



**INSTITUTO POLITÉCNICO NACIONAL**  
**ESCUELA SUPERIOR DE CÓMPUTO**  
**TEORÍA DE LA COMPUTACIÓN**  
**ING. EN SISTEMAS COMPUTACIONALES**

**Práctica 3: Análisis léxico de un código fuente**

**Integrantes:**

Espinoza León Jaqueline Viridiana

García Zacarías Angel Emmanuel

Hurtado Morales Emiliano

Profesora: Luz María Sánchez García

Fecha de entrega: 13/04/22

Ciclo Escolar: 2022 - 1

# Índice

<b>Planteamiento del problema</b>	<b>2</b>
<b>Actividades</b>	<b>2</b>
<b>Pruebas</b>	<b>3</b>
1.- Programación de detección de los caracteres requeridos.	3
2.- Lectura de archivo fuente, detección de caracteres y muestra de pares	5
3.- Salida en pantalla de los resultados	6
<b>Anexos</b>	<b>7</b>
<b>Bibliografía</b>	<b>14</b>

## 1. Planteamiento del problema

Se diseñará un programa en lenguaje C, capaz de leer un archivo y detectar los siguientes caracteres, además de mencionar cuántos hay de cada par:

- Paréntesis ( )
- Llaves { }
- Corchetes [ ]
- Paréntesis triangulares < >

## 2. Actividades

1. Se realizó el programa en C, capaz de detectar los siguientes caracteres:

- Paréntesis ( )
- Llaves { }
- Corchetes [ ]
- Paréntesis triangulares < >

2. El programa es capaz de leer un archivo fuente, además de detectar los caracteres antes mencionados, y finalmente, es capaz de mencionar la cantidad de pares que podemos encontrar..

3. Mostramos los resultados obtenidos.

```
Frecuencia de cada caracter.  
  
El caracter [ se repite 12 veces  
El caracter ] se repite 12 veces  
El caracter < se repite 19 veces  
El caracter { se repite 42 veces  
El caracter } se repite 42 veces  
El caracter ( se repite 72 veces  
El caracter ) se repite 72 veces  
El caracter > se repite 121 veces  
  
Frecuencia de cada par.  
  
El par de caracteres ( ) aparece 72  
El par de caracteres { } aparece 42  
El par de caracteres [ ] aparece 12  
El par de caracteres < > aparece 19
```

5. Finalmente responda a las siguientes preguntas

a) **¿Cuál es la utilidad de analizar símbolos?**

Nos sirve de muchas maneras, un buen ejemplo sería para conocer si estamos cumpliendo con estructuras que están previamente determinadas.

b) **¿Fue fácil trabajar con archivos? ¿Por qué?**

Si, aunque lo más complicado era poder manejarlos de tal manera que se mantuviera la información de forma correcta.

c) **¿Qué aplicaciones puede tener su programa?**

Una buena manera de aplicarlo, sería realizando un analizador de códigos con el propósito de determinar si no tenemos llaves, paréntesis o corchetes de más. Otra manera de aplicarlo sería para aplicaciones relacionadas con las matemáticas, más que nada para mantener un orden y establecer de forma correcta la jerarquía de operaciones.

**d) De los errores detectados, ¿se pudieron resolver? ¿cuáles sí y cuáles no?**

El principal problema que teníamos, era que no podíamos leer archivos con muchos saltos de línea, al final fue posible resolverlo.

**e) ¿Qué recomendaciones daría a nuevos equipos para realizar la práctica?**

Conocer cómo manejar archivos en el lenguaje que estén utilizando.

