F.G.H.
3/12/85

LispCourse #12: CLISP

CLISP

Introduction

CLISP (Conversational LISP) is a package that implements an alternative syntax for Interlisp expressions. CLISP is intended to make Interlisp "easier to read and write, especially for beginners".

An example of CLISP is the expression *A_5*. According to standard Interlisp, this is simply an atom whose name is 3 characters long. However, according to CLISP it is alternative syntax for *(SETQ A 5)*.

Another example might be: *A*5+(6/2)*. According to standard Interlisp, this is simply an atom whose name is 4 characters long followed by a list containing a single atom. However, according to CLISP it is alternative syntax for *((PLUS (TIMES A 5) (QUOTIENT 6 2)))*.

CLISP is part of DWIM. It works as follows:

> Expressions that the Lisp interpreter cannot intepret due to u.b.a (unabound atom) or u.d.f. (undefined function) errors are passed to DWIM.
>
> DWIM checks to see if the "error" is in fact a legal CLISP expression. If so, it translates the CLISP to standard Lisp and returns the standard expression to the evaluator.
>
> Otherwise, DWIM tries to automatically correct the error. DWIM knows about CLISP as well as standard Lisp. Therefore, it will try to "correct" expression that it thinks are CLISP expressions with typos. The "automatic correction" of CLISP works just like the "automatic correction" of standard Lisp described last time.
>
> If the expression isn’t CLISP and can’t be corrected, it is passed to the error handler.

For example, when the user types *"A_5"*, a u.b.a error occurs because A_5 is an atom with no value. The expression "A_5" is passed to DWIM/CLISP which

recognizes it as a CLISP expression. CLISP translates the expression into "(SETQ A 5)" which is then evaluated. ***The result is the same as if (SETQ A 5) had been typed in.***

Note this all is a bit wacky!!! From the user's point of view, CLISP expressions are simply Lisp expressions written in an alternative syntax. They are in no way "errors". From the system's point of view a legal CLISP expression is an error that is "corrected" by the CLISP translator in DWIM.

CLISP has four basic components:

1. **CLISP character operators (including infix operators):** CLISP provides an alternative syntax based on a series of special characters. For example, CLISP provides infix arithmetic expressions such as *"A+B"*, where the *"+"* is a CLISP character operator. CLISP uses the *"+"* as a clue to translate this expression into *(PLUS A B)*.

2. **IF-THEN-ELSE statements:** CLISP provides a "more intuitive" syntax for the COND statement that is similar to the IF-THEN-ELSE statements in PASCAL and FORTRAN. CLISP just translates these statements into their equivalent COND forms.

3. **Iterative statements:** The FOR and WHILE loops discussed in Session #5 are implemented in CLISP. CLISP translates these iterative statements into a series of standard Lisp statements that carry out the iteration.

4. **Record Package:** The Record Package is a package that implements various data structures in addition to lists. We will cover the Record Package in detail in the programming section of this course.

We will not discuss the Record Package until much later in the course.

We have already covered the Iterative statements (i.e., FOR and WHILE statements) in detail. The fact that they are CLISP rather than standard Lisp has little effect on the user. In most interactions with the system, the user will see only FOR or WHILE statements and not their translations into standard Lisp. Exceptions to this rule do exist, however. So you should remember that FOR and WHILE loops are in fact CLISP and not standard Lisp expressions.

The CLISP character operators and the IF-THEN-ELSE statements are discussed in detail below. The fact that these are CLISP rather than standard Lisp is very evident to the user.

During the translation from CLISP to Lisp, the Lisp expression actually replaces the CLISP statement. All future accesses to the expression return the translation (i.e., Lisp) rather than the original (i.e., CLISP). For example, the typed-in expression *A_5* is replaced by the Lisp expression *(SETQ A 5)*. *A_5* is forgotten. In particular, the entry on the history list is *(SETQ A 5)* and NOT *A_5*. *"FIX SETQ"* will work but *"FIX A_5"* will not.

CLISP Character Operators

CLISP interprets several characters as operators to be applied to the surrounding S-expressions. Most of the CLISP operators are ***infix*** operators such as the "+" in *A+B*. There are some ***prefix*** operators such as the "~" in the expression *~(NUMBERP X)*.

