INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

TEORÍA DE LA COMPUTACIÓN

ING. EN SISTEMAS COMPUTACIONALES

Práctica 4: Análisis sintáctico de un código fuente

Integrantes:

Espinoza León Jaqueline Viridiana

García Zacarías Angel Emmanuel

Hurtado Morales Emiliano

Profesora: Luz María Sánchez García

Fecha de entrega: 06/05/22

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:

- Identificadores.
- Operadores
- Signos de puntuación

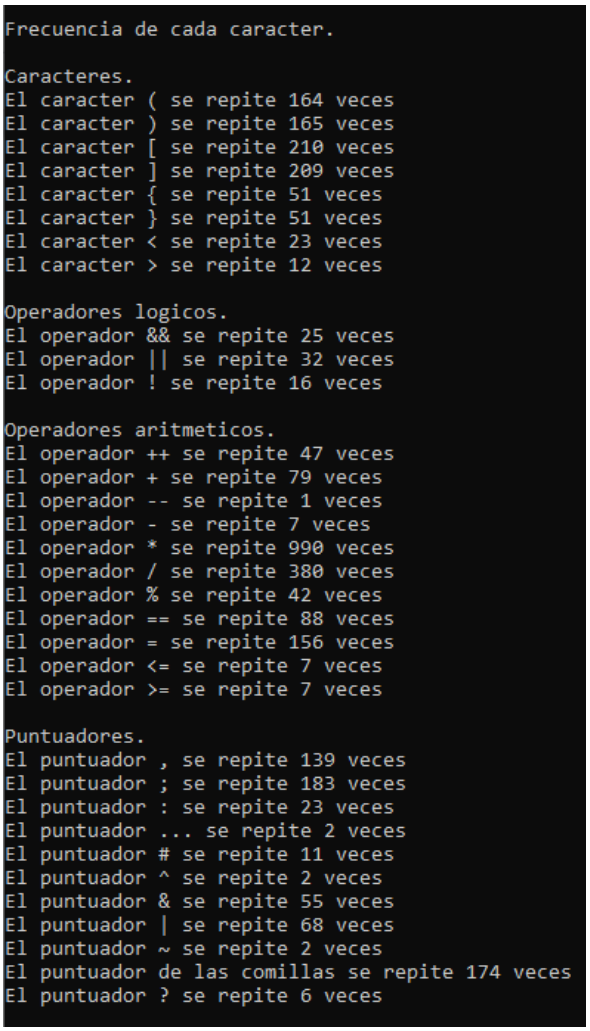
2. Actividades

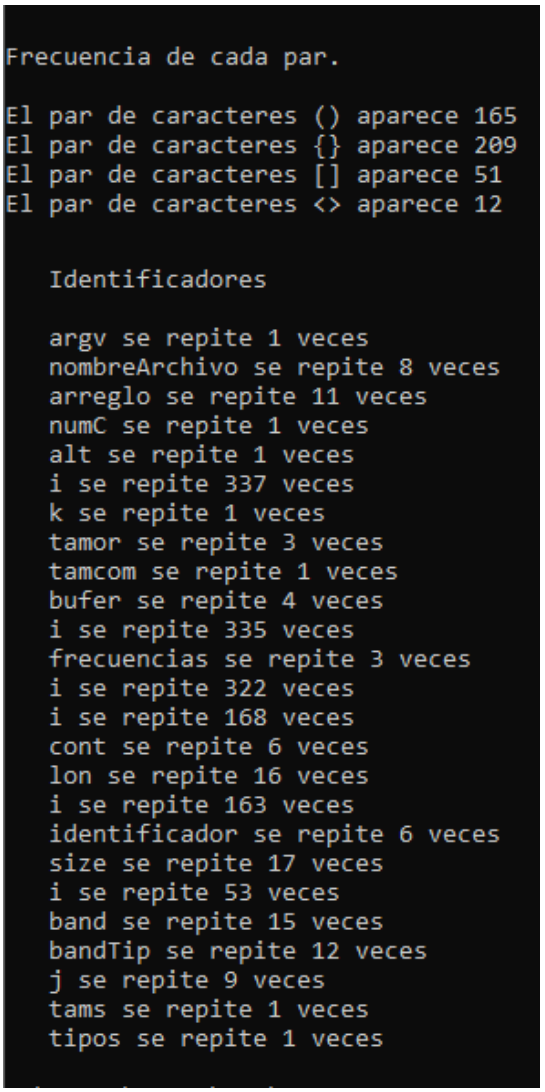
1. Se diseñó un programa en lenguaje C, capaz de leer un archivo y detectar los siguientes tokens:

- Identificadores.
- Operadores
- Signos de puntuación

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

3. Mostramos los resultados obtenidos.





4. Finalmente responda a las siguientes preguntas

a) **¿Cuál es la utilidad de analizar los tokens?**
   Nos sirve para llevar un control del uso de estos tokens, además de que podemos evitar errores al identificar los distintos tokens, ya que muchas veces pueden utilizarse de una manera errónea..

b) **¿Fue fácil trabajar con tokens?¿Por qué?**
   No, ya que cada token tiene unas reglas preestablecidas para que funcione de manera correcta, esto genera que tengamos que establecer varias cosas en el código, siendo esto muy complejo si es que seguimos todo lo establecido.

c) **¿Qué aplicaciones puede tener su programa?**
   Por el momento podemos utilizar el programa para identificar si es que tenemos algunos errores de sintaxis en nuestros códigos, aunque este programa aún le falta mucho para que pueda funcionar de esta manera y que a la vez sea eficiente.

d) **De los errores detectados, ¿se pudieron resolver? ¿cuáles sí y cuáles no?**
   Un problema era que en algunos momentos el programa dejaba de ejecutarse en la búsqueda de identificadores, al final lo pudimos resolver utilizando una estructura de datos tipo FIFO (una Queue).

e) **¿Qué recomendaciones daría a nuevos equipos para realizar la práctica?**
   Manejar de forma correcta los archivos, usar memoria dinámica de manera correcta y si es posible, saber cómo utilizar estructuras de datos como las Queues..

3. Pruebas

1.- Programación de detección de los caracteres requeridos.

a) Detección de operadores y signos de puntuación.

```
void analisisLexico(char *arreglo, struct stat sb){
    //Variables
    int frecuencias[TAM];
    int i;

    //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]++;
        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]++;
    }
```

```
    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 de las comillas se repite %i veces \n", frecuencias[31]);
    printf("El puntuador ? se repite %i veces \n", frecuencias[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]);
}
```

b) Detección de identificadores.

```
void encontrarTipo(char *arreglo, struct stat sb){
    //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};
    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); // 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;
    }
}
```

```
int validar(char *arreglo, struct stat sb, int pos){
    // 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); // 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 contarIden(char *arreglo, char *identificador, struct stat sb, int pos){
    // 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+1] == '    ' || arreglo[i+1] == ' ' || arreglo[i+1] ==
'\n' || arreglo[i+1] == '=' || arreglo[i+1] == '>' || arreglo[i+1] == '<' ||
arreglo[i+1] == '*' || arreglo[i+1] == '+' || arreglo[i+1] == '-'|| arreglo[i+1]
== '['|| arreglo[i+1] == ')'|| arreglo[i+1] == ';'|| arreglo[i+1] == ',') //
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;
        }
    }

    printf("   %s se repite %d veces\n",identificador,cont);
}
```

