EC

(A poetry mode C-based Imperative programming language)

Baguinang, Jude Clarence
Baleña, Richard
Urmeneta, Kriztoper

Paradigm: Imperative
Domain: Educational, Teaching programming
Description: EC is a C-based programming language which is also imperative. The twist for this language is that it's design is in poetry mode inspired from Ruby where 'do ... end' are preferred over '{ ... }'.

EBNF Syntax of EC

```
<program>           → <main>
<main>              → main do <paragraph> end
<paragraph>         → {( <sentence> | <stmt_block> )}
<sentence>          → {( <assignment> | <operation> | <comment> |
                      <print> | <print_with_newline> | <scanner> )}
<assignment>        → <word> = ( <operation> | <word> | <constant> |
                    <string> )
<operation>         → <arithmetic>
<word>              → @<lower_alpha>{( <lower_alpha> | <upper_alpha>
                      | <number>)}
<lower_alpha>       → ( a | b | c | d | e | f | g | h | i | j | k | l | m | n |
               o | p | q | r | s | t | u | v | w | x | y | z )
<upper_alpha>       → ( A | B | C | D | E | F | G | H | I | J | K | L | M
               | N | O | P | Q | R | S | T | U | V | W | X | Y | Z )
<constant>          → [ - ]<number>{<number>}[.<number>{number}]
<number>            → ( 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 )
<string>            → '{? all_characters ?}'
<stmt_block>        → ( <conditional> | <iteration> | <comment> )
<conditional>       → if <condition> do <sentence> { else if
        <condition> do <sentence> } [ else do <sentence> ] end
<iteration> → for <assignment> ; <condition> ; <(operation|assignment)>
                      do <sentence> end | while <condition> do <sentence>
                    end | do <sentence> while <condition> end
<condition>         → [ not ] <expression> ( and | or | == | != | < | > | <=
                      | >= ) <expression>
<expression>      → ( <word> | <constant> | <string> )
```

```
<arithmetic>        → <term> <arith_op> <term>
<term>              → <constant> | <word>
<arith_op> → ( + | - | <hi_order_op> )
<hi_order_op>       → ( / | * | % )
<print>             → print ( <word> | <string> ){ + ( <word> | <string> )}
<print_with_newline>  → puts ( <word> | <string> ){ + (<word> | <string>)}
<scanner>           → scan <word>
<comment>           → /* {?all_characters?} */
```

Sample codes:

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
main do
        /* Print 'Hello, World!' with new line */
        puts 'Hello, World!'
end
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
main do
        @b = 4
        for @a = 1 ; @a < @b ; @a = @a + 1 do
                puts @a
        end
end
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
main do
        @a = 1
        while @a < 4 do
                puts @a
        end
end
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
main do
	@a = 1
	do
		puts @a
	while @a < 4 end
end
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
main do
	scan @a
	print 'You entered ' + @a
end
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
main do
	@a = 3
	@b = 2
	@c = @a + @b
	puts @c
	@c = @a - @b
	puts @c
	@c = @a * @b
	puts @c
	@c = @a / @b
	puts @c
	@c = @a % @b
	puts @c
end
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
main do
        scan @a
        if @a == 0 do
                puts ‘equals’
        else if @a < 0 do
                puts ‘less than’
        else do
                puts ‘else’
        end
end
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```