INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

TEORÍA DE LA COMPUTACIÓN

ING. EN SISTEMAS COMPUTACIONALES

Práctica 5: Análisis sintáctico de un código fuente (Continuación)

Integrantes:

Espinoza León Jaqueline Viridiana

García Zacarías Angel Emmanuel

Hurtado Morales Emiliano

Profesora: Luz María Sánchez García

Fecha de entrega: 18/05/22

Ciclo Escolar: 2022 - 1

Índice

1. Planteamiento del problema

Se diseñará un programa en lenguaje C, capaz de leer un archivo y detectar los siguientes tokens:

- Dígitos
  - Enteros [0 a 9]+.
  - Decimales [0 a 9]+.[0 a 9]+.
- Palabras reservadas
- Comentarios
  - Sencillos //
  - Varias líneas /* */
- Cantidad total de tokens

2. Actividades

1. Se diseñó un programa en lenguaje C, capaz de leer un archivo y detectar los siguientes tokens:

- Dígitos
  - Enteros [0 a 9]+.
  - Decimales [0 a 9]+.[0 a 9]+.
- Palabras reservadas
- Comentarios
  - Sencillos //
  - Varias líneas /* */
- Cantidad total de tokens

2. El programa es capaz de leer un archivo fuente, además de detectar los tokens antes mencionados, y mostrar de qué tipo es cada uno.

3. Mostramos los resultados obtenidos.

```
Numeros.
Hay 312 enteros
Hay 3 decimales
```

```
Comentarios.

Hay 84 comentarios sencillos
Hay 18 comentarios de varias lineas
```

```
 Reservadas
do se repite 9 veces
else se repite 55 veces
float se repite 3 veces
int se repite 54 veces
return se repite 6 veces
sizeof se repite 4 veces
struct se repite 17 veces
and se repite 8 veces
char se repite 52 veces
for se repite 10 veces
if se repite 65 veces
or se repite 82 veces
signed se repite 10 veces
unsigned se repite 5 veces
void se repite 11 veces
while se repite 8 veces
Total de tokens: 8316
```

4. Finalmente responda a las siguientes preguntas

a) ¿Ya había trabajado con tokens en otras unidades de aprendizaje?

Anteriormente, solo en la materia de algoritmos y estructuras de datos se trabajó con un código capaz de detectar el cierre correcto de paréntesis y corchetes, es decir, el número de paréntesis o corchetes abiertos debía ser el mismo que el número de paréntesis o corchetes cerrados.

b) ¿Fue fácil trabajar con tokens?¿Por qué?

Al principio fue algo complicado, ya que no sabíamos bien cómo deberíamos abordar el problema. Sin embargo una vez que se tuvo una idea clara, el proceso fue más sencillo.

c) ¿Qué aplicaciones puede tener su programa?

El programa puede ayudar a verificar si se está comentando apropiadamente cada función y descripción del programa. Asimismo, puede aportar a la detección rápida de las variables que se están empleando y mejorar la estética del código.

d) ¿Cuáles errores se pudieron resolver y cuáles no?

Se logró que corriera correctamente la parte de la detección de los nombres de las variables, debido a que se paraba después de una cierta cantidad de avistamiento. Sin embargo, no se consiguió que no se repitieran algunos nombres.

De igual forma, se detectaron correctamente los comentarios, al poner sus if's antes que la detección de la diagonal normal, debido a que si se ponía después, no se lograba detectar ningún comentario.

e) ¿Qué otros tokens agregaría para que nuevos equipos realicen las prácticas?

Consideramos que los tokens que se trabajaron durante estas prácticas son más que suficientes para comprender correctamente un analizador léxico, además de que se nos es complicado pensar en más tokens a agregar.

Por otro lado, podría ser buena idea agregar una parte, donde se detecte si los pares de paréntesis, corchetes o llaves se cierran correctamente.

