PROCESADORES DEL LENGUAJE

PRÁCTICA OBLIGATORIA: ANALIZADORES LÉXICOS Y SINTÁCTICO

Autores:

Daniel Requena Garrido

David Paredes Gómez

Tabla de contenido

PARTE OBLIGATORIA

Especificación Léxica

En cuanto a las especificaciones léxicas del lenguaje fuente, para poder reconocer la gramática, hemos creado en total 13 producciones:

1. ***IDENTIFIER***: reconoce los identificadores
2. ***NUMERIC_INTEGER_CONST***: reconoce las constantes numéricas enteras
3. ***NUMERIC_REAL_CONST***: reconoce las constates numéricas reales y esta compuesta por 3 producciones, que permiten reconocer dichos números reales de formas diferentes, denominadas:
   - 3.1: ***FIJO***: reconoce un real utilizando el carácter '.' para separar la parte entera de la decimal
   - 3.2: ***EXPONENCIAL***: reconoce un real expresado de forma exponencial
   - 3.3: ***MIXTO***: reconoce un real compuesto por las 2 producciones anteriores
4. ***STRING_CONST***: reconoce las constantes literales y esta compuesta por 2 producciones, una para cada tipo de comillas, llamadas:
   - 4.1: ***SIMPLE***: reconoce constantes literales mediante comillas simples
   - 4.2: ***DOBLE***: reconoce constantes literales mediante comillas dobles
5. ***COMENTARIO***: reconoce los comentarios de propósito general y esta formado por otras 2 producciones denominadas:
   - 5.1: ***CORCHETE***: reconoce comentarios para el caso de una sola línea
   - 5.2: ***PARENTESIS_ASTERICO***: reconoce comentarios para el caso de varias líneas
6. **WHITESPACE**: ignora los espacios, saltos de líneas y tabulaciones

Los comentarios están puestos en el canal "hidden" para que sean reconocidos, pero no mostrados en el html final.

Especificación sintáctica

En cuanto a la incorporación de la especificación sintáctica del programa fuente, para que la gramática usada sea LL(1), hay que tener en cuenta las condiciones de dicha gramática, es decir, sin recursividad por la izquierda y un símbolo de anticipación es suficiente.

Hemos solucionado la recursividad por la izquierda tanto de la propia declaración (dcllist) como de la sentencia (sentlist). También y dentro de la zona de declaraciones, hemos solucionado la recursividad por la izquierda de la declaracion del listado de constantes (ctelist) y del listado de definición de variables (defvarlist). Y, por último, en la zona de sentencias, hemos solucionado la recursividad por la izquierda de la sentencia de las expresiones (exp)

Por otro lado, hemos sustituido el carácter 'ʌ' por el espacio en blanco ya que así es como se define dicho carácter en una gramática ANTLR.

Daniel Requena Garrido P.L 2021-2022
David Paredes Gómez G-09

Especificación de la traducción dirigida por la sintaxis

La explicación de esta parte se dividirá en 3 secciones destacables:

1. Clase Inicializar

   Para empezar, nos definimos 2 arraylist denominados “cabeceras” y “func_proc” que contendrán las cabeceras de los subprogramas, tanto funciones como procedimientos, y el cuerpo de dichos subprogramas respectivamente. Utilizamos otra variable “prg_principal” con el contenido del código principal excluyendo los subprogramas.
   Terminando con las variables, utilizamos un conjunto de variables de tipo String para definir el estilo no semántico de palabras reservadas, identificadores y constantes.
   A continuación, definimos un método llamado “ImprimeOperacion” que mostrara todas las cabeceras de los subprogramas con sus correspondientes enlaces para referenciar al cuerpo del propio subprograma.
   Por otro lado, el método “ImprimeSubprograma” no solo mostrara las cabeceras si no también los cuerpos de cada subprograma.
   Otro método que encontramos se trata de “ImprimePrograma”,este método es el mas complejo, debido a que recibimos el programa entero y suprimimos la parte correspondiente a los subprogramas, obteniendo al final las constantes, variables y el programa principal.
   Por último, tenemos métodos add de cada arraylist y métodos tanto get como set para las variables

2. Clase Principal

   Respecto a la clase principal, hemos añadido en el constructor Parser la nueva clase creada (clase inicializar). Hemos establecido la funcionalidad necesaria para crear un fichero .html cuyo nombre será el fichero de entrada (argumento).
   Para finalizar, definimos la estructura del html, donde establecemos en el <head> los diferentes estilos de las palabras reservadas, constantes e identificadores.

