Core 4.0 Implementation of the for Construct

Let's start by saying that every construct must have a unique identifier. For this we define a global function that deals with a global counter:

```
public function unique_id ( null ) {{ if ( ..id ) {{ ..id = ..id + 1 ; return text ..id }} else {{ set ..id = 0 ; return "0" }} ;
```

Then we move on to the language level to define the statement:

```
for&: for|(str) (|.INIT|(**) ;|.COND|(**) ;|.ONCYCLE|(**) )|(str) {{|.%STATEMENT|(**) ;|(**)|(opt) }}
```

We do everything in autoexec, simply because in this case it is enough, putting the code of the various parts, in the right place marked by the right labels, and we are finished:

```
public function for.autoexec ( null ) {{
        set ::id = "for_" + exec unique_id ( null ) ;
        set ::code = ::id + ":" + ::INIT.code + "\n" + ::id + "_cond:" + ::COND.code + "\ncmp eax , 0\njz " + ::id + "_end" ;
        for ( set .i = 0 ; .i < size ::STATEMENT ; .i = .i + 1 ) {{ ::code = ::code + ::STATEMENT[ .i ].code }} ;
        ::code = ::code + "\n" + ::id + "_onclycle:" + ::ONCYCLE.code + "\njmp " + ::id + "_cond" + "\n" + ::id + "_end:" ;
        case ( ::__CALLER type SYSstart ) {{ print ::code }}
}}
```

```
Core 4.0:for ( .a = 0 ; .a < 6 ; .a = .a + 1 ) {{ .b = .b * 2 ; .c = .c - 1 }}

for_0:
push .a
mov eax , 0
pop ebx
mov [ ebx ] , eax
for_0_cond:
mov eax , [ .a ]
cmp eax , 6
mov eax , 0
setl al
cmp eax , 0
jz for_0_end
push .b
mov eax , [ .b ]
mul eax , 2
pop ebx
mov [ ebx ] , eax
push .c
mov eax , [ .c ]
sub eax , 1
pop ebx
mov [ ebx ] , eax
for_0_onclycle:
push .a
mov eax , [ .a ]
add eax , 1
pop ebx
mov [ ebx ] , eax
jmp for_0_cond
for_0_end:
```