Debugging LISP Code When Using CLISP

**©2018 Larry W. Clark,** UTSA CS students may make copies for their personal use

# 1. TRACE Macro

Common LISP provides a (TRACE *funcName*) macro which turns **ON** tracing of that function. TRACE must be executed after *funcName* is defined, and actually modifies the executable version of the code for *funcName.* When the specified function is executed, TRACE causes it to print

* a line indicating what is passed
* a subsequent line indicating the result of the function call.

## Example Using the TRACE Macro – (trace memSet)

Suppose we have the **memSet** function which returns T if the first parameter is a top-level member of second parameter.

(defun **memSet** (atm L)

(cond ( (NULL L) NIL )

( (EQL atm (CAR L)) T )

( T (MEMSET atm (CDR L)) )

) )

When we execute (memSet 'y '(x y z)), it returns T; whereas, (memset 'w '(x y z)) returns NIL. After defining memSet, we can use (trace memSet) to turn on tracing.

> (trace memset)

;; tracing function MEMSET.

(MEMSET)

> (memset 'y '(x y z))

1. Trace: (MEMSET 'Y '(X Y Z))

2. Trace: (MEMSET 'Y '(Y Z))

2. Trace: MEMSET ==> T

1. Trace: MEMSET ==> T

T

> (memset 'w '(x y z))

1. Trace: (MEMSET 'W '(X Y Z))

2. Trace: (MEMSET 'W '(Y Z))

3. Trace: (MEMSET 'W '(Z))

4. Trace: (MEMSET 'W 'NIL)

4. Trace: MEMSET ==> NIL

3. Trace: MEMSET ==> NIL

2. Trace: MEMSET ==> NIL

1. Trace: MEMSET ==> NIL

NIL

## Another Example Using the TRACE Macro – (trace memAll)

Suppose we have the **memAll** function which is similar to memSet, but doesn't look just at the top-level. memAll will also search inside embedded lists at all levels.

(defun MEMALL (atm L)

(cond ( (NULL L) NIL )

( (ATOM L) (EQL atm L) )

( T (OR (MEMALL atm (CAR L))

(MEMALL atm (CDR L)) ) )

) )

When we execute (memAll 'X '( (Y X) Z)), it returns T. After defining memAll, we can use (trace memAll) to turn on tracing of memAll.

> (trace memAll)

;; tracing function MEMALL.

(MEMALL)

> (memall 'X '( (Y X) Z))

1. Trace: (MEMALL 'X '((Y X) Z))

2. Trace: (MEMALL 'X '(Y X))

3. Trace: (MEMALL 'X 'Y)

3. Trace: MEMALL ==> NIL

3. Trace: (MEMALL 'X '(X))

4. Trace: (MEMALL 'X 'X)

4. Trace: MEMALL ==> T

3. Trace: MEMALL ==> T

2. Trace: MEMALL ==> T

1. Trace: MEMALL ==> T

T

# 2. Error Display within clisp

When an error occurs in CLISP, CLISP will break execution to allow you to examine the problem. It will display an error, you can then use its debugger commands, use a restart option or just ignore the problem.

## 2.1 Understanding the error display and listed restart options.

Consider the following code for memAll2:

(defun MEMALL2 (atm L)

(cond ( (NULL L) NIL )

( (ATOM L) (EQL amt L) )

( T (OR (MEMALL2 atm (CAR L))

(MEMALL2 atm (CDR L)) ) )

) )

When we execute memall2 by loading a file named "memall.txt", CLISP breaks execution and shows the following error message:

> (memall2 'X '( (Y X) Z))

\*\*\* - COND: variable AMT has no value

The following restarts are available:

USE-VALUE :R1 Input a value to be used instead of AMT.

STORE-VALUE :R2 Input a new value for AMT.

SKIP :R3 skip (MEMALL2 'X '#)

RETRY :R4 retry (MEMALL2 'X '#)

STOP :R5 stop loading file E:\cs3723 Pgm Lang\memall.txt

ABORT :R6 Abort main loop

Break 1 [2]>

The first line of its error message shows that execution encountered a variable, AMT, which has no value. CLISP then provides multiple options. You don't have to choose any of them. Instead you can get more information (and we will come back to these options later in this discussion).

The debugger command **:u** goes up to the surrounding list where the error occurred. It is going UP in the call chain.

Break 1 [2]> :u

[191] EVAL frame for form (EQL AMT L)

This showed that the list that contained the variable AMT was the list (EQL AMT L). We can then go further up in the call chain to see additional surrounding lists.

Break 1 [2]> :u

<1/188> #<SPECIAL-OPERATOR COND>

[187] EVAL frame for form (COND ((NULL L) NIL) ((ATOM L) (EQL AMT L)) (T (OR (MEMALL2 ATM (CAR L)) (MEMALL2 ATM (CDR L))

)))

It shows that this is part of the COND .

We can examine the value of a variable by simply typing its name.

Break 1 [2]> L

Y

Break 1 [2]> atm

X

If we continue going up the call chain, we can see where this COND was called by recursively calling MEMALL2 within the function MEMALL2 itself.

Break 1 [6]> :u

[201] APPLY frame for call (MEMALL2 'X 'Y)

Break 1 [6]> :u

[195] EVAL frame for form (MEMALL2 ATM (CAR L))

We see that MEMALL2 was called passing 'X and 'Y. To better see the code for that call, we went UP again. We can examine the value of L at that point in execution:

Break 1 [6]> L

(Y X)

Break 1 [6]>

Refer to the list of options presented by CLISP when it broke execution. Since the error was caused by a typo in our variable name (AMT instead of ATM), we won't choose either of those first two options. We could stop loading the file by either using the STOP or ABORT options by entering :R5 or :R6.

Break 1 [2]> :r5

;; Loaded file memall.txt

T

[3]>

This also took us out of the debugger for this execution which is indicated by the prompt not including "Break". Note that the value of **:Ri** vary with each error.

Instead of stopping execution of the file, we also could have chosen to SKIP execution of the text line(s) that caused to execution error. That would cause CLISP to continue with the file contents after those text line(s).

## 2.2 Additional debugging example

Consider the following code for memAll3:

(defun MEMALL3 (atm L)

(cond ( (NULL L) NIL )

( (ATOM L) (EQL atmX L) )

( T (OR (MEMALL3 (atm (CAR L)))

(MEMALL3 atm (CDR L)) ) )

)

)

When we execute memall3 by loading a file named "memall.txt", CLISP breaks execution and shows the following error message:

> (memall3 'X '( (Y X) Z))

\*\*\* - EVAL: undefined function ATM

The following restarts are available:

USE-VALUE :R1 Input a value to be used instead of (FDEFINITION 'ATM).

RETRY :R2 Retry

STORE-VALUE :R3 Input a new value for (FDEFINITION 'ATM).

SKIP :R4 skip (MEMALL3 'X '#)

RETRY :R5 retry (MEMALL3 'X '#)

STOP :R6 stop loading file E:\cs3723 Pgm Lang\memall.txt

ABORT :R7 Abort main loop

CLISP is telling us that we are trying to invoke a function named ATM. Let's see where that was in the code by going UP in the call stack.

Break 1 [9]> :u

[162] EVAL frame for form (MEMALL3 (ATM (CAR L)))

Break 1 [9]>

We can now see that we were trying to call MEMALL3 passing (ATM (CAR L)) . This is a common mistake when people begin using LISP. They have used parentheses around arguments in other languages (e.g., func(obj.attr, anotherObj). In LISP, that pair of parentheses for the list (ATM (CAR L)) indicates that ATM is a function.