Universidad Rafael Landívar
Tradición Jesuita en Guatemala

Facultad de Ingeniería
Compiladores
Segundo Ciclo 2018
Mgtr. Diana Gutiérrez

Proyecto Mini C#

Fase #1 - Analizador léxico

Fecha de entrega: 30 de agosto, 2018, 06:30 pm

El objetivo
En la primera fase del proyecto, usted iniciará su compilador con la aplicación del análisis léxico. Para la primera tarea del front-end, va a utilizar Jflex (flex o cualquier otro generador de analizadores de léxico) para crear un escáner para un lenguaje de programación. El escáner se ejecutará leyendo un archivo escrito en lenguaje fuente, reconociendo los tokens de Mini C# en el orden en que se leen, hasta el final del archivo. Para cada modo, el escáner determinará sus atributos adecuadamente (éstos eventualmente serán utilizados por otros componentes de su compilador) para que la información sobre cada símbolo deba estar correctamente impresa.

Estructura Lexicográfica
Estos son los tokens del lenguaje Mini C#:

- Palabras clave (reservadas):

void int double bool string class interface null this extends implements for while if else return break New NewArray

- Identificadores:

Un identificador es una secuencia de letras, dígitos y guiones bajos, comenzando con una letra. La longitud máxima de un identificador es de 31 caracteres. Mini C# distingue entre mayúsculas y minúsculas (case sensitive).

- **if** es una palabra clave pero **IF** es un identificador
- **binky** y **Binky** son dos identificadores distintos.

- Espacios en blanco

El espacio en blanco (es decir, espacios, tabuladores y saltos de línea) sirve para separar tokens, pero por lo demás debe ser ignorado. Palabras clave y los identificadores deben estar separados por espacios en blanco, o por una señal de que no es ni una palabra ni un identificador.

- **if (23 this** se escanea como cuatro tokens al igual que **if(23this**.

- Comentarios

Mini C# ha adoptado los dos tipos de comentarios disponibles en C++. Un comentario de una sola línea se inicia con // y se extiende hasta el final de la línea. Comentarios de varias líneas comienzan con /* y terminan con la primera */ posterior. Cualquier símbolo

se permite en un comentario, excepto el de secuencia */ que pone fin al comentario actual. Los comentarios de varias líneas no se anidan.

Si un archivo incluye un comentario sin terminar, el escáner debe informar de un error.

- Constantes
  - Las constantes booleanas son: **true** o **false**.
  - Una constante entera puede ser expresada en decimal (base 10) o en hexadecimal (Base 16).
    - Un entero decimal es una secuencia de dígitos decimales (**0-9**).
    - Un entero hexadecimal debe comenzar con **0X** o **0x** (es el carácter cero) y es seguida por una secuencia de dígitos hexadecimales. Los dígitos hexadecimales incluyen los dígitos hexadecimales y las letras **a** a la **f** (ya sea en minúsculas o mayúsculas).
    - Ejemplos de enteros válidos: 8,012,0X0,0x12aE.
  - Una constante double es una secuencia de dígitos, un punto, seguido de una secuencia de dígitos, o nada. Así, **.12** no es una constante tipo double, pero **12.** y **0.12** lo son. Una constante doble puede tener parte exponencial, por ejemplo, **12.2E+2**. Para la constante doble en la notación científica el punto decimal es requerido, el signo del exponente es opcional (si no está especificado, + es asumido), y el **E** puede ser en mayúscula y minúscula. Entonces, **.12E+2** es inválido, pero **12.E+2** es válido. Ceros al inicio de la mantisa y el exponente son permitidos.
  - Una constante string o cadena de caracteres es una secuencia de caracteres encerrada por comillas dobles “”. Los strings pueden contener cualquier carácter excepto una línea nueva o doble comilla. Una constante string debe comenzar y finalizar en una misma línea, y no puede partirse en líneas múltiples. Ejemplos:

    "Está es una cadena de caracteres que no tiene su doble comilla
    Esta no es parte de la cadena de arriba

- Operadores y caracteres de puntuación:

  + - * / % < <= > >= = == != && || ! ; , . [ ] ( ) { } [] () {}

Note que **[**, **]**, y **[]** son tres tokens diferentes y que para el operador **[]**, al igual que los otros dos operadores de dos caracteres, no debe haber ningún espacio en blanco entre los dos caracteres.

