stichting
mathematisch
centrum

MC

LR 2.3

JUNI 1971

D. GRUNE

HANDLEIDING BIJ DE INLEESPROCEDURE 'READVAR'

2e boerhaavestraat 49 amsterdam

1. Doel.

Met de procedure readvar kunnen gegevens van een invoermedium gelezen worden, die gesteld zijn als ALGOL-achtige assignment statements; het verband tussen variabelen op het invoermedium en variabelen in het programma wordt gelegd door strings die door de programmeur gespecificeerd worden. Deze faciliteit is vergelijkbaar met (maar uitgebreider dan) de NAMELIST in FORTRAN en DATA-oriented input in PL/I.

2. Inleiding.

2.1. Overwegingen.

2.1.1. Vele programma's beginnen met het inlezen van een aantal basisgegevens. Vaak zijn een aantal van deze basisgegevens van keer tot keer gelijk, en zijn ze slechts in de lijst van in te lezen grootheden opgenomen, omdat men de mogelijkheid niet wil uitsluiten, ook eens andere waarden te gebruiken. Men zou dus de mogelijkheid willen hebben, op de getallenband aan te geven dat deze of gene variabele deze keer niet behoeft te worden ingelezen; het liefst zou men dit willen aangeven door niets te schrijven.

2.1.2. Bij de gebruikelijke wijze van invoer van gegevens wordt de betekenis van een getal alleen bepaald door zijn plaats op de band. In hogere programmeertalen is het echter gebruikelijk gegevens te identificeren door namen; volgens dit principe zou dus het inlezen van gegevens op het invoermedium de vorm van een assignment kunnen hebben, b.v. gewicht:= 1500.

2.1.3. Wellicht zal men bij het uitvoeren van assignments vanaf het invoermedium gebruik willen maken van reeds berekende gegevens; ook in de rechter kant van de assignments moeten dus namen toegestaan zijn, en, wil men er iets niet-triviaals mee kunnen doen, ook aritmetische operaties.

2.1.4. Het moet mogelijk zijn booleans en strings in te lezen.

2.2. Implementatie.

De procedure readvar kan, in overeenstemming met bovenstaande overwegingen, assignments van aritmetisch en boolean type lezen. In de hierin voorkomende expressies zijn de monadische operatoren +, − en ¬ toegestaan benevens de diadische operatoren +, −, ×, /, ∧, en ∨. Om wille van de eenvoud is voor de expressies een grammatica gebruikt (zie 6.) die iets afwijkt van die van ALGOL 60.

Verder kan de procedure readvar procedure-aanroepen lezen die als enige parameter een string hebben (string transfers); deze string wordt dan ingevoerd.

De statements worden op het invoermedium gescheiden door puntkomma's, terwijl het gehele <data block> (de eenheid die door een aanroep van readvar gelezen wordt) begint met begin en eindigt met end. Het ontbreken van invoergegevens kan eenvoudig aangeduid worden door begin end. Een <data block> kan er bij voorbeeld als volgt uitzien:

```
comment gegevens d.d. 1 - 1 - '71, Q5;
begin   kopje("Inkoop 1-1-'71");
        gewicht:= 850;   prijs:= 7140.00;
        aantal:= benodigd - 40
end
```

Voor een uitgebreidere beschrijving van <data block> zie 4..
Het verband tussen variabelen op het invoermedium en variabelen in het programma wordt aangegeven door een procedure die als actuele parameter aan de aanroep van readvar wordt meegegeven. Zie verder 3..

2.3. De procedure heading luidt:

procedure readvar(varspec, error); procedure varspec, error;

De procedure varspec wordt gebruikt voor het specificeren van het verband tussen variabelen "binnen" en variabelen "buiten"; zie 3.. De procedure error wordt gebruikt voor het afleveren van foutmeldingen; zie 5..

3. Het correspondentiemechanisme.

Aangezien het niet mogelijk is de ALGOL 60-compiler namen uit het programma in toegankelijke vorm te laten onthouden, moet de programmeur zelf opgeven welke variabele in het programma correspondeert met welke naam op het invoermedium, waarbij de naam in de invoer gegeven moet zijn als string. Er moet dus in het programma een aantal paren van identifiers en strings (b.v. (‡num‡,num), (‡export‡,export), (‡gewicht‡, weight),...) aan readvar meegegeven worden, waarbij num, export en weight eventueel van verschillend type kunnen zijn. Dit geschiedt als volgt.
Als eerste (actuele) parameter wordt aan readvar een procedure meegegeven, die we, naar de formele parameter, ook varspec zullen noemen. De procedure varspec heeft als enige parameter een procedure die we verder pair zullen noemen. De taak van varspec is nu, de (meegekregen) procedure pair achtereenvolgens aan te roepen voor alle paren van strings en corresponderende variabelen. In het bovengenoemde voorbeeld zou varspec dus de vorm kunnen hebben:

```
procedure varspec(pair); procedure pair;
begin   pair(‡num‡, num);
        pair(‡export‡, export);
        pair(‡gewicht‡, weight)
end varspec;
```

