PROCESADORES DE LENGUAJES

Prácticas de laboratorio

Curso 2017/2018

CONSTRUCCIÓN DE UN ANALIZADOR DEL LENGUAJE BASIC

Universidad de Alcalá
Dpto. Ciencias de la Computación
Asignatura 780018 2017-18

Índice

1. Introducción

El lenguaje BASIC, original creado por John George Kemeny y Thomas Eugene Kurtz en el Colegio Dartmouth en 1964, fue diseñado inicialmente para enseñar programación a personas no técnicas y funcionó bien para eso.

El BASIC mínimo es la forma más simple de la lengua que alguna vez se ha estandarizado, y proporciona suficientes características para resolver muchos problemas matemáticos. Usando un lenguaje simple con un número pequeño de palabras clave y un pequeño número de características reduce la cantidad de Sintaxis y Semántica que debe memorizar, y le permite pasar más tiempo en la realidad de la programación.

Este dialecto BASIC permite aprender los conceptos esenciales de programación iterativa y procedural sin perderse en las funciones avanzadas de un lenguaje de programación moderno y de tamaño completo. Este lenguaje era interpretado, no compilado, de tal modo que el ordenador tomaba una línea y la ejecutaba.

Se pide crear un analizador léxico, sintáctico y semántico del '*ECMA-55 Minimal BASIC*' de tal modo que podamos chequear cualquier programa escrito con esa especificación y saber si tiene errores léxicos, sintácticos o semánticos, antes de proceder a su ejecución.

El BASIC es un lenguaje orientado a líneas. Un programa BASIC es una secuencia de líneas, la última de las cuales debe ser una línea final (END) y cada una de las cuales contiene, al menos, una palabra clave. Cada línea contendrá un número de línea único que sirva de etiqueta para la declaración contenida en esa línea.

En esta práctica usaremos el BASIC dialecto estándar '*ECMA-55 Minimal BASIC*'[1] de BASIC

[1] Una referencia completa del estándar puede consultarse en: http://buraphakit.sourceforge.net/ECMA-55,1st_Edition,_January_1978.pdf

Primera Parte (PCL1)

2. Analizador Léxico

Especificaciones Léxicas

La parte léxica reconoce identificadores, números, operadores, signos de puntuación, símbolos y palabras reservadas, de todos ellos se generará token para el analizador sintáctico.

El juego de caracteres para BASIC está contenido en el conjunto de caracteres ECMA de 7 bits codificado.

Las cadenas son secuencias de caracteres y se utilizan en los programas BASIC tanto para comentarios, REM (*REMARK*), como constantes de cadena (declaración de Constantes) o como datos de entrada (INPUT)

EL 1. Los Caracteres y Cadenas permitidos vienen definidos por la siguiente sintaxis:

1. **Letra** → 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
2. **Dígito** → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
3. **Car_Cadena** → ? | **Car_Cad_Delimitado**
4. **Car_Cad_Delimitado** → ! | # | $ | % | & | ‘ | (|) | * | , | / | : | ; | < | =| > | ? |^| _ | **Car_No_Delimitado**
5. **Car_No_Delimitado** → *espacio* | **Car_Cadena_Simple**
6. **Car_Cadena_Simple** → + | - | . | **Digito** | **Letra**
7. **Cad_REM** → **Car_Cadena***
8. **Cad_Delimitada** → “ **Car_Cad_Delimitado*** “
9. **Cad_No_Delimitada** → **Car_Cadena_Simple** | **Car_Cadena_Simple Cad_No_Delimitada*** **Car_Cadena_simple**
10. **LF** → `lf`
11. **CR** → `cr`
12. **EOF** → `eof`

Se considera un error léxicos la detección de un carácter no definido

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

EL 2. El conjunto de las palabras reservadas del lenguaje será: **DATA, DEF, DIM, END, FOR, GO, GOSUB, GOTO, IF, INPUT, LET, NEXT, ON, PRINT, RANDOMIZE, READ, REM, RESTORE, RETURN, STEP, STOP, THEN** y **TO**, las cuales generan el token correspondiente a cada una de ellas.

EL 3. El conjunto de las funciones predefinidas del lenguaje será: **ABS, ATN, COS, EXP, INT, LOG, RND, SGN, SIN, SQR** y **TAN** las cuales generan el token correspondiente a cada una de ellas.

EL 4. IDENTIFICADORES: Las variables en BASIC están asociadas con valores numéricos o de cadena.

a. Las variables numéricas simples se nombrarán con una letra. Las variables numéricas suscritas se denominarán por una letra seguida de una o dos expresiones numéricas encerradas entre paréntesis[2].

b. Las variables de cadena se denominarán con una letra seguida de un signo de dólar.

No se requieren declaraciones explícitas de tipos de variables; Un signo de dólar sirve para distinguir la cadena de las variables numéricas.

