragaszkodnunk; a mezők kiirását tetszőleges poziciótól kezdődően kérhetjük. A tipusok természetesen nem módosithatók, de erre -- igen ritka kivételektől eltekintve -- nincs is szükség. Az egyetlen indokolt tipuskonverzió -- a valós-egész konverzió-- formálisan /nulla számu tizedesjegy kiiratásával/ megvalósitható.

A rekord-output utasitás paraméter-listája az alább felsorolt elemekből áll:

1./ rekord/file megnevezés,
2./ referencia-változó /konstans/,
3./ soremelések száma,
4./ valós számok nyomtatásánál a tizedesjegyek száma,
5./ pozició -- mezőnév párok felsorolása.

5.2.2.4. Ha input-, output-, valamint tömbökkel kapcsolatot létesitő utasitásról van szó, az elvégzendő müveletet maga az utasitás egyértelmüen meghatározza. A LIBRARY PROGRAM 50 esetén azonban ezt a tevékenységet -- bizonyos határokon belül -- a 0 paraméter irja elő. Ha a müveleti jelek használata engedélyezve lenne a paraméter-listában, semmiféle nehézség nem lépne fel. Igy azonban arra vagyunk utalva, hogy valamilyen változónévvel vagy paraméterértékkel jelöljük ki a kérdéses müveletet.

A rendszer szempontjából az lenne az egyszerübb megoldás, ha paraméterértékeket használnánk fel, éspedig ugy, hogy pl. konstans formájában a müvelet kódját irnánk a 0 paraméter helyére. Mivel azonban ez nehezen értelmezhető, szinte áttekinthetetlenné válna a program. Emiatt a változónév használata mellett döntöttünk, mely ugyan kényelmetlenebb a rendszer szempontjából, de azzal az előnnyel jár, hogy lehetővé válik a programozó által bevezetett mnemonikus kódok alkalmazása.

A LIBRARY PROGRAM 51 paraméter-listájában rögzitett sorrendben felsoroljuk a müveletek megnevezését, amelyek -- célszerüen -- skalárváltozókat jelölő azonositók. A paraméterlistában elfoglalt helyzet a jelölt változónak olyan jelentést ad, hogy a rá való hivatkozás a 0 müveleti kód helyen a kivánt müvelet végrehajtását eredményezi.
Az egyes paraméterek rendre a következők:

1./ egyenlőség-reláció azonositója
2./ kisebb-reláció "
3./ nagyobb-reláció "
4./ összeadás-jel "
5./ kivonás-jel "
6./ szorzásjel "
7./ osztásjel "
8./ átvitel "
9./ csere "

A müveleti jel azonositásához több lehetőség kinálkozott. Az első lehetőség az, hogy a LIBRARY PROGRAM 51 rendre az 1,2,....,9 egészeket adja érték gyanánt a paraméter-listában felsorolt skalárváltozóknak és ezek az értékek irányitják a LIBRARY PROGRAM 50-et a megfelelő müvelet elvégzésére.

Ezt azonban nem valósitottuk meg, mivel kettős hátránnyal jár. Egyrészt nincs ellenőrzési lehetőség; ha tévedésből más azonositót irunk a 0 helyébe és annak értéke a megadott tartományba esik, a müvelet végrehajtása megtörténik. Másrészt -- ugyancsak hibás programozás esetén -- előfordulhat, hogy a müveletet azonositó változónév a futás során más értéket kaphat, ami szintén nem szándékosan kijelölt müveletek végrehajtásához vezethet.

Egy másik lehetőség az, hogy a művelet azonositójának identifikálását nem a megfelelő skalárváltozó értéke, hanem a LIBRARY PROGRAM 51 felhivásánál található paraméter-listában elfoglalt helyzete alapján hajtjuk végre. Ez azonban valamennyi müveletvégzés esetére a paraméterlista helyének meghatározását és az azonositónak megfelelő gépi cim abban elfoglalt helyének megállapitását követelné.

Egy harmadik módszer -- és ez az, amit megvalósitottunk -- lényegét tekintve az előzők kombinációja. Egyesiti magában azok előnyeit, ti. lehetővé teszi, hogy a müveleteket azonositókkal jelöljük. Ez a müveleti utasitásban való felhasználáskor azonban már olyan egész értéket szolgáltat, amely kapcsolóban közvetlenül felhasználható, de maga a felhasználó -- a megengedett utasitásokkal -- nem tudja megváltoztatni. Ezáltal biztositva van a rendszerbeli kezelés egyszerüsége és a program áttekinthetősége, valamint az az előny, hogy megszünik a hibalehetőség.

Ugyanakkor viszont a müveleti kódokat deklaráló LIBRARY PROGRAM 51-nek utófordităsi tevékenységet is kell végeznie. Ehhez pedig az egész programban meg kell határoznia azokat a helyeket, ahol müveleti makróra vonatkozó felhivó utasitás található és az ezt követő paraméter-listában a harmadik paramétert ki kell cserélnie a saját paraméter-listája által meghatározott kódszámmal /tehát az 1,2,...,9 egész számok valamelyikével/.

Minthogy ez az érték bekerül a paraméter-listába, forrásnyelvi változóval a programban már nem módositható, tehát védett a felülirás ellen.

Vizsgáljuk most a felismerhetőség feltételeit. Automatikusan ismeretes az átvizsgálandó tartománynak /a forrásprogram képének/ a felső határa. Ezt az AKI tárgyprogramokban a mindenkori inditási cimen található vezérlésátadó utasitás tartalmazza a második cimrészen. Az első cimrészen a program végrehajtási kezdő cime van. Tehát, ha a forrásprogramra azt a megszoritást tesszük, hogy a START utasitásban megadott cimkéje /amelynél a program végrehajtása kezdődik/ előzze meg az összes rekord-müveleti utasitást, akkor a tartomány kezdetét és végét megadó információ ugyanabban a szabványrekeszben rendelkezésre áll.