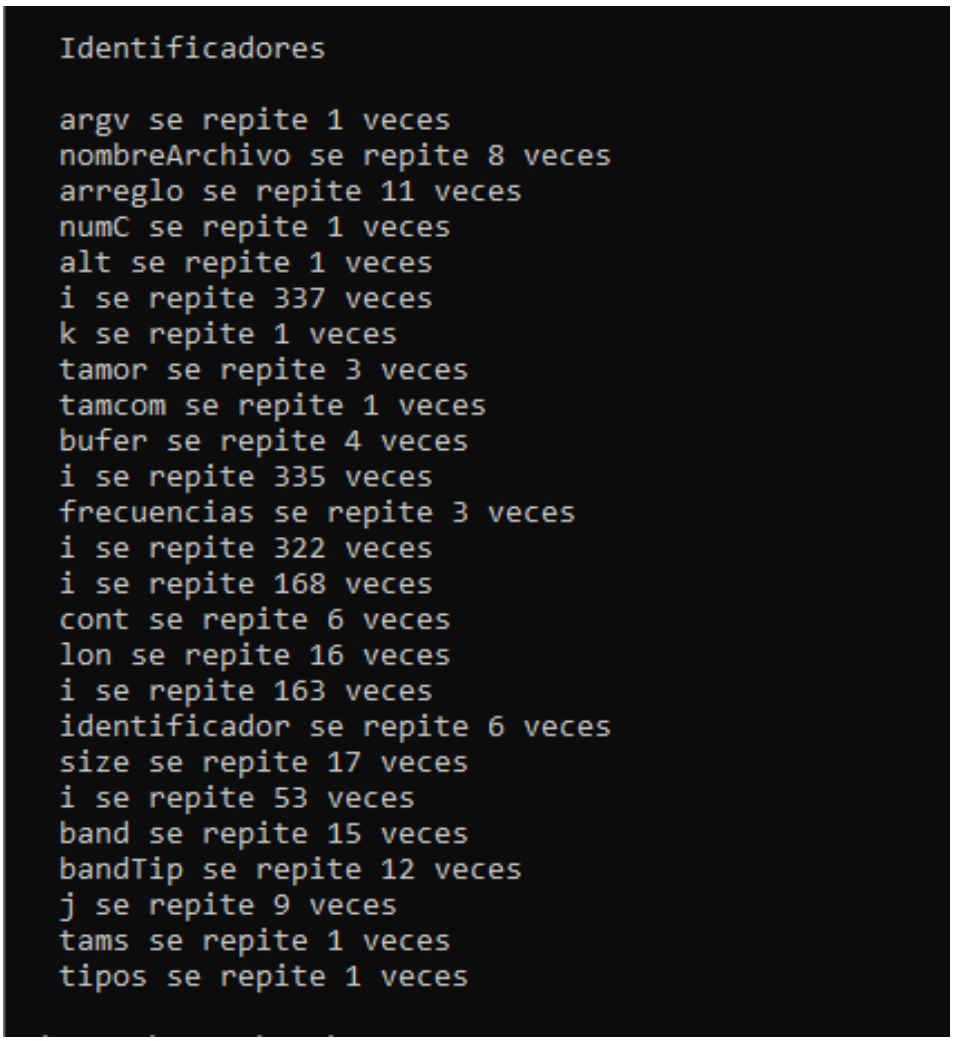
2.- Lectura de archivo fuente

```
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

```
C:\Users\escom\Desktop>gcc p4.c -o p4.exe

C:\Users\escom\Desktop> p4.exe p4.c

Frecuencia de cada caracter.

Caracteres.
El caracter ( se repite 164 veces
El caracter ) se repite 165 veces
El caracter [ se repite 210 veces
El caracter ] se repite 209 veces
El caracter { se repite 51 veces
El caracter } se repite 51 veces
El caracter < se repite 23 veces
El caracter > se repite 12 veces

Operadores logicos.
El operador && se repite 25 veces
El operador || se repite 32 veces
El operador ! se repite 16 veces

Operadores aritmeticos.
El operador ++ se repite 47 veces
El operador + se repite 79 veces
El operador -- se repite 1 veces
El operador - se repite 7 veces
El operador * se repite 990 veces
El operador / se repite 380 veces
El operador % se repite 42 veces
El operador == se repite 88 veces
El operador = se repite 156 veces
El operador <= se repite 7 veces
El operador >= se repite 7 veces