EL 5. CONSTANTES pueden indicar tanto valores numéricos escalares como valores de cadena.

a. Una constante numérica es una representación decimal en la notación posicional de un número. Hay cuatro formas sintácticas generales de constantes numéricas:

- Representación número entero con o sin signo `sd ... d`
- Representación número real con o sin signo `sd..d•d..d`
- Representación número escalada entero con o sin signo `sd..dEsd..d`
- Representación número escalar real con o sin signo `sd..d•d..dEsd..d`

dónde:

- `d` es un dígito decimal,
- `•` es el punto decimal
- `s` es un signo y es opcional
- `E` es el carácter que indica notación exponencial.

b. Una constante de cadena es una cadena de caracteres entre comillas dobles (**"**).

EL 6. Puede haber espacios en blanco, CR, LF y/o tabuladores en el fichero a analizar.

EL 7. El carácter EOF determina el final del análisis léxico.

Requisitos Léxicos

RL 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 léxico.

[2] Este caso no debería ser detectado en el analizador léxico

RL 2. En la memoria justificativa se deberá incluir un Autómata Finito Determinista (AFD) por cada una de las expresiones regular definidas[3].

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

RL 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.

Ejemplo de entrada:

```
10 INPUT "Cual es tu nombre?"; U$
20 PRINT "Hola "; U$
30 INPUT "Cuantas estrellas quieres?"; N
40 LET S$ = ""
50 FOR I = 1 TO N
60 LET S$ = S$ + "*"
70 NEXT I
80 PRINT S$
90 INPUT "Quieres mas estrellas?"; A$
120 IF A$ = "S" THEN GOTO 30
130 PRINT "Adios !! "; U$
140 END
```

Ejemplo de Salida del analizador léxico:

```
Análisis Léxico completado, sin errores

Resultado del análisis léxico

[ent(10), INPUT, const(Cual es tu nombre?), punto_coma
,ide(U$), CRLF, ent(20), PRINT, const(Hola), punto_coma,
ide(U$), CRLF, ent(30), INPUT , const(Cuantas estrellas
quieres?)", punto_coma, id(N), CRLF, ent(40), LET,
id(S$), igual, const(), CRLF, ent(50), FOR, ide(I),
igual, const(1), TO, ide(N) CRLF, ent(60), LET, ide(S$),
igual, id(S$), mas, const(*), CRLF, ent(70), NEXT,
ide(I), CRLF, ent(80), PRINT ide(S$), CRLF, ent(90),
INPUT, const(Quieres mas estrellas?), punto_coma,
ide(A$), CRLF, ent(120), IF, id(A$), igual, const(S),
THEN, GOTO, const(30), CRLF, ent(130), PRINT, const
(Adios !!), punto_coma, ide(U$), CRLF, ent(140),END,
EOF]
```

[3] 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.

Segunda parte (PCL2)

3. Analizador Sintáctico

Especificaciones del Lenguaje

Un programa BASIC es una secuencia de líneas y cada una de las cuales contiene, al menos, una palabra clave. Cada línea contendrá un número de línea único que sirva de etiqueta para la declaración contenida en esa línea.

Cada línea comenzará con un número de línea. Los valores de los números enteros representados por los números de línea serán positivos no nulos; Los ceros a la izquierda no tendrán ningún efecto. Las declaraciones se realizarán en orden de número de línea ascendente.

Las líneas del programa estarán en orden secuencial para su ejecución, empezando por la primera línea, hasta que:

- Alguna otra acción es ejecutada por una declaración de control, o
- Se produce una condición de excepción, que da como resultado una terminación del programa, o
- Se ejecuta una sentencia STOP o una sentencia END.

Se observarán convenciones especiales respecto a los espacios. Con las siguientes excepciones, los espacios pueden ocurrir en cualquier lugar de un programa sin afectar la ejecución y pueden usarse para mejorar la apariencia y la legibilidad del programa.

Los espacios no aparecerán:

- Al principio de una línea
- Dentro de las palabras reservadas
- Dentro de constantes numéricas
- Dentro de los números de línea
- Dentro de nombres de funciones o variables
- Dentro de símbolos de relación de dos caracteres

Todas las palabras reservadas de un programa estarán precedidas por al menos un espacio y, si no es final de una línea, deberán estar seguidas por al menos un espacio.

La forma en que se detecta el final de una línea de la sentencia viene determinada por la implementación; p.ej. El final de línea puede ser un carácter de retorno de carro (`CR`), un carácter de retorno de carro seguido de un carácter de avance de línea (`CR+LF`) o el final de un registro físico(`EOF`).

Las líneas en un programa de conformidad con el estándar pueden contener hasta 72 caracteres; El indicador de fin de línea (CR o CR+LF o EOF) no está incluido dentro de este límite de 72 caracteres.

La sentencia final (END) sirve tanto para marcar el final físico del cuerpo principal de un programa como para terminar la ejecución del programa cuando se encuentre.