Igy meghatározható a forrásprogramnak egy $Z \supseteq z = \bigcup_i z^i$ szelete.
Tekintettel arra, hogy minden esetben a LIBRARY PROGRAM 50 paraméterlistájában harmadik helyen álló, a LIBRARY PROGRAM 51-ben meghatározott azonositóju skalárváltozó képét /első cimrészen álló gépi cim/ kell felismerni, mind az s_t^i , mind a $\bar{g}^i$ azonosak minden i-re; és egyaránt könnyen megállapithatók. Az a tárgynyelvi szintaktikus egység, amelyben a keresett elem, mint kisebb egység elhelyezkedik, a LIBRARY PROGRAM 50 makro felhivó sorozata. Ha egy ilyen hivást elhelyezünk a felismerő makroban, -- természetesen ugy, hogy ne kerüljön végrehajtásra --, akkor megkapjuk a felhivó utasitás alakját is:

- 31 00 ⟨müveleti makro kezdőcime⟩ 0017 ,

ahol az első cimrész programonként változik. Ilyen alaku utasitás a tényleges felhivásokon kivül véletlenszerüen is előfordulhat a tárgyprogramban beépitett gépi kód blokkok konstansai között.

Ezért a gépi kód blokkok konstansaira vonatkozóan egy megszoritást kell tennünk: amennyiben a végrehajtás kezdetét jelölő cimke után előfordulnak, ott nem tartalmazhatják a fenti alaku konstanst. Az első cimrészre nem tudunk pontos értéket megadni, csak korlátokat, mert az a beépitéssel kapcsolatos átcimzés eredménye.

Ezzel a megszoritással elérjük, hogy Z egyenlő legyen z-vel, ami az előzőkkel együtt az 1. fejezetben tárgyaltak alapján már elegendő a felismerhetőséghez. A felsorolt feltételek nem jelentenek lényeges megszoritást a forrásprogramokra, s emiatt az utófordités feltételei a rendszer használhatóságának korlátozása nélkül biztosithatók.

5.3. Befejezésül röviden áttekintünk az egyes programok felépitésénél felmerülő néhány kérdést és foglalkozunk a tárgyprogram azon tulajdonságaival, amelyeket az egyes tevékenységekben kihasználtunk.

A deklarációs makro által előállitott DL-t a DM paraméterlistája helyére irjuk vissza. Ez abból a szempontból előnyös, hogy nem lép fel külön helyigény. Ugyanakkor azonban gondoskodnunk kell arról, hogy ne történhessen ismételt végrehajtás. Ezért a

- 31 00 <DM kezdőcime> 0017

felhivo utasitást a végrehajtás utolsó lépéseként kicseréljük egy olyan

- 30 00 C 0000

vezérlésátadó utasitásra, amelyben a C a paraméterlista után következő első cimet jelöli. A paraméter-lista általános szerkezete a 3.sz. ábrán látható. Az ily módon elhelyezett DL cimét /a 4.4.1. pontban irottaknak megfelelően/ a rekord/file azonositójához tartozó skalárváltozóba irjuk be a második cimre.

A skalárváltozó tárgyprogrambeli cime a forditást követően a paraméter-lista első helyén található az aktuális DM-hivásban. A DM tevékenysége a DL előállitására és a hivatkozási cim kitöltésére korlátozódik, mivel az ily módon deklarált változót csak uj müveletekben használjuk fel, tehát utóforditásra nincs szükség /lásd 4.4.3. pontot/. A DM forrásnyelvi és tárgynyelvi felhivósorozatát, ill. a DM tevékenységének eredményeként kapott DL szerkezetét a 6.sz. ábra szemlélteti.

A müveletvégzésnél, az inputnál és az outputnál a makrok munkaterületén kialakitott "regisztereket" használunk fel. A müveletvégzés a "regiszterekbe" áthelyezett információkon történik és a tényleges input, ill. output-tevékenység is ezekre vonatkozik. Az egyes utasitások hatását és a végrehajtás logikai sémáját a 7.sz. ábra szemlélteti.

Az emlitett müveleteknél kihasználjuk azt, hogy a forditó által kialakitott paraméter-listákhoz tartozó gépi szavak utasitáskódja +00, amelytől különbözik a felhivást követő bármely utasitás forditás során előálló kódja. Emiatt a paraméter-lista vége a paraméterek számának előzetes ismerete nélkül is minden esetben megállapitható.

A bevezetett uj változókra való hivatkozásoknál csupán az előbb emlitett "regiszterekbe" való áttöltést, ill. a belőlük való visszatöltést kellett megvalósitanunk. Ennek folyamatábrája a 8.sz. ábrán látható.

I R O D A L O M

[1] C.Strachey: A General Porpose Macrogenerator.The Computer J. vol.8, No.3./1965/, 225-241.

[2] I.Flores:Computer Software.Prentice Hall,Englewood Cliffs, N.J. /1965/

[3] MINSZK-22 dokumentáció, K-9.

[4] I.A. MacLeod: MP/1 -- a FORTRAN macroprocessor.The Computer vol.14, No.3. /1971/, 229-231.

