PROCESADORES DE LENGUAJES

Prácticas de laboratorio

Curso 2016/2017

CONSTRUCCIÓN DE UN ANALIZADOR DE UN LENGUAJE IMPERATIVO

Universidad de Alcalá
Dpto. Ciencias de la Computación
Asignatura 780018 2016-17

Índice

1. Introducción

Se desea construir un analizador (léxico, sintáctico y semántico) para un pequeño lenguaje imperativo cuyo único estructuras de control para la selección es el comando '*if*' y para la iteración el comando '*while*'. Las variables se declaran de forma explícita como un *entero* o un *booleano*, y la semántica del lenguaje siguen una fuerte disciplina con las expresiones.

Para el análisis de un lenguaje de programación, habitualmente, se divide en dos partes, la parte léxica que describe las unidades más pequeñas con significado, llamado tokens, y la parte de sintaxis que explica cómo se organizan los tokens en los programas.

Primera Parte

2. Analizador Léxico

Especificaciones Léxicas

La parte léxica reconoce identificadores, números, operadores, signos de puntuación, símbolos especiales y palabras reservadas.

EspLex 1. El conjunto de las palabras reservadas del lenguaje es: **`program`**, **`is`**, **`begin`**, **`end`**, **`var`**, **`integer`**, **`boolean`**, **`read`**, **`write`**, **`skip`**, **`while`**, **`do`**, **`if`**, **`then`**, **`else`**, **`and`**, **`or`**, **`true`**, **`false`** y **`not`**, las cuales generan el token correspondiente a cada una de ellas.

También, existen una serie de tokens en la gramática, que permiten las diferentes operaciones del lenguaje, y estos son:

EspLex 2. Operador de asignación: Símbolo '**:=**' .

EspLex 3. Operadores de relación: Símbolos '**<=**' , '**<**' , '**=**' , '**>**' , '**>=**' y '**<>**'.

EspLex 4. Operadores matemáticos: Símbolos '**+**' , '**-**' , '*****' y '**/**' .

EspLex 5. Signos de puntuación: Símbolos '**(**' , '**)**' , '**,**' , '**;**' y '**:**' .

EspLex 6. Identificadores: Siempre comienzan por una letra y a continuación pueden aparecer letras y números. Ejemplo: Ident01, a994.

No son sensibles a mayúsculas y minúsculas, lo que significa que “ident01” e “IDENT01” son el mismo identificador.

EspLex 7. Los valores de tipo entero (**integer**) están compuestos por una secuencia con, al menos, un dígito y, opcionalmente, al principio, pueden tener un símbolo (+ o -). Ejemplo: 4212, - 4212 ó +4212. Los valores de tipo booleano (**boolean**) aceptan únicamente *true* o *false*.

EspLex 8. Puede haber espacios y/o tabuladores en cualquier parte del fichero a analizar.

EspLex 9. Se considera un error léxicos la detección de un carácter no definido.

Requisitos Léxicos

ReqLex 1. Se deben presentar los errores detectados e indicar para cada uno el número de línea y columna donde se ha producido. Estos errores pueden mostrarse a medida que se detectan, o almacenarse en una lista y mostrarlos al final del análisis.

ReqLex 2. En la memoria justificativa se deberá incluir un Autómata Finito Determinista (AFD) por cada una de las expresiones regular definidas[1].

ReqLex 3. El empleo o no de estados léxicos deberá justificarse en la memoria justificativa.

ReqLex 4. Al analizar el archivo que contenga un programa en el lenguaje dado, creará una salida en pantalla con una lista de tokens que representan los componentes atómicos significativas del programa.

[1] Existen muchas herramientas que construyen AFD, ejemplos de ellas son: https://regexper.com/, http://ivanzuzak.info/noam/webapps/fsm2regex/, http://www.regular-expressions.info/, etc.

Ejemplo de entrada:

```
program switch is
     var sum,k : integer;
     var switch : boolean;
begin
     switch := true; sum := 0; k := 1;
     while k<10 do
          switch := not(switch);
          if switch then
               sum := sum+k
          end if;
          k := k+1
     end while;
     write sum
end
```

Ejemplo de Salida del analizador léxico:

```
Análisis Léxico completado

[program,ide(switch),is,var,ide(sum),coma,ide(k),do
s_puntos,integer,punto_coma,var,ide(switch),dos_pun
tos,boolean,punto_coma,begin,ide(switch),asign,true
,punto_coma,ide(sum),asign,num(0),punto_coma,ide(k)
,asign,num(1),punto_coma,while,ide(k),comp,num(10),
do,ide(switch),asign,not,paren_izq,ide(switch),pare
n_der,punto_coma,if,ide(switch),then,ide(sum),asign
,ide(sum),mas,ide(k),end,if,punto_coma,ide(k),asign
,ide(k),mas,num(1),end,while,punto_coma,write,ide(s
um),end]
```

3. Analizador Sintáctico

En esta fase se analiza la estructura de las expresiones utilizando como base la gramática propuesta por el alumno. Con este análisis ya se puede determinar si una estructura está bien o mal formada. El análisis que se realiza es jerárquico, es decir, se lleva a cabo a partir de árboles de derivación que se obtienen de la propia gramática.

