RA

stichting
mathematisch
centrum

MC

REKENAFDELING MR 127/71 NOVEMBER

RA

C.H.A. KOSTER
A COMPILER COMPILER

2e boerhaavestraat 49 amsterdam

BIBLIOTHEEK MATHEMATISCH CENTRUM
AMSTERDAM

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

0. Introduction.
   0.1 Notation and terminology of CF grammars.
   0.2 CF grammars of type LL(1).
1. Syntax-directed compiling.
   1.1 What is a compiler compiler?
   1.2 Parsing according to a CF grammar.
      1.2.1 The first parsing methode.
      1.2.2 A second parsing method.
      1.2.3 Comparison of the two methods.
   1.3 Extensions to CF syntax.
      1.3.1 Embedding actions into the syntax.
      1.3.2 Affixes.
      1.3.3 Grouping; jumps and labels.
      1.3.4 Some more syntactic sugar.
   1.4 What is the use of a Compiler Compiler?
2. About this compiler compiler.
   2.1 Compiler description.
   2.2 Symbols.
   2.3 Specifications.
   2.4 Declarations.
   2.5 Commands.
   2.6 Comments.
   2.7 Rules.
   2.8 Starting symbol.
   2.9 Diagnostics.
3. Examples.
   3.1 Reading a number.
   3.2 Updating a chain.
   3.3 Reading tags.
   3.4 Printing, errormessages.
   3.5 An input adminsitration.
   3.6 Reading declarers.
   3.7 Collecting defining occurrences.

4. Compiler Compiler described in CDL.
4.1 General environment.
4.1.1 Interface with machine.
4.1.2 Stacks.
4.1.3 Macros.
4.1.4 Pointers and flags.
4.2 Output.
4.2.1 Printer section.
4.2.2 Trace administration.
4.2.3 Card punch section.
4.2.4 Result administration.
4.3 Input.
4.3.1 Reading characters.
4.3.2 Tag symbols.
4.3.3 Bold symbols.
4.3.4 Special symbols.
4.3.5 Constants.
4.3.6 Input administration.
4.4 Compiler compiler to ALGOL 60.
4.4.1 Auxiliary actions.
4.4.2 Treatment of types.
4.4.3 Specifications.
4.4.4 Declarations.
4.4.5 Affix expressions.
4.4.6 Building stones of a rule.
4.4.7 General form of a rule.
4.4.8 Other building stones of a grammar.
4.4.9 Terminals.
4.4.10 Post mortem.
References.

0. Introduction.

The purpose of this report is to give an informal description of a compiler compiler for languages defined by means of a Compiler Description Language based on affix grammars, together with a description of that compiler compiler itself in that Compiler Description Language (CDL). Affix grammars have been described formally in [2].

In the first chapters of this report, an explanation is given of the use of this compiler compiler. This explanation is neither very formal nor very complete, but should enable the reader to understand the description given in the last chapter of the report.

The compiler compiler is a syntax-directed compiler of the top-to-bottom variety equipped with a macro mechanism, which accepts input written in a two-level extension of CF grammar, giving as output a program in ALGOL 60. It is intended as a tool in the study and production of compilers for high level languages, such as ALGOL 68.

0.1 Notation and terminology.

A Context-Free grammar (CF grammar) consists of two distinct finite collections of symbols (,termed the collections of "nonterminal" and "terminal" symbols respectively), and furthermore of one specific nonterminal, its "starting symbol", and a finite collection of "rules".

Each rule associates with some nonterminal its "alternatives", a number of possibly empty sequences of nonterminals and terminals.

In writing down a grammar, we will write down one rule for each nonterminal, with that nonterminal as its "left-hand-side", followed by a colon (:), followed by the "right-hand-side" of the rule, followed by a point (.). The right-hand-side consists of the alternatives of the left-hand-side, each separated from the next by a semicolon (;). The symbols within one alternative are separated by commas (,). As symbols we will use only "tags", i.e., sequences of letters and digits beginning with a letter.

The notation just described, the "Van Wijngaarden notation" [3], is a variation on the more common Backus-Naur form [4], with as advantages ease of writing and reading, and the fact that the end of a rule is clearly indicated.

By a "direct production" of a nonterminal we mean any of its alternatives. By a "production" of a nonterminal x we mean a sequence of symbols, which is either a direct production of x, or is the result of replacing, in some production of x, a nonterminal y by a direct production of y.

By a "terminal production" of a nonterminal x we mean a production of x that contains no nonterminals.

By a "sentence" of a CF grammar we mean a terminal production of its starting symbol.

By the "language" of a CF grammar G we mean the collection of its sentences. This language is denoted by L(G).

0.2 CF grammars of type LL(1).

By the productions of a sequence of symbols y we mean the productions of a new nonterminal x with y as its only direct production, with the exclusion of y itself.

For an alternative x, we will indicate by first (x) the set of terminal symbols which are the first symbol of some production of x, and by follow (x) the set of terminal symbols which immediately follow x in any production of the starting symbol.

We then say a CF grammar is "of type LL(1)" [6] if it satisfies the following 3 conditions:

1) For any rule A: a_1; a_2; ...; a_n. the sets first (a_1), first (a_2), ..., first (a_n) are mutually disjoint.
2) At most one of a_1, ..., a_n can produce an empty string ε.
3) If a_p produces ε, then first (a_q) has no elements in common with follow (A).

It is decidable whether a given CF grammar is of type LL(1). Without loss of generality we may assume that only the last alternative can produce ε (condition 2).

1. Syntax-directed compiling.

1.1 What is a compiler compiler?

A "program" is a text in some programming language more or less directly executable on a computer, which, when executed with some input, gives some output.

A "compiler" for some high level language is a program which, when executed with a program in that language as input, gives as output a program in some other language (preferably more directly executable).

A "compiler compiler" is a compiler which, when executed with a formal description of a compiler for some language as input, gives as output a compiler for that language.

A "family of compiler compilers" is a collection of compiler compilers which, when executed with identical input, give as output equivalent programs, possibly in different languages.

Ideally, a formal description of some programming language should be capable of yielding also a formal description of a compiler for it; certainly a formal description of a compiler may be used to define a language. The formalism for definition of a language consists of two coupled mechanisms, its syntax and semantics. Strictly speaking, the syntax should indicate whether a given sequence of symbols is a program in the language, and if so, derive a parsing tree for it, while the semantics should attach a meaning to the parsing tree of a correct program.
In the definition of most languages, for instance that of ALGOL 60, this separation is not so clean cut: one can argue that part of the syntax is given under the heading of semantics. In ALGOL 68, the "context-conditions" are an example of syntactical restrictions on programs appearing as semantics, so that the syntax by itself is not sufficient to define what is a proper program.

Formalization of the syntax of programming languages is now well enough understood to obtain part of a compiler by mechanical means, using "syntax-directed" techniques. On the other hand,there exists no formalization of the semantics which could usefully give the "semantics-directed" parts of a compiler. For this, manual techniques are used, for instance by translating to some set of macros which then reflect the semantics.

Our goal is to extend the applicability of syntax-directed techniques so far that all but the very kernel of the semantics of a compiler can be defined by syntax, so that only that kernel remains to be defined by other means. To this end, a notational system, "Compiler Description Language" (CDL) will be introduced, strongly based on an extension of CF grammars ("affix grammars"), a kind of grammars which, on the one hand, are powerful enough to define major programming languages, and, on the other hand, lend themselves to syntax-directed compiling techniques.

In the light of the foregoing discussion the Compiler Description Language can best been seen as a programming language, which allows one, at a high level of abstraction, to write compilers in a machine independent way.

Although it is our intention to describe a family of compiler compilers, the discussion will center around one particular member of that family, whose input is in Compiler Description Language, and whose output is in ALGOL 60. ALGOL 60 as an output language provides a clear enough framework for the notions involved while being broadly available. ALGOL 68 would certainly provide a better framework, but its availability is still very limited.

Other members of the same family of compiler compilers are being constructed, giving output in various assembly languages and in ALGOL 68.

1.2 Parsing according to a CF grammar.

The basis of a syntax-directed compiler is a parser, so we must first go into matters of parsing.

By a "parse" of a sentence according to a grammar we mean a sequence of substitutions of a direct production for some nonterminal, which leads from the starting symbol of the grammar to that sentence. The problem of parsing a given sequence of symbols then consists of

1) deciding whether that sequence of symbols is a sentence of the grammar;
2) if so, then constructing a parse for it.

Using a CF grammar as a generating device is simple enough, but it takes more trouble to find a practical algorithm for parsing, even though the parsing problem can easily be shown to be decidable. Still, at the moment the parsing problem for CF grammars is wholly solved, and literature abounds with parsing methods.

These methods can be classified along various criteria:

a) bottom up <u>or</u> top down
b) general algorithm & table for the particular grammar <u>or</u> particular parser for that grammar only
c) parser for general CF grammars <u>or</u> parser for a reduced class obtained by imposing restrictions on the grammars.

The compiler compiler to be described generates a top down particular parser, written in ALGOL 60, for a CF grammar which satisfies the LL(1) conditions.

From a grammar, the particular parser is obtained by a transscription process where each rule is turned into a <declaration> for a parsing procedure corresponding to the nonterminal in its left-hand-side; these procedures are then embedded in an environment, which contains at least some as yet unspecified means of input and output. Several such transcriptions are possible, and we will investigate two of them.

1.2.1 The first parsing method.

The transcription process is best introduced by an example. Assume the following part of a grammar:

```
number: digit, numbertail.                                    G1
numbertail: digit, numbertail; endmarker.
```

Its transcription would run:

```
Boolean procedure number;                                     P1
begin if ¬ digit then goto 1;
      if ¬ numbertail then goto 0;
      number:= true; goto end;
 1:0: number:= false; end:
end;
```

```
Boolean procedure numbertail;
begin if ¬ digit then goto 1;
      if ¬ numbertail then goto 0;
      numbertail:= true; goto end;
   1: if ¬ endmarker then goto 2;
      numbertail:= true; goto end;
 2:0: numbertail:= false; end:
end;
```

We will assume *digit* and *endmark* to be procedures declared in the environment, such that they return true if the current symbol in the input-stream is a digit or an endmark respectively, simultaneously advancing the input by one symbol, and return false otherwise.

When we now call *number* it returns either true or false, depending on whether the input stream, starting with the current symbol, contained a sequence of digits followed by an endmarker.

Now consider another CF grammar for number 1:

number 1: digit, number 1; digit, endmarker. G2

with transcription:

P2.a

```
Boolean procedure number 1;
begin if ¬ digit then goto 1;
      if ¬ number 1 then goto 0;
      number 1:= true; goto end;
   1: if ¬ digit then goto 2;
      if ¬ endmarker then goto 0;
      number 1:= true; goto end;
 2:0: number 1:= false; end:
end;
```

The first transcription method fails to parse any number 1. Consider, e.g., *9 ⧺*, where *9* is a digit and ⧺ the endmarker. It finds the string begins with a digit, so advances the input, and then tries to parse the remainder as a number 1, which fails; so number 1 returns false.

Clearly, there are CF grammars for which method 1 does not work satisfactorily.

1.2.2 A second parsing method.

The trouble with grammar G2 had to do with the fact that, upon meeting a digit, the input could not safely be advanced. Assume the environment to contain an input pointer *pin*, indicating at every moment the current symbol of an input array. Advancing the input is done by incrementing *pin*, so backtracking the input can be done by restoring *pin* to its original value.

The second transcription method transforms G2 into

```
Boolean procedure number 1;                                P2.b
begin integer pold; pold:= pin;
      if ¬ digit then goto 1;
      if ¬ number 1 then goto 1;
      number 1:= true; goto end;
   1: pin:= pold;
      if ¬ digit then goto 2;
      if ¬ endmarker then goto 2;
      number 1:= true; goto end;
   2: pin:= pold; number 1:= false; end:
end;
```

This second version will recognize the string *9 #* as follows: it finds *9* is a digit and increments *pin*; it finds *#* is not a number 1, so control goes to the label *1*; *pin* is restored to its old value; a digit is found, incrementing *pin*; an endmarker is found, again incrementing *pin* and the procedure returns true.

Clearly this second transcription method works for a wider class of grammars than the first did.

1.2.3 Comparison of the two methods

In this section we will compare the merits of the two methods and the conditions under which they can be used.

First some terminology: We will term a procedure *x*, yielding a Boolean value, and obtained from a nonterminal x by one of the transcription processes, a "parsing procedure" for x.

We say a parsing procedure *x* "recognizes" the nonterminal x if it satisfies the following two requirements:

1) *x* always terminates;
2) if the inputstream, starting with the current symbol, contains a terminal production of x, then *x* yields <u>true</u>, simultaneously advancing the input to the first symbol after that terminal production; otherwise it yields <u>false</u>.

We will say *x* "exactly recognizes" x when *x* recognizes x and advances the inputstream only when returning <u>true</u>.

In this terminology *digit* is a parsing procedure exactly recognizing a digit. By induction on the length of the inputstream, one can deduce that *number* and *numbertail* recognize number and numbertail, though not exactly, as shown by the inputstring *1a* where *a* is not an endmarker : number 1 will return <u>false</u>, but has then already advanced the input.

According to the first method, number 1 (P2.a) fails to recognize anything, but the second version (P2.b) exactly recognizes number 1.

For convenience we will term the first transcription method "nonrestoring" and the second "restoring".

We can make a number of observations concerning the two transcription methods:

1) For a CF grammar its <u>nonrestoring</u> parser <u>recognizes</u> its starting symbol if and only if that grammar satisfies the LL(1) conditions. The first condition assures that, upon recognizing a terminal symbol, the input can safely be advanced without a chance of taking a wrong alternative. The second and third condition assure that no problems arise because of empty terminal productions.
2) For a CF grammar, its <u>restoring</u> parser <u>exactly recognizes</u> its starting symbol if that grammar satisfies the LL(1) conditions. These conditions assure that an alternative taken and found present is indeed the only correct one. They are sufficient, but not necessary: the class of grammars recognized exactly contains the LL(1) grammars.

The difference between the restoring and nonrestoring parser for a LL(1) grammar comes out only for input which does not form a sentence: while the restoring parser rejects it (recognizing exactly), the non-restoring parser may reject it but still have advanced the input, or even accept it, as is clear from the example:

statement: ass stat; if stat. G3.a
ass stat: identifier, becomes symbol, expression.
if stat: if symbol, bool expr, then symbol, statement.

With statement as starting symbol, and appropriate exact recognizers for identifier, becomes symbol, etc., the nonrestoring parser for G3 would accept not only

v:= 0
and *if v > 2 then v:= v + 1*
but also *v if v > 2 then v:= v + 1*

without giving any error message. A nonrestoring parser may accept erroneous sentences without warning. This deficiency must be mended by explicitly taking possible errors into account while constructing the syntax:

ass stat: identifier, rest ass stat. G3.b
rest ass stat: becomes symbol, expression; errormessage.

Errormessage should print out some appropriate warning.

Advantages of the nonrestoring parser over the restoring parser are:

1) Time-efficiency: the parser has less work to do than the restoring parser, which may have to perform lots of backtracking.
2) Memory-efficiency: because backtracking can never occur, there is no need for any array to hold all of the input, and no need for temporary storage to retain old values of *pin*.

The disadvantage of the nonrestoring parser lies in the restrictions it imposes on the grammar.

In a practical situation, the advantages of the nonrestoring parser may outweigh its disadvantage: when constructing a syntax, one has good grounds to make it LL(1).

The compiler compiler allows one to choose between the two parsing methods, translating some rules in a restoring, others in a nonrestoring fashion. This facility is essential for making compilers as efficient as possible, but makes explaining the workings of the compiler compiler somewhat complicated. For this reason we will show mainly nonrestoring parsers in the rest of this chapter, leaving the construction of the restoring version to the reader.

1.3 Extensions to CF syntax.

In this section we will give an account of the extensions that have to be made to CF syntax in order to turn it into an appropriate input-language for the compiler compiler.

1.3.1 Embedding actions into the syntax.

In order to turn a parser into a compiler, one has to provide it with means to perform, as a side-effect of parsing, some semantic actions. To accomplish this, these actions are embedded in the syntax as follows: Consider the parsing of a number with as translation its value:

```
number 2: digit, action 1, numbertail 2.                    G4
numbertail 2: digit, action 2, numbertail 2; endmarker.
```

where *action 1* assigns the value of the last digit read to some global variable *s* while *action 2* multiplies *s* by *10* and then adds the value of the last digit read to *s*.

We divide the nonterminals into "predicates" and "actions", where predicates are transscribed as described in 1.2, while actions are transscribed as insertion of a procedure call at the corresponding place of the parser:

```
Boolean procedure number 2;                                   P4
begin if ¬ digit then goto 1;
      action 1;
      if ¬ numbertail 2 then goto 0;
      number 2:= true; goto end;
  1:0: number 2:= false; end:
end;

Boolean procedure numbertail 2;
begin if ¬ digit then goto 1;
      action 2;
      if ¬ numbertail 2 then goto 0;
      numbertail 2:= true; goto end;
   1: if ¬ endmarker then goto 2;
      numbertail 2:= true; goto end;
  2:0: numbertail 2:= false; end:
end;
```

We will allow "primitive" actions and predicates to be defined not by a rule of the grammar but by some other means, e.g., macros. If we take as such primitive actions:

alpha = *s:= 0*
beta = *s:= 10 * s*
and gamma = *s:= s + last digit read*

then we can define the actions action 1 and action 2 in terms of those primitive actions:

action 1: alpha, gamma. G5
action 2: beta, gamma.

with transcription:

```
procedure action 1; begin s:= 0; s:= s + last digit read end;
procedure action 2; begin s:= 10 * s; s:= s + last digit read end;
```

An example of a primitive predicate could be digit.

The transcription of a rule for an action is the <declaration> of a procedure (not a Boolean procedure) which may involve again actions and predicates, primitive or not primitive. More involved examples will follow.

1.3.2 Affixes.

The primitive actions alpha, beta and gamma mentioned in 1.3.1 are neither really primitive nor very practical: they involve a global variable *s* which may not be used for other purposes by other actions, e.g., some action taken by digit.

We want to provide parsing procedures with parameters, as a mechanism for communicating information to and from other parsing procedures.

To this end, a nonterminal may be accompanied by a number of "affixes" in the form of symbols written after it in the syntax and each preceded by a plus (+), e.g.:

```
number 3 + value:                                          G6
        digit, action 3, numbertail 3 + value.
numbertail 3 + value:
        digit, action 4, numbertail 3 + value;
        endmarker.
```

Where action 3 and action 4 are primitive actions:

action 3 = *value:= last digit read*

and action 4 = *value:= 10 * value + last digit read*

with transcription:

```
Boolean procedure number 3 (value); integer value;         P6
begin if ¬digit then goto 1;
        value:= last digit read;
        if ¬numbertail 3 (value) then goto 0;
        number 3:= true; goto end;
  1:0: number 3:= false; end:
end;
```

```
Boolean procedure numbertail 3 (value); integer value;
begin if ¬ digit then goto 1;
      value:= 10 * value + last digit read;
      if ¬ numbertail 3 (value) then goto 0;
      numbertail 3:= true; goto end;
   1: if ¬ endmarker then goto 2;
      numbertail 3:= true; goto end;
 2:0: numbertail 3:= false; end:
end;
```

Note that *value* is an output parameter. All parameters are called by name. Of course, also primitive actions and predicates can have parameters.

Apart from parameters, we will also need local variables, which we will indicate in the left-hand-side of the rule as symbols each preceded by a minus (-), e.g.:

```
number 4 + value - d:                                              G7
      digit 4 + d, action 5 + value + d, numbertail 4 + value.
numbertail 4 + value - d:
      digit 4 + d, action 6 + value + d, numbertail 4 + value;
      endmarker.
digit 4 + d: digit, action 5 + d + last digit read.
```

where

action 5 + x + y = *x:= y*

and action 6 + x + y = *x:= 10 * x + y*

with transcription

```
Boolean procedure number 4 (value); integer value;                 P7
begin integer d;
      if ¬ digit 4 (d) then goto 1;
      value:= d;
      if ¬ numbertail 4 (value) then goto 0;
      number 4:= true; goto end;
 1:0: number 4:= false; end:
end;
```

```
Boolean procedure numbertail 4 (value); integer value;
begin integer d;
      if ¬ digit 4 (d) then goto 1;
      value:= 10 * value + d;
      if ¬ numbertail 4 (value) then goto 0;
      numbertail 4:= true; goto end;
   1: if ¬ endmarker then goto 2;
      numbertail 4:= true; goto end;
 2:0: numbertail 4:= false; end:
end;

Boolean procedure digit 4 (d); integer d;
begin if ¬ digit then goto 1;
      d:= last digit read;
      digit 4:= true; goto end;
   1: digit 4:= false; end:
end;
```

Those symbols which occur in the grammar to indicate a parameter or local variable are affixes, and must be distinct from all other symbols.

Affixes occurring in the left-hand-side of a rule we term "bound" affixes of that rule if they are preceded by a plus and "free" affixes if preceded by a minus. Affixes occurring in a rule which are neither bound nor free are "terminal".

For sake of simplicity we assume the bound, free and terminal affixes to be disjoint sets. If a bound affix is used as an input parameter to a parsing procedure we will term it "inherited"; otherwise, it is "derived".

A nonterminal must, when occurring in a right-hand-side, always be accompanied by as many affixes as there are bound affixes in the left-hand-side of its defining rule.

Of course, care has to be taken that an inherited affix should get a value before being used. In an article of D.E. Knuth [5] it is shown how to investigate by graph-theoretical means whether a system of inherited and derived affixes is consistent (in his terminology, whether a system of inherited and synthesized attributes is well defined).

A nonterminal together with its affixes is termed an "affix expression", of which the nonterminal is the "handle". The handle with its appended affixes, in a definition of an affix grammar as a generating device, takes the same place as a nonterminal in the definition of a CF grammar. Affixes derive their name from the fact that they are considered to be attached to the handle of an affix expression.

1.3.3 Grouping; jumps and labels.

Consider again G2:

number 1: digit, number 1; digit, endmarker.

When recognizing a number 1, a restoring parser for this grammar has to recognize the the last digit twice. The grammar can be rewritten to G1 in order to obviate this backtracking, but then we must introduce an extra nonterminal numbertail. When we have recognized the first digit of a number 1, we expect either an endmarker or another number 1; this could be denoted as

```
number 5: digit,                                        G8
      (number 5; endmarker).
```

with as transcription:

```
Boolean procedure number 5;                             P8.a
begin if ¬ digit then goto 1;
      begin if ¬ number 5 then goto 2;
            number 5:= true; goto end;
         2: if ¬ endmarker then goto 3;
            number 5:= true; goto end;
         3: goto 0;
      end;
 1:0: number 5:= false; end:
end;
```

The restoring version of this parser runs:

```
Boolean procedure number 5;                                    P8.b
begin integer pold; pold:= pin;
      if ¬ digit then goto 1;
      begin integer pold; pold:= pin;
            if ¬ number 5 then goto 2;
            number 5:= true; goto end;
         2: pin:= pold;
            if ¬ endmarker then goto 3;
            number 5:= true; goto end;
         3:
      end;
   1: pin:= pold; number 5:= false; end:
end;
```

As a restoring parser, P8.b recognizes the same language as P2.b but without recognizing the last digit twice. As a nonrestoring parser, P8.a succeeds where P2.a failed; consequently G8 can be considered as of type LL(1). The introduction of grouping brackets allows one to construct grammars which can be parsed more efficiently, and to enlarge the applicability of nonrestoring parsers.

Still, the parser for number 5 is inefficient because it is an unnecessarily recursive procedure; the recursion in number 5 is used only to effectuate repetition: a number consists of a first digit, optionally followed by some number of digits, followed by an endmarker. We will need a device for performing repetition. Therefore we will allow both labels and jumps within one rule of a grammar, indicating a label as a symbol acting as label-identifier followed by a colon, and a jump by a colon followed by such a symbol, e.g. (with label *rep*):

```
number 6:                                                      G9
      digit,
      rep: (digit, :rep; endmarker).
```

with transcription:

```
Boolean procedure number 6;                                  P9
begin if ¬ digit then goto 1;
 rep: begin if ¬ digit then goto 2;
            goto rep;
         2: if ¬ endmarker then goto 3;
            number 6:= true; goto end;
         3: goto 0;
      end;
 1:0: number 6:= false; end:
end;
```

A jump is an action and may only occur at the end of some alternative, to a label within that same rule.

Under these conditions, the introduction of jumps and labels can easily be understood as shorthand for a corresponding recursive definition, so that the language accepted by a parser is unchanged, and none of the problems associated with the unrestricted introduction of jumps and labels is encountered. As an example, the recursive counterpart of number 6 is simply:

```
number 7: digit, rep.
rep: digit, rep; endmarker.
```

1.3.4 Some more syntactic sugar.

In order to turn affix grammars into a Compiler Description Language, some more features are needed.

Apart from parameters and local variables, parsing procedures may require global variables. The variable *pin* used by restoring parsers is an example. For the sake of simplicity we will use <u>integer</u> variables only, as a very general and useful datatype. Thus, for a pointer pin, a <declaration> will appear in the parser, as

integer pin;

Furthermore, a facility is needed to declare <u>arrays</u>, e.g. for holding input, and, as we will see later, to be used as stacks. Lastly, a facility is needed for declaring a Boolean variable, termed a "flag", which may then occur instead of a predicate. In 2.4 the precise notation for such declarations is given. In 2.3 it is described how one can indicate to the compiler compiler that a specific symbol is an action or predicate, defined globally, and thus to be used without further definition.

A simple mechanism for specifying macros is also described in 2.4.

1.4 What is the use of a compiler compiler?

The conventions introduced in the previous sections, and described in chapter 2, allow one to program the input to the compiler compiler in a reasonable level detail giving a reasonable efficient compiler, where especially one can leave many administrative details to the system. Of course, ALGOL 60 is not the most practical language to write compilers in. But also in the hardest machine code, subroutines (with local variables and parameters kept on a stack), global variables, labels and jumps etc. have their counterparts, so a compiler compiler can be written towards every decent machine code.

The primitive actions and predicates are not defined in the Compiler Description Language itself, but are borrowed from some other language in the form of macros. These primitive actions and predicates therefore have to be chosen with particular care, so that they can be implemented in an equivalent way on any reasonable computer.

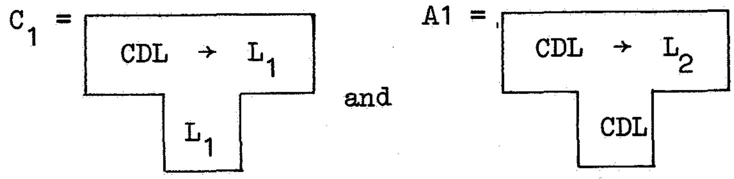
The Compiler Description Language allows one to define, in a machine independent fashion, how to construct recursively more involved actions and predicates out of the primitive ones, and out of other actions and predicates.

A compiler description in CDL describes not one single compiler, but a collection of these, one for every language to which a compiler compiler exists.

In particular, because the compiler compiler itself is defined in CDL (chapter 4) it is possible to obtain a compiler compiler to another language, and written in that other language, by a process of bootstrapping [7]:

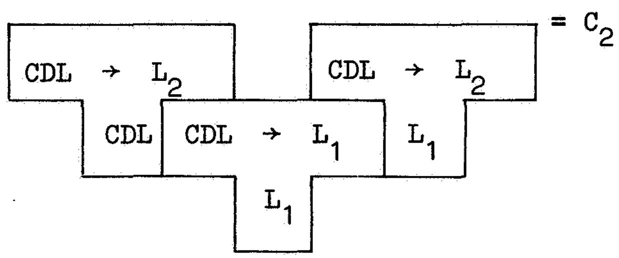
Preparation Assume we have a compiler compiler C_1, written in L_1, which turns a compiler description (= text in CDL) into a compiler written in L_1 again. We then construct a compiler description A1 of a compiler compiler which turns a compiler description into a compiler written in L_2.

In pictures:



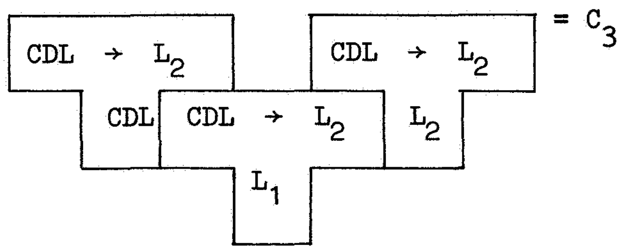
Step 1 We execute C_1 with A1 as input, giving as output C_2, which is a compiler compiler written in L_1, turning a compiler description into a compiler written in L_2.

In a picture:



Step 2 We execute C_2 with A1 as input, giving as output C_3, which is a compiler compiler written in L_2, turning a compiler description into a compiler written in L_2.

In a picture:



Using CDL for writing some compiler is preferable both to using machine-code and to using a general purpose high-level language as, e.g., ALGOL 60, for the following reasons:

- Compiler Description Language forms a very high framework for writing compilers, where one is little bothered with the administrative details of the parsing process.