[5] L.Bolliet: Compiler writing techniques.Programming Languages ed. by F. Genuys, Acad.Press, London - New York /1968/

[6] Revised Report on the Algorithmic Language ALGOL 60.Annual Review in Automatic Programming, vol.4. /1962/

[7] PL/1 Language Reference Manual.IBM Systems Reference Library, File No. S-360-29

[10] IBM System/360 Disc Operating System,Supervisor and I/O Macros.IBM Syst. Ref.Library, File No.S-360-30.

[12] Kalmár L.: Ist ALGOL wirklich eine algorithmische Sprache? Automatentheorie und formale Sprache. Bericht einer Tagung des Mathematischen Forschungsinstituts Oberwolfach. Oktober 1969. Mannheim /1970/. 305-315 old.

[13] Lőcs Gy.; Révész Gy.: Az ALGOL 60 nemzetközi algoritmikus nyelv. NIM IGÜSZI, Müszaki dokumentációs és fordító iroda, Budapest /1965/

[14] R.W.Watson: Timesharing System Design Concepts.McGraw-Hill, New-Yrok /1970/

[15] R.Gauthier,S.Ponto: Designing Systems Programs.Prentice Hall, Englewood Cliffs, N.J. /1970/

[16] Makay A.,Hunya P.: MINORG programozási nyelv a MINSZK-23-ra. Belső dokumentáció /1971/

[17] Collected Language Descriptions of 17 MOLs.IFIP/TC-2 Working Conference on Machine-Oriented Higher Level Languages, Trondheim /1973/

[18] Lázár Zs.: Az AKI forditónak az INZSENYER autokód szintaxisát érintő fejlesztéséről. Belső dokumentáció /1972/

[21] Kalmár L. et al.: Az elektronikus, digitális számitógépek eddigi fejlődése és a várható fejlődés fő irányai. Tanulmány. Kézirat, Szeged /1972/

[22] P.J.Brown: A Survey of Macroprocessors in Automatic Programming, vol.6, Pergamon Press Ltd., Oxford /1969/.

[23] R.M.Davis: Programming Language Processors.Advances in Computers, vol 7. /1966/

[24] C.A.R.Hoare: Record Handling. Programming Languages, ed.by F.Genuys, Acad.Press,London-New-York /1968/

[25] A. van Wijngaarden, B.J. Mailloux, J.E.L. Peck and C.H.A. Koster: Final draft report on the algorithmic language ALGOL 68. Amsterdam /1968/.

/A felsorolásban nem szerepelnek a [8] , [9] , [11], [19] és [20] sorszámok/

1. sz. melléklet

Skalárváltozó; egyszerü változó:

⟨skalárváltozó⟩ :: = ⟨azonositó⟩

⟨egyszerü változó⟩ :: = ⟨skalárváltozó⟩ | ⟨számindexü tömbelem⟩

⟨számindexü tömbelem⟩ :: = ⟨tömbnév⟩/⟨természetes szám⟩/ ⟨tömbnév⟩/⟨term.szám⟩ , ⟨term.szám⟩/

⟨tömbnév⟩ :: = ⟨azonositó⟩

Könyvtári program forrásnyelvi hivása:

⟨forrásnyelvi hivás⟩ :: = LIBRARY PROGRAM ⟨sorszám⟩ ((⟨paraméterlista⟩))

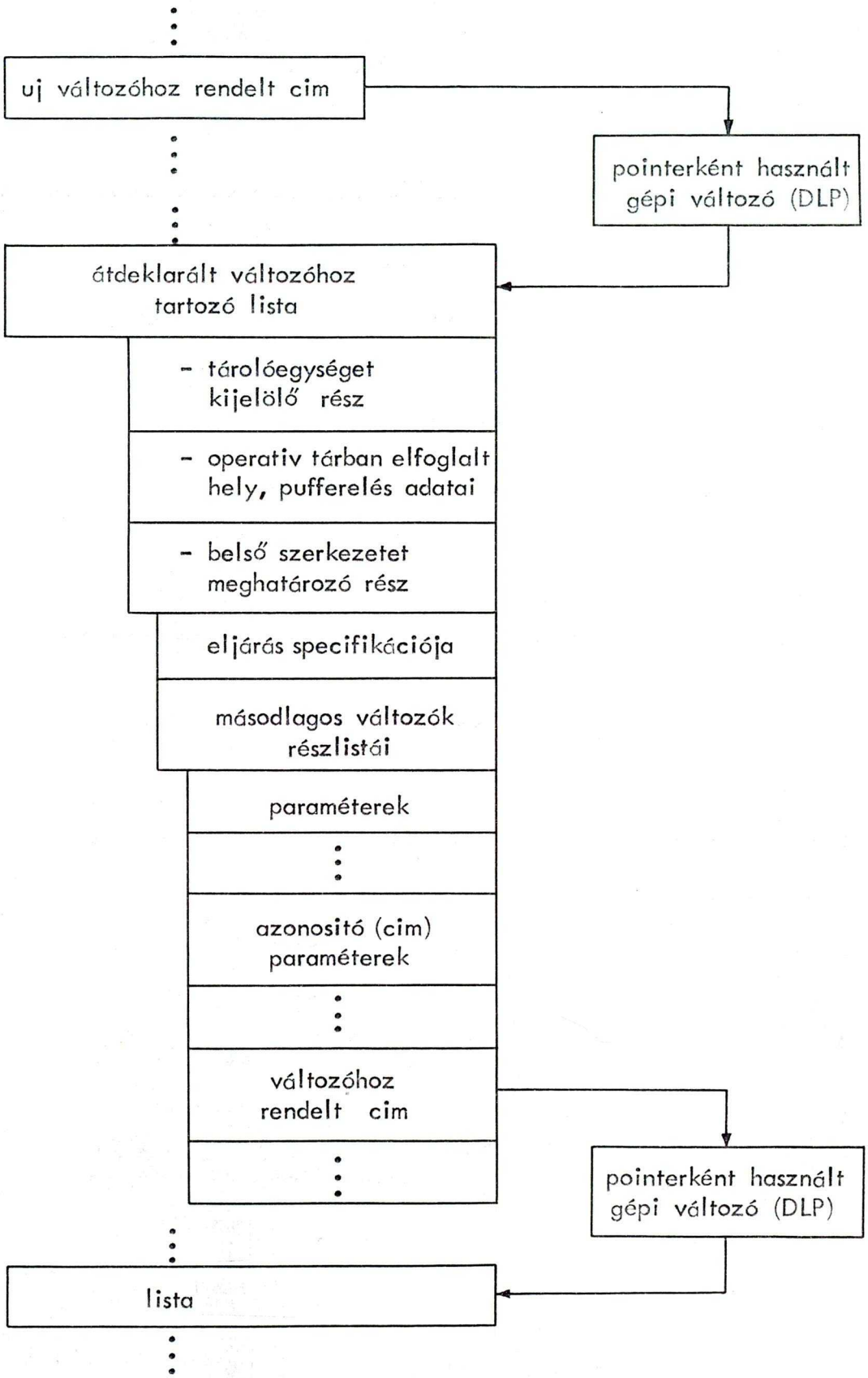
⟨paraméterlista⟩ :: = ⟨paraméter⟩ | ⟨paraméterlista⟩ , ⟨paraméter⟩

