The LISP Version of the Meta Compiler

ABSTRACT

This paper describes a meta-compiler program which processes a BNF-like language and produces a LISP II intermediate language program. The program produced is a syntax translator.

The version of the compiler described here exists as a LISP 1.5 program and operates on Q-32 LISP 1.5. It will produce itself as a LISP II intermediate language program.

The work reported herein is based upon the accomplishments of Val Schorre and Lee Schmidt, of the Los Angeles Chapter of ACM, SIGPLAN Working Group I.
1. **INTRODUCTION**

The LISP II programming language is to be processed into LISP II intermediate language by a syntax-directed compiler. This compiler is to be produced by using a meta-compiler. This document describes the Meta Compiler. The technique is based on the work done by Working Group I of the Los Angeles Chapter of ACM.

The Meta Compiler is a model of a machine with an input tape and a push down accumulator; the accumulator is referred to as the star stack and is symbolized by * . The compiler also has a true/false indicator cell called SIGNAL. The Meta Compiler translates a program written in its input language, which resembles BNF with extensions, into a tree structure. This tree structure is a LISP II intermediate language program. The program so translated is usually referred to as a compiler. The Meta Compiler used here is itself a Q-32 LISP 1.5 program.

A meta-language program is organized into a body of rules. Each rule corresponds to a syntax equation of BNF. A Rosetta paper follows:

<table>
<thead>
<tr>
<th>BNF</th>
<th>META</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>⟨something⟩</td>
<td>SOMETHING</td>
<td>Meta-linguistic variables</td>
</tr>
<tr>
<td>A</td>
<td>'A'</td>
<td>Terminal character or string</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>Alternation</td>
</tr>
<tr>
<td>{ }</td>
<td>( )</td>
<td>Meta-linguistic parentheses</td>
</tr>
</tbody>
</table>

Writing two entities side by side (such as AB) means that an A is followed by a B. The ending symbol of a rule in LISP-META is the semicolon. BNF has no ending symbol for its syntax equations.

Identifiers are meta-linguistic variables, to wit, other definitions. They may also be the names of subroutines. If an identifier is followed by square brackets, the identifier then is the name of a routine to be executed. Its parameters are enclosed by the brackets and are separated by commas. Strings are groups of characters enclosed in primes. These correspond to terminal characters. If a prime is to be used within a string, two primes are written.

The remainder of this document is organized as follows:

A description of the various routines which are not defined in the syntax equation of the Meta Compiler, appears first. If the meta language refers to these routines by other than their names, the encoding is also shown. The next section is an English-language description of selected equations. A listing of the Meta Compiler, written in its own language, appears last.

---

*Backus Naur Form.*
2. SYNTACTIC ROUTINES

These routines are principally concerned with asking questions about the characters on the input tape and what to do with them.

2.1 META LANGUAGE: 'CHARACTERS'

Routine: CMPR string (Compare)

Meaning:

When a string is written in a syntax equation it means: "If the next group of characters on the input tape matches the exhibited string, move the read past those characters and report true. Otherwise report false and do not move the read head".

2.2 META LANGUAGE: + 'CHARACTERS'

Routine: COMPS string (Compare and store)

Meaning:

This expression has the same effect as is presented in Section 2.1, except that if the answer is true the matched characters are put into the accumulator (*).

2.3 META LANGUAGE: - 'CHARACTERS'

Routine: NCOMP string (No compare)

Meaning:

If the next characters on the input tape match the exhibited string, report false. If there is no match, report true. However, do not move the read head in either event.

2.4 META LANGUAGE: ! 'CHARACTERS'

Routine: CMPR2 string

Meaning:

If the next characters on the input tape match the exhibited string, make a token (atom) out of the characters, push the token into the accumulator, move the read head past those characters, and report true. Otherwise, report false and do not move the read head.
2.5 **META LANGUAGE: ..**
Routine: MARK

**Meaning:**

Syntax equations which have a double period instead of an equal sign are used to collect characters and to make tokens out of them. The routine MARK is executed when these syntax equations are entered. MARK skips blanks on the input tape and stops at the first non-blank character. The routine then sets the skip blanks flag to "off" so that blanks become significant to all routines which look at characters on the input tape; that is, the routines do not bypass leading blanks while this flag is off. MARK then sets a mark in the accumulator so that all characters put into the accumulator on top of this mark will be collected as one token in a first-in first-out manner.

2.6 **META LANGUAGE: ; (AT THE END OF .. EQUATION)**
Routine: TOKEN

**Meaning:**

TOKEN collects all characters, starting with the character above the mark and going to the top of the accumulator. These characters are formed into a token. TOKEN then sets the skip blanks flag to "on," so that routines which look at the input tape characters ignore leading blanks.

2.7 **META LANGUAGE: ANY**
Routine: ANY

**Meaning:**

Put the next character on the input tape into the accumulator.

2.8 **META LANGUAGE: DELETE**
Routine: DELETE

**Meaning:**

Skip the next character on the input tape.

2.9 **META LANGUAGE: $**
Routine: $

**Meaning:**

Recognize zero or more of the following syntactic entities.
3. SEMANTIC ROUTINES

These routines are concerned with building up the tree structure which reflects the parse of the syntax.

3.1 META LANGUAGE: ⟨
Routine: FLAG

Meaning:
Set a flag in the accumulator so that a sub-tree will be formed out of the tokens and expressions collected until SEQ (⟩) is executed.

3.2 META LANGUAGE: ⟩
Routine: SEQ

Meaning:
Completes the formation of a sub-tree out of whatever has been collected since FLAG was executed.

3.3 META LANGUAGE: ¯n
Routine: STARn

Meaning:
STARn produces the nth element of the accumulator and removes it from the accumulator.

3.4 META LANGUAGE: +*n
Routine: STARnP

Meaning:
Copies the nth element of the accumulator onto the top of the accumulator without removing it.

3.5 META LANGUAGE: *[ and $[
Routine: PUSH parameter

Meaning:
Creates a list or node out of the parameters of *[ or $[ and leaves it on top of the accumulator (*).
3.6  META LANGUAGE: 'CHARACTERS'
Routine: INSERT string

Meaning:
Push the string of characters shown in the syntax equation into the accumulator.

3.7  META LANGUAGE: t IDENTIFIER
Routine: LOAD x

Meaning:
Push the identifier into the stack.

3.8  META LANGUAGE: GN1 or GN2
Routine: GN1 or GN2

Meaning:
The GN1 and GN2 routines are concerned with obtaining labels for transfer points. They manipulate a stack called GEN, which is organized into pairs. The first element of each pair concerns the GN1 routine; the second element of each pair concerns the GN2 routines. If the first (second) element of the top pair is empty, a symbol is generated and put there. The first (second) element of the top pair is always produced as output.

3.9  META LANGUAGE: GEN1 or GEN2
Routine: GEN1 or GEN2

Meaning:
Push the output of GN1 (GN2) into the accumulator.

