Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus QRO

TC2037.601

Implementación de métodos computacionales (Gpo 601)

Actividad Integradora 3.4: Resaltador de sintaxis (evidencia de competencia)

Presentado por:

María Fernanda Moreno Gómez A01708653

Uri Jared Gopar Morales A01709413

Profesor:

Pedro Oscar Pérez Murueta

Fecha:

14 de abril de 2023

Índice

Sobre este documento

En presente documento se expone las categorías léxicas del lenguaje de programación C#, identificando palabras reservadas, operadores, literales, comentarios, etc. de este lenguaje de programación.

La finalidad de la obtención de las categorías léxicas de C# es poder crear con ellas expresiones regulares, las cuales nos permitan generar a través de ellas un código en Racket que funcione como resaltador de sintaxis, que reciba un archivo fuente cualquiera de C# y permita resaltar cada categoría léxica identificada por la expresión regular, y, posteriormente, convertir la entrada en documentos de HTML+CSS que puedan resaltar el léxico propuesto.

Finalmente, se presenta un análisis de la complejidad algorítmica y un reporte sobre esta tomando en cuenta el tiempo de ejecución, así como una reflexión acerca de las implicaciones éticas de la tecnología en la sociedad.

Categorías léxicas de C#

Comentarios

Número	Elemento
C01	// Comentario de una línea
C02	/* Comentario multilínea */

Identificadores

Número	Tipo	Ejemplo
I01	Identificador empezando con minúscula	**identifier1**
I02	Identificador comenzando con guión bajo	**_identifier2**
I03	Identificador comenzando con mayúscula	**Identifier3**

Palabras reservadas

Número	Elemento
P01	**abstract**
P02	**as**
P03	**base**
P04	**bool**
P05	**break**
P06	**byte**
P07	**case**
P08	**catch**
P09	**char**
P10	**checked**
P11	**class**
P12	**const**
P13	**continue**
P14	**decimal**
P15	**default**
P16	**delegate**
P17	**do**
P18	**double**
P19	**else**
P20	**enum**
P21	**event**
P22	**explicit**
P23	**extern**

P24	**false**
P25	**finally**
P26	**fixed**
P27	**float**
P28	**for**
P29	**foreach**
P30	**goto**
P31	**if**
P32	**implicit**
P33	**in**
P34	**int**
P35	**interface**
P36	**internal**
P37	**is**
P38	**lock**
P39	**long**
P40	**namespace**
P41	**new**
P42	**null**
P43	**object**
P44	**operator**
P45	**out**
P46	**override**
P47	**params**
P48	**private**
P49	**protected**
P50	**public**

P51	**readonly**
P52	**ref**
P53	**return**
P54	**sbyte**
P55	**sealed**
P56	**short**
P57	**sizeof**
P58	**stackalloc**
P59	**static**
P60	**string**
P61	**struct**
P62	**switch**
P63	**this**
P64	**throw**
P65	**true**
P66	**try**
P67	**typeof**
P68	**uint**
P69	**ulong**
P70	**unchecked**
P71	**unsafe**
P72	**ushort**
P73	**using**
P74	**virtual**
P75	**void**
P76	**volatile**
P77	**while**

Literales

Número	Tipo	Ejemplo
L01	Enteros	**5**
L02	Reales	**-5.1**
L03	Lógicos	

Operadores

Número	Tipo	Elementos
O01	Aritméticos	+, -, *, /, **%**
O02	Lógicos	**&**, **&&**, |, ||, **!**, ^
O03	Relacionales	==, **!=**, >, <, >=, <=,
O04	De manipulación de bits	**&**, |, ^, ~, <<, >>
O05	De asignación	=, +=, -=, *=, /=, **%=**, **&=**, |=, ^=, <<=, >>=, ++, **- -**
O06	De delgados	+, += ,-, -=
O07	Con punteros	**&**, *, ->
O08	Otros varios	/, **:**, **;**, **,**, **.**, **??**

Separadores

Número	Elementos
S01	**;**, **,**, **.**, **(" ")**, **[" "]**, **{" "}**, **<" ">**, **:**, **::**, **...**, **=>**, **??**,

Cadenas

Número	Elementos
C01	“Esto es una cadena de caracteres “

Expresiones regulares

Comentarios

"//." Para los comentarios de una línea

"/*.*/" Para los comentarios multilínea

La primera es debido a que debe de encontrar una línea donde se empiece con doble diagonal (que es la sintaxis para declarar un comentario) y posteriormente pueda contener cualquier otro carácter después de estas dos diagonales.

