Tecnológico de Monterrey

Actividad 3.4:

Actividad integradora

Elaborado por :

Manuel Villalpando Linares

Implementación de métodos computacionales TC2037.601

Comenzaremos analizando cuales son las categorías léxicas en las que se divide C# para poder comprender qué es lo que estaremos separando:

- Palabras reservadas (keywords): palabras que tienen un significado especial en el lenguaje y no pueden ser utilizadas como identificadores (variables, funciones, etc.). Ejemplos: public, class, if, for, while, switch, etc.
- Identificadores (identifiers): nombres dados a variables, funciones, clases, etc. Ejemplos: numero, miFuncion, MiClase, etc.
- Operadores (operators): símbolos que indican operaciones matemáticas o lógicas. Ejemplos: +, -, *, /, &&, ||, etc.
- Literales (literals): valores constantes que aparecen directamente en el código fuente. Ejemplos: 42 (entero), 3.14 (flotante), "hola mundo" (cadena de caracteres), etc.
- Comentarios (comments): texto que se utiliza para documentar o explicar el código y que es ignorado por el compilador. Ejemplos: // esto es un comentario de una sola línea, /* esto es un comentario de varias líneas */, etc.

Ahora analizaremos las expresiones regulares para reconocer cada categoría léxica (estos son solo ejemplos, pueden variar según la implementación):

- Palabras reservadas: public|class|if|else|while|for|switch|case|default|break|continue|return|void|using|namespace|new|static|const|true|false|null
- Identificadores: [a-zA-Z_]\w*
- Operadores: [\+\-*/%\&\|\^\!~]=?|<<=?|>>=?|==|!=|<=?|>=?|\?|\:|\|\||&&
- Literales: 0[xX][0-9a-fA-F]+|\d+\.\d+|\d+|\"[^\"]*\"
- Comentarios: \/\/[^\n]*|\/*.*?*\/

Generación de documentos HTML+CSS:

Para generar documentos HTML+CSS que resalten el léxico de un archivo fuente de C#, utilicé en este caso las etiquetas de HTML y CSS para

los estilos de la página. Por ejemplo, puedes definir una clase CSS para cada categoría léxica y luego envolver cada ocurrencia de esa categoría con una etiqueta con la clase correspondiente.

Reflexión sobre la solución planteada:

Algo a tener en cuenta es que la complejidad de la solución dependerá no solo de la cantidad de expresiones regulares, sino también del tamaño del archivo fuente de C# que se está analizando. Si el archivo es muy grande, el proceso de escaneo y resaltado del léxico puede tardar bastante tiempo, especialmente si se está realizando en tiempo real mientras se edita el archivo. Por lo tanto la complejidad sería O(n), donde n es el número de caracteres del archivo fuente que se está analizando. Esto se debe a que es necesario recorrer todos los caracteres del archivo para detectar los elementos léxicos. Sin embargo la complejidad ideal que se debería alcanzar con el código es O(n(m)), para convertirlo en un código más eficaz por su capacidad de compilar rápidamente.

Otro factor a considerar es la eficiencia de las expresiones regulares utilizadas. Algunas expresiones regulares pueden ser más eficientes que otras, y en general se debe tratar de utilizar expresiones regulares más simples y específicas siempre que sea posible.

En general, la solución propuesta es una buena manera de analizar y resaltar el léxico de un archivo fuente de C# de una manera clara y legible para los desarrolladores. Sin embargo, hay que tener en cuenta que puede ser un proceso intensivo en recursos y es importante optimizar el código tanto como sea posible para garantizar una buena experiencia de usuario.