- A Compiler Description is highly machine-independent and transferable, since a compiler can be obtained in any object language for which a compiler compiler exists. Because of their simplicity, compiler compilers for CDL can easily be made compatible.
- There is the possibility of first testing the compiler description using a high-level object code with good diagnostics, which is guaranteed never to blow up the system, and later to use a version in machine-code for production.
- By forcing oneself to use only specific programming tools, there is less possibility for tricky programming, using local features of some system which make transfer impossible.
- A Compiler Description is well enough readable to be of help in documentation.

Of course a carefully handwritten program can be more efficient both in time and in space than the mechanically obtained result of a compiler compiler. But one must not exagerrate the importance of this point: by a suitable choice of macros and careful construction of the crucial part of the syntax, one can get as near to machine-code as one should wish.

On a more general level, the Compiler Description Language can be seen as a mechanism for the syntactical composition of semantics, which should be suitable for the definition of programming languages, not by an in principle generative grammar plus verbally formulated semantics, but as a system describing the meaning of a program in terms of very simple and basic semantics actions, i.e., a machine-independent compiler.

2. About this compiler compiler.

In this chapter, a description is given of the Compiler Description Language, in the form of a context-free grammar exhibiting its syntax, accompanied by semantic remarks. For a precise definition, the description of the compiler compiler itself is to be found in chapter 4.

2.1 Compiler description.

```
compiler description:
      specification, compiler description;
      declaration, compiler description;
      command, compiler description;
      comment, compiler description;
      rule, compiler description;
      starting symbol.
```

A compiler description consists of various building stones; some of them provide only information to the compiler compiler (as specifications and commands), others give rise to a translation. The compiler compiler treats those building stones in one scan as it goes along, at the same time printing out a copy of the text with possibly some diagnostic messages, producing a translation and collecting information, some of which is displayed afterwards as a diagnostic aid (see section 2.9).

Since all work is done in one forward scan, as a general rule defining occurrences of symbols must precede applied occurrences: if one wants to use, e.g., some symbol as an action, one has to specify it as such before its defining rule. Very limited use is made of block structure: there are only two levels; symbols are either global, or local to some rule; in the latter case they may be redefined in a later rule.

2.2 Symbols.

The input is seen as a sequence of symbols, between which layout characters are ignored. A symbol is of one of four kinds:

1) tag, consisting of a letter, possibly followed by a number of letters or digits, between which spaces are ignored.
2) constant, a sequence of digits, between which spaces are ignored.
3) spec, one of the following reserved special symbols:

plus	+
minus	-
times	*
semicolon	;
comma	,
sub	[
bus	]
open	(
close	)
colon	:
point	.
equals	=

4) bold, a sequence of characters other than accents, enclosed between accents. The following bolds are reserved bold symbols:

external symbol	'external'
action symbol	'action'
predicate symbol	'predicate'
pointer symbol	'pointer'
flag symbol	'flag'
macro symbol	'macro'
list symbol	'list'
restore symbol	'restore'
unrestore symbol	'unrestore'
short symbol	'short'
long symbol	'long'
trace symbol	'trace'
untrace symbol	'untrace'
first parameter symbol	'1'
second parameter symbol	'2'

```
third parameter symbol   '3'
fourth parameter symbol  '4'
fifth parameter symbol   '5'
result symbol            'result'
```

Tags are used as nonterminals, terminals, affixes, labels, in macros, etc. Constants may occur, e.g. as inherited affixes and in macros. Non-reserved bold symbols may occur in macros.

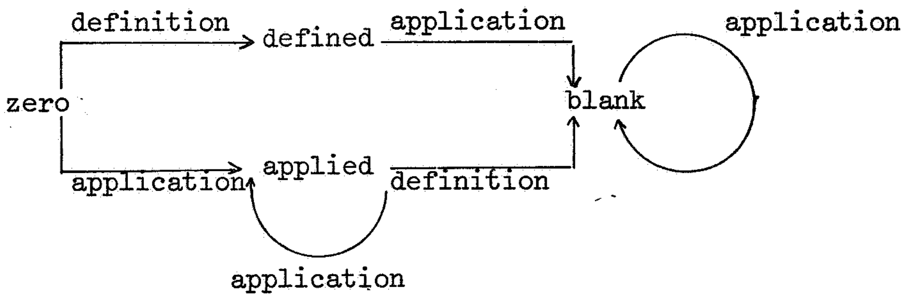
2.3 Specifications.

```
specification:
      external specification;
      internal specification;
      macro specification.
external specification:
      external symbol, type, tag list, point.
type:
      action symbol; predicate symbol;
      pointer symbol; flag symbol.
tag list:
      tag, (comma, tag list;).
internal specification:
      type of internal, tag list, point.
type of internal:
      action symbol; predicate symbol.
macro specification:
      macro symbol, type, rest macro spec, point.
rest macro spec:
      macro def,
            (comma, rest macro spec;).
macro def:
      tag, equals, macrotext.
```

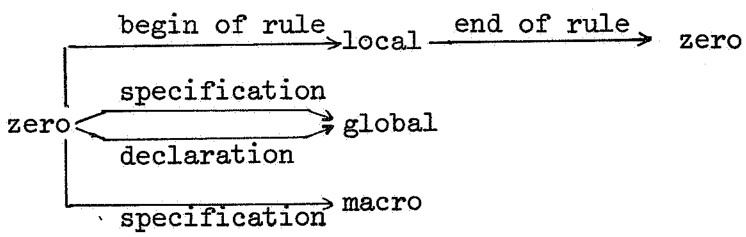
During execution of a compiler compiler with a compiler description as input, every tag occurring in the compiler description has three attributes, its "state", "sort" and "type", which may change according to fixed rules from occurrence to occurrence. Before its first occurrence, the atributes are all assumed to be zero. At every occurrence, the context may prescribe a state, sort and type, and an attempt is made to change the attributes of the tag accordingly. If such a change is not allowed, a diagnostic message is printed. Thus, it is not allowed to redefine a predicate, or use it as an affix, etc. Affixes local to a rule get, at the end of that rule, attributes zero, so that they can be used afterwards with other attributes. The attributes also serve to influence the output of the compiler compiler, so that it translates an action in a different way than a predicate.

Allowed changes of the attributes are

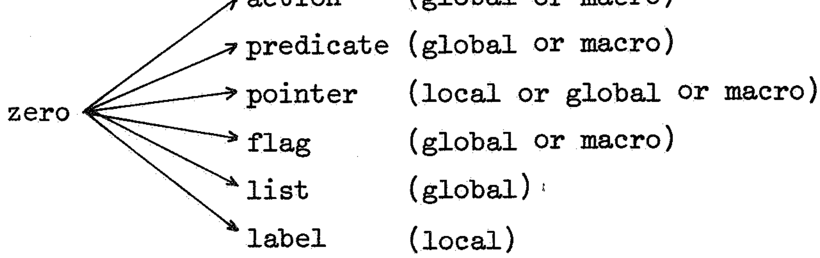
state:



sort:



type:



At the end of each rule, the state of each local should be blank (i.e. both defined and applied; if not then an errormessage is given); all locals again obtain state, sort and type zero.

At the end of the compiler description, the state of each global and macro should be blank, with the exception that an applied but not defined global pointer is treated as a terminal symbol, i.e. a declaration is produced for it, and also a read statement. (See 2.8.)

Specifications serve to prescribe the attributed of a tag, as follows:

		state	sort	type
'external'	'action'	defined	global	action
'external'	'predicate'	defined	global	predicate
'external'	'pointer'	defined	global	pointer
'external'	'flag'	defined	global	flag
'action'		applied	global	action
'predicate'		applied	global	predicate
'macro'	'action'	defined	macro	action
'macro'	'predicate'	defined	macro	predicate
'macro'	'flag'	defined	macro	flag
'macro'	'pointer'	defined	macro	pointer

An external specification indicates that the tags mentioned are supposed to have been declared externally by other means; such a tag may not be redefined and must be applied later on in the compiler description. In this way, e.g., system procedures can be used.

An internal specification indicates that the tags mentioned are to be defined later in in a specific way; it serves as an application: the tags mentioned are henceforth applied globals. Example: 'action' x.

If now a rule follows with x in the left-hand-side, then it will be translated as a <declaration> for an action-procedure. Had the rule not been preceded by this specification, then x would have been treated as a global predicate: an internal specification 'predicate' is always assumed by default.

A macro specification not only establishes the tags mentioned as

defined macros of some type, but also stores their macrotext. A macrotext is any sequence of symbols not containing a comma or a point.
Examples of parameterless macros:

'macro' 'pointer' max list = 5000.

Whenever now in the text max list appears, it will be expanded to *5000*.

If a macrotext must contain a point or comma, then this symbol can be turned into a different (bold) symbol, by enclosing it between accents:

'macro' 'pointer' pi = 3'.'141592654.

On output, the accents will disappear. If one wants to output an accent, then the quote may serve as accent-image, being output as an accent. (There is, unfortunately, no quote image, so a macrotext cannot contain a quote on output). Example:

'macro' 'flag' true = '"true"'.

which, expanded on output, gives *'true'*.

Macros can also be parameterized, with up to 5 parameters. (Another upper limit migh have been chosen.) In the macrotext, the reserved bold symbols '1', '2', '3', '4' and '5' stand for first, second, up to fifth parameter respectively. For a parameter may be substituted any affix, i.e. a local, global or macro, pointer, flag or list. In 2.7 is described how and when a macro is expanded. The macro system described here is admittedly simpleminded and limited; another system might be used instead.

2.4 Declarations.

declaration:
  pointer declaration;
  flag declaration;
  list declaration.
pointer declaration:
  pointer symbol, tag list, point.
flag declaration:
  flag symbol, tag list, point.

list declaration:

list symbol, rest list declaration.

rest list declaration:

tag, sub, expr, comma, expr, bus,

(comma, rest list declaration; point).

expr: tag, rest expr; constant, rest expr.

rest expr: plus, expr; minus, expr; .

A declaration has an effect on the attributes of the tags mentioned, making then defined globals.

Furthermore, a pointer declaration is translated as a <declaration> for a number of integer variables, a flag declaration as one for a number of Boolean variables, and a list declaration as one for a number of integer arrays.

Macros occurring in an expr are expanded. Example:

'macro' 'pointer' min text = 1, max text = 10000.
'list' text [min text : max text].
'pointer' stackpointer.

with translation:

integer array text [1 : 10000];
integer stackpointer;

After these declarations, text is a defined global list, min text is a macro pointer (both defined and applied), and stackpointer a defined global pointer.

2.5 Commands.

command: restore symbol; unrestore symbol;
short symbol; long symbol;
trace symbol; untrace symbol.

The command 'restore' causes the compiler compiler to translate rules in the restoring mode, 'unrestore' in the nonrestoring mode. The default mode is nonrestoring.

The command 'short' causes the compiler compiler to produce compact output by suppressing most layout; 'long' causes more legible output to be produced. The default command is 'long'.

The command 'trace' causes, at the end of each rule, and at the end of the compiler description, tracing output to appear indicating attributes of all locals and globals respectively; the normal situation is 'untrace'.

2.6 Comments.

comment: sub, rest comment.
rest comment: bus; nonbus, rest comment.

(Here, nonbus stands for any symbol except].)

A comment is translated as <comment>, e.g. [input] is translated as *comment input;*

2.7 Rules.

```
rule: left hand side, middle, right hand side, point.
left hand side:
      handle, optional bound affixes.
handle: tag.
optional bound affixes:
      plus, bound affix, optional bound affixes;
      times, bound affix, optional bound affixes; .
bound affix: tag.
middle:
      optional free affixes, colon.
optional free affixes:
      minus, free affix, optional free affixes; .
free affix: tag.
right hand side:
      alternative,
            (semicolon, right-hand-side;); .
```

```
alternative:
      affix expression,
            (comma, alternative;);
      group;
      jump;  .
affix expression:
      label, colon, affix expression;
      handle, optional affixes.
label: tag.
optional affixes:
      plus, affix, optional affixes;  .
affix: tag; constant.
group: open, right hand side, close.
jump: colon, label.
```

A rule is translated as a <declaration> for a parsing procedure corresponding to its handle, where the left-hand-side furnishes the <procedure heading>.

Bound affixes that occur preceded by a plus appear in the <specification part> specified integer, whereas those preceded by a times remain unspecified (so as to allow, e.g., arrays to be passed as parameters).

The middle of a rule is translated as *begin* followed by <declarations> for the free affixes, if any, possibly followed by *integer pold; pold:= pin;*.

The translation of the right-hand-side consists of the translation of its alternatives, followed by

0: handle:= false; end:
end;

where *handle* is an <identifier> corresponding to the handle of the rule. The <label> *0* labels the only way in which the parsing procedure can return false.

The translation of an alternative consists of the translation of its constituents, if any, followed by *handle:= true; goto end;* followed by an unique <label>.

The translation of a jump is the corresponding <go to statement>. A jump furnishes an applied occurrence of a local label.

The translation of a group is

1) In the restoring mode:

begin integer pold; pold:= pin; followed by the translation of its constituent alternatives, followed by *end;*

2) In the nonrestoring mode:

begin followed by the translation of its constituent alternatives, followed by *goto 0 ; end;*

The translation of an affix expression depends on sort and type of its handle, as follows:

Let λ stand for:

i) in the restoring mode: the <label> at the very end of the alternative of which the affix expression is a constituent (termed the current <label>);

ii) in the nonrestoring mode: if this affix expression is the first one of the alternative with a predicate as handle, then the current <label>, and, otherwise, the <label> *0*.

Let α stand for:

i) if the handle is a flag, then the <identifier> corresponding to it;

ii) if the handle is global, then a call of the procedure corresponding to it, with its affixes translated as <actual parameter>s;

iii) if the handle is a macro, then its expansion, with its first, second, etc. affix substituted for the corresponding parameter in its macro text. If the handle is a flag or predicate, then this expansion is furthermore enclosed between brackets (and).

For an affix occurring as a parameter the following translation is given:

i) if the affix is a constant, then the corresponding <unsigned integer>;

ii) if the affix is a tag but not a macro, then the corresponding <identifier>;

iii) if the affix is a macro, then its expansion (without parameter substitution; a parameterized macro is not, for the moment, allowed as an affix).

The expansion of a macro with some number of parameters is its macro text after substitution of parameters and expansion of all parameterless macros occurring in it. (A parameterized macro occurring in the macro text is not, for the moment, expanded.)

The expansion of a parameterless macro is its macro text. (Clearly, recursive macros cannot occur.)

The translation of an affix expression is:

1) If the handle is a flag or predicate, then

$\underline{if}\ \neg\ \alpha\ \underline{then}\ \underline{goto}\ \lambda;$

2) If the handle is an action, then

$\alpha\ ;$

The compiler compiler may delete some elements from this most general translation:

1) it deletes the assignations to the procedure identifier if the handle of the rule is an action, correspondingly changing the Boolean procedure into a procedure;

2) it deletes the <label>s to which no jumps occur, together with those parts of the procedure that can only be reachted via those <label>s.

As an example, consider a part of a compiler description:

```
'action' set to zero.
'macro' 'action' put = '1' ['2']:= '3', incr = '1':= '1' + 1,
      make = '1':= '2'.
'macro' 'predicate' equal = '1' = '2'.
set to zero * list + min + max - p:
      make + p + min,
   rep: put + list + p + 0,
          (equal + p + max;
           incr + p, :rep).
```

Translation:

```
procedure set to zero (list, min, max); integer max, min;
begin integer p;
      p:= min;
 rep: list [p]:= 0;
      begin if ¬(p = max) then goto 2;
            goto end;
         2: p:= p + 1; goto rep;
      end;
end;
```

A tag with type zero, when applied as an affix, becomes by context an applied global pointer. If no declaration or external specification for some applied global pointer occurs in the compiler description, then it will be treated at the end as a terminal (2.8). This is the only provision for terminal symbols: in a compiler description they occur as affixes without defining occurrence.

2.8 Starting symbol.

starting symbol:
      result symbol, handle, point.

The starting symbol is translated as a call for the procedure corresponding to its handle. In front of this call, for each applied but not defined global pointer a <declaration> and a read <statement> is given. Between these <declaration>s and the read <statement>s a preparation procedure *initialize for reading* is called once. If there are no terminals, then also this call is obviated.

Example: assume one terminal, t, and starting symbol sentence:

> *integer t;*
> *initialize for reading;*
> *read (t);*
> *sentence*
>
> *end*

Consequences:

1) in executing the obtained ALGOL 60 program, a representation for each terminal is to be provided as input.
2) If there are terminals, the following actions have to be defined in the compiler description:

   read, out and *initialize for reading*.

3) In restoring parsers, the pointer *pin* must be defined. (Other representations may be taken if the relevant input to the compiler compiler is changed.)

2.9 Diagnostics.

For debugging a compiler description, various diagnostic aids are provided.

In the first place, the whole text is printed out while it is read, with two empty lines between building stones such as rules, declarations, etc. Input is assumed to come from cards, so lines have a maximum width of 80 positions. Output is assumed to proceed on cards, with a maximum of 72 positions used. A $ in column 72 of an output card is used to indicate continuation on the next card. Continuation cards are not counted. The number of the input line appears on the left of its printout, the number of the current output card on its right.

If, at the end of a rule, some local has not been both defined and applied, then a warning message is printed.

If, at the end of the compiler description, some global has not been both defined and applied, then also a warning message appears. Various other errors are signalled by messages which should speak for themselves.

The compiler compiler makes use of a number of tables of fixed size. Whenever the upper bound of one of those tables is exceeded, but also at the end of successful execution, a "post mortem" is printed, giving the lower and upper bound of each table, with the space occupied at that moment, and finally the number of cards read and punched.
These tables, with lower and upper bounds are:

ttag	100001	110000	(tags)
tbold	200001	200300	(bolds)
tspec	300001	300100	(specs)
tcons	400001	400300	(constants)
lloc	500001	500200	(locals)
lglob	600001	601000	(globals)
lmacr	700001	701100	(macrotext)
ltext	800001	804000	(line numbers of occurrences of globals).

The bounds are such that a compiler of the size and complexity of the compiler compiler itself needs only 50% of each of those tables.

Finally, at the end of the execution, a list is printed in alphabetical order of all the globals and macros occurring in the grammar, with their linenumbers of occurrence.

3. Examples.

In this chapter some examples are given and explained of texts in CDL. They are all assumed to occur in an environment containing:

```
'unrestore'
'macro' 'action'
get = '3':= '1' ['2'], put = '1' ['2']:= '3',
make = '1':= '2',
mark = '1':= -'1',
add = '3':= '1' + '2', subtr = '3':= '1' - '2',
addmult = '4':= '1' * '2' + '3',
divrem = '3':= '1' ÷ '2'; '4':= '1' - '2' * '3',
incr = '1':= '1' + 1, decr = '1':= '1' - 1.
'macro' 'predicate'
marked = '1' < 0,
less = '1' < '2', equal = '1' = '2', lseq = '1' ≤ '2'.
'global' 'action' resym, prsym,  exit.
'global' 'pointer' newlinechar, spacechar.
```

This is a rather minimal environment with which a lot can be done. The macro actions get and put are for accessing array elements; make performs the assignation; mark inverts the sign of an <u>integer</u> variable, which we will use as marking bit; add and subtr allow arithmetic; addmult and divrem serve for packing and unpacking small <u>integers</u> into a word; and incr and decr serve to increment and decrement a variable by one.

The system procedure *resym* reads one character from input (without possibility of backtracking) and assigns to its only parameter an integer, viz. the internal value of that character, which does not exceed 255. We will assume that the digits have internal value 0 - 9 respectively, and the letters the internal values 10 - 35. Complementarily, *prsym* prints out the character corresponding to the value of its parameter. The global variables *newlinechar* and *spacechar* contain the internal values of the suggested charecters. A call of *exit* terminates execution.

The following set of examples introduces one by one some building stones for a compiler for an ALGOL like language. Each example may make use of objects defined in a previous example.

3.1 Reading a number.

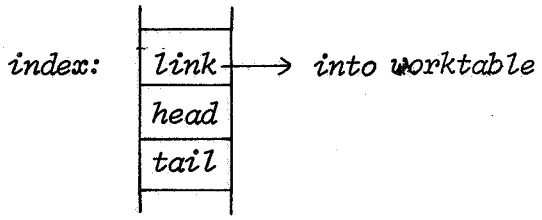
By now we have seen so many versions of number that it is time to see a good one:

```
'action' nextchar.
'pointer' char.
number + val - d:
      digit + d, make + val + d,
         rep: (digit + d, addmult + val + 10 + d + val, :rep;).
digit + d:
      lseq + char + 9, make + d + char, nextchar.
nextchar:
      resym + char,
         rep: (equal + char + spacechar, resym + char, :rep;).
```

Before the first number is read, *initialize for reading* has to take care of calling *nextchar* once.

3.2 Updating a chain.

In a worktable, we want to keep a number of chains of two elements of the form:



Each element points, via its link, to the next of the chain, the last element having link 0. Each chain is accessible by a variable which either points to its first element or contains 0 to indicate an empty chain. We want to be able to

1) search a chain for an element whose head has a specific value, and, if such an element exists, obtain its tail.
2) insert an element at the head of a chain.
3) see whether a specific element is already there, and in that case

obtain its index.

4) detach an element, if any, from the head of a chain, but leave it in the worktable (no garbage-collection).

An updating system with these properties is needed in a compiler, e.g., for collecting declarations: The most recently found declaration for a specific identifier is always the first to be looked at and the first to lose its significance, although the information obtained is needed again afterwards and may therefore not disappear from the worktable.

```
'macro' 'pointer' min work = 1, max work = 4095.
'list' work [min work : max work].
'pointer' pwork.
'action' insert 2, initialize chain adm, errormessage.
'macro' 'action'
get link = '2':= work ['1'], put link = work ['1']:= '2',
get head = '2':= work ['1' + 1], put head = work ['1' + 1]:= '2',
get tail = '2':= work ['1' + 2], put tail = work ['1' + 2]:= '2'.

search 2 + p + hd + tl - q - x:
      make + q + p,
   fnd: (lseq + min work + q, get head + q + x,
            (equal + x + hd, get tail + q + tl;
            get link + q + q, :fnd)).
insert 2 + p + hd + tl - q:
      make + q + p, make + p + pwork, add + pwork + 3 + pwork,
            (less + max work + pwork, errormessage + work full;
            put head + p + hd, put tail + p + tail, put link + p + q).
already there + p + hd + tl + q - x:
      make + q + p,
         fnd: (lseq + min work + q,
            (get head + q + x, equal + x + hd, get tail + q + x,
                                                equal + x + tl;
            get link + q + q, :fnd)).
```

```
detach 2 + p + hd + tl:
      lseq + min work + p, get head + p + hd, get tail + p + tail,
            get link + p + p.
'pointer' access.
initialize chain adm:
      make + pwork + min work, make + access + 0.
```

After one call of *initialize chain adm*, *access* gives access to an empty list. If the worktable becomes too small, *errormessage* is called (see 3.4).

If the compiler is to be run on a computer where the wordlength is such that the head and tail may fit together in one word, then one may simply replace the macro text of put head, get head, put tail and get tail by a version which manipulates with halfwords.

An advantage of using macros for accessing fields of a datastructure is that one can change the datastructures without invalidating the whole grammar. One could, e.g., in testing the compiler perform no packing, and in a later production version introduce packing. This allows the development of quite machine-independent compilers without a resulting price in efficiency.

As an initialization, *initialize chain adm* has to be called.

Note that work full is a terminal symbol (being an affix without defining occurrence) and that a representation of that symbol will have to be at the beginning of the input to the compiler resulting from this compiler description.

3.3 Reading tags.

We want to read <identifier>s, i.e. tags, and obtain a unique key for each different tag. We want to be able to list the tags in alphabetical order. Accompanying each tag we must remember a pointer, originally zero.

We choose the following storage organization: the information is kept in cells linked together in a text table, one cell for each tag, a cell consisting of a fixed and a variable part.

The fixed part consists of 3 pointer:

defn an access to a definition chain;

left a pointer to the chain of all tags alphabetically preceding the tag, zero if empty;

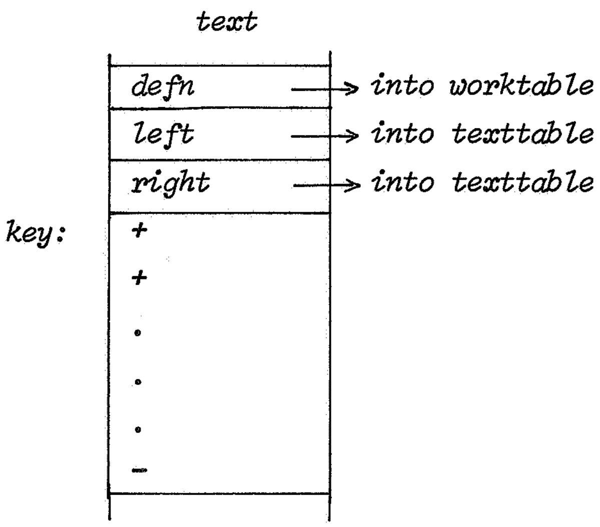
right idem for the alphabetically following tags.

The variable part contains the successive characters of the tag, packed three to a word, the last word being filled out with dummies (= 255) and provided with a - sign, the others with a + sign. (We will assume that the wordlength of the computer on which the compiler will run is sufficient, i.e. at least 25 bits.)

We will give out the index (in the text table) of the first word of the variable part as a key.

The chainstructure just described is a doubly linked namelist. It is more efficient to use than a linear namelist, and less efficient than a namelist using hash coding, over which again it has the advantage of keeping the tags in alphabetical order.

In a picture:



When reading a tag, we will first build up a cell for it in the text table, and then enter it, checking whether the same text already occurs in the table; if so, then the key of the old occurrence is delivered and the new cell removed; if not, then the new cell is fitted into the chain.

```
'macro' 'pointer' mintext = 1, max text = 4095.
'list text [min text : max text].
'pointer' ptext.
'macro' 'action'
put defn = text ['1' - 3]:= '2', get defn = '2':= text ['1' - 3],
put left = text ['1' - 2]:= '2', get left = '2':= text ['1' - 2],
put right= text ['1' - 1]:= '2', get right= '2':= text ['1' - 1].
'macro' 'predicate'
      is letter = '1' > 9 ∧ '1' < 36,
      is letgit = '1' < 36.

[treatment of text]
'action' add to text, stack text, stack last, enter, reserve fixed part.
'pointer' word, c.
'macro' 'pointer' dummy = 255, bytesize = 256, maxbytes = 3.
'macro' 'predicate' left = abs('1') < abs('2').
add to text + x:
      equal + c + maxbytes, stack text + word, make + word + x,
                                                   make + c + 1;
      addmult + word + bytesize + x + word, incr + c.
stack text + x:
      lseq + ptext + max text, put + text + ptext + x, incr + ptext;
      errormessage + text full.
stack last:
   rep: (less + c + maxbytes, addmult + word + bytesize + dummy + word,
            incr + c, :rep;
      mark + word, stack text + word).
```

Now follows the *pièce de résistance* of the treatment of text. The action enter checks whether the topmost element of the texttable, which is pointed to by x, already occurred somewhere else in the texttable; if such is the case, then x is made to point to that older element, and the new element is deleted. Also, this action takes care of the alphabetic ordering of the tags.

```
enter + x + pchain - y - x1 - y1 -wx - wy:
      equal + pchain + 0, make + pchain + x, reserve fixed part;
      make + y + pchain,
   nxy: make + x1 + x, make + y1 + y,
   nxw: get + text + x1 + wx, get + text + y1 + wy,
            (equal + wx + wy,
                  (marked + wx,
                        (equal + x + y, reserve fixed part;
                        make + ptext + x, make + x + y);
                  incr + x1, incr + y1, :nxw);
            left + wx + wy, get left + y + wy,
                  (equal + wy + 0, put left + y + x, reserve fixed part;
                  make + y + wy, :nxy);
            get right + y + wy,
                  (equal + wy + 0, put right + y + x, reserve fixed part;
                  make + y + wy, :nxy)).
reserve fixed part:
      stack text + 0, stack text + 0, stack text + 0.

[reading a tag]
'pointer' ptag.
read tag + x - t:
      letter + t, make + word + t, make + c + 1, make + x + ptext,
         nxt: (letgit + t, add to text + t, :nxt;
            stack last, enter + x + ptag).
letter + t:
      is letter + char, make + t + char, nextchar.
letgit + t:
      is letgit + char, make + t + char, nextchar.
initialize text adm:
      make + ptext + min text, reserve fixed part, make + ptag + 0.
```

Of course the left- and right-pointer of a tag can be packed together again in one word. As initialization, *initialize text adm* has to be called. The pointer ptag gives access to the chain of tags, text full is a terminal.

3.4 Printing, errormessages.

We are now in a position to define the printing of tags, and the treatment of errormessages.

```
'action' print, print 1, print 2, nlcr, errormessage.
print + x - p - el:
      make + p + x,
         rep: get + text + p + el,
            (marked + el, mark + el, print 1 + el;
            print 1 + el, incr + p, :rep).
print 1 + t - c:
      prsym + spacechar,
      make + c + max bytes, print 2 + c + t.
print 2 + c + t - el:
      equal + c + 1, prsym + el;
      divrem + t + bytesize + t + el, decr + c, print 2 + c + t,
                                                      prsym + el.
nlcr: prsym + newlinechar.
errormessage + x: nlcr, print + x, exit.
```

The definition of print 2 has to be recursive in order to get the characters out in the right order.