La segunda expresión regular para los comentarios multilínea la definimos de esa manera para que se pueda buscar en una línea aquella estructura que comience con una diagonal y asterisco, que posteriormente contenga cualquier carácter o caracteres antes de cerrar con un asterisco y una diagonal, que sirven para declarar la sintaxis de cierre de comentarios multilínea.

Identificadores

"^[a-zA-Z_][a-zA-Z0-9_]*$"

Esto se declaró debido a que “^” indica el comienzo de una cadena de texto, seguida de la declaración de “a-zA-Z_” para que se busque una cadena de texto que comience con alguna letra del alfabeto en minúscula o en mayúscula o que comience con un guion bajo. Posteriormente, puede contener 0 o varias veces alguna letra del alfabeto, ya sea en mayúscula o en minúscula o algún número del 0 al 9, o en su defecto, también puede ser un guion bajo, teniendo que terminar con alguno de estos caracteres en alguna parte.

Palabras reservadas

("abstract" "as" "base" "bool" "break" "byte" "case" "catch" "char" "checked" "class" "const" "continue" "decimal" "default" "delegate" "do" "double" "else" "enum" "event" "explicit" "extern" "false" "finally" "fixed" "float" "for" "foreach"

"goto" "if" "implicit" "in" "int" "interface" "internal" "is" "lock" "long" "namespace" "new" "null" "object" "operator" "out" "override" "params" "private" "protected" "public" "readonly" "ref" "return" "sbyte" "sealed" "short" "sizeof" "stackalloc" "static" "string" "struct" "switch" "this" "throw" "true" "try" "typeof" "uint" "ulong" "unchecked" "unsafe" "ushort" "using" "virtual" "void" "volatile" "while")

En esta parte no se pudo hacer una expresión regular como tal, pues cada palabra reservada es distinta en temas de caracteres y la cantidad que esta tiene de ellos. Por esta razón, en el proyecto no definimos una expresión regular para las palabras reservadas, sino que se creó una lista con estas palabras reservadas para posteriormente pintar aquellas que concuerdan con alguna palabra de la lista.

Literales

"^[0-9x]+$"

Esta expresión regular quiere decir que inicia la cadena de caracteres, la cual contiene uno o más caracteres que sean del 0 al 9 o en su defecto, una letra "x", y que en algún momento se termine esta cadena de caracteres.

Operadores

("+" "-" "" "/" "%" "^" "&" "|" "~" "!" "=" "<" ">" "?" ":" ";" "," "." "++" "--" "&&" "||" "==" "!=" "<=" ">=" "+=" "-=" "=" "/=" "%=" "^=" "&=" "|=" "<<=" ">>=" "=>" "??")

Al igual que en las palabras reservadas, no se puede hacer una expresión regular como tal, debido a la gran variedad de operadores, por lo que sería lo mismo el hacer una expresión regular muy larga a hacer una lista de estas, que al igual que en la lista de las palabras reservadas, esta se empareje con el documento de C# con la función "Match".

Separadores

(";" "," "." "(" ")" "[" "]" "{" "}" "<" ">" ":" "::" "..." "=>" "??")

Al haber una gran variedad de separadores y ningún patrón en común, hacer una expresión regular para esta categoría léxica no iba a ser la mejor opción, ya que iba a ser algo muy largo y no muy eficiente. Optamos de igual manera por hacer una lista de los separadores para poder llamarla posteriormente en nuestra implementación.

Cadenas

“^\".*\"$”

Esta expresión regular quiere decir que inicia la cadena de caracteres, la cual inicia forzosamente con comillas dobles, para posteriormente tener 0 o más caracteres para finalizar con otras comillas dobles forzosamente, denotando la sintaxis de los strings en C#.

Reflexión

En este trabajo utilizamos Racket, que es un lenguaje de programación multiparadigma, es decir, que tiene varios estilos de programación.

Por ello, lo implementamos en la solución a este problema, el cual era la creación de un resaltador de sintaxis basado en las categorías léxicas del lenguaje de programación C#. Para llegar a la solución de este reto, se investigó acerca de las categorías léxicas de C#, y posteriormente, se crearon las expresiones regulares de aquellas categorías léxicas donde era posible, tomando en cuenta los caracteres, su posición y la cantidad de estos. Aquellas categorías léxicas donde no era posible crear una buena expresión regular para crear un patrón que las definiera a todas, hicimos una lista con todas estas para hacer más adelante una búsqueda en el archivo de Cs.

