O Codes

Table of Contents

O-codes provide for flow control in NC programs. Each block has an associated number, which is the number used after O. Care must be taken to properly match the O-numbers. O codes use the letter *O* not the number zero as the first character in the number like O100 or o100.

1. Numbering

Numbered O codes must have a unique number for each subroutine, .Numbering Example

```
(the start of o100)
o100 sub
(notice that the if-endif block uses a different number)
  (the start of o110)
  o110 if [#2 GT 5]
    (some code here)
  (the end of o110)
  o110 endif
  (some more code here)
(the end of o100)
o100 endsub
```

2. Comments

Comments on the same line as the O word should not be used as the behavior can change in the future.

The behavior is undefined if:

- The same number is used for more than one block
- Other words are used on a line with an O- word
- Comments are used on a line with an O-word

Note Using the lower case o makes it easier to distinguish from a 0 that might have been mistyped. For example o100 is easier to see than O100 that it is not a 0.

3. Subroutines

Subroutines starts at *Onnn sub* and ends at *Onnn endsub*. The lines between *Onnn sub* and *Onnn endsub* are not executed until the subroutine is called with *Onnn call*. Each subroutine must use a unique number.

Subroutine Example

```
o100 sub
  G53 G0 X0 Y0 Z0 (rapid move to machine home)
o100 endsub

(the subroutine is called)
o100 call
M2
```

See G53 & G0 & M2 sections for more information.

O- Return

Inside a subroutine, *O- return* can be executed. This immediately returns to the calling code, just as though *O- endsub* was encountered.

O- Return Example

```
o100 sub
  (test if parameter #2 is greater than 5)
  o110 if [#2 GT 5]
    (return to top of subroutine if test is true)
    o100 return
  o110 endif
    (this only gets executed if parameter #2 is not greater than 5)
    (DEBUG, parameter 1 is [#1])
o100 endsub
```

See the Binary Operators & Parameters sections for more information.

O- Call

O- Call takes up to 30 optional arguments, which are passed to the subroutine as *#1*, *#2* , ..., #N. Parameters from #N+1 to #30 have the same value as in the calling context. On return from the subroutine, the values of parameters #1 through #30 (regardless of the number of arguments) will be restored to the values they had before the call. Parameters #1 - #30 are local to the subroutine.

Because *1 2 3* is parsed as the number 123, the parameters must be enclosed in square brackets. The following calls a subroutine with 3 arguments:

O- Call Example

```
o100 sub
  (test if parameter #2 is greater than 5)
  o110 if [#2 GT 5]
    (return to top of subroutine if test is true)
    o100 return
  o110 endif
    (this only gets executed if parameter #2 is not greater than 5)
    (DEBUG, parameter 1 is [#1])
o100 endsub

o100 call [1] [2]
```

Subroutine bodies may not be nested. They may only be called after they are defined. They may be called from other functions, and may call themselves recursively if it makes sense to do so. The maximum subroutine nesting level is 10.

Subroutines can change the value of parameters above #30 and those changes will be visible to the calling code. Subroutines may also change the value of global named parameters.

4. Looping

The *while loop* has two structures: *while/endwhile*, and *do/while*. In each case, the loop is exited when the *while* condition evaluates to false. The difference is when the test condition is done. The *do/while* loop runs the code in the loop then checks the test condition. The *while/endwhile* loop does the test first.

While Endwhile Example

```
(draw a sawtooth shape)
G0 X1 Y0 (move to start position)
#1 = 0 (assign parameter #1 the value of 0)
F25 (set a feed rate)
o101 while [#1 LT 10]
  G1 X0
  G1 Y[#1/10] X1
  #1 = [#1+1] (increment the test counter)
o101 endwhile
M2 (end program)
```

Do While Example

```
#1 = 0 (assign parameter #1 the value of 0)
o100 do
  (debug, parameter 1 = #1)
  o110 if [#1 EQ 2]
    #1 = 3 (assign the value of 3 to parameter #1)
    (msg, #1 has been assigned the value of 3)
    o100 continue (skip to start of loop)
  o110 endif
  (some code here)
  #1 = [#1 + 1] (increment the test counter)
o100 while [#1 LT 3]
(msg, Loop Done!)
M2
```

Inside a while loop, *O- break* immediately exits the loop, and *O- continue* immediately skips to the next evaluation of the *while* condition. If it is still true, the loop begins again at the top. If it is false, it exits the loop.

