**Description of the Next HLA language**

NextHLA v2 is a sequence of statements ; there is no real organised structure as such. Everything behaves in the same way, so language elements can be mixed almost randomly ; you can put a procedure in code and it will work. However, it is not advised !

Assignments/Expressions

An assignment expression is a sequence of terms followed by a binary operator and other terms. The assignment itself is a binary operator.

A term is either an address, or a constant. Terms are :-

Integer constant 0-65535

Hexadecimal constant $0-$FFFF

Character constant ‘<char>’ (32..127 only)

String constant address of ASCIIZ String

Named constant <identifier>

Variable constant @<identifier>

Address of block of memory [<constant, size in bytes:]

A negative constant -<any constant>

Variable address <identifier>

When an identifier is found (e.g. it was not preceded by an operator) it can be ; an expression/assignment, a while/if/for structure, or a procedure call. Identifiers begin with underscore or a-z, and can have underscore, a-z, 0-9 : and . characters. : should not be used (it will be used for a cheap OOP design later)

These are determined by the name of the identifier. Because identifiers are typeless, the procedure call is detected by the following parentheses ()

Binary Operators

The standard binary operations are + - \* / & | ^ ! ? which are achieved using a mixture of code and optimisations (e.g. inc dec and shifts). There is an assignment binary operator - the : operator

e.g. to add a and b and store at C you would use:

a+b:c

this operator is unusual in that in can be followed by either identifier, (identifier!term) or (identifier?term). All other binary operators take a single standard term.

The : does not change the ‘current value’. This is the *only* use of parenthesis in expressions and it is for syntactic clarity.

So it permits a+b:(c!4)and a+b:(c?4)

a+b:c?4 does something different (assigns a+b to c, then adds 4 to it and does a byte read)

If two terms are found consecutively e.g. a b then the working value is b.

Procedure calls

Procedure calls consist of an identifier, and bracketed parameters. The brackets are present even if there are no parameters. Parameters are static, and are stored in the parameter space, which is a contiguous block of memory that belongs to the procedure. Each procedure identifier has a memory address (may be zero) and a number of parameters, that are local variables.

e.g. hello(1,2,a+4) calls procedure hello. If this is declared as proc hello(x,y,z) then x = 1, y = 2 and z = a+4 .

Procedure code is the single instruction following the proc definition ; after this all locals are dumped.

The last calculated value is ‘returned’ and can be processed using operators etc.

Variables are declared using the keywords local and global followed by a single identifier. So you comment them :) They can be declared at any time, but cannot be forward referenced. Local identifiers always have priority. At the end of the module any identifier beginning with an underscore is dumped.

Code can be grouped together using { } and the ; can be used for clarity if you wish. It has no actual effect and is not required syntactically.

A function would look like this.

proc add.together(a,b,c)

{

local total

a+b+c : total

total

}

and you can write add.together(3,4,5) : x for example. When the application is run, all functions called <something>.boot() are run in the order they are compiled to allow for module initiation. The last of these is your main program.

**Structures**

If (<expr>) <cond> <statement>

if+ (a) { count+1:count }

while (<expr>) <cond> <statement>

while= (keypressed()) { do.waiting(count) count+1:count }

for (<expr>) <statement>

for (100) { flash.stars() }

(downcounts expr from (n-1)-0, stored in index if defined)

<cond> conditions are currently <0, =0, or #0 (the default if omitted)

Note an instruction is deemed to be a single language element. So if you want to write the equivalent of “if (a < 4) b = b + 1” you might think of writing.

If (a-4) <0 b+1:b

this is wrong. What is executed is this, because the single instruction is the first b.

If (a-4) <0 b

+1:b

so while it is not mandatory it is advisable to always use curly brackets as in ‘C’.

if (a-4) <0 { b+1:b }