Proyecto 2

Analizador Sintáctico

Lenguaje C-

Descripción

Hacer un programa en Python, llamado **`Parser.py`** que contentan una función llamada **`parser(imprime = true)`**, la cual, recibe una bandera booleana **`imprime`** , con valor por defecto **`true`** (su uso se explica más adelante) y regresa el Árbol Sintáctico Abstracto (AST, por sus siglas en inglés) o un mensaje de error.

Utilice un algoritmo Top-Down. Su función, cuando necesite el siguiente token, llamará a su función **`getToken`**, que implementó en su **lexer**.

El programa **`Parser.py`**, al inicio, deberá importar el archivo que contenga el **lexer**:

```
from lexer import *
```

Posteriormente, continuará con la definición de la función **`parser(imprime = true)`** y de todas las funciones auxiliares que requiera.

La función **`parser(imprime = true)`**:

- Deberá declarar las variables globales que requiera para que funcione su **lexer** (ver la sección de "prueba del programa").
- Mediante esas variables podrá manejar el archivo de texto conteniendo un programa en C- que se desea analizar.
- Si la bandera **`imprime`** es true, la función deberá imprimir al final, el AST generado, simplemente con endentación por nodo, es decir, los hijos de un nodo endentados con respecto a su papá.

Detección y recuperación de errores

Si el analizador sintáctico encuentra un error (recuerde que el analizador sintáctico sólo encuentra errores en la estructura del programa con respecto a los tokens), debe marcar la línea y el lugar (token) donde encontró el error. Por ejemplo, si la línea 22 del archivo es:

```
contador += contador + indice
```

Recibe el token: **`(ID, "contador")`**

Y al recibir el token: **`(PLUS, "+")`**

Marcará un error, mandando un mensaje como por ejemplo (sabiendo que después del identificador debe seguir una asignación):

```
Línea 22: Error en la expresión de asignación:
contador += contador + indice
         ^
```

Donde se indica el número de línea y el acento circunflejo (^) indicará la posición del token donde encontró el error.

Después de esto, deberá tener un mecanismo para tratar de **recuperarse del error** (recuerde que se recomienda implementar el método de "botón de pánico", para recorrer los siguientes tokens hasta encontrar uno que vuelva a tener sentido) y continuar con el análisis sintáctico. Desde luego que, nada garantiza la exactitud de lo que se detecte después, lo cual dependerá del mecanismo utilizado y, sobre todo, de la intención que tenga el programador al escribir el código, la cual desconocemos por completo y sólo la podemos suponer.

Prueba del programa

Todos los archivos necesarios para que corra (todos lo que se entregan en Bb, incluyendo **lexer.py** y el que contiene el archivo de prueba) se colocarán en la misma carpeta para evitar el uso de paths adicionales.

El analizador sintáctico será probado con un script que iniciará importando tanto el archivo con los tipos globales como el que contiene su analizador sintáctico. Posteriormente seguirá invocando a su función para obtener el AST, con la opción de impresión activada, hasta que se termine el archivo, es decir, hasta que llegue el token que indica el fin de archivo.

La función **`parser(imprime)`** deberá manejar las siguientes variables globales:

`posicion`: contiene la posición del siguiente carácter del string que se debe analizar. Deberá poder modificarla en su funcionamiento.

`progLong`: contiene la logitud del programa. Sólo la leerá cuando la requiera.

`programa`: contiene el string del programa completo. Sólo la leerá cuando lo requiera.

Por la forma en la que funciona Python al definir las variables y al hacer los **`import`**, la única forma que tenemos de pasarles estas variables globales será por medio de una función que las reciba y le pase el valor recibido a las variables globales que utiliza su programa.

La función para el paso de valores globales la tienen que agregar a su programa y tendrá la siguiente forma:

```
def globales(prog, pos, long):
    global programa
    global posicion
    global progLong
    programa = prog
    posicion = pos
    progLong = long
```

El script con el que se prueba será el siguiente:

```
from globalTypes import *
from Parser import *

f = open('sample.c-', 'r')
programa = f.read()          # lee todo el archivo a compilar
progLong = len(programa)     # longitud original del programa
programa = programa + '$'    # agregar un caracter $ que represente EOF
posicion = 0                 # posición del caracter actual del string

# función para pasar los valores iniciales de las variables globales
globales(programa, posicion, progLong)

AST = parser(true)
```

Para la entrega

- Un documento con:
    - La gramática en la forma en la que usó para programar el parser.
- Todos los archivo Python (comentados) y TXT necesarios para que corra.

La gramática adecuada para la implementación se puede crear a mano, siempre y cuando acepte los programas correctos de C-.

El manual del usuario no será necesario porque ya se dio la definición del lenguaje C- y la forma en la que se probará el programa.

Si este proyecto se fuera a entregar a un tercero, es indispensable entregarla toda la definición del lenguaje y la forma en la que el usuario deberá usar el analizador de léxico, paso a paso.

Nota FINAL:

Este analizador sintáctico también puede generarse utilizando una herramienta de generación automática como Yacc o Bison. Sin embargo, tendría que hacer las adecuaciones (recuerde que estos generadores dan como salida código en C) para que funcione tal como se realizó la descripción anterior de “prueba del programa”.