El lenguaje de programación JKL

[versión 1.0, 2006-02-20, JosuKa]

1. Referencia del lenguaje JKL

1a. Léxico

Los identificadores (`id`) siguen el criterio habitual (solo letras y dígitos, empezando por letra); mayúsculas y minúsculas son distintas. Las palabras reservadas son:

`program`, `var`
`if`, `then`, `else`, `while`, `do`, `for`, `to`, `downto`, `begin`, `end`
`read`, `write`, `writec`, `writeln`
`and`, `or`, `not`

y también mayúsculas y minúsculas son distintas.

Las constantes literales enteras (`num`) son una secuencia de dígitos (al menos uno). Existen constantes literales de tipo cadena (`cad`), que se escriben usando comillas dobles (por ejemplo, `"abc"` o `""`; dentro de las comillas no se admiten ni tabuladores ni cambios de línea ni comillas).

Diversas cadenas de signos especiales representan operadores o símbolos de puntuación:

`+` `-` `*` `/` `%` `<` `>` `<=` `>=` `=` `<>`
`:=` `(` `)` `;` `.`

Finalmente, los blancos separan distintos tokens, al igual que cambios de línea y tabuladores. Los comentarios comienzan con la secuencia `//` y se extienden hasta el final de la línea.

1b. Gramática

La notación que utilizaremos para describir la gramática es la reflejada en la siguiente tabla:

⟨*Variable*⟩	indica que es una variable o no terminal
`reservada`	se trata de un terminal (unidad léxica)
`'x'`	lexema entre comillas simples; se trata de un terminal
[*x*]	significa que *x* es opcional
*x**	cero o más ocurrencias de *x*
x+	una o más ocurrencias de *x*
|	separa las alternativas

Un programa está compuesto por una lista (quizá vacía) de declaraciones de variables y por una sentencia compuesta como cuerpo:

⟨*prog*⟩ → `program id ';'` ⟨*decl*⟩* ⟨*sentc*⟩ `'.'`
⟨*decl*⟩ → `var id ';'`
⟨*sentc*⟩ → `begin` ⟨*sent*⟩* `end`

Las sentencias que se admiten en el lenguaje JKL son similares a las de PASCAL:

⟨*sent*⟩ → `';'`
| `id ':='` ⟨*expr*⟩ `';'`
| ⟨*sentc*⟩
| `read id ';'`
| `write` ⟨*expr*⟩ `';'`
| `writec cad ';'`
| `writeln ';'`
| `if` ⟨*expr*⟩ `then` ⟨*sent*⟩ [`else` ⟨*sent*⟩]
| `while` ⟨*expr*⟩ `do` ⟨*sent*⟩
| `do` ⟨*sent*⟩ `while` ⟨*expr*⟩ `';'`
| `for id ':='` ⟨*expr*⟩ `to` ⟨*expr*⟩ `do` ⟨*sent*⟩
| `for id ':='` ⟨*expr*⟩ `downto` ⟨*expr*⟩ `do` ⟨*sent*⟩

Sin embargo, se sigue la norma de C de que cada sentencia esté terminada en `;` y se admite la sentencia vacía (primera regla de ⟨*sent*⟩).

Una expresión es una constante literal (`num`), una variable (`id`), otra expresión entre paréntesis o la aplicación de operadores binarios o unarios a otras expresiones. Sólo existe un tipo de datos: enteros. La precedencia de los operadores (de mayor a menor) viene dada por la siguiente tabla:

Nombre token	***Operador***	***Semántica***
`op_not op_adit`	`not + -` (unarios)	negación lógica, signo, cambio de signo
`op_mult`	`* / %`	producto, división, resto de la divisón
`op_adit`	`+ -` (binarios)	suma, resta
`op_rel`	`< <= == != >= >`	comparaciones
`op_and`	`and`	and lógico
`op_or`	`or`	or lógico

siendo la asociatividad de izquierda a derecha para los operadores binarios.