3. Pruebas

1.- Programación de detección de los caracteres requeridos.

a) Detección de Dígitos y decimales.

```
// Enteros y decimales
else if(arreglo[i] >= 48 && arreglo[i] <= 57)
{
    while((arreglo[i] >= 48 && arreglo[i] <= 57) || arreglo[i] == '.')
    {
        if(arreglo[i] == '.')
            f_dec = 1;
        i++;
    }
    if(f_dec == 1)
        frecuencias[34]++;
    else
        frecuencias[33]++;
    f_dec = 0;
    i--;
}

printf("\nNumeros. \n");
printf("Hay %i enteros \n", frecuencias[33]);
printf("Hay %i decimales \n", frecuencias[34]);
```

b) Detección de palabras reservadas.

```
void contarRes(char *arreglo, char *res, struct stat sb, int pos, int *sum){
    // variables
    int i;
    int cont;
    int lon;

    lon = 0;
    cont = 1;
    // recorrido
    for(i = pos; i<sb.st_size;i++){
        //recorremos
        if(lon == strlen(res)){ // Son iguales por su longitud

            if(arreglo[i] == '  ' || arreglo[i] == ' ' || arreglo[i] == '\n' || arreglo[i] == '<' ||
arreglo[i] == '*' || arreglo[i] == '+' || arreglo[i] == '-'|| arreglo[i] == '['|| arreglo[i] ==
'{'|| arreglo[i] == '('|| arreglo[i] == ')'|| arreglo[i] == ';'|| arreglo[i] == ',' || arreglo[i] ==
'"') // revisamos el siguiente valor
                cont++;
            else lon = 0; //reseteamos la longitud de la palabra
        } else{
            if(res[lon] == arreglo[i]){ // Si son iguales, su tamaño coincide
                lon++;
            } else lon = 0;
        }
    }
    if(cont > 1){
        printf("%s se repite %d veces\n",res,cont);
        sum[0] += cont;
    }
}
```

c) Detección de comentarios

```
//Comentarios
else if(arreglo[i] == '/' && arreglo[i+1] == '/' && arreglo[i+2] != '/')
    frecuencias[35]++;

else if(arreglo[i] == '/' && arreglo[i+1] == '*')
    frecuencias[36]++;

//Comentarios
printf("\n\n\nComentarios. \n\n");
printf("Hay %i comentarios sencillos\n", frecuencias[35] );
printf("Hay %i comentarios de varias lineas\n", frecuencias[36]);
```

2.- Lectura de archivo fuente

```
void entradaDatos(char *nombreArchivo, char *arreglo, struct stat sb){
    char bufer[1];
    FILE *archivo;
    size_t bytesLeidos;
    int i;

    i = 0;

    archivo = fopen(nombreArchivo, "rb"); // Abrir en modo read binario
    // Si es NULL, entonces no existe, o no se pudo abrir
    if (!archivo) {
        printf("¡No se pudo abrir el archivo %s!", nombreArchivo);
    }

    // Mientras no alcancemos el EndOfLine del archivo...
    while (i<sb.st_size) {
        // Leer dentro del búfer; fread regresa el número de bytes leídos
        bytesLeidos = fread(bufer, sizeof(char), sizeof(bufer), archivo);
        arreglo[i] = bufer[0];
        i++;

    }
    // Al final, se cierra el archivo
    fclose(archivo);
}
```

Para ejecutar el programa, se debe escribir el nombre del archivo a leer al lado del .exe y este debe estar en el mismo directorio.