5. Conditional

The *if* conditional consists of a group of statements with the same *o* number that start with *if* and end with *endif*. Optional *elseif* and *else* conditions may be between the starting *if* and the ending *endif*.

If the *if* conditional evaluates to true then the group of statements following the *if* up to the next conditional line are executed.

If the *if* conditional evaluates to false then the *elseif* conditions are evaluated in order until one evaluates to true. If the *elseif* condition is true then the statements following the *elseif* up to the next conditional line are executed. If none of the *if* or *elseif* conditions evaluate to true then the statements following the *else* are executed. When a condition is evaluated to true no more conditions are evaluated in the group.

If Endif Example

```
(if parameter #31 is equal to 3 set S2000)
o101 if [#31 EQ 3]
  S2000
o101 endif
```

If ElseIf Else EndIf Example

```
(if parameter #2 is greater than 5 set F100)
o102 if [#2 GT 5]
  F100
o102 elseif [#2 LT 2]
(else if parameter #2 is less than 2 set F200)
  F200
(else if parameter #2 is 2 through 5 set F150)
o102 else
  F150
o102 endif
```

Several conditons may be tested for by *elseif* statements until the *else* path is finally executed if all preceding conditons are false:

If Elseif Else Endif Example

```
(if parameter #2 is greater than 5 set F100)
O102 if [#2 GT 5]
  F100
(else if parameter #2 less than 2 set F200)
O102 elseif [#2 LT 2]
  F20
(parameter #2 is between 2 and 5)
O102 else
  F200
O102 endif
```

6. Repeat

The *repeat* will execute the statements inside of the repeat/endrepeat the specified number of times. The example shows how you might mill a diagonal series of shapes starting at the present position.

Repeat Example

```
(Mill 5 diagonal shapes)
G91 (Incremental mode)
o103 repeat [5]
... (insert milling code here)
G0 X1 Y1 (diagonal move to next position)
o103 endrepeat
G90 (Absolute mode)
```

7. Indirection

The O-number may be given by a parameter and/or calculation.

Indirection Example

```
o[#101+2] call
```

Computing values in O-words

For more information on computing values see the following sections

- Parameters
- Expressions
- Binary Operators
- Functions

8. Calling Files

To call a separate file with a subroutine name the file the same as your call and include a sub and endsub in the file. The file must be in the directory pointed to by *PROGRAM_PREFIX* or *SUBROUTINE_PATH* in the ini file. The file name can include **lowercase** letters, numbers, dash, and underscore only. A named subroutine file can contain only a single subroutine definition.

Named File Example

```
o<myfile> call
```

Numbered File Example

```
o123 call
```

In the called file you must include the oxxx sub and endsub and the file must be a valid file.

Called File Example

```
(filename myfile.ngc)
o<myfile> sub
  (code here)
o<myfile> endsub
M2
```

Note The file names are lowercase letters only so *o<MyFile>* is converted to *o<myfile>* by the interpreter. More information about the search path and options for the search path are in the INI Configuration Section.

9. Subroutine return values

Subroutines may optionally return a value by an optional expression at an *endsub* or *return* statement.

Return value example

```
o123 return [#2 *5]
...
o123 endsub [3 * 4]
```

A subroutine return value is stored in the *<_value>* predefined named parameter , and the *<_value_returned>* predefined parameter is set to 1, to indicate a value was returned. Both parameters are global, and are cleared just before the next subroutine call.

10. Errors

The following statements cause an error message and abort the interpreter:

- a `return` or `endsub` not within a sub defintion
- a label on `repeat` which is defined elsewhere
- a label on `while` which is defińed elsewhere and not referring to a `do`
- a label on `if` defined elsewhere
- a undefined label on `else` or `elseif`
- a label on `else`, `elseif` or `endif` not pointing to a matching `if`
- a label on `break` or `continue` which does not point to a matching `while` or `do`
- a label on `endrepeat` or `endwhile` no referring to a corresponding `while` or `repeat`

To make these errors non-fatal warnings on stderr, set bit 0x20 in the `[RS274NGC]FEATURE=` mask ini option.

Last updated 2020-05-19 20:27:42 MDT