La parte de gramática que refleja lo anterior con recursividad por la izquierda es:

⟨*expr*⟩ → ⟨*eand*⟩
| ⟨*expr*⟩ `op_or` ⟨*eand*⟩
⟨*eand*⟩ → ⟨*erel*⟩
| ⟨*eand*⟩ `op_and` ⟨*erel*⟩
⟨*erel*⟩ → ⟨*arit*⟩
| ⟨*erel*⟩ `op_rel` ⟨*arit*⟩
⟨*arit*⟩ → ⟨*term*⟩
| ⟨*arit*⟩ `op_adit` ⟨*term*⟩
⟨*term*⟩ → ⟨*fact*⟩
| ⟨*term*⟩ `op_mult` ⟨*fact*⟩
⟨*fact*⟩ → ⟨*rando*⟩
| `op_not` ⟨*fact*⟩
| `op_adit` ⟨*fact*⟩
⟨*rando*⟩ → `num`
| `id`
| `'('` ⟨*expr*⟩ `')'`

En un programa en el lenguaje JKL todas las variables son globales. Aparte de eso, todo identificador debe ser declarado antes de ser usado, y es obvio que no se admite dos veces el mismo nombre en distintas declaraciones.

Las sentencias `if-then`, `if-then-else`, `while`, `do` y `for` tienen la semántica usual de lenguajes como PASCAL o C. Por otro lado, las operaciones de entrada/salida en el lenguaje JKL se realizan mediante sentencias especiales:

- `read id` : lee un entero de la entrada estándar y lo asigna a la variable `id`;
- `write` ⟨*expr*⟩: escribe el entero resultado de evaluar ⟨*expr*⟩ en la salida estándar;
- `writec cad` : escribe la cadena literal `cad` en la salida estándar;
- `writeln` : escribe un salto de línea en la salida estándar.

1c. Ejemplo: prog1.jkl

```
// Comentario: programa de prueba
// (solo para comprobar la sintaxis)

program prueba;
var i;
begin
        i := 128;
        writec "Valor de i: ";
        write i;
        writeln;
        read i;
        writec "Valor de i: ";
        write i;
        writeln;
        if -12 <= +15 and not -13 <> +27 then i := -27;
        ;;;;
        begin
                ;;;
        end
        if i <= 2 then i := 27;
        else i := 28;
        while i < 5 do ;
        while i < 5 do i := 7;
        while i < 5 do begin
                i := 5;
                write i;
        end
        do ; while i < 6;
        do write i; while i < 8;
        do begin
                write i;
                i := i + 7;
        end while i < 10;
        for i := i + 5 to i - 7 do ;
        for i := (i + 5) * 22 to i - 7 do
                write i;
        for i := (i + 5) * 22 to i - 7 do begin
                read i;
                write i;
        end
        for i := i + 5 downto i - 7 do ;
        for i := (i + 5) * 22 downto i - 7 do
                write i;
        for i := (i + 5) * 22 downto i - 7 do begin
                read i;
```

```
                write i;
        end
        for i := (i + 5) * 22 downto i - 7 do begin
                ;;;
                if i <= 2 then i := 27;
                else begin
                        while i < 5 do ;
                        while i < 5 do i := 7;
                        while i < 5 do begin
                                i := 5;
                                write i;
                                do ; while i < 6;
                                do write i; while i < 8;
                                do begin
                                        write i;
                                        i := i + 7;
                                end while i < 10;
                                for i := i + 5 to i - 7 do ;
                                for i := (i + 5) * 22 to i - 7 do
                                        write i;
                                for i := (i + 5) * 22 to i - 7 do begin
                                        read i;
                                        write i;
                                end
                                for i := i + 5 downto i - 7 do ;
                                for i := (i + 5) * 22 downto i - 7 do
                                        write i;
                                for i := (i + 5) * 22 downto i - 7 do begin
                                        read i;
                                        write i;
                                end
                        end
                        read i;
                end
        end
end .

// El programa tiene que acabar con un punto
```