### 3. Pruebas

#### 1.- Programación de detección de los caracteres requeridos.

```
283 //////////////////////////////////////////////////
284
285 /*Descripción de función:
286 Funcion frecuenciaDatos, Imprime Los caracteres requeridos, su frecuencia y el número de pares.
287 Input: Puntero al archivo, puntero a la lista y struct stat sb, que habilita saber de que tamaño
288 es el archivo.
289 Output: Impresión de información.
290 */
291
292 void frecuenciaDatos(FILE* archivo, list* lista, struct stat sb)
293 {
294     huf *actual;
295     int aux1 = 0, aux2 = 0, aux3 = 0, aux4 = 0;
296
297     // Mostrar la lista
298     actual = lista -> inicio;
299     if(actual == NULL)
300         printf("\nLa lista esta vacia\n");
301     else
302     {
303         printf("\nFrecuencia de cada caracter. \n\n");
304         while(actual != lista -> fin)
305         {
306             if(actual -> car == '(' || actual -> car == ')')
307             {
308                 if(actual -> car == '(')
309                 {
310                     aux1 = actual -> f;
311                 }
312                 else if(actual -> car == ')' && actual -> f < aux1)
313                 {
314                     aux1 = actual -> f;
315                 }
316                 printf("El caracter %c se repite %i veces \n", actual -> car, actual -> f);
317             }
318
319             else if(actual -> car == '{' || actual -> car == '}')
320             {
321                 if(actual -> car == '{')
322                 {
323                     aux2 = actual -> f;
324                 }
325                 else if(actual -> car == '}' && actual -> f < aux2)
326                 {
327                     aux2 = actual -> f;
328                 }
329                 printf("El caracter %c se repite %i veces \n", actual -> car, actual -> f);
330             }
331             else if(actual -> car == '[' || actual -> car == ']')
332             {
333                 // ...
334             }
335         }
336     }
337 }
```

```

333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
}

if(actual -> car == '[')
{
    aux3 = actual -> f;
}
else if(actual -> car == ']' && actual -> f < aux3)
{
    aux3 = actual -> f;
}
printf("El caracter %c se repite %i veces \n", actual -> car, actual -> f);

}
else if(actual -> car == '<' || actual -> car == '>')
{
    if(actual -> car == '<')
    {
        aux4 = actual -> f;
    }
    else if(actual -> car == '>' && actual -> f < aux4)
    {
        aux4 = actual -> f;
    }
    printf("El caracter %c se repite %i veces \n", actual -> car, actual -> f);
}
actual = actual -> siguiente;
}

printf("\n\nFrecuencia de cada par. \n\n");
printf("El par de caracteres () aparece %i\n", aux1);
printf("El par de caracteres {} aparece %i\n", aux2);
printf("El par de caracteres [] aparece %i\n", aux3);
printf("El par de caracteres <> aparece %i\n", aux4);
}

```

## 2.- Lectura de archivo fuente, detección de caracteres y muestra de pares

### Programación de lectura de archivo

```
67  /*Descripción de la función:
68  Funcion main, se ingresa el archivo a analizar y realiza todas las llamadas a funciones que permiten
69  su análisis léxico.
70  Input: Archivo de entrada.
71  Output: Ninguno.
72  */
73
74  int main(int argc, char *argv[]) {
75
76      //*****
77      //Variables del main
78      //*****
79
80      FILE *archivo;
81      char *nombreArchivo = argv[1];
82      char *arreglo;
83      int *numC;
84      int alt, i, k;
85      list *lista;
86      float tamor, tamcom;
87
88      //*****
89
90      //*****
91
92      //*****
93
94      //*****
95
96      //*****
97
98      //*****
99
100     //*****
101
102     //*****
103
104     //*****
105
106     //*****
107
108     //*****
109
110     //*****
111
112     //*****
113
114     //*****
115
116     //*****
117
118     //*****
119
120     //*****
121
122     //*****
123
124     //*****
125
126     /*Descripción de función:
127     Funcion entradaDatos, permite la inserción de cada byte del archivo en el arreglo previamente creado.
128     Input: Puntero al archivo, puntero al arreglo y struct stat sb, que habilita saber de que tamaño es
129     el archivo.
130     Output: Arreglo llenado completamente.
131     Observaciones: La inserción es muy rápida y no conlleva
132     mucho tiempo de procesamiento.
133     */
134
135     void entradaDatos(char *nombreArchivo, char *arreglo, struct stat sb){
136         char bufer[1];
137         FILE *archivo;
138         size_t bytesLeidos;
139         int i=0;
140
141         archivo = fopen(nombreArchivo, "rb"); // Abrir en modo read binario
142         // Si es NULL, entonces no existe, o no se pudo abrir
143         if (!archivo) {
144             printf("No se pudo abrir el archivo %s!", nombreArchivo);
145         }
146
147         // Mientras no alcancemos el EndOfLine del archivo...
148         while (i<sb.st_size) {
149             // Leer dentro del búfer; fread regresa el número de bytes leídos
150             bytesLeidos = fread(bufer, sizeof(char), sizeof(bufer), archivo);
151             arreglo[i] = bufer[0];
152             i++;
153         }
154
155         // Al final, se cierra el archivo
156         fclose(archivo);
157     }
158 }
```

Lectura de archivo fuente, contabilización de caracteres y muestra de cantidad de pares

