F.G.H.
2/21/85

LispCourse #10: Miscellaneous Issues Regarding Type-in

READ macros

In general, the Lisp reader just takes Lisp expressions from the P.A. and passes them on to the Lisp evaluator.

However, the Lisp reader can be set to give special interpretation to certain characters or character sequences it reads in. When it receives a Lisp expression containing one of these special characters or character sequences, it carries out a prespecified action usually resulting in some change to the Lisp expression it received.

The character sequences that are special to the Lisp reader together with their associated actions are known as *READ macros*.

READ macros can be arbitrarily defined by any user. But doing so requires a fair understanding of programming. So we won't discuss it here.

However, there are three READ macros that are predefined in the standard Lisp reader. The READ macro characters are "'", "|'" and "Ctrl-Y". When these characters are typed in a Lisp expression, the reader carries out the following actions:

> **'** ž places the immediately following (i.e., with no spaces) S-expression within a call to QUOTE. For example, 'A becomes (QUOTE A) and '(A B C) becomes (QUOTE (A B C)). Note that this only happens if the "'" is preceded by a space or a parenthesis and followed by an S-expression. Otherwise, the "'" is not altered by the reader.
>
> |' ž places the immediately following (i.e., with no spaces) S-expression within a call to BQUOTE. For example, |'A becomes (BQUOTE A) and |'(A , B C) becomes (BQUOTE (A B C)). Note that this only happens if the "|'" is preceded by a space or a parenthesis and followed by an S-expression. Otherwise, the "|'" is not altered by the reader.
>
> > Note: BQUOTE is like QUOTE except that it allows exceptions. Any element in a BQUOTEd list that is preceded by a "," IS evaluated. The rest of the elements are not evaluated (as in QUOTE). Example: (BQUOTE (A , B C)) would evaluate to (A 7 C) when B had a value of 7.

If B had a value of (A B C), then (BQUOTE (Q , B)) would evaluate to (Q (A B C)).

Ctrl-Y ž causes the immediately following S-expression to be evaluated before the whole expression is passed to the Lisp evaluator. For example, if NEATO has the value 7 then the expression *(LIST 2 3 ^YNEATO ^Y(PLUS44 3))* will be changed to *(LIST 2 3 7 47)* BEFORE it is passed to the Lisp Evaluator.

Note: this functionality is seldom used by mere mortals.

Documentation: Read macros are documentated in Section 6.6.3 of the IRM. But the documentation is aimed mainly at programmers

Final Note: Read macros need seldom be worried about. You use "'" so often that it becomes second nature. "|'" and "Ctrl-Y" are never used by the non-programmer.

But some documentation mentions Read macros and every once in a while you'll here a mentyion in the hall. Now you know what they are.

Redefining the keyboard

All of the keys on the Inerlisp-D keyboard are *soft* keys. The interpretation of each key on the keyboard is not fixed, but can be changed by the user.

For example, the "A" key defaults to producing an "a" when the SHIFT key is up and an "A" when the SHIFT key is down.

While, it is unlikely anyone would want to change the "A" key, users sometimes wish to change the interpretation of the non-alphanumeric keys such as the blank keys on the right of the Dolphin/Dorado keyboard or the row of editing keys on the top of the Dandelion keyboard.

KEYACTION is the function that can be used to change the interpretation of a key. Keyaction takes two arguments: a *KeyName* and an *Action*.

KeyName is the name of the key whose interpretation is to be changed. Most keys are named by the character (or any of the characters) that appears on them. For example, the "A" key is named "A" or "a", while the "1" key is named "1" or "!".

Special keys are named in the obvious way: For the Dolphin/Dorado they are: TAB, LF, DEL, BLANK-TOP, BLANK-MIDDLE, BLANK-BOTTOM, SPACE, LSHIFT, RSHIFT, CTRL, ESC.

For the Dandelion, the special keys are SKIP, NEXT, UNDO, MOVE, MARGINS, FIND, LARGER, SMALLER, etc.

Action is either NIL or the action that should be taken when the key is pushed down and when it is let up.

If *Action* is NIL, then KEYACTION just returns the current interpretation of the key.

Otherwise, *Action* is of the form (*DownAction* . *UpAction*), where *DownAction* is the specification of what should happen when the key is pressed down and *UpAction* is a specification of what should happen when the key is let up.

Each action specifications can be one of the following:

NIL ž specifies that no action should be taken.