1d. Ejemplo: fact.jkl

```
// Programa de prueba: factorial iterativo
program factorial;
var i;
var res;
begin
        writec "Escriba un número: ";
        read i;
        writec "Valor de i: ";
        write i;
        writeln;
        writec "Se va a calcular el factorial";
        writeln;
        res := 1;
        while i >= 1 do begin
                res := res * i;
                i := i - 1;
        end
        writec "Resultado: ";
        write res;
        writeln;
end .
```

2. Análisis sintáctico descendente

2a. Gramática:

```
 1   prog  ->  PPROG  ID  P_COMA  decl  sentc  PUNTO
 2   decl  ->  lambda
 3          |  PVAR  ID  P_COMA  decl
 4   sentc ->  PBEGIN  lsent  PEND
 5   lsent ->  lambda
 6          |  sent  lsent
 7   sent  ->  P_COMA
 8          |  ID  ASIGN  expr  P_COMA
 9          |  sentc
10          |  PREAD  ID  P_COMA
11          |  PWRITE  expr  P_COMA
12          |  PWRITC  CAD  P_COMA
13          |  PWRITL  P_COMA
14          |  PIF  expr  PTHEN  sent  pelse
15          |  PWHILE  expr  PDO  sent
16          |  PDO  sent  PWHILE  expr  P_COMA
17          |  PFORP  ID  ASIGN  expr  PTODO  expr  PDO  sent
18   pelse ->  lambda
19          |  PELSE sent
20   expr  ->  eand  exprp
21   exprp ->  lambda
22          |  OP_OR  eand  exprp
23   eand  ->  erel  eandp
24   eandp ->  lambda
25          |  OP_AND  erel  eandp
26   erel  ->  arit  erelp
27   erelp ->  lambda
28          |  OP_REL  arit  erelp
29   arit  ->  term  aritp
30   aritp ->  lambda
31          |  OP_ADIT  term  aritp
32   term  ->  fact  termp
33   termp ->  lambda
34          |  OP_MULT  fact  termp
35   fact  ->  OP_NOT  fact
36          |  OP_ADIT  fact
37          |  rando
38   rando ->  NUM
39          |  ID
40          |  PAR_ABR  expr  PAR_CER
```

2b. PRIMERO

```
prog  ->  PPROG
decl  ->  lambda, PVAR
sentc ->  PBEGIN
lsent ->  lambda, P_COMA, ID, PBEGIN, PREAD, PWRITE, PWRITC, PWRITL, PIF, PWHILE,
          PDO, PFORP
sent  ->  P_COMA, ID, PBEGIN, PREAD, PWRITE, PWRITC, PWRITL, PIF, PWHILE, PDO,
          PFORP
pelse ->  lambda, PELSE
expr  ->  OP_NOT, OP_ADIT, NUM, ID, PAR_ABR
exprp ->  lambda, OP_OR
eand  ->  OP_NOT, OP_ADIT, NUM, ID, PAR_ABR
eandp ->  lambda, OP_AND
erel  ->  OP_NOT, OP_ADIT, NUM, ID, PAR_ABR
erelp ->  lambda, OP_REL
arit  ->  OP_NOT, OP_ADIT, NUM, ID, PAR_ABR
aritp ->  lambda, OP_ADIT
```

```
term  ->  OP_NOT, OP_ADIT, NUM, ID, PAR_ABR
termp ->  lambda, OP_MULT
fact  ->  OP_NOT, OP_ADIT, NUM, ID, PAR_ABR
rando ->  NUM, ID, PAR_ABR
```

