Instituto Tecnológico de Estudios Superiores Monterrey

CAMPUS QUERÉTARO

Implementación de métodos computacionales

Pedro Óscar Murueta

Actividad integradora 3.4

REFLEXIÓN

PRESENTA

Ariann Fernando Arriaga Alcántara - A01703556

Pablo César Jiménez Villeda - A01703517

Fecha:

29 de Abril de 2022

Categorías léxicas identificadas

Token	Ejemplos	Expresiones regulares
Variables	miString, mi_vriable1, var123, a1, MIvar	#rx"[a-zA-Z][a-zA-Z0-9_]*"
Reales	4.5, 0.005, 3.1E, 6.1E-8	#rx"\\-*[0-9]+\\.[0-9]+((E|e)-*[0-9]*)*"
Enteros	7, 6, 8, -4	#rx"\\-*[0-9]+"
Comentarios	// esto es un comentario, // desarrollar funcion 3 + 5	#rx"\\/\\/.*"
Strings	“esto es una cadena 123 + 567”, “+ - * /”	#rx"\".*\""
Palabras reservadas	#include, scanf, printf, auto, else, long, switch, break, enum, register, typedef, case, extern, return, union, char, float, short, unsigned, const, for, signed, void, continue, goto, sizeof, volatile, default, if, stati, while, do, int, struct_Packed, double	#rx"\\#include|\\scanf|\\printf|\\auto|\\else|\\long|\\switch|\\break|\\enum|\\register|\\typedef|\\case|\\extern|\\return|\\union|\\char|\\float|\\short|\\unsigned|\\const|\\for|\\signed|\\void|\\continue|\\goto|\\sizeof|\\volatile|\\default|\\if|\\stati|\\while|\\do|\\int|\\struct_Packed|\\double"
Operadores	\\^, /, &, +, -, *, =, (,)	#rx"\\^|\\/|\\&|\\\|\\+|\\-|*|\\=|\\(|\\)"

Solución:

La solución implementada en scheme para el resaltador de sintaxis se hizo usando el lenguaje de programación funcional racket para identificar y colorear los distintos tokens del lenguaje de programación C. para usar esto se utilizaron expresiones regulares para detectar patrones, primero definimos la cabecera de lo que vendrá en el documento html para ahí colocar el output del subrayador y añadir los estilos necesarios. Estos estilos buscan permitir al usuario diferenciar entre los distintos tokens usando los colores de la temática DOBRI NEXT DARK disponible para visual studio. La solución se enfoca en detectar los tokens y reconociendo su tipo agregarlo a un string envuelto en una tag de html llamada span, a la cual se le agregará una clase determinada por su misma token la cual le agregará el color que le corresponde. Los colores y sus tokens respectivos son los siguientes:

```
.comment {
    color: #5c6370;
}
.operand {
    color: #56d6d6;
}
```

```
.integer {
    color: #ff3c00;
}
.real {
    color: #ffcc00;
}
.variable {
    color: #f78c6c;
}
.palabraReservada {
    color: #cb6cfe;
}
.string {
    color: #c3e88d;
}
```

Una vez definido este header que incluía los estilos anteriores se creó una función lexer que recibiría dos cadenas de texto, el nombre del archivo de entrada y el nombre del archivo de salida en el que se colocaría el código de html. Lo que se hace en esta función es añadir un string generada por la llamada de la función lexer-lines que recibe una lista de cadenas de texto con el contenido de cada renglón del archivo de entrada. Esta función usa una expresión regular que se encarga de detectar todos los tokens presentes en cada línea en orden gracias a una expresión regular que detecta todos los tipos de tokens identificadas. para esto se usó la función integrada en el lenguaje llamada regexp-match*, la cual detecta todos los tokens en una cadena y retorna una lista de coincidencias. Esta posteriormente llama la función lexer-line que se encarga de iterar todas las coincidencias y llamar la función detect-token para cada una, la cual usando un condicional y las expresiones regulares definidas identifica en qué categoría entra cada match y retorna una string con la clase agregada a una etiqueta span que envuelve al token seleccionado.

Con esto se obtiene un programa que puede recibir un archivo de entrada, dicho archivo de entrada, puede ser cualquier código escrito en el lenguaje C, y el resultado o lo que devuelve el programa es un archivo html, donde todos los tokens detectados son retornados como etiquetas de span y tienen su respectiva clase la cual por medio de CSS lo asocia a un color, entregando un archivo html donde se visualizan los tokens de un código de C por diferentes colores, cumpliendo la funcionalidad de ser un subrayador de sintaxis, además de esto, el tiempo de ejecución de el programa utilizando la función time, en su totalidad fue 1 gc.

Complejidad de solución:

La complejidad de esta solución es de O(N * T), en la que N representa la longitud del archivo, es decir, la cantidad de caracteres que contiene el archivo y T representa la cantidad de tokens a buscar o la cantidad de expresiones regulares usadas. Esto se debe a que en una cadena de caracteres, las expresiones regulares usadas se convierten en un DFA que va

buscando todos los matches desde la posición del último match, por lo que busca por cada carácter uno de los patrones determinados en el que en el peor de los casos busca T patrones, es decir la cantidad total de expresiones regulares usadas, por N caracteres.

Para comprobar esto se probó con dos archivos de entrada adicionales que tenían el doble y el triple de longitud que el original respectivamente, con estos se obtuvo una complejidad de 2 gc y 3 gc respectivamente, lo que prueba que teniendo la misma cantidad de tokens, si solo cambia la longitud del archivo esta complejidad de lineal. Coincidiendo el tiempo de ejecución con el cálculo del orden de complejidad.

Resultado:


```c
#include <stdio.h>
int main() {
char c;
printf("Enter a character: ");
scanf("%c", &c);

// %d displays the integer value of a character
// %c displays the actual character
printf("ASCII value of %c = %d", c, c);

return 0;

b=7

int a = 32.4 *(-8.6 - b)/ 6.1E-8

d = a ^ b //Esto es un comentario
}
```

Conclusión:

En conclusión, los beneficios para la sociedad de estos algoritmos y de los DFA son la facilidad para encontrar errores en caso de que no se cumplan reglas gramaticales, permitiendo mejorar la calidad de vida de quienes trabajen con diferentes tipos de archivos, ya que esto facilita la depuración de errores. Además, estos permiten la mejora en la eficiencia de trabajo, ya que lógicamente resaltan los distintos tipos de conjuntos de átomos, permitiéndole al usuario una construcción más acertada de textos que cumplan con las reglas gramaticales estipuladas por los múltiples lenguajes de codificación y escritura. Con esto, uno puede ver la utilidad real de esto para la sociedad, pues son algoritmos que mejoran la calidad de vida de las personas.