3.10 META LANGUAGE: MAKEATOM
Routine: MAKEATOM

Meaning:
Replace the string of characters on top of the accumulator by an atom with the same print name.
3.11  META LANGUAGE:  MAKENUMBER
Routine:  MAKENUMBER
Meaning:
Replace the string of digits on top of the accumulator with its integer value.
4. **BACKUP ROUTINES**

If more than one syntax equation or alternative start with the same construct, there is a possibility that an ambiguous situation will arise where backup over that first construct must occur in order to go on with the parsing. In order to accomplish the backup, the state of the machine must be saved and restored at critical places. Six routines, a stack called BACK and one called NAME are used to attempt to recover from ambiguous situations.

4.1 **META LANGUAGE: RPT1**

Routine: RPT1

Meaning:

This routine is invoked at the top of a loop set up by the sequence operation ($). It increments a cell called BACKUP-COUNT.

4.2 **META LANGUAGE: RPT2**

Routine: RPT2

Meaning:

This routine is invoked at the bottom of a loop set up by the sequence operation ($). It decrements a cell called BACKUP-COUNT. Whenever this cell is greater than zero, nothing is saved and backup does not take place.

4.3 **META LANGUAGE: ENTER X**

Routine: ENTER

Meaning:

This routine is used upon entering a syntax equation. The name of the syntax equation being entered is saved on a NAME list. A "blip" is pushed into the top of BACK for constructs which are collected by this syntax equation. A blip is an empty list and is used to collect information.

The GEN stack has two blips pushed into it, in case generated labels are needed by this syntax equation. Then the next entity on the INPUT tape is examined. If it is the name of the routine being entered, that name is removed from the input tape and SIGNAL is set to true.
4.4 **META LANGUAGE:** LEAVE  
**Routine:** LEAVE  
**Meaning:**  
Invoked upon leaving a syntax equation. The GEN stack has its top two elements popped off. If the BACKUP-COUNT cell is zero, then SIGNAL is checked. If SIGNAL is true, then the name of the routine being left is put on top of BACK. Otherwise the top element of BACK is popped. In any event, the top element of NAME is popped.

4.5 **META LANGUAGE:** SAVER  
**Routine:** SAVER  
**Meaning:**  
This routine is called at the beginning of an expression and it merely pushes a blip into BACK. The blip gets filled by all constructs which the expression collects.

4.6 **META LANGUAGE:** RSTOR  
**Routine:** RSTOR  
**Meaning:**  
This routine is called at the end of expressions. If SIGNAL is true, it takes the constructs which have been collected by the expression just processed, and groups them with the constructs being collected by the next higher level expression, which may be a syntax equation. Otherwise it puts those constructs on the INPUT tape. In either event BACK is popped.

Now it is possible to examine Figure 1, which shows the Meta Compiler, and get an approximate idea of what it does. A few sample syntax equations are followed through; the remainder of the equations can be used as exercises for the interested reader. The extreme left column shows line numbers and is not part of the syntax equations.

```
4500 ID .. LET $(LET / DGT / \-\- ,\-\-\- ) MAKEATOM;
```

This is the definition of an identifier. The double dot shows we are forming a token and are going to skip leading blanks on the input tape. When the first non-blank character is encountered, the SKIP-BLANKS flag is set to false. We then transfer to a routine called LET which sees whether the next character on the input tape is a letter. If not, we exit ID false. If it is, we look for a sequence, which may be empty of LET letters or DGT digits or minus signs, which represent hyphens. By examining the LET and DGT equations, we see that if
Figure 1. A Listing of the LISP Meta Compiler
an appropriate character is recognized it is pushed into the accumulator, *, because of the + before each string. However, the ID equation indicates that the minus, if recognized, is not put into the accumulator. The comma followed by the string period, "," shows that a period is inserted instead. After processing such a sequence of characters, a routine called MAKEATOM is called. This subroutine takes all the characters collected as an identifier token and makes an atom of them.

200 SYNTAX = '.META'
< $(RULES / -.FINISH' Gobble) '.FINISH' > COMPILE;

This equation defines a meta-language program as starting with .META. A flag is set up so that the entire program will be collected as one list. Then we encounter zero or more of RULES. If we do not find the characters .FINISH, we call Gobble (4300). A cursory glance at Gobble shows that it goes to ERROR and then reads the input tape deleting characters until it finds a semi-colon, at which time it throws that away also and returns to SYNTAX. If we do find .FINISH, we close the list which contains the parse of the program being defined. Then we go to the LISP compiler.

It is hoped that the availability of a meta compiler in LISP will make it possible to produce the syntax translator for LISP II to intermediate language more easily. A meta compiler should also facilitate the processing of modifications and improvements to the LISP II source language.

The method of production used was to take the already existing Meta Compiler on the Q-32 and make it produce LISP 1.5 output. By the well known bootstrap cannibalism it reproduced itself as a LISP 1.5 program. The syntax equations are modified to produce LISP II intermediate language and the cannibal eats again.
The definition of META and LISP in META XI
The definition of META and LISP written in META:

0005700-ELEMENT = NUM / STRING / CHARACTER / IDENT / STACK /
0005800- LISTX / '(' EXPNQ ')' /
0005900- IDENT = ID /
0006000- PARAMSQ = [*, -[*2, -*1-] ] /
0006100- ('*' EXPNQ ')'/
0006200- ('*' EXPNQ 'IFDEFLIST /
0006300- ['LIST, ['LIST, *2, *1]], [QUOTE, *1]] /
0006400- EMPTY [GET, *1, [QUOTE, *1]] ] /
0006600- [SETQ, *2, *1], [QUOTE, *1], [EMPTY]) ;
0006700- LIST-SEQ = ['L' EXPNQ 'I'-'] /
0006800- ['LIST-SEQ [APPEND, *2, *1], [EMPTY] /
0006900- EXPQ ['L' LIST-SEQ 'EMPTY'] /
0007100- LISTX = 'L' LIST-SEQ 'EMPTY'] ')'/
0007200- EXPQ = WHERE [FUNCTION EXPX], TJ /
0007300- EXPNQ = WHERE [FUNCTION EXPX], F /
0007400- LISTEXP = ELEMENT S ('*1', [CAR, *1] /
0007900- ['EMPTY] /
0008200- ['EMPTY], /
0008300- NEGATION = '!', RELATION, [NOT, *1] /
0008400- RELATION /
0008500- FACTOR = NEGATION ['A', FACTOR /
0008600- ['AND, *2, *1], [EMPTY] ];
0008700- EXPX = FACTOR ['V', EXPX /
0008800- ['OR, *2, *1], [EMPTY];
0008900- LOOPST = ' 'LOOP', 'UNTIL', GEN1 /
0009000- (EXPNQ \ ERRORX S< '-BEGIN', 'DELETEX>) /
0009100- ['BEGIN', [COND, 'L1', [GO, GN2]] ]); /
0009200- $ST \ END', 'GOBBLE] /
0009300- ['END', [GO, GN1]], GEN2; /
0009400- IFST = ' 'IF', (EXPNQ \ ERRORX S< '-BEGIN', 'DELETEX>) /
0009500- ['BEGIN', '[THEN, [EMPTY] ];
0009600- [COND, 'L1', [GO, GN1]] ]); /
0009700- $ST /
0009800- ['ELSE', [GO, GN2]], GEN1 /
0009900- $ST \ END', 'GOBBLE] /
0010000- ['END', GEN2, ['END', GEN1] ];
00101000- PRINTST = 'PRINT', $OUTPUT ';
00102000- ST = EXPNQ 'I'; / LOOPST / IFST / PRINTST /
00103000- IDSEQ = ['L', '<', [FORMAL $S', 'FORMAL, [EMPTY] '''] ));
00104000- 'FORMAL = ID \ 'LOC', 'ID' /
00105000- 'PROCEDURE = 'PROCEDURE', ID IDSEQ 'I'; /
00106000- ['LOCAL', 'IDSEQ ] '\ EMPTY] []) /
00107000- $ST \ RETURN', 'GOBBLE] /
00108000- ['RETURN', ['I EXPNQ 'RETURN', *1]'], [EMPTY] '''] ));
00109000- $DEFINE, [L1, ['L4]
00110000- [$LAMBDA, *3, [PROG, *2, [-*1-]]]), COMPilee /
00111000- LISP-DIVISION = 'LISP', $PROCEDURE 'FINISH' ;
00112000- FLUID-DECLARATION = 'DECLARE', 'I'/
00113000- FLUIDI $S', 'FLUID] '''] ''';
00114000- FLUID1 = ID \ ICSET, '*1, N[1]], COMPilee /
00115000- PROGRAM = '$SYNTAX \ LISP-DIVISION /
00116000- FLUID-DECLARATION 'STOP' /
00117000- 'FINISH /
00118000- 'STOP
(1 DEFINE (((META (LAMBDA (X Y))
    (COMPLETE (INITIALIZE X Y) (PROGRAM)))()))
(1 DEFINE (((SYNTAX (LAMBDA NIL (LEAVE (ENTER))
    (AND (CMPR (QUOTE ("'M 'E 'T 'A"))
        (OR (AND (CR (AND (CMPR (QUOTE (""))))
            (CR (ID) (SETQ OK F))
            (CR (ID) (SETQ UK F))))
            (CR (CMPR (QUOTE (""))) (SETQ OK F))))
        (OR (LCAD (CONS (QUOTE DEFINE)
            (CONS (CONS (CONS (CONS (STAR2))
                (CONS (CONS (QUOTE LAMBDA)
                    (CONS (CONS (QUOTE X) (CONS (QUOTE Y) NIL))
                    (CONS (CONS (QUOTE COMPLETE)
                        (CONS (CONS (QUOTE INITIALIZE)
                            (CONS (QUOTE X) (CONS (QUOTE Y) NIL)))
                            (CONS (CONS (STAR1) NIL) NIL))) NIL))) NIL)) NIL))
            (SETQ OK F))
        (OR (CCMPILE) (SETQ UK F))) (NOT OK) TRUE) OK)
        (SETQ OK F))
    (OR (PROG NIL MO0001 (COND ((RESTORE (SAVER)
        (AND (OR (BACKUP (RULE)))
        (AND (NCGMP (QUOTE ("'F 'I 'N 'I 'S 'H"))
            (OR (GOBBLE) (SETQ CK F)))) (OK)) (GO MO0001)))
        (RETURN (K1) (SETQ OK F)))
    (OR (CMPR (QUOTE ("'F 'I 'N 'I 'S 'H")) (SETQ UK F)))))))))
(1 DEFINE (((RULE (LAMBDA NIL (LEAVE (ENTER))
    (AND (IC)
    (OR (AND (CR (AND (CMPR (QUOTE (""))))
        (OR (EXPR) (SETQ OK F)))
        (NOT OK)
        (AND (CMPR (QUOTE ("")))
        (OR (EXPR) (SETQ OK F))))
    (OR (LCAD (CONS (QUOTE TOKEN)
        (CONS (CONS (QUOTE MARK) NIL) (CONS (STAR1) NIL))
        (SETQ OK F))) (OK) (SETQ OK F))
    (OR (CMPR (QUOTE (""))) (SETQ OK F))
    (OR (LCAD (CONS (QUOTE DEFINE)
        (CONS (CONS (CONS (CONS (STAR2))
            (CONS (CONS (QUOTE LAMBDA)
                (CONS (QUOTE NIL)
                (CONS (CCNS (QUOTE LEAVE)
                    (CONS (CONS (QUOTE ENTER) NIL)
                    (CONS (STAR1) NIL))) NIL) NIL) NIL))) NIL)))
        (SETQ OK F)) (OR (COMPARE) (SETQ UK F)))))))))
(1 DEFINE (((EXPR (LAMBDA NIL (LEAVE (ENTER))
    (AND (SUBEXP)
    (OR (AND (CR (AND (EXPR))
        (OR (SEQ (FLAGS))
        (OR (PROG NIL MO0002 (COND ((EXPR) (GO MO0002)))
            (RETURN OK))) (SETQ OK F))
    (OR (LCAD (CONS (QUOTE AND)
        (CONS (CONS (QUOTE OR)
            (CONS (STAR4)
            (CONS (STAR3) (CONS (STAR2) (STAR1))))
        (CONS (QUOTE OK) NIL))) (SETQ OK F))))
    (NOT OK)
    (AND (CMPR (QUOTE ("")))
    (OR (LCAD (CONS (QUOTE BACKUP) (CONS (STAR1) NIL)))
        (SETQ OK F))
    (OR (SEQ (FLAGS))
        (AND (SUBEXP)
        (OR (PROG NIL MO0003 (CONS ((EXPR2) (GO MO0002))))))
))
(RETURN OK)) (SETQ OK F))) (SETQ OK F))
(OR (LCAD (CONS (QUOTE RESTORE)
(CONS (CONS (QUOTE SAVER) NIL)
(CONS (CONS (QUOTE AND)
(CONS (CONS (QUOTE OR) (CONS (STAR2) (STAR1)))
(CONS (QUOTE OK) NIL)) NIL))) NIL)) (SETQ OK F)))
(NOT OK) TRUE) OK) (SETQ OK F)))))))))
(1 DEFINE (!!(EXPR1 (LAMBDA NIL ( LEAVE ( ENTER)
(AND (CMPR (QUOTE \\))
(OR (LCAD (CONS (QUOTE NOT) (CONS (QUOTE OK) NIL))))
(SETQ OK F)) (OR (SUBEXP) (SETQ OK F)))))
(1 DEFINE (((EXPR2 (LAMBDA NIL ( LEAVE ( ENTER)
(AND (CMPR (QUOTE \\))
(OR (LCAD (CONS (QUOTE BACKUP) (CONS (STAR1) NIL)))
(SETQ OK F)) (OR (SUBEXP) (SETQ OK F)))))
(1 DEFINE (((SUBEXP (LAMBDA NIL ( LEAVE ( ENTER)
(AND (TESTS)
(OR (AND (CR (AND (BACKTEST)
(OR (SEQ (FLAGS)
(PROC NIL M00004 (COND ((BACKTEST) (GO M00004)))
(RETURN OK))) (SETQ OK F))
(OR (LCAD (CONS (QUOTE AND)
(CONS (STAR3) (CONS (STAR2) (STAR1))) (SETQ OK F)))
(NOT OK) TRUE) OK) (SETQ OK F)))
(1 DEFINE (((BACKTEST (LAMBDA NIL ( LEAVE ( ENTER)
(AND (TESTS)
(OR (LCAD (CONS (QUOTE OK)
(CONS (STAR1))
(CONS (CONS (QUOTE SETQ)
(CONS (QUOTE OK) (CONS (QUOTE F) NIL)) NIL)))
(SETQ OK F)))))))))
(1 DEFINE (((TESTS (LAMBDA NIL ( LEAVE ( ENTER)
(AND (OR (AND (ID)
(OR (AND (OR (AND (PARAMSQ)
(OR (LOAD (CONS (STAR2) (STAR1))) (SETQ OK F))
(NOT CK))
(AND TRUE (OR (LOAD (CONS (STAR1) NIL))
(SETQ OK F))) OK) (SETQ OK F)))
(NOT CK)
(AND (STRING1)
(OR (LOAD (CONS (QUOTE CMPR) (CONS (STAR1) NIL)))
(SETQ OK F))
(NOT CK)
(AND (CMPR (QUOTE '+))
(OR (STRING1) (SETQ OK F))
(OR (LOAD (CONS (QUOTE CCMPS) (CONS (STAR1) NIL)))
(SETQ OK F))
(NOT CK)
(AND (CMPR (QUOTE '-))
(OR (STRING1) (SETQ OK F))
(OR (LOAD (CONS (QUOTE NCCMP) (CONS (STAR1) NIL)))
(SETQ OK F))
(NOT CK)
(AND (CMPR (QUOTE '*))
(OR (AND (OR (AND (ID)
(OR (LOAD (CONS (QUOTE LOAD)
(CONS (CONS (QUOTE QUOTE)
(CONS (STAR1) NIL)) NIL))) (SETQ OK F)))
(NOT CK)
(AND (STRING1)
(OR (LOAD (CONS (QUOTE CMPR2) (CONS (STAR1) NIL)))
(SETQ OK F)))))) OK) (SETQ OK F)))