Una vez con estas bases, ahora sí pudimos programar en Racket. Primeramente, definimos las expresiones regulares, así como las listas de aquellas categorías léxicas donde no era posible hacer expresiones regulares en un formato corto. Se implementó una función “Match”, donde empareja operadores, cadenas, literales, identificadores y comentarios, para poder colorearlas de acuerdo a las condiciones del archivo de CSS, que da estilo a nuestra página de HTML.

Las palabras reservadas, al ser iguales que una palabra o varias en el archivo Cs, se pudieron colorear y emparejar por medio de una función llamada “Member”, donde prácticamente buscaba en el archivo Cs aquella(s) palabras que coincidieran con alguna palabra de la lista de keywords o palabras reservadas.

Luego, se mandó a llamar un archivo de entrada de Cs y uno de salida de tipo HTML, que tuviera como hoja de estilos el archivo CSS que contiene el código de colores para cada categoría léxica de C#. En este archivo HTML se manda a imprimir el código de C# siguiendo el estilo de CSS, por lo que para el final, tendremos un ejecutable de HTML que contenga el código de Cs pero resaltado conforme al CSS.

Los algoritmos implementados fueron de entrada y salida para poder recibir y mostrar información, algoritmos de ordenamiento y búsqueda para poder manipular los datos de

entrada (el código de Cs), estructuras de datos para la manipulación de los datos y algoritmos de flujo, para poder hacer ciclos que lean el código línea por línea.

Sobre el algoritmo

La complejidad de nuestro código fue de O(n), esto es debido a que tenemos ciclos donde “n” aparece como el límite del ciclo, ya sea para recorrer la lista y para recorrer línea por línea el código. No puede ser O(n^2), ya que no tenemos ciclos anidados, ni O(log2 n) porque el contador no aumenta de otra forma que no sea lineal. Este algoritmo es proporcional al tamaño de la entrada y es eficiente en tiempos de ejecución debido a que este aumenta dependiendo del tamaño de entrada, esto lo podemos verificar con el tiempo de ejecución de nuestra implementación, la cual fue: cpu time: 15 real time: 9 gc time: 0.

Ética en el desarrollo de tecnologías

La ética juega un papel muy importante para el desarrollo de tecnologías, ya que se tiene un gran poder cuando se crean estas debido al manejo de datos que estas almacenan y necesitan. Como ingeniero de software, es una responsabilidad utilizar toda aquella información obtenida para los sistemas de manera responsable, sin fines de lucro personales ni para terceros.

La información hoy en día se ha vuelto un recurso muy valioso, ya que nos define, nos dice quienes somos, nos da información acerca de qué son las cosas. La cosa es que la información no siempre es manejada de la mejor manera, uno como usuario presta menos atención a los permisos y condiciones de las apps, no se les presta demasiada atención a los datos que les estamos entregando, y lo mínimo que esperas de aquellas empresas a las cuales les estás regalando tu información es que la usen de manera responsable, sin vender a alguien más, pero eso es algo que no podemos evitar, el mal uso de información es algo que se puede hacer a escondidas de la ley, por lo que si tienes una gran responsabilidad como ingeniero, como persona que sabes lo que estás haciendo, úsala para bien, úsala para no perjudicar a las personas. Todos merecemos la privacidad de nuestros datos y la seguridad de los mismos, busquemos siempre hacer un buen impacto en la sociedad para maximizar los beneficios y trascender.

Referencias

Anónimo. (Septiembre 16, 2021). *Estructura léxica.* Microsoft Build. Recuperado el 11 de abril del 2023 de: https://learn.microsoft.com/es-es/dotnet/csharp/language-reference/language-specification/lexical-structure#line-terminators

Carleton. (s.f.). *CS 251: Tokenizer.* Recuperado el 12 de abril del 2023 de: https://www.cs.carleton.edu/faculty/dmusican/cs251f14/interp3tokenizer.html

González, J. (s.f.). *El lenguaje de programación C#.* Recuperado el 11 de abril del 2023 de: https://dis.um.es/~bmoros/privado/bibliografia/LibroCsharp.pdf

Racket (s.f.). *Racket Documentation.* Recuperado el 12 de abril del 2023 de: https://docs.racket-lang.org/

ZetCode (Enero 4, 2023). *C# lexical structure.* Recuperado el 11 de abril del 2023 de: https://zetcode.com/csharp/lexical-structure/