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:

ES 1. Las ***variables*** en BASIC están asociadas con valores numéricos o de cadena y, en el caso de variables numéricas, pueden ser variables simples o referencias a elementos de matrices unidimensionales o bidimensionales; Tales referencias se denominan variables suscritas.

- Las variables numéricas simples se nombrarán con una letra.
- Las variables numéricas suscritas se denominarán por una letra seguida de una o dos expresiones numéricas encerradas entre paréntesis, si son dos expresiones se separarán por una coma. Una variable con subíndice se refiere al elemento en la matriz de una o dos dimensiones seleccionado por el valor (s) del subíndice (s).
  - El valor de cada subíndice se redondea al entero más próximo.
  - El rango de cada subíndice es de cero a diez inclusive. Las expresiones de subíndice tendrán valores dentro del rango apropiado
- Las variables de cadena se denominarán con una letra seguida de un signo de dólar. Las expresiones de cadena se componen de una variable de cadena o una constante de cadena.
- No se requieren declaraciones explícitas de tipos de variables; Un signo de dólar sirve para distinguir la cadena de las variables numéricas, y la presencia de un subíndice distingue una variable con subíndice de una simple.
- Un identificador de variable puede ser declarado más de una vez y este sobre escribirá en contenido que pudiese tener.

Ejemplos:

```
X   A5   V(3)   W(X, X + Y/2)   S$   C$
```

ES 2. Las **expresiones numéricas** pueden construirse a partir de variables, constantes y referencias de funciones utilizando las operaciones de suma, resta, multiplicación, división y potenciación.

La formación y evaluación de las expresiones numéricas sigue las reglas algebraicas normales.

Los símbolos `^`, `*`, `/`, `+` y `-` representan las operaciones de potenciación, multiplicación, división, suma y resta, respectivamente. A menos que los paréntesis dicten lo contrario, las potencias se realizan primero, luego multiplicaciones y divisiones, y finalmente, adiciones y sustracciones.

En ausencia de operaciones de la misma precedencia están asociadas a la izquierda.

```
A-B-C se interpreta como (A-B) -C,
A ^ B ^ C se interpreta como (A ^ B) ^ C,
A / B / C se interpreta como (A / B) / C
-A ^ B se interpreta como - (A ^ B).
```

Cuando el orden de evaluación de una expresión no está limitado por el uso de paréntesis, y si el uso matemático de los operadores es asociativo, conmutativo o ambos, entonces el uso completo de estos se pueden hacer propiedades para revisar el orden de evaluación de la expresión.

NOTA : No se utilizará declaración de precedencias en el fichero de especificación CUP y deberá reflejar estas precedencias con el diseño de la gramática

ES 3. Los valores de las **funciones suministradas** por la implementación, así el número de argumentos requeridos para cada función definidas en EL3 se ven seguidamente. En todos los casos, X representa una expresión numérica.

- ABS (X) El valor absoluto de X.
- ATN (X) El arco tangente de X en radianes, es decir, el ángulo cuya tangente es X. El rango de la función es (pi / 2) <ATN (X) < (pi / 2)
- COS (X) El coseno de X, donde X está en radianes.
- EXP (X) La exponencial de X, es decir, el valor de la base de logaritmos naturales (e = 2,71828 ...) elevado a la potencia X; Si EXP (X) es menor que la máquina infinitesimal, entonces su valor será reemplazado por cero.
- INT (X) El entero más grande no mayor que X;
  p.ej. INT (1.3) = 1 e INT (-1.3) = -2.
- LOG (X) El logaritmo natural de X; X debe ser mayor que cero.
- RND El siguiente número pseudoaleatorio en una secuencia proporcionada por la implementación de números pseudoaleatorios distribuidos uniformemente en el rango 0 <= RND <1
- SGN (X) El signo de X: -1 si X < 0, 0 si X = 0 y +1 si X > 0.
- SIN (X) El seno de X, donde X está en radianes.
- SQR (X) La raíz cuadrada no negativa de X; X debe ser no negativo.
- TAN (X) La tangente de X, donde X está en radianes.

ES 4. El lenguaje BASIC permite definir **nuevas funciones** (funciones de usuario) dentro de un programa. La forma general de declaraciones para definir funciones es

```
DEF FNX = expresión  o  DEF FNX (parámetro) = expresión
```

Donde X es una sola letra mayúscula y un parámetro, que es una simple variable numérica.

Una referencia de función definida por el usuario es una notación para la invocación de un algoritmo definido, en el cual el valor del argumento, si lo hay, es sustituido por el parámetro que se utiliza en la definición de la función. Todas las funciones a las que se hace referencia en una expresión deben ser implementadas o definidas en una declaración **DEF**. El resultado de la evaluación de la función, lograda mediante la ejecución del algoritmo de

definición, es un valor numérico que sustituye a la referencia de función en la expresión.

