INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

TEORÍA DE LA COMPUTACIÓN

ING. EN SISTEMAS COMPUTACIONALES

Práctica 7: Analizador de Texto (segunda parte)

Integrantes:

Espinoza León Jaqueline Viridiana

García Zacarías Angel Emmanuel

Hurtado Morales Emiliano

Profesora: Luz María Sánchez García

Fecha de entrega: 10/06/22

Ciclo Escolar: 2022 - 1

Índice

1. Planteamiento del problema

Se diseñó un programa en lenguaje C, capaz de leer un archivo y detectar cadenas.

Después de eso, indica la frecuencia con la que se repite cada uno y mostrará el porcentaje que ocupa en el archivo.

2. Actividades

1. Se programó en ANSI C, el número de veces que aparece una cadena y el porcentaje en el que el carácter ocupa en el texto.
2. El programa es capaz de leer un archivo fuente, además de detectar cadenas, y mostrar su frecuencia y su porcentaje.
3. Mostramos los resultados obtenidos.

```
asm se repite una vez, es un 0.0433 %
auto se repite una vez, es un 0.0578 %
bool se repite una vez, es un 0.0578 %
catch se repite una vez, es un 0.0722 %
compl se repite una vez, es un 0.0722 %
continue se repite una vez, es un 0.1156 %
do se repite 7 veces, es un 0.2023 %
else se repite 5 veces, es un 0.2890 %
export se repite una vez, es un 0.0867 %
float se repite 13 veces, es un 0.9392 %
goto se repite una vez, es un 0.0578 %
int se repite 17 veces, es un 0.7369 %
namespace se repite una vez, es un 0.1300 %
or_eq se repite una vez, es un 0.0722 %
public se repite una vez, es un 0.0867 %
return se repite 4 veces, es un 0.3468 %
sizeof se repite 4 veces, es un 0.3468 %
struct se repite 11 veces, es un 0.9536 %
this se repite una vez, es un 0.0578 %
try se repite una vez, es un 0.0433 %
typename se repite una vez, es un 0.1156 %
not_eq se repite una vez, es un 0.0867 %
using se repite una vez, es un 0.0722 %
volatile se repite una vez, es un 0.1156 %
and se repite una vez, es un 0.0433 %
bitand se repite una vez, es un 0.0867 %
break se repite una vez, es un 0.0722 %
char se repite 28 veces, es un 1.6183 %
const se repite una vez, es un 0.0722 %
default se repite una vez, es un 0.1011 %
double se repite una vez, es un 0.0867 %
enum se repite una vez, es un 0.0578 %
extern se repite una vez, es un 0.0867 %
for se repite 3 veces, es un 0.1300 %
if se repite 9 veces, es un 0.2601 %
long se repite una vez, es un 0.0578 %
new se repite una vez, es un 0.0433 %
operator se repite una vez, es un 0.1156 %
private se repite una vez, es un 0.1011 %
register se repite una vez, es un 0.1156 %
short se repite una vez, es un 0.0722 %
static se repite una vez, es un 0.0867 %
```

```
switch se repite una vez, es un 0.0867 %
throw se repite una vez, es un 0.0722 %
typedef se repite una vez, es un 0.1011 %
union se repite una vez, es un 0.0722 %
virtual se repite una vez, es un 0.1011 %
wchar_t se repite una vez, es un 0.1011 %
xor se repite una vez, es un 0.0433 %
and_eq se repite una vez, es un 0.0867 %
bitor se repite una vez, es un 0.0722 %
case se repite una vez, es un 0.0578 %
class se repite una vez, es un 0.0722 %
```

```
const_cast se repite una vez, es un 0.1445 %
delete se repite una vez, es un 0.0867 %
dynamic_cast se repite una vez, es un 0.1734 %
explicit se repite una vez, es un 0.1156 %
false se repite una vez, es un 0.0722 %
friend se repite una vez, es un 0.0867 %
inline se repite una vez, es un 0.0867 %
mutable se repite una vez, es un 0.1011 %
not se repite una vez, es un 0.0433 %
or se repite 10 veces, es un 0.2890 %
protected se repite una vez, es un 0.1300 %
reinterpret_cast se repite una vez, es un 0.2312 %
signed se repite una vez, es un 0.0867 %
static_cast se repite una vez, es un 0.1589 %
template se repite una vez, es un 0.1156 %
true se repite una vez, es un 0.0578 %
typeid se repite una vez, es un 0.0867 %
unsigned se repite una vez, es un 0.1156 %
void se repite 5 veces, es un 0.2890 %
xor_eq se repite una vez, es un 0.0867 %
while se repite 3 veces, es un 0.2167 %

Hay 74 cadenas.
```