3.- Salida en pantalla de los resultados

```
Frecuencia de cada caracter.

Caracteres.
El caracter ( se repite 193 veces
El caracter ) se repite 194 veces
El caracter [ se repite 257 veces
El caracter ] se repite 256 veces
El caracter { se repite 138 veces
El caracter } se repite 136 veces
El caracter < se repite 27 veces
El caracter > se repite 13 veces

Operadores logicos.
El operador && se repite 30 veces
El operador || se repite 48 veces
El operador ! se repite 17 veces

Operadores aritmeticos.
El operador ++ se repite 57 veces
El operador + se repite 83 veces
El operador -- se repite 2 veces
El operador - se repite 9 veces
El operador * se repite 1007 veces
El operador / se repite 310 veces
El operador % se repite 50 veces
El operador == se repite 113 veces
El operador = se repite 201 veces
El operador <= se repite 9 veces
El operador >= se repite 9 veces

Puntuadores.
El puntuador , se repite 246 veces
El puntuador ; se repite 226 veces
El puntuador : se repite 24 veces
El puntuador ... se repite 2 veces
El puntuador # se repite 11 veces
El puntuador ^ se repite 2 veces
El puntuador & se repite 65 veces
El puntuador | se repite 100 veces
El puntuador ~ se repite 2 veces
El puntuador " se repite 300 veces
El puntuador ? se repite 6 veces

Numeros.
Hay 312 enteros
Hay 3 decimales
```

```
Frecuencia de cada par.

El par de caracteres () aparece 193
El par de caracteres {} aparece 256
El par de caracteres [] aparece 136
El par de caracteres <> aparece 13



Comentarios.

Hay 84 comentarios sencillos
Hay 18 comentarios de varias lineas



   Identificadores

   argv se repite 2 veces
   nombreArchivo se repite 14 veces
   arreglo se repite 154 veces
   numC se repite 1 veces
   alt se repite 1 veces
   i se repite 693 veces
   k se repite 1 veces
   sum se repite 28 veces
   tamor se repite 4 veces
   tamcom se repite 1 veces
   reser se repite 2 veces
   bufer se repite 7 veces
   i se repite 680 veces
   frecuencias se repite 96 veces
   i se repite 667 veces
   f_dec se repite 11 veces
   i se repite 369 veces
   cont se repite 24 veces
   lon se repite 44 veces
   i se repite 280 veces
   identificador se repite 13 veces
   size se repite 26 veces
   i se repite 139 veces
   band se repite 21 veces
   bandTip se repite 17 veces
   j se repite 16 veces
   tams se repite 2 veces
   tipos se repite 2 veces
   i se repite 84 veces
   cont se repite 14 veces
   lon se repite 22 veces
```

```
 Reservadas
do se repite 9 veces
else se repite 55 veces
float se repite 3 veces
int se repite 54 veces
return se repite 6 veces
sizeof se repite 4 veces
struct se repite 17 veces
and se repite 8 veces
char se repite 52 veces
for se repite 10 veces
if se repite 65 veces
or se repite 82 veces
signed se repite 10 veces
unsigned se repite 5 veces
void se repite 11 veces
while se repite 8 veces
Total de tokens: 8316
```

4. Anexos

Instrucciones de compilación.

Estando en el símbolo de sistema para desarrolladores (Windows) o en la terminal (Linux), se escribe en la línea de comando lo siguiente:

- Windows
    - Compilación: cl Practica5.c
    - Ejecución: Practica5.c codigoFuente.c
- Linux
    - Compilación: gcc Practica5.c
    - Ejecución: ./Practica5.c codigoFuente.c

Código fuente

- cola.h

```c
#ifndef cola_h
#define cola_h

typedef struct NodoC{
   char dato;
   struct NodoC *sig;
}*ApNodo;

typedef struct Cnodo{
   ApNodo prim;
   ApNodo ulti;
}*Cola;

Cola nueva(){
   Cola t = (Cola)malloc(sizeof(struct Cnodo));
   t->prim=t->ulti=NULL;
   return t;
}

int esnueva(Cola q){return ((q->prim==NULL)&&(q->ulti==NULL));}

char primero(Cola q){return q->prim->dato;}

Cola formar(Cola q, char e){
   ApNodo t = (ApNodo)malloc(sizeof(struct NodoC));
   t->dato=e;
   t->sig=NULL;
   if(esnueva(q)){
      q->prim=q->ulti=t;
   } else{
      q->ulti->sig=t;
      q->ulti=t;
   }
   return q;
}
```

```
Cola desformar(Cola q){
   ApNodo t;
   if(q->prim==q->ulti){
      free(q->prim);
      free(q);
      return nueva( );
   } else{
      t = q->prim;
      q->prim=q->prim->sig;
      free(t);
      return q;
   }
}

int comparar(Cola q, char *cadena, int size, int pos){
   if(esnueva(q) && cadena[pos] == '\0'){
      return 1;
   } else if(esnueva(q) && cadena[pos] != '\0'){
      return 0;
   } else if(!esnueva(q) && cadena[pos] == '\0'){
      return 0;
   } else if(primero(q) != cadena[pos]){
      return 0;
   } else if(primero(q) == cadena[pos]){
      return comparar(desformar(q),cadena,size,pos + 1);
   }
}
#endif
```