Especificaciones Sintácticas

El lenguaje posee las siguientes especificaciones sintácticas:

EspSimt 1. El identificador para el nombre del programa no puede ser declarado en otro lugar en el programa más que al comienzo.

EspSimt 2. Un identificador de variable no puede ser declarado más de una vez.

EspSimt 3. La conjunción o disyunción de expresiones lógicas utiliza los operadores *and* y *or* respectivamente.

EspSimt 4. Una expresión lógica negada, utiliza el operador *not* antes de la expresión lógica entre paréntesis.

EspSimt 5. La igualdad entre dos expresiones utiliza el operador "=". Las dos expresiones comparadas deberán de ser del mismo tipo.

EspSimt 6. La comparación entre dos expresiones de tipo entero utiliza los operadores de relación.

EspSimt 7. Las palabras reservadas no se pueden utilizar como nombre para un identificador.

EspSimt 8. La precedencia de los operadores es, de menor a mayor precedencia (todos los operadores asocian por la derecha) la siguiente:

1. or
2. and
3. not
4. Comparaciones
5. + y -
6. * y /

Requisitos Sintácticos

ReqSimt 1.- El analizador solamente ha de aceptar ficheros con extensión ".prog".

ReqSimt 2.- La definición de la gramática, para el lenguaje, deberá estar libre de ambigüedades. El alumno se encontrará con el problema del '*else ambiguo*'. Deberá indicar en la memoria justificativa cómo se ha resuelto este problema en las definiciones de la gramática. Ej:

```
orden ::=  if  expr_booleana  then orden
         | if  expr_booleana  then orden else orden
```

ReqSimt 3.- El analizador también debe ser capaz de detectar los errores sintácticos, tales como:

a. Errores en la declaración de variables. Por ejemplo:
   var sum k **: integer** *(error en el nombre del identificador)*
b. Expresiones mal formadas. Por ejemplo:
   4 **+ +** 5**;** ó id **= (**6 **+** 6**;**
c. etc.

ReqSimt 4.- Cuando detecte un error, el analizador propuesto deberá mostrar un mensaje que indique la línea donde se ha producido.

Al final del análisis, y en caso de haber encontrado errores, deberá mostrar un informe con el número total de errores que se han producido, tanto léxicos como sintácticos. Por el contrario, si el analizador hubiera terminado sin encontrar ningún error deberá mostrar un mensaje indicando que ha finalizado sin errores.

ReqSimt 5.- El analizador debe ser capaz de recuperarse cuando encuentre un error recuperable y continuar analizando el resto del programa.

ReqSimt 6.- Al analizar el archivo que contenga un programa en el lenguaje dado, analizará las producciones creando un árbol de sintaxis abstracta (AST) que represente la estructura del programa que mostrara por pantalla, similar al ejemplo dado.

Ejemplo de entrada [2] :

```
program switch is
     var sum,k : integer;
     var switch : boolean;
begin
     switch := true; sum := 0; k := 1;
     while k<10 do
          switch := not(switch);
          if switch then sum := sum+k end if;
          k := k+1
     end while;
     write sum
end
```

Ejemplo de Salida :

```
Análisis Sintáctico completado
AST resultado del analisis
program([decl(integer,[sum,k]),decl(boolean,[switch])],
      [asign(switch,true),asign(sum,num(0)),asign(k,num(1)),
            while(exp(menor,ide(k),num(10)),
              [asign(switch,notbooleno(ide(switch))),
                        if(ide(switch),
                         [asign(sum,exp(mas,ide(sum),ide(k)))],skip),
                        asign(k,exp(mas,ide(k),num(1)))]),
            write(ide(sum))]

        )
```

Nota: La lista de símbolos y el árbol de sintaxis abstracta se han formateado para que sean más fáciles de leer.

[2] Observe que el programa, si bien cumple la sintaxis de una gramática libre, es sintácticamente incorrecto. Revisar las limitaciones de contexto.

Segunda parte

4. Analizador Semántico

Tal y como se realiza en un compilador habitualmente, en la fase de análisis semántico se debe comprobar que el programa fuente se ajusta a todas las especificaciones del lenguaje de programación fuente que no hayan podido ser comprobadas en la fase de análisis sintáctico.

Esto aplica particularmente a las comprobaciones de tipos de las expresiones, pero también a otras como la comprobación del ámbito, la privacidad de los atributos y métodos, etc. Las expresiones que combinan una o más subexpresiones se deberán ajustar a lo indicado anteriormente.

Por tanto, el analizador semántico deberá además de asegurarse de que la semántica del archivo de entrada es correcta, comprobar lo siguiente (especificaciones semánticas):

EspSemt 1.- Todos los identificadores de variable utilizados deberán haber sido declarados previamente y una sola vez.

EspSemt 2.- En una sentencia condicional (`if`), la expresión debe ser de tipo booleano, mientras que las sentencias que forman parte de la sección `then` y de la sección `else` no deben tener errores de concordancia de tipo.

EspSemt 3.- En una sentencia bucle (`while`), la expresión debe ser de tipo booleano, mientras que las sentencias que forman el cuerpo del bucle no deben tener error de concordancia de tipo.