4. Finalmente responda a las siguientes preguntas

a) **¿Es fácil o difícil trabajar con cadenas ?¿Por qué?**

Es complejo trabajar con cadenas, porque implica una concatenación de carácteres que puede llegar a ser difícil de manejar. Esto implica que se empleen apuntadores y estructuras de datos como colas, listas, entre otros. Es por ello que es muy distinto trabajar con caracteres solitarios que con un conjunto ordenado de ellos.

b) **¿Qué errores se presentaron al realizar la práctica?¿Se pudieron resolver?**

El programa dejaba de funcionar al tener que realizar una gran cantidad de ciclos, la manera de solucionar esto fue dejar de lado los arreglos para reemplazarlos por *queues*, ya que estas no nos generaban estos inconvenientes.

c) **¿En qué aplicaciones ha visto se ocupa la detección de cadenas con su porcentaje de ocurrencia?**

Lo más cercano a eso sería la herramienta de búsqueda de algunos IDE, Editores de Código, Procesadores de Texto, etc.

d) **¿En qué otros lenguajes se pueden manejar las cadenas con facilidad?**

C++ y Python plantean muchas facilidades para el manejo de cadenas, puesto que ya tiene varias funciones o métodos predefinidos que sólo se requiere ingresar un argumento para que de todo lo demás se encargue la máquina.

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

Empezar a aprender nuevos lenguajes de programación de alto nivel, de forma que puedan hacer uso de ellos en estás prácticas y logren facilitar el desarrollo de la misma. Asimismo, si van a hacer uso de lenguajes de bajo nivel, tratar de hacer su programa lo más elemental posible y comprobar cada parte del código, de modo que entiendan plenamente el funcionamiento de su programa.

3. Pruebas

1.- Programación de detección de cadenas.

a) Detección de cadenas.

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

/*Descripción de función:
Funcion contarRes, Cuenta las palabras reservadas.
Input: un arreglo del texto, la palabra reservada, struct de tipo stat, la posición y la suma total
Output: void.
*/

void contarRes(char *arreglo, char *res, 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(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;
        }
    }
```

b) Mostrar frecuencia y porcentaje

```
    if(cont > 1)
        printf("%s se repite %d veces, es un %0.4f %% \n", res ,cont, obtenerPorcentaje((float)
(strlen(res) * cont),sb));
    else
        printf("%s se repite una vez, es un %0.4f %% \n", res , obtenerPorcentaje((float)
(strlen(res)),sb));
}
```

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

/*Descripción de función:
Funcion obtenerPorcentaje, se obtiene el porcentaje.
Input: un float que representa la frecuencia con la que se repite, struct de tipo stat,
Output: retorna un porcentaje.
*/

float obtenerPorcentaje(float num, struct stat sb){
    return (num * 100 /sb.st_size);
}
```