Wanneer readvar nu een naam op het invoermedium aantreft, roept hij de procedure varspec aan, met een van readvar's interne procedures als parameter; deze procedure (die dus correspondeert met pair) wordt nu een aantal malen door varspec aangeroepen, en vergelijkt bij elke aanroep de als eerste parameter ontvangen string met de zojuist gelezen naam op het invoermedium. Wordt de juiste string gevonden, dan wordt de tweede parameter van de onderhavige aanroep van pair via een andere interne procedure doorgegeven naar de plaats van bestemming.
Gedurende deze handelingen is het type van tweede parameter onbekend, met alle gevolgen van dien:

a. Als de variabele alleen in het rechter lid van een assignment voorkomt of als procedurenaam gebruikt wordt om een string in te lezen, dan vindt geen enkele controle plaats, en kan men zowel zinvolle constructies als

```
if m>0 then a+b else a-b
```

als ook zinloze constructies als switch- of array-identifiers meegeven; het effect van dergelijke zinloze constructies is ongedefinieerd (zie ook 5.3.).

b. Komt de variabele in het linker lid voor (en moet er dus aan geassigneerd worden), dan voert het ALGOL 60-systeem een controle uit, waarbij op assigneerbaarheid getest wordt. Als de tweede parameter van pair dan geen variabele is waaraan geassigneerd kan worden, wordt het programma afgebroken met systeem-foutmelding 500 in de interne procedure 'set'.
c. Is de parameter wel assigneerbaar, maar van verkeerd type (b.v. boolean in plaats van integer), dan is het effect ongedefinieerd (zie ook 5.3.).

Bij het vergelijken van de string uit de aanroep van pair met de naam op het invoermedium worden spaties, tabs en nlcr's buiten beschouwing gelaten.

De procedure varspec wordt voor elke keer dat een naam op het invoermedium voorkomt eenmaal aangeroepen. In principe is men vrij varspec ook andere dingen te laten doen; b.v. kan men afhankelijk van voorgaande resultaten bepaalde variabelen met andere strings laten corresponderen, een telling bijhouden, etc. Hierbij is echter wel de nodige omzichtigheid geboden.

Variabelen die als tweede parameter van pair optreden en op het invoermedium alleen aan de rechterkant van assignments mogen voorkomen, kunnen tegen assignering beschermd worden door ze tussen haakjes te zetten. Een eventuele poging om toch aan zo'n variabele te assigneren leidt tot een foutmelding, zoals beschreven in 5.2.1..

4. Het <data block>.

Een <data block> is een eenheid die door een (1) aanroep van readvar gelezen wordt. Voor de grammatica zie 6.. Hieronder volgt een informele beschrijving.

4.1. In het <data block> worden spaties, tabs en nlcr's overal geskipt, behalve in strings en in de symbolen begin, end, true, false en comment; layoutsymbolen mogen dus voorkomen in namen, getallen, het wordt-symbool (:=) en het aanhalingsteken binnen strings (""). Bij het vergelijken van een naam in de invoer met een string uit het programma worden van de naam en de string hoogstens de eerste 80 symbolen beschouwd.

4.2. Een <data block> bestaat uit begin en end met daartussen een aantal statements, van elkaar gescheiden door puntkomma's. Het laatste symbool dat door readvar verwerkt wordt is het symbool onmiddellijk na end (bij voorbeeld een nlcr). Een statement kan een assignment zijn (voor reals, integers en Booleans) (zie 4.3.) of een string transfer (voor strings) (zie 4.4.) of kan leeg zijn (zie 4.5.). Assignments worden in volgorde uitgevoerd, en de resultaten van de ene assignment kunnen in de volgende assignment meteen gebruikt worden. De constructie

begin x:= 3; x:= x×x; x:=x×x end

laat dus in x de waarde 81 achter.

