

RA

**stichting  
mathematisch  
centrum**



REKENAFDELING

NR 21/71

OKTOBER

RA

M. REM  
DE MC-ELAN MACROPROCESSOR

**2e boerhaavestraat 49 amsterdam**

BIBLIOTHEEK MATHEMATISCH CENTRUM  
AMSTERDAM



*Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.*

*The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.*

## Voorwoord

Dit rapport beschrijft de MC-ELAN Macroprocessor.

Het kan tevens beschouwd worden als een handleiding voor de ELAN-programmeur, die macroinstructies in zijn programma wil opnemen.

De schrijver wil zijn dank betuigen voor de waardevolle suggesties die hij mocht ontvangen van Prof.Dr. R.P. van de Riet.



| <u>Inhoud</u>                            | blz. |
|------------------------------------------|------|
| 0. Inleiding                             | 1    |
| 1. Syntax                                | 1    |
| 2. Semantiek                             | 4    |
| 3. Substitutieproces                     | 6    |
| 4. Samenwerking assembler-macroprocessor | 9    |
| 5. define macro                          | 11   |
| 6. expand macro                          | 13   |
| 7. macro sym                             | 15   |
| 8. Regeldrukker-uitvoer                  | 16   |
| 9. Wijzigingen in de assembler           | 17   |
| 10. Lijst van nieuwe foutmeldingen       | 18   |
| 11. Enkele opmerkingen                   | 19   |
| 12. Twee voorbeelden                     | 21   |
| 13. Programmatekst                       | 28   |
| 14. Literatuur                           | 40   |



## 0. Inleiding

De MC-ELAN Macroprocessor bestaat uit een aantal procedures, die, opgenomen in een wat aangepaste versie van de MC-ELAN Assembler, aan deze de mogelijkheid geven macro's te assembleren. Toen in 1967 een werkgroep, bestaande uit F.E.J. Kruseman Aretz, B.J. Mailloux, K.K. Koksma en E.G.M. Broerse een voorstudie maakte om de bestaande ELX8-Assembler te vervangen door een 2-scans assembler in ALGOL 60, kwam al direkt de wens tot deze mogelijkheid naar voren. (Zie voor een verslag hiervan: [2]).

Het algemene principe is dat een macro-aanvraag wordt vervangen door het blok dat gedefinieerd is in de bijbehorende macrodeclaratie. terwijl tijdens dit vervangen eventuele formele parameters weer vervangen worden door de korresponderende aktuele parameters. In de keuze van de aktuele parameters is de ELAN-programmeur vrijwel geheel vrij. Wil men de verkregen tekst ook laten assembleren, dan zal men er voor moeten zorgen dat er na de vervanging een ELAN-programma ontstaat. Deze eis vervalt als het alleen de bedoeling is om teksten te laten genereren door de macroprocessor.

In een macrodeclaratie mogen weer macro-aanvragen voorkomen. Ook mogen macrodeclaraties genest voorkomen. In aktuele parameters mogen weer macro-aanvragen en-declaraties voorkomen. Rekursie is echter niet toegestaan. Dit geeft de mogelijkheid tot herdeclaratie van ELAN-instructies als macro.

De inbouw van de macroprocessor in de assembler is op zo'n wijze ver-richt dat de programmeur die geen gebruik van de macro-faciliteit maakt, er ook geen hinder van ondervindt. Dit zal veranderen als er een macrobibliotheek komt, waarbij dus een aantal macro's (I/O routines bijv.) standaard gedeclareerd zijn.

## 1. Syntax

Het verwerken van macro's vereist een aantal uitbreidingen en veranderingen van de ELAN-syntaxregels, zoals die voorkomen in [1]. Niet alle regels kunnen gemakkelijk in BNF geschreven worden, daarom

zijn geen produktieregels opgenomen voor <non-empty actual parameter> en <macro block>. Hiervoor in de plaats is gekozen voor een informele omschrijving.

```
<ELAN declaration> ::= <ord decl and MT decl and mac decl>
    <statement separator> | <MT decl and mac decl>
    <statement separator> | <macro declaration> <statement separator> |
    <statement separator>
<ord decl and MT decl and mac decl> ::= <list of initializations> |
    <list of initializations> <comma> <MT declaration> |
    <list of initializations> <comma> <MT declaration> <comma>
    <macro declaration>
<MT decl and mac decl> ::= <MT declaration> |
    <MT declaration> <comma> <macro declaration>
<macro declaration> ::= 'MACRO' <list of mac decl>
<list of mac decl> ::= <mac decl> <comma> <list of mac decl> | <mac decl>
<mac decl> ::= <identifier> <formal parameter list> : <statement separator>
    <macro block> <letgit string option>
<ELAN instruction> ::= <empty> | <STAT address operand> |
    <unsigned real> | <adding operator> <unsigned number> |
    <BI or IP instruction> | <SKIP instruction> | <only mac instruction> |
    <ELAN block> <letgit string option> | <macro call> |
    <UYN part> <functional instruction> <PZE part> |
    <UYN part> <arithmetic instruction> <PZE part>
<only mac instruction> ::= 'ONLY MAC'
<macro call> ::= <identifier> <actual parameter list>

<formal parameter list> ::= <form par list with commas> |
    <form par list with quote> | <empty>
<form par list with commas> ::= (<list of formal parameters>)
<list of formal parameters> ::= <formal parameter> |
    <formal parameter> <comma> <list of formal parameters>
<form par list with quotes> ::= <quoted formal parameter> |
    <quoted formal parameter> <separator option> <form par list with quotes>
<separator option> ::= <statement separator> | <empty>
<quoted formal parameter> ::= <quote open> <formal parameter> <quote close>
<quote open> ::= <
<quote close> ::= >
<formal parameter> ::= <identifier>
```

```
<actual parameter list> ::= <act par list with commas> |  
    <act par list with quotes> | <empty>  
<act par list with commas> ::= (<separator option> <list of actual parameters>)  
<list of actual parameters> ::= <actual parameter> |  
    <actual parameter> <comma> <list of actual parameters>  
<act par list with quotes> ::= <quoted actual parameter> . |  
    <quoted actual parameter> <separator option> <act par list with quotes>  
<quoted actual parameter> ::= <quote open> <actual parameter> <quote close>  
<actual parameter> ::= <non-empty actual parameter> | <empty>
```

<non-empty actual parameter> = een string van één of meer symbolen,  
met de volgende beperking:

a. indien hij deel uitmaakt van een aktuele parameterlijst in  
komma-stijl:

- 1) er moet een bijectie zijn van de verzameling van de openingshaakjes "("  
op de verzameling van de sluihaakjes ")" waarbij het beeld tekstueel  
na het origineel moet komen.
- 2) voor een komma moeten evenveel (als) staan.

b. indien hij deel uitmaakt van een aktuele parameterlijst in quote-  
stijl:

er moet een bijectie zijn van de verzameling van de openings-  
quotes "<" op de verzameling van de sluitquotes ">", waarbij het beeld  
tekstueel na het origineel moet komen.

<macro block> = stuk tekst dat begint met 'BEGIN', en wel vanaf die 'BEGIN'  
het kleinste stuk dat evenveel 'BEGIN's als 'END's bevat. Hierbij  
tellen 'BEGIN's en 'END's tussen apostrophe \*) en eerstvolgende  
scheider("in het commentaar") niet mee.

Voor het overige gelden de produktieregels uit [1].

\*) Om spraakverwarring te voorkomen is in de macroprocessor dezelfde  
terminologie aangehouden, als in de assembler, d.w.z. '='accent en  
"=apostrophe.

## 2. Semantiek

### 2.1. Macrodeclaratie

Een macrodeclaratie bestaat uit een macronaam, een al dan niet lege formele parameterlijst, een colon en een macroblok.

#### 2.1.1. Blok en body

Uit het macroblok wordt op de volgende wijze de macrobody verkregen:

Het eerste symbool van de macrobody is het eerste accent van 'BEGIN', behalve als er geen declaraties aan het begin van het blok staan, dan is het het eerste niet-layoutsymbool na de scheider die (dan) direkt volgt op 'BEGIN'.

Het laatste symbool dat nog tot de macrobody behoort is, als er declaraties in het blok voorkomen het tweede accent van 'END', en anders het laatste symbool voor de scheider die aan de laatste 'END' voorafgaat.

Vanaf een apostrophe worden alle symbolen tot de eerstvolgende semicolon of nlcr weggelaten.

In dit blok kunnen alleen produkties van <quoted formal parameter> opgevat worden als formele parameters.

#### 2.1.2. De scope van een macro

De scope van een macro is het blok waarin hij gedeclareerd wordt, met uitzondering van het macroblok dat die macro definieert.

Indien er echter meerdere macro's met dezelfde naam gedefinieerd worden in één ELAN-declaratie, dan refereert die naam naar de laatst-gedeclareerde geldige macro met die naam. (Een macro is "geldig" op de plaatsen die binnen zijn scope liggen).