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

```
asm se repite una vez, es un 0.0433 %
auto se repite una vez, es un 0.0578 %
bool se repite una vez, es un 0.0578 %
catch se repite una vez, es un 0.0722 %
compl se repite una vez, es un 0.0722 %
continue se repite una vez, es un 0.1156 %
do se repite 7 veces, es un 0.2023 %
else se repite 5 veces, es un 0.2890 %
export se repite una vez, es un 0.0867 %
float se repite 13 veces, es un 0.9392 %
goto se repite una vez, es un 0.0578 %
int se repite 17 veces, es un 0.7369 %
namespace se repite una vez, es un 0.1300 %
or_eq se repite una vez, es un 0.0722 %
public se repite una vez, es un 0.0867 %
return se repite 4 veces, es un 0.3468 %
sizeof se repite 4 veces, es un 0.3468 %
struct se repite 11 veces, es un 0.9536 %
this se repite una vez, es un 0.0578 %
try se repite una vez, es un 0.0433 %
typename se repite una vez, es un 0.1156 %
not_eq se repite una vez, es un 0.0867 %
using se repite una vez, es un 0.0722 %
volatile se repite una vez, es un 0.1156 %
and se repite una vez, es un 0.0433 %
bitand se repite una vez, es un 0.0867 %
break se repite una vez, es un 0.0722 %
char se repite 28 veces, es un 1.6183 %
const se repite una vez, es un 0.0722 %
default se repite una vez, es un 0.1011 %
double se repite una vez, es un 0.0867 %
enum se repite una vez, es un 0.0578 %
extern se repite una vez, es un 0.0867 %
for se repite 3 veces, es un 0.1300 %
if se repite 9 veces, es un 0.2601 %
long se repite una vez, es un 0.0578 %
new se repite una vez, es un 0.0433 %
operator se repite una vez, es un 0.1156 %
private se repite una vez, es un 0.1011 %
register se repite una vez, es un 0.1156 %
short se repite una vez, es un 0.0722 %
static se repite una vez, es un 0.0867 %
switch se repite una vez, es un 0.0867 %
throw se repite una vez, es un 0.0722 %
typedef se repite una vez, es un 0.1011 %
union se repite una vez, es un 0.0722 %
virtual se repite una vez, es un 0.1011 %
wchar_t se repite una vez, es un 0.1011 %
xor se repite una vez, es un 0.0433 %
and_eq se repite una vez, es un 0.0867 %
bitor se repite una vez, es un 0.0722 %
case se repite una vez, es un 0.0578 %
class se repite una vez, es un 0.0722 %
const_cast se repite una vez, es un 0.1445 %
delete se repite una vez, es un 0.0867 %
dynamic_cast se repite una vez, es un 0.1734 %
explicit se repite una vez, es un 0.1156 %
false se repite una vez, es un 0.0722 %
friend se repite una vez, es un 0.0867 %
inline se repite una vez, es un 0.0867 %
mutable se repite una vez, es un 0.1011 %
not se repite una vez, es un 0.0433 %
or se repite 10 veces, es un 0.2890 %
protected se repite una vez, es un 0.1300 %
reinterpret_cast se repite una vez, es un 0.2312 %
signed se repite una vez, es un 0.0867 %
static_cast se repite una vez, es un 0.1589 %
template se repite una vez, es un 0.1156 %
true se repite una vez, es un 0.0578 %
typeid se repite una vez, es un 0.0867 %
unsigned se repite una vez, es un 0.1156 %
void se repite 5 veces, es un 0.2890 %
xor_eq se repite una vez, es un 0.0867 %
while se repite 3 veces, es un 0.2167 %

Hay 74 cadenas.
```

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 Practica7.c
  - Ejecución: Practica7.c codigoFuente.c
- Linux
  - Compilación: gcc Practica7.c
  - Ejecución: ./Practica7.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));}

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
```

- Practica7.c

```c
/*
NOMBRE DE PROGRAMA: Analizador de texto (segunda parte)
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
la información necesaria
FECHA: junio 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);
float obtenerPorcentaje(float, struct stat);
void contarRes(char*, char*, struct stat, 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.
*/
```

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

    FILE *archivo;
    char *nombreArchivo;
    char *arreglo;
    int *numC;
    int alt;
    int i;
    int k;
    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];

    // 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);

    for(i = 0; i<74; i++){
        contarRes(arreglo,reser[i],sb,0);
    }

    printf("\nHay %d cadenas.",i);

    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 obtenerPorcentaje, se obtiene el porcentaje.
Input: un float que representa la frecuencia con la que se repite, struct de tipo stat,
Output: retorna un porcentaje.
*/

float obtenerPorcentaje(float num, struct stat sb){
    return (num * 100 /sb.st_size);
}
```

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

/*Descripción de función:
Funcion contarRes, Cuenta las palabras reservadas.
Input: un arreglo del texto, la palabra reservada, struct de tipo stat, la posición y la suma total
Output: void.
*/

void contarRes(char *arreglo, char *res, 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(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, es un %0.4f %% \n", res ,cont, obtenerPorcentaje((float)
(strlen(res) * cont),sb));
    else
        printf("%s se repite una vez, es un %0.4f %% \n", res , obtenerPorcentaje((float)
(strlen(res)),sb));
}
```