2c. SIGUIENTE

```
prog  ->  DOLAR
decl  ->  PBEGIN
sentc ->  PUNTO, P_COMA, ID, PBEGIN, PREAD, PWRITE, PWRITC, PWRITL, PIF, PWHILE,
          PDO, PFORP, PEND, PELSE
lsent ->  PEND
sent  ->  P_COMA, ID, PBEGIN, PREAD, PWRITE, PWRITC, PWRITL, PIF, PWHILE, PDO,
          PFORP, PEND, PELSE
pelse ->  P_COMA, ID, PBEGIN, PREAD, PWRITE, PWRITC, PWRITL, PIF, PWHILE, PDO,
          PFORP, PEND, PELSE
expr  ->  P_COMA, PTHEN, PDO, PTODO, PAR_CER
exprp ->  P_COMA, PTHEN, PDO, PTODO, PAR_CER
eand  ->  OP_OR, P_COMA, PTHEN, PDO, PTODO, PAR_CER
eandp ->  OP_OR, P_COMA, PTHEN, PDO, PTODO, PAR_CER
erel  ->  OP_AND, OP_OR, P_COMA, PTHEN, PDO, PTODO, PAR_CER
erelp ->  OP_AND, OP_OR, P_COMA, PTHEN, PDO, PTODO, PAR_CER
arit  ->  OP_REL, OP_AND, OP_OR, P_COMA, PTHEN, PDO, PTODO, PAR_CER
aritp ->  OP_REL, OP_AND, OP_OR, P_COMA, PTHEN, PDO, PTODO, PAR_CER
term  ->  OP_ADIT, OP_REL, OP_AND, OP_OR, P_COMA, PTHEN, PDO, PTODO, PAR_CER
termp ->  OP_ADIT, OP_REL, OP_AND, OP_OR, P_COMA, PTHEN, PDO, PTODO, PAR_CER
fact  ->  OP_MULT, OP_ADIT, OP_REL, OP_AND, OP_OR, P_COMA, PTHEN, PDO, PTODO,
          PAR_CER
rando ->  OP_MULT, OP_ADIT, OP_REL, OP_AND, OP_OR, P_COMA, PTHEN, PDO, PTODO,
          PAR_CER
```

2d. Tabla LL(1)

```
prog  ->  [PPROG]1
decl  ->  [PBEGIN]2, [PVAR]3
sentc ->  [PBEGIN]4
lsent ->  [PEND]5,
          [P_COMA, ID, PBEGIN, PREAD, PWRITE, PWRITC, PWRITL, PIF, PWHILE, PDO,
           PFORP]6
sent  ->  [P_COMA]7, [ID]8, [PBEGIN]9, [PREAD]10, [PWRITE]11, [PWRITC]12,
          [PWRITL]13, [PIF]14, [PWHILE]15, [PDO]16, [PFORP]17
pelse ->  [P_COMA, ID, PBEGIN, PREAD, PWRITE, PWRITC, PWRITL, PIF, PWHILE, PDO,
           PFORP, PEND, PELSE*]18,
          [PELSE]19
expr  ->  [OP_NOT, OP_ADIT, NUM, ID, PAR_ABR]20
exprp ->  [P_COMA, PTHEN, PDO, PTODO, PAR_CER]21, [OP_OR]22
eand  ->  [OP_NOT, OP_ADIT, NUM, ID, PAR_ABR]23
eandp ->  [OP_OR, P_COMA, PTHEN, PDO, PTODO, PAR_CER]24, [OP_AND]25
erel  ->  [OP_NOT, OP_ADIT, NUM, ID, PAR_ABR]26
erelp ->  [OP_AND, OP_OR, P_COMA, PTHEN, PDO, PTODO, PAR_CER]27, [OP_REL]28
arit  ->  [OP_NOT, OP_ADIT, NUM, ID, PAR_ABR]29
aritp ->  [OP_REL, OP_AND, OP_OR, P_COMA, PTHEN, PDO, PTODO, PAR_CER]30,
          [OP_ADIT]31
term  ->  [OP_NOT, OP_ADIT, NUM, ID, PAR_ABR]32
termp ->  [OP_ADIT, OP_REL, OP_AND, OP_OR, P_COMA, PTHEN, PDO, PTODO, PAR_CER]33,
          [OP_MULT]34
fact  ->  [OP_NOT]35, [OP_ADIT]36, [NUM, ID, PAR_ABR]37
rando ->  [NUM]38, [ID]39, [PAR_ABR]40
```

* El conflicto por la ambigüedad del ELSE se resuelve quitando la regla 18 de la casilla ELSE.