Sugerencias

- Cuidado con los espacios entre los patrones. Es muy fácil poner espacios por error y que dichos espacios interpreten de manera diferente un patrón.
- No coloque nuevas líneas en los patrones.
- Cuando tenga duda, coloque paréntesis entre el patrón para asegurar que está obteniendo la precedencia que necesita

- Coloque cada acción entre llaves (si bien es cierto no se requiere una línea simple, es mejor colocarla en una)
- Utilice la sección de definiciones para definir la sustitución de patrones (nombres como Digit, Exponent, etc). Hace las reglas más leíbles y fáciles de modificar, extender o debugear
- Siempre coloque paréntesis alrededor del cuerpo de una definición para asegurar que la correcta procedencia sea mantenida cuando sea substituida

Implementación

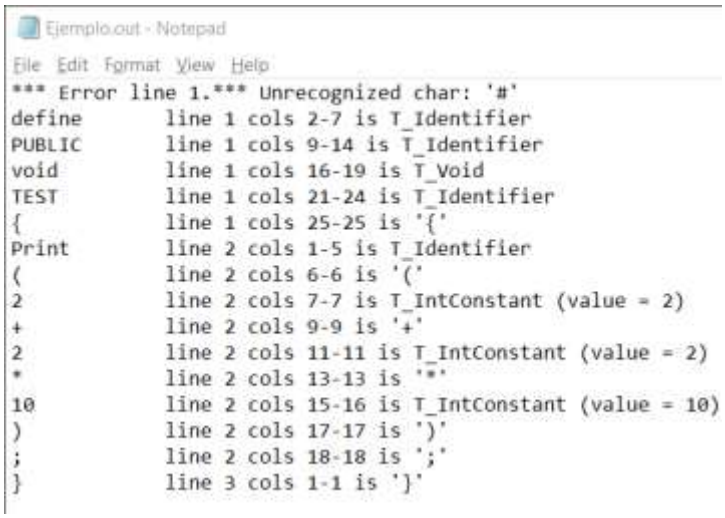
1. Crear un archivo de entrada para JFlex que contenga la definición del lenguaje Mini C#.
2. JFlex leerá la estructura del archivo anterior y generará un programa en Java. Este programa debe ser compilado en vivo.
3. El programa compilado lee un archivo de entrada en lenguaje Mini C# y determina si pertenece o no al lenguaje. Es decir, su analizador debe:
    a. “comerse” los espacios en blanco
    b. Reconocer todas las palabras claves, los símbolos de puntuación, operadores de un carácter, operadores de dos caracteres, las constantes (int, double, bool, and string) e identificadores, retornando el token correcto
    c. Guardar el número de línea, la primera y última columna para todos los tokens
    d. El escáner debe reportar los errores para las cadenas de caracteres no válidas, identificadores de longitud no permitida y caracteres inválidos. Los errores deben ser presentados claramente en pantalla y llevarlos también en el archivo de salida. Si un identificador es más grande que el máximo (31 caracteres) que acepta Mini C#, reporte el error truncando el identificador con los primeros 31 caracteres (descartando el resto), y continuar.

Ejemplos de archivo de entrada en lenguaje mini C#

Example.frag

```
1 #define PUBLIC void TEST{
2 Print(2 + 2 * 10);
3 }
```

Example.out



Ejemplo.out - Notepad

File Edit Format View Help

```
*** Error line 1.*** Unrecognized char: '#'
define       line 1 cols 2-7 is T_Identifier
PUBLIC       line 1 cols 9-14 is T_Identifier
void         line 1 cols 16-19 is T_Void
TEST         line 1 cols 21-24 is T_Identifier
{            line 1 cols 25-25 is '{'
Print        line 2 cols 1-5 is T_Identifier
(            line 2 cols 6-6 is '('
2            line 2 cols 7-7 is T_IntConstant (value = 2)
+            line 2 cols 9-9 is '+'
2            line 2 cols 11-11 is T_IntConstant (value = 2)
*            line 2 cols 13-13 is '*'
10           line 2 cols 15-16 is T_IntConstant (value = 10)
)            line 2 cols 17-17 is ')'
;            line 2 cols 18-18 is ';'
}            line 3 cols 1-1 is '}'
```

Requerimientos generales

- El ejecutable debe ser llamado **minic** y debe pedir el archivo de entrada y escribir todos los resultados en un archivo de salida del mismo nombre al de entrada con extensión **.out**
- El proyecto se entrega en un repositorio en GitHub; se calificará el proyecto hasta el último commit realizado antes de la hora de entrega. Se revisará el historial del repositorio. **No se aceptarán cambios en el momento de la calificación**.
- En el repositorio deberá incluir un archivo README.txt donde explique todo el funcionamiento y porque cree que su programa funciona correctamente y es robusto, así también detallando cómo se manejan los errores.

¡Buena suerte!