In a CLISP expression, an operator can be surrounded by spaces or it can be in the middle of an atom. To CLISP, *"A+B"* is equivalent to *"A + B"*.

CLISP uses these operators as the basis for translating CLISP expressions into the appropriate Lisp function calls.

Arithmetic Operators:

+, -, *, /, ^ ž these are the CLISP infix arithmetic operators. They are translated into the appropriate calls to PLUS, DIFFERENCE, TIMES, QUOTIENT, and EXPT.

Examples:

4/2 translates to (QUOTIENT 4 2)

(4+5)*3 translates to (TIMES (PLUS 4 5) 3)

(A+(SETQ A 5))*(SETQ A 7)) translates to
(TIMES (PLUS A (SETQ A 5)) (SETQ A 7))

Parentheses can be used to insure that the operators are interpreted in the correct order. For example: *5*4+6* can be either *(5*4)+6* versus *5*(4+6)*.

In the absence of parentheses, normal rules of precedence are used. Thus, ^ is higher than ***** and / which in turn are higher than **ÿ** and **+.** Within a precedence level, precedence is given to the leftmost operator.

Examples:

*4*2+5* translates to *(PLUS (TIMES 4 2) 5)*
[rather than to (TIMES 4 (PLUS 2 5))]

4+2-3 translates to *(DIFFERENCE (PLUS 4 2) 3)*
[rather than to (PLUS 4 (DIFFERENCE 2 3))]

*4*x^2* translates to *(TIMES 4 (EXPT X 2))*
[rather than to (EXPT (TIMES 4 X) 2)]

ÿ (i.e., unary minus) ž the minus sign can also be used as a prefix unary minus to indicate arithmetic negation. Since the minus sign is both a binary (infix) and a unary (prefix) operator the following rule is in effect: the minus sign is interpreted as a binary minus except when it is the first item in a list or it follows another operator. Examples: *ÿA, (ÿA), (B*ÿA).*

Logical Operators:

=, GT, LT, GE, LE ž are infix operators for EQUAL, GREATERP, LESSP, "Greater than or equal to", and "Less than or equal to". These are translated to a call to the appropriate predicate or predicate composition.

Examples:

A GT B translates to *(GREATERP A B)*

A=B translates to *(EQUAL A B)*

A LE B translates to *(OR (EQUAL A B) (LESSP A B))*

Note that except for the =, all of these operators must be surrounded by spaces. For example, *A GT B* cannot be written *AGTB*. However, *A = B* can be written *A=B*.

AND, OR, MEMBER, EQUAL ž are infix operators standing for the Lisp functions of the same name. They are translated accordingly.

Examples:

A EQUAL B translates to *(EQUAL A B)*

A OR B translates to *(OR A B)*

A MEMBER B translates to *(MEMBER A B)*

~ (unary) ž is prefix operator meaning NOT (or NULL).

Examples:

~(MEMBER A '(1 2 3)) translates to
(NULL (MEMBER A '(1 2 3)))

~(A OR B) translates to *(NULL (OR A B))*

Note: Parentheses can be used to insure proper interpretation of logical combinations: *A OR B AND C EQUAL D* can be written as *(A OR B) AND (C EQUAL D)* or as *(A OR (B AND (C EQUAL D)))*

In the absence of parentheses, the precedence rules for the logical operators are as follows:

=
LT, GT, LE, GE, EQUAL, MEMBER
AND
OR.

Otherwise, precedence goes from left to right.

All of the logical operators have a lower precedence than the arithmetic operators.

Examples:

A=B OR C=D translates to *(OR (EQUAL A B)(EQUAL C D))*

A EQUAL B + C translates to *(EQUAL A (PLUS B C))*

A AND B OR C translates to *(OR (AND A B) C)*

Other Operators:

: ž is an infix operator that extracts elements from a list. *X:N* stands for the Nth element of the list X. For example, *X:2* stands for *(CADR X)* while *X:4* stands for *(CAR (CDDDR X))*.

The : operator has a higher precedence than any of the arithmetic or logical operators.