Ejemplos:

```
DEF FNF (X) = X ^ 4 - 1          DEF FNP = 3,14159
DEF FNA (X) = A * X + B
```

ES 5. La asignación, como ya habrá deducido, utiliza el operador "=". La instrucción **LET** permite que la asignación de valor de una expresión a una variable. La forma sintáctica general será

```
LET variable = expresión
```

Ejemplos:

```
LET P = 3,14159 LET A(X,3) = SIN(X) * Y + 1
LET A$ = "ABC"
```

ES 6. La **comparación** entre dos expresiones utiliza los operadores de relación.

```
<  |  <=  |  =  |  >=  |  >  | <>
```

ES 7. Las **sentencias de control** permiten la interrupción de la secuencia normal de ejecución de sentencias haciendo que la ejecución continúe en una línea especificada, en lugar de en la que tiene el número de línea superior siguiente.

- Sentencia **GOTO** - Ir al número de línea y realizas una transferencia incondicional.

- La Sentencia **IF-THEN**

  ```
  IF  exp1  comparación  exp2 THEN número de línea
  ```

  Donde "exp1" y "exp2" son expresiones y "comparación" es un operador relacional, permite una transferencia condicional.

- Las sentencias de **GOSUB** y **RETURN** - Permiten las llamadas de subrutina y cuando termina volver a la siguiente línea.

  ```
  GOSUB número de línea
  ...
  RETURN
  ```

- La Sentencia **ON GOTO** Permite transferir el control a una línea seleccionada

  ```
  ON expresión GOTO número de línea,..., número de línea
  ```

- La Sentencia **STOP** se usa para detener un programa antes de llegar a la sentencia **END**.

  Un programa sólo puede tener una instrucción END y debe estar en la última línea, pero puede contener cualquier número de instrucciones STOP.

Ejemplos

```
GOTO 999                        IF X > Y + 83 THEN 200
IF A$ <> B$ THEN 550            ON L + 1 GOTO 300,400,500
```

ES 8. La instrucción **FOR** y **TO** proporcionan la construcción de bucles. La forma sintáctica general de la sentencia for y la siguiente sentencia es

```
FOR v = valor inicial TO límite STEP incremento
...
NEXT v
```

Donde "v" es una variable numérica simple y el "valor inicial", "límite" y "incremento" son expresiones numéricas; La cláusula " STEP incremento " es opcional.

Ejemplos

```
FOR I = 1 TO 10                 FOR I = A TO B STEP -1
...                                ...
NEXT I                          NEXT I
```

ES 9. La sentencia **PRINT** está diseñado para la generación una salida de resultados tabulados en un formato coherente. La forma sintáctica general de la sentencia PRINT es

```
PRINT item p item p ... p item
```

Donde cada item es una expresión y cada signo de puntuación `p` es una coma (añade un tabulador) o un punto y coma (espacio en blanco).

- Una vez terminada la impresión generará un final de línea, completando así la línea de salida actual.
- Si esta línea no contiene caracteres, entonces se produce una línea en blanco

ES 10. Las sentencias de entrada de datos, **INPUT,** proporcionan la interacción con el usuario con uy el programa en ejecución al permitir que se asignen variables a los valores que son suministrados por un usuario.

La instrucción de entrada permite la entrada de datos de cadena y numéricos mixtos, con elementos de datos separados por comas. La forma sintáctica general de la instrucción de entrada es

```
INPUT variable, ..., variable
```

Ejemplos

```
INPUT X      INPUT X, A$, Y(2)      INPUT A, B, C
  3.14159           2, SMITH, -3          25,0, -15
```

ES 11. La sentencia **DATA** proporciona la creación de una secuencia de representaciones para elementos de datos para su uso por la instrucción de lectura. La forma sintáctica general de la declaración de datos es

```
DATA dato, ..., dato
```

Donde cada `dato` es una constante numérica, una cadena-constante o una cadena.

Ejemplos

```
DATA 3.14159, 100, 5E-10, ","
```

ES 12. La instrucción de lectura, **READ,** proporciona la asignación de valores a variables de una secuencia de datos creada a partir de estados de datos (Véase ES 11). La instrucción **RESTORE** permite que se vuelvan a leer los datos del programa. Las formas sintácticas generales de las sentencias READ y RESTORE son

```
READ variable, ..., variable

RESTORE
```

Ejemplos

```
READ X, Y, Z          READ X(1), A$, C
```

ES 13. La instrucción de **DIM** se utiliza para reservar espacio para matrices. A menos que se indique lo contrario, todos los subíndices de matriz tendrán un límite inferior de cero y un límite superior de diez. Por lo tanto, la asignación de espacio predeterminada reserva espacio para 11 elementos en el array unidimensionales y 121 elementos en arrays bidimensionales. La forma sintáctica general de la declaración de dimensión es

```
DIM declaration, ..., declaration
```

Donde cada declaración tiene la forma de *letra(entero)* o *letra(*entero*, entero)*

Ejemplos

```
DIM A(6), B(10,10)
```

ES 14. La declaración de comentario (**REM**) permite las anotaciones en el programa. Si la ejecución de un programa alcanza una línea que contiene una declaración de comentario, entonces procederá a la siguiente línea sin ningún otro efecto.

