Universidad Tecnológica Nacional
Regional Buenos Aires

Análisis de Sistemas

Trabajo Práctico N° 3

Scanner Lenguaje Micro

Profesora: Roxana Leituz

GRUPO

Choque Mamani, Ariel	1632413	ariel0choque0mamani@gmail.com
Lannert, Nicolás	1601283	nicolas.lannert@gmail.com
Tamborini, Agustín	1680122	atamborinicriscueli@est.frba.utn.edu.ar

-2019-

CONSIGNA

Dado el autómata reconocedor de tokens del lenguaje micro, programar la función:

TOKEN Scanner(void) donde TOKEN es un enum:

```
typedef enum {
INICIO, FIN, LEER, ESCRIBIR, ID, CONSTANTE, PARENIZQUIERDO,
PARENDERECHO, PUNTOYCOMA, COMA, ASIGNACION, SUMA, RESTA, FDT
} TOKEN;
```

- La función debe leer de un archivo una secuencia de caracteres.
- En la tabla de transición debe utilizar ‘L’ para los caracteres y ‘D’ para los dígitos (puede usar las funciones ISALPHA, ISDIGIT e ISSPACE).
- Puede usar variables globales si lo considera necesario.

RESOLUCIÓN

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define NUMESTADOS 15
#define NUMCOLS 13
#define TAMARCHIVOMAX 100
#define TAMLEX 32

typedef enum {
INICIO, FIN, LEER, ESCRIBIR, ID, CONSTANTE, PARENIZQUIERDO, PARENDERECHO, PUNTOYCOMA, COMA, ASIGNACION, SUMA, RESTA, FDT, ERRORLEXICO
} TOKEN;
char tokenDescripcion [14][TAMLEX];
char nomArchi[TAMARCHIVOMAX];

typedef struct {
char identifi[TAMLEX];
TOKEN t;
} RegTS;

TOKEN tokenActual;
RegTS TS[1000] = { {"inicio", INICIO}, {"fin", FIN}, {"leer", LEER}, {"escribir", ESCRIBIR}, {"$", 99} };
int flagToken = 0;
char buffer[TAMLEX];
FILE * archivo;

/***************Declaración de funciones******************/
void cargarTokens();
int abrirArchivo();
void cerrarArchivo();
void listaSentencias();
void ProximoToken();
int Buscar(char * id, RegTS * TS, TOKEN * t);
TOKEN scanner();
int estadoFinal(int e);
int columna(int c);
void ErrorLexico();

int main()
{
cargarTokens();
int status = abrirArchivo();
if(status > 0){
listaSentencias();
cerrarArchivo();
```

```
}
return 0;
}

void cargarTokens(){
strcpy(tokenDescripcion[0], "INICIO");
strcpy(tokenDescripcion[1], "FIN");
strcpy(tokenDescripcion[2], "LEER");
strcpy(tokenDescripcion[3], "ESCRIBIR");
strcpy(tokenDescripcion[4], "CONSTANTE");
strcpy(tokenDescripcion[5], "PARENIZQUIERDO");
strcpy(tokenDescripcion[6], "PARENDERECHO");
strcpy(tokenDescripcion[7], "PUNTOYCOMA");
strcpy(tokenDescripcion[8], "COMA");
strcpy(tokenDescripcion[9], "ASIGNACION");
strcpy(tokenDescripcion[10], "SUMA");
strcpy(tokenDescripcion[11], "RESTA");
strcpy(tokenDescripcion[12], "FDT");
strcpy(tokenDescripcion[13], "ERRORLEXICO");
}

void cerrarArchivo(){
fclose(archivo);
}

int abrirArchivo(){
printf("Ingrese nombre de archivo: ");
scanf("%s", &nomArchi);

if ( (archivo = fopen(nomArchi, "r") ) == NULL ) {
printf("No pudo abrirse el archivo fuente\n");
return -1;
}
return 1;
}

//Recorre las lineas del archivo
void listaSentencias(){
system("cls");
printf("Lista de tokens del archivo %s:\n\n", nomArchi);
while ( 1 ) {
ProximoToken();
if(tokenActual == 13){
break;
}
printf("%s\n", tokenDescripcion[tokenActual]);
}
printf("\n");
system("pause");
}
void ProximoToken() {
```

```
tokenActual = scanner();
if ( tokenActual == ERRORLEXICO ) ErrorLexico();
if ( tokenActual == ID ) {
Buscar(buffer, TS, &tokenActual);
}
}

int Buscar(char * id, RegTS * TS, TOKEN * t) {
/* Determina si un identificador esta en la TS */
int i = 0;
while ( strcmp("$", TS[i].identifi) ) {
if ( !strcmp(id, TS[i].identifi) ) {
//printf("%s", TS[i].identifi);
*t = TS[i].t;
return 1;
}
i++;
}
return 0;
}

/*************************Scanner**********************************/
TOKEN scanner(){
int tabla[NUMESTADOS][NUMCOLS] =
/*0*/ { { 1, 3 , 5 , 6 , 7 , 8 , 9 , 10, 11, 14, 13, 0, 14 },
/*1*/ { 1, 1 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 },
/*2 ID*/ { 14, 14, 14, 14, 14, 14, 14, 14, 14 , 14, 14, 14, 14 },
/*3*/ { 4, 3 , 4 , 4 , 4 , 4 , 4 , 4 , 4 , 4, 4, 4, 4 },
/*4 CTE*/ { 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 },
/*5 +*/ { 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 },
/*6 -*/ { 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 },
/*7 (*/ { 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 },
/*8 )*/ { 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 },
/*9 ,*/ { 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 },
/*10 ;*/ { 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 },
/*11 */{ 14, 14, 14, 14, 14, 14, 14, 14, 14, 12, 14, 14, 14, 14 },
/*12 ASIG*/{ 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 },
/*13 fdt */{ 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 },
/*14 Err */{ 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 } };
int car;
int col;

int estado = 0;
int i = 0;
do {
car = fgetc(archivo);
col = columna(car);
estado = tabla[estado][col];
if ( col != 11 ) { //si es espacio no lo agrega al buffer
buffer[i] = car;
i++;
```

```
}
} while ( !estadoFinal(estado) && !(estado == 14) );
buffer[i] = '\0'; //complete la cadena
switch ( estado ){
case 2 : if ( col != 11 ){
//si el caracter espureo no es blanco
ungetc(car, archivo); // lo retorna al flujo
buffer[i-1] = '\0';
}
return ID;
case 4 : if ( col != 11 ) {
ungetc(car, archivo);
buffer[i-1] = '\0';
}
return CONSTANTE;
case 5 : return SUMA;
case 6 : return RESTA;
case 7 : return PARENIZQUIERDO;
case 8 : return PARENDERECHO;
case 9 : return COMA;
case 10 : return PUNTOYCOMA;
case 12 : return ASIGNACION;
case 13 : return FDT;
case 14 : return ERRORLEXICO;
}
return 0;
}

int estadoFinal(int e){
if ( e == 0 || e == 1 || e == 3 || e == 11 || e == 14 ) return 0;
return 1;
}

int columna(int c){
if ( isalpha(c) ) return 0;
if ( isdigit(c) ) return 1;
if ( c == '+' ) return 2;
if ( c == '-' ) return 3;
if ( c == '(' ) return 4;
if ( c == ')' ) return 5;
if ( c == ',' ) return 6;
if ( c == ';' ) return 7;
if ( c == ':' ) return 8;
if ( c == '=' ) return 9;
if ( c == EOF ) return 10;
if ( isspace(c) ) return 11;
return 12;
}

void ErrorLexico() { printf("Se encontro un error Lexico\n"); }
```