(NOT CK)
  (AND (CMPR (QUOTE (""')))
    (OK (EXPR (setq OK F))
      (OR (CMPR (QUOTE (""')) (setq OK F))
        (OR (LOAD (CONS (QUOTE SEQ)
                    (CONS (CONS (QUOTE FLAGS) NIL) (CONS (STAR1) NIL)))
            (setq OK F)))
      (NOT CK).
      (AND (CMPR (QUOTE (""')))
        (CR (EXPR) (setq .OK F))
        (CR (CMPR (QUOTE (""'))))) (setq OK F)))
      (NOT CK)
      (AND (CMPR (QUOTE (""')))
        (OR (STRING1) (setq OK F))
        (OR (LOAD (CONS (QUOTE INSERT) (CONS (STAR1) NIL)))
            (setq OK F)))
      (NOT CK)
      (AND (LISTX)
        (OR (LOAD (CONS (QUOTE LOAD) (CONS (STAR1) NIL)))
            (setq OK F)))
      (NOT CK)
      (AND (CMPR (QUOTE (""')))
        (CR (SEQ (FLAGS))
          (PROG NIL MOO005 (COND ((CUTPUT) (GO MOO005)))
            (RETURN OK))) (setq CK F))
      (CR (LOAD (CONS (QUOTE PROG)
                    (CONS (QUOTE NIL)
                      (APPEND (STAR1)
                        (CONS (CONS (QUOTE RETURN)
                          (CONS (CONS (QUOTE T) NIL) NIL))))))
                (setq OK F))
      (CR (CMPR (QUOTE (""'))))) (setq OK F)))
      (NOT CK) (REPEAT (OK)))
(1 DEFINE ((CUTPUT) (LAMBDA NIL (LEAVE (ENTER))
    (AND (OR (AND (STRING1)
                  (OR (LOAD (CONS (QUOTE PRINT) (CONS (STAR1) NIL)))
                      (setq OK F)))
    (NOT CK))
    (AND (ID)
      (OR (AND (OR (AND (PARAMSG)
                      (OR (LOAD (CONS (QUOTE PRINT)
                          (CONS (CONS (STAR2) (STAR1) NIL))) (setq OK F))))
        (NOT CK)
        (AND TRUE (OR (LOAD (CONS (QUOTE PRINT)
                    (CONS (STAR1) NIL))) (setq OK F))) OK)
      (setq OK F)))
    (NOT CK)
    (AND (STACK)
      (OR (LOAD (CONS (QUOTE PRINT) (CONS (STAR1) NIL)))
          (setq OK F)))
    (NOT CK)
    (AND (CMPR (QUOTE ("/")))
      (OR (LOAD (CONS (QUOTE TERMINAL) NIL))
          (setq OK F))) (OK))))))))
(1 DEFINE ((REPEAT (LAMBDA NIL (LEAVE (ENTER))
    (AND (CMPR (QUOTE ("$")))
      (OR (TESTS) (setq OK F))
      (OR (LOAD (CONS (QUOTE PROG)
                    (CONS (QUOTE NIL)
                      (CCNS (G1))
                      (CONS (QUOTE NIL)
                        (CCNS (G1))))
(SETQ OK F)) (NOT CK) TRUE) OK) OK)
(SETQ OK F))

(1 DEFINE (((LIST*SEQ (LAMBDA NIL (LEAVE (ENTER)
(AND (OR (AND (CMPR (QUOTE ('- '))
(OR (EXPQ) (SETQ OK F))
(OR (CMPR (QUOTE (' ')) (SETQ OK F))
(OR (AND (OR (AND (CMPR (QUOTE (',')))
(OR (LIST*SEQ) (SETQ CK F))
(OR (LOAD (CONS (QUOTE APPEND)
CONS (STAR2) (CONS (STAR1) NIL))))) (SETQ OK F)))
(NOT CK) TRUE) OK) (SETQ OK F))
(NOT CK)
(AND (EXPQ)
(OR (AND (OR (AND (CMPR (QUOTE (','))))
(OR (LIST*SEQ) (SETQ CK F)))
(NOT CK) (AND TRUE (OR (LOAD NIL) (SETQ OK F))))) OK)
(SETQ OK F))
(OR (LOAD (CONS (QUOTE CONS)
(CONS (STAR2) (CONS (STAR1) NIL)))
(SETQ OK F))))))

(1 DEFINE (((LIST*SEQ (LAMBDA NIL (LEAVE (ENTER)
(AND (CMPR (QUOTE ('- '))
(OR (AND (OR (LIST*SEQ)
(NOT OK) (AND TRUE (OR (LOAD NIL) (SETQ OK F)))))) OK)
(SETQ OK F)) (OR (CMPR (QUOTE (' ')) (SETQ OK F)))))))

(1 DEFINE (((EXPQ (LAMBDA NIL (LEAVE (ENTER)
(WHERE (FUNCTION EXPX) T)))))))

(1 DEFINE (((EXPQ (LAMBDA NIL (LEAVE (ENTER)
(WHERE (FUNCTION EXPX) F)))))))

