[11.4] Using *solve()* with multiple equations and solutions in programs

This rather involved tip relates to using the *solve()* function in programs, when multiple equations to be solved are generated within the program. The basic idea is to build a string of the equations to be used in the *solve()* function, evaluate the solve function string with *expr(),* then change the results into a more useful list form. This tip only applies if you are interested in numeric solutions, not symbolic solutions.

As an example, I'll use this system of equations to be solved:

$ax_1^2 + bx_1 + c = y_1$
$ax_2^2 + bx_2 + c = y_2$
$ax_3^2 + bx_3 + c = y_3$

where the unknown variables are *a*, *b* and *c*. x_1, x_2, x_3, y_1, y_2 and y_3 are given. Actually, since this is a linear system, you wouldn't really use *solve();* instead you would use the matrix algebra functions built into the 89/92+. But it makes a good example.

Suppose that we know that

$x_1 = 1$ $y_1 = -1$
$x_2 = 2$ $y_2 = -11$
$x_3 = 3$ $y_3 = -29$

so the equations to be solved are

$a + b + c = -1$
$4a + 2b + c = -11$
$9a + 3b + c = -29$

To make the eventual routine more general-purpose, rewrite the equations to set the right-hand sides equal to zero:

$a + b + c + 1 = 0$
$4a + 2b + c + 11 = 0$
$9a + 3b + c + 29 = 0$

This routine, called *solvemul(),* returns the solutions as a list:

```
solvemul(eqlist,vlist)
func
©Solve multiple equations
©eqlist: list of expressions
©vlist: list of variables
©returns list of solutions in vlist order
©calls strsub() in same folder
©25dec99/dburkett@infinet.com

local s,vdim,k,t,vk,vloc,dloc

dim(vlist)→vdim

©Build expression to solve
""→s
for k,1,vdim-1
 s&string(eqlist[k])&"=0 and "→s
endfor
s&string(eqlist[vdim])&"=0"→s
```

```
©Solve for unknown variables
string(expr("solve("&s&","&string(vlist)&")"))→s

©Convert solution string to list
newlist(vdim)→t

strsub(s,"and",":")→s ©Change "and" to ":"
strsub(s," ","")→s ©Strip blanks

for k,1,vdim
 instring(s,string(vlist[k]))→vloc
 instring(s,":",vloc)→dloc
 if dloc=0: dim(s)+1→dloc
 mid(s,vloc+2,dloc-vloc-2)→t[k]
endfor

©Return coefficient list
seq(expr(t[k]),k,1,vdim)

Endfunc
```

The input parameters are *eqlist* and *vlist*, where *eqlist* is a list of expressions to solve, and *vlist* is the list of variables for which to solve. *solvemul()* assumes that the expressions in *eqlist* are equal to zero. The routine returns the solutions in *vlist* order. Note that an external function *strsub()* is called. *strsub()* must be in the same folder as *solvemul().* See tip [8.2] for details.

To use *solvemul()* to solve our example, store two variables:

eqs1 is {a+b+c+1, 4*a+2*b+c+11, 9*a+3*b+c+29}

vars1 is {a,b,c}

then call *solvemul()* like this:

```
solvemul(eqs1,vars1)
```

and it returns

{-4, 2, 1}

which means that a = -4, b = 2 and c = 1, which happens to be the correct answer.

Again, there is no advantage to using this function to solve equations at the command line, or even equations in a program, if the equations are known in advance. This type of program is necessary when your application program generates the functions to be solved, and you don't necessarily know what they are before you run the program. Further, *solvemul()* returns the solutions as a list, which your program can use in further calculations, or display, as appropriate.

If you've read this tip list, word for word, up to this point, the first three parts of the function hold no surprises. The two input lists are built into a string that can be used in *solve(),* and then *expr()* is used to evaluate the string and solve for unknown variables.

However, the result of *solve()* is returned as a string that looks something like this:

"a=-4.0 and b = 2.0 and c=1.0"

The last parts of *solvemul()* convert this string to a list. First, I convert all the "and" occurrences to ":", and delete all the spaces, for a string like this:

"a=-4.0:b = 2.0:c=1.0"

Next, I loop to search the string for the variables in *vlist*, and extract the values for these variables. Using the ":" character as a delimiter makes this easier. Finally, I build the string of these result and return it.

To use this routine, you have to be sure that the solution string consists of only a single numeric solution for each unknown variable, so that the solution string does not have any "or"s in it. It would be easy to add a test for this error condition, and return an error code, but my example doesn't show that.

Further, a real application should test to ensure that *solve()* really did return valid solutions.