Ejemplos

```
REM FINAL CHECK
```

ES 15. La sentencia **RANDOMIZE** anula la secuencia predefinida de ejecución de números pseudoaleatorios como valores para la función *RND*, permitiendo secuencias diferentes (e impredecibles) cada vez que se ejecuta un programa.

Ejemplo

```
RANDOMIZE
```

Requisitos Sintácticos

RS 1.- El analizador solamente ha de aceptar ficheros con extensión "**.bas**".

RS 2.- El analizador también debe ser capaz de detectar los errores sintácticos que no se ajusten a las especificaciones dadas. 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.

Por el contrario, si el analizador hubiera terminado sin encontrar ningún error deberá mostrar un mensaje indicando que ha finalizado sin errores.

RS 3.- El analizador debe ser capaz de recuperarse cuando encuentre un error recuperable y continuar analizando el resto del programa.

RS 4.- Al analizar el archivo que contenga un programa en el lenguaje dado, analizará las producciones creando un Árbol de Derivación[4] (S=>*α) que represente la estructura del programa que mostrara por pantalla, similar al ejemplo en el Anexo I.

[4] Un árbol de derivación, también llamado *parse tree*, posee un diseño en el cual cada producción de la gramática tiene su correspondiente objeto. Ver Anexo II

4. Analizador Semántico

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 de un no terminal.

La clasificación de un error de este tipo semántico 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. 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):

Luego nuestro analizador semántico cumplirá:

ESm 1.- Todos los identificadores de variable utilizados en las expresiones deberán haber sido declarados previamente.

ESm 2.- Un identificador que aparece como un elemento numérico debe ser una variable numérica.

ESm 3.- Un identificador que aparece como un elemento cadena debe ser una variable cadena.

ESm 4.- Una expresión suma, resta, multiplicación, división o potenciación que se aplique será de tipo numérica.

ESm 5.- Un programa puede contener representaciones numéricas que tienen un número arbitrario de dígitos, aunque las implementaciones pueden redondear los valores de tales representaciones a una precisión definida por la aplicación de no más de seis dígitos decimales significativos.

ESm 6.- La misma letra no debe ser el nombre de una variable simple y una matriz, ni el nombre de una matriz unidimensional y una bidimensional.

ESm 7.- La longitud de la cadena de caracteres asociada con una variable de cadena puede variar durante la ejecución de un programa desde una longitud de cero caracteres (significando la cadena nula o vacía) y puede contener hasta un máximo de 18 caracteres.

ESm 8.- Una variable con subíndice se refiere al elemento en la matriz de una o dos dimensiones seleccionado por el valor (s) del subíndice (s). El valor de cada subíndice se redondea al entero más próximo. A menos que se declare explícitamente en una instrucción de dimensión, las variables suscritas se declaran implícitamente por su primera aparición en un programa.

ESm 9.- El valor del argumento de la función LOG no será cero o negativo. El valor del argumento de la función SQR no será es negativo.

ESm 10.- La magnitud del valor de la función exponencial o tangente es mayor que el infinito de la máquina (no fatal, el procedimiento de recuperación recomendado es suministrar el infinito de la máquina con el signo apropiado y continuar).

ESm 11.- Una definición de función, DEFx, debe producirse en una línea numerada inferior a la de la primera referencia a la función. El parámetro que aparece en la lista de parámetros de una definición de funciones local a esa definición, es decir, es distinto de cualquier variable con el mismo nombre fuera de la función definición. Las variables que no aparecen en la lista de parámetros son las variables del mismo nombre fuera de la definición de la función.

ESm 12.- En una sentencia de asignación, LET, 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.

ESm 13.- Una instrucción de entrada, INPUT, hace que las variables de la lista de variables se asignen, en orden, a los valores de la entrada-respuesta. Los tipos de la variable en la parte izquierda de la asignación y los tipos de la expresión en la parte derecha debe ser del mismo tipo.

ESm 14.- En una sentencia condicional, IF, la expresión debe ser de tipo booleano, mientras que las sentencias que forman parte de la sección THEN será una constante natural.

ESm 15.- La expresión ON-GOTO se evaluará y redondeará para obtener un número entero, cuyo valor se utilizará para seleccionar un número de línea de la lista que sigue al GOTO (los números de línea de la lista se indexan de izquierda a derecha, Empezando por 1). La ejecución del programa continuará en la declaración con el número de línea seleccionado. Todos los números de línea en las instrucciones de control se referirán a las líneas del programa. El número entero obtenido como valor de una expresión en una sentencia ON-GOTO no puede ser menor que uno o mayor que el número de números de línea en la lista.