### 2.2. Macro-aanvraag

Een macro-aanvraag bestaat uit een macronaam en een al dan niet lege aktuele parameterlijst. Hij mag niet van varianten zijn voorzien. De macro-aanvraag wordt vervangen door de gemodificeerde macrobody (= macrobody waarin de formele door de aktuele parameters zijn vervangen).

Hierna wordt de gesubstitueerde tekst opnieuw in behandeling genomen. Dit substitueren is beschreven in het volgende hoofdstuk.

### 2.3. Formaat

Het eerste symbool van een aktuele parameter, die deel uitmaakt van een aktuele parameterlijst in komma-stijl, is het eerste niet-layoutsymbool na de openingshaak van de parameterlijst of na de parameterscheidingskomma.

Een macro-aanvraag heeft als eerste symbool de eerste letter van de naam, en als laatste het laatste symbool voor de scheider, die direkt moet volgen op de macro-aanvraag. (Het is dus dit stuk tekst dat vervangen wordt).

### 3. Substitutieproces

Een macro-aanvraag in de tekst wordt als volgt verwerkt:

```
procedure expandeer macro;  
begin neem de bijbehorende macrobody;  
    vervang de parameters;  
    maak de macronaam onbekend;  
    while er zijn nog macro-aanvragen in de gemodificeerde body do  
        begin neem de tekstueel eerste;  
            expandeer macro  
        end;  
        maak de macronaam weer bekend;  
        vervang de macro-aanvraag door de gemodificeerde body  
    end expandeer macro;  
  
vervang parameters: vervang in de macrobody de formele door  
    de (wat de plaats in de parameterlijst betreft) korresponderende  
    aktuele parameters.
```

Het substitutieproces is dus een rekursieve bezigheid, en een vraag die dan direkt naar voren komt, is:

"Is er een maximum aan te geven voor de rekursiediepte?"  
of:  
"Komt het proces noodzakelijk klaar?"

Dat het antwoord op deze vragen bevestigend is wordt in de rest van dit hoofdstuk aangetoond.

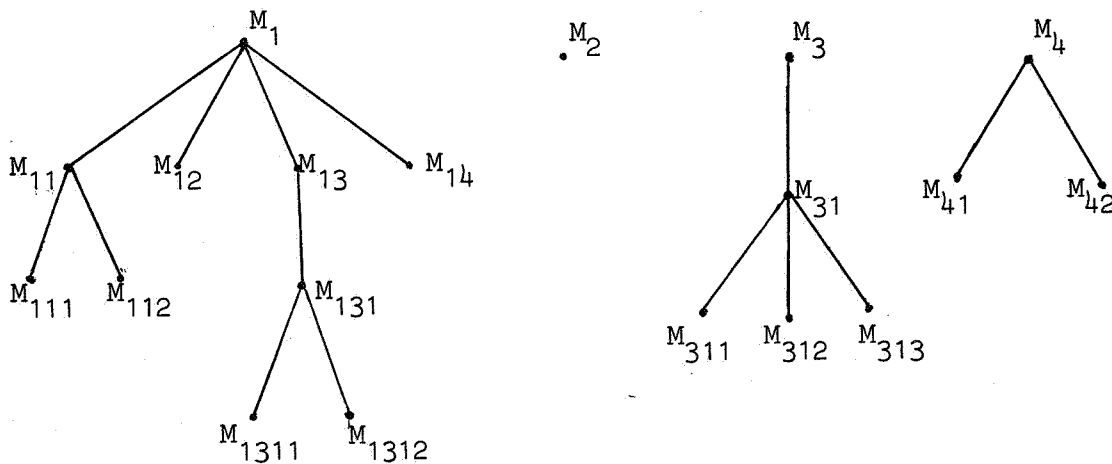
A. We beschouwen eerst het geval dat alle macro's parameterloos zijn. Er vindt dan geen modificatie van de body plaats.

Als er een aanvraag voor de macro  $M_i$  is, dan nemen we het kleinste blok waar de invraag in staat, en beschouwen alle macrodeclaraties uit dat blok (met uitzondering van blokken daarbinnen) en uit de omvattende blokken. Deze noemen we  $M_1, \dots, M_n$ . ( $M_i$  is er één van).

We bepalen nu bij iedere declaratie  $M_j$  ( $1 \leq j \leq n$ ) weer welke macro's daarin gedeclareerd worden. Deze noemen we de zonen van  $M_j$ . Vervolgens bepalen we daar weer de zonen van, enz. Aangezien de programmatekst, en dus ook het aantal declaraties, eindig is, zal dit proces zeker eindigen.

We kunnen nu een woud opstellen van  $n$  bomen, waarin ieder knooppunt staat voor een macrodeclaratie.  $M_1, \dots, M_n$  vormen de wortels van de bomen, en de vertakkingen korresponderen met de zonen.

We noemen dit woud het declaratiewoud. Het volgende is een voorbeeld van een declaratiewoud van 4 bomen:



Alleen aanvragen voor wortels kunnen als macro-aanvragen herkend worden. We voeren nu het proces uit van de geschetste procedure 'expandeer macro'.

Als er een aanvraag voor  $M_1$  moet worden uitgewerkt, dan wordt de naam  $M_1$  onbekend, d.w.z.  $M_1$  en zijn (4) takken worden uit het declaratiewoud verwijderd. (Hierdoor is het aantal bomen in het woud van 4 op 7 gekomen, waardoor er aanvragen voor 7 verschillende macro's mogelijk zijn).

We zoeken daarna de tekstueel eerste macro-aanvraag in de body van  $M_1$ . Als dit bijv.  $M_{13}$  is, dan voeren we hetzelfde proces met  $M_{13}$  uit, enz. Aangezien er iedere keer een knooppunt uit het woud verwijderd wordt, kunnen we konkluderen dat de rekursiediepte ten hoogste gelijk is aan het aantal (statisch te bepalen) knooppunten uit het declaratiewoud.

- B. Het toevoegen van parameters kan het aantal declaraties in een body dynamisch (d.w.z. tijdens het modificeren) veranderen. Er zal echter altijd een bovengrens voor het aantal declaraties aan te geven zijn, zodat er ook dan een eindig declaratiewoud zal zijn.

#### 4. Samenwerking assembler-macroprocessor

Er zijn twee gevallen waarin de assembler en de macroprocessor samenwerken. Aangezien na de eerste scan het programma gezuiverd moet zijn van alle macro-aanvragen dient deze samenwerking zich geheel tijdens de eerste scan af te spelen.

Het eerste geval is wanneer er aan het begin van een blok een macrodeclaratie voorkomt. De procedure 'define macro' zorgt er dan voor dat alle macrodeclaraties uit dit blok verwerkt worden. Dit verwerken bestaat uit het leggen van de naam op de 'namestack' en de macrobody op de 'definitionstack'. In hoofdstuk 5 wordt 'define macro' besproken.

Het tweede geval is wanneer de assembler een macro-aanvraag tegenkomt. Voordat de assembler een naam in de naamlijst opzoekt, kijkt hij of er iets op de 'namestack' ligt. Als dit inderdaad het geval is vergelijkt hij de naam met de namen op de 'namestack', te beginnen met het bovenste element van de stapel. Is dit vergelijken succesvol dan wordt de procedure 'expand macro' aangeroepen. Deze wordt in hoofdstuk 6 besproken. 'expand macro' leest de aktuele parameters en legt deze op de 'actualstack'. Tevens zorgt hij ervoor dat de assembler voortaan symbolen uit de 'definitionstack' en de 'actualstack' krijgt aangeboden. Indien de assembler al uit deze stapels aan het lezen was, wordt de plaats waar hij aan het lezen was gered in de 'savestack', om daarmee weer te kunnen vervolgen als de zojuist aangevatte macro-expansie beëindigd is.

De symbolen uit de twee genoemde stapels worden geleverd door de procedure 'macro sym'. De assembler doet gewoon zijn werk en heeft niet door dat de symbolen, die hij leest, in plaats van uit de inputbuffer uit de stapels van de macroprocessor komen.