(***Character ShiftedCharacter LockedShifted?***) ž *Character* and *ShiftedCharacter* are either single characters or ASCII character codes standing for characters. *Character* is the code to be transmitted when the key is pressed or let up without the SHIFT key down. *ShiftedCharacter* is the code to be transmitted when the SHIFT key is down.

LockedShifted can be either LOCKSHIFT or NOLOCKSHIFT, indicating whether the *Character* or the *ShiftedCharacter* is to be transmitted when the LOCK SHIFT is down. For example, alphabetic keys are usually LOCKSHIFT and numeric keys are usually NOLOCKSHIFT. When the LOCK SHIFT is down, the "1" key transmits a "1" and not a "!", but when the SHIFT is down it transmits a "!". The "A" key transmits a "A" when both the SHIFT and the LOCK SHIFT are down.

LOCKUP, LOCKDOWN, CTRLUP, CTRLDOWN, METAUP, METADOWN, 1SHIFTUP, 1SHIFTDOWN, 2SHIFTUP, 2SHIFTDOWN ž when one of these is used as an action specification it changes the internal variables in Lisp that determine whether characters will be transmitted as Shifted, Control, Meta, and or LockShifted characters.

For example, if a key is specified as having a down action of LOCKDOWN, the pressing this key will be like pressing down the LOCK key on most keyboards. If another key has a down action of LOCKUP, the *pressing **down*** this key will be like letting ***up*** on the LOCK key.

Examples:

To set the "A" key to its ordinary interpretation:

(KEYACTION 'A '((a A LOCKSHIFT) . NIL))

To set the "1" key to its ordinary interpretation:

(KEYACTION '1 '((49 ! NOLOCKSHIFT) . NIL))

To set the "A" key so that it works when the key is let up rather than when it is pressed down:

(KEYACTION 'A '(NIL . (a A LOCKSHIFT)))

To set the "A" key so that it works BOTH when the key is let up and when it is pressed down:

(KEYACTION 'A '((a A LOCKSHIFT) . (a A LOCKSHIFT)))

To set the "A" key work only when pressed down, but to transmit a "B" instead of an "A":

(KEYACTION 'A '((b B LOCKSHIFT) . NIL))

On a Dandelion to make the LOCK key be the CTRL key:

(KEYACTION 'LOCK '(CTRLDOWN . CTRLUP))

To make the BS key transmit a BS (Ctrl-A) when unshifted and a BackWord (Ctrl-W) when shifted, both to take effect on the down press:

(KEYACTION 'BS '((1 23 NOLOCKSHIFT) . NIL))

Documentation: KEYACTION is documented on page 18.8 of the IRM.

CHARCODE is a function that returns the ASCII code of a character. This is useful in figuring out the *Character* and *ShiftedCharacter* arguments in KEYACTION.

CHARCODE takes one argument, a character. Since it is an NLAMBDA function, this character need not be quoted. CTRL characters are indicated by a "^" prefix, as in "^A". META characters are indicated by a "#" prefix as in "#A" or "#^A". Certain special characters have their own names such as CR, LF, SPACE, ESC, BS, TAB, and DEL.

Examples:

(CHARCODE BS) returns 8

(CHARCODE A) returns 65

(CHARCODE a) returns 97

(CHARCODE ^A) returns 1

(CHARCODE #A) returns 193

(CHARCODE #^A) returns 129.

Documentation can be found on page 2.12 of the IRM.

The TTY Process

At any given time, there are several things going on in your Interlisp-D environment. For example, you may be editing a file in a TEdit window while you are copying another file in your Exec window. Also the clock window is constantly updating the time and Lafite is constanttly checking if you have new mail.

Roughly speaking, each thing that is going on in your environment at one time is called a *process*. The Interlisp-D environment is said to allow *multiple processes*, i.e., many things going on simultaneously.

A process usually is associated with one or more windows. For example, each TEdit is a process and is associated with a TEdit window. There are, however, process with several windows as well as processes without any windows.

Understanding multiple processes, how to manage them and how to use them effectively is the topic for another whole session.

However, we need to know about one special process called the ***TTY process***.

At any given time there may be many processes running in your environment, but there is only one keyboard. Therefore, there is only one process that can receive input from the keyboard at any given time. Interlisp handles this problem by allowing only one process at a time to "own" the keyboard.

The process which "owns" the keyboard at a given moment is known as the ***TTY process***. When the user types at the keyboard, the typed input gets sent to the TTY process for processing. If the TTY process is the Lisp exec, then the input gets evaluated. If the TTY process is a TEdit, then the input gets entered into the file being edited.