- Practica5.c

```
/*
NOMBRE DE PROGRAMA: Análisis sintáctico
DESCRIPCIÓN: Partiendo de cualquier tipo de archivo, se lee de forma binaria y se guarda cada byte
en un arreglo del tamaño del archivo. Posteriormente, se hace un análisis del arreglo. De esta forma,
se consigue la frecuencia de cada byte y se presenta sólo la
información de los carácteres necesarios.
FECHA: mayo 2022
VERSIÓN: 2.0
AUTOR(ES):
    Espinoza León Jaqueline Viridiana
    García Zacarías Angel Emmanuel
    Hurtado Morales Emiliano
*/

//*****************************************************************
//LIBRERIAS INCLUIDAS
//*****************************************************************

#include <stdio.h>// Todas las funciones como fread, fwrite, fopen, fclose y printf
#include <stdlib.h>// EXIT_FAILURE y EXIT_SUCCESS
#include <inttypes.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include "cola.h"

#define TAM 100
```

```
//*****************************************************************************
//DECLARACION DE ESTRUCTURAS
//*****************************************************************************

//*************************************************************
//DECLARACIÓN DE FUNCIONES
//*************************************************************
void entradaDatos(char*, char*, struct stat);
void analisisLexico(char*, struct stat, int*);
void encontrarTipo(char*, struct stat, int*);
int validar(char*, struct stat, int, int*);
void contarIden(char*, char*, struct stat, int, int*);
void contarRes(char*, char*, struct stat, int, int*);
```

```
//*************************************************************
//PROGRAMA PRINCIPAL
//*************************************************************

/*Descripcion de la función:
Funcion main, se ingresa el archivo a analizar y realiza todas las llamadas a funciones que permiten
su análisis léxico.
Input: Archivo de entrada.
Output: Ninguno.
*/
```

```
int main(int argc, char *argv[]) {

    //*****************************************************************
    //Variables del main
    //*****************************************************************

    FILE *archivo;
    char *nombreArchivo;
    char *arreglo;
    int *numC;
    int alt;
    int i;
    int k;
    int *sum;
    float tamor;
    float tamcom;
        char reser[74][17] = {{"asm\0"},{"auto\0"},{"bool\0"},{"catch\0"},{"compl\0"},{"continue\0"},{"do
\0"},{"else\0"},{"export\0"},{"float\0"},{"goto\0"},{"int\0"},{"namespace\0"},{"or_eq\0"},
{"public\0"},{"return\0"},{"sizeof\0"},{"struct\0"},{"this\0"},{"try\0"},{"typename\0"},{"not_eq
\0"},{"using\0"},{"volatile\0"},{"and\0"},{"bitand\0"},{"break\0"},{"char\0"},{"const\0"},
{"default\0"},{"double\0"},{"enum\0"},{"extern\0"},{"for\0"},{"if\0"},{"long\0"},{"new\0"},{"operator
\0"},{"private\0"},{"register\0"},{"short\0"},{"static\0"},{"switch\0"},{"throw\0"},
{"typedef\0"},{"union\0"},{"virtual\0"},{"wchar_t\0"},{"xor\0"},{"and_eq\0"},{"bitor\0"},{"case
\0"},{"class\0"},{"const_cast\0"},{"delete\0"},{"dynamic_cast\0"},{"explicit\0"},{"false\0"},
{"friend\0"},{"inline\0"},{"mutable\0"},{"not\0"},{"or\0"},{"protected\0"},{"reinterpret_cast
\0"},{"signed\0"},{"static_cast\0"},{"template\0"},{"true\0"},{"typeid\0"},{"unsigned\0"},
{"void\0"},{"xor_eq\0"},{"while\0"}};

    nombreArchivo = argv[1];

    //*****************************************************************
    //Algoritmo
    //*****************************************************************

    // Checa el tamaño del archivo y manda un error si no logra realizarlo
    struct stat sb;
    if (stat(nombreArchivo, &sb) == -1){
        perror("stat");
        exit(EXIT_FAILURE);
    }
    tamor = sb.st_size;

    // Se crea el arreglo dinámico
    arreglo = malloc(sizeof(char)*sb.st_size);
```

```
    entradaDatos(nombreArchivo, arreglo, sb);