⟨paraméter⟩ :: = ⟨egyszerü változó⟩ | ⟨cimke⟩ | ⟨konstans⟩

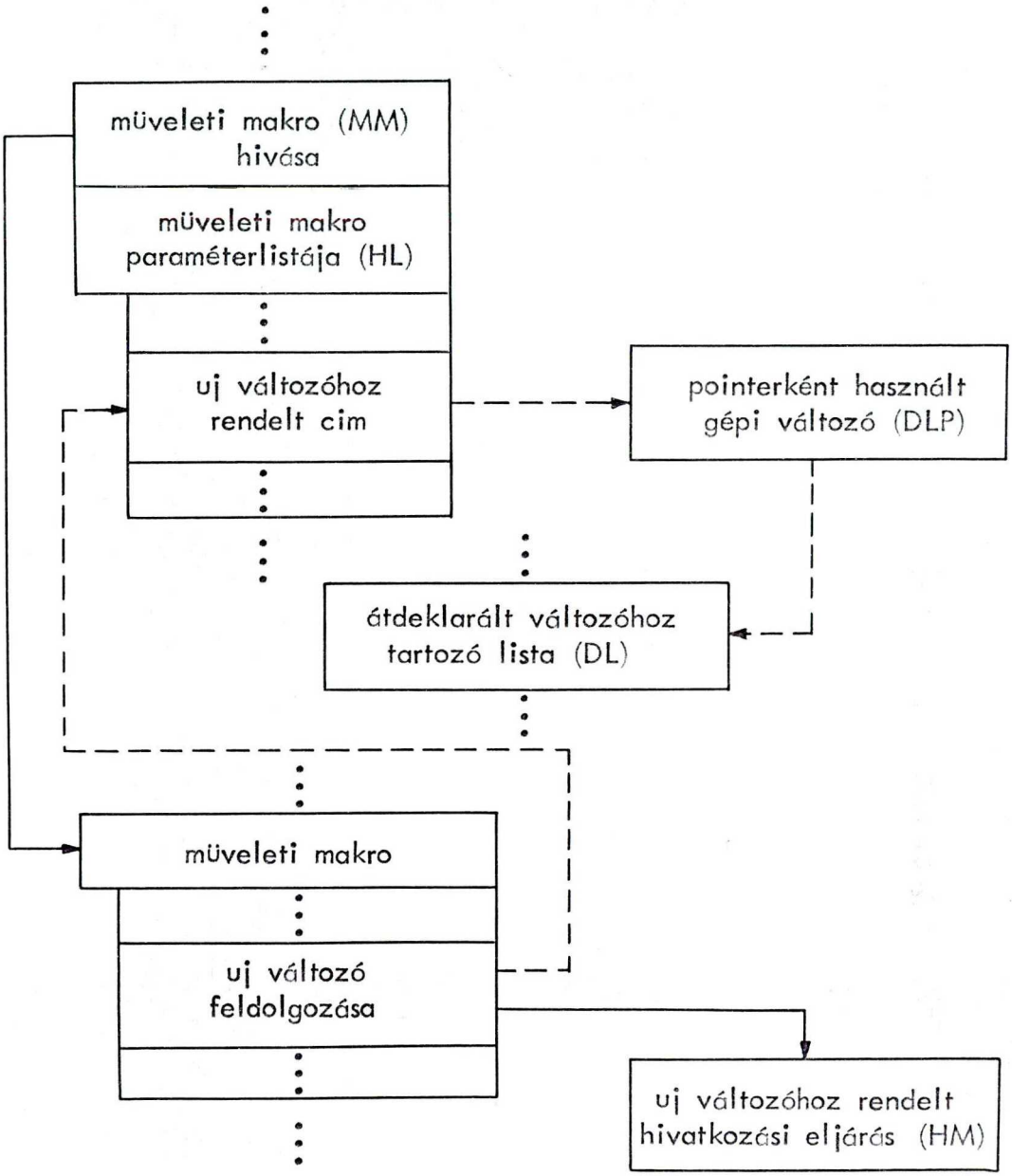
⟨konstans⟩ :: = ⟨egész szám⟩ | ⟨valós szám⟩

⟨cimke⟩ :: = ⟨természetes szám⟩

A ⟨természetes szám⟩, ⟨egész szám⟩ , ⟨valós szám⟩ és ⟨azonositó⟩ fogalmakat a szokásos értelemben használjuk.