The procedure *errormessage* prints a message and then terminates execution. A more subtle reaction might be envisaged.

3.5 An input administration.

Assuming that there also exist texts, similar to that in 3.3, for reading bold symbols, special symbols and constants, a complete input administration can be constructed:

```
'macro' 'pointer' min inpt = 1, max inpt = 2047.
'list' inpt [min inpt : max inpt].
'pointer' pin, xin, symb.
'action' next symbol, skip layout.
next symbol:
   nxt: skip layout,
      (read tag + symb;
      read bold + symb, add + symb + 20000 + symb;
      read spec + symb, add + symb + 30000 + symb;
      read const+ symb, add + symb + 40000 + symb;
      errormessage + incorrect char, nextchar, :nxt).
skip layout:
   skp: (equal + char + new line char, nextchar, :skp;).
rq + x:
      equal + symb + x, incr + pin,
            (equal + pin + xin, next symbol, put + inpt + pin + symb,
                                                          incr + xin;
            get + inpt + pin + symb).
identifier + x:
      less + symb + 20000, make + x + symb, rq + x.
initialise for reading:
      nextchar, initialize chain adm, initialize text adm,
      make + xin + min inpt, make + pin + xin, nextsymbol, put + inpt +
                                                          pin + symb.
```

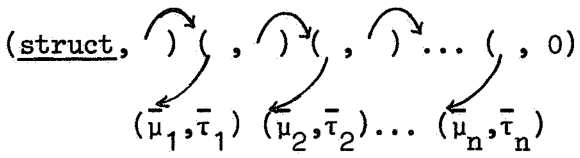
The pointer *pin* serves as reading pointer, *xin* indicates up to what index the inputarray has been filled with symbol-keys. In *next symbol*, a multiple of 10000 is added to some keys as a type-indication, allowing discrimination between, e.g., tags and bolds. By means of *rq*, one can ask whether the current symbol (kept in *symb*), is equal to a given symbol, while *identifier* returns <u>true</u> only if the current symbol is a tag. A restoring parser can now work on the basis of this input administration.

3.6 Reading declarers.

We will now show how to recognize declarers not unlike those of ALGOL 68. During recognition of a declarer, some (head, tail) pairs may be added to the worktable (3.2), and the mode of the declarer is represented by the index of one such pair.

Let μ stand for a declarer, $\bar{\mu}$ for its index, τ for a tag, $\bar{\tau}$ for its key.

int gives an index to (0, int)
real gives (0, real)
long μ gives (long, $\bar{\mu}$)
ref μ gives (ref, $\bar{\mu}$) and
struct $(\mu_1\tau_1, \mu_2\tau_2, \ldots, \mu_n\tau_n)$ gives



In adding a pair to the worktable, we take care that no two copies of a pair are ever inserted: in adding a pair that was already there, the index of the old copy is obtained. This implies that equivalent modes automatically get equal indices.

```
'action' add to decl.
'pointer' pdecl.
declarer + mode:
      primitive declarator + mode;
      long   declarator + mode;
      ref    declarator + mode;
      struct declarator + mode.
'restore'
primitive declarator + mode:
      rq + int,  add to decl + 0 + int  + mode;
      rq + real, add to decl + 0 + real + mode.
```

long declarator + mode:

rq + long, declarer + mode, add to decl + long + mode + mode.

ref declarator + mode:

rq + ref, declarer + mode, add to decl + ref + mode + mode.

struct declarator + mode:

rq + struct, rq + open, fields + mode, add to decl + struct + mode + mode.

fields + mode - m1 - m2:

rq + close, make + mode + 0;

field + m1, fields + m2, add to decl + m1 + m2 + mode.

field + mode - tag:

declarer + mode, identifier + tag, add to decl + mode + tag + mode.

'unrestore'

add to decl + hd + tl + mode - oldp:

already there + hd + tl + pdecl + oldp, make + mode + oldp;

make + oldp + pwork, insert 2 + hd + tl + pdecl, make + mode + oldp.

Note that for the declarators we want restoring parsers. Also note that affixes that have no defining occurrence, e.g., int, real, long, etc., are terminals, so that a representation for them has to be given as input to the resulting compiler.

For already there and insert 2, see 3.2.

3.7 Collecting defining occurrences.

We will now show how to collect defining occurrences of identifiers. We assume the existence of a global pointer *blocknumber* that is automatically updated by some other part of the compiler description.

A defining occurrence of an identifier is an occurrence in a declaration. For each defining occurrence we want to store its blocknumber and mode in the worktable. We will not allow an identifier to be defined twice in one block.

```
'global' 'pointer' blocknumber.
'action' define identifier
declaration - mode - idf:
      declarer + mode, identifier + idf,
         rep: (define identifier + idf + mode,
                 (rq + comma,
                       (identifier + idf, :rep;
                       declarer + mode, identifier + idf, :rep;
                       errormessage + incorrect declaration);
                 rq + semicolon)).
define identifier + idf + mode - def - dummy:
      get defn + idf + def,
            (search 2 + def + blocknumber + dummy, errormessage +
                                                   defined twice;
            insert  2 + def + blocknumber + mode, put defn + idf + def).
```

