NORG2

Introduction

Two years after having introduced NORG I feel that it is time for a revision (or reinvention) of a small language and I move on to NORG2.
Note that NORG2 draws some ideas and methods from the original NORG but is in no way compatible with it so you will have to rewrite NORG programs in order to execute them with NORG2.

While - in a certain sense - NORG2 is less esoteric than the original NORG language it still tries to be concise. It wasn't designed with having code golfing in mind but moves a little bit in that direction.

- Note that for a 'successful' code golfing language you would need a combination of a stack based language like FORTH combined with an array language like APL and using the full UTF8 alphabet. This is not what we have in mind here. -

Be aware that (contrary to NORG) NORG2 programs live in a plane (rectangular area).

Conventions

- Command names may sometimes be given surrounded by quotes, but these are not part of the command but signify that the text enclosed in quotes is meant literally.
- If text is not meant literally but rather its meaning then we usually enclose it in angle brackets like so: <number> doesn't represent the text number but an entry whose meaning is a number.

Basics

Each NORG2 program starts with defining the length of the square area that it 'lives' in. This length is given as a series of digits (at least one), terminated by a dot ('.'). Alternatively, you can define a rectangular area by giving two numbers, separated by an 'x' and terminated by a dot, namely: <number of columns>'x'<number of rows>'.'

The area consists of cells where each cell has three registers:

- an integer register containing an integer number (initially 0)
- a string register, containing a string (initially empty)
- an exec register, containing a string (initially empty)

- the integer register of the current cell is referred to as c.i.c.r
- the string register of the current cell is referred to as c.s.c.r
- the exec register of the current cell is referred to as c.e.c.r

Execution is affected by the position of a cursor. The cell to which the cursor currently points is called 'the current cell'.

If a command requires a second operand, this is usually (but not always) determined relative to the current cell as the operand direction **opdir**. Initially this direction is set to right (meaning that the 2nd operand is the cell just right to the current cell) but may be changed by the program, see below.

Initially, the cursor is set to the center cell of the area, which is given as cell(m,n) where m = <number of columns>/2 and n = <number of rows>/2 and / representing integer division (discarding the rest if any) and the upper left cell is cell(0,0).

The program continues with a list of commands which may be separated by newlines for readability (newline characters are ignored).

While most commands have a fixed length, some require the terminator character '.' or ';' (without quotes) to end them.

Program execution is strictly from left to right.

There are a number of global values:

- global integer value registers 0..9, initially set to 0. On program start global register 0 is selected, in the following these are referenced as g.i.v.r
- global string value registers 0..9, initially set to empty. On program start global register 0 is selected, in the following these are referenced as g.s.v.r
- global execution code registers 0..9, initially set to empty. On program start global register index 0 will be the first to be defined by an E. command, see below. If you define more than 10 such registers, the index wraps. These registers are referenced as g.e.c.r

Calling a program

The general format for calling a NORG2 program via the interpreter is as follows:

NORG2.exe <calling options> <norg2 program source>

NORG2.exe
the name of the interpreter, including the path if necessary

<calling options>
options for calling the interpreter (may be empty). Options are introduced by a minus sign. Available options:

-s = debug output for stack levels
-p = debug output for commands
-a = treat commands i,I as a,A (for tests)

<norg2 program>
The program to be executed, including path if necessary

program arguments are not supported.

Commands

com-mand	param	meaning	explanation
a or A	none	ask file	if no file is selected: select the file in the same directory that has the same name as the program, but the extension .nin, then proceed with the selected file. A = read the next line and set the c.s.c.r to its string value a = read the next line assume it contains a number and set the c.i.c.r to its value If eof is reached, the **g.i.v.r[3]** is set to 1.
b	0..3	move to base point	move the cursor to the (pre)defined point where b0 = (0,0) upper left b1 = (dim,0) upper right b2 = (0,dim) lower left b3 =(dim,dim) lower right points may be redefined using the p cmd
B	0..3	set base point	redefine the commands b0..b3 to pointing to the current cell
c	0..9 +(command)	condition	next command is only executed if **g.i.v.r[<digit>]** is 1 NOTE: - the sequences cC and cc are not supported - if the digit is missing, 0 is assumed
C	0..9 +(command)	condition	next command is only executed if **g.i.v.r[<digit>]** is not 1 NOTE: - the sequences Cc and CC are not supported - if the digit is missing, 0 is assumed
d	none	move down	move the cursor down
D	none	down	move the cursor down 3 steps
e	none	execute	execute the **c.e.c.r** content (see command E).