3. Gramática.g4

   Al principio del fichero hemos declarado el @parser::members para utilizar la clase anteriormente creada “Inicializar”.

   Hemos introducido acciones semánticas utilizando atributos tanto sintetizados como heredados para ir comunicando a las ramas de árbol sintáctica los valores necesarios para su correcta creación. Entre las acciones semánticas utilizamos métodos definidos en la clase “Inicializar”

PARTE OPCIONAL

- ## Analizador Léxico
  Dado que los tokens a reconocer en esta parte son los mismos que en la obligatoria no hemos añadido nada a la parte léxica.

- ## Analizador Sintáctico
  Para añadir las sentencias de control del flujo de ejecución hemos ampliado la propia sentencia principal (sent) para introducirlas y a continuación todas las nuevas sentencias restantes.

  Al igual en la parte obligatoria, hemos tenido que solucionar la recursividad por la izquierda, en este caso, de la sentencia de las expresiones condicionales (expcond)

CASOS DE PRUEBA

1. **ESTRUCTURA PRINCIPAL DEL PROGRAMA**

   Con este primer caso de prueba queremos comprobar si reconoce correctamente la estructura básica del programa, concretamente las producciones ‘prg’ y ‘blq’

CORRECTO

```
PROGRAM program;
VAR
    n: integer;
BEGIN
    n:=1;
END.
```

INCORRECTO

```
PROGRAM program
VAR
    n: integer;
BEGIN

END
```

En el caso de uso incorrecto, como podemos comprobar en la consola, el compilador nos informa de todas las ausencias en el código del programa: el ‘;’ al final de identificador, falta de sentencias entre ‘BEGIN’ y ‘END’, y, por último, el ‘.’ del final

```
line 2:0 missing ';' at 'VAR'
line 6:0 mismatched input 'END' expecting {'IF', 'WHILE', 'REPEAT', 'FOR', IDENTIFIER}
line 7:0 missing '.' at '<EOF>'
```

2. ZONA DE DECLARACIONES

En este caso verificamos que se reconozcan todas las posibles declaraciones que hemos definido:

CORRECTO

```
PROGRAM program;
CONST
    pi=3.14;
    palabra='hola';
    numero=7;
VAR
    n: integer;
    r,s,t: real;
PROCEDURE Suma(s:integer; n:integer; m:integer);
    BEGIN
        s:=n+m;
    END;
FUNCTION SumaR(s:integer; n:integer; m:integer): integer;
    BEGIN
        s:=n+m;
    END;
BEGIN
    n:=1;
END.
```

INCORRECTO

```
PROGRAM program;
CONST
    pi=3.14;
    palabra:='hola';
    numero=7;
VAR
    n: integer;
    r,s,t: real;
PROCEDUR Suma(s:integer; n:integer, m:integer);
    BEGIN
        s:=n+m;
    END;
FUNCTION SumaR(s:integer; n:integer; m:integer);
    BEGIN
        s:=n+m;
    END;
BEGIN
    n:=1;
END.
```

En el caso de uso incorrecto, el compilador nos mostrara los 4 errores introducidos:

```
line 4:11 mismatched input ':=' expecting '='
line 9:9 mismatched input 'Suma' expecting ':'
line 9:25 no viable alternative at input 'n'
line 12:7 mismatched input ';' expecting {';', '('}
```

3. ZONA DE SENTENCIAS

En este caso vamos a comprobar que se reconocen todas las posibles sentencias que hemos definido en la parte sintáctica:

CORRECTO

```
PROGRAM program;
VAR
    n,n1,n2,n3: integer;
FUNCTION Suma(s:integer; n:integer; m:integer): integer;
    BEGIN
        s:=n+m;
    END;
BEGIN
    n := Suma(0,1,1);
    n1 := n * 2;
    n2 := 8 DIV n1;
    n3 := 8 MOD n1;
END.
```

INCORRECTO

```
PROGRAM program;
VAR
    n,n1,n2,n3: integer;
FUNCTION Suma(s:integer; n:integer; m:integer): integer;
    BEGIN
        s:=n+m;
    END;
BEGIN
    n := Suma(0,1,1;
    n := Suma(0,1;1;
    n1 := n +(2;
    n2 := *n;
    n3 = 8 MOD n1;
END.
```

Errores producidos:

```
line 9:19 missing ')' at ';'
line 10:17 missing ')' at ';'
line 10:18 no viable alternative at input '1'
line 9:15 no viable alternative at input '(2;'
line 10:10 extraneous input '*' expecting {'(', IDENTIFIER, NUMERIC_INTEGER_CONST, NUMERIC_REAL_CONST, STRING_CONST}
line 11:7 no viable alternative at input 'n3='
```

4. AMPLIACIÓN DE LA PARTE SINTÁCTICA

En este apartado se va a verificar las sentencias ampliadas en la parte sintáctica:

CORRECTO

IF-ELSE

```
PROGRAM prog;
VAR
    n,n1,n2: integer;
BEGIN
    n:=0;
    IF (n > 1) THEN
        BEGIN
            n:=1;
        END
    ELSE
        BEGIN
            n:=0;
        END
END.
```

FOR

```
PROGRAM prog;
VAR
    n,n1,n2: integer;
BEGIN
    n:=0;
    FOR i:=0 TO 10 DO
        BEGIN
            n:= 1 + 1;
        END
END.
```

WHILE

```
PROGRAM prog;
VAR
    n,n1,n2: integer;
BEGIN
   n:=0;
      WHILE (n<10) DO
          BEGIN
              n := n+1;
          END
END.
```

REPEAT

```
PROGRAM prog;
VAR
    n,n1,n2: integer;
BEGIN
    REPEAT
        BEGIN
            n:=n+1;
        END
    UNTIL (n>10);
END.
```

Para probar un caso mas elaborado hemos juntado sentencias dentro de otras, en este caso un bucle FOR con una sentencia IF ELSE para comprobar que se pueden combinar:

```
PROGRAM prog;
VAR
    n,n1,n2: integer;
BEGIN
    FOR i:=10 DOWNTO 0 DO
        BEGIN
            IF (TRUE) THEN
                BEGIN
                    n:=1 + 1;
                END
            ELSE
                BEGIN
                    n:= n -1;
                END
        END
END.
```

INCORRECTO

```
PROGRAM program;
VAR
    n,n1,n2,n3: integer;
BEGIN
    n := 0;
    IF (n>1) THEN
        BEGIN
            n:=1;
        END

    FOR (i:=0 TO 10) DO
        BEGIN
            n:= 1 + 1;
        END
END.
```

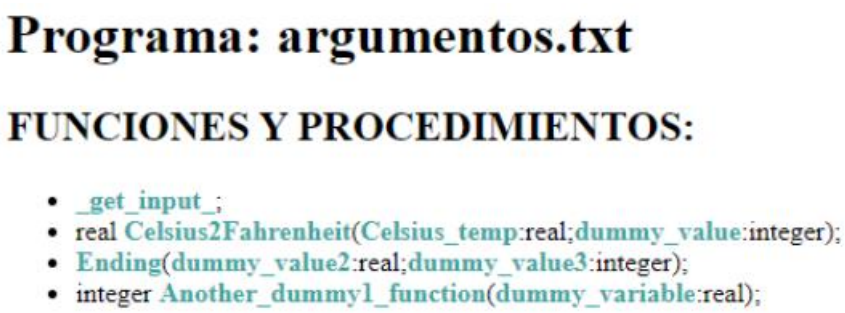
```
line 10:4 mismatched input 'FOR' expecting {'END', 'IF', 'WHILE', 'REPEAT', 'FOR', IDENTIFIER}
line 10:8 extraneous input '(' expecting IDENTIFIER
line 10:19 extraneous input ')' expecting 'DO'
```