Limitaciones de contexto (comprobaciones de tipos)

Cada programa en el lenguaje dado se puede considerar como una cadena de tokens, aunque no todas las cadenas de tokens son un programa legal. La especificación de una gramática libre contexto (en BNF) restringe el conjunto de posibles cadenas que se pueden obtener mediante una derivación del no terminal *program.*

La notación BNF puede definir sólo aquellos aspectos de la sintaxis que son libres de contexto, ya que cada producción se puede aplicar independientemente de los símbolos de los alrededores. Por tanto, el programa siguiente pasa los requisitos descritos por la gramática.

```
program illegal is
     var   a : boolean;
begin
     a := 5;
end
```

El error en el programa "*ilegal*" implica una violación del tipo definido mediante una declaración ya que la variable "a" ha sido declarada de tipo booleano, pero en el cuerpo del programa, se hace un intento de asignarle un valor entero.

La clasificación de un error de este tipo implica una cierta controversia. Algunos autores dicen que tal error pertenece a la semántica estática de un lenguaje, ya que implica el significado de los símbolos. Se argumenta que los errores estáticos pertenecen a la sintaxis, no a la semántica, de un lenguaje.

Luego nuestro analizador semántico cumplirá:

EspSemt 4.- En una sentencia de asignación (:=), el tipo de la variable en la parte izquierda de la asignación y el tipo de la expresión en la parte derecha debe ser el mismo.

EspSemt 5.- Un identificador que aparece como un elemento entero debe ser una variable entera. Una expresión suma, resta, multiplicación o división que se aplique será de tipo entero.

EspSemt 6.- Un identificador que aparece como un elemento booleano debe ser una variable booleana.

EspSemt 7.- Un identificador o expresión que aparece en un comando *`read`* debe ser una variable entera.

EspSemt 8.- Un identificador o expresión que aparece en un comando *`write`* puede ser una variable entera o booleana.

Deberá realizar una comprobación de tipos con objeto de cumplir las restricciones de compatibilidad de tipos impuestas.

Gramática de atributos

Nuestro lenguaje posee una estructura plana en el sentido de que sólo hay un bloque en un programa. Como consecuencia, todas las declaraciones pertenecen a una única secuencia de declaración en el nivel principal del programa. Cabe señalar que existe una pequeña excepción a nuestro único conjunto de declaraciones:

EspSemt 9.- El nombre del programa en sí no es parte de la estructura de bloque. Es una decisión de diseño ningún objeto puede tener el mismo nombre que el programa. El nombre del programa es único y no se pueden utilizar en otras partes del programa. No se deben declarar en el programa fuente dos identificadores de variables con el mismo nombre (incluida la variable que se declaran en la cabecera del programa).

EspSemt 10.- Todas las variables son de tipo *entero* o *booleano*, y se introduce un tipo de *pseudo* para el identificador de nombre del programa, y un valor por defecto sin definir para representar la ausencia de un tipo. Dado que todas las declaraciones en nuestro lenguaje son globales, únicamente hay una única tabla de símbolos que se transmite a la secuencia de comandos.

La tabla de símbolos

Dado que los nombres de variables y tipos no pueden aparecer mágicamente en la tabla de símbolos, toda esta información debe ser sintetizada en el árbol utilizando atributos tales como nombre, tipo, y la lista de variables.

La siguiente tabla contiene una lista de los atributos y tipos de valores asociados.

Atributo	Tipos de valor
Tipo	integer, boolean, undefined, pseudo[3]
Nombre[4]	Cadena de letras o dígitos
Lista Variables [5]	Secuencia nombres de valores
Tabla de símbolos	Conjunto de pares de la forma [Nombre, Tipo]

Estos atributos se sintetizan a partir de la *secuencia de declaraciones* y se heredan en la *secuencia de comandos* de un programa. El valor del atributo es transferido a nivel de bloque.

La tabla de símbolos contiene, al menos, un conjunto de pares de cada asociación de un nombre con un tipo.

EspSemt 11.- Deberá construir una estructura para el almacenamiento de la tabla de símbolos similar a la descrita.

3 Hemos añadido el valor de tipo pseudo del programa para el tipo de atributo para que se identifica de forma exclusiva el nombre del programa.

4 Un valor de nombre es una cadena de uno o más letras o dígitos.

5 Un valor Var-lista es una secuencia de valores de nombre

EspSemt 12.- En la memoria deberá presentar una tabla, tabla 1, (Atributos asociados con símbolos no terminales) con tres columnas, en la que se establecen los atributos si son sintetizados o heredados y su procedencia (Tipo, Nombre, Lista Variables o Tabla de Símbolos).

Tabla 1 Ejemplo de tabla de Atributos

No terminal	Atributo sintetizado	Atributo heredado
<block>	-	*Tabla de Símbolos*
<declarationsequence>	*Tabla de Símbolos*	-
<declaration>	*Tabla de Símbolos*	-
….		
<term>	-	Tabla de Símbolos, Tipo

Como cualquier programador sabe, incluso después de haber depurado todos los errores de estas fases, en su programa todavía pueden existir defectos. El fallo puede ser que el programa se ejecute hasta su finalización, pero su comportamiento no esté de acuerdo con la especificación del problema que el programa está tratando de resolver.