A negative N indicates the Nth element from the end of the list. For example *X:-1* stands for the last element in the list.

The : operator can be composed as in *X:2:3* which stands for *the third element of the second element of X* or *(CADDR (CADR X))*.

:: ž is an infix operator that extracts tails of lists. For example: *X::1* is the *(CDR X)*, *X::3* is the *(CDDDR X)*, and *X::-1* is the *(LAST X)*.

_ ž is an infix operator that indicates assignment. For our purposes, assignment means SETQ. For example, *X_Y* translates to *(SETQ X Y)*.

[Note: The _ operator can also be used in conjunction with the : operator to alter the composition of a list. For example, X:2_5 means replace the second element of X with 5. However, we have not yet covered how to say this in straight Lisp!].

<, > ž are special operators in CLISP used to construct lists. A balanced pair of angle brackets indicates that a list is to be constructed containing everything between the "<" and the ">". For example, *<A B C>* translates to *(LIST A B C)*, while *<1 2 3 <4 5 6>>* translates to *(LIST 1 2 3 (LIST 4 5 6))*.

IF-THEN-ELSE expressions

CLISP provides an IF-THEN-ELSE statement similar to that found in PASCAL and FORTRAN.

The form of the IF-THEN-ELSE expression is:
(IF *A* **THEN** *B* **ELSEIF** *C* **THEN** *D* **ELSEIF** *E* **THEN** *F* **... ELSE** *G***)**

This expression is directly translated into:

(COND

(A B)
(C D)
(E F)
...
(T G))

In "English":

If A is non-NIL, then evaluate B and exit

Otherwise, if C is non-NIL, then evaluate D and exit

Otherwise, if E is non-NIL, then evaluate F and exit

...

Otherwise, evaluate G and exit.

The IF, THEN, ELSEIF and ELSE keywords have the lowest precedence of any of the CLISP character operators.

Example:

(IF A = B + C THEN C + D ELSE E - F)

translates to

(COND

((EQUAL A (PLUS B C)) (PLUS C D))

(T (DIFFERENCE E F)))

Using CLISP - Advantage and Disadvantages

The advantage of CLISP is that it allows you to type-in complex expressions using a syntax that is more concise and supposedly more intuitive than the standard Lisp syntax.

The following examples make the case:

A^2+B^2+C^2

instead of *(PLUS (EXPT A 2)(EXPT B 2)(EXPT C 2))*

(IF A^2+B^2+C^2=C+D-E*2 THEN A_C+E ELSE C_X:2)

instead of

(COND

```
((EQ
    (IPLUS (EXPT A 2) (EXPT B 2) (EXPT C 2))
    (IDIFFERENCE (IPLUS C D) (ITIMES E 2)))
    (SETQ A (IPLUS C E)))
(T (SETQ C (CADR X))))
```

The **disadvantages** of CLISP are many! First CLISP is another whole set of rules for the user to learn. The rules are inconsistent with the rules of Lisp. For example, CLISP violates the notion that every Lisp expression starts with the name of a function to be applied to the arguments which follow. Thus, CLISP has opted for a syntax that is locally optimized (or assumed to be optimized) for certain special cases at the expense of consistency across the system as a whole. It is not at all clear that this was a good trade-off to make!!

CLISP is not well integrated into the Interlisp environment.

When CLISP expressions are translated into Lisp, the Lisp expressions replace the CLISP expressions. Thus if you look at an entry on the history list or if you DEdit a function, the CLISP you typed-in will be gone and a very different expression will be in its place. This can be very, very confusing!

Example:

```
65_ (COND (T NIL))
NIL
66_X_(IF ~(LISTP X) THEN A_C+D ELSE X:3)
3
67_REDO IF
IF  ?
68_REDO COND
8

...

72_(DEFINEQ (TEST (LAMBDA (X) (IF ~(LISTP X) THEN
A_C+D ELSE X:3) )))

(TEST)

73_PP TEST
```

```
(TEST
 (LAMBDA (X)  **COMMENT**
  (IF ~(LISTP X)
      THEN A_C+D
    ELSE X:3)))
(TEST)
74_(TEST 1)
8
75_PP TEST
(TEST
 (LAMBDA (X)  **COMMENT**
  (COND
   ((NLISTP X)
    (SETQ A (IPLUS C D)))
   (T (CADDR X)))))
(TEST)
```