#### 'ONLY MAC'

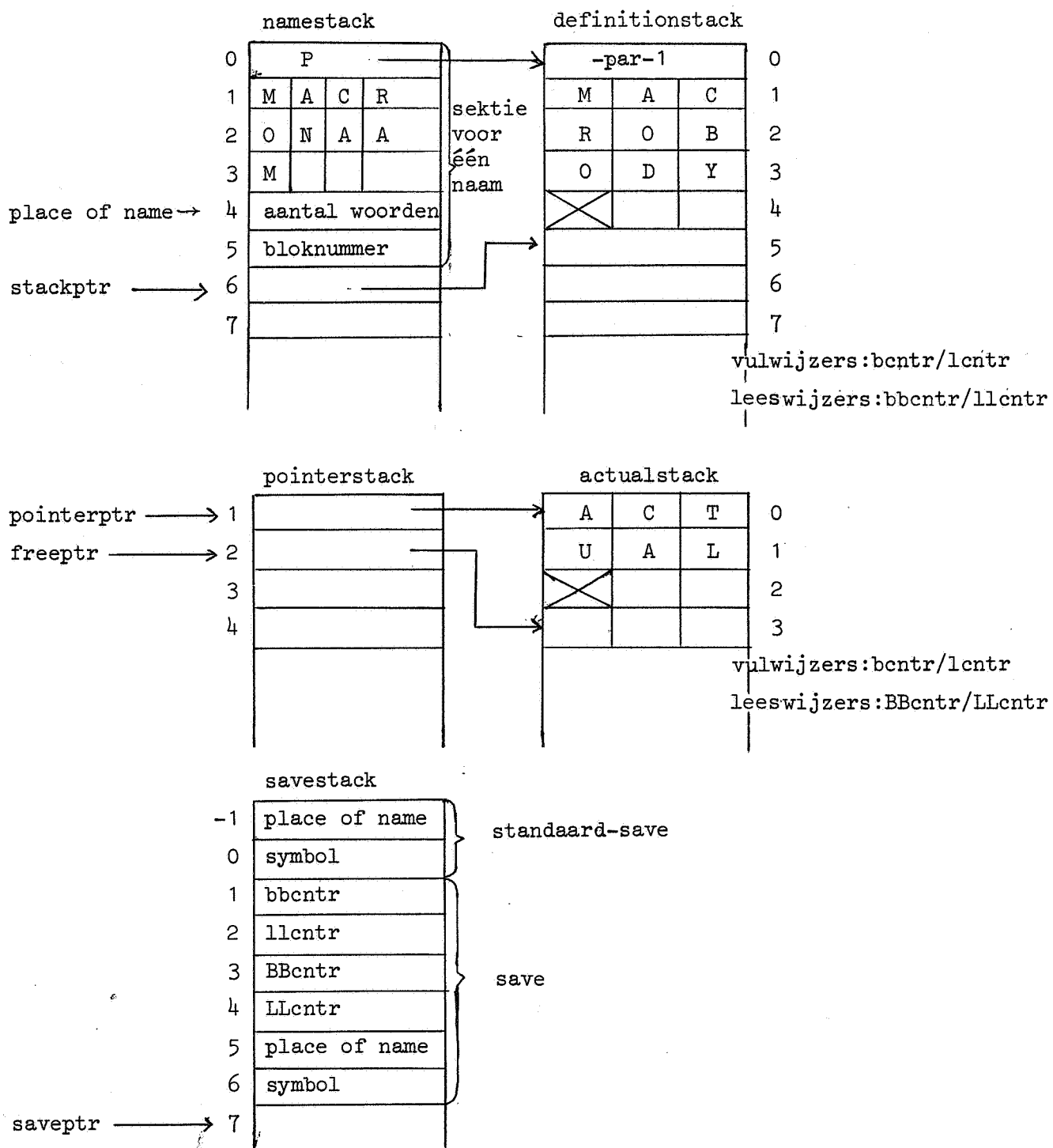
In de tweede scan skipt de assembler alle macrodeclaraties, terwijl de macro-aanvragen dan al verdwenen zijn.

De assembler komt niet in zijn tweede scan indien men de instructie 'ONLY MAC' in zijn programma heeft opgenomen. Hij print en ponst dan na de eerste scan de verkregen tekst uit. Op deze manier kan men de macro-

processor als tekstgenerator gebruiken, zij het dat men dan bij "on-ELANse" teksten wel veel foutmeldingen tijdens de eerste scan voor lief zal moeten nemen.

### De stapels van de macroprocessor

Hier volgt een overzicht van de stapels die door de macroprocessor gebruikt worden. De wijze waarop dit gebeurt zal duidelijk worden in de volgende drie hoofdstukken.



## 5. define macro

De algemene structuur van deze procedure is de volgende:

```
procedure define macro;
while er zijn nog declaraties te lezen do
  begin read name;
    read formals;
    read block
  end define macro;
```

### read name

leest een identifier, plaatst deze op de 'namestack' en vult het aantal geheugenwoorden in, dat de naam in beslag neemt, alsmede het heersende bloknummer. In de 'namestack' wordt ook een pointer gezet naar de tekst van de bijbehorende body in de 'definitionstack'.

### read formals

leest formele parameters tot de colon, die de parameterlijst afsluit, wordt gevonden. Ook bij het lezen van een accent wordt (onder foutmelding 3023) overgegaan naar 'read block'.

De formelen worden met behulp van de procedure 'store letgits' in de 'formallist' opgenomen. ('formallist' is een lokaal array in 'define macro').

### formallist

|                |   |   |   |   |                                                                                                                                        |
|----------------|---|---|---|---|----------------------------------------------------------------------------------------------------------------------------------------|
| aantal woorden |   |   |   | 0 | Het aantal parameters wordt (negatief ten einde te voorkomen dat dit getal als tekst wordt opgevat) genoteerd in de 'definitionstack'. |
| F              | O | R | M | 1 |                                                                                                                                        |
| A              | L | 1 |   | 2 |                                                                                                                                        |
| aantal woorden |   |   |   | 3 |                                                                                                                                        |
| F              | O | R | M | 4 |                                                                                                                                        |
| A              | L | 2 |   | 5 |                                                                                                                                        |
| 0              |   |   |   | 6 |                                                                                                                                        |
|                |   |   |   |   |                                                                                                                                        |

### read block

leest het macroblock en verkrijgt hieruit, op de in hoofdstuk 2 genoemde wijze, de macrobody. 'stow into stack' buffert deze in de 'definitionstack', er zorg voor dragend dat de formele parameters vervangen worden door

speciale symbolen.

Het herkennen van de formele parameters in de body wordt verricht door 'compare parameters', die telkens aangeroepen wordt wanneer er een < in de tekst voorkomt.

Verder worden door 'stow into stack' (ten hoogste 105) achtereenvolgende spaties opgeborgen als één symbool. (Dit speciaal met het oog op ELAN-teksten op ponskaarten). Het eind van de body wordt aangegeven door een speciale endmarker.

Opm. Het bufferen in de 'definitionstack' gebeurt al op het laagste niveau door de leesprocedure 'reaffer', dit is noodzakelijk om geen layout-symbolen verloren te laten gaan.

## 6. expand macro

Algemene structuur:

```
procedure expand macro;  
begin read actuals;  
    maak naam onbekend;  
    standaard-save;  
    if from macro then store expansion  
    else from macro:= true;  
    zet bbcntr en llcntr  
end    expand macro;
```

### read actuals

leest al naar gelang de aktuele parameterlijst in de komma-stijl of in de quote-stijl staat tot de toepasselijke) of . gepasseerd is. Indien in de quote-stijl na de sluitquote van een parameter en na eventuele scheiders geen punt en ook geen openingsquote staat, wordt ook (onder foutmelding 3025 of 3031) aangenomen dat de parameterlijst beëindigd is.

In de 'pointerstack' wordt bijgehouden waar in de 'actualstack' de aktuelen staan. ('pointerptr' wijst naar de pointer van de eerste parameter, 'freeptr' naar de eerste vrije cel).

Het bufferen in de 'actualstack' geschiedt op het niveau van 'read and buffer' om geen layout of commentaar verloren te laten gaan.

'read actuals' controleert ook nog of het aantal aktuele parameters gelijk is aan het aantal formelen in de declaratie.

### maak naam onbekend

In de 'namestack' wordt het bloknummer negatief gemaakt, waardoor de assembler een voorkomen van deze macronaam niet meer als zodanig herkent (tot 'macro sym' aan het eind van de expansie het bloknummer weer positief maakt).

### standaard-save

Het nalaatste symbool van de parameterlijst is al gelezen, dit moet echter nog even bewaard worden omdat eerst de uitwerking van de macro-aanvraag er tussen moet.

Ook 'place of name' (waarin de assembler aangeeft waar in de 'namestack' hij de macronaam gevonden heeft) dient te worden gered, omdat tijdens de uitwerking van de macro-aanvraag de assembler weer een nieuwe macro-aanvraag kan tegenkomen, met het gevolg dat 'place of name' overschreven wordt.

#### store expansion

De genoemde twee gegevens worden altijd gered, maar als we al bezig waren met het uitwerken van een macro-aanvraag, dan worden ook de wijzers, met behulp waarvan we aan het lezen waren, gered. Dat gebeurt in 'store expansion'. Tenslotte krijgen de leeswijzers 'bbcctr' en 'llcctr' hun nieuwe waarde.

## 7. macro sym

Algemene structuur:

```
procedure macro sym;  
if from actualstack then  
begin neem volgend symbool uit de actualstack;  
    if symbool = endmarker then  
        begin from actualstack: = false;  
            symbool:= macro sym  
        end  
    end else  
begin neem volgend symbool uit de definitionstack;  
    if symbool = parametersymbool then  
        begin from actualstack:= true;  
            symbool:= macro sym  
        end else  
if symbool = endmarker then  
begin symbool:= geredde nalaatste symbool;  
    maak macronaam weer bekend;  
    if savestack leeg then from macro:= false  
    else restore expansion  
    end  
end macro sym;
```

'macro sym' is de procedure, die de symbolen aflevert, tijdens het uitwerken ("expanderen") van een macro-aanvraag.