Una segunda posibilidad es que el programa no finaliza normalmente porque se ha tratado de llevar a cabo una operación que no se puede ejecutar por el sistema en tiempo de ejecución. Llamamos a estos errores semánticos en tiempo de ejecución.

Los errores semánticos que pueden cometerse durante la ejecución de un programa son:

1. Se hace un intento de dividir por cero.
2. Se accede a una variable que no se ha inicializado. Por ejemplo, un comando de *write* se ejecuta cuando la variable no ha sido inicializada.
3. Un comando de iteración (*while*) no termina.

EspSemt 13.- En la medida de lo posible, deberá realizar una comprobación de este tipo de errores (dinámicos).

5. Normas de la Práctica

La práctica está propuesta para ser realizada de forma individual.

La práctica consiste en el diseño e implementación de un procesador de lenguajes, que realice el Análisis Léxico, Sintáctico y Semántico (incluyendo la Tabla de Símbolos, Gestor de Errores y Árbol de Análisis Sintáctico), para el lenguaje de programación propuesto. El trabajo se abordará de una manera incremental durante el curso.

Funcionamiento de la Solución

El analizador deberá leer el programa fuente de un archivo de texto y generar información relativa a la lista de tokens, análisis sintáctico, tabla de símbolos, errores y árbol de análisis sintáctico.

El funcionamiento tiene que ser obligatoriamente el siguiente:

- Entrada (Fichero fuente): El analizador propuesto, como solución, ha de recibir (a través de la línea de comandos) un archivo de texto cuyo contenido es el programa que se desea analizar.

- Salida: Para facilitar las tareas de depuración y de corrección de la práctica, es necesario mostrar los resultados de las distintas partes del proceso. Por ello, el procesador de la práctica deberá generar obligatoriamente a siguiente información *(ver Ejemplos del Anexo I)*:

    1. Lista de tokens que representan todos los componentes el programa.

    2. Listado de la Tabla de Símbolos: Volcado completo y legible con toda la información de la Tabla de Símbolos.

    3. Listado del análisis sintáctico: Listado de los números de las reglas utilizadas para realizar el Análisis Sintáctico de la entrada.

    4. Listado de errores: Si el programa fuente que se está analizando es incorrecto, deberá proporcionarse un listado en formato libre con los errores detectados. Para cada error habrá que indicar al menos el número de la línea donde se ha detectado, el tipo de error (léxico, sintáctico o semántico) y un mensaje claro que explique el error.

    5. Árbol de Análisis Sintáctico (AST): Si el programa fuente que se está analizando no tuviese errores irrecuperables, deberá presentar el AST resultado del análisis de programa fuente.

6. Evaluación de la práctica

Elementos a entregar por el alumno

Elementos que se esperan en el envío de la práctica:

1. Ficheros fuentes de la implementación. (**`jflex`** y **`cup`**)
2. Un fichero de texto con la gramática en la que los elementos no-terminales estén en mayúsculas y los terminales en minúsculas.
3. Al menos, **10** casos de prueba, **3** correctos y **7** erróneos (formato **`txt`**).
4. Memoria justificativa, de la práctica, con los elementos solicitados (formato **`pdf`**).

   La **memoria justificativa** (extensión aproximada de 20-30 páginas, sin contar anexos) incluirá:

   - Una descripción del diseño de la solución correspondiente, así como cualquier otro aspecto o característica que se desee hacer notar por su interés, sin incluir los listados fuente.
   - Diseño del Analizador Léxico: tokens, gramática, autómata, acciones y errores.
   - Diseño del Analizador Sintáctico: gramática, demostración de que la gramática es adecuada y las tablas o procedimientos de dicho Analizador.
   - Diseño del Analizador Semántico: Traducción Dirigida por la Sintaxis con las acciones semánticas.
   - Diseño de la Tabla de Símbolos: Descripción de su estructura y organización.
   - Diseño del AST: Descripción de su estructura y organización.
   - Diez casos de prueba. Deberá incluirse en la memoria un anexo con los diez casos listados. Tres de ellos serán correctos y los otros erróneos, de tal manera que permitan observar el comportamiento de la solución dada.

   Para uno de los ejemplos correctos, se incluirá el listado de tokens, la salida de las reglas aplicadas por el analizador sintáctico, el árbol de análisis sintáctico y el volcado de la Tabla de Símbolos.

   Para los ejemplos erróneos se incluirá el mensaje o mensajes de error obtenidos.

El software entregado debe estar libre de virus, en caso de encontrar alguna práctica con virus se considerará como no presentada.

Es imprescindible que la solución entregada cumpla con todos los requisitos de ficheros y formatos indicados para poder superar la Práctica.

Calificación

La evaluación de la práctica se divide en cuatro niveles:

1. Funcional - El analizador presentado hace lo que se pide (cumple los requisitos y las especificaciones).

2. Estructura de la gramática - La gramática no tiene problemas de diseño (desplazamiento/reducción o reducción/reducción) o se han solucionado con las precedencias.

3. Arquitectura de la solución - Se hace utilizando y explicado, en la memoria justificativa, las estructuras de JAVA usadas (map, hashmap, list, arraylist…) y la programación es acorde al nivel del curso del alumno.

4. Aspectos Formales - Se explica toda la práctica adecuadamente en la memoria justificativa y se demuestra su funcionamiento con los casos de prueba requeridos.