The TTY process can be moved from process to process in many ways under both user and program control.

Clicking with the mouse in a window usually makes the process associated with that window become the TTY process. For example, if you have two TEdits running. If you click in the first TEdit window, your type in will be dispatched to that TEdit. However, if you then click in the second TEdit window subsequent type-in will be dispatched to the second TEdit. If you then click in the Exec window, subsequent type-in will go to the Lisp Exec.

A window whose process is the TTY process usually contains a blinking caret that indicates where the type-in will be put. Usually, windows that are not associated with the TTY process have a caret that is not blinking. Therefore, the blinking caret generally indicates which window contains the TTY process.

The concept of a TTY process should become second nature to you as you use the Interlisp environment.

Interrupt Characters

Some characters, known as *interrupt characters*, bypass the normal Lisp type-in processing altogether. Immediately after these characters are typed-in, they are dispatched to the interrupt mechanism. The interrupt mechanism then interrupts the ongpoing Lisp processing. How processing is interrupted is determined by what specific interrupt character was typed.

Interrupt characters generally affect only the current TTY process. Moreover, which interrupt characters are activated at a given time is determined by the TTY process at that time. Many programs (e.g., TEdit) turn off all or certain interrupt characters while they are the TTY process. Thus interrupt characters may differ depending on what programs are running and what programs have the TTY process.

There is a default set of interrupt characters that are generally in effect when you are typing into the Lisp Exec window. These characters are the following:

Ctrl-B ž causes the TTY process to go into a break, opening abreak window.

Ctrl-C ž causes Lisp to enter RAID or TELERAID, a debugger where no sane Lisp user wishes to tread.

Ctrl-D ž immediately aborts the current TTY process.

Ctrl-E ž causes a soft abort of the TTY process at the next sensible time. The abort is soft because the aborted process can abort the abort if it has the mechanism to handle the situation.

Ctrl-H ž Pops up a menu listing all currently running processes. Selecting a process from this menu cause that process to go into a break.

Ctrl-T ž prints some statistics about the TTY process in the TTY processes window.

INTERRUPTCHAR is a function that can be used to alter the interrupt status of a given character. INTERRUPTCHAR takes two arguments: *CharacterCode* and *InterruptType*.

CharacterCode is the ASCII code for the character whose interrupot status is being altered. The ASCII code can be obtained using CHARCODE. For example, Ctrl-C is (CHARCODE ^C) is 3.

InterruptType is an atom stating what the new interrupt status should be. The possible values are:

NIL ž turn off this character's interrupt status altogether. The character will no longer be processed by the interrupt mechanism.

RESET ž make this character act like Ctrl-D ordinarly does.

ERROR ž make this character act like Ctrl-E ordinarly does.

HELP ž make this character act like Ctrl-H ordinarly does.

BREAK ž make this character act like Ctrl-B ordinarly does.

RAID ž make this character act like Ctrl-C ordinarly does.

CONTROL-T ž make this character act like Ctrl-T ordinarly does.

Finally, if *InterruptType* is T, no change is made to the character's interrupt status, but its current status is returned as the value of the call to INTERRUPTCHAR.

INTERRUPTCHAR is normally used to turn off unpleasant interrupt characters like Ctrl-C or to change the location of interrupts assigned to characters that are often hit by mistake. For example, Ctrl-H is BackSpace on many other systems such as the Vax. I often hit Ctrl-H by mistake when I meant BackSpace, especially after an extended session using the Vax. Therefore, I have the Ctrl-H function moved over to Ctrl-N.

For example, my Init file contains the following three calls to INTERRUPTCHAR:

(INTERRUPTCHAR (CHARCODE ^C) NIL) ž turn off Ctrl-C

(INTERRUPTCHAR (CHARCODE ^H) NIL) ž turn off Ctrl-H

(INTERRUPTCHAR (CHARCODE ^N) 'HELP) ž make Ctrl-N be old Ctrl-H

Documentation: interrupt characters and INTERRUPTCHAR are discussed in Sections 9.6, 18.1, and 18.20.6.2 of the IRM. Most of the documentation is pretty technical.

Final Note: Don't fool with interrupt characters. Turn off the odious ones in your Init file and then let them be. Don't turn off Ctrl-D or Ctrl-E as it would then be hard to abort a program running amok.

References

See Documentation notes under the various topics above.

Exercises

Work on old homeworks.