A more elaborate version of this last example can be found in [2].

```
[4     COMPILER COMPILER DESCRIBED IN CDL ************************************] 2
                                                                              2
'SHORT'                                                                       2
                                                                              2
[4.1   GENERAL ENVIRONMENT]                                                   3
                                                                              3
[4.1.1   INTERFACE WITH MACHINE]                                              4
'EXTERNAL''ACTION'                                                            4
NEW PAGE,EXIT,PRSYM,CSYM.

                                                                              4
'EXTERNAL''POINTER'RESYM.

                                                                              4
'MACRO''FLAG'                                                                 4
WAS LETTER='1'>9^'1'<36,                                                      4
WAS LETGIT='1'<36,                                                            4
WAS DIGIT ='1'<10,                                                            4
WAS SPECCH='1'>63.

                                                                              4
'MACRO''POINTER'                                                              4
NIX=63,MINUSCODE=65,SPACECODE=93,TABCODE=118,NLCRCODE=119,ACCC=120,QUOTE=121.

                                                                              4
                                                                              4
[4.1.2   STACKS]                                                              5
'MACRO' 'POINTER'                                                             5
MIN TAG =100001,MAX TAG =110000,                                              5
MIN BOLD=200001,MAX BOLD=200600,                                              5
MIN SPEC=300001,MAX SPEC=300100,                                              5
MIN CONS=400001,MAX CONS=400300,                                              5
MIN LOC =500001,MAX LOC =500200,                                              5
MIN GLOB=600001,MAX GLOB=601000,                                              5
MIN MACR=700001,MAX MACR=701100,                                              5
MIN TEXT=800001,MAX TEXT=804000.

                                                                              5
'LIST'                                                                        5
TTAG [MIN TAG :MAX TAG ],                                                     6
TBOLD[MIN BOLD:MAX BOLD],                                                     6
TSPEC[MIN SPEC:MAX SPEC],                                                     6
TCONS[MIN CONS:MAX CONS],                                                     6
LLOC [MIN LOC :MAX LOC ],                                                     6
LGLOB[MIN GLOB:MAX GLOB],                                                     6
LTEXT[MIN TEXT:MAX TEXT],                                                     6
LMACR[MIN MACR:MAX MACR].

                                                                              6
'MACRO''FLAG'                                                                 6
WAS TAG =MIN TAG '"LE"''1'^'1'<PTAG ,                                         6
WAS BOLD=MIN BOLD'"LE"''1'^'1'<PBOLD,                                         6
WAS SPEC=MIN SPEC'"LE"''1'^'1'<PSPEC,                                         6
WAS CONS=MIN CONS'"LE"''1'^'1'<PCONS,

                                                                              6
```

```
                                                                        6
[4.1.3   MACROS]                                                        7
'MACRO' 'ACTION'                                                        7
GET   ='3':='1'['2'],                                                   7
PUT   ='1'['2']:='3',                                                   7
MAKE='1':='2',                                                          7
SET ='1':='2',                                                          7
MARK='1':=-'1',                                                         7
ADD='3':='1'+'2',                                                       7
SUBTR='3':='1'-'2',                                                     7
ADDMULT='4':='1'*'2'+'3',                                               7
DIVREM='3':='1'"/"'2';'4':='1'-'2'*'3',                                 7
PACK3='4':=('1'*128+'2')*128+'3',                                       7
UNPACK3='2':='1'"/"16384;'4':='1'-16384*'2';'3':='4'"/"128;'4':='4'-128*'3', 7
PACK2='3':=8192*'1'+'2',                                                7
UNPACK2='2':='1'"/"8192;'3':='1'-8192*'2',                              7
GET TAIL='2':='1'-'1'"/"8192*8192,                                      7
INCR  ='1':='1'+1,                                                      7
DECR  ='1':='1'-1.

                                                                        7
'MACRO''FLAG'                                                           7
MARKED='1'<0,                                                           7
LEFT=ABS('1')<ABS('2'),                                                 7
LESS='1'<'2',                                                           7
LSEQ='1'≤'2',                                                           7
EQUAL='1'='2',

                                                                        7
'MACRO' 'POINTER'                                                       7
FALSE='"FALSE"',                                                        7
TRUE='"TRUE"',

                                                                        7
[DATA STRUCTURES AND THEIR ACCESS]                                      8
'MACRO''ACTION'                                                         8
GET NPARS='2':=TTAG['1'-8],PUT NPARS=TTAG['1'-8]:='2',                  8
GET MTEXT='2':=TTAG['1'-7],PUT MTEXT=TTAG['1'-7]:='2',                  8
GET PLACE='2':=TTAG['1'-6],PUT PLACE=TTAG['1'-6]:='2',                  8
GET STATE='2':=TTAG['1'-5],PUT STATE=TTAG['1'-5]:='2',                  8
GET SORT ='2':=TTAG['1'-4],PUT SORT =TTAG['1'-4]:='2',                  8
GET TYPE ='2':=TTAG['1'-3],PUT TYPE =TTAG['1'-3]:='2',                  8
GET LEFT ='2':=TTAG['1'-2],PUT LEFT =TTAG['1'-2]:='2',                  8
GET RIGHT='2':=TTAG['1'-1],PUT RIGHT=TTAG['1'-1]:='2',

                                                                        8
                                                                        8
[4.1.4   POINTERS AND FLAGS]                                            9
'POINTER'                                                               9
PTAG,PBOLD,PSPEC,PCONS,                                                 10
PLOC,PGLOB,PTEXT,PMACR,XLOC.

                                                                        10
'FLAG' GIVE TEXT,GIVE TRACE,LEGIBLE.

                                                                        11
'POINTER' FIRST TAG.

                                                                        12
                                                                        12
```

```
                                                                                  12
                                                                                  12
                                                                                  12
                                                                                  12
[4.2      OUTPUT]                                                                 13
                                                                                  13
[4.2.1    PRINTER SECTION]                                                        14
'ACTION'OUT,OUTINT,PRINT,PRINT1,NLCR,TAB,SPACE,OUTINT1,PRCHAR,SPACES,TABS,        14
POSITION,SHIFT2LINES.
                                                                                  14
'POINTER'POS.
                                                                                  15
OUT+X-EL:                                                                         16
        WAS TAG +X,PRINT+TTAG +X;                                                 16
        WAS BOLD+X,PRINT+TBOLD+X;                                                 16
        WAS SPEC+X,PRINT+TSPEC+X;                                                 16
        WAS CONS + X, GET+TCONS+X+EL,OUTINT+EL;                                   16
        OUTINT+X.
                                                                                  16
OUTINT+X-QUOT-REM:                                                                17
        SPACE,                                                                    17
        LESS+X+0,MAKE+REM+X,MARK+REM,PRCHAR+MINUSCODE,OUTINT+REM;                 17
        EQUAL+X+0,PRCHAR+0,SPACE;                                                 17
        DIVREM+X+10+QUOT+REM,OUTINT1+QUOT,PRCHAR+REM,SPACE.
                                                                                  17
OUTINT1+X-QUOT-REM:                                                               18
        EQUAL+X+0;                                                                18
        DIVREM+X+10+QUOT+REM,OUTINT1+QUOT,PRCHAR+REM.
                                                                                  18
PRINT+LIST+Y-X-LX:                                                                19
        MAKE+X+Y,SPACE,                                                           19
   RST: GET+LIST+X+LX,                                                            19
                (MARKED+LX,MARK+LX,PRINT1+LX;PRINT1+LX,INCR+X,:RST).
                                                                                  19
PRINT1+X-X1-X2-X3:                                                                20
        UNPACK3+X+X1+X2+X3,                                                       20
        PRCHAR+X1,PRCHAR+X2,PRCHAR+X3.
                                                                                  20
POSITION+X-TBS-SPCES:                                                             21
      SUBTR+X+POS+SPCES,LSEQ+0+SPCES,SPACES+SPCES;                                21
        NLCR,DIVREM+X+8+TBS+SPCES,TABS+TBS,                                       21
                (EQUAL+SPCES+0;SPACES+SPCES).
                                                                                  21
SPACES+X-N:                                                                       22
        MAKE+N+0,                                                                 22
   SPC: LESS+N+X,SPACE,INCR+N,:SPC;
                                                                                  22
TABS+X-N:                                                                         23
        MAKE+N+0,                                                                 23
   TBS: LESS+N+X,TAB,INCR+N,:TBS;
```

```
                                                                          23
NLCR:                                                                     24
          PRSYM +NLCRCODE,MAKE+POS+0.

                                                                          24
SHIFT 2 LINES-OLD POS:                                                    25
          MAKE+OLDPOS+POS,NLCR,NLCR,SPACES+OLD POS.

                                                                          25
TAB:PRSYM+TABCODE,ADD+POS+8+POS.

                                                                          26
SPACE:                                                                    27
          PRCHAR+SPACECODE.

                                                                          27
PRCHAR+X:                                                                 28
          EQUAL+X+NLCRCODE,NLCR;                                          28
          EQUAL+X+TABCODE,TAB;                                            28
          EQUAL+X+NIX;                                                    28
          PRSYM+X,INCR+POS.

                                                                          28
                                                                          28
[4.2.2   TRACE ADMINISTRATION]                                            29
'ACTION'TRACE,SIGNAL,INFORM,ERROR.

                                                                          29
TRACE+X:                                                                  30
          GIVE TRACE,POSITION+32,INFORM+X;  .

                                                                          30
SIGNAL+X:                                                                 31
          POSITION+16,INFORM+X.

                                                                          31
INFORM+X-STATE-SORT-TYPE:                                                 32
          GET STATE+X+STATE,GET SORT+X+SORT,GET TYPE+X+TYPE,              32
          OUT+STATE,OUT+SORT,OUT+TYPE,OUT+X.

                                                                          32
ERROR+TEXT+INFO-OLD POS:                                                  33
          MAKE+OLD POS+POS,NLCR,OUT+TEXT,                                 33
                    (WAS TAG+INFO,INFORM+INFO,POSITION+OLD POS;           33
                     EQUAL+INFO+0,POSITION+OLD POS;                       33
                     OUT+INFO,POSITION+OLD POS).

                                                                          33
                                                                          33
[4.2.3   CARD PUNCH SECTION]                                              34
'ACTION'WRITE,WRITEINT,WRITEINT1,PUNCH,PUNCH1,NEW CARD,TAB CARD,SPACE CARD,   34
PUCHAR.

                                                                          34
'MACRO''POINTER'                                                          34
MAX CARD=72,DOLLAR CODE=133.

                                                                          34
'POINTER' CARD,CPOS.
```

```
WRITE +X - EL:
          WAS TAG  + X, PUNCH + TTAG  + X;
          WAS BOLD + X, PUNCH + TBOLD + X;
          WAS SPEC + X, PUNCH + TSPEC + X;
          WAS CONS + X,GET+TCONS+X+EL,WRITE INT+EL;
          WRITE INT+X.

WRITE INT+X-QUOT-REM:
          LESS+X+0,MAKE+REM+X,MARK+REM,PUCHAR+MINUSCODE,WRITE INT+REM;
          EQUAL+X+0,PUCHAR+0:
          DIVREM+X+10+QUOT+REM,WRITE INT1+QUOT,PUCHAR+REM.

WRITE INT1+X-QUOT-REM:
          EQUAL+X+0;
          DIVREM+X+10+QUOT+REM,WRITE INT1+QUOT,PUCHAR+REM.

PUNCH+LIST+Y-X-LX:
          MAKE+X+Y,
     RST: GET+LIST+X+LX,
                    (MARKED+LX,MARK+LX,PUNCH1+LX;
                     PUNCH1+LX,INCR+X,:RST),

PUNCH1+X-X1-X2-X3:
          UNPACK3+X+X1+X2+X3,
          PUCHAR+X1,PUCHAR+X2,PUCHAR+X3.

NEW CARD:
     SPC: LSEQ +CPOS + MAX CARD,CSYM+SPACECODE,INCR+CPOS,:SPC;
          MAKE+CPOS+1,WRITE INT+CARD,CSYM+NLCR CODE,INCR+CARD,MAKE+CPOS+1.

TAB CARD-SPCES-TBS:
          DIVREM+CPOS+8+TBS+SPCES,
     SPC: LESS+SPCES +8,SPACE CARD,INCR+SPCES ,:SPC;

SPACE CARD:
         PUCHAR+SPACECODE.

PUCHAR+X:
          EQUAL + X + NLCRCODE, NEW CARD;
          EQUAL + X + TABCODE, TAB CARD;
          EQUAL + X + NIX;
          LESS+CPOS+MAX CARD.CSYM+X,INCR+CPOS;
          CSYM+DOLLARCODE,CSYM+NLCRCODE,MAKE+CPOS+1,PUCHAR+X.


[4.2.4  RESULT ADMINISTRATION]
'ACTION'G,LINE UP,PUTABS,U,L,NEW LINE,BLANK LINE,TAB LINE,

'FLAG'LINED UP.
```

```
                                                          46
'POINTER'INDENTATION.

                                                          47
G + X:                                                    48
        LINE UP, WRITE + X.

                                                          48
LINE UP:                                                  49
        LINED UP;                                         49
        LEGIBLE,PUTABS+INDENTATION,SET+LINED UP+TRUE.

                                                          49
PUTABS + N - N1:                                          50
        LSEQ + N + 0;                                     50
        MAKE + N1 + N, DECR + N1, TAB CARD, PUTABS+ N1.

                                                          50
U: INCR + INDENTATION.

                                                          51
L: DECR + INDENTATION.

                                                          52
NEW LINE:                                                 53
        LEGIBLE,LINED UP,BLANK LINE.

                                                          53
BLANK LINE:                                               54
        NEW CARD, SET + LINED UP + FALSE.

                                                          54
TAB LINE: LEGIBLE,TAB CARD.

                                                          55
                                                          55
                                                          55
                                                          55
                                                          55
                                                          55
                                                          55
[4.3    INPUT]                                            56
'ACTION' POST MORTEM.

                                                          56
'POINTER'                                                 56
LINE,CHAR,INPT.

                                                          57
                                                          57
[4.3.1   READING CHARACTERS]                              58
'ACTION'NEXTCHAR,DISPLAY CHARACTER,NEXT NON SPACE CHAR.

                                                          58
LETTER+X:                                                 59
        WAS LETTER+CHAR,MAKE+X+CHAR,NEXT NON SPACE CHAR.

                                                          59
LETGIT+X:                                                 60
```

```
                WAS LETGIT+CHAR,MAKE+X+CHAR,NEXT NON SPACE CHAR.

                                                                        60
DIGIT+X:                                                                61
                WAS DIGIT +CHAR,MAKE+X+CHAR,NEXT NON SPACE CHAR.

                                                                        61
BOLDCHAR+X:                                                             62
                EQUAL+CHAR+QUOTE,MAKE+X+ACCC,NEXTCHAR;                  62
                MAKE+X+CHAR,NEXTCHAR.

                                                                        62
ACCENT:EQUAL+CHAR+ACCC,NEXTCHAR.

                                                                        63
SPECCHAR + X:                                                           64
                WAS SPECCH+CHAR,MAKE+X+CHAR,NEXT NON SPACE CHAR.

                                                                        64
NEXTCHAR:                                                               65
                DISPLAY CHARACTER,MAKE + CHAR + RESYM.

                                                                        65
DISPLAY CHARACTER:                                                      66
                EQUAL+CHAR+NLCRCODE,INCR+LINE,                          66
                        (GIVE TEXT,POSITION+133,OUTINT1+CARD,           66
                                POSITION+48,OUTINT1+LINE,POSITION+52;  );  66
                GIVE TEXT,PRCHAR+CHAR; .

                                                                        66
NEXT NON SPACE CHAR:                                                    67
   CHR: NEXTCHAR,                                                       67
                (EQUAL+CHAR+SPACECODE,:CHR;   ).

                                                                        67
LAYOUT SYMBOL:                                                          68
                EQUAL + CHAR + SPACECODE;                               68
                EQUAL + CHAR + NLCRCODE;                                68
                EQUAL + CHAR + TABCODE .

                                                                        68
                                                                        68
[4.3.2   TAG SYMBOLS]                                                   69
'ACTION'STACK TAG,ENTER TAG,RESERVE ADMIN SPACE.

                                                                        69
                READ TAG + X - T - T1 - T2 - T3 - T4:                   70
                        LETTER+T1,MAKE+X+PTAG,                          70
                           NXT: (LETGIT + T2,                           70
                                        (LETGIT + T3,                   70
                                                (LETGIT + T4, PACK3 + T1 + T2 + T3 + T,   70
                                                 STACK TAG+T,MAKE+T1+T4,:NXT;             70
                                                 PACK3 + T1 + T2 + T3 + T, :LST);         70
                                         PACK3 + T1 + T2 + NIX + T, :LST);                70
                                 PACK3 + T1 + NIX + NIX + T,                              70
                           LST: MARK + T, STACK TAG + T).

                                                                        70
                STACK TAG + X:                                          71
                        LSEQ + PTAG + MAX TAG, PUT + TTAG + PTAG + X, INCR + PTAG;        71
```

```
                 ERROR + TAG FULL+0,POST MORTEM,EXIT.
                                                                              71
        ENTER TAG + X - Y - X1 - Y1 - WX - WY:                                72
                 MAKE + Y + FIRST TAG,                                        72
           NXY: MAKE + X1 + X, MAKE + Y1 + Y,                                 72
           NXW: GET+TTAG+X1+WX,GET+TTAG+Y1+WY,                                72
                         (EQUAL + WX + WY,                                    72
                                 (MARKED + WX,                                72
                                         (EQUAL + Y + X,RESERVE ADMIN SPACE;  72
                                          MAKE+PTAG+X,MAKE+X+Y);              72
                                  INCR+X1,INCR+Y1,:NXW);                      72
                          LEFT+WX+WY,GET LEFT+Y+WY,                           72
                                 (EQUAL+WY+0,PUT LEFT+Y+X,RESERVE ADMIN SPACE;  72
                                  MAKE+Y+WY,:NXY));                           72
                          GET RIGHT+Y+WY,                                     72
                                 (EQUAL+WY+0,PUT RIGHT+Y+X,RESERVE ADMIN SPACE;  72
                                  MAKE+Y+WY,:NXY)).
                                                                              72
RESERVE ADMIN SPACE:                                                          73
        ADD+PTAG+8+PTAG,LESS+PTAG+MAX TAG,PUT NPARS+PTAG+999,                 73
        PUT MTEXT+PTAG+0,PUT PLACE+PTAG+0,PUT STATE+PTAG+0,PUT SORT+PTAG+0,   73
        PUT TYPE+PTAG+0,PUT LEFT+PTAG+0,PUT RIGHT+PTAG+0;                     73
        ERROR+TAG FULL+0,POST MORTEM,EXIT.
                                                                              73
                                                                              73
[4.3.3   BOLD SYMBOLS]                                                        74
'ACTION'STACK BOLD,ENTER BOLD.
                                                                              74
READ BOLD + X - B1-B2-B3-D:                                                   75
        ACCENT, MAKE+X+PBOLD,                                                 75
        (ACCENT,PACK3+NIX+NIX+NIX+D,MARK+D,STACK BOLD+D;                      75
         BOLDCHAR+B1,                                                         75
           RST: (ACCENT,PACK3+B1+NIX+NIX+D,MARK+D,STACK BOLD+D;               75
                 BOLDCHAR+B2,                                                 75
                         (ACCENT,PACK3+B1+B2+NIX+D,MARK+D,STACK BOLD+D;       75
                          BOLDCHAR+B3,PACK3+B1+B2+B3+D,                       75
                                 (ACCENT,MARK+D,STACK BOLD+D;                 75
                                  BOLDCHAR+B1,STACK BOLD+D,:RST)))).
                                                                              75
        STACK BOLD + X:                                                       76
                 LSEQ + PBOLD + MAX BOLD, PUT + TBOLD + PBOLD + X, INCR+PBOLD;  76
                 ERROR + BOLD FULL+0,POST MORTEM,EXIT.
                                                                              76
        ENTER BOLD + X - Y - X1 - Y1 - WX - WY:                               77
                 MAKE+Y+MIN BOLD,                                             77
           NXY: MAKE + X1 + X, MAKE + Y1 + Y,                                 77
           NXW: GET+TBOLD+X1+WX,GET+TBOLD+Y1+WY,                              77
                         (EQUAL + WX + WY,                                    77
                                 (MARKED + WX,                                77
                                         (EQUAL+Y+X;                          77
                                          MAKE+PBOLD+X,MAKE+X+Y);             77
                                  INCR+X1,INCR+Y1,:NXW);                      77
                    SKP:  MARKED + WY, MAKE + Y + Y1, INCR  + Y, :NXY;        77
                          INCR  + Y1, GET + TBOLD + Y1 + WY, :SKP).
```

```
[4.3.4   SPECIAL SYMBOLS]
'ACTION'STACK SPEC,ENTER SPEC.

         READ SPEC + X - S - S1:
                   SPECCHAR+S1,MAKE+X+PSPEC,PACK3+S1+NIX+NIX+S,MARK+S,STACK SPEC+S.

         STACK SPEC + X:
                   LSEQ + PSPEC + MAX SPEC, PUT + TSPEC + PSPEC + X, INCR+PSPEC;
                   ERROR + SPEC FULL+0,POST MORTEM,EXIT.

         ENTER SPEC + X - Y - WX - WY:
                   MAKE+Y+MIN SPEC,GET+TSPEC+X+WX,
              NXY: GET+TSPEC+Y+WY,
                             (EQUAL + WX + WY,
                                       (EQUAL+Y+X;
                                        MAKE+PSPEC+X,MAKE+X+Y);
                              INCR  + Y , :NXY).


[4.3.5   CONSTANTS]
'ACTION'STACK CONS,ENTER CONS.

READ CONS + X - D - S:
         DIGIT + D, MAKE + S + D,
            RST: (DIGIT + D, ADDMULT + S + 10 + D + S,:RST;
                    MAKE + X + PCONS, STACK CONS + S).

STACK CONS + X:
         LSEQ + PCONS + MAX CONS,PUT + TCONS + PCONS+ X, INCR + PCONS;
         ERROR + CONS FULL+0,POST MORTEM,EXIT.

ENTER CONS + X - Y - S - T:
         GET + TCONS + X + S, MAKE + Y + MIN CONS,
    NXT: GET + TCONS + Y + T,
                   (EQUAL + S + T,
                             (EQUAL + X + Y;
                              MAKE + PCONS + X, MAKE + X + Y);
                    INCR + Y, :NXT).


[4.3.6   INPUT ADMINISTRATION]
'ACTION'INITIALIZE FOR READING,NEXT SYMBOL,READ.

INITIALIZE FOR READING:
         MAKE+GIVE TEXT+FALSE,MAKE+PCONS+MIN CONS,
         MAKE+PTAG+MIN TAG,MAKE+PBOLD+MIN BOLD,MAKE +PSPEC+MIN SPEC,MAKE+LINE+0,
         MAKE+CHAR+RESYM,RESERVE ADMIN SPACE,MAKE+FIRST TAG+PTAG,
```

```
                                                                              87
IS TAG  + X: WAS TAG + INPT,  MAKE + X + INPT, NEXT SYMBOL.
                                                                              88
IS BOLD+X:WAS BOLD+INPT,MAKE+X+INPT,NEXT SYMBOL.
                                                                              89
IS CONS + X: WAS CONS + INPT, MAKE + X + INPT, NEXT SYMBOL.
                                                                              90
R + X: AHEAD + X, NEXT SYMBOL.
                                                                              91
AHEAD + X: EQUAL + INPT + X.
                                                                              92
NEXT SYMBOL:                                                                  93
        READ+INPT.
                                                                              93
READ+X:                                                                       94
SKP: LAYOUT SYMBOL,NEXTCHAR,:SKP;                                             94
        READ TAG +X,ENTER TAG+X;                                              94
        READ BOLD+X,ENTER BOLD+X;                                             94
        READ SPEC+X,ENTER SPEC+X;                                             94
        READ CONS+X,ENTER CONS+X;                                             94
        ERROR+WRONG INPUT CODE+CHAR,NEXTCHAR,READ+X.
                                                                              94
                                                                              94
                                                                              94
                                                                              94
                                                                              94
                                                                              94
[4.4    COMPILER COMPILER TO ALGOL60]                                         95
'FLAG' GIVE RSTO,ADVANCED.
                                                                              96
'POINTER' HANDLE,BOUNDX,SPECX,TO NEXT,TO LOST,TO END,CUR LAB,NEW LAB.
                                                                              97
                                                                              97
[4.4.1   AUXILIARY ACTIONS]                                                   98
'ACTION'                                                                      98
FORGET LOCALS,WARN,ENTER LOCAL,ENTER GLOBAL,ENTER MACRO,ENTER TEXT,           98
ADD PLACE,REPLACE.
                                                                              98
ADD PLACE+TAG-H-K-Q:                                                          99
        GET PLACE+TAG+H,                                                      99
                (EQUAL + H + 0, REPLACE + TAG + H;                            99
                 GET+LTEXT+H+Q,GET TAIL+Q+K,LESS+K+LINE,REPLACE+TAG+H;  ).
                                                                              99
REPLACE + TAG + H:                                                            100
        PUT PLACE+TAG+PTEXT,                                                  100
                (EQUAL+H+0,ENTER TEXT+LINE;                                   100
                 SUBTR+H+MIN TEXT+H,INCR+H,PACK2+H+LINE+H,ENTER TEXT+H).
                                                                              100
```

```
FORGET LOCALS-LOC:                                                                    101
   RST: EQUAL+PLOC+MIN LOC:                                                           101
        DECR+PLOC,GET+LLOC+PLOC+LOC,WARN+LOC,                                         101
                PUT STATE+LOC+0,PUT SORT+LOC+0,PUT TYPE+LOC+0,:RST.
                                                                                      101
WARN+X-STATE:                                                                         102
        GET STATE+X+STATE,                                                            102
                (EQUAL+STATE+BLANK,TRACE+X;                                           102
                 EQUAL+STATE+0,TRACE+X;                                               102
                 SIGNAL+X).
                                                                                      102
ENTER LOCAL + HEAD:                                                                   103
        LSEQ + PLOC + MAX LOC, PUT + LLOC + PLOC + HEAD, INCR + PLOC,                 103
                (LSEQ + PLOC + XLOC; MAKE + XLOC + PLOC);                             103
        ERROR + LOC FULL+0,POST MORTEM,EXIT.
                                                                                      103
ENTER GLOBAL + HEAD:                                                                  104
        LSEQ + PGLOB + MAX GLOB, PUT + LGLOB + PGLOB + HEAD, INCR + PGLOB;            104
        ERROR + GLOB FULL+0,POST MORTEM,EXIT.
                                                                                      104
ENTER MACRO + HEAD:                                                                   105
        LSEQ + PMACR + MAX MACR, PUT + LMACR + PMACR + HEAD, INCR + PMACR;            105
        ERROR + MACR FULL+0,POST MORTEM,EXIT.
                                                                                      105
ENTER TEXT + X:                                                                       106
        LSEQ + PTEXT + MAX TEXT, PUT + LTEXT + PTEXT + X, INCR + PTEXT;               106
        ERROR + TEXT FULL+0,POST MORTEM,EXIT.
                                                                                      106
                                                                                      106
[4.4.2   TREATMENT OF TYPES]                                                          107
'ACTION'                                                                              107
DEFINE ACTION,DEFINE LABEL,DEFINE PREDICATE,DEFINE AFFIX,APPLY ACTION,                107
APPLY LABEL,REDEFINE,NEW PLACE,NEW NPARS,APPLY,APPLY PREDICATE.
                                                                                      107
DEFINE ACTION+HEAD:                                                                   108
        REDEFINE+HEAD+DEFINED+GLOBAL+ACTION.
                                                                                      108
DEFINE PREDICATE+HEAD:                                                                109
        REDEFINE+HEAD+DEFINED+GLOBAL+PREDICATE.
                                                                                      109
DEFINE AFFIX+HEAD:                                                                    110
        REDEFINE+HEAD+DEFINED+LOCAL+POINTER.
                                                                                      110
DEFINE LABEL+LAB:                                                                     111
        WAS TAG +LAB,REDEFINE+LAB+DEFINED+LOCAL+LABEL;                                111
        WAS CONS+LAB.
                                                                                      111
APPLY ACTION+HEAD:                                                                    112
        REDEFINE+HEAD+APPLIED+GLOBAL+ACTION.
```

```
                                                                  112
APPLY PREDICATE+HEAD:                                             113
        REDEFINE+HEAD+APPLIED+GLOBAL+PREDICATE.

                                                                  113
APPLY LABEL+LAB:                                                  114
        WAS TAG +LAB,REDEFINE+LAB+APPLIED+LOCAL+LABEL;            114
        WAS CONS+LAB.

                                                                  114
APPLY+X:                                                          115
        NEW PLACE+X,TRY STATE+X+APPLIED.

                                                                  115
REDEFINE+X+NSTATE+NSORT+NTYPE-OLD POS:                            116
        REDEFINE1+X+NSTATE+NSORT+NTYPE,NEW PLACE+X;               116
        MAKE+OLD POS +POS,NLCR,OUT+X IMPOSSIBLE,INFORM+X,         116
             OUT+NSTATE,OUT+NSORT,OUT+NTYPE,POSITION+OLD POS.

                                                                  116
REDEFINE1+X+NSTATE+NSORT+NTYPE:                                   117
        TRY TYPE+X+NTYPE,TRY SORT+X+NSORT,
BACKTRACK?                                  TRY STATE+X+NSTATE.
BACKTRACK?

                                                                  117
TRY STATE+X+S-Q:                                                  118
        GET STATE+X+Q,EQUAL+Q+0,PUT STATE+X+S;                    118
        EQUAL+Q+DEFINED,EQUAL+S+APPLIED,PUT STATE+X+BLANK;        118
        EQUAL+Q+APPLIED,                                          118
                (EQUAL+S+APPLIED;                                 118
                 EQUAL+S+DEFINED,PUT STATE+X+BLANK);              118
        EQUAL+Q+BLANK ,EQUAL+S+APPLIED.

                                                                  118
TRY SORT+X+P-Q:                                                   119
        GET SORT+X+Q,EQUAL+Q+0,PUT SORT+X+P,                      119
                (EQUAL+P+LOCAL,ENTER LOCAL+X;                     119
                 EQUAL+P+GLOBAL,ENTER GLOBAL+X;                   119
                 EQUAL+P+MACRO,ENTER GLOBAL+X);                   119
        EQUAL+P+Q.

                                                                  119
TRY TYPE+X+P-Q:                                                   120
        GET TYPE+X+Q,EQUAL+Q+0,PUT TYPE+X+P;                      120
        EQUAL+P+Q.

                                                                  120
NEW PLACE+X-SORT:                                                 121
        GET SORT+X+SORT,EQUAL+SORT+LOCAL; ADD PLACE+X.

                                                                  121
NEW NPARS+X+P-Q:                                                  122
        GET NPARS+X+Q,EQUAL+Q+P;                                  122
        EQUAL+Q+999,PUT NPARS+X+P;                                122
        ERROR+WRONG NUMBER OF PARAMETERS+X.

                                                                  122
WAS ACTION+HEAD-T:                                                123
```

```
                 GET TYPE+HEAD+T,EQUAL+T+ACTION.
                                                                   123
WAS AFFIX+HEAD-T:GET TYPE+HEAD+T,                                  124
                 EQUAL+T+POINTER;                                  124
                 EQUAL+T+FLAG;                                     124
                 EQUAL+T+LIST.
                                                                   124
WAS MACRO+HEAD-S:                                                  125
                 GET SORT+HEAD+S,EQUAL+S+MACRO.
                                                                   125
WAS PARAMLESS MACRO+HEAD:                                          126
                 WAS TAG+HEAD,WAS MACRO+HEAD,WAS AFFIX+HEAD.
BACKTRACK?
                                                                   126
                                                                   126
[4.4.3   SPECIFICATIONS]                                           127
'ACTION'TREAT SPEC LIST,READ MACRO.
                                                                   127
SPECIFICATION:                                                     128
                 EXTERNAL SPECIFICATION;                           128
                 INTERNAL SPECIFICATION;                           128
                 MACRO SPECIFICATION.
                                                                   128
EXTERNAL SPECIFICATION-TYPE:                                       129
                 R+EXTERNAL,IS TYPE+TYPE,
BACKTRACK?                                 TREAT SPEC LIST+DEFINED+GLOBAL+TYPE.
                                                                   129
IS TYPE+X:MAKE+X+INPT,                                             130
                 R+ACTION;                                         130
                 R+PREDICATE;                                      130
                 R+POINTER;                                        130
                 R+FLAG.
                                                                   130
TREAT SPEC LIST+STATE+SORT+TYPE-HEAD:                              131
     NXT: IS TAG+HEAD,REDEFINE+HEAD+STATE+SORT+TYPE,               131
                 (R+COMMA,:NXT;SHIFT2LINES,R+POINT).
                                                                   131
INTERNAL SPECIFICATION-TYPE:                                       132
                 IS TYPE OF LOCAL+TYPE,TREAT SPEC LIST+APPLIED+GLOBAL+TYPE.
                                                                   132
IS TYPE OF LOCAL+X:MAKE+X+INPT,                                    133
                 R+ACTION;                                         133
                 R+PREDICATE.
                                                                   133
MACRO SPECIFICATION-TYPE-HEAD:                                     134
                 R+MACRO,IS TYPE+TYPE,
BACKTRACK?                                                         134
     NXT: IS TAG+HEAD,
BACKTRACK?                     REDEFINE+HEAD+DEFINED+MACRO+TYPE,READ MACRO+HEAD.   134
```

```
                  (R+COMMA,:NXT;SHIFT2LINES,R+POINT),
                                                                     134
READ MACRO+HEAD-X:                                                   135
          PUT MTEXT+HEAD+PMACR,                                      135
    NXT: (AHEAD + POINT, ENTER MACRO + POINT;                        135
          AHEAD + COMMA, ENTER MACRO + POINT;                        135
          IS TAG+X,TRY STATE+X+APPLIED,
BACKTRACK?                                   ENTER MACRO+X,:NXT;     135
          ENTER MACRO + INPT, NEXT SYMBOL, :NXT).
                                                                     135
                                                                     135
[4.4.4    DECLARATIONS]                                              136
'ACTION'TREAT DECL LIST,CONSTANT TEXT.
                                                                     136
'ACTION'PUT PARAMLESS MACRO.
                                                                     136
DECLARATION:                                                         137
          POINTER DECLARATION;                                       137
          FLAG DECLARATION;                                          137
          LIST DECLARATION.
                                                                     137
POINTER DECLARATION:                                                 138
          R+POINTER,TREAT DECL LIST+Q INTEGER+POINTER.
                                                                     138
TREAT DECL LIST+ALGOLTYPE+TYPE-HEAD:                                 139
          BLANK LINE,G+ALGOLTYPE,                                    139
    NXT: IS TAG+HEAD,REDEFINE+HEAD+DEFINED+GLOBAL+TYPE,G+HEAD,       139
                  (R+COMMA,G+COMMA,:NXT;                             139
                   SHIFT2LINES,R+POINT,G+SEMICOLON).
                                                                     139
FLAG DECLARATION:                                                    140
          R+FLAG,TREAT DECL LIST+Q BOOLEAN+FLAG.
                                                                     140
LIST DECLARATION-HEAD:                                               141
          R+LIST,BLANK LINE,G+Q INTEGER,G+Q ARRAY,U,                 141
    NXT: IS TAG+HEAD,
BACKTRACK?             G+HEAD,REDEFINE+HEAD+DEFINED+GLOBAL+LIST,     141
          R+SUB,
BACKTRACK?      G+SUB,CONSTANT TEXT,                                 141
          R+COLON,
BACKTRACK?          G+COLON,CONSTANT TEXT,                           141
          R+BUS,
BACKTRACK?      G+BUS,                                               141
                  (R+COMMA,G+COMMA,NEW LINE,:NXT;                    141
                   SHIFT2LINES,R+POINT,G+SEMICOLON,L).
                                                                     141
CONSTANT TEXT - SYMB:                                                142
    NXT: IS TAG + SYMB,APPLY+SYMB,                                   142
                  (WAS PARAMLESS MACRO+SYMB,PUT PARAMLESS MACRO+SYMB,:NXT;  142
                   G + SYMB, :NXT);                                  142
          IS CONS + SYMB, G + SYMB, :NXT;                            142
```

```
        R + PLUS, G + PLUS, :NXT;                                                        142
        R + MINUS, G + MINUS, :NXT;
                                                                                          142
                                                                                          142
[4.4.5   AFFIX EXPRESSIONS]                                                               143
'ACTION'PUTAFFIX EXPRESSION,PUT DIRECT,AFFIX PACK OPTION,TREAT AFFIX,PUT SINGLE,          143
PUT PARAMLESS MACRO,PUT MACRO,GET PARAMETERS,GET PAR,PUT PAR.
                                                                                          143
PUT AFFIX EXPRESSION+HEAD:                                                                144
        TRY SORT+HEAD+GLOBAL,PUT DIRECT+HEAD;                                             144
        TRY SORT+HEAD+MACRO,                                                              144
                (WAS ACTION+HEAD,PUT MACRO+HEAD;                                          144
                 G+OPEN,PUT MACRO+HEAD,G+CLOSE).
                                                                                          144
PUT DIRECT+HEAD-NPARS:                                                                    145
        MAKE+NPARS+0,G+HEAD,AFFIX PACK OPTION+NPARS,NEW NPARS+HEAD+NPARS.
                                                                                          145
AFFIX PACK OPTION+NPARS:                                                                  146
        R + PLUS, G + OPEN, TREAT AFFIX,                                                  146
   RST: (INCR+NPARS,R+PLUS,G+COMMA,TREAT AFFIX,:RST;                                      146
         G + CLOSE);
                                                                                          146
TREAT AFFIX - AFFX:                                                                       147
        IS TAG + AFFX, PUT SINGLE + AFFX,APPLY+AFFX;                                      147
        IS BOLD+AFFX,G+STRINGQUOTE,G+AFFX,G+STRINGQUOTE;                                  147
        IS CONS + AFFX, G + AFFX;                                                         147
        ERROR+WRONG AFFIX+AFFX.
                                                                                          147
PUT SINGLE+X:                                                                             148
        WAS AFFIX+X,                                                                      148
                (WAS MACRO+X,PUT PARAMLESS MACRO+X; G+X);                                 148
        REDEFINE+X+APPLIED+GLOBAL+POINTER,G+X.
                                                                                          148
PUT PARAMLESS MACRO+Y-X-M:                                                                149
        GET MTEXT+Y+X,NEW NPARS+Y+0,                                                      149
   NXT: INCR + X, GET + LMACR + X + M,                                                    149
                (EQUAL + M + POINT; G + M, :NXT).
                                                                                          149
PUT MACRO+X-P1-P2-P3-P4-P5-Q-M-NPARS:                                                     150
        MAKE+NPARS+0,GET MTEXT+X+Q,                                                       150
        GET PARAMETERS+NPARS+P1+P2+P3+P4+P5, NEW NPARS+X+NPARS,                           150
   NXT: INCR + Q, GET + LMACR + Q + M,                                                    150
                (EQUAL + M + POINT;                                                       150
                EQUAL+M+ONE  ,PUT PAR+P1+1,:NXT;                                          150
                EQUAL+M+TWO  ,PUT PAR+P2+2,:NXT;                                          150
                EQUAL+M+THREE,PUT PAR+P3+3,:NXT;                                          150
                EQUAL+M+FOUR ,PUT PAR+P4+4,:NXT;                                          150
                EQUAL+M+FIVE ,PUT PAR+P5+5,:NXT;                                          150
                 WAS PARAMLESS MACRO+M,PUT PARAMLESS MACRO+M,:NXT;                        150
                 G + M, :NXT).
                                                                                          150
```

```
GET PARAMETERS+P+P1+P2+P3+P4+P5:
        GET PAR+P+P1,GET PAR+P+P2,GET PAR+P+P3,GET PAR+P+P4,GET PAR+P+P5.

GET PAR+P+X:
        R+PLUS,INCR+P,
                (IS TAG+X,APPLY+X; IS CONS+X);
        MAKE + X + X PARAMETER ERROR.

PUT PAR + X + Y:
        WAS CONS+X,G+X;
        EQUAL+X+X PARAMETER ERROR,ERROR+X+Y;
        WAS BOLD+X,G+STRINGQUOTE,G+X,G+STRINGQUOTE;
        PUT SINGLE + X.


[4.4.6   BUILDING STONES OF A RULE]
'ACTION' PUT CONNECT AND,PUT CONNECT OR,END JUMP,END LABEL,LOST LABEL,PUT INIT,
PUT RESTORE,FRESH LABEL,TRUE RESULT,FALSE RESULT,PUT JUMP,PUT LABEL,SEMICOL,
BEGIN,ENDER,BECOMES.

PUT CONNECT AND:
        GIVE RSTO,PUT JUMP+CUR LAB,INCR+TO NEXT;
        ADVANCED,ERROR+POSSIBLY BACKTRACK NECESSARY+0,PUT JUMP+0,INCR+TO LOST;
        PUT JUMP+CUR LAB,INCR+TO NEXT.

PUT CONNECT OR:
        AHEAD+SEMICOLON,PUT RESTORE,FRESH LABEL,
                (EQUAL+TO NEXT+0,ERROR+ALTERNATIVE NEVER REACHED+0;  );
        AHEAD+CLOSE,
                (EQUAL+TO NEXT+0;
                 PUT LABEL+CUR LAB,FRESH LABEL,
                         (GIVE RSTO;PUT JUMP+0,INCR+TO LOST));
        PUT RESTORE,LOST LABEL.

LOST LABEL:
        EQUAL+TO LOST+0,
                (EQUAL+TO NEXT+0;FALSE RESULT);
        PUT LABEL+0,FALSE RESULT.

END LABEL:
        EQUAL+TO END+0;PUT LABEL+999.

END JUMP:
        EQUAL+INPT+POINT,EQUAL+TO LOST+0,EQUAL+TO NEXT+0;
        PUT JUMP+999,INCR+TO END.

PUT INIT:
        GIVE RSTO,G+INIT RESTORE;

PUT RESTORE:
```

```
        EQUAL+TO NEXT+0;                                                              161
        PUT LABEL+CUR LAB,                                                            161
                (GIVE RSTO,G+DO RESTORE;  ),
                                                                                      161
FRESH LABEL:                                                                          162
        MAKE+CUR LAB+NEW LAB,INCR+NEW LAB.
                                                                                      162
TRUE RESULT:                                                                          163
        WAS ACTION+HANDLE;                                                            163
        G+HANDLE,BECOMES,G+Q TRUE,SEMICOL.
                                                                                      163
FALSE RESULT:                                                                         164
        WAS ACTION+HANDLE;                                                            164
        G+HANDLE,BECOMES,G+Q FALSE,SEMICOL.
                                                                                      164
PUT JUMP + LAB:                                                                       165
        G+Q GOTO, G + LAB, G + SEMICOLON, NEW LINE,APPLY LABEL+LAB.
                                                                                      165
PUT LABEL + LAB:                                                                      166
        NEW LINE,L,G+LAB,G+COLON,U,DEFINE LABEL+LAB,TAB LINE.
                                                                                      166
SEMICOL: G+SEMICOLON.
                                                                                      167
BECOMES:                                                                              168
        G+COLON,G+EQUALS.
                                                                                      168
BEGIN: NEW LINE, G + Q BEGIN,U,TAB LINE.
                                                                                      169
ENDBR: NEW LINE, L, G + Q END.
                                                                                      170
                                                                                      170
[4.4.7   GENERAL FORM OF A RULE]                                                      171
'ACTION'OPTIONAL BOUND AFFIXES,REST BOUND AFFIXES,OPTIONAL FREE AFFIXES,RESTLHS.
                                                                                      171
RULE:                                                                                 172
        LEFT HAND SIDE,MIDDLE,
BACKTRACK?                          RIGHT HAND SIDE,
BACKTRACK?                                                                            172
        FORGET LOCALS,SHIFT2LINES,R+POINT.
BACKTRACK?
                                                                                      172
LEFT HAND SIDE:                                                                       173
        IS TAG+HANDLE,BLANK LINE,MAKE+BOUNDX+0,MAKE+SPECX+0,                          173
                (WAS ACTION+HANDLE,DEFINE ACTION+HANDLE,REST LHS;                     173
                 G+Q BOOLEAN,DEFINE PREDICATE+HANDLE,REST LHS).
                                                                                      173
REST LHS:                                                                             174
```

```
        G+Q PROCEDURE,G+HANDLE,OPTIONAL BOUND AFFIXES,NEW NPARS+HANDLE+BOUNDX,    174
        SEMICOL.

                                                                                  174
OPTIONAL BOUND AFFIXES-AFFX:                                                      175
        BOUND AFFIX1+AFFX,G+OPEN,G+AFFX,REST BOUND AFFIXES;                       175
        BOUND AFFIX2+AFFX,G+OPEN,G+AFFX,REST BOUND AFFIXES,G+AFFX; .

                                                                                  175
REST BOUND AFFIXES-AFFX:                                                          176
        BOUND AFFIX1+AFFX,G+COMMA,G+AFFX,REST BOUND AFFIXES;                      176
        BOUND AFFIX2+AFFX,G+COMMA,G+AFFX,REST BOUND AFFIXES,G+AFFX,DECR+SPECX,    176
             (EQUAL+SPECX+0;G+COMMA);                                             176
        G+CLOSE,                                                                  176
             (EQUAL+SPECX+0;SEMICOL,G+Q INTEGER).

                                                                                  176
BOUND AFFIX1+AFFX:                                                                177
        R+TIMES,IS TAG+AFFX,
BACKTRACK?                       DEFINE AFFIX+AFFX,INCR+BOUNDX.

                                                                                  177
BOUND AFFIX2+AFFX:                                                                178
        R+PLUS ,IS TAG+AFFX,
BACKTRACK?                       DEFINE AFFIX+AFFX,INCR+BOUNDX,INCR+SPECX.

                                                                                  178
MIDDLE:                                                                           179
        BEGIN, OPTIONAL FREE AFFIXES, R + COLON,PUT INIT.

                                                                                  179
OPTIONAL FREE AFFIXES - AFFX:                                                     180
        R + MINUS, G + Q INTEGER,                                                 180
   FFX: IS TAG + AFFX,
BACKTRACK?                 G + AFFX, DEFINE AFFIX + AFFX,                         180
             (R + MINUS, G + COMMA, :FFX;                                         180
              G + SEMICOLON);  ,

                                                                                  180
RIGHT HAND SIDE:                                                                  181
        MAKE+CUR LAB+1,MAKE+NEW LAB+2,MAKE+TO LOST+0,MAKE+TO END+0,               181
   NXT: ALTERNATIVE,PUT CONNECT OR,                                               181
             (R+SEMICOLON,:NXT;                                                   181
              END LABEL,END BR,SEMICOL).

                                                                                  181
ALTERNATIVE-LAB-OLD NEXT:                                                         182
        MAKE+TO NEXT+0,SET+ADVANCED+FALSE,                                        182
   NXT: R+COMMA,:NXT;                                                             182
        R+COLON,IS TAG+LAB,
BACKTRACK?                      PUT JUMP+LAB;                                     182
        R+OPEN,MAKE+OLD NEXT+TO NEXT,CHOICE,
BACKTRACK?                                          R+CLOSE,
BACKTRACK?                                                                        182
             (GIVE RSTO,MAKE+TO NEXT+1;MAKE+TO NEXT+OLD NEXT);                    182
        MEMBER,:NXT;                                                              182
        TRUE RESULT,END JUMP.

                                                                                  182
CHOICE-LAB:                                                                       183
```

```
          MAKE+LAB+CUR LAB,BEGIN,PUT INIT,FRESH LABEL,                         183
      NXT: ALTERNATIVE,PUT CONNECT OR,                                        183
                    (R+SEMICOLON,:NXT;ENDBR,SEMICOL,MAKE+CUR LAB+LAB).
                                                                              183
MEMBER-HEAD:                                                                  184
          IS TAG+HEAD,APPLY+HEAD,                                             184
                    (R+COLON,PUT LABEL+HEAD;                                  184
                     TRY TYPE+HEAD+PREDICATE,                                 184
                              G+QIF,G+NOT,PUT AFFIX EXPRESSION+HEAD,G+Q THEN, 184
                                        PUT CONNECT AND,SET+ADVANCED+TRUE;    184
                     TRY TYPE+HEAD+FLAG,                                      184
                              G+QIF,G+NOT,PUT AFFIX EXPRESSION+HEAD,G+Q THEN, 184
                                        PUT CONNECT AND;                      184
                     TRY TYPE+HEAD+ACTION,                                    184
                              PUT AFFIX EXPRESSION+HEAD,SEMICOL).

                                                                              184
                                                                              184
[4.4.8   OTHER BUILDING STONES OF A GRAMMAR]                                  185
'ACTION'SKIP UNTIL POINT,TERMINALS.

                                                                              185
COMMAND:                                                                      186
          R+RSTO ON,SET+GIVE RSTO+TRUE;                                       186
          R+RSTO OFF,SET+GIVE RSTO+FALSE;                                     186
          R+SHORT,MAKE+LEGIBLE+FALSE;                                         186
          R+LONG,MAKE+LEGIBLE+TRUE;                                           186
          R + TRACE ON, SET + GIVE TRACE + TRUE;                              186
          R + TRACE OFF, SET + GIVE TRACE + FALSE.

                                                                              186
COMMENT:                                                                      187
          R+SUB,BLANK LINE,G+Q COMMENT,                                       187
             RST: (R + BUS, G + SEMICOLON;                                    187
                     G+INPT,READ+INPT,:RST).

                                                                              187
                                                                              187
STARTING SYMBOL-HEAD:                                                         188
          R+RESULT,IS TAG+HEAD,
BACKTRACK?                                NEW PAGE,                           188
                    (TRY TYPE+HEAD+ACTION,APPLY ACTION+HEAD,TERMINALS,G+HEAD; 188
                     APPLY PREDICATE+HEAD,TERMINALS,G+HEAD).

                                                                              188
SKIP UNTIL POINT:                                                             189
      NXT: ERROR + SKIPPED + INPT,                                            189
                    (R + POINT; NEXT SYMBOL,:NXT).

                                                                              189
                                                                              189
[4.4.9   TERMINALS]                                                           190
'ACTION' TERMINALS,FORGET GLOBALS,MAY BE TERMINAL,PUT DECL,FORGET TERMINALS,   190
PUT CALL.

                                                                              190
TERMINALS:                                                                    191
          BLANK LINE,FORGET GLOBALS,NEW LINE,EQUAL+PLOC+MIN LOC;              191
          APPLY ACTION+XREAD,APPLY ACTION+XOUT,APPLY ACTION+INIT READ,        191
```

```
              BLANK LINE, G + INIT READ, G + SEMICOLON, NEW LINE,          191
              BLANK LINE,FORGET TERMINALS,NEW LINE,BLANK LINE.
                                                                           191
FORGET GLOBALS-P-TERM:                                                     192
              MAKE+P+MIN GLOB,MAKE+PLOC+MIN LOC,                           192
         NXT: EQUAL + P + PGLOB;                                           192
              GET+LGLOB+P+TERM,MAY BE TERMINAL+TERM,INCR+P,:NXT.
                                                                           192
MAY BE TERMINAL+X:                                                         193
              WAS TERMINAL+X,PUT DECL+X,ENTER LOCAL+X;                     193
              WARN+X.
                                                                           193
WAS TERMINAL+X-S:                                                          194
              GET STATE+X+S,EQUAL+S+APPLIED,                               194
              GET SORT +X+S,EQUAL+S+GLOBAL,                                194
              GET TYPE+X+S,EQUAL+S+POINTER.
                                                                           194
PUT DECL + X:                                                              195
              NEW LINE,G+Q INTEGER,G+X,G+SEMICOLON,POSITION+64,INFORM+X.
                                                                           195
FORGET TERMINALS-P-TERM:                                                   196
              MAKE+P+MIN LOC,                                              196
         NXT: EQUAL+P+PLOC;                                                196
              GET+LLOC+P+TERM,PUT CALL+X READ+TERM,INCR+P,:NXT.
                                                                           196
PUT CALL +PROC + X:                                                        197
              NEW LINE, G + PROC, G + OPEN, G + X, G + CLOSE, G + SEMICOLON.
                                                                           197
                                                                           197
[4.4.10  POST MORTEM]                                                      198
'ACTION'POST MORTEM,DICTIONARY,LIST TAGS,ENTRIES.
                                                                           198
POST MORTEM:                                                               199
              NLCR,OUTINT+MIN TAG ,OUTINT+PTAG ,OUTINT+MAX TAG ,           199
              NLCR,OUTINT+MIN BOLD,OUTINT+PBOLD,OUTINT+MAX BOLD,           199
              NLCR,OUTINT+MIN SPEC,OUTINT+PSPEC,OUTINT+MAX SPEC,           199
              NLCR,OUTINT+MIN CONS,OUTINT+PCONS,OUTINT+MAX CONS,           199
              NLCR,OUTINT+MIN LOC ,OUTINT+XLOC ,OUTINT+MAX LOC ,           199
              NLCR,OUTINT+MIN GLOB,OUTINT+PGLOB,OUTINT+MAX GLOB,           199
              NLCR,OUTINT+MIN MACR,OUTINT+PMACR,OUTINT+MAX MACR,           199
              NLCR,OUTINT+MIN TEXT,OUTINT+PTEXT,OUTINT+MAX TEXT,           199
              NLCR,OUTINT+LINE,OUTINT+CARD,NLCR,NLCR,NLCR,DICTIONARY.
                                                                           199
DICTIONARY:                                                                200
              LIST TAGS + FIRST TAG.
                                                                           200
LIST TAGS + X - P-STATE-SORT-TYPE:                                         201
              EQUAL + X + 0;                                               201
              GET LEFT+X+P,LIST TAGS+P,                                    201
              GET STATE+X+STATE,GET SORT+X+SORT,GET TYPE+X+TYPE.           201
```

```
              EQUAL+SORT+0,:SKP;                                                    201
              NLCR,OUT+X,POSITION+32,OUT+STATE,OUT+SORT,OUT+TYPE,                   201
              POSITION+59,ENTRIES+X,                                                201
         SKP: GET RIGHT+X+R,LIST TAGS+P.
                                                                                    201
ENTRIES+P-Q-H-K:                                                                    202
              GET PLACE+P+Q,EQUAL+Q+0,NLCR;                                         202
         LNE: GET+LTEXT+Q+H,UNPACK2+H+Q+K,OUTINT+K,                                 202
                       (EQUAL+Q+0,NLCR;                                             202
                        ADD+Q+MIN TEXT+Q,DECR+Q,:LNE).
                                                                                    202
                                                                                    202
[4.5,      HEART OF COMPILER COMPILER]                                              203
'ACTION'SENTENCE,START SYSTEM,COMPILER DESCRIPTION.
                                                                                    203
'POINTER' INIT RESTORE,DO RESTORE,XREAD,XOUT,INIT READ,TITLE.
                                                                                    204
SENTENCE:                                                                           205
              START SYSTEM, BLANK LINE,                                             205
              BEGIN,COMPILER DESCRIPTION,ENDBR,                                     205
              BLANK LINE,NEW PAGE,POST MORTEM.
                                                                                    205
START SYSTEM:                                                                       206
              READ+INIT RESTORE,READ+DO RESTORE,READ+X READ,READ+X OUT,             206
              READ+INIT READ,READ+TITLE,                                            206
              SET+GIVERSTO+FALSE,SET+GIVE TRACE+FALSE,SET+GIVE TEXT+TRUE,           206
              MAKE+PLOC+MIN LOC,MAKE+PGLOB+MIN GLOB,MAKE+PTEXT+MIN TEXT,            206
              MAKE+PMACR+MIN MACR,MAKE+CARD+0.MAKE+CPOS+1,MAKE+XLOC+0,              206
              SET+LINED UP+FALSE,SET+LEGIBLE+TRUE,                                  206
              MAKE+INDENTATION+0,MAKE+LINE+0,MAKE+POS+0,OUT+TITLE,READ+INPT.
                                                                                    206
COMPILER DESCRIPTION:                                                               207
         NXT: SPECIFICATION, :NXT;                                                  207
              DECLARATION, :NXT;                                                    207
              COMMAND, :NXT;                                                        207
              COMMENT, :NXT;                                                        207
              STARTING SYMBOL;                                                      207
              RULE, :NXT;                                                           207
              SKIP UNTIL POINT, :NXT.
                                                                                    207
                                                                                    207
'RESULT' SENTENCE.
```

```
APPLIED GLOBAL POINTER TAGFULL
APPLIED GLOBAL POINTER BOLDFULL
APPLIED GLOBAL POINTER SPECFULL
APPLIED GLOBAL POINTER CONSFULL
APPLIED GLOBAL POINTER WRONGINPUTCODE
APPLIED GLOBAL POINTER BLANK
APPLIED GLOBAL POINTER LOCFULL
APPLIED GLOBAL POINTER GLOBFULL
APPLIED GLOBAL POINTER MACRFULL
APPLIED GLOBAL POINTER TEXTFULL
APPLIED GLOBAL POINTER DEFINED
APPLIED GLOBAL POINTER GLOBAL
APPLIED GLOBAL POINTER ACTION
APPLIED GLOBAL POINTER PREDICATE
APPLIED GLOBAL POINTER LOCAL
APPLIED GLOBAL POINTER POINTER
APPLIED GLOBAL POINTER LABEL
APPLIED GLOBAL POINTER APPLIED
APPLIED GLOBAL POINTER XIMPOSSIBLE
APPLIED GLOBAL POINTER MACRO
APPLIED GLOBAL POINTER WRONGNUMBEROFPARAMETERS
APPLIED GLOBAL POINTER FLAG
APPLIED GLOBAL POINTER LIST
APPLIED GLOBAL POINTER EXTERNAL
APPLIED GLOBAL POINTER COMMA
APPLIED GLOBAL POINTER POINT
APPLIED GLOBAL POINTER QINTEGER
APPLIED GLOBAL POINTER SEMICOLON
APPLIED GLOBAL POINTER QBOOLEAN
APPLIED GLOBAL POINTER QARRAY
APPLIED GLOBAL POINTER SUB
APPLIED GLOBAL POINTER COLON
APPLIED GLOBAL POINTER BUS
APPLIED GLOBAL POINTER PLUS
APPLIED GLOBAL POINTER MINUS
APPLIED GLOBAL POINTER OPEN
APPLIED GLOBAL POINTER CLOSE
APPLIED GLOBAL POINTER STRINGQUOTE
APPLIED GLOBAL POINTER WRONGAFFIX
APPLIED GLOBAL POINTER ONE
APPLIED GLOBAL POINTER TWO
APPLIED GLOBAL POINTER THREE
APPLIED GLOBAL POINTER FOUR
APPLIED GLOBAL POINTER FIVE
APPLIED GLOBAL POINTER XPARAMETERERROR
APPLIED GLOBAL POINTER POSSIBLYBACKTRACKNECESSARY
APPLIED GLOBAL POINTER ALTERNATIVENEVERREACHED
APPLIED GLOBAL POINTER QTRUE
APPLIED GLOBAL POINTER QFALSE
APPLIED GLOBAL POINTER QGOTO
APPLIED GLOBAL POINTER EQUALS
APPLIED GLOBAL POINTER QBEGIN
APPLIED GLOBAL POINTER QEND
APPLIED GLOBAL POINTER QPROCEDURE
APPLIED GLOBAL POINTER TIMES
APPLIED GLOBAL POINTER QIF
APPLIED GLOBAL POINTER NOT
APPLIED GLOBAL POINTER QTHEN
APPLIED GLOBAL POINTER RSTOON
```