Puntuadores.
El puntuador , se repite 139 veces
El puntuador ; se repite 183 veces
El puntuador : se repite 23 veces
El puntuador ... se repite 2 veces
El puntuador # se repite 11 veces
El puntuador ^ se repite 2 veces
El puntuador & se repite 55 veces
El puntuador | se repite 68 veces
El puntuador ~ se repite 2 veces
El puntuador de las comillas se repite 174 veces
El puntuador ? se repite 6 veces


Frecuencia de cada par.

El par de caracteres () aparece 165
El par de caracteres {} aparece 209
El par de caracteres [] aparece 51
El par de caracteres <> aparece 12
```



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 Practica4.c
    - Ejecución: Practica4.c codigoFuente.c

- Linux
    - Compilación: gcc Practica4.c
    - Ejecución: ./Practica4.c codigoFuente.c

Código fuente

- cola.h

```
#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));}

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;
}

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

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
```

- p4.c

```c
/*
NOMBRE DE PROGRAMA: Análisis léxico
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, para obtener
una lista ordenada. De esta forma, se consigue la frecuencia de cada byte y se presenta sólo la
información de los carácteres necesarios.
FECHA: abril 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);
void encontrarTipo(char*, struct stat);
int validar(char*, struct stat, int);
void contarIden(char*, char*, struct stat, int);
```

```c
//*****************************************************************
//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;
    float tamor;
    float tamcom;

    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);

    analisisLexico(arreglo, sb);

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

    encontrarTipo(arreglo,sb);

    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){
    //Variables
    int frecuencias[TAM];
    int i;

    //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]++;
        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]++;
    }
```

```
    }
    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 de las comillas se repite %i veces \n", frecuencias[31]);
    printf("El puntuador ? se repite %i veces \n", frecuencias[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]);
}

void contarIden(char *arreglo, char *identificador, struct stat sb, int pos){
```

```
] void contarIden(char *arreglo, char *identificador, struct stat sb, int pos){
    // 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+1] == '    ' || arreglo[i+1] == ' ' || arreglo[i+1] == '\n' || arreglo[i+1] == '='
            || arreglo[i+1] == '>'|| arreglo[i+1] == '<' || arreglo[i+1] == '*' || arreglo[i+1] == '+' || arreglo[i+1] == '-'
            || arreglo[i+1] == '['|| arreglo[i+1] == ')' || arreglo[i+1] == ';'|| arreglo[i+1] == ',')
            // 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;
-       }
-   }

    printf("   %s se repite %d veces\n",identificador,cont);
- }
```

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

    identificador[0] = '\0';
    size = 0;
    //Continuamos recorrido
    for(i = pos; i<sb.st_size;i++){
        //Inicia con minuscula
        if(arreglo[i] >= 97 && arreglo[i] <= 122 && identificador[0] == '\0'){
            identificador[0] = arreglo[i];
            size++;
            // Inicia con '_' y deepués una minuscula
        } else if(arreglo[i] == '_' && (arreglo[i+1] >= 97 && arreglo[i+1] <= 122) && identificador[0] == '\0'){
            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); // Se cuentan identificadores
            return i;
            //Cualquier valor no mencionado al inicio genera un error
        } else if(identificador[0] == '\0' && !(arreglo[i] == ' ' || arreglo[i] == ' ' || arreglo[i] == '\n' || arreglo[i] == '*' )){
            return pos;
        }
    }

    return 0;
}
```

```
void encontrarTipo(char *arreglo, struct stat sb){
    //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};
    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); // 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;
    }
}
```

5. Bibliografía

Tokens. (2016). Zator.com. Recuperado el 26 de abril de 2022, de https://www.zator.com/Cpp/E3_2.htm