(1 DEFINE (((LISTEXP (LAMBDA NIL (LEAVE (ENTER)
(AND (ELEMENT)
(OR (PROCG NIL MOO011 (COND ((AND (OR (AND (CMPR (QUOTE ('- '1)
))))
(OR (LOAD (CONS (QUOTE CAR) (CONS (STAR1) NIL)))
(SETQ OK F)))
(NOT OK)
(AND (CMPR (QUOTE ('- '2))))
(OR (LOAD (CONS (QUOTE CADR) CONS (STAR1) NIL)))
(SETQ OK F)))
(NOT OK)
(AND (CMPR (QUOTE ('- '3))))
(OR (LOAD (CONS (QUOTE CADDR) CONS (STAR1) NIL)))
(SETQ OK F)))
(NOT OK)
(AND (CMPR (QUOTE ('- '4))))
(OR (LOAD (CONS (QUOTE CADDOR) CONS (STAR1) NIL)))
(SETQ OK F)))
(NOT OK)
(AND (CMPR (QUOTE ('- '2))))
(OR (LOAD (CONS (QUOTE CDR) CONS (STAR1) NIL)))
(SETQ OK F)))
(NOT OK)
(AND (CMPR (QUOTE ('- '3))))
(OR (LOAD (CONS (QUOTE CDDR) CONS (STAR1) NIL)))
(SETQ OK F)))
(NOT OK)
(AND (CMPR (QUOTE ('- '4))))
(OR (LOAD (CONS (QUOTE CDDDR) CONS (STAR1) NIL)))
(SETQ OK F)))
(OK) (GO MOO011))) (RETURN OK))
(SETQ OK F))))))))

(1 DEFINE (((BASIC (LAMBDA NIL (LEAVE (ENTER)
(AND (LISTEXP)
(OR (AND (CR (AND (CMPR (QUOTE ("*")) (OR (BASIC) (SETQ OK F)) (OR (LCAD (CONS (QUOTE CONS) (CONS (STAR2) (CONS (STAR1) NIL)))) (SETQ OK F)))) (NOT OK) TRUE) OK) (SETQ OK F))))))

(1 DEFINE (((RELATION (LAMBDA NIL (LEAVE (ENTER)
(ANC (BASIC)
(OR (AND (CR (AND (CMPR (QUOTE (=)) (OR (BASIC) (SETQ OK F)) (OR (LCAD (CONS (QUOTE NOT) (CONS (CONS (QUOTE EQUAL)
(CONS (STAR2) (CONS (STAR1) NIL)))) NIL)))) (SETQ OK F)) (NOT OK) TRUE) OK) (SETQ OK F)))))))))

(1 DEFINE (((NEGATION (LAMBDA NIL (LEAVE (ENTER)
(ANC (NEGATION)
(OR (AND (CR (AND (CMPR (QUOTE (', 'A 's))) (OR (FACTOR) (SETQ OK F)) (OR (LCAD (CONS (QUOTE AND)
(CONS (STAR2) (CONS (STAR1) NIL)))) (SETQ OK F)))) (NOT OK) TRUE) OK) (SETQ OK F)))))))))

(1 DEFINE (((EXPX (LAMBDA NIL (LEAVE (ENTER)
(ANC (FACTOR)
(OR (AND (CR (AND (CMPR (QUOTE (', 'V 's))) (OR (EXPX) (SETQ OK F)) (OR (LCAD (CONS (QUOTE OK)
(CONS (STAR2) (CONS (STAR1) NIL)))) (SETQ OK F)))) (NOT OK) TRUE) OK) (SETQ OK F)))))))))

(1 DEFINE (((LCOPST (LAMBDA NIL (LEAVE (ENTER)
(ANC (CMPR (QUOTE (', 'L 'O 'G 'P)) (OR (CMPR (QUOTE (', U 'N 'I 'L))) (SETQ OK F)) (OR (GENI) (SETQ OK F)) (OR (RESTORE (SAVER)
(AND (OR (BACKUP (EXPNQ))) (AND"(ERRURX)
(CR (PROG NIL M00012 (COND (AND (NCOMP (QUOTE (', 'B 'E
'G 'I 'N)) (OK (DELETEx) (SETQ OK F))) (GO M00012))) (RETURN OK)) (SETQ OK F)))))) UK))

(SETQ OK F)) (OR (CMPR (QUOTE (', 'B 'E 'G 'I 'N)))) (SETQ OK F)) (OR (LCAD (CONS (QUOTE COND)
(CONS (CONS (STAR1)
(CONS (CONS (QUOTE GO) (CONS (GNI) NIL)) NIL)) NIL))))

(SETQ OK F)) (OR (PROG NIL M00013 (COND (RESTORE (SAVER)
(AND (OR (BACKUP (ST)))
(ANC (NCOMP (QUOTE (', 'E 'N 'D))))
(OR (GOBBLE) (SETQ OK F))) (GO M00013)))

(RETURN CK)) (SETQ OK F)) (OR (CMPR (QUOTE (', 'E 'N 'D)))) (SETQ OK F)) (OR (LCAD (CONS (QUOTE GO) (CONS (GNI) NIL)) (SETQ OK F)) (OR (GENI2) (SETQ OK F)))))))))

(1 DEFINE (((IFST (LAMBDA NIL (LEAVE (ENTER)

(ANC (IFST)
(OR (AND (CR (AND (CMPR (QUOTE ("*"))) (OR (BASIC) (SETQ OK F)) (OR (LCAD (CONS (QUOTE CONS) (CONS (STAR2) (CONS (STAR1) NIL)))) (SETQ OK F)))) (NOT OK) TRUE) OK) (SETQ OK F)))))))))

(1 DEFINE (((RELATION (LAMBDA NIL (LEAVE (ENTER)
(ANC (BASIC)
(OR (AND (CR (AND (CMPR (QUOTE (=)) (OR (BASIC) (SETQ OK F)) (OR (LCAD (CONS (QUOTE NOT) (CONS (CONS (QUOTE EQUAL)
(CONS (STAR2) (CONS (STAR1) NIL)))) NIL)))) (SETQ OK F)) (NOT OK) TRUE) OK) (SETQ OK F)))))))))

(1 DEFINE (((NEGATION (LAMBDA NIL (LEAVE (ENTER)
(ANC (NEGATION)
(OR (AND (CR (AND (CMPR (QUOTE (', 'A 's))) (OR (FACTOR) (SETQ OK F)) (OR (LCAD (CONS (QUOTE AND)
(CONS (STAR2) (CONS (STAR1) NIL)))) (SETQ OK F)))) (NOT OK) TRUE) OK) (SETQ OK F)))))))))

(1 DEFINE (((EXPX (LAMBDA NIL (LEAVE (ENTER)
(ANC (FACTOR)
(OR (AND (CR (AND (CMPR (QUOTE (', 'V 's))) (OR (EXPX) (SETQ OK F)) (OR (LCAD (CONS (QUOTE OK)
(CONS (STAR2) (CONS (STAR1) NIL)))) (SETQ OK F)))) (NOT OK) TRUE) OK) (SETQ OK F)))))))))

