Procesadores del Lenguaje

Segunda aproximación

Autores: Álvaro Marco Pérez (100383382), David Rico Menéndez (100384036)
Datos: Grupo de trabajo 2

Índice

1. Estudio del código

Se ha realizado un estudio previo de la gramática con el objetivo de comprender el alcance inicial de la gramática.
La gramática original del fichero es:

```
axioma:          expresion '\n'
                 r_expr
           |   VARIABLE '=' expresion '\n'
                 r_expr
           ;
r_expr:          /* lambda */
           |   axioma
           ;
expresion:       termino
           |   expresion '+' expresion
           |   expresion '-' expresion
           |   expresion '*' expresion
           |   expresion '/' expresion
           ;
termino:         operando
           |   '+' operando %prec SIGNO_UNARIO
           |   '-' operando %prec SIGNO_UNARIO
           ;
operando:        VARIABLE
           |   NUMERO
           |   '(' expresion ')'
           ;
```

2. Diseño Jerárquico

Se ha tenido en cuenta la precedencia así como estudiado la gramática propuesta

3. Acciones semánticas

Utilizando la misma estructura aprendida del ejercicio de generación de código diferida, aunque en este caso como ya está predefinido como cadena, no hace falta usar $X.cadena. Queda de la siguiente manera:

```
|   expresion '+' expresion
     { sprintf(temp,"(+%s%s)", $1, $3); $$=genera_cadena (temp); }
|   expresion '-' expresion
     { sprintf(temp,"(-%s%s)", $1, $3); $$=genera_cadena (temp); }
|   expresion '*' expresion
     { sprintf(temp,"(*%s%s)", $1, $3); $$=genera_cadena (temp); }
|   expresion '/' expresion
     {sprintf(temp,"(/%s%s)",$1,$3); $$=genera_cadena (temp); }
```

4. Variables

Para las variables, hemos de tener en cuenta tres casos:

1. Definición de variables: Para ello, se añade una nueva regla en axioma la cual imprime la asignación en formato de LISP

   ```
   |   VARIABLE '=' expresion '\n'
             { printf ("( setq %c %s )\n", $1, $3) ;}
       r_expr
   ```

2. Input de variable sola y uso de variables en expresiones: Para ello, debemos de modificar las acciones semánticas de VARIABLE de la siguiente manera

   ```
   VARIABLE
      { sprintf (temp, "%c", $1) ; $$=genera_cadena (temp); }
   ```

5. Reglas referentes a la impresión

Para la impresion de datos, se debe detectar un # expresion, y transformar en un formato que el intérprete de lisp entienda, siendo esta (print expresion).
Por ello, modificamos **axioma** introduciendo una nueva regla:

```
|    '#' expresion '\n'
           { printf ("( print %s )\n", $2) ; }
     r_expr
```

En este caso, la acción semántica imprime la expresión entre parentesis, precedida por print

6. Conclusiones

Esta práctica nos ha parecido una buena aproximación a los conceptos a tratar en la práctica final. Las pautas secuenciales y los documentos de apoyo han ayudado al desarrollo de la práctica.