[7.23] Passing user function names as program/function arguments

You cannot directly pass user functions as arguments to programs or functions, because the 89/92+ will try to evaluate them when the function is called. Many routines require a function as an input.

The general method is to pass the function name as a string. The called function then evaluates the function using either *expr()* or indirection.

Using expr()

To pass a function successfully, pass the name and argument variable (or variables) as a string, like this:

```
t("f1(x)","x")
```

where *f1(x)* is the function to be evaluated, and *x* is the function argument variable. To evaluate this function in program *t(),* build a string and evaluate the string with *expr().* We want a string of the form

f1(x)|x=xval

where *xval* is the value at with to evaluate the function. So, the complete program looks like this:

```
t(fxx,xx)
Prgm
local xval,result
-1Ø→xval
expr(fxx&"|"&xx&"="&string(xval))→result
EndPrgm
```

Usually the function must be evaluated several times, and there is no need to build the entire string expression each time. Avoid this by saving most of the expression as a local variable:

```
t(fxx,xx)
Prgm
local xval,result,fstring
fxx&"|"&xx&"="→fstring
1Ø→xval
expr(fstring&string(xval))→result
EndPrgm
```

Most of the string is saved in local variable *fstring*, which is then used in *expr().*

t() can be be a function or a program.

Using indirection

Instead of using *expr(),* you can use indirection. The function name must still be passed as a string, but the parameters can be passed as simple expressions. For example, suppose we have a function *t3()* that we want to evaluate from function *t1(). t3()* looks like this:

```
t3(xx1,xx2)
Func

if xx1<Ø then
 return -1
else
 return xx1*xx2
endif
```

```
EndFunc
```

This is the calling routine routine *t1():*

```
t1(fname,x1,x2)
Func

#fname(x1,x2)

EndFunc
```

Then, to run *t1()* with function *t3(),* use this call:

```
t1("t3",2,3)
```

which returns 6.

When passing just the function name DOES work:

Sometimes you *can* pass the function and variable, and it will work. For example, suppose we use the call

```
t(f1(x),x)
```

with this program

```
t(fxx,xx)
Prgm
local xval,result
1→xval
fxx|xx=xval→result
EndPrgm
```

In this case, the calculator will try to evaluate *f1(x)* when *t()* is called. If *x* is undefined, *f1()* will be symbolically evaluated (if possible) with the variable *x*. This expression is then passed as *fxx*. Later in the line fxx|xx=..., the symbolic expression is evaluated. For example, suppose that *f1(x)* is defined as

```
f1(x)
Func
2*x-3
EndFunc
```

so, the first evaluation (when *t()* is called) results in fxx = 2*x-3, which is then correctly evaluated later. However, if the function performs a conditional test on *x*, the program will fail. For example, if we have

```
f1(x)
Func
if x>0 then
 2*x-3
else
 2*x+3
endif
EndFunc
```

the value of *x* is not defined at the time of the *t(fxx,xx)* call, so the program fails with the error message *A test did not resolve to TRUE or FALSE*.

(credit for indirection method to Eric Kobrin)