



Instituto Tecnológico de Estudios Superiores Monterrey

CAMPUS QUERÉTARO

Implementación de métodos computacionales

Pedro Óscar Murueta

Actividad integradora 5.3

PRESENTA

Ariann Fernando Arriaga Alcántara - A01703556

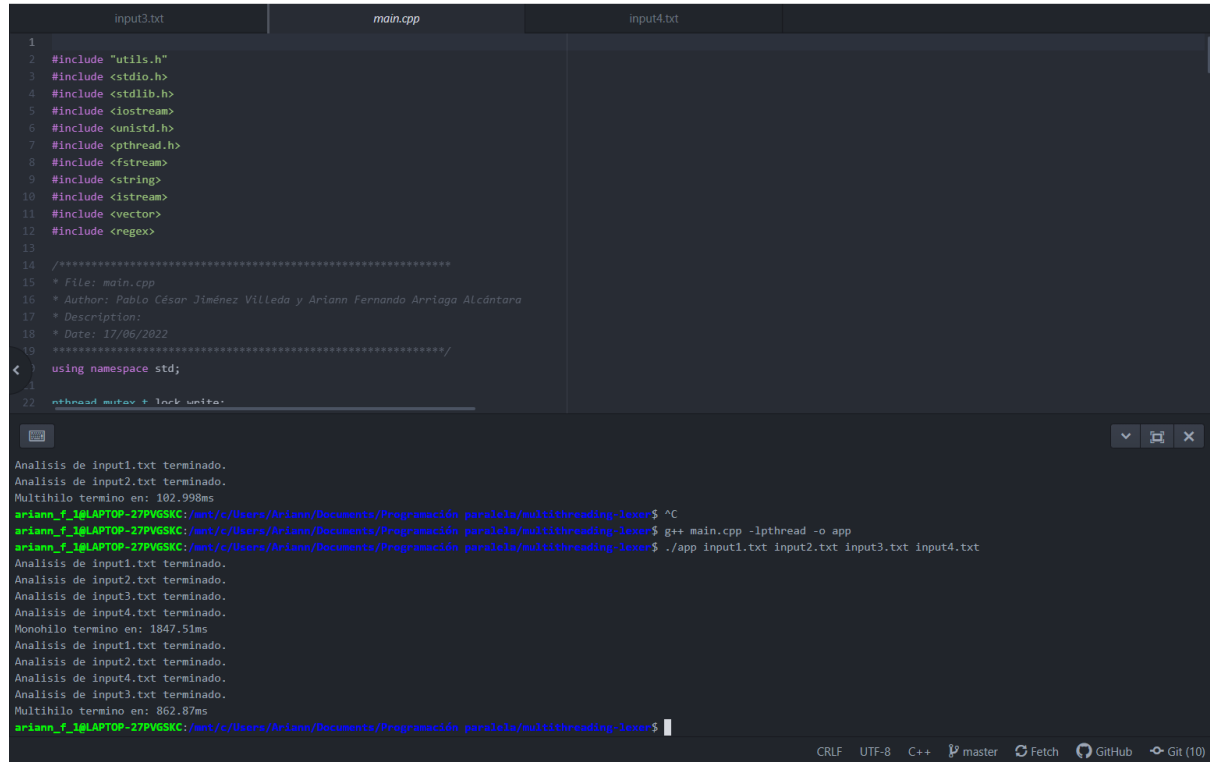
Pablo César Jiménez Villeda - A01703517

Fecha:

18 de junio de 2022

Reporte de Lexer:

Resultado de ejecución del programa:



```
1
2 #include "utils.h"
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <iostream>
6 #include <unistd.h>
7 #include <pthread.h>
8 #include <fstream>
9 #include <string>
10 #include <sstream>
11 #include <vector>
12 #include <regex>
13
14 /*****
15  * File: main.cpp
16  * Author: Pablo César Jiménez Villeda y Ariann Fernando Arriaga Alcántara
17  * Description:
18  * Date: 17/06/2022
19  *****/
20 using namespace std;
21
22 pthread_mutex_t lock write;
```

```
Analisis de input1.txt terminado.
Analisis de input2.txt terminado.
Multihilo termino en: 102.998ms
ariann_f_1@LAPTOP-27PVGSKC:/mnt/c/Users/Ariann/Documents/Programación paralela/multithreading-lexer$ ^C
ariann_f_1@LAPTOP-27PVGSKC:/mnt/c/Users/Ariann/Documents/Programación paralela/multithreading-lexer$ g++ main.cpp -lpthread -o app
ariann_f_1@LAPTOP-27PVGSKC:/mnt/c/Users/Ariann/Documents/Programación paralela/multithreading-lexer$ ./app input1.txt input2.txt input3.txt input4.txt
Analisis de input1.txt terminado.
Analisis de input2.txt terminado.
Analisis de input3.txt terminado.
Analisis de input4.txt terminado.
Monohilo termino en: 1847.51ms
Analisis de input1.txt terminado.
Analisis de input2.txt terminado.
Analisis de input4.txt terminado.
Analisis de input3.txt terminado.
Multihilo termino en: 862.87ms
ariann_f_1@LAPTOP-27PVGSKC:/mnt/c/Users/Ariann/Documents/Programación paralela/multithreading-lexer$
```