'macro sym' begint te lezen uit de 'definitionstack'. Indien hij daarbij een symbool tegenkomt dat de representatie is van een formele parameter, dan gaat hij lezen uit de 'actualstack'. Komt hij in de 'actualstack' een endmarker tegen, dan leest hij weer verder uit de 'definitionsstack'. De macro-expansie is voltooid als 'macro sym' bij het lezen uit de 'definitionstack' een endmarker tegenkomt. Het geredde nalaatste symbool wordt dan weer opgepikt, en het bloknummer in de 'namestack' positief gemaakt, waardoor de naam weer bekend wordt.

Aan de grootte van 'saveptr' valt te zien of we nog bezig waren met een macro-expansie toen met de zojuist voltooide uitwerking werd begonnen. Als dit het geval was zorgt 'restore expansion' er voor dat de status-quo van voor de expansie hersteld wordt. Dit houdt ook een aflagen van de 'pointerstack' (en dus van de 'actualstack') en de 'savestack' in.

## 8. Regeldrukker-uitvoer

Tijdens de eerste scan worden alle macro-aanvragen geprint met een marge van 7 posities. In deze marge komt het regelnummer van het eerste symbool van de aanvraag. Dat is hetzelfde regelnummer als na de vervanging het eerste symbool van de gesubstitueerde tekst krijgt.

Als de aanvraag over meerdere regels verdeeld is, wordt geen nieuw regelnummer afgedrukt.

Aangezien de assembler pas ontdekt met een macro-aanvraag te doen te hebben op een niveau waar de layout-symbolen en de case al verloren zijn gegaan, wordt de macronaam zonder spaties en tabulaties in de lower case afgedrukt. De rest van de macro-aanvraag (aktuele parameterlijst, commentaar) wordt door de macroprocessor in zijn oorspronkelijke layout afgedrukt.

Tijdens de tweede scan wordt, indien er geen 'ONLY MAC'-instructie in de tekst voorkomt, de tekst, waarin alle substituties al hebben plaats gehad, gelist. Dit gebeurt op dezelfde wijze als vóór de invoering van de macroprocessor het geval was. Het enige verschil is dat regels waarop macrosubstitutie is toegepast, voorzien zijn van een sterretje (na het regelnummer).

Als er wel een 'ONLY MAC'-instructie is gegeven, wordt na de eerste scan de tekst met regelnummering en sterretjes afgedrukt, en zonder regelnummering en sterretjes in dezelfde layout geponst.

## 9. Wijzigingen in de assembler

### procedure RE ELAN block;

Bij blokverlating wordt door de procedure 'unstack macros' de 'namestack' afgelaagd met alle namen die het bloknummer van het blok, dat verlaten wordt, hebben. (Ook als dit bloknummer negatief in de 'namestack' staat, dat kan namelijk, zie 11.1). Aangezien de 'namestack' de pointers naar de 'definitionstack' bevat wordt dan ook gelijk de 'definitionstack' afgelaagd.

### procedure RE ELAN declaration;

Als deze procedure een macrosymbool ('MACRO') tegenkomt roept hij in de eerste scan 'define macro' en in de tweede scan 'skip macro declarations' aan.

### procedure RE ELAN instruction;

Deze procedure kan een macro identifier tegenkomen. Hij controleert dan of er geen varianten zijn, en roept 'expand macro' aan. Hierna krijgt de assembler symbolen uit de stapels van de macroprocessor aangeboden. Hij moet nu zodanig in het rechte spoor gebracht worden dat hij de tekst tijdens deze overgang in beide scans op dezelfde wijze interpreteert. Hiertoe is het van belang te weten of er al een inzetaanwijzing of een label gelezen is, omdat er dan geen inzetaanwijzing meer mag komen. Dit wordt bijgehouden in de boolean 'margin read'.

### integer procedure READ synt unit;

Als deze een macronaam leest, wordt de naam met regelnummer afgedrukt en uit de tekstbuffer verwijderd. In 'place of name' wordt ingevuld waar de naam in de 'namestack' staat, terwijl de syntaktische eenheid wordt: 'macro identifier'.

### procedure NS;

NS kan bericht geven aan 'PRINT LINE' dat het sterretjes printen moet beginnen of ophouden.

### procedure RESYM 1;

Toegevoegd is: if from macro then s:= macro sym;

### procedure PRINT LINE;

PRINT LINE is bedacht op sterretjes printen en tekst uitponsen.

## 10. Lijst van nieuwe foutmeldingen

### 10.1. Foutmeldingen van de macroprocessor

- 3000     namestack vol
- 3001     macro-aanvraag niet gevolgd door scheider
- 3002     meer dan 22 formele parameters
- 3004     in formele parameterlijst volgt op   ) geen :
- 3005     in formele parameterlijst volgt op parameter geen , of )
- 3006     in formele parameterlijst volgt op parameter geen >
- 3007     in formele parameterlijst volgt op > geen : of <separator> <
- 3008     formele parameter begint niet met een letter
- 3009     formallist vol
- 3010     macroblok begint niet met 'BEGIN'
- 3011     formele parameter heeft meer dan 84 niet-layout symbolen
- 3013     pointerstack vol
- 3016     aantal aktuelen in macro-aanvraag ongelijk aantal formelen  
          in declaratie
- 3017     actualstack vol
- 3018     actualstack of definitionstack vol
- 3019     formallist of namestack vol
- 3023     formele parameterlijst onjuist afgesloten
- 3024     op formele parameterlijst volgt geen scheider
- 3025     aktuele parameterlijst in quote-stijl niet afgesloten door .
- 3026     macronaam in declaratie niet gevolgd door < of ( of :
- 3030     macronaam in declaratie begint niet met een letter

### 10.2. Foutmeldingen van de assembler

- 335     macronaam in declaratie begint niet met een letter
- 525     macro-aanvraag met variant U, Y of N
- 812     'M niet gevolgd door A of T

## 11. Enkele opmerkingen

### 11.1. 'BEGIN' en 'END' in de aktuele parameters

Dit heeft tot gevolg dat door de parameterssubstitutie de blokstructuur verandert.

Indien een aktuele parameter in een aanvraag van macro M meer 'ENDS's dan 'BEGIN's bevat, kunnen we in de situatie terecht komen dat het blok waarin M gedeclareerd is, tijdens de uitwerking van M wordt afgesloten, waarbij de macro M van de stapel wordt genomen. Dit kan aanleiding geven tot vreemde situaties omdat daarna nog doorgelezen wordt uit de 'definitionstack' tot de endmarker bereikt is.

Meer 'BEGIN's dan 'END's kan aanleiding geven tot een verandering van de blokstructuur waardoor een macro langer blijft bestaan dan men (niet lettend op de aktuele parameter) zou zeggen.

Ook aktuele parameters met bijvoorbeeld eerst een 'END' en even later een 'BEGIN' kunnen vreemde effecten veroorzaken, evenals bijv. een apostrophe als aktuele parameter of een oneven aantal accenten in een aktuele parameter.

Het spreekt voor zich dat het gebruik van dergelijke "trucjes" moet worden afgeraden.

### 11.2. Waarom twee stijlen voor de parameterlijsten?

De komma-stijl heeft het voordeel dat ELAN-instructies die het formaat van een macro-aanvraag hebben (bijv. sprongopdrachten, schuifopdrachten, PLUSA, DO enz.) ook als macro's gedeclareerd kunnen worden. Ook komt deze stijl meer overeen met de macrofaciliteit van de ELX8-Assembler (Zie voor een beschrijving hiervan: [3]).

De quote-stijl heeft het voordeel dat er meer vrijheid is in de samenstelling van de aktuele parameters. Deze stijl sluit ook aan bij de wijze waarop de aktuele parameters in de macrobody voorkomen.

### 11.3. Verboden namen

De volgende namen zijn als macronaam verboden:

A,B,E,F,G,M,N,P,S,T,U,Y,Z,

MA,MC,MD,MG,MS,MT,

M1,M2,...,M99.

#### 11.4. Accenten en hoofdletters

Als de leesprocedure 'reaffer' een accent leest, leest hij nog een extra symbool om te kijken of dat ook een accent is, aangezien twee accenten (zonder spaties of tabulaties ertussen) worden opgevat als een apostrophe. Vb.: 'BEGIN' 'MACRO' MAC: wordt 'BEGIN' 'MACRO' MAC: (waarbij 'MACRO' MAC: wordt opgevat als commentaar).

De macroprocessor maakt geen onderscheid tussen hoofdletters en kleine letters.