La calificación se determinará en función del nivel de cumplimiento de los requisitos y especificaciones:

Análisis Léxico y Sintáctico

1. **Aprobado (5-5,9]**: Implementación de los requisitos y las especificaciones léxicas y sintácticas (completas) y, además, sobre escrito el tratamiento de errores sintácticos.

Análisis Semántico

- Realiza la comprobación de tipos.
- Utiliza la gramática de atributos.
- Comprueba errores dinámicos

2. **Aprobado (6-6,9]**: A lo anterior se le añade, la implementación de los requisitos y especificaciones semánticas del 1 a 8.

3. **Notable (7-8,5]**: A lo anterior se le añade, la implementación de las especificaciones semánticas del 9 y siguientes.

Se otorgará **un 1,5 extra** en función de la calidad de la memoria justificativa y el trabajo entregado (aportaciones del alumno).

Se podrá pedir a los alumnos que defiendan la práctica realizada presencialmente, si así lo considera el profesor.

Si se detecta alguna práctica copiada, los alumnos afectados tendrán la práctica y la asignatura suspensa en la presente convocatoria

Además, se procederá a informar al Directo de la Escuela Politécnica con objeto de incoación de expediente y, si procede, aplicación del Real Decreto 1791/2010 Artículo 13.apt. d

7. ANEXO I - EJEMPLOS

Ejemplo 1

```
>>> Práctica 2016/2017 <<<
Nombre del programa: programa1.prog
program prog1 is
      var x,y: integer;
      var b,c: boolean;
begin
      read x; read y; write x+y;
      b := x < y;
      if x = y
            then c := x <= y
      else c := x > y end if;
      while c do x := x + 1 end while;
      b := b and (b or c)
end

Analisis Lexico completado
[program,ide(prog1),is,var,ide(x),coma,ide(y),dos_puntos,integer,punto_co
ma,var,ide(b),coma,ide(c),dos_puntos,boolean,punto_coma,begin,read,ide(x)
,punto_coma,read,ide(y),punto_coma,write,ide(x),mas,ide(y),punto_coma,ide
(b),asign,ide(x),menos,ide(y),punto_coma,if,ide(x),igual,ide(y),then,ide(
c),asign,ide(x),menor_igual,ide(y),else,ide(c),asign,ide(x),mayor_igual,i
de(y),end,if,punto_coma,while,ide(c),do,ide(x),asign,ide(x),mas,Num(1),en
d,while,punto_coma,ide(b),asign,ide(b),and,paren_izq,ide(b),or,ide(c),par
en_der,end]

Analisis Sintactico completado
AST resultado del analisis
 [program,ide(prog1),is,
      var,ide(x),coma,ide(y),dos_puntos,integer,punto_coma,
      var,ide(b),coma,ide(c),dos_puntos,boolean,punto_coma,
 begin,
      read,ide(x),punto_coma,read,ide(y),punto_coma,
      write,ide(x),mas,ide(y),punto_coma,
      ide(b),asign,ide(x),menos,ide(y),punto_coma,
      if,ide(x),igual,ide(y),
            then,ide(c),asign,ide(x),menor_igual,ide(y),
            else,ide(c),asign,ide(x),mayor_igual,ide(y),
      end,if,punto_coma,
      while,ide(c),do,
            ide(x),asign,ide(x),mas,num(1),
      end,while,punto_coma,
      ide(b),asign,ide(b),and,paren_izq,ide(b),or,ide(c),paren_der,
end]
```

Ejemplo 2

```
>>> Práctica 2016/2017 <<<
Nombre del programa: programa2.prog
program prog2 is
	var x,y,b: integer;
	var b,c: boolean;
begin
	read x; read c; write x+a;
	b := x < c;
	if x = y
	   then c := x <= y
	   else y := x > y
	end if;
	while c > b do x := x + 1 end while;
	b := b and (y or z)
end

Analisis Lexico completado
[program,ide(prog2),is,var,ide(x),coma,ide(y),coma,ide(b),dos_puntos,integer,
punto_coma,var,ide(b),coma,ide(c),dos_puntos,boolean,punto_coma,begin,read,id
e(x),punto_coma,read,ide(c),punto_coma,write,ide(x),mas,ide(a),punto_coma,ide
(b),asign,ide(x),menor,ide(c),punto_coma,if,ide(x),igual,ide(y),then,ide(c),a
sign,ide(x),menor_igual,ide(y),else,ide(y),asign,ide(x),mayor_igual,ide(y),en
d,if,punto_coma,while,ide(c),mayor_igual,ide(b),do,ide(x),asign,ide(x),mas,Nu
m(1),end,while,punto_coma,ide(b),asign,ide(b),and,paren_izq,ide(y),or,ide(z),
paren_der,end]

Analisis Sintactico completado
AST resultado del analisis
[program,ide(prog2),is,
	var,ide(x),coma,ide(y),coma,ide(b),dos_puntos,integer,punto_coma,
	var, ide(b), coma,ide(c), dos_puntos, boolean, punto_coma,
	ERROR: Declaracion Duplicada de un identificador,
begin,
	read, ide(x),punto_coma,
	read, ide(c),
	ERROR: Variable Entera para read, punto_coma,
	write, ide(x),mas, ide(a),
	ERROR: Variable no declarada, punto_coma,
	ide(b), asign, ide(x), menor, ide(c),
	ERROR: Variable Entera esperada,
	ERROR: Expresion Entera esperada,
	punto_coma,
	if, ide(x),igual, ide(y),
		then, ide(c),asign, ide(x),menor_igual, ide(y),
		else, ide(y),asign, ide(x),mayor_igual, ide(y),
		ERROR: Expresion Entera esperada,
	end, if, punto_coma,
	while,ide(c),
	ERROR: Variable Entera esperada, mayor_igual, ide(b),do,
		ide(x), asign,ide(x), mas,num(1), end, while, punto_coma,
	ide(b), asign,ide(b),
	ERROR: Variable Booleana esperada, and, paren_izq, ide(y),
	ERROR: Variable Booleana esperada, or, ide(z),
	ERROR: Variable no declarada, paren_der,
	ERROR: Expresion Entera esperada,
end]
```