Se debe escribir el nombre del archivo al lado del .exe y este debe estar en el mismo directorio.

```
C:\Users\cafe_\Documents\4_Semestre\Teoria_de_la_Computacion\Escandalosos-TC\Practica_3>Practica3.exe Practica3.c

Frecuencia de cada caracter.

El caracter [ se repite 12 veces
El caracter ] se repite 12 veces
El caracter < se repite 19 veces
El caracter { se repite 42 veces
El caracter } se repite 42 veces
El caracter ( se repite 72 veces
El caracter ) se repite 72 veces
El caracter > se repite 121 veces

Frecuencia de cada par.

El par de caracteres () aparece 72
El par de caracteres {} aparece 42
El par de caracteres [] aparece 12
El par de caracteres <> aparece 19
```

### 3.- Salida en pantalla de los resultados

```
Frecuencia de cada caracter.

El caracter [ se repite 12 veces
El caracter ] se repite 12 veces
El caracter < se repite 19 veces
El caracter { se repite 42 veces
El caracter } se repite 42 veces
El caracter ( se repite 72 veces
El caracter ) se repite 72 veces
El caracter > se repite 121 veces

Frecuencia de cada par.

El par de caracteres () aparece 72
El par de caracteres {} aparece 42
El par de caracteres [] aparece 12
El par de caracteres <> aparece 19
```



## 4. Anexos

```
1  /*
2  NOMBRE DE PROGRAMA: Análisis Léxico
3  DESCRIPCIÓN: Partiendo de cualquier tipo de archivo, se lee de forma binaria y se guarda cada byte
4  en un arreglo del tamaño del archivo. Posteriormente, se hace un análisis del arreglo, para obtener
5  una lista ordenada. De esta forma, se consigue la frecuencia de cada byte y se presenta sólo la
6  información de los caracteres necesarios.
7  FECHA: abril 2022
8  VERSIÓN: 2.0
9  AUTOR(ES):
10     Espinoza León Jaqueline Viridiana
11     García Zacarías Angel Emmanuel
12     Hurtado Morales Emiliano
13  */
14
15  //*****
16  //LIBRERIAS INCLUIDAS
17  //*****
18
19  #include <stdio.h> // Todas las funciones como fread, fwrite, fopen, fclose y printf
20  #include <stdlib.h> // EXIT_FAILURE y EXIT_SUCCESS
21  #include <inttypes.h>
22  #include <unistd.h>
23  #include <sys/stat.h>
24  #include <fcntl.h>
25  #include <string.h>
26
27  //*****
28  //DECLARACION DE ESTRUCTURAS
29  //*****
30
31  /*Descripción de estructura: huf
32  Estructura que simula un nodo.
33  Cada nodo, posee el byte deseado y las veces que se repite dentro del archivo original. Asimismo,
34  tiene los apuntadores siguiente y anterior para crear la lista.
35  */
36
37  typedef struct nodo{
38     char car;
39     int f;
40     // Apuntadores para la lista
41     struct nodo *siguiente;
42     struct nodo *anterior;
43 }huf;
44
45  /*Descripción de estructura: ListaDoble
46  Estructura que tiene los apuntadores al principio y fin de la cola.
47  */
48
49  typedef struct ListaDoble{
50     huf *inicio;
51     huf *fin;
52     int tam;
```

```

53 }list;
54
55 //*****
56 //DECLARACIÓN DE FUNCIONES
57 //*****
58
59 void entradaDatos(char*, char*, struct stat);
60 void creacionLista(list*, char*, struct stat);
61 void frecuenciaDatos(FILE*, list*, struct stat);
62
63 //*****
64 //PROGRAMA PRINCIPAL
65 //*****
66
67 /*Descripción de la función:
68 Función main, se ingresa el archivo a analizar y realiza todas las llamadas a funciones que permiten
69 su análisis léxico.
70 Input: Archivo de entrada.
71 Output: Ninguno.
72 */
73
74 int main(int argc, char *argv[]) {
75
76     //*****
77     //Variables del main
78     //*****
79
80     FILE *archivo;
81     char *nombreArchivo = argv[1];
82     char *arreglo;
83     int *numC;
84     int alt, i, k;
85     list *lista;
86     float tamor, tamcom;
87
88     //*****
89     //Algoritmo
90     //*****
91
92     // Checa el tamaño del archivo y manda un error si no logra realizarlo
93     struct stat sb;
94     if (stat(nombreArchivo, &sb) == -1){
95         perror("stat");
96         exit(EXIT_FAILURE);
97     }
98     tamor = sb.st_size;
99
100     // Se crea el arreglo dinámico
101     arreglo = malloc(sizeof(char)*sb.st_size);
102
103     // Inicializa memoria lista