#### 11.5. Conditional Assembly

De faciliteit van "conditional assembly" (ook een van de wensen van de werkgroep [2]) is nog niet verwezenlijkt, maar de methode die in de macroprocessor is gevolgd kan ook hiervoor in grote lijnen toegepast worden. Telkens wanneer de assembler een 'IF'...'THEN'...'ELSE'...'FI' - constructie tegenkomt, roept hij een speciale procedure aan, die de tekst omvormt tot de gewenste, op analoge wijze (maar veel eenvoudiger) als dat nu gebeurt wanneer de assembler een macro-aanvraag tegenkomt. Ook andere faciliteiten (repetitie over een lijst, bijv.) kunnen op analoge wijze eenvoudig verwezenlijkt worden. Nu men de beschikking heeft over een macroprocessor, zal de behoefte aan dergelijke faciliteiten sterker gevoeld worden, dan in het verleden het geval was. Een beschrijving van mogelijke faciliteiten is te vinden in [4] en [5].

## 11. Enkele opmerkingen

### 11.1. 'BEGIN' en 'END' in de aktuele parameters

Dit heeft tot gevolg dat door de parameterssubstitutie de blokstructuur verandert.

Indien een aktuele parameter in een aanvraag van macro M meer 'ENDS's dan 'BEGIN's bevat, kunnen we in de situatie terecht komen dat het blok waarin M gedeclareerd is, tijdens de uitwerking van M wordt afgesloten, waarbij de macro M van de stapel wordt genomen. Dit kan aanleiding geven tot vreemde situaties omdat daarna nog doorgelezen wordt uit de 'definitionstack' tot de endmarker bereikt is.

Meer 'BEGIN's dan 'END's kan aanleiding geven tot een verandering van de blokstructuur waardoor een macro langer blijft bestaan dan men (niet lettend op de aktuele parameter) zou zeggen.

Ook aktuele parameters met bijvoorbeeld eerst een 'END' en even later een 'BEGIN' kunnen vreemde effecten veroorzaken, evenals bijv. een apostrophe als aktuele parameter of een oneven aantal accenten in een aktuele parameter.

Het spreekt voor zich dat het gebruik van dergelijke "trucjes" moet worden afgeraden.

### 11.2. Waarom twee stijlen voor de parameterlijsten?

De komma-stijl heeft het voordeel dat ELAN-instructies die het formaat van een macro-aanvraag hebben (bijv. sprongopdrachten, schuifopdrachten, PLUSA, DO enz.) ook als macro's gedeclareerd kunnen worden. Ook komt deze stijl meer overeen met de macrofaciliteit van de ELX8-Assembler (Zie voor een beschrijving hiervan: [3]).

De quote-stijl heeft het voordeel dat er meer vrijheid is in de samenstelling van de aktuele parameters. Deze stijl sluit ook aan bij de wijze waarop de aktuele parameters in de macrobody voorkomen.

### 11.3. Verboden namen

De volgende namen zijn als macronaam verboden:

A,B,E,F,G,M,N,P,S,T,U,Y,Z,

MA,MC,MD,MG,MS,MT,

M1,M2,...,M99.

#### 11.4. Accenten en hoofdletters

Als de leesprocedure 'reaffer' een accent leest, leest hij nog een extra symbool om te kijken of dat ook een accent is, aangezien twee accenten (zonder spaties of tabulaties ertussen) worden opgevat als een apostrophe. Vb.: 'BEGIN' 'MACRO' MAC: wordt 'BEGIN' 'MACRO' MAC: (waarbij 'MACRO' MAC: wordt opgevat als commentaar).

De macroprocessor maakt geen onderscheid tussen hoofdletters en kleine letters.

#### 11.5. Conditional Assembly

De faciliteit van "conditional assembly" (ook een van de wensen van de werkgroep [2]) is nog niet verwezenlijkt, maar de methode die in de macroprocessor is gevolgd kan ook hiervoor in grote lijnen toegepast worden. Telkens wanneer de assembler een 'IF'...'THEN'...'ELSE'...'FI' - konstruktie tegenkomt, roept hij een speciale procedure aan, die de tekst omvormt tot de gewenste, op analoge wijze (maar veel eenvoudiger) als dat nu gebeurt wanneer de assembler een macro-aanvraag tegenkomt. Ook andere faciliteiten (repetitie over een lijst, bijv.) kunnen op analoge wijze eenvoudig verwezenlijkt worden. Nu men de beschikking heeft over een macroprocessor, zal de behoefte aan dergelijke faciliteiten sterker gevoeld worden, dan in het verleden het geval was. Een beschrijving van mogelijke faciliteiten is te vinden in [4] en [5].

## 12. Twee voorbeelden

Het eerste voorbeeld is de generatie van het kinderliedje "Ik zag twee beren...".

We zien allereerst de inputtape, en op de pagina daarna een afdruk van de ponsband, zoals die na de eerste scan ten gevolge van de 'ONLY MAC'-instructie is uitgeponst.

Het tweede voorbeeld is wat ingewikkelder.

We zien hierin de macro 'rgt' die de ELAN-instructies voor de snelle registertransporten "bakt".

Verder zien we in de macro-aanvraag 'macro' een voorbeeld van een blok (inclusief macrodeclaratie en -aanvraag) als aktuele parameter.

Bij de macro 'mac' zien we een declaratie binnen een declaratie.

Verder wordt het begrip "scope" geïllustreerd door het feit dat 'subc' soms voor de macro, en soms voor de ELAN-instructie staat.

We zien dat het gebruik van de komma- en de quote-stijl afgewisseld mogen worden.

Van het tweede voorbeeld is achtereenvolgens opgenomen:

de inputtape, de output tijdens de eerste scan en de output tijdens de tweede scan.

```
'only mac'  
'begin' 'macro' ik zag <nummer> <dieren> <voorwerp> <bewerken>:  
    'begin'  
    <nummer> couplet:  
  
    ik zag twee <dieren> <voorwerp> <bewerken>.  
    oh, het was een wonder.  
    het was een wonder, bovenwonder,  
    dat die <dieren> <bewerken> konden.  
    hihhi, hahaha,  
    ik stond erbij en ik keek er naar.  
    'end'  
  
    ik zag twee beren  
  
    ik zag <eerste> <beren> <broodje> <smeren>.  
  
    ik zag <tweede> <apen> <wortelen> <schrappen>.  
  
    ik zag <derde> <koeien> <luidkeels> <loeien>.  
  
    ik zag <vierde> <rekenautomaten> <met elkaar> <praten>.  
  
'end'
```

'only mac'

'begin' 'macro' ik zag <nummer> <dieren> <voorwerp> <bewerken>:

'begin'

<nummer> couplet:

ik zag twee <dieren> <voorwerp> <bewerken>.

oh, het was een wonder.

het was een wonder, bovenwonder,

dat die <dieren> <bewerken> konden.

hihihi, hahaha,

ik stond erbij en ik keek er naar.

'end'

ik zag twee beren

eerste couplet:

ik zag twee beren broodje smeren.

oh, het was een wonder.

het was een wonder, bovenwonder,

dat die beren smeren konden.

hihihi, hahaha,

ik stond erbij en ik keek er naar.

tweede couplet:

ik zag twee apen wortelen schrapen.

oh, het was een wonder.

het was een wonder, bovenwonder,

dat die apen schrapen konden.

hihihi, hahaha,

ik stond erbij en ik keek er naar.

derde couplet:

ik zag twee koeien luidkeels loeien.

oh, het was een wonder.

het was een wonder, bovenwonder,

dat die koeien loeien konden.

hihihi, hahaha,

ik stond erbij en ik keek er naar.

vierde couplet:

ik zag twee rekenautomaten met elkaar praten.

oh, het was een wonder.

het was een wonder, bovenwonder,

dat die rekenautomaten praten konden.

hihihi, hahaha,

ik stond erbij en ik keek er naar.

'end'

```

'begin' label,                                " Testprogramma MC ELAN Macroprocessor,
                                              " MR 011071

'macro' rgt(v1,r1,s,r2,v2):
'begin' uu = '100 000', yy = '200 000', nn = '300 000',
      pp = '400 000', zz = '1 000 000', ee = '1 400 000'
(<v1> + '660000400' + (1 <s> 1) × '400 000' + (: <r2> - 59) × '40' +
                                              (<v2> + : <r1> - 59))
'end' rgt,

macro <parameter>:                            " Zal worden aangevraagd
                                              " met een blok als
                                              " aktuele parameter

'begin'
ivon
<parameter>
ivoff
'end' macro,

mac(par1,
  par2):
'begin' <par1>,

      'macro' subc(par):                      " Macrodeclaratie binnen mac
      'begin' naam
naam:  subc(mc[-1])                          " Dit is niet de macro subc
      <par2>
      <par>
      'end' subc

<par1>: a + 2
      subc(rgt(,a,-,b,))                     " a:= -b
      subc(<subc(:<par1>)>).                 " De eerste subc is de macro,
                                              " de tweede de ELAN instructie

'end' mac

m[1000]: rgt(uu,s,+,a,pp)                    " u, s:= a, p

mac <label>
  <a + 5
  a + 6>.
label: subc(mc[-1])                          " Dit is niet de macro subc

macro<'begin' 'macro' naam:
      'begin'
      a + 1
      'end' naam

      naam
'end'>.

'end'