APPLIED GLOBAL POINTER RSTOOFF
APPLIED GLOBAL POINTER SHORT
APPLIED GLOBAL POINTER LONG
APPLIED GLOBAL POINTER TRACEON
APPLIED GLOBAL POINTER TRACEOFF
APPLIED GLOBAL POINTER QCOMMENT
APPLIED GLOBAL POINTER RESULT
APPLIED GLOBAL POINTER SKIPPED

```
100001 104890 110000
200001 200173 200600
300001 300018 300100
400001 400048 400300
500001 500068 500200
600001 600353 601000
700001 700534 701100
800001 801910 804000
809 211
```

ACCC	MACRO POINTER	244 242 20
ACCENT	GLOBAL PREDICATE	308 306 304 302 301 244
ACTION	APPLIED GLOBAL POINTER	723 704 501 491 472 435 426
ADD	MACRO ACTION	783 293 137 54
ADDMULT	MACRO ACTION	344 56
ADDPLACE	GLOBAL ACTION	466 390 389
ADVANCED	GLOBAL FLAG	700 684 603 383
AFFIXPACKOPTION	GLOBAL ACTION	556 555 547
AHEAD	GLOBAL PREDICATE	608 606 510 509 367 366
ALTERNATIVENEVERREACHED	APPLIED GLOBAL POINTER	607
ALTERNATIVE	GLOBAL PREDICATE	693 683 680
APPLIED	APPLIED GLOBAL POINTER	745 568 511 499 455 453 452 451 442 439 437 435
APPLYACTION	GLOBAL ACTION	734 723 434 423
APPLYLABEL	GLOBAL ACTION	637 438 424
APPLYPREDICATE	GLOBAL ACTION	724 436 424
APPLY	GLOBAL ACTION	696 589 561 539 441 424
BECOMES	GLOBAL ACTION	641 635 632 600
BEGIN	GLOBAL ACTION	790 692 672 643 600
BLANKLINE	GLOBAL ACTION	791 789 736 735 733 717 652 531 524 219 218 204
BLANK	APPLIED GLOBAL POINTER	455 454 451 404
BOLDCHAR	GLOBAL PREDICATE	309 307 305 303 241
BOLDFULL	APPLIED GLOBAL POINTER	312
BOUNDAFFIX1	GLOBAL PREDICATE	667 662 659
BOUNDAFFIX2	GLOBAL PREDICATE	669 663 660

BOUNDX	GLOBAL POINTER	670 668 656 652 384
BUS	APPLIED GLOBAL POINTER	718 535
CARD	GLOBAL POINTER	797 768 251 190 166
CHAR	GLOBAL POINTER	376 362 260 259 258 256 253 250 248 246 244 243 242 240 238 236 231
CHOICE	GLOBAL PREDICATE	691 687
CLOSE	APPLIED GLOBAL POINTER	755 687 665 608 559 553
COLON	APPLIED GLOBAL POINTER	697 686 672 642 639 534
COMMAND	GLOBAL PREDICATE	803 709
COMMA	APPLIED GLOBAL POINTER	685 676 664 663 662 558 536 526 510 506 497
COMMENT	GLOBAL PREDICATE	804 716
COMPILERDESCRIPTION	GLOBAL ACTION	800 790 786
CONSFULL	APPLIED GLOBAL POINTER	348
CONSTANTTEXT	GLOBAL ACTION	538 534 533 515
CPOS	GLOBAL POINTER	797 201 200 192 190 189 166
CSYM	GLOBAL ACTION	201 200 190 189 12
CURLAB	GLOBAL POINTER	694 692 679 629 626 610 604 602 384
DECLARATION	GLOBAL PREDICATE	802 517
DECR	MACRO ACTION	783 663 400 216 214 64
DEFINEACTION	GLOBAL ACTION	653 425 423
DEFINEAFFIX	GLOBAL ACTION	675 670 668 429 423
DEFINED	APPLIED GLOBAL POINTER	532 525 505 489 454 451 432 430 428 426
DEFINELABEL	GLOBAL ACTION	639 431 423
DEFINEPREDICATE	GLOBAL ACTION	654 427 423
DICTIONARY	GLOBAL ACTION	769 768 758
DIGIT	GLOBAL PREDICATE	344 343 239
DISPLAYCHARACTER	GLOBAL ACTION	249 248 234
DIVREM	MACRO ACTION	192 179 176 125 115 112 57
DOLLARCODE	MACRO POINTER	201 165
DORESTORE	GLOBAL POINTER	793 787 627
ENDBR	GLOBAL ACTION	790 694 682 644 600

```
ENDJUMP                     GLOBAL ACTION             690  619  598
ENDLABEL                    GLOBAL ACTION             682  617  598
ENTERBOLD                   GLOBAL ACTION             373  313  299
ENTERCONS                   GLOBAL ACTION             375  349  341
ENTERGLOBAL                 GLOBAL ACTION             460  459  411  388
ENTERLOCAL                  GLOBAL ACTION             742  458  407  388
ENTERMACRO                  GLOBAL ACTION             512  511  510  509  414  388
ENTERSPEC                   GLOBAL ACTION             374  332  326
ENTERTAG                    GLOBAL ACTION             372  277  263
ENTERTEXT                   GLOBAL ACTION             417  397  396  388
ENTRIES                     GLOBAL ACTION             779  777  758
EQUALS                    APPLIED GLOBAL POINTER      642
EQUAL                       MACRO FLAG                782  780  775  772  752  747  746  745  739  733  666  664  625  620  618  615  614
609  607  593  582  581  580  579  578  577  572  478  476  475  474  472  469  468  466  464  463  461  460  459  458  457  455  454  453  452
 451  450  405  404  399  396  392  367  353  352  336  335  319  317  290  287  283  281  260  259  258  256  250  244  242  199  198  197  17
8  175  158  143  142  141  126  114  111  70
ERROR                       GLOBAL ACTION             726  607  603  593  564  470  419  416  413  410  376  348  331  312  296  276  155
147
EXIT                        GLOBAL ACTION             419  416  413  410  348  331  312  296  276  12
EXTERNALSPECIFICATION       GLOBAL PREDICATE          488  485
EXTERNAL                  APPLIED GLOBAL POINTER      489
FALSERESULT                 GLOBAL ACTION             633  616  615  599
FALSE                       MACRO POINTER             798  795  715  712  711  684  360  220  72
FIRSTTAG                    GLOBAL POINTER            770  362  278  90
FIVE                      APPLIED GLOBAL POINTER      582
FLAGDECLARATION             GLOBAL PREDICATE          528  519
FLAG                      APPLIED GLOBAL POINTER      701  529  494  475
FORGETGLOBALS               GLOBAL ACTION             737  733  730
FORGETLOCALS                GLOBAL ACTION             650  398  388
FORGETTERMINALS             GLOBAL ACTION             750  736  730
FOUR                      APPLIED GLOBAL POINTER      581
FRESHLABEL                  GLOBAL ACTION             692  628  610  606  599
```

```
GET                     MACRO ACTION            781  753  740  576  571  400  393  351  350  334  333  323  316  280  182  171  118
106  49

GETLEFT                 MACRO ACTION            773  286  82

GETMTEXT                MACRO ACTION            574  570  77

GETNPARS                MACRO ACTION            468  76

GETPARAMETERS           GLOBAL ACTION           585  575  548

GETPAR                  GLOBAL ACTION           587  586  548

GETPLACE                MACRO ACTION            780  391  78

GETRIGHT                MACRO ACTION            778  289  83

GETSORT                 MACRO ACTION            774  746  478  466  457  153  80

GETSTATE                MACRO ACTION            774  745  450  403  153  79

GETTAIL                 MACRO ACTION            393  62

GETTYPE                 MACRO ACTION            774  747  473  472  463  153  81

GIVERSTO                GLOBAL FLAG             795  711  710  688  627  623  611  602  383

GIVETEXT                GLOBAL FLAG             795  360  253  251  89

GIVETRACE               GLOBAL FLAG             795  715  714  149  89

GLOBAL                  APPLIED GLOBAL POINTER  746  568  550  532  525  499  489  459  437  435  428  426

GLOBFULL                APPLIED GLOBAL POINTER  413

G                       GLOBAL ACTION           755  749  735  724  723  719  718  717  702  699  677  676  675  674  666  665  664
663  662  660  659  656  654  644  643  642  640  639  637  635  632  627  623  594  592  584  572  568  567  563  562  559  558  557  555  553
 544  543  542  541  537  536  535  534  533  532  531  527  526  525  524  207  204

HANDLE                  GLOBAL POINTER          656  654  653  652  635  634  632  631  384

INCR                    MACRO ACTION            753  740  670  668  629  621  611  604  603  602  588  576  571  558  418  415  412
406  397  355  347  338  330  323  322  321  311  285  275  250  215  200  193  190  189  184  144  132  129  119  63

INDENTATION             GLOBAL POINTER          799  216  215  211  206

INFORM                  GLOBAL ACTION           749  445  157  152  151  149  147

INITIALIZEFORREADING    GLOBAL ACTION           809  359  358

INITREAD                GLOBAL POINTER          794  787  735  734

INITRESTORE             GLOBAL POINTER          793  787  623

INPT                    GLOBAL POINTER          799  726  719  620  512  500  490  369  367  365  364  363  231

INTERNALSPECIFICATION   GLOBAL PREDICATE        498  486

ISEOLD                  GLOBAL PREDICATE        562  364
```

ISCONS	GLOBAL PREDICATE	589 563 542 365
ISTAG	GLOBAL PREDICATE	722 696 686 675 670 668 652 589 561 539 532 525 511 505 496 363
ISTYPE	GLOBAL PREDICATE	504 490 489
ISTYPEOFLOCAL	GLOBAL PREDICATE	500 499
LABEL	APPLIED GLOBAL POINTER	439 432
LAYOUTSYMBOL	GLOBAL PREDICATE	371 257
LEFTHANDSIDE	GLOBAL PREDICATE	651 649
LEFT	MACRO FLAG	286 67
LEGIBLE	GLOBAL FLAG	798 713 712 221 218 211 89
LESS	MACRO FLAG	393 293 200 193 174 132 129 110 68
LETGIT	GLOBAL PREDICATE	268 267 266 237
LETTER	GLOBAL PREDICATE	265 235
LGLOB	GLOBAL LIST	740 412 38
LINEDUP	GLOBAL FLAG	798 220 218 211 210 205
LINEUP	GLOBAL ACTION	209 208 204
LINE	GLOBAL POINTER	799 768 397 396 393 361 252 250 231
LISTDECLARATION	GLOBAL PREDICATE	530 520
LISTTAGS	GLOBAL ACTION	778 773 771 770 758
LIST	APPLIED GLOBAL POINTER	532 531 476
LLOC	GLOBAL LIST	753 408 400 37
LMACR	GLOBAL LIST	576 571 415 40
LOCAL	APPLIED GLOBAL POINTER	466 458 439 432 430
LOCFULL	APPLIED GLOBAL POINTER	410
LONG	APPLIED GLOBAL POINTER	713
LOSTLABEL	GLOBAL ACTION	613 612 598
LSEQ	MACRO FLAG	418 415 412 409 408 347 330 311 275 213 189 124 69
LTEXT	GLOBAL LIST	781 418 393 39
L	GLOBAL ACTION	644 639 537 216 204
MACRFULL	APPLIED GLOBAL POINTER	416
MACROSPECIFICATION	GLOBAL PREDICATE	503 487

```
MACRO            APPLIED GLOBAL POINTER  551 505 504 478 460

MAKE             MACRO ACTION            799 797 796 751 738 713 712 694 692 688 687 684 679 652 629 590 574
555 500 490 445 409 365 364 363 362 361 360 354 350 345 343 337 333 328 322 320 315 314 301 291 288 284 279 278 269
265 248 246 243 242 240 238 236 214 201 190 181 174 156 136 134 131 128 117 110 51

MARKED           MACRO FLAG              322 318 282 183 119 66

MARK             MACRO ACTION            328 308 306 304 302 273 183 174 119 110 53

MAXBCLD          MACRO POINTER           761 311 34 25

MAXCARD          MACRO POINTER           200 189 165

MAXCONS          MACRO POINTER           763 347 36 27

MAXGLOB          MACRO POINTER           765 412 38 29

MAXLOC           MACRO POINTER           764 408 37 28

MAXMACR          MACRO POINTER           766 415 40 30

MAXSPEC          MACRO POINTER           762 330 35 26

MAXTAG           MACRO POINTER           760 293 275 33 24

MAXTEXT          MACRO POINTER           767 418 39 31

MAYBETERMINAL    GLOBAL ACTION           741 740 730

MEMBER           GLOBAL PREDICATE        695 689

MIDDLE           GLOBAL PREDICATE        671 649

MINBCLD          MACRO POINTER           761 361 314 34 25

MINCONS          MACRO POINTER           763 360 350 36 27

MINGLOB          MACRO POINTER           796 765 738 38 29

MINLOC           MACRO POINTER           796 764 751 738 733 399 37 28

MINMACR          MACRO POINTER           797 766 40 30

MINSPEC          MACRO POINTER           762 361 333 35 26

MINTAG           MACRO POINTER           760 361 33 24

MINTEXT          MACRO POINTER           796 783 767 397 39 31

MINUSCODE        MACRO POINTER           174 110 20

MINUS            APPLIED GLOBAL POINTER  676 674 544

NEWCARD          GLOBAL ACTION           220 197 188 162

NEWLAB           GLOBAL POINTER          679 629 384

NEWLINE          GLOBAL ACTION           755 749 736 735 733 644 643 639 637 536 217 204
```

NEWNPARS	GLOBAL ACTION	656 575 570 555 467 424
NEWPAGE	GLOBAL ACTION	791 722 12
NEWPLACE	GLOBAL ACTION	465 444 442 424
NEXTCHAR	GLOBAL ACTION	376 371 255 247 244 243 242 234
NEXTNONSPACECHAR	GLOBAL ACTION	254 246 240 238 236 234
NEXTSYMBOL	GLOBAL ACTION	727 512 368 366 365 364 363 358
NIX	MACRO POINTER	328 306 304 302 272 271 199 143 20
NLCRCODE	MACRO POINTER	259 250 201 197 190 141 134 20
NLCR	GLOBAL ACTION	782 780 776 768 767 766 765 764 763 762 761 760 445 156 141 136 133 125 99
NOT	APPLIED GLOBAL POINTER	702 699
ONE	APPLIED GLOBAL POINTER	578
OPEN	APPLIED GLOBAL POINTER	755 687 660 659 557 553
OPTIONALBOUNDAFFIXES	GLOBAL ACTION	658 656 647
OPTIONALFREEAFFIXES	GLOBAL ACTION	673 672 647
OUT	GLOBAL ACTION	809 799 776 446 445 159 156 154 102 99
OUTINT	GLOBAL ACTION	781 768 767 766 765 764 763 762 761 760 110 108 107 106 99
OUTINT1	GLOBAL ACTION	252 251 115 113 112 99
PACK2	MACRO ACTION	397 60
PACK3	MACRO ACTION	328 307 306 304 302 272 271 270 268 58
PBOLD	GLOBAL POINTER	761 361 320 311 301 87
PCONS	GLOBAL POINTER	763 360 354 347 345 87
PGLOB	GLOBAL POINTER	796 765 739 412 88
PLOC	GLOBAL POINTER	796 752 738 733 409 408 400 399 88
PLUS	APPLIED GLOBAL POINTER	670 588 558 557 543
PMACR	GLOBAL POINTER	797 766 508 415 88
POINTERDECLARATION	GLOBAL PREDICATE	521 518
POINTER	APPLIED GLOBAL POINTER	747 568 522 493 474 430
POINT	APPLIED GLOBAL POINTER	727 650 620 577 572 537 527 510 509 506 497
POSITION	GLOBAL ACTION	777 776 749 446 252 251 159 158 157 151 149 123 100

POS	GLOBAL POINTER	799 445 156 144 137 136 134 124 101
POSSIBLYBACKTRACKNECESSARY	APPLIED GLOBAL POINTER	603
POSTMORTEM	GLOBAL ACTION	791 759 758 419 416 413 410 348 331 312 296 276 229
PRCHAR	GLOBAL ACTION	253 140 139 122 115 112 111 110 99
PREDICATE	APPLIED GLOBAL POINTER	698 502 492 437 428
PRINT_	GLOBAL ACTION	120 119 99
PRINT	GLOBAL ACTION	116 105 104 103 99
PRSYM	GLOBAL ACTION	144 137 134 12
PSPEC	GLOBAL POINTER	762 361 337 330 328 87
PTAG	GLOBAL POINTER	760 362 361 295 294 293 284 275 265 87
PTEXT	GLOBAL POINTER	796 767 418 395 88
PUCHAR	GLOBAL ACTION	201 196 195 187 179 176 175 174 163
PUNCH1	GLOBAL ACTION	185 184 183 162
PUNCH	GLOBAL ACTION	180 170 169 168 162
PUT	MACRO ACTION	418 415 412 408 347 330 311 275 50
PUTABS	GLOBAL ACTION	214 212 211 204
PUTAFFIXEXPRESSION	GLOBAL ACTION	705 702 699 549 547
PUTCALL	GLOBAL ACTION	754 753 731
PUTCONNECTAND	GLOBAL ACTION	703 700 601 598
PUTCONNECTOR	GLOBAL ACTION	693 680 605 598
PUTDECL	GLOBAL ACTION	748 742 730
PUTDIRECT	GLOBAL ACTION	554 550 547
PUTINIT	GLOBAL ACTION	692 672 622 598
PUTJUMP	GLOBAL ACTION	686 636 621 611 604 603 602 599
PUTLABEL	GLOBAL ACTION	697 638 626 618 616 610 599
PUTLEFT	MACRO ACTION	295 287 82
PUTMACRO	GLOBAL ACTION	573 553 552 548
PUTMTEXT	MACRO ACTION	508 294 77
PUTNPARS	MACRO ACTION	469 293 76
PUTPARAMLESSMACRO	GLOBAL ACTION	583 569 567 548 540 516

PUTPAR	GLOBAL ACTION	591 582 581 580 579 578 548
PUTPLACE	MACRO ACTION	395 294 78
PUTRESTORE	GLOBAL ACTION	624 612 606 599
PUTRIGHT	MACRO ACTION	295 290 83
PUTSINGLE	GLOBAL ACTION	595 565 561 547
PUTSCRT	MACRO ACTION	457 401 294 80
PUTSTATE	MACRO ACTION	454 451 450 401 294 79
PUTTYPE	MACRO ACTION	463 401 295 81
QARRAY	APPLIED GLOBAL POINTER	531
QBEGIN	APPLIED GLOBAL POINTER	643
QBOOLEAN	APPLIED GLOBAL POINTER	654 529
QCOMMENT	APPLIED GLOBAL POINTER	717
QEND	APPLIED GLOBAL POINTER	644
QFALSE	APPLIED GLOBAL POINTER	635
QGOTO	APPLIED GLOBAL POINTER	637
QIF	APPLIED GLOBAL POINTER	702 699
QINTEGER	APPLIED GLOBAL POINTER	749 674 666 531 522
QPROCEDURE	APPLIED GLOBAL POINTER	656
QTHEN	APPLIED GLOBAL POINTER	702 699
QTRUE	APPLIED GLOBAL POINTER	632
QUOTE	MACRO POINTER	242 20
READBOLD	GLOBAL PREDICATE	373 300
READCONS	GLOBAL PREDICATE	375 342
READMACRO	GLOBAL ACTION	507 505 483
READSPEC	GLOBAL PREDICATE	374 327
READTAG	GLOBAL PREDICATE	372 264
READ	GLOBAL ACTION	809 799 794 793 719 376 370 369 358
REDEFINE1	GLOBAL PREDICATE	447 444
REDEFINE	GLOBAL ACTION	568 532 525 505 496 443 439 437 435 432 430 428 426 424
REPLACE	GLOBAL ACTION	394 393 392 389

```
RESERVEADMINSPACE      GLOBAL ACTION            362 292 290 287 283 263
RESTBOUNDAFFIXES       GLOBAL ACTION            663 662 661 660 659 647
RESTLHS                GLOBAL ACTION            655 654 653 647
RESULT                 APPLIED GLOBAL POINTER   722
RESYM                  GLOBAL POINTER           362 248 13
RIGHTHANDSIDE          GLOBAL PREDICATE         678 649
RSTOCFF                APPLIED GLOBAL POINTER   711
RSTOCN                 APPLIED GLOBAL POINTER   710
RULE                   GLOBAL PREDICATE         806 648
R                      GLOBAL PREDICATE         727 722 718 717 715 714 713 712 711 710 697 694 687 686 685 681 676
674 672 670 668 650 588 558 557 544 543 537 536 535 534 533 531 529 527 526 522 506 504 502 501 497 494 493 492 491
 489 366
SEMICOLON              APPLIED GLOBAL POINTER   755 749 735 718 694 681 677 640 637 606 537 527
SEMICOL                GLOBAL ACTION            705 694 682 666 657 640 635 632 599
SENTENCE               GLOBAL ACTION            809 788 786
SET                    MACRO ACTION             798 795 715 714 711 710 700 684 220 211 52
SHIFT2LINES            GLOBAL ACTION            650 537 527 506 497 135 100
SHORT                  APPLIED GLOBAL POINTER   712
SIGNAL                 GLOBAL ACTION            406 150 147
SKIPPED                APPLIED GLOBAL POINTER   726
SKIPUNTILPOINT         GLOBAL ACTION            807 725 708
SPACECARD              GLOBAL ACTION            194 193 162
SPACECODE              MACRO POINTER            258 256 195 189 139 20
SPACES                 GLOBAL ACTION            136 127 126 124 99
SPACE                  GLOBAL ACTION            138 129 117 112 111 109 99
SPECCHAR               GLOBAL PREDICATE         328 245
SPECFULL               APPLIED GLOBAL POINTER   331
SPECIFICATION          GLOBAL PREDICATE         801 484
SPECX                  GLOBAL POINTER           670 666 664 663 652 384
STACKBOLD              GLOBAL ACTION            310 309 308 306 304 302 299
STACKCONS              GLOBAL ACTION            346 345 341
```

STACKSPEC	GLOBAL ACTION	329 328 326
STACKTAG	GLOBAL ACTION	274 273 269 263
STARTINGSYMBOL	GLOBAL PREDICATE	805 721
STARTSYSTEM	GLOBAL ACTION	792 789 786
STRINGQUOTE	APPLIED GLOBAL POINTER	594 562
SUBTR	MACRO ACTION	397 124 55
SUB	APPLIED GLOBAL POINTER	717 533
TABCARD	GLOBAL ACTION	221 214 198 191 162
TABCODE	MACRO POINTER	260 198 142 137 20
TAB	GLOBAL ACTION	142 137 132 99
TABLINE	GLOBAL ACTION	643 639 221 204
TABS	GLOBAL ACTION	130 125 99
TAGFULL	APPLIED GLOBAL POINTER	296 276
TBOLD	GLOBAL LIST	323 316 311 169 104 34
TCONS	GLOBAL LIST	351 350 347 171 106 36
TERMINALS	GLOBAL ACTION	732 730 724 723 708
TEXTFULL	APPLIED GLOBAL POINTER	419
THREE	APPLIED GLOBAL POINTER	580
TIMES	APPLIED GLOBAL POINTER	668
TITLE	GLOBAL POINTER	799 794 787
TOEND	GLOBAL POINTER	679 621 618 384
TOLOST	GLOBAL POINTER	679 620 614 611 603 384
TONEXT	GLOBAL POINTER	688 687 684 625 620 615 609 607 604 602 384
TRACEOFF	APPLIED GLOBAL POINTER	715
TRACEON	APPLIED GLOBAL POINTER	714
TRACE	GLOBAL ACTION	405 404 148 147
TREATAFFIX	GLOBAL ACTION	560 558 557 547
TREATDECLLIST	GLOBAL ACTION	529 523 522 515
TREATSPECLIST	GLOBAL ACTION	499 495 489 483
TRUERESULT	GLOBAL ACTION	690 630 599