4.3. Assignments.
Een assignment is een ALGOL 60-assignment met de volgende afwijkingen:

4.3.1. Als operatoren kunnen voorkomen +, − en ¬ (monadisch) en +, −, ×, /, ∧ en ∨ (diadisch).

4.3.2. Alleen ongeindiceerde variabelen zijn toegestaan.

4.3.3. if-expressies zijn niet toegestaan.

4.3.4. een diadische operator mag gevolgd worden door een monadische: vormen als p:= y×-8 zijn toegestaan.

4.3.5. Een assignment kan zelf weer als operand gebruikt worden; de waarde is die van het rechter lid. Naast de bekende vormen als x:=y:=c zijn ook vormen als x:=(y:=c) en x:=(y:=c)+d toegestaan.

4.3.6. Naast de uit ALGOL 60 bekende getallen (b.v. $8.1_{10}-7$) mogen in aritmetische assignments ook octale getallen voorkomen. Deze bestaan uit een rij cijfers tussen 0 en 7 voorafgegaan en gevolgd door een apostrof. Deze rij cijfers wordt beschouwd als de octale representatie van een X8-woord, waarbij eventueel links nullen toegevoegd worden. De waarde van het octale getal is de integer waarde van dit X8-woord.

B.v.	octaal	waarde
	'60'	48
	'777777772'	-5

4.4. String transfers.

Een string bestaat uit een (eventueel lege) rij symbolen tussen aanhalinstekens ("); het symbool " zelf kan in de string voorgesteld worden door "". Evenals in ALGOL 60 mag een in te voeren string alleen als parameter in een procedure-aanroep voorkomen. Met de naam van deze procedure op het invoermedium moet dan in het programma de naam corresponderen van een procedure met een enkele parameter van het type integer. Deze procedure wordt nu door readvar aangeroepen met de interne representaties van de opeenvolgende symbolen uit de string als parameter; hierbij worden de buitenste aanhalingstekens buiten beschouwing gelaten.
Als b.v. op de invoer de aanroep

```
print("text");
```

voorkomt, en met de naam print de procedure prsym correspondeert, dan worden achtereenvolgens de aanroepen prsym(29); prsym(14); prsym(33); prsym(29); uitgevoerd.

4.5. Statements mogen leeg zijn: b.v. begin end of begin x:= 3;; end.

4.6. Uitbreidingen van de grammatica.

4.6.1. In en ook voor het <data block> mag overal buiten strings commentaar voorkomen, ingeleid door het symbool comment en afgesloten door een puntkomma.

4.6.2. In plaats van begin mag men b schrijven, voor end e, voor true t, voor false f en voor comment c. De invoer voor een lege aanroep van readvar kan dus geschreven worden als b e.

4.6.3. De vorm (<assignment>)X mag vervangen worden door <assignment>X waarbij X een puntkomma, een haakje-sluiten of een end-symbool voorstelt.
Toelichting:
In de grammatica moeten alle assignments binnen expressies omgeven zijn door haakjes; multiple assignments ontstaan dus in de vorm x:=(y:=(z:=3)); volgens de uitbreiding 4.6.3. mogen we de haakjes hierin weglaten. Met andere woorden: de expressie na het :=-symbool loopt tot een), een ; of een end.

4.7. Representatie van symbolen.

De symbolen mogen op het invoermedium slechts voorkomen in de representatie waarin ze in deze beschrijving genoemd worden. Zo mag begin niet geschreven worden als 'begin', ∧ niet als and, ; niet als ., , ⏨ niet als e, enz..

5. Foutmeldingen.

Er kunnen bij een aanroep van readvar drie soorten fouten optreden: die welke door readvar ontdekt worden, die welke door het ALGOL 60-systeem ontdekt worden en die welke niet ontdekt worden.

5.1. Fouten ontdekt door readvar.

Deze fouten worden gemeld door een aanroep van de procedure error (tweede parameter van readvar), welke als enige parameter het foutnummer meekrijgt. Deze procedure kan dan maatregelen nemen en eventueel normaal naar readvar terugkeren; de gevonden fout wordt dan zo mogelijk provisorisch hersteld (met alle risico's van dien) en het lezen wordt voortgezet. Algemeen geldt dat indien bij het lezen van een expressie een fout wordt gevonden, de betreffende assignment niet uitgevoerd wordt. De assignment n:= m + true roept de procedure error aan met 5 als parameter en laat n ongewijzigd.

