Do Not Borrow DP

Artificial Intelligence Project--RLE and MIT Computation Center
Memo 22A--Character-Handling Facilities in the LISP System

by

Paul Abrahams

January 27, 1961

Introduction

Because of the new read program, a number of facilities are being added to the LISP system to permit manipulation of single characters and print names. Machine-language functions have been provided for breaking print names down into a list of their characters, for forming a list of characters into a print name, for creating a numerical object from a list of its characters, for reading in characters one by one from an input medium, and for testing characters to see whether they are letters, numbers, operation characters, etc. A number of auxiliary objects and subroutines are also described in this memo.

Characters and Character Objects

Each of the 64 6-bit binary numbers corresponds to a BCD character, if we include illegal characters. Therefore, in order to manipulate these characters via LISP functions, each of them has a corresponding object. Of the 64 characters, 48 correspond to characters on the keypunch, and the print name of an object corresponding to a keypunch character is simply that character. The print names of the remaining characters will be described later. When a LISP function is described which has a character as either value or argument, we really mean that it has an object corresponding to a character as value or argument respectively.

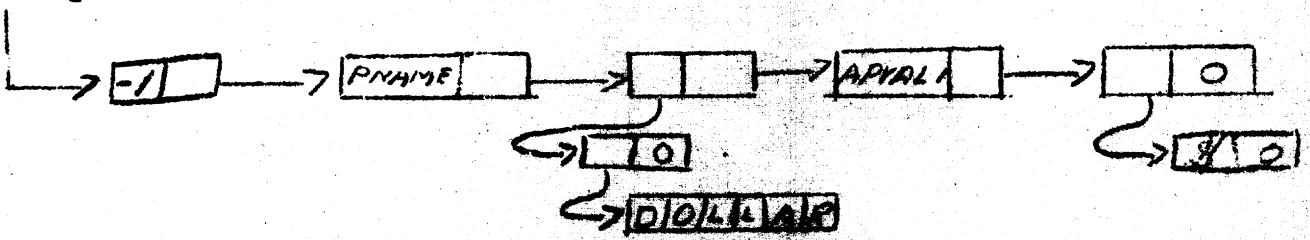
The first group of legal characters are the letters of the alphabet from A to Z. Each letter is a legitimate atomic symbol, and therefore may be referred to in a straightforward way, without ambiguity.

The second group of legal characters are the digits from 0 to 9. These must be handled with some care, because if a digit is considered as an ordinary integer rather than a character, a new non-unique object will be created corresponding to it, and this object will not be the same as the character object for the same digit, even though it has the same print name. Since the character-handling programs depend on the character objects being in specific locations, this will lead to error.

Both the current read program RDA and the new read program have been arranged so that digits 0 through 9 read in as the corresponding character objects. These may be used in arithmetic just like any other number, but even though the result of an arithmetic operation lies between 0 and 9, it will not point to the corresponding character object. Thus character objects for 0 through 9 may be obtained only by reading them or by manipulation of print names.

The third group of legal characters are the special characters. These correspond to the remaining characters on the keypunch, such as "$" and "=". Since these characters are not legitimate atomic symbols, there is a set of special character value objects which can be used to refer to them.

A typical special character value object, say DOLLAR, has the following structure:



Thus "DOLLAR" has value "$". Note that "$" is not a legitimate atomic symbol as far as the read program RDS is concerned, so that there is no way of reading in (QUOTE,$) via RDA.

The special character value objects and their permanent values are:

DOLLAR	$
SLASH	/
LPAR	(
RPAR	)
COMMA	,
PERIOD	.
PLUSS	+
DASH	- (11 punch)
STAR	*
BLANK	blank
EQSIGN	=

The following examples illustrate the use of these objects and their raison d'être. Each example consists of a triplet for the APPLY operator followed by the result.

Examples:

EVAL (DOLLAR) ()) = $

EVAL ((PRINT PERIOD) ()) = blank and causes "." to be printed.

The remaining characters are all illegal as far as the key-punch is concerned. The two characters corresponding to $12_{(8)}$ and $72_{(8)}$ have been reserved for end-of-file and end-of-record respectively. The end-of-file character has print name EOF and the end-of-record character has print name EOR; corresponding to these character objects are two character value objects EOR and EOF, whose values are EOR and EOF respectively. The rest of the illegal character objects have print names corresponding to their octal representations preceded by $IL and followed by $. For instance, the character $77_{(8)}$ corresponds to a character object with pirnt name $IL77$.