A second example of problems with CLISP arises when DWIM error "correction" interacts with CLISP. The following is a simple example of a DWIM/CLISP interaction that is not at all intuitive:

```
86_ (SETQ PATIENT-RECORD 99)
99
87_(SETQ PATIENT 7)
7
88_(SETQ RECORD 2)
2
89_PATENT-RECORD
=PATIENT
5
```

[Why should DWIM correct this to (DIFFERENCE PATIENT RECORD) rather than to the atom PATIENT-RECORD ????]

Moral: CLISP is nice, but watch out for all its traps. CLISP can be very handy when typing complex expressions. But its utility is limited by the fact that it is just a coating of syntactic sugar spread on top of Interlisp that is bothinconsistent with the rest of Interlisp languageAND not well integrated into the eInterlisp environment.

DWIMIFY and CLISPIFY

DWIMIFY is a function that applies DWIM to an expression returning the expression with all of the "errors" corrected. This forces the translation of CLISP expressions as well as other DWIM corrections.

DWIMIFY takes one argument which is either the name of a function or an expression. DWIMIFY runs DWIM on the function or expression, making all the necessary changes and returns the translated/corrected expression.

Example:

```
99_(DWIMIFY '(IF A=B THEN C+D ELSE E+F))
E+F TREAT AS CLISP ?  yyes
(COND
  ((EQ A B)
    (IPLUS C D))
  (T (IPLUS E F)))
(COND ((EQ A B) (IPLUS C D)) (T (IPLUS E F)))
```

CLISPIFY is a function that takes a standard Lisp expression and translates it into CLISP using as many CLISP operators as possible. CLISPIFY is a sort of inverse to DWIMIFY.

CLISPIFY takes one argument which is either the name of a function or an expression. CLISPIFY translates the function or expression to CLISP

and returns the translation. If X is a function name, the function is redefined to the be the CLISPIFYed translation.

Example:

```
4_(CLISPIFY
        (QUOTE
            (COND ((LISTP X) (SETQ A 5))
                  (T (SETQ B 6)))))
```

(if (LISTP X) then A_5 else B_6)

DWIMIFY and CLISPIFY are available in DEdit as subcommands of the EDITCOM command. Select the expression you want DWIMIFYed or CLISPIFYed then click on EDITCOM in the DEdit menu with the missle mouse button. Choose DW (DWIMIFY) or CL (CLISPIFY) from the submenu that appears. The selected expression will be replaced by its DWIMIFYed or CLISPIFYed equivalent.

Tailoring CLISP

There are several parameters and functions that alter the behavior of CLISP. Some of the more interesting ones are the following:

CLISPFLG ž If NIL all CLISP infix and prefix operators are disabled but IF-THEN-ELSE and Iterative expressios remain in force. If the value is TYPE-IN, then CLISP is in effect only on user type-in and not on the body of defined functions. If the value is T, CLISP is in effect on all evaluated expressions. Initial value is T.

(CLDISABLE X) ž disables the CLISP operator X. For example, (CLDISABLE ’-) disables the - CLISP character operator while (CLDISABLE ’IF) disables IF expressions.

CLISPIFTRANFLG ž If T, the original IF statements are left alone during CLISP translation to Lisp. If NIL, the original CLSIP is replaced by the Lisp translations. Initially, NIL.

CLISPIFYPRETTYFLG ž If ALL, causes PP and MAKEFILE to CLISPIFY functions before printing them. If a list, causes all functions on the list to be CLISPIFYed before printing. If NIL, nothing is CLISPIFYed before printing, i.e., functions are printed as is.

CLISP Documentation

CLISP is documented in Chapter 16 of the IRM. Sections 16.1 thru 16.8 contain material of general interest. Section 16.9 is for hackers only. Description of tailoring parameters and functions appears mostly in Sections 16.6 thru 16.8.

IF-THEN-ELSE and Iterative expressions are covered in Chapter 4 of the IRM.

The Record Package is covered in Chapter 3.