TRUE	MACRO POINTER	798 795 714 713 710 700 211 73
TRYSCRT	GLOBAL PREDICATE	551 550 456 448
TRYSTATE	GLOBAL PREDICATE	511 449 448 442
TRYTYPE	GLOBAL PREDICATE	723 704 701 698 462 448
TSPEC	GLOBAL LIST	334 333 330 170 105 35
TTAG	GLOBAL LIST	280 275 168 103 33
TWO	APPLIED GLOBAL POINTER	579
UNPACK2	MACRO ACTION	781 61
UNPACK3	MACRO ACTION	186 121 59
U	GLOBAL ACTION	643 639 531 215 204
WARN	GLOBAL ACTION	743 402 400 388
WASACTION	GLOBAL PREDICATE	653 634 631 552 471
WASAFFIX	GLOBAL PREDICATE	566 480 473
WASBOLD	MACRO FLAG	594 364 169 104 43
WASCONS	MACRO FLAG	592 440 433 365 171 106 45
WASDIGIT	MACRO FLAG	240 17
WASLETGIT	MACRO FLAG	238 16
WASLETTER	MACRO FLAG	236 15
WASMACRO	GLOBAL PREDICATE	567 480 477
WASPARAMLESSMACRO	GLOBAL PREDICATE	583 540 479
WASSPEECH	MACRO FLAG	246 18
WASSPEC	MACRO FLAG	170 105 44
WASTAG	MACRO FLAG	480 439 432 363 168 157 103 42
WASTERMINAL	GLOBAL PREDICATE	744 742
WRITEINT1	GLOBAL ACTION	179 177 176 162
WRITEINT	GLOBAL ACTION	190 174 173 172 171 162
WRITE	GLOBAL ACTION	208 167 162
WRCNGAFFIX	APPLIED GLOBAL POINTER	564
WRCNGINPUTCODE	APPLIED GLOBAL POINTER	376
WRCNGNUMBEROFPARAMETERS	APPLIED GLOBAL POINTER	470

XIMPOSSIBLE	APPLIED GLOBAL POINTER	445
XLOC	GLOBAL POINTER	797 764 409 88
XOUT	GLOBAL POINTER	793 787 734
XPARAMETERERROR	APPLIED GLOBAL POINTER	593 590
XREAD	GLOBAL POINTER	793 787 753 734