The character objects are arranged in the machine so that their first cells occupy successive storage locations. Thus it is possible to go from a character to the corresponding object or conversely by a single addition or subtraction. This speeds up character-handling considerably, because it isn't necessary to search property lists of character objects for their print names: the names may be deduced from the object locations.

Packing and Unpacking Characters

When a sequence of characters is to be made into either a print name or a numerical object, the characters must be put one by one into a buffer called BOFFO. BOFFO is used to store the characters until they are to be combined. It is not available explicitly to the LISP programmer, but the character-packing functions are described in terms of their effects on BOFFO. At any point, BOFFO contains a sequence of characters. Each operation on BOFFO either adds another character at the end of the sequence or clears BOFFO, i.e., sets BOFFO to the null sequence. The maximum length of the sequence is 120 characters; an attempt to add more characters will cause an error.
The character-packing functions are:

1. pack [c]:
   The argument of pack must be a character object. Pack adds the character c at the end of the sequence of characters in BOFFO. The value of pack is NIL.
2. clearbuff []:
   Clearbuff is a function of no arguments. It clears BOFFO and has value NIL. The contents of BOFFO are undefined until a clearbuff has been performed.
3. mknam []:
   Mknam is a function of no arguments. Its value is a list of full words containing the characters in BOFFO in packed BCD form. The last word is filled out with the illegal character code 77 if necessary. After mknam is performed, BOFFO is automatically cleared. Note that intern [mknam[]] yields the object whose print name is in BOFFO.
4. numob []:
   Numob is a function of no arguments. Its value is the numerical object represented by the sequence of characters in BOFFO. Numob uses the subroutine NUMBR

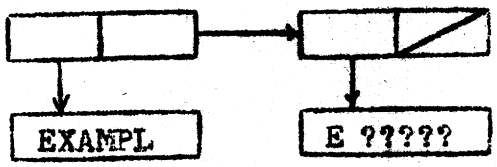
to obtain the number from its BCD representation, and the precise conventions on numbers will be given later when NUMBR is described. Numob will accept floating-point decimal numbers, decimal integers, fixed-point decimal numbers, and octal numbers. Octal numbers created by numob have a pointer to the object OCTAL on their property lists. They are considered as 36-bit rather than 35-bit numbers, so that applying numob to a negative octal number leads to the creation of a single numerical object which contains a pointer to a full word with 1 in the sign bit, rather than leading to a pair whose first element is MINUS, as is the case with negative decimal numbers. (Positive decimal integers from 0 to 9 are converted so as to point to the corresponding character object.)

5. unpack [x]:

This function has as argument a pointer to a full word. Unpack considers the full word to be a set of 6 BCD characters, and has as value a list of these characters ignoring all characters including and following the first 77.

6. intern[pname]

This function has as argument a pointer to a PNAME type structure such as -



Its value is the atomic symbol having this print name. If it does not already exist, then a new atomic symbol will be created.

The Character-Classifying Functions

1. liter [c]:

   Liter has as argument a character object. Its value is T if the character is a letter of the alphabet, and F otherwise.

2. digit [c]:

   Digit has as argument a character object. Its value is T if the character is a digit between 0 and 9, and F otherwise.

3. opchar [c]:

   Opchar has as argument a character object. Its value is T if the character is +, -, /, *, or =, and F otherwise. Opchar treats both minus signs equivalently.

4. dash [c]:

   Dash has as argument a character object. Its value is T if the character is either an 11-punch minus or an 8-4 punch minus, and F otherwise.

The Character-Reading Functions

The character-reading functions make it possible to read characters one by one from data records. The choice of input medium is determined by sense switch 1, though at a future time mode-set facilities may be added so that input may be taken from any medium. This description will be given in terms of hollerith cards as input; the procedure for tape is completely analogous. However, the tape records must be 72 characters long.

There is an object CURCHAR whose value is the character most recently read (as an object). There is also an object CHARCOUNT whose value is an integer object giving the column just read on the card, i.e., the column number of the character given by CURCHAR: There are three functions which affect the value of CURCHAR:

1. startread []:

   Startread is a function of no arguments which causes a new card to be read. The value of startread is the first character on that card, or more precisely, the object corresponding to the first character on the card. If an end-of-file condition exists, the value of startread is EOF. The value of CURCHAR becomes the same as the output of startread, and the value of CHARCOUNT becomes 1. Both CURCHAR and CHARCOUNT are undefined until a startread is performed. A startread may be performed before the current card has been completely read.

2. advance []:

   Advance is a function of no arguments which causes the next character to be read. The value of advance is that character. After the 72nd character on the card has been read, the next advance will have value EOR. After reading EOR, the next advance will act like a startread, i.e., will read the first character of the next card unless an end-of-file condition exists. The new value

of CURCHAR is the same as the output of advance; executing advance also increases the value of CHARCOUNT by 1. However, CHARCOUNT is undefined when CURCHAR is either EOR or EOF.