```

131071 - 214

MC ELAN1 ASSEMBLER AND MACRO PROCESSOR D.D, 01 10 71

35 RGT(UU,S,+,A,PP) " U, S:= A, P

40 MAC <LABEL>  
    <A + 5  
      A + 6>.

51 SUFC(RGT(,A,-,B,))

55 RGT(,A,-,B,)

60 SUEC <SUEC(:LABEL)>.

70 MACRC<'BEGIN' 'MACRO' NAAM:  
    'BEGIN'  
    A + 1  
    'END' NAAM  
    NAAM  
    'END'>.

76 NAAM

```

1      1      'BEGIN' LABEL,
2
3      'MACRO' RGT(V1,R1,S,R2,V2):
4      'BEGIN' UV = '100 000', VY = '200 000', NN = '300 000',
5      PP = '400 000', ZZ = '1 000 000', EE = '1 400 000'
6      (<V1> + '660000400' + (1<S> 1) * '400 000' + (:<R2> - 59) * '40' + (<V2> + 1<R1> - 59))
7      'END' RGT,
8
9      MACRO <PARAMETER>:
10
11      'BEGIN'
12      IVON
13      <PARAMETER>
14      IVOFF
15      'END' MACRO,
16
17      MAC(PAR1,
18      PAR2):
19      'BEGIN' <PAR1>,
20
21      'MACRO' SUBC(PAR):
22      'BEGIN' NAAM
23      SUBC(MC[-1])
24      <PAR2>
25      <PAR>
26      'END' SUBC
27
28      <PAR1>: A + 2
29      SUBC(RGT(A,-B))
30      SUBC <SUBC(:<PAR1>)>.
31
32      " AI=-B
33      " DE EERSTE SUBC IS DE MACRO,
34      " DE TWEEDE DE EEN INSTRUCTIE
35
36      'END' MAC
37
38      'BEGIN' LABEL,
39
40      'MACRO' SUBC(PAR):
41      'BEGIN' NAAM
42      SUBC(MC[-1])
43
44      A + 5
45      A + 6
46      <PAR>
47      'END' SUBC
48
49      LABEL: A + 2
50      'BEGIN' NAAM
51      NAAM: SUBC(MC[-1])
52      A + 5
53      A + 6
54      'BEGIN' UV = '100 000', VY = '200 000', NN = '300 000',
55      PP = '400 000', ZZ = '1 000 000', EE = '1 400 000'
56      (< + '660000400' + (1 - 1) * '400 000' + (:B - 59) * '40' + ( + :A - 59))
57      'END'
58      'END'
59      'BEGIN' NAAM
60

```

```

131071 - 214
61 *'001756': '566475377' NAAM: SUBC(MC1-1))
62 *'001757': '002000005' A + 5
63 *'001760': '002000006' A + 6
64 *'001761': '562401751' SUBC(!LABEL)
65 * 6 END'
66 * 3
67 * 3
68 *'001762': '566475377' LABEL: SUBC(MC1-1))
69 *'001763': '760060000' IVOFF
70 *'001763': '760060000' IVOFF
71 * 7
72 * 'BEGIN' 'MACRO' NAAM:
73 * 'BEGIN'
74 * A + 1
75 * 'END' NAAM
76 *'001764': '002000001'
77 * 7
78 *'001765': '660060000'
79 1 'END'

```

43

" DIT IS NIET DE MACRO SUBC

13. Programmatekst

integer max of namestack, max of defstack, max of actualstack, max of  
pointerstack, max of savestack, stackptr, freeptr, pointerptr, saveptr,  
spacecntr, lcntr, bcntr, llcntr, bbntr, SPACEcntr, LLcntr, BBcntr, t8j,  
t8J, word, Word, nextacc, endmarker, place of name, tt, asterisk, ksiretsa;  
boolean in def mode, in actual mode, only mac, from macro, from  
actualstack, accent read;  
integer array namestack[-2:255], definitionstack[0:4095], actualstack  
[0:2047], pointerstack[1:128], savestack[-1:120];

procedure initialize macro variables;

begin in def mode:= in actual mode:= only mac:= from macro:=  
from actualstack:= accent read:= false;  
max of namestack:= 255;  
max of defstack:= 4095;  
max of actualstack:= 2047;  
max of pointerstack:= 128;  
max of savestack:= 120;  
asterisk:= 254; ksiretsa:= 255;  
stackptr:= namestack[0]:= pointerstack[1]:= 0;  
endmarker:= spacecntr:= SPACEcntr:= 150;  
namestack[-1]:= saveptr:= -5;  
tt:= 1 + t6 + t12 + t18;  
freeptr:= 1

end initialize macro variables;

procedure define macro;

begin integer i, savel, max of formallist;  
boolean empty;  
integer array formallist[0:127];

procedure read name;

begin integer save;  
ERROR(stackptr > max of namestack, 3000);  
lcntr:= savel:= namestack[stackptr]; ERROR(lcntr = -1, 3027);  
bcntr:= 2; save:= stackptr:= stackptr + 1;  
store letgits(namestack, stackptr, max of namestack, reaffer);  
ERROR(stackptr + 1 > max of namestack, 3000);  
namestack[stackptr]:= stackptr - save;  
namestack[stackptr + 1]:= blocknumber;  
stackptr:= stackptr + 2

end read name;

```
procedure read formals;
begin integer i, ptr, aux, par;
  boolean in comma mode;

  integer procedure reaffer1;
  if symbol = accent symbol then
  begin ERROR(true, 3023); goto outaccent end
  else reaffer1 := reaffer;

  procedure reaffer1 while(condition); boolean condition;
  begin integer i;
    for i := 1 while condition do reaffer1
  end reaffer1 while;

  ptr := 1; par := 0;
  if symbol ≠ colon symbol ∧ symbol ≠ open symbol ∧ symbol ≠
  smaller symbol then
  begin ERROR(true, 3026);
    reaffer1 while(symbol ≠ colon symbol ∧
    symbol ≠ open symbol ∧ symbol ≠ smaller symbol)
  end;
  if symbol ≠ colon symbol then
  begin i := aux := 0;
    in comma mode := symbol = open symbol;
    for i := 1 while symbol ≠ colon symbol do
    begin par := i; reaffer1;
      if in comma mode ∧ i > 1 then reaffer1 while
      (symbol = nlcr symbol ∨ symbol = semicolon symbol);
      if 10 < symbol ∧ symbol ≤ 62 then
      begin if i = 23 then
        begin ERROR(true, 3002); reaffer1 while
          (symbol ≠ colon symbol)
        end else
        begin store letgits(formallist, ptr, max of
          formallist, reaffer);
          if ptr - aux > 22 then
          begin ERROR(true, 3011); ptr := aux + 22
          end;
          formallist[aux] := ptr - aux - 1;
          aux := ptr; ptr := ptr + 1;
          if symbol = accent symbol then
          begin ERROR(true, 3023); goto outaccent
          end;
          if in comma mode then
          begin if symbol = close symbol then
            begin if reaffer1 ≠ colon symbol then
              begin ERROR(true, 3004);
                reaffer1 while
                  (symbol ≠ colon symbol)
              end
            end else if symbol ≠ comma symbol then
              begin ERROR(true, 3005);
                reaffer1 while(symbol ≠ comma
                  symbol ∧ symbol ≠ colon symbol)
              end
            end else
            end else
          end else
        end else
      end else
    end else
  end else
end else
```

```

begin if symbol ≠ greater symbol then
  begin ERROR(true,3006);
    reafferl while(symbol ≠ smaller
      symbol ∧ symbol ≠ colon symbol)
    end else
    begin if reafferl ≠ colon symbol then
      begin reafferl while(symbol =
        nlc symbol ∨ symbol = semicolon
        symbol);
        ERROR(symbol = colon symbol,3007)
      end;
      if symbol ≠ smaller symbol ∧
        symbol ≠ colon symbol then
        begin ERROR(true,3007);
          reafferl while(symbol ≠
            smaller symbol ∧ symbol ≠
            colon symbol)
          end
        end
      end
    end else
    begin ERROR(true,3008);
      reafferl while(symbol ≠ comma symbol ∧
        symbol ≠ smaller symbol ∧ symbol ≠ colon
        symbol)
      end
    end
  end if symbol;
  if ptr - 1 < max of formallist then formallist[ptr - 1] := 0
  else ERROR(true,3009);
  reaffer;
  ERROR(symbol ≠ nlc symbol ∧ symbol ≠ semicolon symbol,3024);
  read while(symbol = nlc symbol ∨ symbol = semicolon symbol);
  outaccent: definitionstack[lcntr] := -par - 1
end read formals;

procedure read block;
begin integer i, begcntr;
  boolean declarations, within accents;
  procedure compare parameters;
  begin integer i, j, l, ptr, length;
    boolean found;
    integer array parameter[0:20];