A12⁶2J.1⁹,KOSTER

```
'BEGIN'
        'COMMENT'4COMPILERCOMPILERDESCRIBEDINCDL***************************************;
'COMMENT'4.1GENERALENVIRONMENT;
'COMMENT'4.1.1INTERFACEWITHMACHINE;
'COMMENT'4.1.2STACKS;
'INTEGER''ARRAY'TTAG[100001:110000],TBOLD[200001:200600],TSPEC[300001:300100],TCONS[400001:400300],LLOC[500001:500200],LGLOB[600001:6010
00],LTEXT[800001:804000],LMACR[700001:701100];
'COMMENT'4.1.3MACROS:
'COMMENT'DATASTRUCTURESANDTHEIRACCESS;
'COMMENT'4.1.4POINTERSANDFLAGS;
'INTEGER'PTAG,PBOLD,PSPEC,PCONS,PLOC,PGLOB,PTEXT,PMACR,XLOC;
'BOOLEAN'GIVETEXT,GIVETRACE,LEGIBLE;
'INTEGER'FIRSTTAG;
'COMMENT'4.2OUTPUT;
'COMMENT'4.2.1PRINTERSECTION;
'INTEGER'POS;
'PROCEDURE'OUT(X);'INTEGER'X;'BEGIN''INTEGER'EL;'IF'¬(100001'LE'X^X<PTAG)'THEN''GOTO'1;PRINT(TTAG,X);'GOTO'999;1:'IF'¬(200001'LE'X^X<PBO
LD)'THEN''GOTO'2;PRINT(TBOLD,X);'GOTO'999;2:'IF'¬(300001'LE'X^X<PSPEC)'THEN''GOTO'3;PRINT(TSPEC,X);'GOTO'999;3:'IF'¬(400001'LE'X^X<PCONS)'THEN''
GOTO'4;EL:=TCONS[X]:OUTINT(EL);'GOTO'999;4:OUTINT(X);999:'END';
'PROCEDURE'OUTINT(X);'INTEGER'X;'BEGIN''INTEGER'QUOT,REM;SPACE;'IF'¬(X<0)'THEN''GOTO'1;REM:=X;REM:=-REM;PRCHAR(65);OUTINT(REM);'GOTO'999
;1:'IF'¬(X=0)'THEN''GOTO'2;PRCHAR(0);SPACE;'GOTO'999;2:QUOT:=X'/'10;REM:=X-10*QUOT;OUTINT1(QUOT);PRCHAR(REM);SPACE;999:'END';
'PROCEDURE'OUTINT1(X);'INTEGER'X;'BEGIN''INTEGER'QUOT,REM;'IF'¬(X=0)'THEN''GOTO'1;'GOTO'999;1:QUOT:=X'/'10;REM:=X-10*QUOT;OUTINT1(QUOT);
PRCHAR(REM);999:'END';
'PROCEDURE'PRINT(LIST,Y);'INTEGER'Y;'BEGIN''INTEGER'X,LX;X:=Y;SPACE;RST:LX:=LIST[X];'BEGIN''IF'¬(LX<0)'THEN''GOTO'2;LX:=-LX;PRINT1(LX);'
GOTO'999;2:PRINT1(LX);X:=X+1;'GOTO'RST;'END';999:'END';
'PROCEDURE'PRINT1(X);'INTEGER'X;'BEGIN''INTEGER'X1,X2,X3;X1:=X'/'16384;X3:=X-16384*X1;X2:=X3'/'128;X3:=X3-128*X2;PRCHAR(X1);PRCHAR(X2);P
RCHAR(X3);'END';
'PROCEDURE'POSITION(X);'INTEGER'X;'BEGIN''INTEGER'TBS,SPCES;SPCES:=X-POS;'IF'¬(0≤SPCES)'THEN''GOTO'1;SPACES(SPCES);'GOTO'999;1:NLCR;TBS:
=X'/'8;SPCES:=X-8*TBS:TABS(TBS);'BEGIN''IF'¬(SPCES=0)'THEN''GOTO'3;'GOTO'999;3:SPACES(SPCES);'GOTO'999;'END';999:'END';
'PROCEDURE'SPACES(X);'INTEGER'X;'BEGIN''INTEGER'N;N:=0;SPC:'IF'¬(N<X)'THEN''GOTO'1;SPACE;N:=N+1;'GOTO'SPC;1:'END';
'PROCEDURE'TABS(X);'INTEGER'X;'BEGIN''INTEGER'N;N:=0;TBS:'IF'¬(N<X)'THEN''GOTO'1;TAB;N:=N+1;'GOTO'TBS;1:'END';
'PROCEDURE'NLCR;'BEGIN'PRSYM(119);POS:=0;'END';
'PROCEDURE'SHIFT2LINES;'BEGIN''INTEGER'OLDPOS;OLDPOS:=POS;NLCR;NLCR;SPACES(OLDPOS);'END';
'PROCEDURE'TAB;'BEGIN'PRSYM(118);POS:=POS+8;'END';
'PROCEDURE'SPACE;'BEGIN'PRCHAR(93);'END';
'PROCEDURE'PRCHAR(X);'INTEGER'X;'BEGIN''IF'¬(X=119)'THEN''GOTO'1;NLCR;'GOTO'999;1:'IF'¬(X=118)'THEN''GOTO'2;TAB;'GOTO'999;2:'IF'¬(X=63)'
THEN''GOTO'3;'GOTO'999;3:PRSYM(X);POS:=POS+1;999:'END';
'COMMENT'4.2.2TRACEADMINISTRATION;
'PROCEDURE'TRACE(X);'INTEGER'X;'BEGIN''IF'¬GIVETRACE'THEN''GOTO'1;POSITION(32);INFORM(X);'GOTO'999;1:999:'END';
'PROCEDURE'SIGNAL(X);'INTEGER'X;'BEGIN'POSITION(16);INFORM(X);'END';
'PROCEDURE'INFORM(X);'INTEGER'X;'BEGIN''INTEGER'STATE,SORT,TYPE;STATE:=TTAG[X-5];SORT:=TTAG[X-4];TYPE:=TTAG[X-3];OUT(STATE);OUT(SORT);OU
T(TYPE);OUT(X);'END';
'PROCEDURE'ERROR(TEXT,INFO);'INTEGER'INFO,TEXT;'BEGIN''INTEGER'OLDPOS;OLDPOS:=POS;NLCR;OUT(TEXT);'BEGIN''IF'¬(100001'LE'INFO^INFO<PTAG)'
THEN''GOTO'2;INFORM(INFO);POSITION(OLDPOS);'GOTO'999;2:'IF'¬(INFO=0)'THEN''GOTO'3;POSITION(OLDPOS);'GOTO'999;3:OUT(INFO);POSITION(OLDPOS);'GOTO'
999;'END';999:'END';
'COMMENT'4.2.3CARDPUNCHSECTION;
'INTEGER'CARD,CPOS;
'PROCEDURE'WRITE(X);'INTEGER'X;'BEGIN''INTEGER'EL;'IF'¬(100001'LE'X^X<PTAG)'THEN''GOTO'1;PUNCH(TTAG,X);'GOTO'999;1:'IF'¬(200001'LE'X^X<P
BOLD)'THEN''GOTO'2;PUNCH(TBOLD,X);'GOTO'999;2:'IF'¬(300001'LE'X^X<PSPEC)'THEN''GOTO'3;PUNCH(TSPEC,X);'GOTO'999;3:'IF'¬(400001'LE'X^X<PCONS)'THEN
''GOTO'4;EL:=TCONS[X];WRITEINT(EL);'GOTO'999;4:WRITEINT(X);999:'END';
'PROCEDURE'WRITEINT(X);'INTEGER'X;'BEGIN''INTEGER'QUOT,REM;'IF'¬(X<0)'THEN''GOTO'1;REM:=X;REM:=-REM;PUCHAR(65);WRITEINT(REM);'GOTO'999;1
:'IF'¬(X=0)'THEN''GOTO'2;PUCHAR(0);'GOTO'999;2:QUOT:=X'/'10;REM:=X-10*QUOT;WRITEINT1(QUOT);PUCHAR(REM);999:'END';
'PROCEDURE'WRITEINT1(X);'INTEGER'X;'BEGIN''INTEGER'QUOT,REM;'IF'¬(X=0)'THEN''GOTO'1;'GOTO'999;1:QUOT:=X'/'10;REM:=X-10*QUOT;WRITEINT1(QU
OT);PUCHAR(REM);999:'END';
'PROCEDURE'PUNCH(LIST,Y);'INTEGER'Y,LIST;'BEGIN''INTEGER'X,LX;X:=Y;RST:LX:=LIST[X];'BEGIN''IF'¬(LX<0)'THEN''GOTO'2;LX:=-LX;PUNCH1(LX);'G
OTO'999;2:PUNCH1(LX);X:=X+1;'GOTO'RST;'END';999:'END';
```

```
'PROCEDURE'PUNCH1(X);'INTEGER'X;'BEGIN''INTEGER'X1,X2,X3;X1:=X'/'16384;X3:=X-16384*X1;X2:=X3'/'128;X3:=X3-128*X2;PUCHAR(X1);PUCHAR(X2);P
UCHAR(X3);'END';
'PROCEDURE'NEWCARD;'BEGIN'SPC:'IF'¬(CPOS≤72)'THEN''GOTO'1;CSYM(93);CPOS:=CPOS+1;'GOTO'SPC;1:CPOS:=1;WRITEINT(CARD);CSYM(119);CARD:=CARD+
1;CPOS:=1;'END';
'PROCEDURE'TABCARD;'BEGIN''INTEGER'SPCES,TBS;TBS:=CPOS'/'8;SPCES:=CPOS-8*TBS;SPC:'IF'¬(SPCES<8)'THEN''GOTO'1;SPACECARD;SPCES:=SPCES+1;'G
OTO'SPC;1:'END';
'PROCEDURE'SPACECARD;'BEGIN'PUCHAR(93);'END';
'PROCEDURE'PUCHAR(X);'INTEGER'X;'BEGIN''IF'¬(X=119)'THEN''GOTO'1;NEWCARD;'GOTO'999;1:'IF'¬(X=118)'THEN''GOTO'2;TABCARD;'GOTO'999;2:'IF'¬
(X=63)'THEN''GOTO'3;'GOTO'999;3:'IF'¬(CPOS<72)'THEN''GOTO'4;CSYM(X);CPOS:=CPOS+1;'GOTO'999;4:CSYM(133);CSYM(119);CPOS:=1;PUCHAR(X);999:'END';
'COMMENT'4.2.4RESULTADMINISTRATION;
'BOOLEAN'LINEDUP;
'INTEGER'INDENTATION;
'PROCEDURE'G(X);'INTEGER'X;'BEGIN'LINEUP;WRITE(X);'END';
'PROCEDURE'LINEUP;'BEGIN''IF'¬LINEDUP'THEN''GOTO'1;'GOTO'999;1:'IF'¬LEGIBLE'THEN''GOTO'2;PUTABS(INDENTATION);LINEDUP:='TRUE';'GOTO'999;2
:999:'END';
'PROCEDURE'PUTABS(N);'INTEGER'N;'BEGIN''INTEGER'N1;'IF'¬(N≤0)'THEN''GOTO'1;'GOTO'999;1:N1:=N;N1:=N1-1;TABCARD;PUTABS(N1);999:'END';
'PROCEDURE'U;'BEGIN'INDENTATION:=INDENTATION+1;'END';
'PROCEDURE'L;'BEGIN'INDENTATION:=INDENTATION-1;'END';
'PROCEDURE'NEWLINE;'BEGIN''IF'¬LEGIBLE'THEN''GOTO'1;'IF'¬LINEDUP'THEN''GOTO'1;BLANKLINE;'GOTO'999;1:999:'END';
'PROCEDURE'BLANKLINE;'BEGIN'NEWCARD;LINEDUP:='FALSE';'END';
'PROCEDURE'TABLINE;'BEGIN''IF'¬LEGIBLE'THEN''GOTO'1;TABCARD;'GOTO'999;1:999:'END';
'COMMENT'4.3INPUT;
'INTEGER'LINE,CHAR,INPT;
'COMMENT'4.3.1READINGCHARACTERS;
'BOOLEAN''PROCEDURE'LETTER(X);'INTEGER'X;'BEGIN''IF'¬(CHAR>9∧CHAR<36)'THEN''GOTO'1;X:=CHAR;NEXTNONSPACECHAR;LETTER:='TRUE';'GOTO'999;1:L
ETTER:='FALSE';999:'END';
'BOOLEAN''PROCEDURE'LETGIT(X);'INTEGER'X;'BEGIN''IF'¬(CHAR<36)'THEN''GOTO'1;X:=CHAR;NEXTNONSPACECHAR;LETGIT:='TRUE';'GOTO'999;1:LETGIT:=
'FALSE';999:'END';
'BOOLEAN''PROCEDURE'DIGIT(X);'INTEGER'X;'BEGIN''IF'¬(CHAR<10)'THEN''GOTO'1;X:=CHAR;NEXTNONSPACECHAR;DIGIT:='TRUE';'GOTO'999;1:DIGIT:='FA
LSE';999:'END';
'BOOLEAN''PROCEDURE'BOLDCHAR(X);'INTEGER'X;'BEGIN''IF'¬(CHAR=121)'THEN''GOTO'1;X:=120;NEXTCHAR;BOLDCHAR:='TRUE';'GOTO'999;1:X:=CHAR;NEXT
CHAR;BOLDCHAR:='TRUE';999:'END';
'BOOLEAN''PROCEDURE'ACCENT;'BEGIN''IF'¬(CHAR=120)'THEN''GOTO'1;NEXTCHAR;ACCENT:='TRUE';'GOTO'999;1:ACCENT:='FALSE';999:'END';
'BOOLEAN''PROCEDURE'SPECCHAR(X);'INTEGER'X;'BEGIN''IF'¬(CHAR>63)'THEN''GOTO'1;X:=CHAR;NEXTNONSPACECHAR;SPECCHAR:='TRUE';'GOTO'999;1:SPEC
CHAR:='FALSE';999:'END';
'PROCEDURE'NEXTCHAR;'BEGIN'DISPLAYCHARACTER;CHAR:=RESYM;'END';
'PROCEDURE'DISPLAYCHARACTER;'BEGIN''IF'¬(CHAR=119)'THEN''GOTO'1;LINE:=LINE+1;'BEGIN''IF'¬GIVETEXT'THEN''GOTO'2;POSITION(133);OUTINT1(CAR
D);POSITION(48);OUTINT1(LINE);POSITION(52);'GOTO'999;2:'GOTO'999;'END';1:'IF'¬GIVETEXT'THEN''GOTO'4;PRCHAR(CHAR);'GOTO'999;4:999:'END';
'PROCEDURE'NEXTNONSPACECHAR;'BEGIN'CHR:NEXTCHAR;'BEGIN''IF'¬(CHAR=93)'THEN''GOTO'2;'GOTO'CHR;2:'GOTO'999;'END';999:'END';
'BOOLEAN''PROCEDURE'LAYOUTSYMBOL;'BEGIN''IF'¬(CHAR=93)'THEN''GOTO'1;LAYOUTSYMBOL:='TRUE';'GOTO'999;1:'IF'¬(CHAR=119)'THEN''GOTO'2;LAYOUT
SYMBOL:='TRUE';'GOTO'999;2:'IF'¬(CHAR=118)'THEN''GOTO'3;LAYOUTSYMBOL:='TRUE';'GOTO'999;3:LAYOUTSYMBOL:='FALSE';999:'END';
'COMMENT'4.3.2TAGSYMBOLS;
'BOOLEAN''PROCEDURE'READTAG(X);'INTEGER'X;'BEGIN''INTEGER'T,T1,T2,T3,T4;'IF'¬LETTER(T1)'THEN''GOTO'1;X:=PTAG;NXT:'BEGIN''IF'¬LETGIT(T2)'
THEN''GOTO'2;'BEGIN''IF'¬LETGIT(T3)'THEN''GOTO'3;'BEGIN''IF'¬LETGIT(T4)'THEN''GOTO'4;T:=(T1*128+T2)*128+T3;STACKTAG(T);T1:=T4;'GOTO'NXT;4:T:=(T1
*128+T2)*128+T3;'GOTO'LST;'END';3:T:=(T1*128+T2)*128+63;'GOTO'LST;'END';2:T:=(T1*128+63)*128+63;LST:T:=-T;STACKTAG(T);READTAG:='TRUE';'GOTO'999;
'END';1:READTAG:='FALSE';999:'END';
'PROCEDURE'STACKTAG(X);'INTEGER'X;'BEGIN''IF'¬(PTAG<110000)'THEN''GOTO'1;TTAG[PTAG]:=X;PTAG:=PTAG+1;'GOTO'999;1:ERROR(TAGFULL,0);POSTMOR
TEM;EXIT;999:'END';
'PROCEDURE'ENTERTAG(X);'INTEGER'X;'BEGIN''INTEGER'Y,X1,Y1,WX,WY;Y:=FIRSTTAG;NXY:X1:=X;Y1:=Y;NXW:WX:=TTAG[X1];WY:=TTAG[Y1];'BEGIN''IF'¬(W
X=WY)'THEN''GOTO'2;'BEGIN''IF'¬(WX<0)'THEN''GOTO'3;'BEGIN''IF'¬(Y=X)'THEN''GOTO'4;RESERVEADMINSPACE;'GOTO'999;4:PTAG:=X;X:=Y;'GOTO'999;'END';3:X
1:=X1+1;Y1:=Y1+1;'GOTO'NXW;'END';2:'IF'¬(ABS(WX)<ABS(WY))'THEN''GOTO'7;WY:=TTAG[Y-2];'BEGIN''IF'¬(WY=0)'THEN''GOTO'8;TTAG[Y-2]:=X;RESERVEADMINSP
ACE;'GOTO'999;8:Y:=WY;'GOTO'NXY;'END';7:WY:=TTAG[Y-1];'BEGIN''IF'¬(WY=0)'THEN''GOTO'11;TTAG[Y-1]:=X;RESERVEADMINSPACE;'GOTO'999;11:Y:=WY;'GOTO'N
XY;'END','END';999:'END';
'PROCEDURE'RESERVEADMINSPACE;'BEGIN'PTAG:=PTAG+8;'IF'¬(PTAG<110000)'THEN''GOTO'1;TTAG[PTAG-8]:=999;TTAG[PTAG-7]:=0;TTAG[PTAG-6]:=0;TTAG[
PTAG-5]:=0;TTAG[PTAG-4]:=0;TTAG[PTAG-3]:=0;TTAG[PTAG-2]:=0;TTAG[PTAG-1]:=0;'GOTO'999;1:ERROR(TAGFULL,0);POSTMORTEM;EXIT;999:'END';
'COMMENT'4.3.3BOLDSYMBOLS;
'BOOLEAN''PROCEDURE'READBOLD(X);'INTEGER'X;'BEGIN''INTEGER'B1,B2,B3,D;'IF'¬ACCENT'THEN''GOTO'1;X:=PBOLD;'BEGIN''IF'¬ACCENT'THEN''GOTO'2;
D:=(63*128+63)*128+63;D:=-D;STACKBOLD(D);READBOLD:='TRUE';'GOTO'999;2:'IF'¬BOLDCHAR(B1)'THEN''GOTO'3;RST:'BEGIN''IF'¬ACCENT'THEN''GOTO'4;D:=(B1*
128+63)*128+63;D:=-D;STACKBOLD(D);READBOLD:='TRUE';'GOTO'999;4:'IF'¬BOLDCHAR(B2)'THEN''GOTO'5;'BEGIN''IF'¬ACCENT'THEN''GOTO'6;D:=(B1*128+B2)*128
+63;D:=-D;STACKBOLD(D);READBOLD:='TRUE';'GOTO'999;6:'IF'¬BOLDCHAR(B3)'THEN''GOTO'7;D:=(B1*128+B2)*128+B3;'BEGIN''IF'¬ACCENT'THEN''GOTO'8;D:=-D;S
```

```
TACKBOLD(D);READBOLD:='TRUE';'GOTO'999;8:'IF'¬BOLDCHAR(B1)'THEN''GOTO'9;STACKBOLD(D);'GOTO'RST;9:'GOTO'0;'END';7:'GOTO'0;'END';5:'GOTO'0;'END';3
:'GOTO'0;'END';1:0:READBOLD:='FALSE';999:'END';
76   'PROCEDURE'STACKBOLD(X);'INTEGER'X;'BEGIN''IF'¬(PBOLD≤200600)'THEN''GOTO'1;TBOLD[PBOLD]:=X;PBOLD:=PBOLD+1;'GOTO'999;1:ERROR(BOLDFULL,0);
POSTMORTEM;EXIT;999:'END';
77   'PROCEDURE'ENTERBOLD(X);'INTEGER'X;'BEGIN''INTEGER'Y,X1,Y1,WX,WY;Y:=200001;NXY:X1:=X;Y1:=Y;NXW:WX:=TBOLD[X1];WY:=TBOLD[Y1];'BEGIN''IF'¬(
WX=WY)'THEN''GOTO'2;'BEGIN''IF'¬(WX<0)'THEN''GOTO'3;'BEGIN''IF'¬(Y=X)'THEN''GOTO'4;'GOTO'999;4:PBOLD:=X;X:=Y;'GOTO'999;'END';3:X1:=X1+1;Y1:=Y1+1
;'GOTO'NXW;'END';2:SKP:'IF'¬(WY<0)'THEN''GOTO'7;Y:=Y1;Y:=Y+1;'GOTO'NXY;7:Y1:=Y1+1;WY:=TBOLD[Y1];'GOTO'SKP;'END';999:'END';
78   'COMMENT'4.3.4SPECIALSYMBOLS;
79   'BOOLEAN''PROCEDURE'READSPEC(X);'INTEGER'X;'BEGIN''INTEGER'S,S1;'IF'¬SPECCHAR(S1)'THEN''GOTO'1;X:=PSPEC;S:=(S1*128+63)*128+63;S:=-S;STAC
KSPEC(S);READSPEC:='TRUE';'GOTO'999;1:READSPEC:='FALSE';999:'END';
80   'PROCEDURE'STACKSPEC(X);'INTEGER'X;'BEGIN''IF'¬(PSPEC≤300100)'THEN''GOTO'1;TSPEC[PSPEC]:=X;PSPEC:=PSPEC+1;'GOTO'999;1:ERROR(SPECFULL,0);
POSTMORTEM;EXIT;999:'END';
81   'PROCEDURE'ENTERSPEC(X);'INTEGER'X;'BEGIN''INTEGER'Y,WX,WY;Y:=300001;WX:=TSPEC[X];NXY:WY:=TSPEC[Y];'BEGIN''IF'¬(WX=WY)'THEN''GOTO'2;'BEG
IN''IF'¬(Y=X)'THEN''GOTO'3;'GOTO'999;3:PSPEC:=X;X:=Y;'GOTO'999;'END';2:Y:=Y+1;'GOTO'NXY;'END';999:'END';
82   'COMMENT'4.3.5CONSTANTS;
83   'BOOLEAN''PROCEDURE'READCONS(X);'INTEGER'X;'BEGIN''INTEGER'D,S;'IF'¬DIGIT(D)'THEN''GOTO'1;S:=D;RST:'BEGIN''IF'¬DIGIT(D)'THEN''GOTO'2;S:=
S*10+D;'GOTO'RST;2:X:=PCONS;STACKCONS(S);READCONS:='TRUE';'GOTO'999;'END';1:READCONS:='FALSE';999:'END';
84   'PROCEDURE'STACKCONS(X);'INTEGER'X;'BEGIN''IF'¬(PCONS≤400300)'THEN''GOTO'1;TCONS[PCONS]:=X;PCONS:=PCONS+1;'GOTO'999;1:ERROR(CONSFULL,0);
POSTMORTEM;EXIT;999:'END';
85   'PROCEDURE'ENTERCONS(X);'INTEGER'X;'BEGIN''INTEGER'Y,S,T;S:=TCONS[X];Y:=400001;NXT:T:=TCONS[Y];'BEGIN''IF'¬(S=T)'THEN''GOTO'2;'BEGIN''IF
'¬(X=Y)'THEN''GOTO'3;'GOTO'999;3:PCONS:=X;X:=Y;'GOTO'999;'END';2:Y:=Y+1;'GOTO'NXT;'END';999:'END';
86   'COMMENT'4.3.6INPUTADMINISTRATION;
87   'PROCEDURE'INITIALIZEFORREADING;'BEGIN'GIVETEXT:='FALSE';PCONS:=400001;PTAG:=100001;PBOLD:=200001;PSPEC:=300001;LINE:=0;CHAR:=RESYM;RESE
RVEADMINSPACE;FIRSTTAG:=PTAG;'END';
88   'BOOLEAN''PROCEDURE'ISTAG(X);'INTEGER'X;'BEGIN''IF'¬(100001'LE'INPT∧INPT<PTAG)'THEN''GOTO'1;X:=INPT;NEXTSYMBOL;ISTAG:='TRUE';'GOTO'999;1
:ISTAG:='FALSE';999:'END';
89   'BOOLEAN''PROCEDURE'ISBOLD(X);'INTEGER'X;'BEGIN''IF'¬(200001'LE'INPT∧INPT<PBOLD)'THEN''GOTO'1;X:=INPT;NEXTSYMBOL;ISBOLD:='TRUE';'GOTO'99
9;1:ISBOLD:='FALSE';999:'END';
90   'BOOLEAN''PROCEDURE'ISCONS(X);'INTEGER'X;'BEGIN''IF'¬(400001'LE'INPT∧INPT<PCONS)'THEN''GOTO'1;X:=INPT;NEXTSYMBOL;ISCONS:='TRUE';'GOTO'99
9;1:ISCONS:='FALSE';999:'END';
91   'BOOLEAN''PROCEDURE'R(X);'INTEGER'X;'BEGIN''IF'¬AHEAD(X)'THEN''GOTO'1:NEXTSYMBOL;R:='TRUE';'GOTO'999;1:R:='FALSE';999:'END';
92   'BOOLEAN''PROCEDURE'AHEAD(X);'INTEGER'X;'BEGIN''IF'¬(INPT=X)'THEN''GOTO'1;AHEAD:='TRUE';'GOTO'999;1:AHEAD:='FALSE';999:'END';
93   'PROCEDURE'NEXTSYMBOL;'BEGIN'READ(INPT);'END';
94   'PROCEDURE'READ(X);'INTEGER'X;'BEGIN'SKP:'IF'¬LAYOUTSYMBOL'THEN''GOTO'1;NEXTCHAR;'GOTO'SKP;1:'IF'¬READTAG(X)'THEN''GOTO'2;ENTERTAG(X);'G
OTO'999;2:'IF'¬READBOLD(X)'THEN''GOTO'3;ENTERBOLD(X);'GOTO'999;3:'IF'¬READSPEC(X)'THEN''GOTO'4;ENTERSPEC(X);'GOTO'999;4:'IF'¬READCONS(X)'THEN''G
OTO'5;ENTERCONS(X);'GOTO'999;5:ERROR(WRONGINPUTCODE,CHAR);NEXTCHAR;READ(X);999:'END';
95   'COMMENT'4.4COMPILERCOMPILERTCALGOL60;
96   'BOOLEAN'GIVERSTO,ADVANCED;
97   'INTEGER'HANDLE,BOUNDX,SPECX,TONEXT,TOLOST,TOEND,CURLAB,NEWLAB;
98   'COMMENT'4.4.1AUXILIARYACTIONS;
99   'PROCEDURE'ADDPLACE(TAG);'INTEGER'TAG;'BEGIN''INTEGER'H,K,Q;H:=TTAG[TAG-6];'BEGIN''IF'¬(H=0)'THEN''GOTO'2;REPLACE(TAG,H);'GOTO'999;2:Q:=
LTEXT[H];K:=Q-Q'/'8192*8192;'IF'¬(K<LINE)'THEN''GOTO'3;REPLACE(TAG,H);'GOTO'999;3:'GOTO'999;'END';999:'END';
100  'PROCEDURE'REPLACE(TAG,H);'INTEGER'H,TAG;'BEGIN'TTAG[TAG-6]:=PTEXT;'BEGIN''IF'¬(H=0)'THEN''GOTO'2;ENTERTEXT(LINE);'GOTO'999;2:H:=H-80000
1;H:=H+1;H:=8192*H+LINE;ENTERTEXT(H);'GOTO'999;'END';999:'END';
101  'PROCEDURE'FORGETLOCALS;'BEGIN''INTEGER'LOC;RST:'IF'¬(PLOC=500001)'THEN''GOTO'1;'GOTO'999;1:PLOC:=PLOC-1;LOC:=LLOC[PLOC];WARN(LOC);TTAG[
LOC-5]:=0;TTAG[LOC-4]:=0;TTAG[LOC-3]:=0;'GOTO'RST;999:'END';
102  'PROCEDURE'WARN(X);'INTEGER'X;'BEGIN''INTEGER'STATE;STATE:=TTAG[X-5];'BEGIN''IF'¬(STATE=BLANK)'THEN''GOTO'2;TRACE(X);'GOTO'999;2:'IF'¬(S
TATE=0)'THEN''GOTO'3;TRACE(X);'GOTO'999;3:SIGNAL(X);'GOTO'999;'END';999:'END';
103  'PROCEDURE'ENTERLOCAL(HEAD);'INTEGER'HEAD;'BEGIN''IF'¬(PLOC≤500200)'THEN''GOTO'1;LLOC[PLOC]:=HEAD;PLOC:=PLOC+1;'BEGIN''IF'¬(PLOC≤XLOC)'T
HEN''GOTO'2;'GOTO'999;2:XLOC:=PLOC;'GOTO'999;'END';1:ERROR(LOCFULL,0);POSTMORTEM;EXIT;999:'END';
104  'PROCEDURE'ENTERGLOBAL(HEAD);'INTEGER'HEAD;'BEGIN''IF'¬(PGLOB≤601000)'THEN''GOTO'1;LGLOB[PGLOB]:=HEAD;PGLOB:=PGLOB+1;'GOTO'999;1:ERROR(G
LOBFULL,0);POSTMORTEM;EXIT;999:'END';
105  'PROCEDURE'ENTERMACRO(HEAD);'INTEGER'HEAD;'BEGIN''IF'¬(PMACR≤701100)'THEN''GOTO'1;LMACR[PMACR]:=HEAD;PMACR:=PMACR+1;'GOTO'999;1:ERROR(MA
CRFULL,0);POSTMORTEM;EXIT;999:'END';
106  'PROCEDURE'ENTERTEXT(X);'INTEGER'X;'BEGIN''IF'¬(PTEXT≤804000)'THEN''GOTO'1;LTEXT[PTEXT]:=X;PTEXT:=PTEXT+1;'GOTO'999;1:ERROR(TEXTFULL,0);
POSTMORTEM;EXIT;999:'END';
107  'COMMENT'4.4.2TREATMENTOFTYPES;
108  'PROCEDURE'DEFINEACTION(HEAD);'INTEGER'HEAD;'BEGIN'REDEFINE(HEAD,DEFINED,GLOBAL,ACTION);'END';
109  'PROCEDURE'DEFINEPREDICATE(HEAD);'INTEGER'HEAD;'BEGIN'REDEFINE(HEAD,DEFINED,GLOBAL,PREDICATE);'END';
110  'PROCEDURE'DEFINEAFFIX(HEAD);'INTEGER'HEAD;'BEGIN'REDEFINE(HEAD,DEFINED,LOCAL,POINTER);'END';
```

```
  111  'PROCEDURE'DEFINELABEL(LAB);'INTEGER'LAB;'BEGIN''IF'¬(100001'LE'LAB^LAB<PTAG)'THEN''GOTO'1;REDEFINE(LAB,DEFINED,LOCAL,LABEL);'GOTO'999;1
:'IF'¬(400001'LE'LAB^LAB<PCONS)'THEN''GOTO'2;'GOTO'999;2:999:'END';
  112  'PROCEDURE'APPLYACTION(HEAD);'INTEGER'HEAD;'BEGIN'REDEFINE(HEAD,APPLIED,GLOBAL,ACTION);'END';
  113  'PROCEDURE'APPLYPREDICATE(HEAD);'INTEGER'HEAD;'BEGIN'REDEFINE(HEAD,APPLIED,GLOBAL,PREDICATE);'END';
  114  'PROCEDURE'APPLYLABEL(LAB);'INTEGER'LAB;'BEGIN''IF'¬(100001'LE'LAB^LAB<PTAG)'THEN''GOTO'1;REDEFINE(LAB,APPLIED,LOCAL,LABEL);'GOTO'999;1:
'IF'¬(400001'LE'LAB^LAB<PCONS)'THEN''GOTO'2;'GOTO'999;2:999:'END';
  115  'PROCEDURE'APPLY(X);'INTEGER'X;'BEGIN'NEWPLACE(X);'IF'¬TRYSTATE(X,APPLIED)'THEN''GOTO'1;'GOTO'999;1:999:'END';
  116  'PROCEDURE'REDEFINE(X,NSTATE,NSORT,NTYPE);'INTEGER'NTYPE,NSORT,NSTATE,X;'BEGIN''INTEGER'OLDPOS;'IF'¬REDEFINE1(X,NSTATE,NSORT,NTYPE)'THEN
''GOTO'1;NEWPLACE(X);'GOTO'999;1:OLDPOS:=POS;NLCR;OUT(XIMPOSSIBLE);INFORM(X);OUT(NSTATE);OUT(NSORT);OUT(NTYPE);POSITION(OLDPOS);999:'END';
  117  'BOOLEAN''PROCEDURE'REDEFINE1(X,NSTATE,NSORT,NTYPE);'INTEGER'NTYPE,NSORT,NSTATE,X;'BEGIN''IF'¬TRYTYPE(X,NTYPE)'THEN''GOTO'1;'IF'¬TRYSORT
(X,NSORT)'THEN''GOTO'0;'IF'¬TRYSTATE(X,NSTATE)'THEN''GOTO'0;REDEFINE1:='TRUE';'GOTO'999;1:0:REDEFINE1:='FALSE';999:'END';
  118  'BOOLEAN''PROCEDURE'TRYSTATE(X,S);'INTEGER'S,X;'BEGIN''INTEGER'Q;Q:=TTAG[X-5];'IF'¬(Q=0)'THEN''GOTO'1;TTAG[X-5]:=S;TRYSTATE:='TRUE';'GOT
O'999;1:'IF'¬(Q=DEFINED)'THEN''GOTO'2;'IF'¬(S=APPLIED)'THEN''GOTO'2;TTAG[X-5]:=BLANK;TRYSTATE:='TRUE';'GOTO'999;2:'IF'¬(Q=APPLIED)'THEN''GOTO'3;
'BEGIN''IF'¬(S=APPLIED)'THEN''GOTO'4;TRYSTATE:='TRUE';'GOTO'999;4:'IF'¬(S=DEFINED)'THEN''GOTO'5;TTAG[X-5]:=BLANK;TRYSTATE:='TRUE';'GOTO'999;5:'G
OTO'0;'END';3:'IF'¬(Q=BLANK)'THEN''GOTO'7;'IF'¬(S=APPLIED)'THEN''GOTO'7;TRYSTATE:='TRUE';'GOTO'999;7:0:TRYSTATE:='FALSE';999:'END';
  119  'BOOLEAN''PROCEDURE'TRYSORT(X,P);'INTEGER'P,X;'BEGIN''INTEGER'Q;Q:=TTAG[X-4];'IF'¬(Q=0)'THEN''GOTO'1;TTAG[X-4]:=P;'BEGIN''IF'¬(P=LOCAL)'
THEN''GOTO'2;ENTERLOCAL(X);TRYSORT:='TRUE';'GOTO'999;2:'IF'¬(P=GLOBAL)'THEN''GOTO'3;ENTERGLOBAL(X);TRYSORT:='TRUE';'GOTO'999;3:'IF'¬(P=MACRO)'TH
EN''GOTO'4;ENTERGLOBAL(X);TRYSORT:='TRUE';'GOTO'999;4:'GOTO'0;'END';1:'IF'¬(P=Q)'THEN''GOTO'6;TRYSORT:='TRUE';'GOTO'999;6:0:TRYSORT:='FALSE';999
:'END';
  120  'BOOLEAN''PROCEDURE'TRYTYPE(X,P);'INTEGER'P,X;'BEGIN''INTEGER'Q;Q:=TTAG[X-3];'IF'¬(Q=0)'THEN''GOTO'1;TTAG[X-3]:=P;TRYTYPE:='TRUE';'GOTO'
999;1:'IF'¬(P=Q)'THEN''GOTO'2;TRYTYPE:='TRUE';'GOTO'999;2:TRYTYPE:='FALSE';999:'END';
  121  'PROCEDURE'NEWPLACE(X);'INTEGER'X;'BEGIN''INTEGER'SORT;SORT:=TTAG[X-4];'IF'¬(SORT=LOCAL)'THEN''GOTO'1;'GOTO'999;1:ADDPLACE(X);999:'END';
  122  'PROCEDURE'NEWNPARS(X,P);'INTEGER'P,X;'BEGIN''INTEGER'Q;Q:=TTAG[X-8];'IF'¬(Q=P)'THEN''GOTO'1;'GOTO'999;1:'IF'¬(Q=999)'THEN''GOTO'2;TTAG[
X-8]:=P;'GOTO'999;2:ERROR(WRONGNUMBEROFPARAMETERS,X);999:'END';
  123  'BOOLEAN''PROCEDURE'WASACTION(HEAD);'INTEGER'HEAD;'BEGIN''INTEGER'T;T:=TTAG[HEAD-3];'IF'¬(T=ACTION)'THEN''GOTO'1;WASACTION:='TRUE';'GOTO
'999;1:WASACTION:='FALSE';999:'END';
  124  'BOOLEAN''PROCEDURE'WASAFFIX(HEAD);'INTEGER'HEAD;'BEGIN''INTEGER'T;T:=TTAG[HEAD-3];'IF'¬(T=POINTER)'THEN''GOTO'1;WASAFFIX:='TRUE';'GOTO'
999;1:'IF'¬(T=FLAG)'THEN''GOTO'2;WASAFFIX:='TRUE';'GOTO'999;2:'IF'¬(T=LIST)'THEN''GOTO'3;WASAFFIX:='TRUE';'GOTO'999;3:WASAFFIX:='FALSE';999:'END
';
  125  'BOOLEAN''PROCEDURE'WASMACRO(HEAD);'INTEGER'HEAD;'BEGIN''INTEGER'S;S:=TTAG[HEAD-4];'IF'¬(S=MACRO)'THEN''GOTO'1;WASMACRO:='TRUE';'GOTO'99
9;1:WASMACRO:='FALSE';999:'END';
  126  'BOOLEAN''PROCEDURE'WASPARAMLESSMACRO(HEAD);'INTEGER'HEAD;'BEGIN''IF'¬(100001'LE'HEAD^HEAD<PTAG)'THEN''GOTO'1;'IF'¬WASMACRO(HEAD)'THEN''
GOTO'1;'IF'¬WASAFFIX(HEAD)'THEN''GOTO'0;WASPARAMLESSMACRO:='TRUE';'GOTO'999;1:0:WASPARAMLESSMACRO:='FALSE';999:'END';
  127  'COMMENT'4.4.3SPECIFICATIONS;
  128  'BOOLEAN''PROCEDURE'SPECIFICATION;'BEGIN''IF'¬EXTERNALSPECIFICATION'THEN''GOTO'1;SPECIFICATION:='TRUE';'GOTO'999;1:'IF'¬INTERNALSPECIFIC
ATION'THEN''GOTO'2;SPECIFICATION:='TRUE';'GOTO'999;2:'IF'¬MACROSPECIFICATION'THEN''GOTO'3;SPECIFICATION:='TRUE';'GOTO'999;3:SPECIFICATION:='FALS
E';999:'END';
  129  'BOOLEAN''PROCEDURE'EXTERNALSPECIFICATION;'BEGIN''INTEGER'TYPE;'IF'¬R(EXTERNAL)'THEN''GOTO'1;'IF'¬ISTYPE(TYPE)'THEN''GOTO'0;TREATSPECLIS
T(DEFINED,GLOBAL,TYPE);EXTERNALSPECIFICATION:='TRUE';'GOTO'999;1:0:EXTERNALSPECIFICATION:='FALSE';999:'END';
  130  'BOOLEAN''PROCEDURE'ISTYPE(X);'INTEGER'X;'BEGIN'X:=INPT;'IF'¬R(ACTION)'THEN''GOTO'1;ISTYPE:='TRUE';'GOTO'999;1:'IF'¬R(PREDICATE)'THEN''G
OTO'2;ISTYPE:='TRUE';'GOTO'999;2:'IF'¬R(POINTER)'THEN''GOTO'3;ISTYPE:='TRUE';'GOTO'999;3:'IF'¬R(FLAG)'THEN''GOTO'4;ISTYPE:='TRUE';'GOTO'999;4:IS
TYPE:='FALSE';999:'END';
  131  'PROCEDURE'TREATSPECLIST(STATE,SORT,TYPE);'INTEGER'TYPE,SORT,STATE;'BEGIN''INTEGER'HEAD;NXT:'IF'¬ISTAG(HEAD)'THEN''GOTO'1;REDEFINE(HEAD,
STATE,SORT,TYPE);'BEGIN''IF'¬R(COMMA)'THEN''GOTO'2;'GOTO'NXT;2:SHIFT2LINES;'IF'¬R(POINT)'THEN''GOTO'3;'GOTO'999;3:'GOTO'0;'END';1:0:999:'END';
  132  'BOOLEAN''PROCEDURE'INTERNALSPECIFICATION;'BEGIN''INTEGER'TYPE;'IF'¬ISTYPEOFLOCAL(TYPE)'THEN''GOTO'1;TREATSPECLIST(APPLIED,GLOBAL,TYPE);
INTERNALSPECIFICATION:='TRUE';'GOTO'999;1:INTERNALSPECIFICATION:='FALSE';999:'END';
  133  'BOOLEAN''PROCEDURE'ISTYPEOFLOCAL(X);'INTEGER'X;'BEGIN'X:=INPT;'IF'¬R(ACTION)'THEN''GOTO'1;ISTYPEOFLOCAL:='TRUE';'GOTO'999;1:'IF'¬R(PRED
ICATE)'THEN''GOTO'2;ISTYPEOFLOCAL:='TRUE';'GOTO'999;2:ISTYPEOFLOCAL:='FALSE';999:'END';
  134  'BOOLEAN''PROCEDURE'MACROSPECIFICATION;'BEGIN''INTEGER'TYPE,HEAD;'IF'¬R(MACRO)'THEN''GOTO'1;'IF'¬ISTYPE(TYPE)'THEN''GOTO'0;NXT:'IF'¬ISTA
G(HEAD)'THEN''GOTO'0;REDEFINE(HEAD,DEFINED,MACRO,TYPE);READMACRO(HEAD);'BEGIN''IF'¬R(COMMA)'THEN''GOTO'2;'GOTO'NXT;2:SHIFT2LINES;'IF'¬R(POINT)'T
HEN''GOTO'3;MACROSPECIFICATION:='TRUE';'GOTO'999;3:'GOTO'0;'END';1:0:MACROSPECIFICATION:='FALSE';999:'END';
  135  'PROCEDURE'READMACRO(HEAD);'INTEGER'HEAD;'BEGIN''INTEGER'X;TTAG[HEAD-7]:=PMACR;NXT:'BEGIN''IF'¬AHEAD(POINT)'THEN''GOTO'2;ENTERMACRO(POIN
T);'GOTO'999;2:'IF'¬AHEAD(COMMA)'THEN''GOTO'3;ENTERMACRO(POINT);'GOTO'999;3:'IF'¬ISTAG(X)'THEN''GOTO'4;'IF'¬TRYSTATE(X,APPLIED)'THEN''GOTO'0;ENT
ERMACRO(X);'GOTO'NXT;4:ENTERMACRO(INPT);NEXTSYMBOL;'GOTO'NXT;'END';0:999:'END';
  136  'COMMENT'4.4.4DECLARATIONS;
  137  'BOOLEAN''PROCEDURE'DECLARATION;'BEGIN''IF'¬POINTERDECLARATION'THEN''GOTO'1;DECLARATION:='TRUE';'GOTO'999;1:'IF'¬FLAGDECLARATION'THEN''G
OTO'2;DECLARATION:='TRUE';'GOTO'999;2:'IF'¬LISTDECLARATION'THEN''GOTO'3;DECLARATION:='TRUE';'GOTO'999;3:DECLARATION:='FALSE';999:'END';
  138  'BOOLEAN''PROCEDURE'POINTERDECLARATION;'BEGIN''IF'¬R(POINTER)'THEN''GOTO'1;TREATDECLLIST(QINTEGER,POINTER);POINTERDECLARATION:='TRUE';'G
OTO'999;1:POINTERDECLARATION:='FALSE';999:'END';
  139  'PROCEDURE'TREATDECLLIST(ALGOLTYPE,TYPE);'INTEGER'TYPE,ALGOLTYPE;'BEGIN''INTEGER'HEAD;BLANKLINE;G(ALGOLTYPE);NXT:'IF'¬ISTAG(HEAD)'THEN''
```

```
GOTO'1;REDEFINE(HEAD,DEFINED,GLOBAL,TYPE);G(HEAD);'BEGIN''IF'¬R(COMMA)'THEN''GOTO'2;G(COMMA);'GOTO'NXT;2:SHIFT2LINES;'IF'¬R(POINT)'THEN''GOTO'3;
G(SEMICOLON);'GOTO'999;3:'GOTO'0;'END';1:0:999:'END';
  140    'BOOLEAN''PROCEDURE'FLAGDECLARATION;'BEGIN''IF'¬R(FLAG)'THEN''GOTO'1;TREATDECLLIST(QBOOLEAN,FLAG);FLAGDECLARATION:='TRUE';'GOTO'999;1:FL
AGDECLARATION:='FALSE';999:'END';
  141    'BOOLEAN''PROCEDURE'LISTDECLARATION;'BEGIN''INTEGER'HEAD;'IF'¬R(LIST)'THEN''GOTO'1;BLANKLINE;G(QINTEGER);G(QARRAY);U;NXT:'IF'¬ISTAG(HEAD
)'THEN''GOTO'0;G(HEAD);REDEFINE(HEAD,DEFINED,GLOBAL,LIST);'IF'¬R(SUB)'THEN''GOTO'0;G(SUB);CONSTANTTEXT;'IF'¬R(COLON)'THEN''GOTO'0;G(COLON);CONST
ANTTEXT;'IF'¬R(BUS)'THEN''GOTO'0;G(BUS);'BEGIN''IF'¬R(COMMA)'THEN''GOTO'2;G(COMMA);NEWLINE;'GOTO'NXT;2:SHIFT2LINES;'IF'¬R(POINT)'THEN''GOTO'3;G(
SEMICOLON);L;LISTDECLARATION:='TRUE';'GOTO'999;3:'GOTO'0;'END';1:0:LISTDECLARATION:='FALSE';999:'END';
  142    'PROCEDURE'CONSTANTTEXT;'BEGIN''INTEGER'SYMB;NXT:'IF'¬ISTAG(SYMB)'THEN''GOTO'1;APPLY(SYMB);'BEGIN''IF'¬WASPARAMLESSMACRO(SYMB)'THEN''GOT
O'2;PUTPARAMLESSMACRO(SYMB);'GOTO'NXT;2:G(SYMB);'GOTO'NXT;'END';1:'IF'¬ISCONS(SYMB)'THEN''GOTO'4;G(SYMB);'GOTO'NXT;4:'IF'¬R(PLUS)'THEN''GOTO'5;G
(PLUS);'GOTO'NXT;5:'IF'¬R(MINUS)'THEN''GOTO'6;G(MINUS);'GOTO'NXT;6:'END';
  143    'COMMENT'4.4.5AFFIXEXPRESSIONS;
  144    'PROCEDURE'PUTAFFIXEXPRESSION(HEAD);'INTEGER'HEAD;'BEGIN''IF'¬TRYSORT(HEAD,GLOBAL)'THEN''GOTO'1;PUTDIRECT(HEAD);'GOTO'999;1:'IF'¬TRYSORT
(HEAD,MACRO)'THEN''GOTO'2;'BEGIN''IF'¬WASACTION(HEAD)'THEN''GOTO'3;PUTMACRO(HEAD);'GOTO'999;3:G(OPEN);PUTMACRO(HEAD);G(CLOSE);'GOTO'999;'END';2:
999:'END';
  145    'PROCEDURE'PUTDIRECT(HEAD);'INTEGER'HEAD;'BEGIN''INTEGER'NPARS;NPARS:=0;G(HEAD);AFFIXPACKOPTION(NPARS);NEWNPARS(HEAD,NPARS);'END';
  146    'PROCEDURE'AFFIXPACKOPTION(NPARS);'INTEGER'NPARS;'BEGIN''IF'¬R(PLUS)'THEN''GOTO'1;G(OPEN);TREATAFFIX;RST:'BEGIN'NPARS:=NPARS+1;'IF'¬R(PL
US)'THEN''GOTO'2;G(COMMA);TREATAFFIX;'GOTO'RST;2:G(CLOSE);'GOTO'999;'END';1:999:'END';
  147    'PROCEDURE'TREATAFFIX;'BEGIN''INTEGER'AFFX;'IF'¬ISTAG(AFFX)'THEN''GOTO'1;PUTSINGLE(AFFX);APPLY(AFFX);'GOTO'999;1:'IF'¬ISBOLD(AFFX)'THEN'
'GOTO'2;G(STRINGQUOTE);G(AFFX);G(STRINGQUOTE);'GOTO'999;2:'IF'¬ISCONS(AFFX)'THEN''GOTO'3;G(AFFX);'GOTO'999;3:ERROR(WRONGAFFIX,AFFX);999:'END';
  148    'PROCEDURE'PUTSINGLE(X);'INTEGER'X;'BEGIN''IF'¬WASAFFIX(X)'THEN''GOTO'1;'BEGIN''IF'¬WASMACRO(X)'THEN''GOTO'2;PUTPARAMLESSMACRO(X);'GOTO'
999;2:G(X);'GOTO'999;'END';1:REDEFINE(X,APPLIED,GLOBAL,POINTER);G(X);999:'END';
  149    'PROCEDURE'PUTPARAMLESSMACRO(Y);'INTEGER'Y;'BEGIN''INTEGER'X,M;X:=TTAG[Y-7];NEWNPARS(Y,0);NXT:X:=X+1;M:=LMACR[X];'BEGIN''IF'¬(M=POINT)'T
HEN''GOTO'2;'GOTO'999;2:G(M);'GOTO'NXT;'END';999:'END';
  150    'PROCEDURE'PUTMACRO(X);'INTEGER'X;'BEGIN''INTEGER'P1,P2,P3,P4,P5,Q,M,NPARS;NPARS:=0;Q:=TTAG[X-7];GETPARAMETERS(NPARS,P1,P2,P3,P4,P5);NEW
NPARS(X,NPARS);NXT:Q:=Q+1;M:=LMACR[Q];'BEGIN''IF'¬(M=POINT)'THEN''GOTO'2;'GOTO'999;2:'IF'¬(M=ONE)'THEN''GOTO'3;PUTPAR(P1,1);'GOTO'NXT;3:'IF'¬(M=
TWO)'THEN''GOTO'4;PUTPAR(P2,2);'GOTO'NXT;4:'IF'¬(M=THREE)'THEN''GOTO'5;PUTPAR(P3,3);'GOTO'NXT;5:'IF'¬(M=FOUR)'THEN''GOTO'6;PUTPAR(P4,4);'GOTO'NX
T;6:'IF'¬(M=FIVE)'THEN''GOTO'7;PUTPAR(P5,5);'GOTO'NXT;7:'IF'¬WASPARAMLESSMACRO(M)'THEN''GOTO'8;PUTPARAMLESSMACRO(M);'GOTO'NXT;8:G(M);'GOTO'NXT;'
END';999:'END';
  151    'PROCEDURE'GETPARAMETERS(P,P1,P2,P3,P4,P5);'INTEGER'P5,P4,P3,P2,P1,P;'BEGIN'GETPAR(P,P1);GETPAR(P,P2);GETPAR(P,P3);GETPAR(P,P4);GETPAR(P
,P5);'END';
  152    'PROCEDURE'GETPAR(P,X);'INTEGER'X,P;'BEGIN''IF'¬R(PLUS)'THEN''GOTO'1;P:=P+1;'BEGIN''IF'¬ISTAG(X)'THEN''GOTO'2;APPLY(X);'GOTO'999;2:'IF'¬
ISCONS(X)'THEN''GOTO'3;'GOTO'999;3:'GOTO'0;'END';1:X:=XPARAMETERERROR;'GOTO'999;0:999:'END';
  153    'PROCEDURE'PUTPAR(X,Y);'INTEGER'Y,X;'BEGIN''IF'¬(400001'LE'X^X<PCONS)'THEN''GOTO'1;G(X);'GOTO'999;1:'IF'¬(X=XPARAMETERERROR)'THEN''GOTO'
2;ERROR(X,Y);'GOTO'999;2:'IF'¬(200001'LE'X^X<PBOLD)'THEN''GOTO'3;G(STRINGQUOTE);G(X);G(STRINGQUOTE);'GOTO'999;3:PUTSINGLE(X);999:'END';
  154    'COMMENT'4.4.6BUILDINGSTONESOFARULE;
  155    'PROCEDURE'PUTCONNECTAND;'BEGIN''IF'¬GIVERSTO'THEN''GOTO'1;PUTJUMP(CURLAB);TONEXT:=TONEXT+1;'GOTO'999;1:'IF'¬ADVANCED'THEN''GOTO'2;ERROR
(POSSIBLYBACKTRACKNECESSARY,0);PUTJUMP(0);TOLOST:=TOLOST+1;'GOTO'999;2:PUTJUMP(CURLAB);TONEXT:=TONEXT+1;999:'END';
  156    'PROCEDURE'PUTCONNECTOR;'BEGIN''IF'¬AHEAD(SEMICOLON)'THEN''GOTO'1;PUTRESTORE;FRESHLABEL;'BEGIN''IF'¬(TONEXT=0)'THEN''GOTO'2;ERROR(ALTERN
ATIVENEVERREACHED,0);'GOTO'999;2:'GOTO'999;'END';1:'IF'¬AHEAD(CLOSE)'THEN''GOTO'4;'BEGIN''IF'¬(TONEXT=0)'THEN''GOTO'5;'GOTO'999;5:PUTLABEL(CURLA
B);FRESHLABEL;'BEGIN''IF'¬GIVERSTO'THEN''GOTO'7;'GOTO'999;7:PUTJUMP(0);TOLOST:=TOLOST+1;'GOTO'999;'END';'END';4:PUTRESTORE;LOSTLABEL;999:'END';
  157    'PROCEDURE'LOSTLABEL;'BEGIN''IF'¬(TOLOST=0)'THEN''GOTO'1;'BEGIN''IF'¬(TONEXT=0)'THEN''GOTO'2;'GOTO'999;2:FALSERESULT;'GOTO'999;'END';1:P
UTLABEL(0);FALSERESULT;999:'END';
  158    'PROCEDURE'ENDLABEL;'BEGIN''IF'¬(TOEND=0)'THEN''GOTO'1;'GOTO'999;1:PUTLABEL(999);999:'END';
  159    'PROCEDURE'ENDJUMP;'BEGIN''IF'¬(INPT=POINT)'THEN''GOTO'1;'IF'¬(TOLOST=0)'THEN''GOTO'1;'IF'¬(TONEXT=0)'THEN''GOTO'1;'GOTO'999;1:PUTJUMP(9
99);TOEND:=TOEND+1;999:'END';
  160    'PROCEDURE'PUTINIT;'BEGIN''IF'¬GIVERSTO'THEN''GOTO'1;G(INITRESTORE);'GOTO'999;1:999:'END';
  161    'PROCEDURE'PUTRESTORE;'BEGIN''IF'¬(TONEXT=0)'THEN''GOTO'1;'GOTO'999;1:PUTLABEL(CURLAB);'BEGIN''IF'¬GIVERSTO'THEN''GOTO'3;G(DORESTORE);'G
OTO'999;3:'GOTO'999;'END';999:'END';
  162    'PROCEDURE'FRESHLABEL;'BEGIN'CURLAB:=NEWLAB;NEWLAB:=NEWLAB+1;'END';
  163    'PROCEDURE'TRUERESULT;'BEGIN''IF'¬WASACTION(HANDLE)'THEN''GOTO'1;'GOTO'999;1:G(HANDLE);BECOMES;G(QTRUE);SEMICOL;999:'END';
  164    'PROCEDURE'FALSERESULT;'BEGIN''IF'¬WASACTION(HANDLE)'THEN''GOTO'1;'GOTO'999;1:G(HANDLE);BECOMES;G(QFALSE);SEMICOL;999:'END';
  165    'PROCEDURE'PUTJUMP(LAB);'INTEGER'LAB;'BEGIN'G(QGOTO);G(LAB);G(SEMICOLON);NEWLINE;APPLYLABEL(LAB);'END';
  166     PROCEDURE'PUTLABEL(LAB);'INTEGER'LAB;'BEGIN'NEWLINE;L;G(LAB);G(COLON);U;DEFINELABEL(LAB);TABLINE;'END';
  167    'PROCEDURE'SEMICOL;'BEGIN'G(SEMICOLON);'END';
  168    'PROCEDURE'BECOMES;'BEGIN'G(COLON);G(EQUALS);'END';
  169    'PROCEDURE'BEGIN;'BEGIN'NEWLINE;G(QBEGIN);U;TABLINE;'END';
  170    'PROCEDURE'ENDBR;'BEGIN'NEWLINE;L;G(QEND);'END';
  171    'COMMENT'4.4.7GENERALFORMOFARULE;
  172    'BOOLEAN''PROCEDURE'RULE;'BEGIN''IF'¬LEFTHANDSIDE'THEN''GOTO'1;'IF'¬MIDDLE'THEN''GOTO'0;'IF'¬RIGHTHANDSIDE'THEN''GOTO'0;FORGETLOCALS;SHI
```

```
FT2LINES:'IF'¬R(POINT)'THEN''GOTO'0;RULE:='TRUE';'GOTO'999;1:0:RULE:='FALSE';999:'END';
  173    'BOOLEAN''PROCEDURE'LEFTHANDSIDE;'BEGIN''IF'¬ISTAG(HANDLE)'THEN''GOTO'1;BLANKLINE;BOUNDX:=0;SPECX:=0;'BEGIN''IF'¬WASACTION(HANDLE)'THEN'
'GOTO'2;DEFINEACTION(HANDLE);RESTLHS;LEFTHANDSIDE:='TRUE';'GOTO'999;2:G(QBOOLEAN);DEFINEPREDICATE(HANDLE);RESTLHS;LEFTHANDSIDE:='TRUE';'GOTO'999
;'END';1:LEFTHANDSIDE:='FALSE';999:'END';
  174    'PROCEDURE'RESTLHS;'BEGIN'G(QPROCEDURE);G(HANDLE);OPTIONALBOUNDAFFIXES;NEWNPARS(HANDLE,BOUNDX);SEMICOL;'END';
  175    'PROCEDURE'OPTIONALBOUNDAFFIXES;'BEGIN''INTEGER'AFFX;'IF'¬BOUNDAFFIX1(AFFX)'THEN''GOTO'1;G(OPEN);G(AFFX);RESTBOUNDAFFIXES;'GOTO'999;1:'I
F'¬BOUNDAFFIX2(AFFX)'THEN''GOTO'2;G(OPEN);G(AFFX);RESTBOUNDAFFIXES;G(AFFX);'GOTO'999;2:999:'END';
  176    'PROCEDURE'RESTBOUNDAFFIXES;'BEGIN''INTEGER'AFFX;'IF'¬BOUNDAFFIX1(AFFX)'THEN''GOTO'1;G(COMMA);G(AFFX);RESTBOUNDAFFIXES;'GOTO'999;1:'IF'¬
BOUNDAFFIX2(AFFX)'THEN''GOTO'2;G(COMMA);G(AFFX);RESTBOUNDAFFIXES;G(AFFX);SPECX:=SPECX-1;'BEGIN''IF'¬(SPECX=0)'THEN''GOTO'3;'GOTO'999;3:G(COMMA);
'GOTO'999;'END';2:G(CLOSE);'BEGIN''IF'¬(SPECX=0)'THEN''GOTO'6;'GOTO'999;6:SEMICOL;G(QINTEGER);'GOTO'999;'END';999:'END';
  177    'BOOLEAN''PROCEDURE'BOUNDAFFIX1(AFFX);'INTEGER'AFFX;'BEGIN''IF'¬R(TIMES)'THEN''GOTO'1;'IF'¬ISTAG(AFFX)'THEN''GOTO'0;DEFINEAFFIX(AFFX);BO
UNDX:=BOUNDX+1;BOUNDAFFIX1:='TRUE';'GOTO'999;1:0:BOUNDAFFIX1:='FALSE';999:'END';
  178    'BOOLEAN''PROCEDURE'BOUNDAFFIX2(AFFX);'INTEGER'AFFX;'BEGIN''IF'¬R(PLUS)'THEN''GOTO'1;'IF'¬ISTAG(AFFX)'THEN''GOTO'0;DEFINEAFFIX(AFFX);BOU
NDX:=BOUNDX+1;SPECX:=SPECX+1;BOUNDAFFIX2:='TRUE';'GOTO'999;1:0:BOUNDAFFIX2:='FALSE';999:'END';
  179    'BOOLEAN''PROCEDURE'MIDDLE;'BEGIN'BEGIN;OPTIONALFREEAFFIXES;'IF'¬R(COLON)'THEN''GOTO'1;PUTINIT;MIDDLE:='TRUE';'GOTO'999;1:MIDDLE:='FALSE
';999:'END';
  180    'PROCEDURE'OPTIONALFREEAFFIXES;'BEGIN''INTEGER'AFFX;'IF'¬R(MINUS)'THEN''GOTO'1;G(QINTEGER);FFX:'IF'¬ISTAG(AFFX)'THEN''GOTO'0;G(AFFX);DEF
INEAFFIX(AFFX);'BEGIN''IF'¬R(MINUS)'THEN''GOTO'2;G(COMMA);'GOTO'FFX;2:G(SEMICOLON);'GOTO'999;'END';1:'GOTO'999;0:999:'END';
  181    'BOOLEAN''PROCEDURE'RIGHTHANDSIDE;'BEGIN'CURLAB:=1;NEWLAB:=2;TOLOST:=0;TOEND:=0;NXT:'IF'¬ALTERNATIVE'THEN''GOTO'1;PUTCONNECTOR;'BEGIN''I
F'¬R(SEMICOLON)'THEN''GOTO'2;'GOTO'NXT;2:ENDLABEL;ENDBR;SEMICOL;RIGHTHANDSIDE:='TRUE';'GOTO'999;'END';1:RIGHTHANDSIDE:='FALSE';999:'END';
  182    'BOOLEAN''PROCEDURE'ALTERNATIVE;'BEGIN''INTEGER'LAB,OLDNEXT;TONEXT:=0;ADVANCED:='FALSE';NXT:'IF'¬R(COMMA)'THEN''GOTO'1;'GOTO'NXT;1:'IF'¬
R(COLON)'THEN''GOTO'2;'IF'¬ISTAG(LAB)'THEN''GOTO'0;PUTJUMP(LAB);ALTERNATIVE:='TRUE';'GOTO'999;2:'IF'¬R(OPEN)'THEN''GOTO'3;OLDNEXT:=TONEXT;'IF'¬C
HOICE'THEN''GOTO'0;'IF'¬R(CLOSE)'THEN''GOTO'0;'BEGIN''IF'¬GIVERSTO'THEN''GOTO'4;TONEXT:=1;ALTERNATIVE:='TRUE';'GOTO'999;4:TONEXT:=OLDNEXT;ALTERN
ATIVE:='TRUE';'GOTO'999;'END';3:'IF'¬MEMBER'THEN''GOTO'6;'GOTO'NXT;6:TRUERESULT;ENDJUMP;ALTERNATIVE:='TRUE';'GOTO'999;0:ALTERNATIVE:='FALSE';999
:'END';
  183    'BOOLEAN''PROCEDURE'CHOICE;'BEGIN''INTEGER'LAB;LAB:=CURLAB;BEGIN;PUTINIT;FRESHLABEL;NXT:'IF'¬ALTERNATIVE'THEN''GOTO'1;PUTCONNECTOR;'BEGI
N''IF'¬R(SEMICOLON)'THEN''GOTO'2;'GOTO'NXT;2:ENDBR;SEMICOL;CURLAB:=LAB;CHOICE:='TRUE';'GOTO'999;'END';1:CHOICE:='FALSE';999:'END';
  184    'BOOLEAN''PROCEDURE'MEMBER;'BEGIN''INTEGER'HEAD;'IF'¬ISTAG(HEAD)'THEN''GOTO'1;APPLY(HEAD);'BEGIN''IF'¬R(COLON)'THEN''GOTO'2;PUTLABEL(HEA
D);MEMBER:='TRUE';'GOTO'999;2:'IF'¬TRYTYPE(HEAD,PREDICATE)'THEN''GOTO'3;G(QIF);G(NOT);PUTAFFIXEXPRESSION(HEAD);G(QTHEN);PUTCONNECTAND;ADVANCED:=
'TRUE';MEMBER:='TRUE';'GOTO'999;3:'IF'¬TRYTYPE(HEAD,FLAG)'THEN''GOTO'4;G(QIF);G(NOT);PUTAFFIXEXPRESSION(HEAD);G(QTHEN);PUTCONNECTAND;MEMBER:='TR
UE';'GOTO'999;4:'IF'¬TRYTYPE(HEAD,ACTION)'THEN''GOTO'5;PUTAFFIXEXPRESSION(HEAD);SEMICOL;MEMBER:='TRUE';'GOTO'999;5:'GOTO'0;'END';1:0:MEMBER:='FA
LSE';999:'END';
  185    'COMMENT'4.4.8OTHERBUILDINGSTONESOFAGRAMMAR;
  186    'BOOLEAN''PROCEDURE'COMMAND;'BEGIN''IF'¬R(RSTOON)'THEN''GOTO'1;GIVERSTO:='TRUE';COMMAND:='TRUE';'GOTO'999;1:'IF'¬R(RSTOOFF)'THEN''GOTO'2
;GIVERSTO:='FALSE';COMMAND:='TRUE';'GOTO'999;2:'IF'¬R(SHORT)'THEN''GOTO'3;LEGIBLE:='FALSE';COMMAND:='TRUE';'GOTO'999;3:'IF'¬R(LONG)'THEN''GOTO'4
;LEGIBLE:='TRUE';COMMAND:='TRUE';'GOTO'999;4:'IF'¬R(TRACEON)'THEN''GOTO'5;GIVETRACE:='TRUE';COMMAND:='TRUE';'GOTO'999;5:'IF'¬R(TRACEOFF)'THEN''G
OTO'6;GIVETRACE:='FALSE';COMMAND:='TRUE';'GOTO'999;6:COMMAND:='FALSE';999:'END';
  187    'BOOLEAN''PROCEDURE'COMMENT;'BEGIN''IF'¬R(SUB)'THEN''GOTO'1;BLANKLINE;G(QCOMMENT);RST:'BEGIN''IF'¬R(BUS)'THEN''GOTO'2;G(SEMICOLON);COMME
NT:='TRUE';'GOTO'999;2:G(INPT);READ(INPT);'GOTO'RST;'END';1:COMMENT:='FALSE';999:'END';
  188    'BOOLEAN''PROCEDURE'STARTINGSYMBOL;'BEGIN''INTEGER'HEAD;'IF'¬R(RESULT)'THEN''GOTO'1;'IF'¬ISTAG(HEAD)'THEN''GOTO'0;NEWPAGE;'BEGIN''IF'¬TR
YTYPE(HEAD,ACTION)'THEN''GOTO'2;APPLYACTION(HEAD);TERMINALS;G(HEAD);STARTINGSYMBOL:='TRUE';'GOTO'999;2:APPLYPREDICATE(HEAD);TERMINALS;G(HEAD);ST
ARTINGSYMBOL:='TRUE';'GOTO'999;'END';1:0:STARTINGSYMBOL:='FALSE';999:'END';
  189    'PROCEDURE'SKIPUNTILPOINT;'BEGIN'NXT:ERROR(SKIPPED,INPT);'BEGIN''IF'¬R(POINT)'THEN''GOTO'2;'GOTO'999;2:NEXTSYMBOL;'GOTO'NXT;'END';999:'E
ND';
  190    'COMMENT'4.4.9TERMINALS;
  191    'PROCEDURE'TERMINALS;'BEGIN'BLANKLINE;FORGETGLOBALS;NEWLINE;'IF'¬(PLOC=500001)'THEN''GOTO'1;'GOTO'999;1:APPLYACTION(XREAD);APPLYACTION(X
OUT);APPLYACTION(INITREAD);BLANKLINE;G(INITREAD);G(SEMICOLON);NEWLINE;BLANKLINE;FORGETTERMINALS;NEWLINE;BLANKLINE;999:'END';
  192    'PROCEDURE'FORGETGLOBALS;'BEGIN''INTEGER'P,TERM;P:=600001;PLOC:=500001;NXT:'IF'¬(P=PGLOB)'THEN''GOTO'1;'GOTO'999;1:TERM:=LGLOB[P];MAYBET
ERMINAL(TERM);P:=P+1;'GOTO'NXT;999:'END';
  193    'PROCEDURE'MAYBETERMINAL(X);'INTEGER'X;'BEGIN''IF'¬WASTERMINAL(X)'THEN''GOTO'1;PUTDECL(X);ENTERLOCAL(X);'GOTO'999;1:WARN(X);999:'END';
  194    'BOOLEAN''PROCEDURE'WASTERMINAL(X);'INTEGER'X;'BEGIN''INTEGER'S;S:=TTAG[X-5];'IF'¬(S=APPLIED)'THEN''GOTO'1;S:=TTAG[X-4];'IF'¬(S=GLOBAL)'
THEN''GOTO'1;S:=TTAG[X-3];'IF'¬(S=POINTER)'THEN''GOTO'1;WASTERMINAL:='TRUE';'GOTO'999;1:WASTERMINAL:='FALSE';999:'END';
  195    'PROCEDURE'PUTDECL(X);'INTEGER'X;'BEGIN'NEWLINE;G(QINTEGER);G(X);G(SEMICOLON);POSITION(64);INFORM(X);'END';
  196    'PROCEDURE'FORGETTERMINALS;'BEGIN''INTEGER'P,TERM;P:=500001;NXT:'IF'¬(P=PLOC)'THEN''GOTO'1;'GOTO'999;1:TERM:=LLOC[P];PUTCALL(XREAD,TERM)
;P:=P+1;'GOTO'NXT;999:'END';
  197    'PROCEDURE'PUTCALL(PROC,X);'INTEGER'X,PROC;'BEGIN'NEWLINE;G(PROC);G(OPEN);G(X);G(CLOSE);G(SEMICOLON);'END';
  198    'COMMENT'4.4.10POSTMORTEM;
  199    'PROCEDURE'POSTMORTEM;'BEGIN'NLCR;OUTINT(100001);OUTINT(PTAG);OUTINT(110000);NLCR;OUTINT(200001);OUTINT(PBOLD);OUTINT(200600);NLCR;OUTIN
T(300001);OUTINT(PSPEC);OUTINT(300100);NLCR;OUTINT(400001);OUTINT(PCONS);OUTINT(400300);NLCR;OUTINT(500001);OUTINT(XLOC);OUTINT(500200);NLCR;OUT
INT(600001);OUTINT(PGLOB);OUTINT(601000);NLCR;OUTINT(700001);OUTINT(PMACR);OUTINT(701100);NLCR;OUTINT(800001);OUTINT(PTEXT);OUTINT(804000);NLCR;
```

```
OUTINT(LINE);CUTINT(CARD);NLCR;NLCR;NLCR;DICTIONARY;'END';
  200    'PRCCEDURE'DICTIONARY;'BEGIN'LISTTAGS(FIRSTTAG);'END';
  201    'PRCCEDURE'LISTTAGS(X);'INTEGER'X;'BEGIN''INTEGER'P,STATE,SORT,TYPE;'IF'¬(X=0)'THEN''GOTO'1;'GOTO'999;1:P:=TTAG[X-2];LISTTAGS(P);STATE:=
TTAG[X-5]:SCRT:=TTAG[X-4];TYPE:=TTAG[X-3];'IF'¬(SORT=0)'THEN''GOTO'2;'GOTO'SKP;2:NLCR;OUT(X);POSITION(32);OUT(STATE);OUT(SORT);OUT(TYPE);POSITIO
N(59);ENTRIES(X);SKP;P:=TTAG[X-1];LISTTAGS(P);999:'END';
  202    'PRCCEDURE'ENTRIES(P);'INTEGER'P;'BEGIN''INTEGER'Q,H,K;Q:=TTAG[P-6];'IF'¬(Q=0)'THEN''GOTO'1;NLCR;'GOTO'999;1:LNE:H:=LTEXT[Q];Q:=H'/'8192
;K:=H-8192*Q;CUTINT(K);'BEGIN''IF'¬(Q=0)'THEN''GOTO'3:NLCR;'GOTO'999;3:Q:=Q+800001;Q:=Q-1;'GOTO'LNE;'END';999:'END';
  203    'COMMENT'4.5.HEARTOFCOMPILERCOMPILER;
  204    'INTEGER'INITRESTORE,DORESTCRE,XREAD,XOUT,INITREAD,TITLE;
  205    'PRCCEDURE'SENTENCE;'BEGIN'STARTSYSTEM;BLANKLINE;BEGIN;COMPILERDESCRIPTION;ENDBR;BLANKLINE;NEWPAGE;POSTMORTEM;'END';
  206    'PRCCEDURE'STARTSYSTEM;'BEGIN'READ(INITRESTORE);READ(DORESTORE);READ(XREAD);READ(XOUT);READ(INITREAD);READ(TITLE);GIVERSTO:='FALSE';GIVE
TRACE:='FALSE';GIVETEXT:='TRUE';PLOC:=500001;PGLOB:=600001;PTEXT:=800001;PMACR:=700001;CARD:=0;CPOS:=1;XLOC:=0;LINEDUP:='FALSE';LEGIBLE:='TRUE';
INDENTATION:=0;LINE:=0;POS:=0;OUT(TITLE);READ(INPT);'END';
  207    'PRCCEDURE'COMPILERDESCRIPTION;'BEGIN'NXT:'IF'¬SPECIFICATION'THEN''GOTO'1;'GOTO'NXT;1:'IF'¬DECLARATION'THEN''GOTO'2;'GCTO'NXT;2:'IF'¬COM
MAND'THEN''GOTO'3;'GOTO'NXT;3:'IF'¬COMMENT'THEN''GOTO'4;'GOTO'NXT;4:'IF'¬STARTINGSYMBOL'THEN''GOTO'5;'GOTO'999;5:'IF'¬RULE'THEN''GOTO'6;'GOTO'NX
T;6:SKIPUNTILPOINT;'GOTO'NXT;999:'END';
  208    'INTEGER'TAGFULL;'INTEGER'BCLDFULL;'INTEGER'SPECFULL;'INTEGER'CONSFULL;'INTEGER'WRONGINPUTCODE;'INTEGER'BLANK;'INTEGER'LOCFULL;'INTEGER'
GLOBFULL;'INTEGER'MACRFULL;'INTEGER'TEXTFULL;'INTEGER'DEFINED;'INTEGER'GLOBAL;'INTEGER'ACTION;'INTEGER'PREDICATE;'INTEGER'LOCAL;'INTEGER'POINTER
;'INTEGER'LABEL;'INTEGER'APPLIED;'INTEGER'XIMPOSSIBLE;'INTEGER'MACRO;'INTEGER'WRONGNUMBEROFPARAMETERS;'INTEGER'FLAG;'INTEGER'LIST;'INTEGER'EXTER
NAL;'INTEGER'COMMA;'INTEGER'POINT;'INTEGER'QINTEGER;'INTEGER'SEMICOLON;'INTEGER'QBOOLEAN;'INTEGER'QARRAY;'INTEGER'SUB;'INTEGER'COLON;'INTEGER'BU
S;'INTEGER'PLUS;'INTEGER'MINUS;'INTEGER'OPEN;'INTEGER'CLOSE;'INTEGER'STRINGQUOTE;'INTEGER'WRONGAFFIX;'INTEGER'ONE;'INTEGER'TWO;'INTEGER'THREE;'I
NTEGER'FOUR;'INTEGER'FIVE;'INTEGER'XPARAMETERERROR;'INTEGER'POSSIBLYBACKTRACKNECESSARY;'INTEGER'ALTERNATIVENEVERREACHED;'INTEGER'QTRUE;'INTEGER'
QFALSE;'INTEGER'QGOTO;'INTEGER'EQUALS;'INTEGER'QBEGIN;'INTEGER'QEND;'INTEGER'QPROCEDURE;'INTEGER'TIMES;'INTEGER'QIF;'INTEGER'NCT;'INTEGER'QTHEN;
'INTEGER'RSTOCN;'INTEGER'RSTOOFF;'INTEGER'SHORT;'INTEGER'LONG;'INTEGER'TRACEON;'INTEGER'TRACEOFF;'INTEGER'QCOMMENT;'INTEGER'RESULT;'INTEGER'SKIP
PED;
  209    INITIALIZEFORREADING;
  210    READ(TAGFULL);READ(BOLDFULL);READ(SPECFULL);READ(CONSFULL);READ(WRONGINPUTCODE);READ(BLANK);READ(LOCFULL);READ(GLOBFULL);READ(MACRFULL);
READ(TEXTFULL);READ(DEFINED);READ(GLOBAL);READ(ACTION);READ(PREDICATE);READ(LOCAL);READ(POINTER);READ(LABEL);READ(APPLIED);READ(XIMPOSSIBLE);REA
D(MACRO);READ(WRONGNUMBEROFPARAMETERS);READ(FLAG);READ(LIST);READ(EXTERNAL);READ(COMMA);READ(POINT);READ(QINTEGER);READ(SEMICOLON);READ(QBOOLEAN
);READ(QARRAY);READ(SUB);READ(COLCN);READ(BUS);READ(PLUS);READ(MINUS);READ(OPEN);READ(CLOSE);READ(STRINGQUOTE);READ(WRONGAFFIX);READ(ONE);READ(T
WO);READ(THREE);READ(FOUR);READ(FIVE);READ(XPARAMETERERROR);READ(POSSIBLYBACKTRACKNECESSARY);READ(ALTERNATIVENEVERREACHED);READ(QTRUE);READ(QFAL
SE);READ(QGOTO);READ(EQUALS);READ(QBEGIN);READ(QEND);READ(QPROCEDURE);READ(TIMES);READ(QIF);READ(NOT);READ(QTHEN);READ(RSTOCN);READ(RSTOOFF);REA
D(SHORT);READ(LCNG);READ(TRACEON);READ(TRACEOFF);READ(QCOMMENT);READ(RESULT);READ(SKIPPED);
  211    SENTENCE'END'
```