De foutnummers hebben de volgende betekenis:

1 data block begint niet met begin
2 assignment niet met ; of end afgesloten
3 variabele in linker lid van assignment onbekend
4 variabele in expressie onbekend
5 in aritmetische expressie komt iets van type boolean voor
6 in boolean expressie komt iets van aritmetisch type voor
7 geen := waar verwacht
8 primary begint met ontoelaatbaar symbool
9 in expressie komt onbekend onderstreept symbool voor
10 in expressie ontbreekt een haakje-sluiten
11 octaal getal niet met apostrof afgesloten
12 octaal getal overschrijdt de integercapaciteit
13 in een getal volgt op +, - of . geen cijfer
14 in een getal volgt op $_{10}$ geen +, - of cijfer
15 procedurenaam in een string transfer onbekend
16 in string transfer ontbreekt een haakje-sluiten
17 geen string waar verwacht

5.2. Fouten ontdekt door het ALGOL 60-systeem.

5.2.1. Bij het optreden van een assignment van het type n:=<expressie> waarbij met n een vorm correspondeert waaraan niet geassigneerd kan worden, wordt het programma beeindigd met een systeem-foutmelding 500 in de interne procedure 'set'.

5.2.2. Het voorkomen van een string transfer van de vorm proc(<string>) waarbij met proc niet een procedure met een (1) integer parameter correspondeert, kan vele foutmeldingen veroorzaken.

5.3. Fouten die niet ontdekt worden.

5.3.1. Fouten van het type n:=<Boolean expression> wanneer met n een aritmetische variabele correspondeert, of b:=<arithmetic expression> wanneer met b een boolean variabele correspondeert. De assignment wordt uitgevoerd met een ongedefinieerde waarde.

5.3.2. Fouten van het type n:=p wanneer met p een vorm correspondeert die niet geevalueerd kan worden (b.v. switch, procedure etc.). De assignment wordt uitgevoerd met een ongedefinieerde waarde.

6. Grammatica.

6.1.

<data block> ::= begin <statements> end

<statements> ::= <statement> |
<statement>;<statements>

<statement> ::= <empty> | <actual statement>

<actual statement> ::= <assignment> |
<string transfer>

<assignment> ::= <variable> := <expression>

<expression> ::= <term> |
<expression> + <term> |
<expression> - <term> |
<expression> ∨ <term>

<term> ::= <factor> |
<term> × <factor> |
<term> / <factor> |
<term> ∧ <factor>

<factor> ::= <primary> |
+ <primary> |
- <primary> |
¬ <primary>

<primary> ::= <variable> |
<unsigned number> |
<octal number> |
<logical value> |
(<assignment>) |
(<expression>)

<variable> ::= <identifier>

<octal number> ::= '<octal integer>'

<octal integer> ::= <empty> |
<octal integer> <octal digit>

<octal digit> ::= 0|1|2|3|4|5|6|7

<string transfer> ::= <variable>(<string>)

<string> ::= "<string items>"

<string items> ::= <empty> |
<string item> <string items>

<string item> ::= <non-quote symbol> | ""

6.2. Voor de in het bovenstaande niet gedefinieerde termen <identifier>, <unsigned number>, <logical value> en <empty> raadplege men het Revised Report on the Algorithmic Language ALGOL 60; de term <non-quote symbol> kan elk symbool voorstellen met uitzondering van het aanhalingsteken (").

6.3. Samentrekkingen. (Zie ook 4.6.3.).

(<assignment>)) => <assignment>)

(<assignment>); => <assignment>;

(<assignment>) end => <assignment> end

Verder kunnen de onderstreepte woorden afgekort worden tot hun eerste letter.

7. Voorbeeld.

7.1. Programmadeel:

```
begin   integer number, required; real weight, price;
        Boolean calc;

        procedure varspec(pair); procedure pair;
        begin   pair(‹kopje›, prsym);
                pair(‹aantal›, number);
                pair(‹benodigd›, (required));
                pair(‹gewicht›, weight);
                pair(‹prijs›, price);
                pair(‹ber›, calc)
        end varspec;

        number:= -10000; required:= 144;
        weight:= 800; price:= 7000; calc:= true;

        readvar(varspec, errormessage);
        if number <0 then
        begin printtext(‹geen aantal vermeld›); exit end;

        process(number, required,  weight, price);
        if calc then calculate (number,price)
end
```

7.2. Invoerband:

```
c gegevens d.d. 1 - 1 - '71, Q5;
begin   kopje("Inkoop 1-1-'71");
        gewicht:= 850; prijs:= 7140.00;
        aantal:= benodigd - 40
end
```