Tiempo de ejecución del programa concurrente: 1847.51 ms usando un hilo

Tiempo de ejecución del programa paralelo: 862.87 ms usando 4 hilos

Threads=4

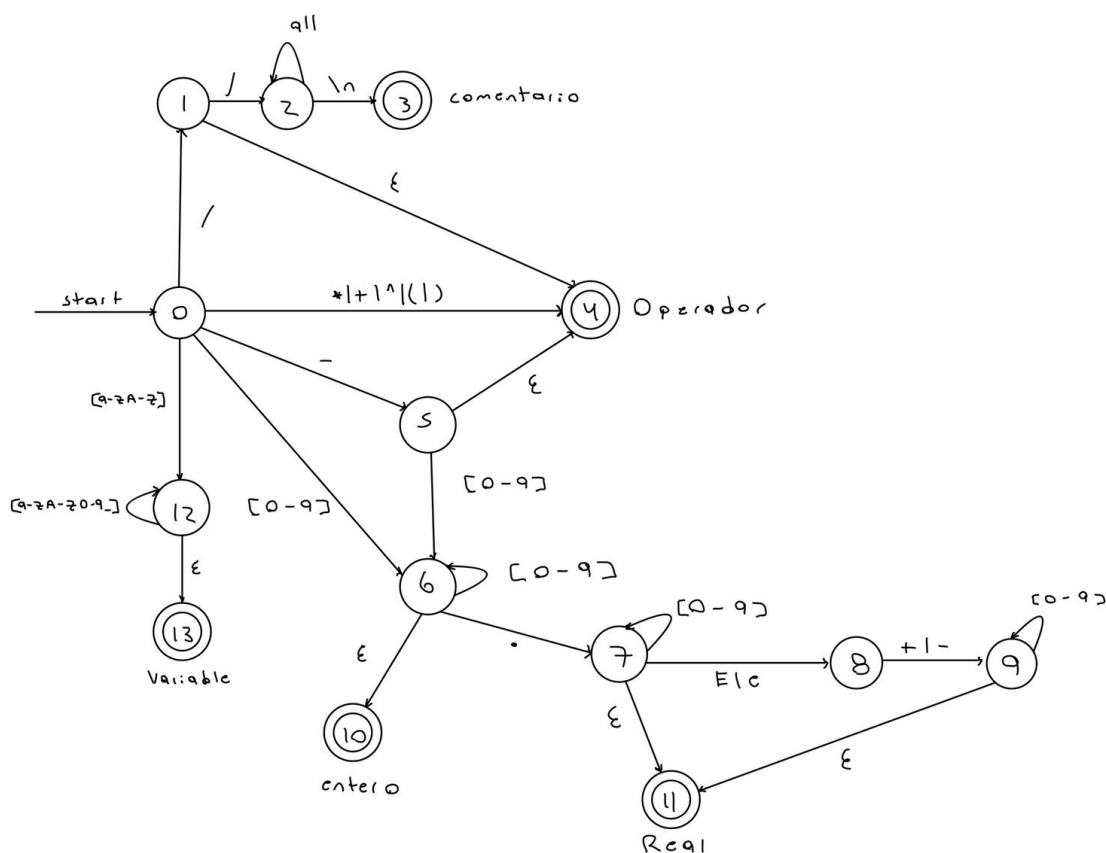
Los hilos en este proceso dependen de la cantidad de archivos que el programa reciba como entrada, pues la creación de los hilos en esta implementación depende de los archivos que se desee analizar. Se asigna un hilo por archivo recibido.

Cálculo del speedup:

$$S_p = \frac{T_1}{T_p}$$

En donde:

- p es el número de procesadores (núcleos).
- T_1 es el tiempo que tarda en ejecutarse la versión secuencial del programa.
- T_p es el tiempo que tarda en ejecutarse la versión paralela del programa utilizando procesadores.