    sum[0] = 0;

    analisisLexico(arreglo, sb,sum);

    printf("\n\n\n");
    printf("   Identificadores\n\n");

    encontrarTipo(arreglo,sb, sum);

    printf("\n\n\n Reservadas\n");
    for(i = 0; i<74; i++){
        contarRes(arreglo,reser[i],sb,0,sum);
    }

    printf("Total de tokens: %i \n",sum[0]);

    return EXIT_SUCCESS;
}
```

```
//*********************************************************************
//DEFINICIÓN DE FUNCIONES
//*********************************************************************

// ////////////////////////////////////////////////////////////////////////////////////////////

/*Descripción de función:
Funcion entradaDatos, permite la inserción de cada byte del archivo en el arreglo previamente creado.
Input: Puntero al archivo, puntero al arreglo y struct stat sb, que habilita saber de que tamaño es
el archivo.
Output: Arreglo llenado completamente.
Observaciones: La inserción es muy rápida y no conlleva
mucho tiempo de procesamiento.
*/

void entradaDatos(char *nombreArchivo, char *arreglo, struct stat sb){
    char bufer[1];
    FILE *archivo;
    size_t bytesLeidos;
    int i;

    i = 0;

    archivo = fopen(nombreArchivo, "rb"); // Abrir en modo read binario
    // Si es NULL, entonces no existe, o no se pudo abrir
    if (!archivo) {
        printf("¡No se pudo abrir el archivo %s!", nombreArchivo);
    }

    // Mientras no alcancemos el EndOfLine del archivo...
    while (i<sb.st_size) {
        // Leer dentro del búfer; fread regresa el número de bytes leídos
        bytesLeidos = fread(bufer, sizeof(char), sizeof(bufer), archivo);
        arreglo[i] = bufer[0];
        i++;

    }
    // Al final, se cierra el archivo
    fclose(archivo);
}

////////////////////////////////////////////////////////////////////////////////////////////
```

```
/*Descripción de función:
Funcion creacionLista, se realiza todo el análisis necesario para poder crear una lista
ordenada,
al observar que no se repitan caracteres dentro de ella y que se sepa cuantas veces
aparece cada
byte en el archivo.
Input: Puntero al struct lista, puntero al arreglo y struct stat sb, que habilita saber
de que
tamaño es el archivo.
Output: Lista creada y ordenada.
*/

void analisisLexico(char *arreglo, struct stat sb, int *sum){
    //Variables
    int frecuencias[TAM];
    int i;
    // Banderas para enteros y decimales
    int f_dec = 0;

    //Estadarización de las frecuencias
    for(i = 0 ; i < TAM ; i++)
        frecuencias[i] = 0;
```

```
    //Análisis
    for(i = 0 ; i < sb.st_size ; i++)
    {
        if(arreglo[i] == '(')
            frecuencias[0]++;
        else if(arreglo[i] == ')')
            frecuencias[1]++;
        else if(arreglo[i] == '[')
            frecuencias[2]++;
        else if(arreglo[i] == ']')
            frecuencias[3]++;
        else if(arreglo[i] == '{')
            frecuencias[4]++;
        else if(arreglo[i] == '}')
            frecuencias[5]++;
        // Agregados de operadores aritméticos
        else if(arreglo[i] == '<' && arreglo[i+1] == '=')
            frecuencias[20]++;
        else if(arreglo[i] == '>' && arreglo[i+1] == '=')
            frecuencias[21]++;
        ////////////////////////////////////////////////
        else if(arreglo[i] == '<')
            frecuencias[6]++;
        else if(arreglo[i] == '>')
            frecuencias[7]++;
        // Operadores lógicos: &&, ||, ! (3)
        else if(arreglo[i] == '&' && arreglo[i+1] == '&')
            frecuencias[8]++;
        else if(arreglo[i] == '|' && arreglo[i+1] == '|')
            frecuencias[9]++;
        else if(arreglo[i] == '!')
            frecuencias[10]++;
```

```
// Operadores aritméticos: (11)
else if(arreglo[i] == '+' && arreglo[i+1] == '+')
    frecuencias[11]++;
else if(arreglo[i] == '+')
    frecuencias[12]++;