7.3. Toelichting.

De procedure-declaratie varspec geeft readvar de beschikking over de namen kopje, aantal, benodigd, gewicht, prijs en ber, welke overeenkomen met resp. prsym, number, required, weight, price en calc; verder mag required niet gewijzigd worden. Aan required, weight, price en calc worden veronderstelde (default) waarden geassigneerd, terwijl number een "onmogelijke waarde" krijgt. Door de aanroep van readvar wordt eerst de tekst Inkoop 1-1-'71 afgedrukt, waarna weight en price de waarden 850 en 7140 krijgen, terwijl de waarde van number 104 wordt. De waarden van required en calc worden niet gewijzigd. Na een test of inderdaad aan number geassigneerd is, worden verdere berekeningen uitgevoerd.

8. Tekst van de procedure readvar.

```
procedure readvar(varspec,error); procedure varspec,error;
begin   comment procedure for reading variables according to name.
                Version:  24 - 6 - 1971

                The procedure uses the following resym values:
                        64 +    80 ∧            98 (
                        65 -    88 .            99 )
                        66 ×    89 ₁₀           118 tab
                        67 /    90 :            119 nlcr
                        70 =    91 semicolon    120 '
                        79 ∨    93 space        121 "
                                                126 _
                ;
        integer last char, stock, true, false, begin, end, comment,
                maxint, type;
        Boolean asgok;
        integer array name[0:79];   real ar;   Boolean bool;

        integer procedure nxt sbl;
        begin   if stock > 0 then
                begin last char:= stock; stock:= -1 end else
        skl:    last char:= resym;

                if layout(last char) then goto skl;
                if last char = 126 then
                begin
                und1:   last char:= resym;
                        if last char = 126 then goto und1;
                        if letter then last char:= last char + 256
                                  else last char:= 511;
                skip:
                        stock:= resym;
                        if stock = 126 then
                        begin
                        und2:   stock:= resym;
                                goto if stock = 126 then und2 else skip
                        end skip
                end und1;

                if last char=comment then
                begin
                comm:   nxt sbl;   if last char ≠ 91 then goto comm;
                        nxt sbl
                end comm;

                nxt sbl:= last char
        end nxt sbl;

        procedure dummy(ch);  ;

        Boolean procedure letter;
        letter:= last char ≥ 10 ∧ last char ≤ 62;
```

```
Boolean procedure layout(sym); value sym; integer sym;
layout:= sym = 93 ∨ sym = 118 ∨ sym = 119;

Boolean procedure num;
num:= last char < 10 ∨ last char = 88 ∨ last char = 89;

procedure er(ne);   integer ne;
begin  asgok:= false;   error(ne);   type:= 3   end er;

procedure enter name;
begin   integer cnt;
        cnt:= 0;
next:   if letter then
        begin   if cnt < 80 then name[cnt]:= last char;
                cnt:= cnt + 1;   nxt sbl;
                goto next
        end name;
        if cnt < 80 then name[cnt]:= 255
end enter name;

Boolean procedure call(proc); procedure proc;
begin   Boolean found;

        procedure compcall(st,var); string st;
        if ¬found then
        begin   integer cnt s,cnt a,sym;
                cnt a:= cnt s:= -1;
        next:
                cnt a:= cnt a + 1;
        skip:   cnt s:= cnt s + 1;
                sym:= stringsymbol(cnt s,st);
                if layout(sym) then goto skip;
                if sym = name[cnt a] then
                begin   if sym = 255 ∨ cnt a = 79 then
                        begin   found:= true;
                                proc(var)
                        end else
                        goto next
                end
        end compcall;

        found:= false;   varspec(compcall);   call:= found
end call;

integer procedure octval;
begin   real x;
        x:= 0;
next:
        if last char > 7 then goto out;
        x:= x × 8 + last char; nxt sbl; goto next;
out:    if last char ≠ 120 then er(11) else nxt sbl;
        if x > maxint then
        begin   x:= x - maxint - maxint - 1;
                if x > 0 then er(12)
        end bit 26 on;
        octval:= x
```

```
end octval;

real procedure uns numb(nxt sbl, last char);
        integer nxt sbl, last char;
begin   integer prev char;

        integer procedure check;
        begin   check:= last char:= nxt sbl;
                if prev char = 88 ∨
                   prev char = 64 ∨ prev char = 65 then
                begin   if last char > 9 then
                        begin er(13); goto cancel
                        end incorrect decimal part