```

```

ptr:= 1:= 0; j:= 127; found:= false;
reaffer;
store letgits(parameter,1,20,reaffer);
if 1  $\neq$  22  $\wedge$  symbol = greater symbol then
for length:= formallist[ptr] while length  $\neq$  0  $\wedge$   $\neg$  found do
begin j:= j + 1;
    if length = 1 then
        begin i:= 0;
            for i:= i + 1 while parameter[i - 1] =
                formallist[ptr + i]  $\wedge$   $\neg$  found do
                if i = 1 then
                    begin for l:= 1, 1 while l  $\neq$  smaller symbol
                        do delete symbol(l); i:= i - 1;
                        stow into stack(definitionstack,
                            max of defstack,j);
                        found:= true
                    end
                end;
            ptr:= ptr + length + 1
        end
    end;
end compare parameters;

procedure delete symbol(s); integer s;
begin integer word;
    if bcntr = 0 then
        begin s:= definitionstack[lcntr];
            lcntr:= lcntr - 1; bcntr:= 2
        end else
        begin word:= definitionstack[lcntr];
            if word < 0 then empty:= true else
                begin definitionstack[lcntr]:= word : t8;
                    s:= word - definitionstack[lcntr]  $\times$  t8;
                    bcntr:= bcntr - 1
                end
            end
        end
    end delete symbol;

in def mode:= true;
stow into stack(definitionstack,max of defstack,symbol);
if  $\neg$  compare( $\langle$ 'begin' $\rangle$ ) then
begin ERROR(true,3010); skip until( $\langle$ 'begin' $\rangle$ ) end;
begcntr:= 1;
declarations:= symbol  $\neq$  nler symbol  $\wedge$  symbol  $\neq$  semicolon
symbol; if  $\neg$  declarations then
begin lcntr:= savel; bcntr:= 2;
    in def mode:= false;
    reaffer;
    stow into stack(definitionstack,max of defstack,
        symbol); in def mode:= true
end;

```

```

within accents:= false;
for i:= i while begcntr > 0 do
  begin read while(symbol ≠ accent symbol
    ∧ symbol ≠ smaller symbol);
    if symbol = smaller symbol then compare parameters;
    if symbol = accent symbol then
      begin within accents:= 1 within accents;
        reaffer;
        if within accents then
          begin if compare({end}) then
            begin if symbol = accent symbol then
              begin begcntr:= begcntr - 1;
                if begcntr = 0 ∧ 1 declarations
                  then
                    begin delete symbol(i);
                      for i:= i while i ≠ n1cr symbol
                        ∧ i ≠ semicolon symbol ∧ 1 empty
                          do delete symbol(i);
                            empty:= false
                        end
                      end
                    end else
                      if compare({begin}) then
                        begin if symbol = accent symbol then
                          begcntr:= begcntr + 1
                        end
                      end
                    end
                  end
                end
              end;
            in def mode:= false; reaffer;
            stow into stack(definitionstack,max of defstack,endmarker);
            if stackptr < max of namestack then
              namestack[stackptr]:= if lcntr + 1 > max of defstack
                then -1 else lcntr + 1
            end read block;
            max of formallist:= 127; empty:= false;
            for i:= i while 1 empty do
              begin read name;
                read formals;
                read block;
                read while(0 < symbol ∧ symbol ≤ 62);
                if symbol = comma symbol then
                  begin reaffer;
                    read while(symbol = n1cr symbol
                      ∨ symbol = semicolon symbol);
                    ERROR(symbol < 10 ∨ symbol > 62,3030)
                  end else empty:= true
                end;
                pr tape symbol:= space symbol
              end define macro;

```

```
procedure expand macro;  
begin integer p,par;
```

```
procedure read actuals;  
begin integer i,opcntr,quotcntr,savel,auxptr;
```

```
procedure complete actual parameter;  
begin if bcntr = 0 then  
  begin lcntr:= lcntr - 1; bcntr:= 2 end else  
  begin actualstack[lcntr]:= actualstack[lcntr] : t8;  
  bcntr:= bcntr - 1  
  end;  
  stow into stack(actualstack,max of actualstack,  
  endmarker);  
  freeptr:= freeptr + 1;  
  if freeptr < max of pointerstack then  
    begin savel:= lcntr;  
    pointerstack[freeptr]:= lcntr + 1  
  end  
end complete actual parameter;  
  
auxptr:= freeptr;  
if symbol = open symbol then  
  begin in actual mode:= true;  
    opcntr:= 1;  
    for i:= 1 while opcntr > 0 do  
      begin ERROR(freeptr > max of pointerstack,3013);  
      if freeptr = auxptr then  
        begin savel:= lcntr:= pointerstack[freeptr] - 1;  
        bcntr:= 2  
      end;  
      reaffer;  
      if symbol = open symbol then  
        opcntr:= opcntr + 1 else  
      if symbol = close symbol then  
        opcntr:= opcntr - 1;  
      read while(symbol = nlcr symbol ∨  
        symbol = semicolon symbol);  
      lcntr:= savel; bcntr:= 2;  
      stow into stack(actualstack,max of actualstack,  
      symbol);  
      for i:= 1 while (symbol ≠ comma symbol ∨ opcntr  
      ≠ 1) ∧ opcntr ≠ 0 do  
        begin reaffer;  
          if symbol = open symbol then opcntr:=  
            opcntr + 1 else  
          if symbol = close symbol then opcntr:=  
            opcntr - 1  
        end;  
      complete actual parameter  
    end;  
    reaffer;  
    in actual mode:= false  
  end else
```

```

if symbol = smaller symbol then
begin in actual mode:= true;
  for i:=1 while symbol = smaller symbol do
  begin ERROR(freeptr > max of pointerstack,3013);
    quotcntr:= 1;
    if freeptr = auxptr then
    lcntr:= pointerstack[freeptr] - 1 else lcntr:= save1;
    bcntr:= 2;
    for i:= 1 while quotcntr > 0 do
    begin reaffer;
      if symbol = smaller symbol then quotcntr:=
        quotcntr + 1 else
      if symbol = greater symbol then quotcntr:=
        quotcntr - 1
    end;
    complete actual parameter;
    reaffer;
    if symbol ≠ point symbol then
    begin read while(symbol = nlcr symbol
      Vsymbols = semicolon symbol);
      ERROR(symbol = point symbol,3025)
    end
  end;
  if symbol = point symbol then reaffer
  else ERROR(true,3025);
  in actual mode:= false
end;
pointerptr:= auxptr;
if freeptr - auxptr ≠ par then
begin ERROR(true,3016);
  auxptr:= auxptr + par - 1;
  for i:= freeptr step 1 until auxptr do
  pointerstack[i]:= -1;
  freeptr:= auxptr + 1
end;
if symbol ≠ nlcr symbol ∧ symbol ≠ semicolon symbol then
begin ERROR(true,3001);
  read while(symbol ≠ nlcr symbol
    ∧ symbol ≠ semicolon symbol)
end
end read actuals;

procedure store expansion;
begin savestack[saveptr]:= bbcntr;
  savestack[saveptr + 1]:= llcntr;
  if from actualstack then
  begin savestack[saveptr + 2]:= BBcntr;
    savestack[saveptr + 3]:= LLcntr;
    from actualstack:= false
  end else savestack[saveptr + 2]:= -1
end store expansion;

```

```

ERROR(saveptr + 5 > max of actualstack,3017);
p:= namestack[place of name - namestack[place of name] - 1];
par:= -definitionstack[p] - 1;
read actuals;
namestack[place of name + 1]:= -namestack[place of name + 1];
savestack[saveptr + 4]:= place of name;
savestack[saveptr + 5]:= symbol;
if from macro then store expansion else
begin from macro:= true; stow into buffer(asterisk) end;
saveptr:= saveptr + 6; bbctr:= 1; llctr:= p;
pr tape symbol:= space symbol;
symbol:= macro sym;
stow into buffer(symbol)
end expand macro;

integer procedure macro sym;
begin integer i,s;

    procedure restore expansion;
    begin bbctr:= savestack[saveptr];
        llctr:= savestack[saveptr + 1];
        if bbctr = 2 then
            begin t8j:= 1; word:= definitionstack[llctr];
                word:= word - word : t8 x t8
            end else
            if bbctr = 3 then
                begin t8j:= t8; word:= definitionstack[llctr];
                    word:= word - word : t16 x t16
                end;
            if savestack[saveptr + 2] # -1 then
                begin BBctr:= savestack[saveptr + 2];
                    LLctr:= savestack[saveptr + 3];
                    if BBctr = 2 then
                        begin t8j:= 1; Word:= actualstack[LLctr];
                            Word:= Word - Word : t8 x t8
                        end else
                        if BBctr = 3 then
                            begin t8j:= t8; Word:= actualstack[LLctr];
                                Word:= Word - Word : t16 x t16
                            end;
                        from actualstack:= true
                    end;
                    place of name:= savestack[saveptr - 2];
                    pointerptr:= pointerptr + definitionstack[namestack
                        [place of name - namestack[place of name] - 1]] + 1
                end restore expansion;

