TC2006 – Lenguajes de Programación

Set de ejercicios 1
Lenguajes de Programación y Compiladores

Matrícula1_____ Matrícula2 _____

Valor: 3 estrellas

Instrucciones:

I. Escoger un compañero de equipo que de preferencia no conozcan bien.

II. Crear el documento **ejercicio1.pdf** con las respuestas de los problemas 1 a 4 y los archivos de Python **obten_token.py** y **parser.py** para los problemas 5 y 6,

III. Uno de los integrantes debe subir un documento compreso **M_ejercicio1.zip** a Canvas, donde la M sea substituida por sus matrículas separadas con guiones bajos (_).

Problemas:

1. Agregar una lista con los **lenguajes de programación que ha utilizado cada integrante del equipo** indicando su nivel de dominio (experto, intermedio, novato).

2. Para el lenguaje de su preferencia (especifíquenlo), **escriban ejemplos de instrucciones que no cumplen las reglas de léxico, sintaxis y semántica**. DOS ejemplos para cada tipo de reglas violadas. En cada caso especificar por qué no cumplen.

3. Describir mediante un **diagrama de sintaxis, un autómata finito y una gramática regular** el léxico de los números de punto flotante de acuerdo a la siguiente descripción.

   INFORMALMENTE: "Un número de punto flotante tiene una mantisa que especifica el valor del número y opcionalmente un exponente que especifica la magnitud del número.

   La mantisa es especificada como una secuencia de dígitos seguida por un punto, seguido por una secuencia uno o más de dígitos que representan la parte fraccionaria del número.

   El exponente (opcional) es especificado usando la letra 'e', seguida opcionalmente por el signo (-), seguido por una secuencia de uno o más dígitos.

Si el exponente está presente, el punto de la mantisa puede omitirse en números enteros.

Ejemplos: 21.46, 15.0, 2.56e2, 3e-2

4. Describir mediante una **gramática libre de contexto y un diagrama de sintaxis** la sintaxis correspondiente a la declaración de variables, de forma que reconozca declaraciones como las tres siguientes siguientes:

```
int         x;
char  c, s[10];
float  f, g, h;
```

En lugar de los nombres de tipos, nombres de variables y números, utilizar los nombres de token TYPE, ID y NUM, respectivamente.

5. **Modifiquen el programa del escáner** visto en clase, de tal manera que sea capaz de reconocer el **separador coma** (,), el **operador de asignación** (=) e **identificadores**.

Los identificadores son nombres de objetos de programación (como variables y funciones) que inician con una letra minúscula seguida por cero o más letras minúsculas, guiones bajos y/o dígitos.

Ejemplos: = , hola tasa_interes a3_b45

¿Cómo?

   a. Reconstruyan el autómata finito descrito por la matriz de transiciones del escáner que encontrarán dentro de los documentos del curso en Canvas.
   b. Agréguenle los estados y transiciones necesarios para reconocer los nuevos elementos léxicos.
   c. Modifiquen el archivo **obten_token.py**, que encontraran en el código de los documentos del curso en Canvas, que contiene el código del scanner de la siguiente forma:
      i. Agreguen nuevos *tokens* y modifiquen la matriz de transiciones para manejar los nuevos estados (renglones de la matriz) y las nuevas transiciones (columnas de la matriz).
      ii. Modifiquen la función de filtro para que reconozca los nuevos tipos de caracteres.
      iii. Modifiquen el programa principal para que maneje el caso de los nuevos elementos.
      iv. Verifiquen que el programa trabaje correctamente.
   d. Suban el programa fuente **obtén_token.py** del escáner a Canvas (uno solo de los integrantes).

6. Utilicen el analizador léxico del problema anterior (Escáner) y **modifiquen el programa del Parser** en Python visto en clase para que, además de expresiones aritméticas, **sea capaz de reconocer asignaciones y llamadas a funciones**.

   Las asignaciones se forman con un identificador seguido de un operador de asignación (=) seguido por una expresión aritmética. Las expresiones aritméticas pueden incluir llamadas a función.

   Las funciones se forman con un identificador seguido por una lista de argumentos entre paréntesis. La lista de argumentos puede tener 0 o más elementos separados por comas, los cuales pueden ser expresiones.

   Ejemplos de expresiones válidas:

   ```
   c = a + e $
   v = 5.3 * (2 + x2()) $
   funcion5(2, hola, 3.4, 3 * 5) $
   tasa = 4 * ratio(t_int(2.3, 1), 4) $
   ```

   ¿Cómo?

   a. Modifiquen la gramática original para aceptar asignaciones y llamadas a funciones como se especifica arriba.
   b. Eliminen cualquier ambigüedad en la gramática modificando, si existen, reglas que incluyan recursión izquierda o factores comunes por la izquierda.
   c. Modifiquen el código de las funciones en el archivo **parser.py**, que también encontrarán en Canvas, que implementan la gramática no ambigua.
   d. Prueben el programa y verifiquen que trabaje correctamente.
   e. Suban el programa fuente (texto) del parser modificado a Canvas (uno solo de los integrantes).