Ejemplo 3

Otro ejemplo de salida del analizador

```
>>> Práctica 2016/2017 <<<
Nombre del programa: programa3.prog
program frombinary is
      var sum, n : integer;
      var bol : boolean;
begin
      bol := true;
      sum := 10; read n;
      while  ( n < 2 ) do
            sum := 2*3+n; read n
      end while;
      write sum;skip
end

** Comienza Parser **
Lexema: frombinary
      Regla 18  => IDENTIFICADOR  ::=  ident
Lexema: sum
      Regla 18  => IDENTIFICADOR  ::=  ident
Lexema: n
      Regla 18  => IDENTIFICADOR  ::=  ident
      Regla 7a  => LIST_VARIABLES  ::=  IDENTIFICADOR
      Regla 7b  => LIST_VARIABLES  ::=  IDENTIFICADOR coma  LIST_VARIABLES
      Regla 6a  => TIPO  ::= tk_integer

...

BLOQUES
      Regla 3 => BLOQUE ::= SEQ_DECLARACIONES tk_begin SEQ_COMANDOS tk_end
Lexema: 'program'
      Regla 2  => PROGRAMA  ::=  tk_program IDENTIFICADOR tk_is  BLOQUE

** Finaliza Parser **

***************
 Report Lex
***************
[program,Ide(frombinary),is,var,Ide(sum),Coma,Ide(n),Dos_puntos,integer,Punt
o_coma,var,Ide(bol),Dos_puntos,boolean,Punto_coma,begin,Ide(bol),Asign,true,
Punto_coma,Ide(sum),Asign,Num(10),Punto_coma,read,Ide(n),Punto_coma,while,pa
ren_izq,Ide(n),Relac,Num(2),paren_der,do,Ide(sum),Asign,Num(2),Operador_fuer
te(*),Num(3),Operador_debil(+),Ide(n),Punto_coma,read,Ide(n),end,while,Punto
_coma,write,Ide(sum),Punto_coma,skip,end,]
***************
Tabla de simbolos
***************
*** Tabla de Simbolos ***
  * Palabras Reservadas *
Lexema : 'read' ; Tipo : 'palabra reservada'; Usado en lineas : 6,8,
Lexema : 'var' ; Tipo : 'palabra reservada'; Usado en lineas : 2,3,
Lexema : 'is' ; Tipo : 'palabra reservada'; Usado en lineas : 1,
```

Ejemplo de salida del analizador (cont)

```
Lexema : 'skip'   ; Tipo : 'palabra reservada'; Usado en lineas : 10,
Lexema : 'program'; Tipo : 'palabra reservada'; Usado en lineas : 1,
Lexema : 'integer'; Tipo : 'palabra reservada'; Usado en lineas : 2,
Lexema : 'do'     ; Tipo : 'palabra reservada'; Usado en lineas : 7,
Lexema : 'while'  ; Tipo : 'palabra reservada'; Usado en lineas : 7,9,
Lexema : 'boolean'; Tipo : 'palabra reservada'; Usado en lineas : 3,
Lexema : 'true'   ; Tipo : 'palabra reservada'; Usado en lineas : 5,
Lexema : 'end'    ; Tipo : 'palabra reservada'; Usado en lineas : 9,11,
Lexema : 'begin'  ; Tipo : 'palabra reservada'; Usado en lineas : 4,
Lexema : 'write'  ; Tipo : 'palabra reservada'; Usado en lineas : 10,
  * Identificadores *
Lexema : 'frombinary' ; Tipo : 'pseudo'; Declarado en linea : '1'
Lexema : 'sum' ; Tipo : 'entero'; Declarado en linea : '2'';
                                        Usado en lineas : 6,8,10,
Lexema : 'n' ; Tipo : 'entero'; Declarado en linea : '2'';
                                        Usado en lineas : 6,7,8,
Lexema : 'bol' ; Tipo : 'booleano'; Declarado en linea : '3'';
                                        Usado en lineas : 5,

*********************************
AST - Arbol de Analisis Sintactico
*********************************
...
|   ├Bloque definiciones
|   |   ┌Asignacion a: bol BOOLEANO, val= true
|   └   └Asignacion a: sum NUMERO, val= 10
|
|   ┌Bloque comandos
|   |   ┌Read: IDENTIFICADOR, nombre= n
|   |   ├While
|   |   |   ┌**Prueba While**
|   |   |   |   ┌** Operacion **
|   |   |   |   ├'<'
|   |   |   |   ├**Expr Izquierda **
|   |   |   |   |   -IDENTIFICADOR, nombre= n
|   |   |   |   └**Expr Derecha **
|   |   |   |       -NUMERO, val= 2
|   |   |   ├**Cuerpo While**
|   |   |   |   ┌Asignacion a: sum
|   |   |   |   ├  ┌** Operacion **
|   |   |   |   |  ├ '+'
|   |   |   |   |  ├**Expr Izquierda **
|   |   |   |   |  |    ┌** Operacion **
|   |   |   |   |  |    ├'*'
|   |   |   |   |  |    ├**Expr Izquierda **
|   |   |   |   |  |    |   -NUMERO, val= 2
|   |   |   |   |  |    └└**Expr Derecha **
|   |   |   |   |  |        -NUMERO, val= 3
|   |   |   |   |  └ └**Expr Derecha **
|   |   |   |   └       -IDENTIFICADOR, nombre= n
|   |   ├Read: IDENTIFICADOR, nombre= n
|   |   ├Write: IDENTIFICADOR, nombre= sum
|   └   └Skip
...
```