else if(arreglo[i] == '-' && arreglo[i+1] == '-')
    frecuencias[13]++;
else if(arreglo[i] == '-')
    frecuencias[14]++;
//Comentarios
else if(arreglo[i] == '/' && arreglo[i+1] == '/' && arreglo[i+2] != '/')
    frecuencias[35]++;

else if(arreglo[i] == '/' && arreglo[i+1] == '*')
    frecuencias[36]++;
```

```
else if(arreglo[i] == '*')
    frecuencias[15]++;
else if(arreglo[i] == '/')
    frecuencias[16]++;
else if(arreglo[i] == '%')
    frecuencias[17]++;
else if(arreglo[i] == '=' && arreglo[i+1] == '=')
    frecuencias[18]++;
else if(arreglo[i] == '=')
    frecuencias[19]++;
// Puntuadores
else if(arreglo[i] == ',')
    frecuencias[22]++;
else if(arreglo[i] == ';')
    frecuencias[23]++;
else if(arreglo[i] == ':')
    frecuencias[24]++;
else if(arreglo[i] == '.' && arreglo[i+1] == '.'  && arreglo[i+2] == '.')
    frecuencias[25]++;
else if(arreglo[i] == '#')
    frecuencias[26]++;
else if(arreglo[i] == '^')
    frecuencias[27]++;
else if(arreglo[i] == '&')
    frecuencias[28]++;
else if(arreglo[i] == '|')
    frecuencias[29]++;
else if(arreglo[i] == '~')
    frecuencias[30]++;
else if(arreglo[i] == '"')
    frecuencias[31]++;
else if(arreglo[i] == '?')
    frecuencias[32]++;
```

```
        //else if(arreglo[i] == '.')
            //frecuencias[31]++;
        // Enteros y decimales
        else if(arreglo[i] >= 48 && arreglo[i] <= 57)
        {
            while((arreglo[i] >= 48 && arreglo[i] <= 57) || arreglo[i] == '.')
            {
                if(arreglo[i] == '.')
                    f_dec = 1;
                i++;
            }
            if(f_dec == 1)
                frecuencias[34]++;
            else
                frecuencias[33]++;
            f_dec = 0;
            i--;
        }

    }
```

```
    printf("\nFrecuencia de cada caracter. \n\n");
    printf("Caracteres. \n");
    printf("El caracter ( se repite %i veces \n", frecuencias[0]);
    printf("El caracter ) se repite %i veces \n", frecuencias[1]);
    printf("El caracter [ se repite %i veces \n", frecuencias[2]);
    printf("El caracter ] se repite %i veces \n", frecuencias[3]);
    printf("El caracter { se repite %i veces \n", frecuencias[4]);
    printf("El caracter } se repite %i veces \n", frecuencias[5]);
    printf("El caracter < se repite %i veces \n", frecuencias[6]);
    printf("El caracter > se repite %i veces \n", frecuencias[7]);
    printf("\nOperadores logicos. \n");
    printf("El operador && se repite %i veces \n", frecuencias[8]);
    printf("El operador || se repite %i veces \n", frecuencias[9]);
    printf("El operador ! se repite %i veces \n", frecuencias[10]);
    printf("\nOperadores aritmeticos. \n");
    printf("El operador ++ se repite %i veces \n", frecuencias[11]);
    printf("El operador + se repite %i veces \n", frecuencias[12]);
    printf("El operador -- se repite %i veces \n", frecuencias[13]);
    printf("El operador - se repite %i veces \n", frecuencias[14]);
    printf("El operador * se repite %i veces \n", frecuencias[15]);
    printf("El operador / se repite %i veces \n", frecuencias[16]);
    printf("El operador %c se repite %i veces \n", 37, frecuencias[17]);
    printf("El operador == se repite %i veces \n", frecuencias[18]);
    printf("El operador = se repite %i veces \n", frecuencias[19]);
    printf("El operador <= se repite %i veces \n", frecuencias[20]);
    printf("El operador >= se repite %i veces \n", frecuencias[21]);
```

```
    printf("\nPuntuadores. \n");
    printf("El puntuador , se repite %i veces \n", frecuencias[22]);
    printf("El puntuador ; se repite %i veces \n", frecuencias[23]);
    printf("El puntuador : se repite %i veces \n", frecuencias[24]);
    printf("El puntuador ... se repite %i veces \n", frecuencias[25]);
    printf("El puntuador # se repite %i veces \n", frecuencias[26]);
    printf("El puntuador ^ se repite %i veces \n", frecuencias[27]);
