PROCESADORES DE LENGUAJES

Práctica 3
Creación de un compilador sencillo

Parte 1. Analizador Léxico y sintáctico

Curso 2019/2020

1. Introducción

Esta práctica será la última que realizaremos este curso, y por tanto la más extensa. Durante su realización aplicaremos todos los conocimientos adquiridos en la asignatura para la creación de un compilador para un lenguaje de programación sencillo cuya especificación se dará más adelante.

Nuestro compilador, tras la realización de las tres fases de análisis generará el AST del programa que se compile. Posteriormente, y partiendo de ese AST realizará la traducción del mismo al lenguaje de la máquina virtual ROSSI (ver documento `Rossi.pdf` distribuido con el emulador), el cual podrá ejecutarse en un emulador para esa máquina virtual llamado `TheDoc`.

La implementación del compilador se podrá realizar en Python, java o C, a elección del grupo, aunque los ficheros que se proporcionen como guía estarán escritos en en lenguaje Python.

2. Analizador Léxico

Empezaremos por implementar el analizador léxico de nuestro lenguaje. En esta implementación obviaremos algunas tareas del mismo que se añadirán mas tarde, como la incorporación de los identificadores a la tabla de simbolos. Tras su implementación, y antes de continuar conviene verificar que nuestro analizador léxico funciona correctamente.

2.1. Especificación informal

En el aspecto léxico, nuestro lenguaje utilizará las categorías léxicas que utilizan la mayoria de los lenguajes, aunque en algunos casos su forma será diferente. A continuación haremos un repaso de las categorias léxicas que utilizaremos y describiremos informalmente las características que tendrán

1. **Comentarios** Como todos los lenguajes el nuestro nos permitirá incluir comentarios que hagan más legibles nuestros programas. En nuestro lenguaje solemente tendremos comentarios de linea que empezarán por una doble barra inclinada (//) y que terminarán al final de la linea donde se escribe.

2. Los espacios en blanco solamente se utilizan para separar los diferentes componentes léxicos. Se utilizarán tabulaciones y saltos de linea simplemente para hacer el código más legible.

3. Los identificadores **id** seran cadenas de letras y números que empiecen por una letra. Estos identificadores pueden tener una longitud máxima.

4. Los números que utilizaremos pueden ser tanto enteros como reales, y su definición viene dada por las siguiente e

$$\begin{array}{ccc} digitos & \rightarrow & digito\ digito^* \\ fraccion\_opt & \rightarrow & .\ digitos|\lambda \\ num & -> & digitos\ fraccion\_opt \end{array}$$

5. El conjunto de palabras clave del lenguaje está formado por las siguientes palabras: PROGRAMA, VAR, VECTOR, ENTERO, REAL, BOOLEANO, INICIO, FIN, SI, ENTONCES, SINO, MIENTRAS, HACER, LEE, ESCRIBE, Y, O, NO, CIERTO y FALSO. Estas palabras reservadas deben estar escritas en mayúscula.

6. Los operadores que podremos utilizar en nuestro lenguaje son los siguientes.

   - Operadores relacionales son: =, <>, <, <=, >=, >

- Operadores aritméticos los representaremos como $+$, $-$, $*$, $/$
- . El operador de asignacion es $:=$

7. Junto a las categorías indicadas anteriormente, en nuestro lenguaje utilizaremos también los siguientes símbolos: dos puntos (**:**), punto y coma (**;**), coma (**,**), corchete de apertura y de cierre (**[** y **]**) y los parentesis de apertura y cierre (**(** y **)**)

2.2. Implementación del Analizador Léxico

Como ya comentamos en clase, una vez que tenemos definida la MDD que especifica las diferentes categorias léxicas, esta MDD debe ser implementada. Para ello tenemos dos opciones. La primera es definir la MDD mediante una tabla de adyacencia,y la segunda es implementar directamente en el lenguaje de programación el comportamiento de la MDD. Esta segunda opción será la que nosotros adoptaremos. Para conseguir una implementación completa de nuestro analizador léxico tendremos que abordar tres elementos que trataremos en los siguientes apartados.

2.2.1. Categorías Léxicas

Cada una de las categorías léxicas debemos implementarla como una clase. Cada clase tendrá unos atributos diferentes en función de la información que sobre esa categoría deberemos almacenar.

Todas estas clases heredarán de una clase genérica que representa una categoría general donde se agrupan los atributos y métodos comunes y que se da ya en el fichero componentes.py.

Debemos decidir que categorias léxicas vamos a utilizar. Para ello debemos decidir, por ejemplo, que vamos a hacer con los operadores relacionales si definiremos una categoria para cada uno o definiremos una categoria en la que englobaremos todos los operadores y los diferenciaremos con un atributo.