8. ANEXO II GUÍA PARA LA REDACCIÓN DE LAS PRÁCTICAS

Título de la práctica

Referencia a autor (nombre y apellidos, dni, e-mail, etc.)
(fecha: día mes año)

1. INTRODUCCION

Este documento describe las directrices generales sobre la preparación de informes para trabajos de laboratorio. La información suplementaria concreta a ciertas actividades puede ser dada por el responsable del laboratorio.

2. LONGITUD

El posible límite de páginas se basa en un tamaño de letra mínimo de entre 10 y 12 puntos (9 y 11 puntos para los títulos, texto de la tabla, el resumen y las referencias), pero siempre uniforme, y un ancho margen total de al menos 35 mm.

3. PLAGIOS

En general, TODOS los materiales, es decir, todo el texto, todas las cifras y todos los diagramas, debe ser original. Las figuras o diagramas de otras fuentes no se deben cortar y pegar en el informe, a menos que sean esenciales, por ejemplo, imágenes astronómicas, mapas, etc. En este caso, el elemento debería incluirse una referencia clara en el título a la fuente.

4. ESTRUCTURA

Un informe típico contiene las siguientes secciones. Tenga en cuenta que algunas actividades pueden requerir elementos adicionales según lo especificado por el responsable del laboratorio:

Resumen: Objetivos y resultados principales (solamente texto, no hay referencias o diagramas). Un resumen del trabajo realizado, la forma en que se realizó, cuál fue el resultado y cuáles son las principales conclusiones obtenidas. Incluir resultados numéricos y comparar con datos de la literatura en su caso. Hasta un máximo de 200 palabras**.**

1. **Introducción:** Uno o dos párrafos sobre los objetivos específicos de su trabajo; diga lo que se propuso hacer, lo que logró y por qué es importante.

2. **Métodos:** En esta sección se incluye tanto la teoría relevante y detalles experimentales en subsecciones separadas, por ejemplo, 2.1 Teoría, 2.2 Experimento, se puede incluir en función del carácter del trabajo.

   Bajo el título teoría no se reproducen grandes trozos de texto, ecuaciones o código que se pueden encontrar en otras fuentes y se hace referencia. Para todos los trabajos, una breve descripción de la metodología que debe ser aplicada. Para los trabajos experimentales esto puede incluir diagramas de puesta a punto, en su caso.

   Describir las características esenciales de cómo se hicieron las pruebas y lo que se probó. No escriba en orden cronológico a menos que realmente es la forma más lógica para presentar la metodología. Escribir en el tiempo pasado.

3. **Resultados y análisis:**

   Resultados: La sección principal del informe con los datos obtenidos en la forma adecuada. Utilice el tiempo presente, por ejemplo, 'la tabla. 1 muestra una comparativa de ... '. En las figuras deben estar marcados los ejes con las unidades. Los datos deben presentarse con barras de error.

Análisis: Para informes breves, un breve análisis sobre la interpretación de los resultados y los casos de prueba puede ser incluido junto con cada resultado. Comparación de los resultados a valores esperados o de la literatura, en su caso. Decir si los resultados se ajustan a lo esperado o a la teoría y dar un argumento razonado para explicar sus observaciones.

Para informes más largos una subsección separada se puede discutir la interpretación de los resultados y posibilidades para trabajos o mejoras futuras podría ser apropiado.

4. **Conclusiones:** Breve (de 1 a 3 párrafos) resumen de los principales resultados y las implicaciones del trabajo.

5. **Referencias**: Utilice un estilo de referencia constante (ya sea numérico, consulte la emulación del documento de estilo, o alfabético). Los títulos pueden ser incluidos, pero de nuevo entre coherentes, o bien no hay títulos o todos los títulos. Para el estilo numérico, las referencias deben ser numeradas en el orden en que aparecen en el texto. Compruebe que cada referencia contiene todos los detalles bibliográficos de la fuente (nombres de los autores, la revista, volumen, número de página, el año).