    printf("El puntuador & se repite %i veces \n", frecuencias[28]+frecuencias[8]);
    printf("El puntuador | se repite %i veces \n", frecuencias[29]+frecuencias[9]);
    printf("El puntuador ~ se repite %i veces \n", frecuencias[30]);
    printf("El puntuador %c se repite %i veces \n", 34, frecuencias[31]);
    printf("El puntuador ? se repite %i veces \n", frecuencias[32]);
```

```
    printf("\nNumeros. \n");
    printf("Hay %i enteros \n", frecuencias[33]);
    printf("Hay %i decimales \n", frecuencias[34]);
    //2.45 345.32
    printf("\n\n\nFrecuencia de cada par. \n\n");
    printf("El par de caracteres () aparece %i\n", frecuencias[0] < frecuencias[1] ?
frecuencias[0] : frecuencias[1]);

    printf("El par de caracteres {} aparece %i\n", frecuencias[2] < frecuencias[3] ?
frecuencias[2] : frecuencias[3]);

    printf("El par de caracteres [] aparece %i\n", frecuencias[4] < frecuencias[5] ?
frecuencias[4] : frecuencias[5]);

    printf("El par de caracteres <> aparece %i\n", frecuencias[6] < frecuencias[7] ?
frecuencias[6] : frecuencias[7]);

    //Comentarios
    printf("\n\n\nComentarios. \n\n");
    printf("Hay %i comentarios sencillos\n", frecuencias[35] );
    printf("Hay %i comentarios de varias lineas\n", frecuencias[36]);
    for(i = 0; i<37; i++){
        sum[0] += frecuencias[i];
    }
}
```

```
void contarIden(char *arreglo, char *identificador, struct stat sb, int pos, int *sum){
    // variables
    int i;
    int cont;
    int lon;

    lon = 0;
    cont = 1;

    // recorrido
    for(i = pos; i<sb.st_size;i++){
        //recorremos
        if(lon == strlen(identificador)){ // Son iguales por su longitud
            if(arreglo[i] == '  ' || arreglo[i] == ' ' || arreglo[i] == '\n' ||
arreglo[i] == '=' || arreglo[i] == '>' || arreglo[i] == '<' || arreglo[i] == '*' ||
arreglo[i] == '+' || arreglo[i]
== '-'|| arreglo[i] == '{'|| arreglo[i] == '['|| arreglo[i] == ']'|| arreglo[i] == ')'||
arreglo[i] == ';'|| arreglo[i] == ',') // revisamos el siguiente valor
                cont++;
            else lon = 0; //reseteamos la longitud de la palabra
```

```
        } else{
            if(identificador[lon] == arreglo[i]){ // Si son iguales, su tamaño coincide
                lon++;
            } else lon = 0;
        }
    }
    sum[0] += cont;
    printf("   %s se repite %d veces\n",identificador,cont);
}
```

```
int validar(char *arreglo, struct stat sb, int pos, int *sum){
    // Variables
    int i;
    char identificador[100];
    int size;

    identificador[0] = '\0';
    size = 0;
    //Continuamos recorrido
    for(i = pos; i<sb.st_size;i++){
        if(arreglo[i] >= 97 && arreglo[i] <= 122 && identificador[0] == '\0'){   //Inicia
con minuscula
            identificador[0] = arreglo[i];
            size++;
        } else if(arreglo[i] == '_' && (arreglo[i+1] >= 97 && arreglo[i+1] <= 122) &&
identificador[0] == '\0'){ // Inicia con '_' y deepués una minuscula
            identificador[0] = arreglo[i];
            size++;
        } else if(identificador[0] != '\0' && ((arreglo[i] >= 48 && arreglo[i] <= 57) ||
(arreglo[i] >= 65 && arreglo[i] <= 90) || (arreglo[i] >= 97 && arreglo[i] <= 122) ||
arreglo[i] == '_')){
            identificador[size] = arreglo[i]; // Se agrega a la cadena
            size++;
        }else if(arreglo[i] == '(' ||arreglo[i] == ')' || arreglo[i] == ']' ||arreglo[i]
== ',' ){ // Estos valores resetean todo
            return pos;
        } else if(identificador[0] != '\0' && (arreglo[i] == ' ' || arreglo[i] == ' ' ||
arreglo[i] == '\n' || arreglo[i] == ';'|| arreglo[i] == ',' || arreglo[i] == '[' )){
//Posible valores que
acompañan una variable
            identificador[size] = '\0';
            contarIden(arreglo, identificador, sb, i, sum); // Se cuentan identificadores