Ejercicio 1

Completa el fichero componentes.py con la definición de todas las clases que representen las categorías léxicas que vayas a utilizar. Es conveniente que en cada componente se almacene el número de linea en el que se encuentra para posteriormente utilizar esa información en los mensajes de error.

2.2.2. El analizador

El analizador deberá leer la entrada carácter a carácter y agrupar estos caracteres en los diferentes elementos. Para ello, tras la lectura de un carácter, deberá comprobar en que rama de la MDD está para continuar por ella. Una vez que se llega el estado final de esa rama, el analizador debe devolver un objeto de la clase que se ha identificado. Este analizador lo implementaremos como una nueva clase que encapsulará el método que implementa el comportamiento de la MDD.

Un esqueleto de esta clase la podeis encontrar en el fichero analex.py que se os proporciona.

Ejercicio 2

Completa el fichero lexico.py para que implemente el analizador léxico.

2.2.3. Tratamiento de errores léxicos

Durante la fase del análisis léxico solamente detectaremos errores de la aparición de caracteres no permitidos en el lenguaje.

3. Analizador sintactico

3.1. Especificación informal

Un programa tras la palabra reservada PROGRAMA y un identificador constará de dos partes:

- Una declaración de variables globales.
- Una sentencia compuesta que representará el cuerpo del programa principal.

Las partes primera será opcionales, mientra que la última es obligatoria y será un conjunto de instrucciones que iran entre las palabras INICIO y FIN. Todas las instrucciones terminarán en punto y coma (;).

Los tipos elementales que admite el lenguaje son los enteros, reales y booleanos. Además, el lenguaje permite la definición de vectores de elementos de cualquier tipo básico del lenguaje. Para ello se emplea la palabra reservada `VECTOR` seguida de un número (que indica el número de elementos[1]) entre corchetes , la palabra reservada `DE` y el tipo base del vector.

Un ejemplo de definiciones variables s sería:

```
  var
x, y: ENTERO;
    lista: VECTOR [10] DE REAL;
```

En nuestro programa podremos definir comentarios de linea de forma que van desde el inicio de los mismos con el simbolo // hasta el final de linea.

Definiremos a continuación de forma más formal la especificación del lenguaje.

3.2. Especificación formal.

La especificación sintáctica de nuestro lenguaje viene dada por la gramática mostrada en el cuadro 1.

4. Ficheros proporcionados

Para facilitar la implementación del analizador léxico y para que sirva de punto de partida, en campus virtual se proporciona el fichero Eq_compila.zip, dentro del cual podéis encontrar cuatro ficheros.

- **componentes.py** donde se ha definido la clase general y algunas de las clases concretas que representan las categorias léxicas.
- **analex.py** Proporciona el esqueleto de la clase que implemente el analizador léxico.
- **Prueba1.eje** Define un ejemplo de un programa en el lenguaje para el que estamos creando este compilador para que os sirva como ejemplo para probar vuestro programa.
- **flujo.py**. Implementa las funciones necesarias para el control del flujo de entrada.

Para facilitar la implementación del analizador sintáctico se proporciona el fichero anasint.py con el esqueleto de algunos métodos.

[1] Los índices del vector van de 0 al tamaño especificado menos uno

⟨Programa⟩	→	**PROGRAMA id ;** ⟨decl_var⟩ ⟨instrucciones⟩ **.**
⟨decl_var⟩	→	**VAR** ⟨lista_id⟩ **:** ⟨tipo⟩ **;** ⟨decl_v⟩ $\|\lambda$
⟨decl_v⟩	→	⟨lista_id⟩ **:** ⟨tipo⟩ **;** ⟨decl_v⟩ $\|\lambda$
⟨lista_id⟩	→	**id** ⟨resto_listaid⟩
⟨resto_listaid⟩	→	**,** ⟨lista_id⟩ $\|\lambda$
⟨Tipo⟩	→	⟨tipo_std⟩ | **VECTOR [num] de** ⟨Tipo_std⟩
⟨Tipo_std⟩	→	**ENTERO** | **REAL** | **BOOLEANO**
⟨instrucciones⟩	→	**INICIO** ⟨lista_inst⟩ **FIN**
⟨lista_inst⟩	→	⟨instrucción⟩ **;** ⟨lista_inst⟩ $\|\lambda$
⟨instrucción⟩	→	**INICIO** ⟨lista_inst⟩**FIN** | ⟨inst_simple⟩ | ⟨inst_e/s⟩
	→	**SI** ⟨expresion⟩ **ENTONCES** ⟨instruccion⟩ **SINO** ⟨instrucción⟩
	→	**MIENTRAS** ⟨expresión⟩ **HACER** ⟨instruccion⟩
⟨Inst_simple⟩	→	**id** ⟨resto_instsimple⟩
⟨resto_instsimple⟩	→	**opasigna** ⟨expresión⟩ | **[** ⟨expr_simple⟩ **]** **opasigna** ⟨ expresión⟩ $\|\lambda$
⟨variable⟩	→	**id** ⟨resto_var⟩
⟨resto_var⟩	→	**[** ⟨expr_simple⟩**]** | λ
⟨inst_e/s⟩	→	**LEE (id)** | **ESCRIBE (** ⟨expr_simple⟩**)**
⟨expresión⟩	→	⟨expr_simple⟩ **oprel** ⟨expr_simple⟩ | ⟨expr_simple⟩
⟨expr_simple⟩	→	⟨término⟩ ⟨resto_exsimple⟩ | ⟨signo⟩ ⟨término⟩ ⟨resto_exsimple⟩
⟨ resto_exsimple⟩	→	**opsuma** ⟨termino⟩ ⟨resto_exsimple⟩ | **O** ⟨termino⟩ ⟨resto_exsimple⟩ $\|\lambda$
⟨termino⟩	→	⟨factor⟩ ⟨resto_term⟩
⟨resto_term⟩	→	**opmult** ⟨factor⟩ ⟨resto_term⟩ | **Y** ⟨factor⟩ ⟨resto_term⟩ $\|\lambda$
⟨factor⟩	→	⟨variable⟩ | **num** | **(** ⟨expresión⟩ **)** | **NO** ⟨factor⟩ | **CIERTO** | **FALSO**
⟨signo⟩	→	$+\|-$