(1 DEFINE (((LCOPST (LAMBDA NIL (LEAVE (ENTER)
(ANC (CMPR (QUOTE (', 'L 'O 'G 'P)) (OR (CMPR (QUOTE (', U 'N 'I 'L))) (SETQ OK F)) (OR (GENI) (SETQ OK F)) (OR (RESTORE (SAVER)
(AND (OR (BACKUP (EXPNQ))) (AND"(ERRURX)
(CR (PROG NIL M00012 (COND (AND (NCOMP (QUOTE (', 'B 'E
'G 'I 'N)) (OK (DELETEx) (SETQ OK F))) (GO M00012))) (RETURN OK)) (SETQ OK F)))))) UK))

(SETQ OK F)) (OR (CMPR (QUOTE (', 'B 'E 'G 'I 'N)))) (SETQ OK F)) (OR (LCAD (CONS (QUOTE COND)
(CONS (CONS (STAR1)
(CONS (CONS (QUOTE GO) (CONS (GNI) NIL)) NIL)) NIL))))

(SETQ OK F)) (OR (PROG NIL M00013 (COND (RESTORE (SAVER)
(AND (OR (BACKUP (ST)))
(ANC (NCOMP (QUOTE (', 'E 'N 'D))))
(OR (GOBBLE) (SETQ OK F))) (GO M00013)))

(RETURN CK)) (SETQ OK F)) (OR (CMPR (QUOTE (', 'E 'N 'D)))) (SETQ OK F)) (OR (LCAD (CONS (QUOTE GO) (CONS (GNI) NIL)) (SETQ OK F)) (OR (GENI2) (SETQ OK F)))))))))

(1 DEFINE (((IFST (LAMBDA NIL (LEAVE (ENTER)

(ANC (IFST)
(OR (AND (CR (AND (CMPR (QUOTE ("*"))) (OR (BASIC) (SETQ OK F)) (OR (LCAD (CONS (QUOTE CONS) (CONS (STAR2) (CONS (STAR1) NIL)))) (SETQ OK F)))) (NOT OK) TRUE) OK) (SETQ OK F)))))))))
DECLARE [STAR, GEN, SKIP-BLANKS, LINE,
INPUT, COLUMN, BACK, CHR, FLAGX, LETTER, DIGIT,
ODGT, ID-V, COUNT, MAXIMUM, OK, TRUE];

LISP

PROCEDURE INITIALIZE [A, B];

IF A = QUOTE[TTY]
  BEGIN
  THEN RDSN[TTY];
  ELSE OPEN[A, QUOTE[DISC], QUOTE[PERM] ];
  RDS[A];
  END

IF B = QUOTE[TTY]
  BEGIN
  THEN WRSN[TTY];
  ELSE IF B = QUOTE[CORE]
    BEGIN
    THEN OPEN[ QUOTE[SDCSDC], QUOTE[DISC] ];
    RDS[QUOTE[SDCSDC] ];
    ELSE OPEN[ B, QUOTE[DISC] ]; WR[TTY];
    END
  END

LOCAL[X];

PROCEDURE COMPLETE[Z, A];

IF A
  BEGIN
  THEN ERROR[X];
  END

IF X = QUOTE[TTY]
  BEGIN
  THEN SHUT[X];
  END

IF X = QUOTE[SDCSDC] .A.
  BEGIN
  THEN POSITION[X, QUOTE[WEOF]] ;
  POSITION[X, QUOTE[REWIND] ];
  LOOP UNTIL LOADEXP[X] = QUOTE[EOF]
  BEGIN
  END
  SHUT[X, QUOTE[DELETE]];}
ELSE IF X = QUOTE[TTY]
  BEGIN
  THEN POSITION[X, QUOTE[WEOF]];}
  ELSE SHUT[X];
  END
  END

RETURN[A];
PROCEDURE READLN(\); LOCAL(X);
X := READCH(\); IF NULL(X) THEN RETURN(NIL); ELSE RETURN(X*READLN(\)); END

PROCEDURE PRINTLN(U); LOCAL(X);
X := PRINTLN(U); LOOP UNTIL NULL(X)
BEGIN PRINTCH(X.1) X := X+1; END

PROCEDURE NXTCHR(\);
IF NULL(INPUT) THEN COLUMN := COUNT; LINE := READLN(\); RPLACED(INPUT, LINE);
END

PROCEDURE LOAD(X);
INPUT := INPUT+1; CHR := INPUT.
COUNT := ADDI(COUNT); MAXIMUM := MAX(COUNT, MAXIMUM);
RETURN(\);

PROCEDURE COMPARE(S, N);
LOCAL(U, V, W, SIGNAL);
IF SKIP-BLANKS THEN COLUMN := COUNT; LINE := READLN(\);
END

BEGIN
END
RETURN(SIGNAL);

If N = 3 THEN SIGNAL := -SIGNAL; INPUT := V;
CHR := INPUT.1; COUNT := W;
ELSE
IF SIGNAL THEN BEGIN
END
ELSE INPUT := V; CHR := INPUT.1; COUNT := W;
END
END
RETURN(SIGNAL);

