[7.41] Indirection bug with local variables

Indirection fails with local variables, both in the *Local* statement list and when passed as a function or program argument. This program demonstrates the bug:

```
testtype(n)
Func
Local t
getType(#n)→t
Return t
EndFunc
```

Note that *n* is a local variable since it is the argument, and *t* is declared as a local variable. For these following initialized variables:

```
"s"→test2\m          123→test1\m          123→test1\n          123→test1\t
```

testtype() gives these results:

```
testtype("test2\m")   correctly returns "STR"
testtype("test1\m")   correctly returns "NUM"
testtype("test1\n")   returns "NONE" in error, since test1\n is a global variable
testtype("test1\t")   returns "NONE" in error, since test1\t is a global variable
```

One work-around is to use special characters for local variable names, for example:

```
ä, ë, ï, ö, ü, ÿ, ä1, äë1, kä
```

With this method, you are assuming that the calculator user will have global variables with these names. You can combine special characters with conventional characters for program readability, and to further reduce the chance that the user will have a global variable of the same name. For example, *testtype()* can be coded as shown, and returns correct results:

```
testtype(än)
Func
Local ät
getType(#än)→ät
Return ät
EndFunc
```

A similar method embeds underscore characters "_" in the variable names. For example, you could use *n_* for *n*, *t_* for *t*, and so on. Again, you assume that the user does not use these names.

As another work-around, *expr()* can be used to build expressions to evaluate, for example

```
v→#n            becomes       expr("v→"&n)
#n→v            becomes       expr(n&"→v")
getType(#n)→n   becomes       expr("getType"&n&")→t")
```

With this method, this version of *testtype()* returns the correct results:

```
testtype(n)
Func
Local t
expr("getType("&n&")→t")
Return t
EndFunc
```

(Credit to Martin Daveluy)