

TRAC - A String Processing Language

by Calvin N. Mooers 1964

An attempt was made to control this language by trademarking the name!?

The only data in a TRAC program is strings

There are only three syntactical constructs in the language

#(...) - active function call

Call the indicated function and replace the call by the result and reinterpret the result

##(...) - neutral function call

Call the indicated function and replace the call by the result and do not reinterpret the result

(...)

Remove the parentheses and do not interpret the enclosed string
()'s properly nest

A function call looks like

(fn, arg1, arg2, arg3)

or

(fn)

The first string before the comma or) is the name of the function

The arguments follow separated by commas

The TRAC Interpreter Algorithm

At the start of the program the program is in the "active" string and the "scanning pointer" points to the first character of this string

Characters processed by the scanner are generally added to the right end of the "neutral" string

The Scanning Algorithm

1. The character under the scanning pointer is examined. If there is no character left (active string empty), go to rule 14.
2. If the character just examined (by rule 1) is a begin parenthesis, the character is deleted and the pointer is moved ahead to the character following the first matching end parenthesis. The end parenthesis is deleted and all nondeleted characters passed over (including nested parentheses) are put into the neutral string without change. Go to rule 1.
3. If the character just examined is either a carriage return, a line feed or a tabulate, the character is deleted. Go to rule 15.
4. If the character just examined is a comma, it is deleted. The location following the right-hand character at the end of the neutral string, called the "current location," is marked by a pointer to indicate the end of an argument substring and the beginning of a new argument substring. Go to rule 15.
5. If the character is a sharp sign, the next character is inspected. If this is a begin parenthesis, the beginning of an active function is indicated. The sharp sign and begin parenthesis are deleted and the current location in the neutral string is marked to indicate the beginning of an active function and the beginning of an argument substring. The scanning pointer is moved to the character following the deleted parenthesis. Go to rule 1.
6. If the character is a sharp sign and the next character is also a sharp sign, the second-following character is inspected. If this is a begin parenthesis, the beginning of a neutral function is indicated. Two sharp signs and the begin parenthesis are deleted and the current location in the neutral string is marked to indicate the beginning of a neutral function and the beginning of an argument substring. The scanning pointer is moved to the character following the deleted parenthesis. Go to rule 1.
7. If the character is a sharp sign, but neither rule 5 or 6 applies, the character is added to the neutral string. Go to rule 15.

8. If the character is an end parenthesis, the character is deleted. The current location in the neutral string is marked by a pointer to indicate the end of an argument substring and the end of a function. The pointer to the beginning of the current function is now retrieved. The complete set of argument substrings for the function have now been defined. The action indicated for the function is performed. Go to rule 10.

9. If the character meets the test of none of the rules 2 through 8, transfer the character to the right-hand end of the neutral string and go to rule 15.

10. If the function has null value, go to rule 13.

11. If the function was an active function, the value string is inserted to the left of (preceding) the first unscanned character in the active string. The scanning pointer is reset so as to point to the location preceding the first character of the new value string. Go to rule 13.

12. If the function was a neutral function, the value string is inserted in the neutral string with its first character being put in the location pointed to by the current begin-of-function pointer. Delete the argument and function pointers back to the begin-of-function pointer. The scanning pointer is not reset. Go to rule 15.

13. Delete the argument and function pointers back to the begin-of-function pointer for the function just evaluated, resetting the current location to this point. Go to rule 15.

14. Delete the neutral string, initialize its pointers, reload a new copy of the idling procedure into the active string, reset the scanning pointer to the beginning of the idling procedure, and go to rule 1.

15. Move the scanning pointer ahead to the next character. Go to rule 1.

Notice all of the goto's (this is before 1968)

TRAC Scanner - Summary

Actually the operation of the interpreter is simple

(...) expressions are copied from the active to the neutral string with the ()'s removed

Functions are copied to the neutral string with marks where their function name and arguments are

On the trailing) of a function call the function is called with the arguments assembled on the neutral string and the result put on the active or neutral string depending on whether the function call was active or neutral

(for functions that return the empty string it doesn't matter)

Carriage returns, tabs, and line feeds are ignored

The idle string is reloaded into the active string if the active string becomes empty or on any error

TRAC Predefined Functions

Input-Output

#(rs) "read string"

#(rc) "read character"

#(cm,X) "change meta"

#(ps,X) "print string"

Define and Call Functions

#(ds,N,X) "define string"

The string symbolized by X is placed in storage and is given the name symbolized by N.

#(ss,X1,X2,...) "segment string"

The form name N is segmented by X1, X2, ... so that a call will replace X1 with the first argument, X2 by the second argument etc.

#(cl,N,X1,X2,...) "call"

The value is generated by bringing the form named N from storage and filling the segment gaps of ordinal value one with string X1, the gaps of ordinal value two with string X2, and so on for all the segment gaps in the form.

Same as **#(N,X1,X2,...)**

#(dd,N1,N2,...) "delete definition"

Arithmetic Functions

#(ad,D1,D2) "add"

#(su,D1,D2) "subtract"

#(ml,D1,D2) "multiply"

#(dv,D1,D2) "divide"

Boolean Functions

#(bu,01,02) "Boolean union,"

#(bi,01,02) "Boolean intersection"

#(bc,01) "Boolean complement"

#(bs,D1,01) "Boolean shift"

#(br,D1,01) "Boolean rotate"

Decision Functions

#(eq,X1,X2,X3,X4) "equals"

If X3 is equal to X2, the value is X3; otherwise it is X4.

#(gr,D1,D2,X1,X2) "greater than"

If the number D1 is algebraically greater than the number D2 the value is X1; otherwise it is X2.

External Storage Management Functions

#(sb,N,N1,N2,...) "store block"

#(fb,N) "fetch block"

#(eb,N) "erase block"

Diagnostic Functions

#(ln,X) "list names"

#(pf,N) "print form"

#(tn) "trace on"

#(tf) "trace off"

The Idle String

The typical idle string is

```
# (ps, # (rs) )
```

which will read a string, interpret it (since the #(rs) is active) and print the result

Example of TRAC code

```
# (ds, AA, CAT)
```

```
    define "AA" to be "CAT"
```

```
# (ds, BB, (# (cl, AA) ) )
```

```
    define "BB" to be "#(cl,AA)"
```

```
# (ps, (# (cl, BB) ) )
```

```
    prints
```

```
# (cl, BB)
```

```
# (ps, ## (cl, BB) )
```

```
    prints
```

```
# (cl, AA)
```

```
# (ps, # (cl, BB) )
```

```
    prints
```

```
CAT
```

Factorial

Here is the factorial function:

```
#(ds, Factorial, (#(eq, 0, X, 1,  
  (#(ml, X, # (cl, Factorial, # (ad, X, -1) ) ) )  
)))  
#(ss, Factorial, X)
```