1. ábra A deklarációval előállitott lista és hivatkozási rendszer sémája

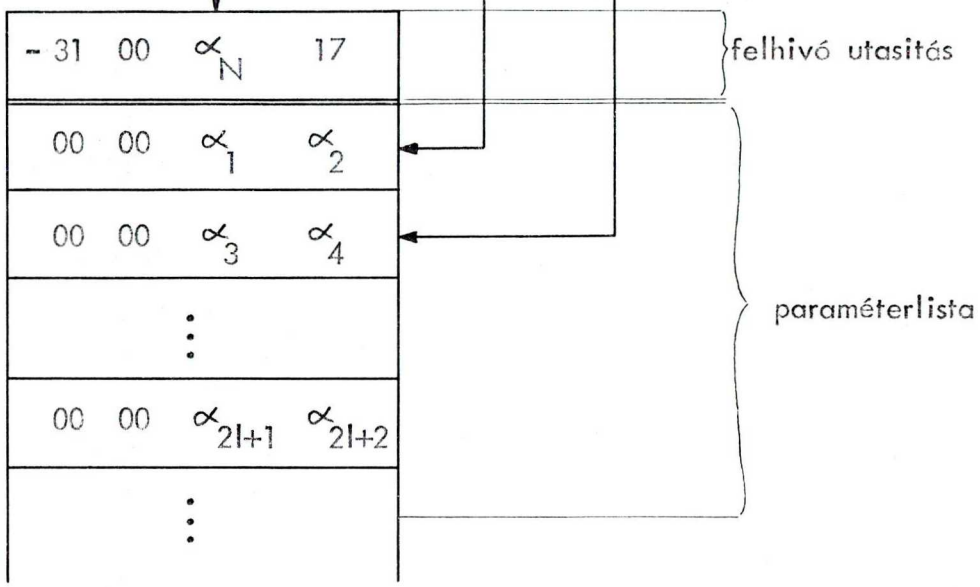


2. ábra A hivatkozás folyamatának logikai vázlata

FORRÁSNYELVI HIVÁS :

LIBRARY PROGRAM N (P_1, P_2, P_3, P_4, ..., P_k)*

TÁRGYNYELVI KÉP :



N : könyvtári program sorszáma

P_i : egyszerü változó (skalár v. számindexü tömbelem), ill. konstans ($i = 1, 2, \ldots, k$)