6. **Apéndices:** El apéndice debe incluir el análisis de errores y otra información de apoyo, por ejemplo, código de ordenador (no formateado), derivación de una ecuación, etc. No debería ser necesario tener que consultar el Anexo al leer el informe principal. Todos los resultados fundamentales, diagramas, etc., deben ser incluidos en el informe principal.

Para informes más largos, por ejemplo, en los que se ha realizado más de un experimento o proyecto distinto, el informe puede ser mejor estructurado de la siguiente manera:

Resumen
1. Introducción.
2. Título del Experimento / Proyecto A.
2.1 Métodos. 2.2 Resultados / análisis.
3. Título del Experimento / Proyecto B.
3.1 Métodos. 2.2 Resultados / análisis.
4. Título del Experimento / Proyecto c. etc. ...
Conclusiones.
Referencias.

El informe debe ser fácil de seguir con el material que se presenta en un orden lógico. El lector no debe tener que volver la página constantemente entre los métodos y resultados o resultados y anexos.

5. FORMATEO Y PUNTOS DE ESTILO GENERAL

Formato: Los informes pueden ser preparados en Word, Latex [1] o cualquier otro paquete de procesamiento de textos.

Estilo:

1 En la escritura científica es aceptable el uso de "nosotros" en lugar de "yo", incluso si trabajó es individual.

2 Todas las figuras deben ser referenciados en el texto, y deben estar numeradas en el orden en que aparecen en el texto.

3 Todo el texto en las figuras, por ejemplo, etiquetas de los ejes, etc., no deben ser más pequeño que el tamaño de la fuente principal de texto.

4 Todas las figuras deben tener un título. Las figuras deben ser generalmente comprensible, por ejemplo, "Experimento" no es aceptable, "Diseño óptico utilizado para medir la distancia entre la franja en un experimento de doble rendija" es mejor.

5 Los gráficos no deberían tener un fondo sombreado o líneas horizontales (Fig. 1 es malo, Fig. 2 es mejor).

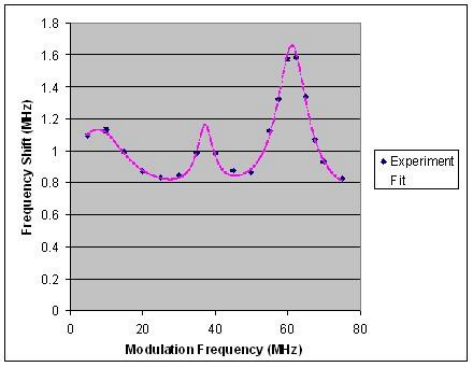

Fig. 1 This graph has the following faults: (i) Grey background; (ii) Horizontal lines; (iii) Box; (iv) Poor choice of *y* axis scale making detail of the plot compressed; (v) No error bars; (vi) Axes font lables too small; and (vii) Legend should be given in caption.

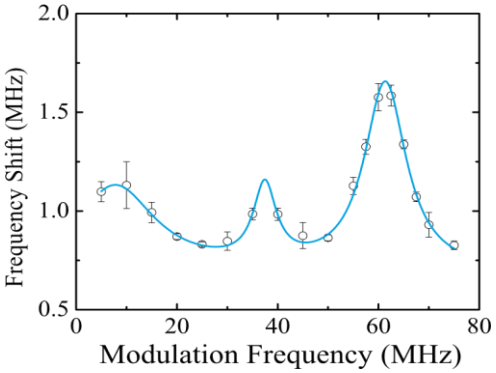

Fig. 2 Plot of the frequency shift of the atomic resonance as a function of the electric field modulation frequency. The data correspond to an average of 10 measurements with the standard error indicated by the error bar. The solid line is a fit to three Lorenztian resonances.

6 Tenga en cuenta que todos los parámetros están en cursiva, por ejemplo, el impulso, p.

7 Las etiquetas en las ecuaciones (por ejemplo, sen, cos, ln, e, etc.) y los subíndices de etiquetas deben estar en letra normal, por ejemplo, $\mathrm{sen}\omega t$, $\mathrm{e}i\omega t$, y el momento pc.

8 Los exponentes (ya sea en el texto o en los gráficos) deben escribirse como 10^{-8} y no 1e-8.

Las ecuaciones aparecen como si fueran parte de una oración o el texto delante de ellos termina con una coma, y la ecuación termina con punto (o coma), por ejemplo, 'La fuerza, *F*, sobre la partícula' se puede escribir como,

$$F = \frac{dp}{dt}\ ; \quad (1)$$

Donde *p* es el impulso y *t* es tiempo. En las ecuaciones en la línea de texto deben escribirse en una sola línea de modo Eq. (1) se transforma en, $F = \mathrm{d}p/\mathrm{d}t$. Insertar un espacio entre un valor y la unidad. Las unidades deben en letra normal, por ejemplo, el impulso de la partícula es $p = 3 \times 10^{-16}\,\mathrm{kg\ m\ s^{-1}}$.

6. REFERENCIAS

[1] El paquete de látex está disponible en el ITS, ver la versión gratuita de PC vaya a MikTeX *http://miktex.org/* o en *http*: *//www.dur.ac.uk/its/software/tex/texresources/*