			This is done by inserting the code into the code string right after the current position.
E	string	set exec register	the E command is followed by a string and terminated by a semicolon ';' The **c.e.c.r** is set to the string value. Previous contents of the register are deleted. If the string inside the E command starts with a dot '.' then the next **g.e.c.r** will be defined instead of the local **c.e.c.r.** Defining more than 10 gecr's will result in overwriting existing ones, starting with **gecr[0].**
f	see explanation	find	format: f<dir><condition><flag> <dir> = d,r <condition> = 0..9,n,z <flag> = 0..9 find next cell in the direction given whose cicr has the searched value. If condition=0..9 then search for **g.i.v.r[<digit>]** If condition=n then search for non-zero If condition=z then search for zero if found then position the cursor to this cell and set givr[<flag>]=1 otherwise don't move and set **givr[<flag>]**=0
g	direction,i,j,0..9	get int	get the i.c.r from the cell in the specified direction and overwrite current cell. 0..9 = set it to **g.i.v.r[<digit>]** i = set it to the column index of the cell (x) j = set it to the row index of the cell (y)
G	direction,0..9	get string	get the s.c.r from the cell in the specified direction and overwrite current cell. 0..9 = set it to **g.s.v.r[<digit>]**
h	0..9	global exec	call global exec register given as the argument
i	none	input int	read the input from the console as an integer and place it in the c.i.c.r.

I	none	input str	read the input from the console as a string and place it in the c.s.c.r.
j	none	jump	end current e/h command sequence execution **Note:** not heavily tested.
J	2 directions	jump <n>	jump as many cells in the specified direction as given by the c.i.c.r . If c.i.c.r <= 0 then jump (once) in direction2
k	direction	decrement	decrement the c.i.c.r. by one. If the result is <= 0, then go in the direction specified
K	2 direction+	increment	increment the c.i.c.r. by one. If the result is >= i.c.r of direction1, then go in the direction2.
l	none	move left	move the cursor left
L	none	move left	move the cursor left 3 steps
m	direction or 0..9	action mode	change the **opdir** direction to the value given as the argument if the argument is a digit then reference **givr[<digit>]** or **gsvr[<digit>]**
n	none	newline	outputs a newline
N	none	newline	writes a newline to the file (see w,W)
o	none	output int	outputs the c.i.c.r value to the console
O	none	output string	outputs the c.s.c.r value to the console
r	none	move right	move the cursor right
R	none	move right	move the cursor right 3 steps
s	direction,c,0..9	send int	send the content of the c.i.c.r to the cell in the direction given as parameter, see below 0..9 = send it to **g.i.v.r[<digit>]** c = send it to the c.s.c.r (=convert to a string)
S	direction,c,0..9	send string	send the content of the c.s.c.r to the cell in the direction given as parameter, see below 0..9 = send it to the **g.s.v.r[<digit>]** c = send it to the c.i.c.r (=convert to integer)
t	string	set int register	the t command is followed by an integer and terminated by a dot '.' The c.i.c.r is set to the string value. Previous contents of the register are deleted.

T	string	set string register	the T command is followed by a string and terminated by a dot '.' The c.s.c.r is set to the string value. Previous contents of the register are deleted.
u	none	move up	move the cursor up
U	none	move up	move the cursor up 3 steps
v	direc-tion	copy exec	set the **c.e.c.r** to the content of the **e.c.r** given by the direction specified
w or W	none	write file	write to the file in the same directory that has the same name as the program, but the ex-tension .nou. New data is appended W = write c.s.c.r to the file w = write c.i.c.r to the file
x	2 direc-tions+	exchange	swap the values of the i.c.r's of the cells given by the directions relative to the current cell
X	2 direc-tions+	exchange	swap the values of the s.c.r.'s of the cells given by the directions relative to the current cell
Z	none	exit	program ends.
+	none	add	add the integer content of the **opdir** cell to the c.i.c.r
-	none	subtract	subtract the integer content of the **opdir** cell from the c.i.c.r
*	none	multiply	multiply the integer content of the c.i.c.r and the **opdir** cell and set the c.i.c.r to the result
/	none	divide	divide the integer content of the c.i.c.r cell by the **opdir** cell content and set the c.i.c.r to the result
%	0..9	mode	get the modulus of the c.i.c.r cell by the **opdir** cell and set the **g.i.v.r[<digit>]** to the result
'='	0..9	compare equal	compare the int. content of the c.i.c.r to the **opdir** cell content and set **g.i.v.r[<digit>]** to the result (1=equal, 0 = unequal)
<	0..9	compare less	compare the int. content of the c.i.c.r to the **opdir** cell content and set **g.i.v.r[<digit>]** to