                end else
                if prev char = 89 then
                begin   if last char > 9 ∧
                           last char ≠ 64 ∧
                           last char ≠ 65 then
                        begin er(14);   goto cancel
                        end incorrect exponent part
                end;
                prev char:= last char
        end check;

        prev char:= last char;   uns numb:= 0;
        uns numb:= read1(check, last char);
cancel:
end uns numb;

procedure set(a,b); a:= b;
comment let not thy left hand know what thy right hand doeth;

procedure confirm(req); value req; integer req;
begin   if type = req then else if type = 3 then else
        if type = 0 then
        begin if req = 1 then callset(ar) else callset(bool)
        end else
        if req = 1 then er(5) else er(6);
        type:= req
end confirm;

procedure callset(var);
begin   procedure value(var2); var:= var2;
        if !call(value) then er(4)
end callset;

procedure primary;
if last char = 98 then
begin   nxt sbl;
        expr;
        if last char ≠ 99 then er(10) else nxt sbl
end parentheses else
if letter then
begin   enter name;
        if last char=90 then act asg else type:= 0
end identifier else
```

```
if num then
begin ar:= uns numb(nxt sbl,last char); type:= 1 end numb else
if last char > 256 then
begin   type:= 2;
        if last char = true then bool:= true else
        if last char = false then bool:= false else er(9);
        nxt sbl
end boolean else
if last char = 120 then
begin   nxt sbl; ar:= octval; type:= 1 end else
er(8);

procedure term;
begin   factor;
next factor:
        if last char = 66 ∨ last char = 67 then
        begin   integer slc;   real sar;
                confirm(1);   slc:= last char;   sar:= ar;
                nxt sbl;
                factor;
                confirm(1);
                ar:= if slc = 66 then sar × ar else sar/ar;
                goto next factor
        end times,divide else
        if last char = 80 then
        begin   Boolean sbool;
                confirm(2);   sbool:= bool;
                nxt sbl;
                factor;
                confirm(2);
                bool:= sbool ∧ bool;
                goto next factor
        end logical and
end term;

procedure expr;
begin   term;
next term:
        if last char = 64 ∨ last char = 65 then
        begin   integer slc;   real sar;
                confirm(1);   slc:= last char;   sar:= ar;
                nxt sbl;
                term;
                confirm(1);
                ar:= if slc = 64 then sar + ar else sar - ar;
                goto next term
        end plus, minus else
        if last char = 79 then
        begin   Boolean sbool;
                confirm(2);   sbool:= bool;
                nxt sbl;
                term;
                confirm(2);
                bool:= sbool ∨ bool;
                goto next term
        end logical or
```

```
end expr;

procedure factor;
if last char = 64 then
begin   nxt sbl;
        primary;
        confirm(1)
end monadic plus else
if last char = 65 then
begin   nxt sbl;
        primary;
        confirm(1);
        ar:= -ar
end monadic minus else
if last char = 76 then
begin   nxt sbl;
        primary;
        confirm(2);
        bool:= ¬ bool
end logical not else
primary;

procedure asg expr(var);
begin   expr;
        if asgok then
        begin   if type = 0 then callset(var) else
                if type = 1 then set(var,ar) else
                set(var,bool)
        end asgok
end asg expr;

procedure act asg;
begin   if last char ≠ 90 ∨ nxt sbl ≠ 70 then er(7) else
        nxt sbl;
        if ¬call(asg expr) then begin er(3);   expr  end
end act asg;

procedure string transfer(proc);   procedure proc;
if last char ≠ 121 then er(17) else
begin   integer char;
rep:    char:= resym;
        if if char = 121 then nxt sbl = 121 else true then
        begin proc(char);  goto rep  end string item
end string transfer;

procedure act stm;
if last char = 98 then
begin   nxt sbl;
        if ¬call(string transfer) then
        begin er(15);  string transfer(dummy) end;
        if last char ≠ 99 then er(16) else nxt sbl
end else
act asg;

procedure stms;
begin
```

```
        rep:    asgok:= true;   nxt sbl;
                if letter then begin enter name; act stm end;
                if last char = 91 then goto rep;
                if last char = end then else
                begin   er(2);
                skip:   nxt sbl;
                        if last char = 91 then goto rep;
                        if last char ≠ end then goto skip
                end error recovery
        end stms;

        begin:= 267;   end:= 270;   true:= 285;   false:= 271;
        comment:= 268;
        maxint:= 2↑26 - 1;   stock:= -1;

sb:     if nxt sbl ≠ begin then begin er(1);   goto sb  end;
        stms
end readvar;
```