005500-PROCEDURE READLN()); LOCAL(X);
005600-X := READCH());
005700-IF NULL(X) BEGIN THEN RETURN(NIL); ELSE RETURN(X*READLN()); END
005900-RETURN();
006300-PROCEDURE PRINTLN(U); LOCAL(X);
006400-X := PRINTLN(U);
006500-LOOP UNTIL NULL(X)
006600-BEGIN PRINTCH(X.1) X := X+1; END
006800-RETURN();
006900-TERPRI();
007000-RETURN();
007100-PROCEDURE NXTCHR();
007200-IF NULL(INPUT) THEN COLUMN := COUNT; LINE := READLN(); RPLACED(INPUT, LINE);
007400-BEGIN END
007600-RETURN(\);
007800-PROCEDURE LOAD(X);
008000-STAR := X * STAR;
008200-RETURN(\);
008300-PROCEDURE COMPARE(S, N);
008400-IF SKIP-BLANKS THEN COLUMN := COUNT; LINE := READLN(); RPLACED(INPUT, LINE);
008600-BEGIN END
008800-RETURN(\);
009000-BEGIN COLUMN := COUNT; LINE := READLN(); RPLACED(INPUT, LINE);
009200-BEGIN U := S; V := INPUT; W := COUNT;
009400-BEGIN END
009600-BEGIN END
009800-BEGIN IF N = 3 THEN SIGNAL := -SIGNAL; INPUT := V; CHR := INPUT.1; COUNT := W;
009900-BEGIN ELSE IF SIGNAL THEN BEGIN
010300-BEGIN IF SIGNAL THEN BEGIN
010800-BEGIN IF N = 2 THEN BEGIN
010900-BEGIN THEN STAR := APPEND(STAR.1, S) * STAR2;
011000-BEGIN ELSE INPUT := V; CHR := INPUT.1; COUNT := W;
011200-BEGIN END
211300-BEGIN END
211400-RETURN(SIGNAL);
PROCEDURE CMPR2(X);
011500  IF CMPR(X)
011600  BEGIN ...
011800  THEN STAR := COMPRESSION(X) * STAR ; RETURN(T);
011900  END
012000  RETURN(F);
012100  PROCEDURE CMPR(S);
012200  RETURN([COMPARE(S1)]);
012300  PROCEDURE CMPS(S1);
012400  RETURN([COMPARE(S2)]);
012500  PROCEDURE NCMPS(S1);
012600  RETURN([COMPARE(S3)]);
012700  PROCEDURE ERRORX(); LOCAL(X);
012800  X := WSNIL;
012900  PRINTLN(LINE); BLANKS[SUB1( DIFFER [MAXIMUM, COLUMN])];
013000  PRINTCH(#); TERNPRT();
013100  WSN(X); OK := T;
013200  RETURN(T);
013300  PROCEDURE MARK();
013400  LOOP UNTIL CHR = #
013500  BEGIN
013600  NITCHR();
013700  END
013800  SKIP-BLANKS := F;
013900  STAR := NIL * STAR ;
014000  RETURN;
014100  PROCEDURE TOKEN(W, X);
014200  SKIP-BLANKS := T ;
014300  IF X
014400  BEGIN
014500  THEN STAR := STAR:2 ;
014600  END
014700  RETURN(X);
014800  PROCEDURE INSERT(S);
014900  STAR := APPEND([STAR1], S) * STAR:2 ;
015000  RETURN(T);
015100  PROCEDURE STAR1(); LOCAL(X);
015200  X := STAR:1 ; STAR := STAR:2;
015300  RETURN(X);
015400  PROCEDURE STAR2(); LOCAL(X);
015500  X := STAR:2 ; STAR := STAR:1 * STAR:2 ;
015600  RETURN(X);
015700  PROCEDURE STAR3(); LOCAL(X);
015800  X := STAR:3 ; STAR := STAR:1 * STAR:2 * STAR:4 ;
015900  RETURN(X);
016000  PROCEDURE STAR4(); LOCAL(X);
016100  X := STAR:4 ; STAR := STAR:1 * STAR:2 * STAR:3 * STAR:4:2 ;
016200  RETURN(X);
016300  PROCEDURE FLAGS();
016400  FLAGX := STAR * FLAGX ; STAR := NIL;
016500  RETURN;
016600  PROCEDURE SEQ(W, X);
016700  IF X
016800  BEGIN
016900  THEN STAR := REVUSERE([STAR] * FLAGX1);
016910  ELSE STAR := FLAGX1;
017000  END
017100  FLAGX := FLAGX2 ;
017200  RETURN(X);
PROCEDURE GEN1();
BEGIN
THEN GEN := GENSYM() * GEN2;
END
RETURN();
PROCEDURE GEN2();
BEGIN
THEN GEN := GEN1 * GENSYM() * GEN3;
END
RETURN();
PROCEDURE GENIC();
STAR := GNIC * STAR;
RETURN();
PROCEDURE GEN2IC();
STAR := APPEND(STAR1, [CHR]) * STAR2;
RETURN();
PROCEDURE ANYC();
STAR := APPEND(STAR1, STAR2);
RETURN();
PROCEDURE DELETEX();
NXTCHR();
RETURN();
PROCEDURE MAKEATOM();
STAR := COMPRESS(STAR1) * STAR2;
RETURN();
PROCEDURE MAKENUMBER();
LOCAL(S, N);
S := STAR;
LOOP UNTIL NULL[S];
BEGIN
N := PLUS(TIMES(N, 10), CHR20(T[S])); S := S2;
END
RETURN();
PROCEDURE COMPILE();
PRINT1; STAR := STAR2;
RETURN();
PROCEDURE WHERE();
LOCAL(A, B);
IF SIGNAL THEN
BEGIN
IF Y THEN
BEGIN
END
END
NXTCHR();
END
RETURN();
PROCEDURE MAKECHR();
STAR := STAR1.1 * STAR2;
RETURN();
PROCEDURE ISITIX();
LOCAL(SIGNAL);
SIGNAL := NOT(0LOGAND(X, CONVERTCHR)));
```lisp
(SESAME

LAP()

[(CHR20G SUBR1)(BAX($ 2)1)(O(E CHR2GCT)1)
 (LDA $A 0 370Q)(SUB 144 0 100370Q)(LDM $A)
 (BSX *MKNO 2 5)(BUC 0 4))]

())

LAP()

[(CONVERT SUBR 1)(BAX($ 2)1)(O(E CONVERT)1)
 (SUB 144 0 100370Q)(LDA CONVERT 17Q)(BSX *MKNO 2 5)
 (BUC 0 4)]

 (5Q)(5Q)(41Q)(1Q)(1Q)(41Q)(41Q)(41Q)
 (1Q)(3Q)(3Q)(3Q)(3Q)(3Q)(3Q)(3Q)
 (3Q)(3Q)(41Q)(1Q)(1Q)(41Q)(41Q)(41Q)
 (1Q)(3Q)(3Q)(3Q)(3Q)(3Q)(3Q)(3Q)
 (3Q)(3Q)(41Q)(1Q)(1Q)(41Q)(41Q)(41Q)
 (21Q)(1Q)(3Q)(3Q)(3Q)(3Q)(3Q)(3Q)
 (3Q)(3Q)(41Q)(1Q)(1Q)(41Q)(41Q)(41Q)

)))

DEFINE(( (GENCH1(LAMBDA()) (PROG())
        (*PLANT (LGCGR 201Q13 777744Q) 41242Q)
        (*PLANT 0 41251Q)))) ))

DEFINE((
        (MAGIC(LAMBDA(X) (PROG(W Y)
            (OPEN X (QUOTE DISC) (QUOTE PERM))
            (SETQ W (RCS X))
            (A1 (COND ((NOT (EQ (LCASEXP X) (QUOTE EDF))) (GO A1)))
            (A2 (RUS W)(SHUT X)
                (RETURN Y ) )))))) ))

ECF CARD ALL PROGRAM COMPATIBLE

Library

Routines written directly in

Lisp 1.5
(1 CSET (STAR NIL))
(1 CSET (GEN NIL))
(1 CSET (SKIP-BLANKS NIL))