            return i;
        } else if(identificador[0] == '\0' && !(arreglo[i] == ' ' || arreglo[i] == ' ' ||
arreglo[i] == '\n' || arreglo[i] == '*' )){ //Cualquier valor no mencionado al inicio
genera un error
            return pos;
        }
    }

    return 0;
}
```

```
void encontrarTipo(char *arreglo, struct stat sb, int *sum){
    //Variables
    int i;
    int band;
    int bandTip;
    int j;
    int tams[14] = {5,14,12,6,15,13,4,13,11,5,14,12,6,7};
```

```
    int tams[14] = {5,14,12,6,15,13,4,13,11,5,14,12,6,7};
    char tipos[14][15] = {{"char\0"},{"unsigned char\0"},{"signed char\0"},{"short
\0"},{"unsigned short\0"},
                {"signed short\0"},{"int\0"},{"unsigned int\0"},{"signed int\0"},{"long
\0"},{"unsigned long\0"},
                {"signed long\0"},{"float\0"},{"double\0"}};
    Cola queue;

    band = 0;
    bandTip = 0;
    queue = nueva();

    //Análisis
    for(i = 0 ; i < sb.st_size ; i++){

        if(band == 1 || !esnueva(queue)){ //Revisamos que band == 1 y que la cola no este
vacia
            if(arreglo[i] >= 97 && arreglo[i] <= 122){ // Solo minusculas
                queue = formar(queue,arreglo[i]);
            } else if(arreglo[i] != ' ' && arreglo[i] != '  ' && arreglo[i] != '\n'){
//Cualquier caracter menos estos genera un error
                while(!esnueva(queue))
                    queue = desformar(queue);
            } else if(!esnueva(queue)){ // Se analiza si es vacia
                if(comparar(queue,"unsigned\0",9,0) == 1 ||
comparar(queue,"signed\0",7,0) == 1){ // Puede ser signed o unsigned
                    queue = formar(queue,arreglo[i]);
                } else{ // No es, por lo tanto continuamos
                    for(j = 0;j<14;j++){
                        if(comparar(queue,tipos[j],tams[j],0) == 1) { //Vemos si
coindice, en caso de que si bandTip = 1;
                            bandTip = 1;
                            j = 14;
                        }
                    }

                    if(bandTip == 1){
                        i = validar(arreglo,sb,i+1, sum); // Se retorna la posición final
                        band = 0; // Se resetean banderas
                        bandTip = 0;
                        while(!esnueva(queue)) // Se elimina la queue
                            queue = desformar(queue);
                    } else {
                        while(!esnueva(queue))
                            queue = desformar(queue);
                    }
                }
            }
        }

        if(arreglo[i] == ' '|| arreglo[i] == '  ' || arreglo[i] == '\n') //Después de
estos valores puede iniciar un identificador
            band = 1;
        else band = 0;
    }
}
```

```
void contarRes(char *arreglo, char *res, struct stat sb, int pos, int *sum){
    // variables
    int i;
    int cont;
    int lon;
```

```
    lon = 0;
    cont = 1;
    // recorrido
    for(i = pos; i<sb.st_size;i++){
        //recorremos
        if(lon == strlen(res)){ // Son iguales por su longitud

            if(arreglo[i] == '  ' || arreglo[i] == ' ' || arreglo[i] == '\n' ||
arreglo[i] == '<' || arreglo[i] == '*' || arreglo[i] == '+' || arreglo[i] == '-'||
arreglo[i] == '['|| arreglo[i] ==
'{'|| arreglo[i] == '('|| arreglo[i] == ')'|| arreglo[i] == ';'|| arreglo[i] == ',' ||
arreglo[i] == '"') // revisamos el siguiente valor
                cont++;
            else lon = 0; //reseteamos la longitud de la palabra
        } else{
            if(res[lon] == arreglo[i]){ // Si son iguales, su tamaño coincide
                lon++;
            } else lon = 0;
        }
    }
    if(cont > 1){
        printf("%s se repite %d veces\n",res,cont);
        sum[0] += cont;
    }
}
```

5. Bibliografía

Tokens. (2016). Zator.com. Recuperado el 17 de mayo de 2022, de https://www.zator.com/Cpp/E3_2.htm