			the result (1= current < l/r, 0 = otherwise)
>	0..9	compare greater	compare the int. content of the c.i.c.r to the **opdir** cell content and set **g.i.v.r[<digit>]** to the result (1= current > l/r, 0 = otherwise)
[	none	minimum	compare the int. content of the c.i.c.r to the **opdir** cell content and set the c.i.c.r to the minimum of both values
]	none	maximum	compare the int. content of the c.i.c.r to the **opdir** cell content and set the c.i.c.r to the maximum of both values
&	2 digits <m><n>	and	logically compute **g.i.v.r[<m>]** and **g.i.v.r[n]** where n is the digit given as parameter, result goes to **g.i.v.r[m]**
|	2 digits <m><n>	or	logically compute **g.i.v.r[<m>]** or **g.i.v.r[n]** where n is the digit given as parameter, result goes to **g.i.v.r[<m>]**
!	0..9	not	invert **g.i.v.r[<digit>]** logically: if g.i.v.r. = 0 then set it to 1 otherwise set it to 0
$	<,>,= and digit +,&,.,l	str actions	'$<': as < but comparing c.s.c.r '$>': as > but comparing c.s.c.r '$=': as = but comparing c.s.c.r '$+': adding **opdir** cell s.c.r to c.s.c.r '$&': prepending **opdir** cell s.c.r before c.s.c.r '$.': append a dot to c.s.c.r. '$l': set c.i.c.r to the string length of c.s.c.r
?	s,- % and digit	int actions	?s: set c.i.c.r to its sign (1 if positive, 0 if 0, -1 if negative) ?-: c.i.c.r = -c.i.c.r ?%<n>: if c.i.c.r. is divisible by **opdir** content then set **g.i.v.r[<n>]** = 1, otherwise = 0
#	i and direction+ or 2 directions	split	# + i + dir1 = If the c.s.c.r is empty, then go in the direction specified, otherwise: the c.s.c.r is split into a head and a tail. The head length is determined by the c.i.c.r. Results are assigned to c.s.c.r. = tail and **opdir** s.c.r=head. # + dir1 + dir2 =

			If the c.s.c.r is empty, then go in the direction dir2(!) otherwise: the c.s.c.r is split into a head and a tail. The head end is determined by the separator given in cell dir1. Results are assigned to c.s.c.r. = tail and **opdir s.c.r** = head. The separator is not included and must be a single character (if the string containing the separator is longer then the first char is used).

direction = one of d,u,r,l,D,U,R,L
direction+ = direction or c(center=this cell)

Notes

cursor management

If a non-existing cell is referenced by one of the commands r,l,u,d,s,S,x,X,R,L,U,D the cursor is set in the following way:

- direction right: referenced cell is the leftmost cell in the same line
- direction left: referenced cell is the rightmost cell in the same line
- direction up: referenced cell is the bottom cell in the same line
- direction down: referenced cell is the topmost cell in the same line

commenting

- The language (currently) doesn't support comments. It is suggested to document programs by using some kind of spreadsheet.
- If you want to include your name as the author of a program, you may achieve this by using the T-command like so:
  TAuthor is John Doe.T.

(current) design principles:

1) Values are bound to cells with the exception of some global variables (givr, gsvr, gecr) the amount of which will remain restricted.
2) Only code defined in exec registers may be executed, no exec of string registers will be introduced.
3) References to 2nd and 3rd operands may be extended to include givr etc. if useful.
4) Commands will remain one-letter-words but may have a variable number of arguments (affecting their operation). Will probably remain restricted to ASCII chars 32-126.
5) Naming of commands may change if I feel that a different choice provides a better mnemonic.
6) Explicit loop commands should be avoided since looping in NORG-style is governed by moving from cell to cell and invoking its code in a tail-recursion-like fashion.

Possible Future enhancements

- '_': simple replace: if csvr contains lrdir then replace it by ... and goto dir, also should have a count switch
- introducing multiple areas in a program where each new area is started by char 'P' and can be 'called' by other areas. Data exchange is via givr/gsvr so somewhat restricted. The calling area has to supply the dimension for the called area.
- a cmd 'V' that forks code execution in two parts, depending on givr like so 'V'<n>(part of givr[<n>] true)'\'(part if givr wrong)'\' or so.
- a method to distribute values or code to other cells,like: down the same col and/or rightwards the same row etc. 'H'<type><num-down>,<num-right>. or so
- a cmd to temporarily shadow a cell value ':s'and revert to it afterwards ':u' without using other cell values or global values (shadow/unshadow)
- modifier prefix ':' to extend the abilities of the next cmd or introduce some special features

so far unused letters:

F,H,M,p,P,q,Q,V,y,Y,z.
:^"{}()\~'_,@