A partir de este autómata se identificaron las categorías léxicas, que se usarán para las expresiones regulares:

Categorías léxicas identificadas

Token	Ejemplos	Expresiones regulares
Variables	miString, mi_variable1, var123, a1, MIvar	<code>#rx"[a-zA-Z][a-zA-Z0-9_]*"</code>
Reales	4.5, 0.005, 3.1E, 6.1E-8	<code>#rx"\-*[0-9]+\.[0-9]+((E e)-*[0-9]*)"</code>
Enteros	7, 6, 8, -4	<code>#rx"\-*[0-9]+"</code>
Comentarios	// esto es un comentario, // desarrollar funcion 3 + 5	<code>#rx"\\/\\.*"</code>
Strings	"esto es una cadena 123 + 567", "+ - * /"	<code>#rx "\".*\""</code>
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, static, while, do, int, struct_Packed, double	<code>#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 \\static \\while \\do \\int \\struct_Packed \\double"</code>
Operadores	\\^, /, &, +, -, *, =, (,)	<code>#rx"\\^ \\ \\& \\ \\+ \\ - * \\ = \\(\\)"</code>

La solución se enfoca en detectar los tokens y reconociendo su tipo se agrega a un string envuelto en una tag de html llamada span, a la cual se le agrega una clase determinada por su misma token la cual le agrega 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 que se define el header que incluye los estilos anteriores se desarrolló una función lexer que recibe una cadena de texto, el nombre del archivo de entrada, lo que se realiza en esta función es sumar o integrar un string generado por la llamada de la función `analyze_row`, que recibe una lista de cadenas de texto con el contenido de cada renglón del archivo de entrada, la función usa una expresión regular que se encarga de detectar todos los tokens que están presentes en cada línea ordenada debido a una expresión regular que se encarga de detectar todos los tipos de tokens identificados, para esto se utiliza la función `regex-match` que forma parte de `analyzeRow`, la cual detecta todos los tokens en una cadena, y posteriormente imprime los tokens en los que el regex ha provocado el match, con esto lo que devuelve el algoritmo es un archivo html, donde todos los tokens detectados son retornados como etiquetas de `span` y dichas etiquetas tienen su respectiva clase, la cual por medio de CSS se le asocia 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.

De esta forma se obtiene un análisis léxico en paralelo para los archivos, en el que el proceso de un archivo muy pesado para la máquina no bloqueará el análisis del resto de archivos. Lo que se hizo fue usar el algoritmo de la solución monohilo, llamado una función en cada hilo que recibe el nombre del archivo que dicho thread debe analizar. Posteriormente se espera a que todos estos hilos terminen sus tareas y se deja de tomar el tiempo. La ventaja principal de esta implementación es que un archivo muy pesado no bloquea al resto de procesos, el programa es capaz de resolver distintas tareas de forma simultánea. Supongamos que ingresamos 11 archivos y 1 de ellos tiene un tamaño muy grande. En el monohilo el programa resolvería cada tarea de forma secuencial, usando varios hilos todos los archivos pequeños no deberían esperar al proceso del más grande para ser analizados.

Complejidad

La complejidad obtenida de esta solución es de $O(N * M * L)$ similar a la versión previamente realizada en monohilo, ya que en la versión multihilo, L representa la longitud del archivo, es decir, la cantidad de líneas que contiene el archivo y N representa la cantidad

de tokens a buscar al igual que la cantidad de expresiones regulares utilizadas, y M es la cantidad de matches a buscar. 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 N patrones, es decir la cantidad total de expresiones regulares usadas, por L líneas de caracteres.

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. Con la distribución de los procesos en múltiples hilos las personas serán capaces de aprovechar al máximo las capacidades del procesador de su equipo, al mejorar los procesos de sus ordenadores serán más productivos y las tareas de su día a día se ven hechas más simples, y los tiempos de la realización de dichas tareas están mejor optimizados, teniendo un mejor desempeño y desarrollo en ámbitos profesionales o de ocio, mejorando de manera significativa la calidad de vida, generando oportunidades para tener mejores herramientas que establezcan mejores resultados, incluso en segmentos de la industria tecnológica donde los tiempos de realización de procesos son vitales, la mejora otorgada por los algoritmos de procesamiento multihilo puede ser fundamental para el desarrollo de estos sectores que tienen que ver con toma de decisiones.