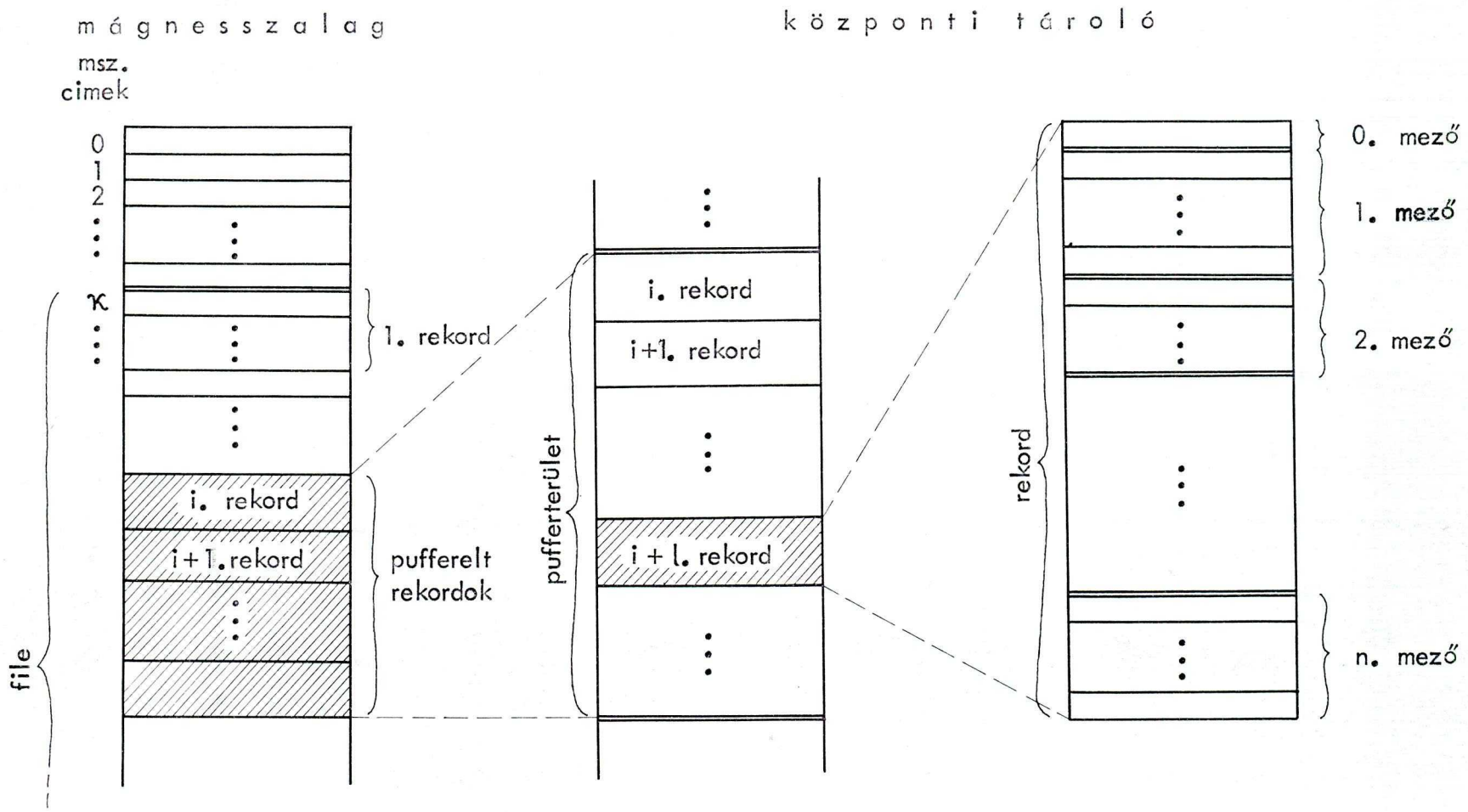
- 31 : szubrutinhivó utasitás kódja

α_N : az N sorszámu könyvtári program cime a szerkesztés után

17 : szabványos indexrekesz a visszatéréshez

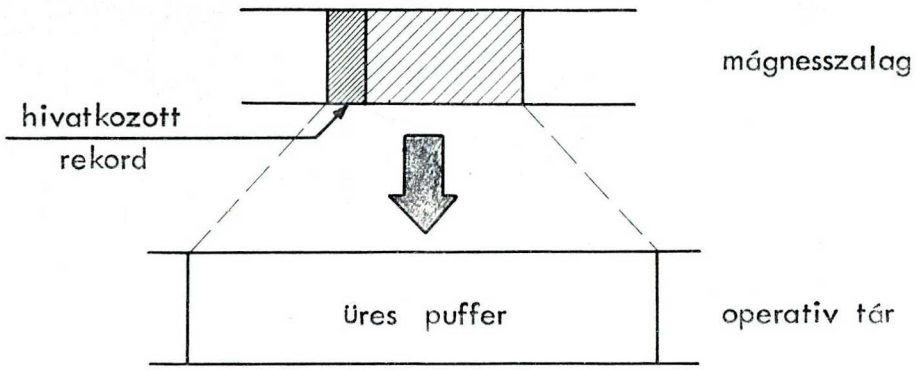
α_i : a P_i gépi cime a forditás után

3. ábra A könyvtári makro hivásának forrásnyelvi és gépi reprezentációja

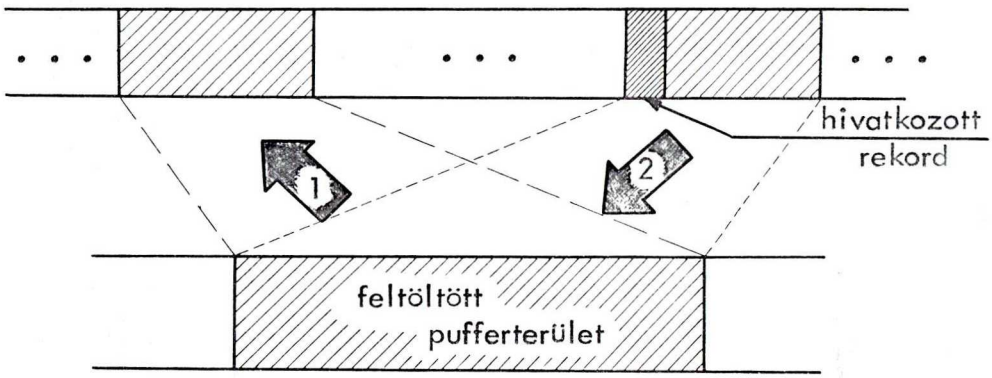


4. ábra File - pufferterület - rekord kapcsolata

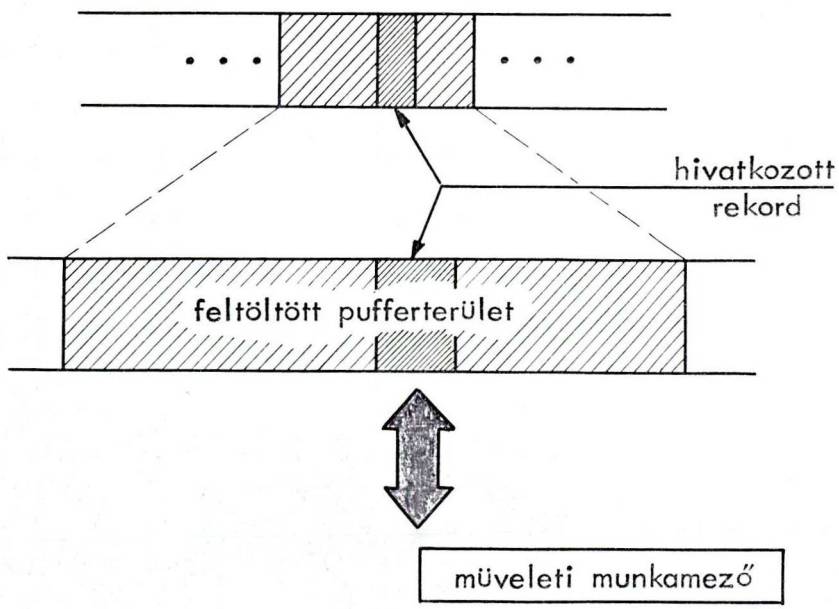
A) ÜRES PUFFER



B) HIVATKOZOTT REKORD NINCS A PUFFERBEN (FELTÖLTÖTT PUFFER)