```

```

if spacecntr > 150 then
begin spacecntr:= spacecntr - 1; s:= space symbol end else
begin if from actualstack then
begin BBcntr:= BBcntr - 1; if BBcntr = 0 then
begin LLcntr:= LLcntr + 1; BBcntr:= 3;
t8j:= t16; Word:= actualstack[LLcntr]
end;
if BBcntr ≠ 1 then
begin s:= Word : t8j; Word:= Word - s × t8j;
t8j:= t8j / t8
end else s:= Word;
if s = endmarker then
begin from actualstack:= false; s:= macro sym
end
end else
begin bbcntr:= bbcntr - 1; if bbcntr = 0 then
begin llcntr:= llcntr + 1; bbcntr:= 3; t8j:= t16;
word:= definitionstack[llcntr]
end;
if bbcntr ≠ 1 then
begin s:= word : t8j; word:= word - s × t8j;
t8j:= t8j / t8
end else s:= word;
if s > 128 ∧ s < 149 then
begin from actualstack:= true; BBcntr:= 1;
LLcntr:= pointerstack[pointerptr + s - 128] - 1;
if LLcntr = -2 then from actualstack:= false;
s:= macro sym
end else
if s = endmarker then
begin saveptr:= saveptr - 6;
freeptr:= pointerptr;
namestack[place of name + 1]:=
-namestack[place of name + 1];
if saveptr = -5 then
begin from macro:= false; stow into buffer(ksiretsa)
end else restore expansion;
s:= savestack[saveptr + 5]
end
end;
if s > endmarker then
begin spacecntr:= s - 1; s:= space symbol end
end;
macro sym:= s
end macro sym;

```

```
integer procedure reaffer;
begin integer i;

    integer procedure read and buffer;
    begin integer s;
        s:= RESYM1;
        if in actual mode then
            begin stow into stack(actualstack,max of actualstack,s);
                prsym(s);
                if s = nlcr symbol then space(7)
            end else
            begin stow into buffer(s);
                if s = nlcr symbol then
                    line number:= line number + 1
            end;
            read and buffer:= s
        end read and buffer;

    for i:= i, i while symbol = space symbol V symbol = tab symbol do
        begin if accent read then
            begin symbol:= nextacc; accent read:= false end
            else symbol:= read and buffer;
            if symbol = accent symbol then
                begin nextacc:= read and buffer;
                    if nextacc = accent symbol
                        then symbol:= apostrophe symbol
                        else accent read:= true
                end;
            if symbol = apostrophe symbol then
                for i:= i while symbol ≠ semicolon symbol
                    ^ symbol ≠ nlcr symbol do
                    symbol:= read and buffer;
                    if in def mode then
                        stow into stack(definitionstack,max of defstack,symbol)
                    end;
                    reaffer:= symbol
                end read and buffer;

    boolean procedure compare(text); string text;
    begin integer s,k;
        k:= 0; compare:= true;
        for s:= stringsymbol(k,text) while s ≠ 255 do
            if s ≠ (if 37 < symbol ∧ symbol < 62 then symbol - 27 else symbol)
            then
                begin compare:= false; k:= -1 end else
                begin k:= k + 1; if first scan then reaffer else NS end
            end compare;

    procedure read while(condition); boolean condition;
    begin integer i;
        for i:= i while condition do if first scan then reaffer else NS
    end read while;
```

```

procedure skip until(text); string text;
begin integer i, first symbol;
    first symbol:= stringsymbol(0,text);
    read while(first symbol ≠
        (if 37 < symbol ∧ symbol < 62 then symbol - 27 else symbol));
    for i:= 1 while 1 compare(text) do read while(first symbol ≠
        (if 37 < symbol ∧ symbol < 62 then symbol - 27 else symbol))
end skip until;

procedure stow into stack(stack,max,char); value max,char;
integer max,char; integer array stack;
begin integer i;
    if char = space symbol ∧ spacecntr < 255
    then SPACEcntr:= SPACEcntr + 1 else
    begin bcntr:= bcntr + 1;
        if bcntr = 3 then
        begin lcntr:= lcntr + 1; bcntr:= 0;
            if lcntr > max then ERROR(true,3018)
            else stack[lcntr]:= 0
        end;
        if SPACEcntr > 150 then
        begin stack[lcntr]:= stack[lcntr] × t8 + SPACEcntr;
            if char = space symbol then SPACEcntr:= 151 else
            begin SPACEcntr:= 150;
                stow into stack(stack,max,char)
            end
        end else
        stack[lcntr]:= stack[lcntr] × t8 + char;
        if char = endmarker then
        for i:= bcntr step 1 until 1 do
            stow into stack(stack,max,0)
        end
    end
end stow into stack;

procedure store letgits(list,pointer,max,letgit); value max;
integer pointer,max,letgit; integer array list;
begin integer i,j,word;
    boolean full;
    word:= j:= 0; full:= false;
    for i:= 1 while symbol < 62 ∧ 1 full do
    begin if symbol > 37 then symbol:= symbol - 27;
        j:= j + 1;
        if j = 4 then
        begin if pointer > max then full:= true else
            list[pointer]:= word × t6 + symbol;
            word:= j:= 0; pointer:= pointer + 1
        end else word:= word × t6 + symbol;
        symbol:= letgit
    end;
    if j ≠ 0 then
    begin for j:= j + 1 while j < 4 do word:= word × t6 + 63;
        if pointer > max then full:= true else list[pointer]:= word;
        pointer:= pointer + 1
    end;
    ERROR(full ∧ 1 in def mode,3019)
end store letgits;

```

```

procedure unstack macros;
begin integer i;
  for i:= i while abs(namestack[stackptr - 1]) = blocknumber  $\wedge$ 
    stackptr > 0 do
    stackptr:= stackptr - namestack[stackptr - 2] - 3
  end unstack macros;

procedure skip macro declarations;
if second scan then
begin integer i, begcntr;
  for i:= i, i while symbol = comma symbol do
  begin skip until( $\langle$ 'begin' $\rangle$ ); begcntr:= 1;
    for i:= i while begcntr > 0 do
    begin read while(symbol  $\neq$  accent symbol
       $\wedge$  symbol  $\neq$  apostrophe symbol);
      if symbol = accent symbol then
      begin NS;
        if symbol = accent symbol
        then symbol:= apostrophe symbol else
        if compare( $\langle$ end' $\rangle$ ) then begcntr:= begcntr - 1 else
        if compare( $\langle$ begin' $\rangle$ ) then begcntr:= begcntr + 1 else
        else begin read while(symbol  $\neq$  accent symbol);
          NS
        end
      end;
      if symbol = apostrophe symbol then
      read while(symbol  $\neq$  nlcr symbol
         $\wedge$  symbol  $\neq$  semicolon symbol)
      end;
      read while(0  $\leq$  symbol  $\wedge$  symbol  $\leq$  62)
    end
  end skip macro declarations;

procedure print elantext;
begin integer i, begcntr;
  pr tape symbol:= space symbol;
  linecounter:= 0;
  skip until( $\langle$ 'begin' $\rangle$ ); begcntr:= 1;
  for i:= i while begcntr > 0 do
  begin read while(symbol  $\neq$  accent symbol
     $\wedge$  symbol  $\neq$  apostrophe symbol);
    if symbol = accent symbol then
    begin NS;
      if symbol = accent symbol
      then symbol:= apostrophe symbol else
      if compare( $\langle$ end' $\rangle$ ) then begcntr:= begcntr - 1 else
      if compare( $\langle$ begin' $\rangle$ ) then begcntr:= begcntr + 1 else
      begin read while(symbol  $\neq$  accent symbol); NS end
    end;
    if symbol = apostrophe symbol then
    read while(symbol  $\neq$  nlcr symbol  $\wedge$  symbol  $\neq$  semicolon symbol)
    end;
    runout; runout
  end print elantext;

```

#### 14. Literatuur

- [1] R.P. van de Riet, L.G.L.Th. Meertens, G.C.J.M. Nogaredé, L.J.M. Geurts, M. Rem, The MC ELAN Assembler, Stichting Mathematisch Centrum, Amsterdam 1971.
- [2] F.E.J. Kruseman Aretz, J.B. Mailloux, E.G.M. Broerse, K.K. Koksma. Verslag bespreking specificatie eigenschappen eventuele MC-Assembler X8. Intern rapport, februari-april 1967.
- [3] Programmering EL X8, N.V. Philips-Electrologica, Den Haag.
- [4] M.D. McIlroy. Macroinstruction Extensions of Compiler Languages. Communications of the ACM, vol. 3, no. 4, pp. 214-220, april 1960.
- [5] D.W. Barron. Assemblers and Loaders. Mc.Donald/Elsevier, Londen 1969.
- [6] D. Grune. Handleiding Milli-systeem voor de EL X8. Stichting Mathematisch Centrum, Amsterdam 1970.
- [7] P. Naur (editor). Revised Report on the Algorithmic Language ALGOL 60. Regnecentralen, Kopenhagen 1962.