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.

Todas las variables o constantes son de tipo numérico o cadena. 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	numérica, cadena
Nombre[5]	nombre de la variable
Tabla de símbolos	Conjunto de pares de la forma [Nombre, Tipo]

Estos atributos se sintetizan a partir de la *secuencia de declaraciones* y/o se heredan en la *secuencia de comandos* de un programa.

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

ESm 16.- Deberá construir una estructura para el almacenamiento de la tabla de símbolos similar a la descrita.

ESm 17.- 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, o Tabla de Símbolos).

Tabla 1 Ejemplo de tabla de Atributos

No terminal	Atributo sintetizado	Atributo heredado
<linea>	-	*Tabla de Símbolos*
<sentencia a>	*Tabla de Símbolos*	-
<declaration>	*Tabla de Símbolos*	-
….		
<cadena>	-	Tabla de Símbolos, Tipo

5 Según especificaciones.

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 Derivación), 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 derivación.

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 Derivación: Si el programa fuente que se está analizando no tuviese errores irrecuperables, deberá presentar el árbol de derivación resultado del análisis de programa fuente.

6. Evaluación de la práctica

Elementos a entregar por el alumno

Elementos obligatorios 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, **2** correctos y **8** erróneos.
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, expresiones, 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 árbol de derivación: Descripción de su estructura y organización.
   - Diez casos de prueba y su salida. Deberá incluirse en la memoria un anexo con los diez casos listados. Dos 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 derivación 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, en caso contrario no será evaluada y obtendrá una calificación de cero puntos.

Si el alumno utilizo para el desarrollo de la práctica algún IDE en concreto, sería aconsejable el envío del proyecto completo en el IDE que hayan utilizado o, en caso de que lo hayan hecho sin utilizar un IDE, un ejecutable.

Calificación

Para proceder a la evaluación de la práctica los analizadores deberán funcionar correctamente. Esta evaluación se divide en cuatro niveles, para lo cual los analizadores funcionan correctamente:

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).

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, indicando las especificaciones y requisitos cumplido, 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:

A.- Primera Parte (10% + 5% de la nota final)
Análisis Léxico

Implementación de los requisitos y las especificaciones léxicas y, además, sobre escrito el tratamiento de errores sintácticos.

B.- Segunda Parte (30 % + 5% de la nota final)

1 Análisis Sintáctico

Implementación de los requisitos y las especificaciones sintácticas y, además, sobre escrito el tratamiento de errores sintácticos.

2 Análisis Semántico

Implementación de los requisitos y las especificaciones léxicas, sintácticas y semánticas y se realiza un tratamiento de errores semánticos.

Se evaluará la memoria justificativa (10% de la nota final) en función de la calidad y las 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 - EJEMPLO

Ejemplo 1

```
>>> Práctica 2017/2018 <<<
Nombre del programa: programa1.bas
```

```
10 INPUT "Cual es tu nombre?: "; U$
20 PRINT "Hola "; U$
30 INPUT "Cuantas estrellas quieres?: "; N
40 LET S$ = ""
50 FOR I = 1 TO N
60 LET S$ = S$ + "*"
70 NEXT I
80 PRINT S$
90 INPUT "Quieres mas estrellas? "; A$
120 IF A$ = "S" THEN GOTO 30
130 PRINT "Adios !! "; U$
140 END
```

Análisis Léxico

```
 [10,INPUT,"Cual es tu nombre?: ",;,Ident{U$},n/l
  ,20,PRINT,"Hola ",;,Ident{U$},n/l,
  ,30,INPUT,"!#$%&'()*,/;.<=>^_ Cuantas estrellas quieres?:",;,Ident{N}, n/l
 ,40,LET,Ident{S$},=,"", n/l
 ,50,FOR,Ident{I},=,1,TO,Ident{N}, n/l
 ,60,LET,Ident{S$},=,Ident{S$},+,"*",
 ,70,NEXT,Ident{I}, n/l
 ,80,PRINT,Ident{S$}, n/l
 ,90,INPUT,"!#$%&'()*,/;.<=>^_ Quieres mas estrellas?",;,Ident{A$}, n/l
 ,120,IF,Ident{A$},=,"S",THEN,GOTO,30, n/l
 ,130,PRINT,"!#$%&'()*,/;.<=>^_ Adios !! ",;,Ident{U$}, n/l
 ,140,REM "!#$%&'()*,/;.<=>^_ Adios !! ,n/l
 ,150,END,n/l
 ,eof
 ]
```

