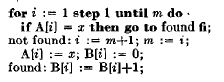
Experimenting with wren flow control statements, using the examples from Knuth’s paper, Structured Programing with go to Statements.

Example 1

for range(1,m)

if A[$val] == x break i=$val

then // executed if no break above

i = m+1

m = i

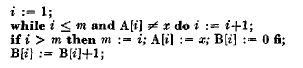
A[i] = x

B[i] = 0

B[i] = B[i]+1

We don’t like this construction as much because the iteration variable is used outside the loop.

Example 1a

i = 1

do if i<=m && A[i]!=x continue i=i+1

if i>m

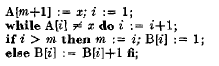
m = i

A[i] = x

B[i] = 0

B[i] = B[i]+1

Example 2

A[m+1] = x

i = 1

do if A[i] != x continue i=i+1

if i>m

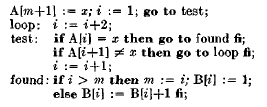
m = i

B[i] = 1

else

B[i] = B[i]+1

Example 2a

A[m+1] = x

i = 1

do if A[i] != x

if A[i+1] != x continue i=i+2

i = i+1

if i>m

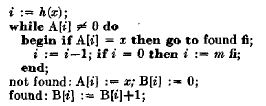
m = i

B[i] = 1

else

B[i] = B[i]+1

Example 3

i = h(x)

do if A[i]

if A[i] == x break

continue i=i-1 or i=m

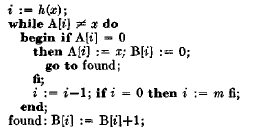
then // executed if no `break` above

A[i] = x

B[i] = 0

B[i] = B[i]+1

Example 3a

i = h(x)

do if A[i] != x

if A[i] == 0

A[i] = x

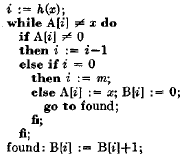
B[i] = 0

break

continue i=i-1 or i=m

B[i] = B[i]+1

Example 3b

i = h(x)

do if A[i] != x

if A[i] != 0

i = i-1

else if i == 0

i = m

else

A[i] = x

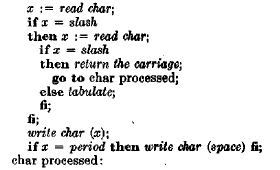
B[i] = 0

break

continue

B[i] = B[i]+1

Example 4

do

x = $sys.read()

if x == slash

x = $sys.read()

if x == slash

break $sys.write("\n")

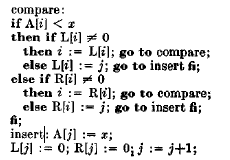
$sys.write("\t")

then // works the same with or without

$sys.write(x)

if x == "." $sys.write(" ")

Example 5

do

if A[i] < x

if L[i] continue i = L[i]

else break L[i] = j

else

if R[i] continue i = R[i]

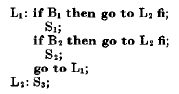
else break R[i] = j

A[j] = x

L[j] = 0; R[j] = 0;

j = j+1

Example after 5

do

if B1 break

S1

if B2 break

S2

continue

S3

So the general form of a do block is:

do <if condition>? <#label>?

<statement>+

<then

<statement>+

>?

Inside the do block, a break will exit and a continue will restart. If no break is encountered, the then block will be executed. The then statement can be thought of as “If no break occurred then…”. Alternatively, one can think of the break as jumping out of both the do *and* the then blocks.

The initial if condition provides for an automatic exit if the condition fails, but doesn’t count as an explicit break, so the then clause will still execute after the condition fails.

The label allows the break and continue statements to exit from multiple levels deep.

An alternate form for iterating over arrays is

for array <#label>?

<statement>+

<then

<statement>+

>?

Which behaves similarly to do except no explicit continue statement is needed at the end (though an early one is fine), and inside the for block, automatic variables $key and $val are set to each subsequent entry in the array.

The break and continue statements take the form

break <#label>? <statement>?

continue <#label>? <statement>?

The most common scenario is an iteration variable is incremented right before a break or continue, so that statement can be combined with and come right after the break or continue statement. The optional label allows the program flow to break or continue out of the current inner block to an arbitrary outer block with the matching label.

I was thinking that a bare expression would append another value to the local scope. But the problem with that is: how is a function call not a bare expression? The programmer would probably not expect every function call to add another entry to the local scope. I guess it depends: if a function doesn’t return a value, then it doesn’t add to the local scope. But if it does return a value, and that value is not captured somewhere, then maybe the default of adding to the local scope isn’t too troublesome. That actually makes for an interesting idea: we can ignore function values if we want to, but we can always check the last result by checking the last entry in the local scope.