3. endread []:

Endread is a function of no arguments which causes the remainder of the card to be read and ignored. Endread sets CURCHAR to EOR and leaves CHARCOUNT undefined; the value of endread is always EOR. An advance following an endread acts like a startread. If CURCHAR already has value EOR and endread is performed, CURCHAR will remain the same and endread will, as usual, have value EOR.

Other LISP Functions

1. errorl []:

Errorl is a function of no arguments and has value NIL. It should be executed only while reading characters from a card (or tape). Its effect is to make the character just read, i.e., CURCHAR, so that when the end of the card is reached, either by successive advances or by an endread, the entire card is printed out along with a visual pointer to the defective character. For a line consisting of ABCDEFG followed by blanks, a pointer to C would look like this:

```
  V
ABCDEFG
  A
```

If errorl is performed an even number of times on the same character, the A will not appear. If errorl is performed prior to the first startread or while CURCHAR has value EOR or EOR, it will have no effect. Executing a startread before the current card has been completed will cause the errorl printout to be lost. The card is considered to have been completed when CURCHAR has been set to EOR. Successive endreads will cause the errorl printout to be reprinted. Any number of characters in a given line may be marked by errorl.

2. numnam [x]:

The argument x of numnam is a pointer to a cell containing a number. Evaluating numnam will cause that number, considered as a signed decimal integer, to be printed without terminating the print line. The value of numnam is NIL.

Subroutine NUMBR

This subroutine will convert a sequence of characters in packed BCD form into the number which they represent. The input conventions are generally those of SAP-FAP decimal input; however, provision is made for handling octal numbers also. This subroutine should not be used as a LISP function because the conventions on input and output are different from normal LISP usage. The routine is a modification of UA DBC1.

The routine will accept floating-point decimal numbers, decimal integers, fixed-point decimal numbers, and octal numbers. For decimal numbers of the three different types mentioned, the conventions are those of the DEC pseudo-operation in FAP. However, the routine will convert only one number each time it is entered; it considers a decimal number to be terminated when it encounters any character other than a digit, ".", "+", "-", "E", or "B". "Q" must not appear immediately after a decimal integer.

Octal numbers are followed by a Q, and may be preceded by a sign. The "Q" may be followed by an unsigned scale factor n, which will cause the number to be shifted n octal places to the left. (n is itself a decimal number.) The number is right-justified, and overflows on the left are ignored. The termination rules are the same as for decimal numbers, although a second "Q" will not act as a terminator.

Some examples of octal conversion are:

Input	Result
-75Q	4 00000 0 00075
+63Q4	0 00000 6 30000
77Q10	7 70000 0 00000
-3Q4	4 00000 0 30000

Note that in both octal and decimal numbers, the two different minus signs are equivalent; and if the initial character is a terminator, the result is a decimal fixed-point zero.

The input sequence of characters must be packed into successive words in storage, 6 characters per word. However, the first significant character need not be the first character of the word in which it occurs.

In order to tell whether the input number is octal or decimal, the program first scans the characters in sequence until it encounters a character other than a digit less than 8. If the first such character is Q, then the number is taken as octal; otherwise it is taken as decimal. This rule accounts in part for the peculiar behavior of "Q" as a terminator.

The calling sequence is as follows:

Let L be the location of the first packed word, and n the location of the first significant character within the word, counting from the left, with $1 \leq n \leq 6$. Then the following word should be placed in the AC:

```
PZE  L,,n
```

The subroutine is entered by the instruction

```
TSX  $NUMBR,4
```

The resulting number appears in the MQ. The AC will contain L and n for the character following the terminating character, so that for reading numbers separated by commas, say, the subroutine may be entered several times in succession, and the AC need not be reloaded each time. The sign of the AC will be - for floating-point decimal numbers and + for all other numbers; however, the sign of the AC is ignored when the routine is entered.

CS-TR Scanning Project
Document Control Form

Date : 11/30/95

Report # AIM-22w

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☐ Technical Report (TR) ☒ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 12(16-images)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☒ Single-sided or
☐ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☒ Other: MIMEOGRAPH (POOR)

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:
IMAGE MAP: (1-12) TITLE PAGE, 2-10
(13-16) SCANCONTROL TARGETS (3)

Scanning Agent Signoff:
Date Received: 11/30/95 Date Scanned: 12/2/95 Date Returned: 12/14/95

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 12/12/1995

M.I.T. Libraries
Document Services