-Todos los casos de prueba correctos muestran por pantalla que se ha ejecutado correctamente:

```
Process finished with exit code 0
```

VISUALIZACIÓN HTML

1. CABECERAS DE LOS SUBPROGRAMAS



2. CUERPO DE LOS SUBRPROGRAMAS

```
PROCEDURE _get_input_;
BEGIN
	Write('Enter Degrees C: ');
	Read(Celsius_temp);
	IF Celsius_temp>=0THEN
	BEGIN
		WriteLn('Positive input');
		IF Celsius_temp>=40.0THEN
		BEGIN
			WriteLn('Too hot ');
		END
		ELSE
		BEGIN
			WriteLn('Alright ');
		END
		WriteLn('Temperature');
	END
	ELSE
	BEGIN
		WriteLn('Negative input');
	END
	WriteLn('Sentence with "double quotes" inside');
END;
INICIO PAGINA Subprograma0
```

3. PROGRAMA PRINCIPAL

PROGRAMA PRINCIPAL

```
CONST
	ratio=1.8;
	r1=+123.456;
	r2=-00.69;
	r3=45.07000;
VAR
	Celsius_temp,Fahrenheit_temp:real;
BEGIN
	int1:=+123;
	int2:=-690;
	int3:=405;
	int4:=000078;
	int5:=-005;
	int6:=+0953;
	_get_input_;
	Fahrenheit_temp:=Celsius2Fahrenheit(Celsius_temp);
	Write('Fahrenheit is ');
	e1:=123E456;
	e2:=-64E-77;
	e3:=+045e16;
	e4:=003E+35;
	m1:=1.23E456;
	m2:=-000.64E-77;
	m3:=+045.0e16;
	m4:=0.03E+35;
	Write(ROUND(Fahrenheit_temp));
	FOR i:=1 TO 10 DO
	BEGIN
		Writeln(i);
	END
	Ending;
END.
```

PROGRAMA PRINCIPAL INICIO PAGINA

PROBLEMAS ENCONTRADOS

Lo primero, el problema mas destacable que nos hemos encontrado que no sabíamos como empezar a mostrar el árbol sintáctico entero, no sabíamos que clases crear, si realmente era necesario crearlas, por lo que empezamos haciendo mediante un String que se fuese concatenando todas las sentencias correspondientes al fichero de entrada.

El segundo problema ha sido el tiempo que hemos invertido desarrollando los algoritmos necesarios para la correcta representación en el .html, debido a el método que hemos utilizado para realizar la traducción dirigida por la sintaxis (concatenaciones de Strings) en algunas partes necesitábamos suprimir ciertas partes del código que contenía el String, por ejemplo para mostrar el programa principal (constantes, variables y el código principal) tuvimos que eliminar los subprogramas del programa principal.

Otro problema encontrado fue la realización de la indentación. Como iba por bloques de BEGIN – END si teníamos un bloque dentro de otro bloque, el segundo bloque debía estar más indentado que el primero, por lo que no sabíamos como solventar el problema al tener todo el programa en un String. Logramos que la solución fuese mas eficaz de lo que pensábamos que iba a ser, ya que tenemos un contador que se incrementa por cada BEGIN y se decrementa por cada END; este contador se multiplica por un número fijo (1.25cm) de indentacion para lograr una correcta tabulación.

OPINIÓN PERSONAL

Para la realización de la practica hemos tenido que aprender a utilizar atributos sintetizados y heredados cosa que nos ha facilitado a la hora de estudiar para el examen final.

Nos hubiese gustado/facilitado que se mostrara un ejemplo de cómo se vería el html final, por ejemplo, en las cabeceras de los subprogramas no nos queda del todo claro el formato a mostrar.