(1 CSET (LINE NIL))
(1 CSET (INPUT NIL))
(1 CSET (COLUMN NIL))
(1 CSET (BACK NIL))
(1 CSET (CHR NIL))
(1 CSET (FLAGX NIL))
(1 CSET (LETTER NIL))
(1 CSET (DIGIT NIL))
(1 CSET (OCR1 NIL))
(1 CSET (IC-W NIL))
(1 CSET (CCOUNT NIL))
(1 CSET (MAXIMUM NIL))
(1 CSET (OK NIL))
(1 CSET (TRUE NIL))
(1 DEFINE ((((INITIALIZE (LAMBDA (A B)
        (PROC NIL (COND ((NOT (EQUAL A (QUOTE TTY))) (GO MO0001)))
            (RDS NIL))
            (GO MOOC02)
            (GO MOOC04)
            (GO MOOC06))))
        (SETQ STAR NIL))
        (SETQ GEN NIL))
        (SETQ SKIP-BLANKS T)
        (SETQ COLUMN 0)
        (SETQ LINE (READLN))
        (SETQ INPUT LINE)
        (SETQ CHR (CAR INPUT))
        (SETQ BACK NIL)
        (SETQ CCOUNT 1)
        (SETQ MAXIMUM 1)
        (SETQ OK T)
        (SETQ FLAGX NIL)
        (SETQ LETTER 2)
        (SETQ DIGIT 12) (SETQ OUGT 8) (GENCH1) (SETQ TRUE T)))))
(1 DEFINE (((COMPLETE (LAMBDA (Z A))
        (PROC X)
            (COND ((NOT (NOT A)) (GO MO0007)))
            (ERRDX)))
            (GO MOOC07)
            (GO MOOC08))
            (GO MOOC09))
            (GO MOOC10))
            (GO MOOC11))
            (GO MOOC12))
            (GO MOOC09))
            (GO MOOC09)))
(DEFINE (DEFINE (MARK (LAMBDA NIL (PROG NIL MOOC30 (COND (( NOT (EQUAL CHR ' ) ) (GO MOOC30)))
(NXCHR)
(GO MOOC30))
MOOC31 (SETQ SKIP*BLANKS F) (SETQ STAR (CONS NIL STAR))))))
(DEFINE (TCKEN (LAMBDA (W X)
(PROG NIL (SETQ SKIP*BLANKS T)
(COND ((NOT (NOT X)) (GO MOOC32)))
(SETQ STAR (CDR STAR)) MOOC32 (RETURN X))))))
(DEFINE (INSERT (LAMBDA (S)
(PROG NIL (SETQ STAR (CONS (APPEND (CAR STAR) S) (CDR STAR)))
(RETURN T))))))
(DEFINE (DEFINE (ISTR1 (LAMBDA NIL (PRCG X)
(SETQ X (CAR STAR)) (SETQ STAR (CDR STAR)) (RETURN X))))))
(DEFINE (DEFINE (ISTR2 (LAMBDA NIL (PRCG X)
(SETQ X (CDR STAR))
(SETQ STAR (CONS (CAR STAR) (CDR STAR))) (RETURN X))))))
(DEFINE (DEFINE (ISTR3 (LAMBDA NIL (PRCG X)
(SETQ X (CDR STAR))
(SETQ STAR (CONS (CAR STAR) (CONS (CDR STAR) (CDR (CDR STAR))))
(RETURN X))))))
(DEFINE (DEFINE (ISTR4 (LAMBDA NIL (PRCG X)
(SETQ X (CDR STAR))
(SETQ STAR (CONS (CAR STAR)
(CONS (CDR STAR) (CONS (CDR STAR) (CDR (CDR STAR))))
(RETURN X)))))
(DEFINE (DEFINE (FLAGS (LAMBDA NIL (PRCG NIL (SETQ FLAGX (CONS STAR FLAGX)
(SETQ STAR NIL)))))))
(DEFINE (DEFINE (SEQ (LAMBDA (W X)
(PROG NIL (COND ((NOT X) (GO MOOC33)))
(SETQ STAR (CONS (REVERSE STAR) (CAR FLAGX)))
(GO MOOC34))
MOOC33 (SETQ STAR (CAR FLAGX))
MOOC34 (SETQ FLAGX (CDR FLAGX)) (RETURN X))))))
(DEFINE (DEFINE (IGN1 (LAMBDA NIL (PRCG NIL (COND ((NOT (NULL (CAR GEN)))
(GO MOOC35)))
(SETQ GEN (CONS (GENSYM) (CDR GEN)))
MOOC35 (RETURN (CAR GEN)))))))
(DEFINE (DEFINE (IGN2 (LAMBDA NIL (PRCG NIL (COND ((NOT (NULL (CDR GEN)))
(GO MOOC36)))
(SETQ GEN (CONS (CAR GEN) (CONS (GENSYM) (CDR GEN))))
MOOC36 (RETURN (CDR GEN)))))))
(DEFINE (DEFINE (IGN1 (LAMBDA NIL (PRCG NIL (SETQ STAR (CONS (IGN1) STAR))
(RETURN T))))))
(DEFINE (DEFINE (IGN2 (LAMBDA NIL (PRCG NIL (SETQ STAR (CONS (IGN2) STAR))
(RETURN T))))))
(DEFINE (DEFINE (ANY (LAMBDA NIL (PRCG NIL (SETQ STAR (CONS (APPEND (CAR STAR) (CONS (CHR NIL)) (CDR STAR))) (NXCHR) (RETURN T))))))
(DEFINE (DEFINE (DELETEX (LAMBDA NIL (PRCG NIL (NXCHR)) (RETURN T))))))
(DEFINE (DEFINE (MAKEATOM (LAMBDA NIL (PRCG NIL (SETQ STAR (CONS
cmpress (CAR STAR)) (CDR STAR))) (RETURN T))))))
(DEFINE (DEFINE (MAKELENBER (LAMBDA NIL (PRCG (S N)
(SETQ S (CAR STAR))
(SETQ N 0))
MOOC37 (COND ((NULL S) (GO MOOC38)))
(SETQ N (PLUS (TIMES N 10) (CHR2OC (CAR S))))
(SETQ S (CDR S))
(GO MOOC37))
MOOC38 (SETQ STAR (CONS N (CDR STAR)) (RETURN T))))))
(DEFINE (DEFINE (CCMPILE (LAMBDA NIL (PRCG NIL (PRINT (CONS ) (CAR STAR))))
(DEFINiE ((PRINTLN (LAMBDA (X) (COND ((NOT (EQUAL X NIL)) (GO MOO045)))
(RETURN NIL))
(MOOC45 (SETQ X (PRINTLN1 (CHR U))))
(COND ((NOT (AND (EQUAL X NIL) (EQUAL (CAR U) '?)))
(GO MOO046)))
(RETURN NIL) MOO046 (RETURN (CONS (CAR U) X))))))
(DEFINiE (((SAVER (LAMBDA NIL (PRCG NIL (SETQ BACK (CONS (CONS INPUT
(CONS STAR (CONS FLAGX (CONS COUNT NIL)))) BACK)))))))))
(DEFINiE (((RESTORE (LAMBDA (W X))
(PRiNG (SETQ BACK (CAR BACK)) (RETURN X)))))))
(DEFINiE (((BACKUP (LAMBDA (X))
(PRiNG (A))
(SEtQ A (AND X CK))
(COND ((NOT (NOT OK)) (GO MOO047)))
(SEtQ INPUT (CAR (CAR BACK))
(SEtQ STAR (CADR (CAR BACK))
(SEtQ FLAGX (CADDR (CAR BACK))
(SEtQ COUNi (CADDDR (CAR BACK)))
(SEtQ CHR (CAR INPUT)) (SETQ OK T) MOO047 (RETURN A)))))))
(DEFINiE (((ENTER (LAMBDA NIL (PRCG NIL (SETQ GEN (CONS NlL (CONS
Nl GEN))))))))
(DEFINiE (((LEAVE (LAMBDA (X Y))
(PRiNG (SETQ GEN (CDR GEN)) (RETURN Y)))))))
(DEFINiE ((((PRINT (LAMBDA (X))
(PRiNG (Z))
(SEtQ Z X)
(MOOC48 (CONS ((EQUAL Z NIL) (GO MOO049)))))
(PRINTCHAR (CAR Z)) (SETQ Z (CDR Z)) (GO MOO048) MOO049))))

ECF CARD ALL PROGRAM COMPATIBLE