```

```

104     lista = (list *)malloc(sizeof(list));
105
106     // Inicializa estructura
107     lista -> inicio = NULL;
108     lista -> fin = NULL;
109     lista -> tam = 0;
110
111     entradaDatos(nombreArchivo, arreglo, sb);
112
113     creacionLista(lista, arreglo, sb);
114
115     frecuenciaDatos(archivo, lista, sb);
116
117     return EXIT_SUCCESS;
118 }
119
120 //*****
121 //DEFINICIÓN DE FUNCIONES
122 //*****
123
124 //////////////////////////////////////
125
126 /*Descripción de función:
127 Funcion entradaDatos, permite la inserción de cada byte del archivo en el arreglo previamente creado.
128 Input: Puntero al archivo, puntero al arreglo y struct stat sb, que habilita saber de que tamaño es
129 el archivo.
130 Output: Arreglo llenado completamente.
131 Observaciones: La inserción es muy rápida y no conlleva
132 mucho tiempo de procesamiento.
133 */
134
135 void entradaDatos(char *nombreArchivo, char *arreglo, struct stat sb){
136     char bufer[1];
137     FILE *archivo;
138     size_t bytesLeídos;
139     int i=0;
140
141     archivo = fopen(nombreArchivo, "rb"); // Abrir en modo read binario
142     // Si es NULL, entonces no existe, o no se pudo abrir
143     if (!archivo) {
144         printf("No se pudo abrir el archivo %s!", nombreArchivo);
145     }
146
147     // Mientras no alcancemos el EndOfLine del archivo...
148     while (i<sb.st_size) {
149         // Leer dentro del búfer; fread regresa el número de bytes leídos
150         bytesLeídos = fread(bufer, sizeof(char), sizeof(bufer), archivo);
151         arreglo[i] = bufer[0];
152         i++;
153     }
154 }

```

```

155 // Al final, se cierra el archivo
156 fclose(archivo);
157 }
158
159 ///////////////////////////////////////////////////
160
161 /*Descripción de función:
162 Funcion creacionLista, se realiza todo el análisis necesario para poder crear una lista ordenada,
163 al observar que no se repitan caracteres dentro de ella y que se sepa cuantas veces aparece cada
164 byte en el archivo.
165 Input: Puntero al struct lista, puntero al arreglo y struct stat sb, que habilita saber de que
166 tamaño es el archivo.
167 Output: Lista creada y ordenada.
168 */
169
170 void creacionLista(list *lista, char *arreglo, struct stat sb){
171 //Variables lista
172 int i=0, j, f, a;
173 int bandera;
174 huf *actual;
175 huf *recorrer;
176 huf *aux;
177
178 // Crea la lista a partir del análisis del arreglo
179 while(i<sb.st_size){
180     bandera = 0;
181
182     // Se revisa que no se repita el caracter en la lista
183     actual = lista -> inicio;
184     while(actual != NULL){
185         if(actual -> car == arreglo[i]){
186             bandera = 1;
187             break;
188         }
189         actual = actual -> siguiente;
190     }
191
192     // Revisa que no esté repetido un caracter en la lista
193     if(bandera != 1){
194
195         // Se mide la frecuencia del caracter dentro del arreglo
196         f=0;
197         for(j=i ; j<sb.st_size ; j++){
198             if(arreglo[i] == arreglo[j])
199                 f++;
200
201         // Se crea el nodo y se insertan los valores
202         actual = (huf*)malloc(sizeof(huf));
203         actual -> car = arreglo[i];
204         actual -> f = f;
205     }