C) HIVATKOZOTT REKORD A PUFFERBEN



5. ábra Pufferkezelés vázlata

A) DM FORRÁSNYELVI HIVÁSA

LIBRARY PROGRAM 37 (rekord/file név, mezőszám, puffer kezdete, puffer vége, msz. egység logikai sorszáma, msz. kezdőcim, mezőtipus/hossz, mezőnév,... ..., mezőtipus/hossz, mezőnév) *

B) DM TÁRGYNYELVI HIVÁSA

cim	müv. kód	index	I. cimrész	II. cimrész
α	- 31	00	⟨rek/file név⟩	17
$\alpha + 1$	+ 00	00	⟨rek/file név⟩	⟨mezőszám⟩
$\alpha + 2$	+ 00	00	⟨msz. e. log. sorszáma⟩	⟨msz. kezdőcim⟩
$\alpha + 4$	+ 00	00	⟨mezőtipus/hossz⟩	⟨mezőnév⟩
⋮	⋮	⋮	⋮	⋮
$\alpha + p$	+ 00	00	⟨mezőtipus/hossz⟩	⟨mezőnév⟩

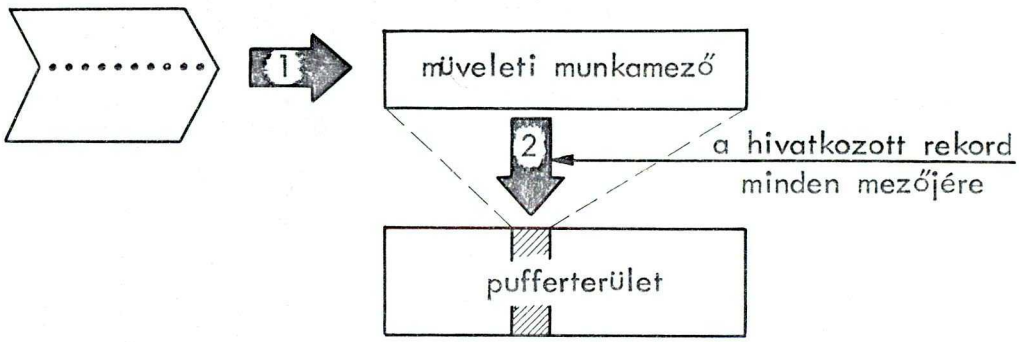
C) DEKLARÁCIÓS LISTA (DL)

α	- 30	00	$\alpha + p + 1$	0
$\alpha + 1$	+ pufferbeli első rek. sorszáma (17 bit)		puffer kezdőcime (13 bit)	mezők száma (6 bit)
$\alpha + 2$	+ puffer effektiv hossza (12 bit)		pufferbeli utolsó rek. sorszáma (18 bit)	rek. hossza (6 bit)
$\alpha + 3$	+	msz. egység log. sorszáma (2 bit)	msz. kezdőcim (17 bit)	
$\alpha + 4$	+ mezőtipus		mező hossza	⟨mezőnév⟩
⋮	⋮		⋮	⋮
$\alpha + p$	+ mezőtipus		mező hossza	⟨mezőnév⟩

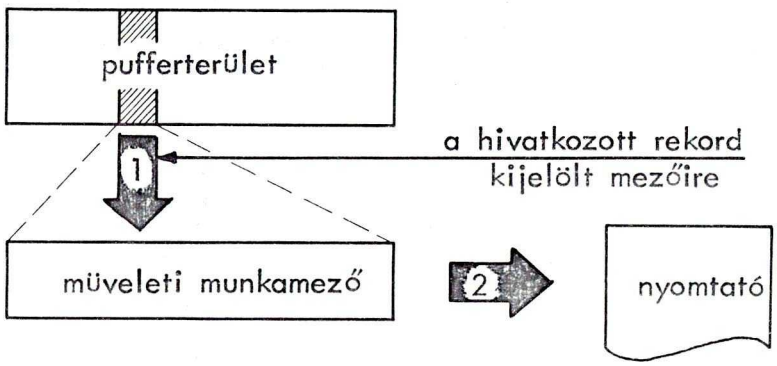
	⟨mezőnév⟩	:	+ mezőtipus	mező hossza	⟨mezőnév⟩
DLP =	⟨rek/file név⟩	:	+ 0	0	$\alpha + 1$

