Universidad del Valle de Guatemala
Compis

NewScanner.py

Base de mi Scanner:
-se asegura de que una secuencia de Peek devuelve tokens siguiendo el token anticipado.
Nodos----------
sirve para el token en donde inicializamos el nombre , el contenido y las excepciones. Ya no termine usando todas las funciones.
Token----------
-para el scanner generado este sirve para guardar las variables en el nuevo scanner pero también sirve para la lectura de textos. Por lo tanto guarda los valores del scanner como tipo de token , posición del token, columna de token, línea de token , valor del token , linked list (el linked list es una estructura de datos lineal en la que los elementos no se almacenan en ubicaciones de memorias contiguas).
Sino también como caracteres, palabras clave, tokens, producciones y las excepciones (para el Core.py)

crear_nodo(): inicializa los nodos en 0
marcar_nodos(): ayuda a reconocer los valores del archivo de texto. Por medio de d definir las palabras clave.

Posición----------
Posición de estiramiento del código fuente (como una acción semántica) lo utilice básicamente en el buffer.
hace comprobaciones entre el inicio relativo del inicio del archivo, el largo del tramo y el número de columna
Buffer----------
Es una clase utilizada por el scanner para leer la secuencia de origen en un buffer y recupera partes de ella. El buffer se inicializa con la secuencia de origen.

Read(): devuelve el siguiente carácter si la entrada está agotada.
funciona: si la posición del token es menor que el largo del buffer entonces el resultado es el buffer con un arreglo de la posición del token. siempre sumándole a la posición del token una posición.

Peek(): permite que el escáner lea los caracteres por delante sin consumirlos.
funciona: si la posición del token es menor que el largo del buffer entonces retorna el buffer con la posición del token sino el buffer retorna EOF osea un carácter unicode.

ObtenerPosición(): marca la posición actual

ObtenerString(): se utiliza para recuperar el intervalo de texto.
funciona: se busca la antigua posición por medio de obtenerposicion() marca la posición como una posición de inicio. Entra en un while donde si inicio es menor que end entonces hace la función de Read en el string vacío y la suma al inicial. retornando al que marca la posición como una antigua posición y devuelve el string .

LeerCaracteres():

funciona: en el resultado es igual al arreglo de buffer en donde se marca la posición que tiene el token y la suma de la posición del token mas el numero de bytes.

determinar_posicion(): permite que el scanner obtenga o marque una posición de lectura que es inicialmente 0.

funciona: si el valor es menor que cero entonces devuelve la posición actual. si el valor mayor o igual al largo del buffer entonces devuelve la posición del token y el largo del buffer

leer_posicion():

funciona: miramos si la posición tiene el valor específico. determinarse la posición y leemos los caracteres.

Scanner----------

-Se ingresa en valores la cantidad de transiciones, símbolos y se inicializan las variables. Se empieza la lectura de archivo y se marca el inicio de lectura del texto de prueba.

init():

```
self.buffer = Buffer( unicode(s) )
```

es la instancia del buffer

```
self.ch
```

es el carácter actual

```
self.posicion_token       = -1
```

numero de columna del carácter actual

```
self.token_linea      = 1
```

numero de linea del carácter actual

```
self.linea_de_inicio = 0
```

posición de la línea actual

```
self.antiguos_n   = 0
```

es para el futuro que es para eols que esten en comentarios

```
self.ignorar    = set( )
```

characteres por ignorar por el scanner (ya no termine utilizando)

```
self.ignore.add( ord(' ') )  # blanks are always white space
```

para ignorar los espacios en blanco.

```
self.corriente_de_tokens = Token( )
```

la corriente completa de tokens.

Siguiente_Caracter(): devuelve el siguiente carácter de entrada.

Función :el carácter es igual que la lectura del buffer. agrega una posición nueva al token. Hace un escaneo del EOL. si este ya tiene EOL entonces se le suma una linea al token en donde se marca la linea de inicio igual a la posicion del token + 1

Check_Literal(): (Es para las keywords , para verificar si es un keyword reservado). Sin embargo, faltó manejar el keyword en el autómata , por lo tanto debe ir

verificando si el keyword lleva un estado de aceptación y determinar qué keyword es entonces.

Siguiente_Token(): método que compara las posiciones de transmisión del siguiente token con el último, en base al autómata obtenido.

clave: ord acepta una cadena de longitud 1 como argumento y devuelve la representación del punto de código unicode del argumento pasado.

Funcion:

-entra en un while verificando el unicode en las variables a ignorar y pasa al siguiente carácter.

-si el unicode de carácter es menor que el largo de la variable de inicializador. entonces el estado es la inicialización con el unicode del carácter. sino devuelve un estado 0.

-buf se inicializa como un unicode en vacío

-buf es la suma del buf con el unicode del carácter

-se marca el inicio del siguiente carácter

- 0 y -1 nos ayudan a ver que el siguiente carácter ya termino

-se sigue una estructura como árbol entonce se marca cada uno de los estados y se prueba si el carácter se encuentra en las variables permitidas. si es así el buf le suma el unicode del carácter obtenido y pasa al siguiente carácter. si no llega a pasar por este mismo entonces el tipo de token entra en la función escanear validando con el número de símbolo.

Escanear():

devuelve el token en base a la validación del token actual igual al token dentro de las linked_list y el peek del token actual que sea igual a las linked list.

Peek - 274 (Scanner)

lee más allá del token de búsqueda anticipada sin eliminar ningún token del flujo de entrada.

Con reiniciar el scanner devuelve otra vez estos mismos tokens.

self.PeekDeTokenActual es igual a la corriente de tokens este mismo es igual al siguiente peek.

Entra en una condición en donde mientras el peek del token actual en su tipo de token es mayor que el máximo número de transiciones entonces el peek del token actual es igual a su linked list (el linked list es una estructura de datos lineal en la que los elementos no se almacenan en ubicaciones de memorias contiguas).

reiniciar():

devuelve otra vez los tokens. por lo tanto hace peek del token actual igual al token.

Core.py

Tabla de Símbolos----------
No la llegue realmente a utilizar todavía.
Sirve para acceder a una tabla de símbolos para descubrir propiedades semánticas de un token. Básicamente tiene métodos para manejar scopes y almacenar y recuperar información de objetos. Entre estos tiene el buscar y comparar.
-Se trabajó en la limpieza y obtención de datos por medio de los Tokens y nodos del Scanner.
-Autómata: (clave: ALL_FOR_IT_2)
el autómata convierte cada palabra en un index , guardando el valor del index en el nuevo scanner. Esto se realizó de esta manera debido a que se manejo los espacios en blanco, los espacios en blanco también los metí dentro de mi autómata.
Se quedó pendiente que los keyword , los keyword se trabaja por medio de ir validando si el keyword lleva un estado de aceptación y decir qué es keyword entonces.
Se la prueba en el newscanner pasar por cada autómata y pasar por un automaton.

El automaton funciona de la siguiente manera: realizando un "MeltStantes" por medio de un OR.

Luego de ello se declararon las siguientes funciones:

1. declarar_variables_de_inicializacion marca todas las variables para validar los automatas con sus respectivos valores
2. la funcion del siguiente token
3. el input del archivo
4. limpieza y definicion de tokens
5. error manager
6. key word manger y otros