```

```

206 // Se aumenta el tamaño de la lista
207 lista -> tam = lista -> tam + 1;
208
209 // Creando nodo lista -> inicio
210 if(lista -> inicio == NULL){
211     actual -> siguiente = NULL;
212     actual -> anterior = NULL;
213     lista -> inicio = actual;
214 }
215 else{
216     if( lista -> fin == NULL){
217         //creando nodo lista -> fin
218         if(actual -> f >= lista -> inicio -> f){
219             //nodo actual sera nuestro lista -> fin
220             //si la actual -> f es mas pequeña que el lista -> inicio lo manda a la
221             //derecha
222             actual -> siguiente = NULL;
223             actual -> anterior = lista -> inicio ;
224             lista -> inicio -> siguiente = actual;
225             lista -> fin = actual;
226         }
227         else{
228             //nodo auxiliar sera el lista -> inicio y nuestro anterior erior lista ->
229             //inicio el lista -> fin
230             actual -> siguiente = lista -> inicio ;
231             actual -> anterior = NULL;
232             lista -> inicio -> anterior = actual;
233             lista -> fin = lista -> inicio ;
234             lista -> inicio = actual;
235         }
236     }
237     else{
238         if(actual -> f >= lista -> fin -> f){
239             //nodo actual al lista -> fin de la lista
240             //si la actual -> f es mas pequeña que el lista -> inicio lo manda a la
241             //derecha
242             actual -> siguiente = NULL;
243             actual -> anterior = lista -> fin ;
244             lista -> fin -> siguiente = actual;
245             lista -> fin = actual;
246         }
247         else{
248             if(actual -> f <= lista -> inicio -> f){
249                 //nodo actual al incio de la lista
250                 actual -> siguiente = lista -> inicio ;
251                 lista -> inicio -> anterior = actual;
252                 actual -> anterior = NULL;
253                 lista -> inicio = actual;
254             }
255             else{
256                 //acomodar el actual en medio de nuestra lista recorriendo

```

```

257 //nuestro nodo desde el lista -> inicio
258 recorrer = lista -> inicio ;
259 a = 0;
260 while(a!=1)
261 {
262     //aux sera nuestro nodo anterior al que esta nuestro nodo recorrer
263     aux = recorrer;
264     recorrer = recorrer -> siguiente ;
265     if(actual -> f < recorrer -> f)
266     {
267         actual -> siguiente = recorrer;
268         recorrer -> anterior = actual;
269         actual -> anterior = aux ;
270         aux -> siguiente = actual;
271         a = 1;
272     }
273 }
274 }
275 }
276 }
277 }
278 }
279 i++;
280 }
281 }
282
283 ///////////////////////////////////////////////////
284
285 /*Descripción de función:
286 Funcion frecuenciaDatos, Imprime los caracteres requeridos, su frecuencia y el número de pares.
287 Input: Puntero al archivo, puntero a la lista y struct stat sb, que habilita saber de que tamaño
288 es el archivo.
289 Output: Impresión de información.
290 */
291
292 void frecuenciaDatos(FILE* archivo, list* lista, struct stat sb)
293 {
294     huf *actual;
295     int aux1 = 0, aux2 = 0, aux3 = 0, aux4 = 0;
296
297     // Mostrar la lista
298     actual = lista -> inicio;
299     if(actual == NULL)
300         printf("\nLa lista esta vacia\n");
301     else
302     {
303         printf("\nFrecuencia de cada caracter. \n\n");
304         while(actual != lista -> fin)
305         {
306             if(actual -> car == '(' || actual -> car == ')')
307

```

```

308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365

```

```

    if(actual -> car == '(')
    {
        aux1 = actual -> f;
    }
    else if(actual -> car == ')' && actual -> f < aux1)
    {
        aux1 = actual -> f;
    }
    printf("El caracter %c se repite %i veces \n", actual -> car, actual -> f);

}
else if(actual -> car == '{' || actual -> car == '}')
{
    if(actual -> car == '{')
    {
        aux2 = actual -> f;
    }
    else if(actual -> car == '}' && actual -> f < aux2)
    {
        aux2 = actual -> f;
    }
    printf("El caracter %c se repite %i veces \n", actual -> car, actual -> f);
}
else if(actual -> car == '[' || actual -> car == ']')
{
    if(actual -> car == '[')
    {
        aux3 = actual -> f;
    }
    else if(actual -> car == ']' && actual -> f < aux3)
    {
        aux3 = actual -> f;
    }
    printf("El caracter %c se repite %i veces \n", actual -> car, actual -> f);
}
else if(actual -> car == '<' || actual -> car == '>')
{
    if(actual -> car == '<')
    {
        aux4 = actual -> f;
    }
    else if(actual -> car == '>' && actual -> f < aux4)
    {
        aux4 = actual -> f;
    }
    printf("El caracter %c se repite %i veces \n", actual -> car, actual -> f);
}
actual = actual -> siguiente;
}
printf("\n\nFrecuencia de cada par. \n\n");

printf("El par de caracteres () aparece %i\n", aux1);
printf("El par de caracteres {} aparece %i\n", aux2);
printf("El par de caracteres [] aparece %i\n", aux3);
printf("El par de caracteres <> aparece %i\n", aux4);
}

```

## **5. Bibliografía**

Debido a que todo fue realizado por el equipo, no se tomaron referencias externas.