Cuadro 1: Especificación sintactica de nuestro lenguaje

5. Ejemplo

Para que podais ir probando vuestras implementaciones se proporciona el siguiente ejemplo

```
PROGRAMA p1;
VAR i,x:INTEGER;
INICIO
  i:=0.0;
   MIENTRAS (i<>5) HACER
    INICIO
      x:=i*4;
      ESCRIBE(x);
    FIN;
FIN.
```

Una vez completado vuestro analizador léxico la salida que debemos obtener sería la mostrada en 1.

```
PR (valor: PROGRAMA, linea: 1)
Identif (valor: p1, linea: 1)
PtoComa
PR (valor: VAR, linea: 2)
Identif (valor: i, linea: 2)
Coma
Identif (valor: x, linea: 2)
DosPtos
Identif (valor: INTEGER, linea: 2)
PtoComa
PR (valor: INICIO, linea: 3)
Identif (valor: i, linea: 4)
OpAsigna
Numero (valor: 0.0, linea: 4, tipo: <type 'float'>)
PtoComa
PR (valor: MIENTRAS, linea: 5)
ParentAp
Identif (valor: i, linea: 5)
OpRel (valor: SimDist, linea: 5)
Numero (valor: 5, linea: 5, tipo: <type 'int'>)
ParentCi
PR (valor: HACER, linea: 5)
PR (valor: INICIO, linea: 6)
Identif (valor: x, linea: 7)
OpAsigna
Identif (valor: i, linea: 7)
OpMult (valor: SimMult, linea: 7)
Numero (valor: 4, linea: 7, tipo: <type 'int'>)
PtoComa
PR (valor: ESCRIBE, linea: 8)
ParentAp
Identif (valor: x, linea: 8)
ParentCi
PtoComa
PR (valor: FIN, linea: 9)
PtoComa
PR (valor: FIN, linea: 10)
Punto
```

Figura 1: Salida del analizador léxico

Tras completar el analisis sintáctico el resultado debe ser que es correcto.

5.0.1. Temas organizativos

Una vez finalizada la implementación de cada uno de los analizadores, y antes de darlos como definitivos y pasar a la siguiente fase, hay que realizar pruebas con diferentes programas. Para ello define, a partir del dado otros casos de prueba que te permitan probar su correcto funcionamiento.

Para el correcto desarrollo de la práctica la implementación del analizador léxico debería concluirse en 3 sesiones de prácticas y la implementación del analizador sintactico con su gestión de errores en 3 o 4 semanas.

6. Cuestiones de organización y evaluación

1. El desarrollo de esta práctica se realizará a lo largo del resto del curso.
2. Al finalizar la misma se deberá realizar una memoria en la que se recojan los siguientes apartados.
   - **Introducción**. Realizar una pequeña Introducción a los objetivos y resultados esperados
   - **Descripción formal del lenguaje**
     - Especificación completa de todas las categorías lexicas del lenguaje y descripción del MDD equivalente
     - Comprobación de que la gramática que define el lenguaje es LL(1).
   - **Detalles de implementación** Descripción de los aspectos más relevantes de la implementación.
   - **Conclusiones**. Opinión personal sobre la práctica, tiempo empleado y sugerencias para mejorar la misma.
3. La memoria, junto al codigo fuente del compilador deberá entregarse antes del dia 21-Diciembre a las 23:55 horas.
4. Tras la finalización de cada una de las partes, el profesor podrá realizar una revisión del trabajo realizado con cada uno de los grupos para resolver dudas y comprobar que todo es correcto hasta el momento.