References.

[1] J. Feldman, D. Gries, "Translator Writing Systems", Communications of the ACM, Volume 11 number 2 (1968).

[2] C.H.A. Koster, "Affix Grammars", in: J.E.L. Peck (Editor), "ALGOL 68 implementation", North-Holland Publishing Company (1971).

[3] A. van Wijngaarden (Editor), "Report on the Algorithmic Language ALGOL 68", Numerische Mathematik, 14 (1969).

[4] P. Naur (Editor), "Revised Report on the Algorithmic Language ALGOL 60", Regnecentralen, Copenhagen (1962).

[5] D.E. Knuth, "Semantics of Context Free Languages", Math. Systems Theory 2, (1968).

[6] D.E. Knuth, "Top-down syntax analysis", International Summer School on Computer Programming, Copenhagen (1967).

[7] J. Earley, H. Sturgis, "A Formalism for Translator Interactions", Communications of the ACM, Volume 13 number 10 (1970).

The differences between versions 17 and 19 of the CDL-compiler; first update to MR 127/71.

1. p. 23, bottom line, ... ignored, add:

   ; moreover any sequence of symbols, not containing a $-character, enclosed between $-characters is also ignored

2. p. 24, l. 16, equals =, add:

   not ¬

3. p. 24, l. 25, list ..., add:

   comment symbol 'comment'

4. p. 25, l. 17, flag symbol, add:

   ; list symbol

5. p. 26, l. 2 from bottom, list (global, add:

   or macro

6. p. 27, l. 13, 'external' ... flag, add:

   'external' 'list' defined global list

7. p. 27, l. 19, 'macro'.... pointer, add:

   'macro' 'list' defined macro list

8. p. 30, l. 8, 2.6. Comments. .... *input;* , replace by

   2.6. Comments.

   comment: skipped comment; translated comment.
   skipped comment: sub, rest skipped comment.
   rest skipped comment: bus; nonbus, rest skipped comment.
   translated comment: comment symbol, rest translated comment.
   rest translated comment: point; nonpointnonsemicolon,
   rest translated comment.

   (Here, nonbus stands for any symbol except] and nonpoint nonsemicolon for any symbol except . or ; .)

BIBLIOTHEEK MATHEMATISCH CENTRUM
AMSTERDAM

A skipped comment is skipped and a translated comment is translated into an ALGOL 60 <comment>; e.g. [input] is skipped while 'comment' input. is translated into *'comment' input;*. The text of a comment may only consist of allowed CDL-symbols (see 2.2), as opposed to the sequence of characters between two $-characters (see 2.2).

9. p. 30, bottom line, replace by:

(semicolon, right hand side;) .

10. p. 31, lines 1-8, replace by:

alternative:
    label, colon, alternative;
    affix expression, (comma, alternative;);
    group; jump; .
affix expression:
    handle, optional affixes;
    not, tag.

11. p. 32, line 10, ... affix expression, add:

that starts with a handle

12. p. 33, after line 20, α; , add:

The translation of an affix expression that starts with a not is:

if tag *then goto* λ;

where λ is determined by means of the algorithm on the previous page.

13. p. 33, bottom, , add:

The above translation scheme makes use of integer labels, which, for some applications, are undesirable. Therefore a symbol is prefixed to every label in the translation; this symbol is the terminal "label prefix" of the CDL-compiler, which is read in during the initialization phase of the compiler. The prefix should be chosen so that the resulting names will not clash with existing names. The present value of "label prefix" is "l"; in the examples in this report its value is "" (i.e. the empty symbol).

14. p. 36, l. 4, delete: Various .... themselves.

15. p. 36, l. 25, ... occurrence. , add:

Line numbers of specification are preceded by S:,
line numbers of definition are preceded by D:.

16. after page 36 add the following pages:

2.10. Error detection and recovery.

All errors fall in one of the following four classes.

1) Warnings:

backtrack?	An alternative may fail although one of its predicates may have succeeded.
alternative never reached	One of the previous alternatives in this right hand side will always succeed.
nonfalse	This predicate will always succeed, and is in effect an action.
may be false	Possibly none of the alternatives of this action will succeed.

2) Unexpected symbols:

wrong input code	Some unreadable item is met on the input medium.
symbol missing, assumed: symbol	1. The grammar requires at this point the presence of the indicated symbol, which, however, was not found in the input. The symbol is assumed to be present, so that e.g. "'list' a[1:10." will yield a correct translation. 2. The grammar requires at this point the presence of a tag, which, however, was not found in the input. The tag "dummy" is inserted. 3. The grammar requires at this point the presence of one of the symbols: 'action' 'predicate'

'pointer'

'flag'

'list'

None of these, however, was found. The symbol 'predicate' was inserted.

symbol not allowed inside comment symbol

The indicated symbol violates the restrictions on the symbols allowed inside a comment. In this version of the compiler the symbol can only be a semicolon.

affix missing

The grammar requires at this point the presence of an affix, which, however, was not found. The tag "dummy" was inserted.

skipped --- state, sort, type, symbol

The indicated symbol was skipped because it was unacceptable as the first symbol of a compiler description.

3) Conflict of attributes

impossible redefinition state, sort, type, tag, new state, new sort, new type

The indicated tag with the indicated state, sort and type is specified, defined or applied with the conflicting attributes new state, new sort and new type.

wrong number of parameters state, sort, type, tag

The number of affixed following the indicated tag is incompatible with that number in the definition of the tag.

wrong sort state, sort, type, tag

The indicated local tag is applied as a handle.