Árbol de Derivacion

```
+--<Lineas> ::= Natural <Sentencias> FinLinea <Lineas>
|  +--10
|  +--<Sentencias> ::= <Sentencia>
|  |  +--<Sentencia> ::= INPUT <Lista Impresion>
|  |  |  +--INPUT
...
|  |  |  |  |  |  |  |  |  |  +--<Valor> ::= <Constante>
|  |  |  |  |  |  |  |  |  |  |  +--<Constante> ::= Cadena
|  |  |  |  |  |  |  |  |  |  |  |  +--"Cual es tu nombre?: "
|  |  |  |  +--;
...
|  |  |  |  |  |  |  |  |  |  +--<Valor> ::= ID
|  |  |  |  |  |  |  |  |  |  |  +--U$
|  +--{ n/l }
...
...
...
|  |  |  |  |  |  |  |  |  |  |  +--<Lineas> ::= Natural <Sentencias> FinLinea
|  |  |  |  |  |  |  |  |  |  |  |  +--140
|  |  |  |  |  |  |  |  |  |  |  |  +--<Sentencias> ::= <Sentencia>
|  |  |  |  |  |  |  |  |  |  |  |  |  +--<Sentencia> ::= END
|  |  |  |  |  |  |  |  |  |  |  |  |  |  +--END
|  |  |  |  |  |  |  |  |  |  |  |  +--{ n/l }
+--{eof}
```

8. ANEXO II - Arboles de Derivación

Un árbol de derivación permite mostrar gráficamente cómo se puede derivar cualquier cadena de un lenguaje a partir del símbolo distinguido de una gramática que genera ese lenguaje.

Un árbol es un conjunto de puntos, llamados nodos, unidos por líneas, llamadas arcos. Un arco conecta dos nodos distintos. Para ser un árbol un conjunto de nodos y arcos debe satisfacer ciertas propiedades:

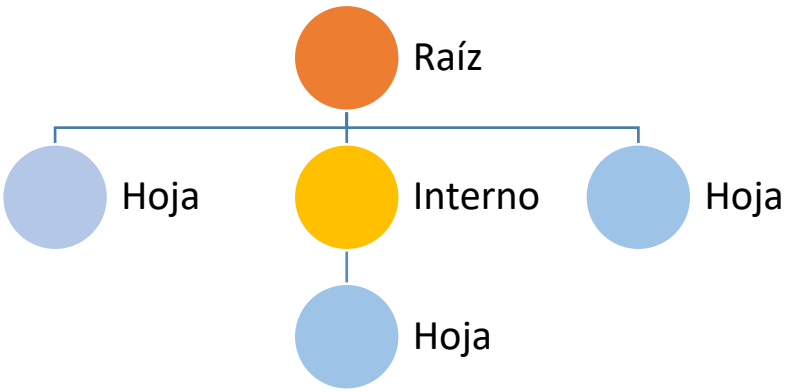
- Hay un único nodo distinguido, llamado raíz (se dibuja en la parte superior) que no tiene arcos incidentes.
- Todo nodo c excepto el nodo raíz está conectado con un arco a otro nodo k, llamado el padre de c (c es el hijo de k). El padre de un nodo, se dibuja por encima del nodo.
- Todos los nodos están conectados al nodo raíz mediante un único camino.
- Los nodos que no tienen hijos se denominan hojas, el resto de los nodos se denominan nodos interiores.

Propiedades de un árbol de derivación.

Sea G = (N, T, S, P) una gramática libre de contexto, sea A ∈ N una variable. Diremos que un árbol T_A = (N, E) etiquetado es un árbol de derivación asociado a G si verifica las propiedades siguientes:

- La raíz del árbol es un símbolo no terminal
- Cada hoja corresponde a un símbolo terminal o ε.
- Cada nodo interior corresponde a un símbolo no terminal.

Para cada cadena del lenguaje generado por una gramática es posible construir (al menos) un árbol de derivación, en el cual cada hoja tiene como rótulo uno de los símbolos de la cadena.



Para cada cadena del lenguaje generado por una gramática es posible construir (al menos) un árbol de derivación, en el cual cada hoja tiene como rótulo uno de los símbolos de la cadena.

Si un nodo está etiquetado con una variable X y sus descendientes (leídos de izquierda a derecha) en el árbol son $X_1, \ldots, X_k$, entonces hay una producción $X \rightarrow X_1 \ldots X_k$ en G.

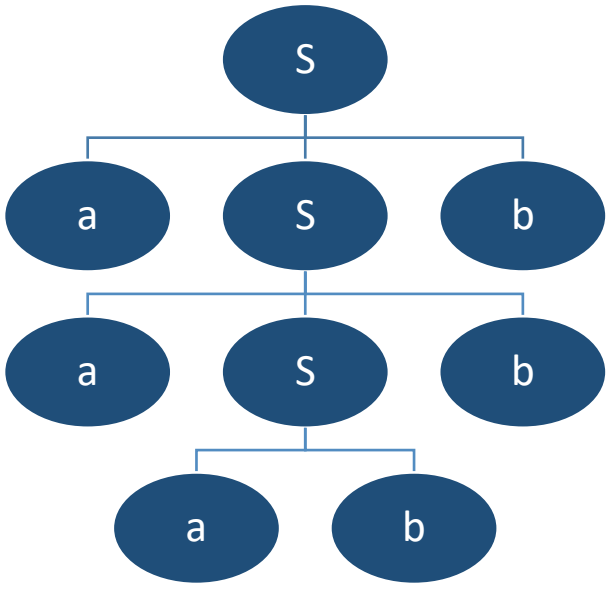
Sea G = (N, T, S, P) una GLC. Un árbol es un árbol de derivación para G si:

1. Todo vértice tiene una etiqueta tomada de $T \cup N \cup (\varepsilon)$
2. La etiqueta de la raíz es el símbolo inicial S
3. Los vértices interiores tienen etiquetas de N
5. Si un nodo n tiene etiqueta A y $n_1\ n_2 \ldots n_k$ respectivamente son hijos del vértice n, ordenados de izquierda a derecha, con etiquetas $x_1, x_2 \ldots x_k$ respectivamente, entonces: $A \rightarrow x_1 x_2 \ldots x_k$ debe ser una producción en P
6. Si el vértice n tiene etiqueta ε, entonces n es una hoja y es el único hijo de su padre.

Ejemplo de Árbol de derivación.

Sea G=(N, T, S, P) una GLC con P: $S \rightarrow ab \mid a\ S\ b$

La derivación de la cadena aaabbb será: $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$ y el árbol de derivación:



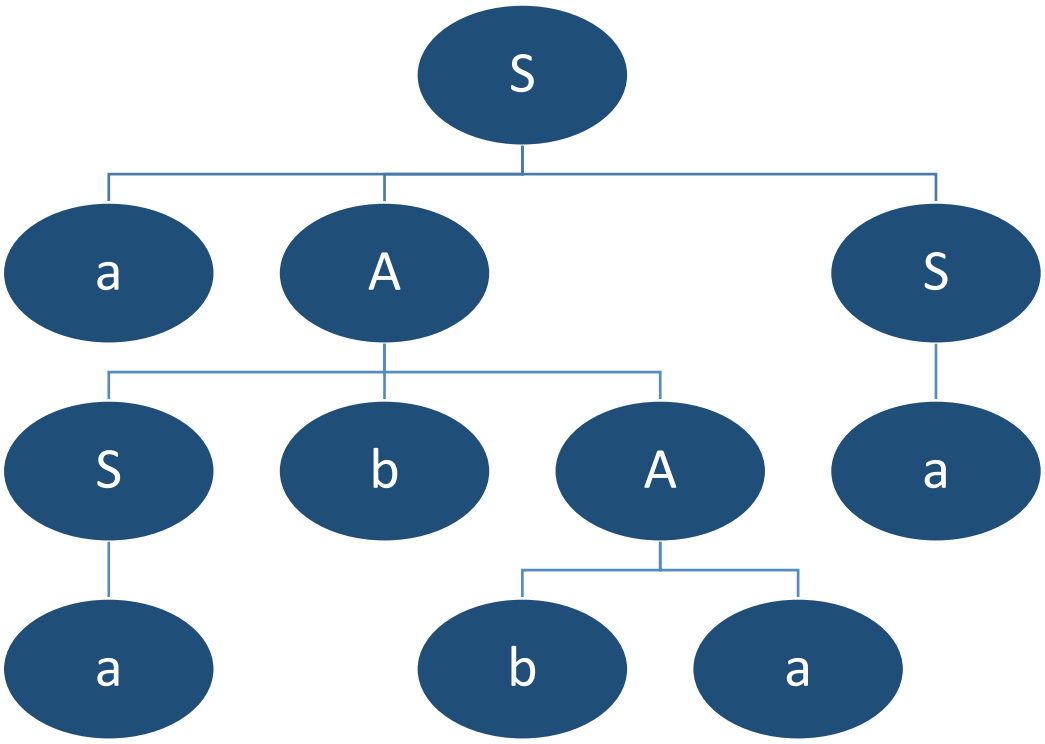
Relación entre derivaciones y árboles

Si leemos las etiquetas de las hojas de izquierda a derecha tenemos una sentencia. Llamamos a esta cadena la producción del árbol de derivación.

Teorema. Sea G=(N,T,S,P) una GLC. Entonces $S \Rightarrow^* \alpha$ (de S se deriva α) si y sólo si hay un árbol de derivación en la gramática G con la producción α.

Si w es una cadena de L(G) para la gramática libre de contexto G, entonces w tiene al menos un árbol de derivación. Referido a un árbol de derivación particular, w tendrá una única derivación a la izquierda y otra única a la derecha.

Ejemplo.



Derivación a la izquierda:

$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaS \Rightarrow aabbaa$

Derivación a la derecha:

$S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa \Rightarrow aSbbaa \Rightarrow aabbaa$