6. ábra DM forrásnyelvi és tárgynyelvi hivása és a deklarációs lista szerkezete

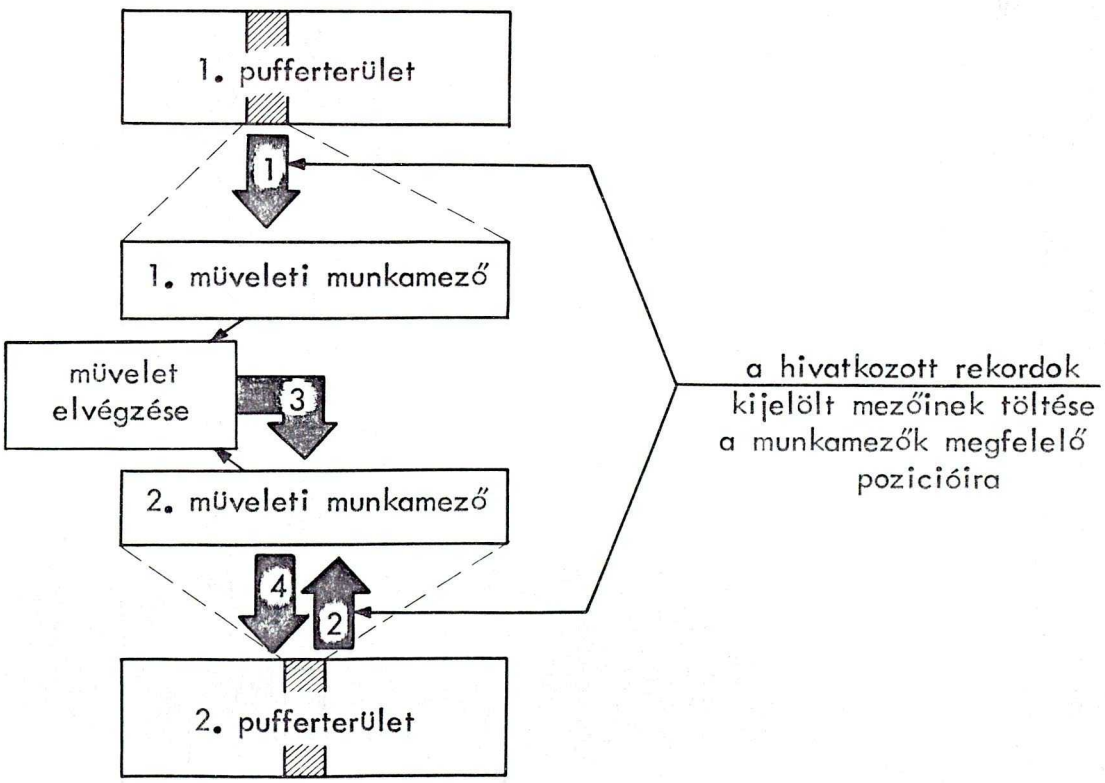
A) INPUT



B) OUTPUT



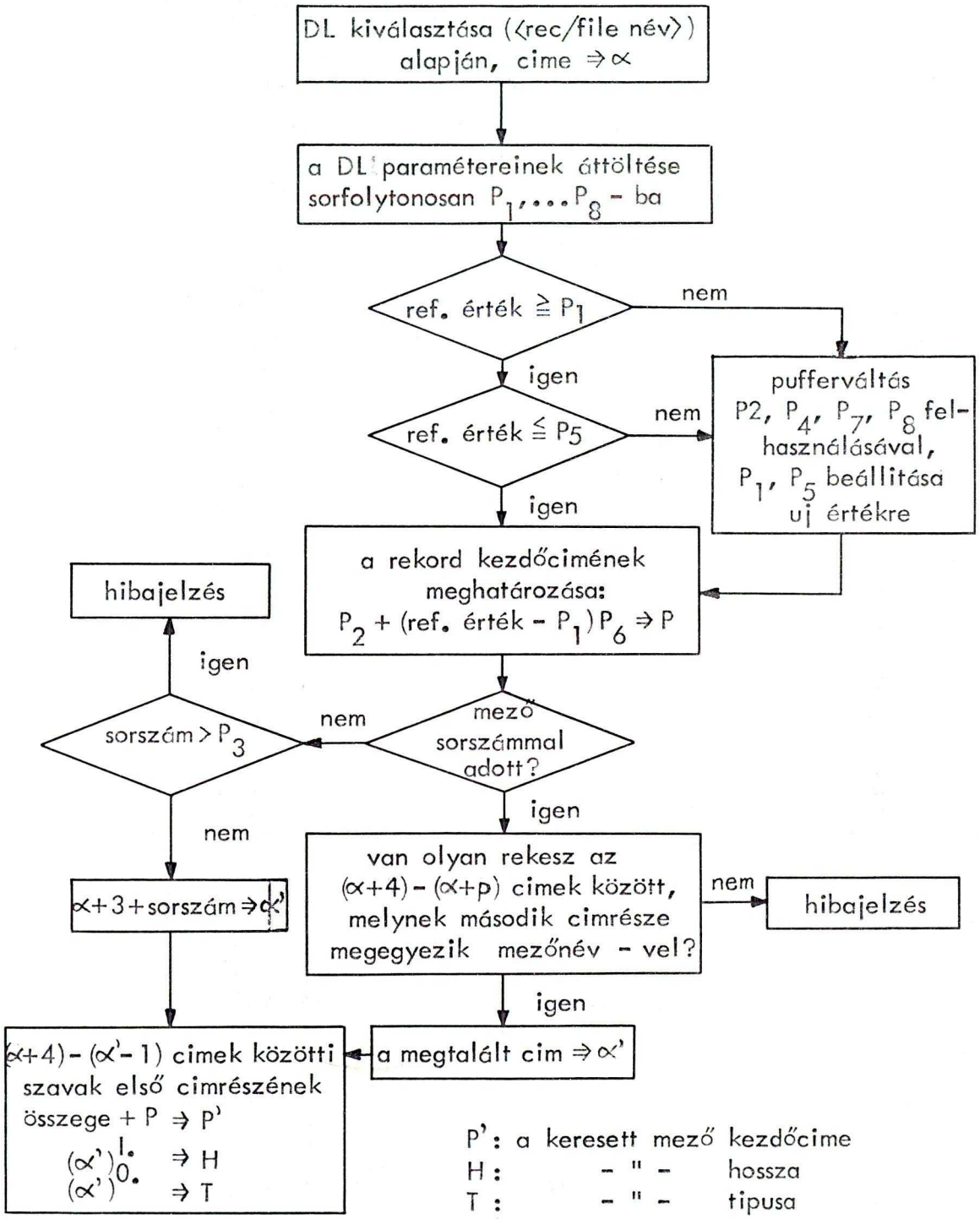
C) BELSŐ MÜVELETEK



7. ábra Input, output és a belső müveletek szervezése

Hivatkozáshoz megadott információk:

- rec/file név
- referencia érték
- mező név/sorszám



8. ábra Kijelölt mező meghatározására szolgáló eljárás folyamatábrája