LENGUAJES DE PROGRAMACIÓN

Practica 2

Curso 2022-23

Grupo 204

Aparicio Fernández, Sara

De La Rosa Zarzuelo, Álvaro

Ovejero Alonso, Gonzalo

Índice

Contenido

1. Introducción

Esta práctica consiste en realizar un analizador sintáctico predictivo, que sea capaz de leer un código fuente en .txt y muestre si el código es correcto o no.

2. Participantes

El código de la práctica ha sido realizado por todos los integrantes del grupo, desarrollando la capacidad de colaboración entre nosotros para encontrar soluciones a los problemas encontrados.

A cada participante del equipo se le ha asignado el desempeño de un determinado papel y será el responsable de ello a lo largo de dicha práctica:

- Documentación: Aparicio Fernández, Sara
- Pruebas: De La Rosa Zarzuelo, Álvaro
- Coordinación: Ovejero Alonso, Gonzalo

3. Desarrollo de la práctica

3.1. Gramática empleada

R1: programa -> lista_procedimientos lista_comandos **.**

R2: lista_procedimientos -> procedimiento lista_procedimientos | **Ɛ**

R3: procedimiento -> **PROCEDURE IDENTIFIER** lista_comandos **;**

R4: lista_comandos -> comando lista_comandos'

R5: lista_comandos'-> , comando lista_comandos' | **Ɛ**

R6: comando -> **START IDENTIFIER NUMBER** | **STOP IDENTIFIER** | **SET** comando_SET | **WAIT** comando_WAIT | **RUN IDENTIFIER** | **PRINT STRING**

R7: comando_SET -> **OFF IDENTIFIER** | **ON IDENTIFIER**

R8: comando_WAIT -> **WHILE** evento | **UNTIL** evento

R9: evento -> **IDENTIFIER** | **SIGNAL IDENTIFIER**

3.2. Tabla de análisis sintáctico predictivo

- Tabla de primeros:

PROG.	LIST_PROC	PROC	LIST COM	LIST COM'	COM	COM_SET	COM_WAIT	EVENTO
PROCEDURE	PROCEDURE	PROCEDURE	START	,	START	OFF	WHILE	IDENTIFIER
START	ε		STOP	ε	STOP	ON	UNTIL	SIGNAL
STOP			SET		SET			
SET			WAIT		WAIT			
WAIT			RUN		RUN			
RUN			PRINT		PRINT			
PRINT								

- Tabla de siguientes:

PROG.	LIST_PROC	PROC	LIST_ COM	LIST_COM'	COM	COM SET	COM WAIT	EVENTO
$	START	PROCEDURE	;	;	,	,	,	,
	STOP	START			;	;	;	;
	SET	STOP						
	WAIT	SET						
	RUN	WAIT						
	PRINT	RUN						
		PRINT						

- TASP:

TASP	PROCEDURE	START	STOP	SET	WAIT	RUN	PRINT	ON	OFF	WHILE	UNTIL	ID.	SIGNAL	,	;	$
PROG	R1	R1	R1	R1	R1	R1	R1									
LISTA_PROC	R2	ε	ε	ε	ε	ε	ε									
PROC	R3															
LISTA_COM		R4	R4	R4	R4	R4	R4									
LISTA_COM'														R5	ε	
COM		R6 (START)	R6 (STOP)	R6 (SET)	R6 (WAIT)	R6 (RUN)	R6 (PRINT)									
COM_SET								R7	R7							
COM_WAIT										R8	R8					
EVENTO												R9	R9			

3.3. Estrategia de recuperación de errores

La estrategia que hemos empleado es la de “modo pánico”. Cuando nuestro programa detecta un error sintáctico y/o léxico, notifica el error que se ha producido, así como el elemento esperado, y continua con la ejecución del programa, hasta que encuentra el elemento correcto. Si no lo encuentra, se detiene la ejecución al llegar al final del fichero.

3.4. Método de construcción del analizador léxico

Para el analizador léxico se ha reutilizado el del profesor Dusan Kolár, con una modificación mínima que consiste en añadir el token PRINT a la lista e identificar los string como el conjunto de caracteres ASCII entre “”.

3.5. Código Lex

```
%{

#include <stdio.h>

#include <string.h>

#include "tokens.h"

#define KWLEN  12

char *keywords[KWLEN] = { "off",  "on","print",  "procedure",  "run",  "set",
"signal","start",  "stop", "until", "wait", "while", };

unsigned keycodes[KWLEN] = { OFF, ON,   PRINT,   PROCEDURE,   RUN,   SET, SIGNAL,
START, STOP,  UNTIL, WAIT, WHILE, };

int yywrap(void) { return 1; }

%}

LETTER    ([_a-zA-Z])

DIGIT     ([0-9])

NUMBER    ({DIGIT}+)

IDENTIFIER     ({LETTER}({LETTER}|{DIGIT})*)

STRING         (["].*["])

COMMA     ([,])

SEMICOLON     ([;])
```

```
DOT      ([\.])
SPACE    ([ \t\f\r])
SPACES   ({SPACE}+)
LINESEP  ([\n])
%%
{STRING}          return STRING;
{SPACES}          ; /* nothing to do, white space */
{IDENTIFIER}          {
                  unsigned i = 0;
                  int r=-1;
                  while (i<KWLEN && r<0)
                     if ((r=strcmp(keywords[i],yytext))==0) {
                        return keycodes[i];
                     }
                     ++i;
                  }
                  return IDENTIFIER;
               }
{LINESEP}         {
                  ++yylineno;
               }
{NUMBER}          {
                  return NUMBER;
               }
{COMMA}           return yytext[0];
{SEMICOLON}       return yytext[0];
{DOT}             return yytext[0];
.              {
                  printf ("Carácter inesperado: %s\nLinea: %i", yytext, yylineno);
                  return ERROR;
               } %%
```

3.6. Lenguaje de programación empleado

El lenguaje que hemos decidido emplear es C junto al analizador léxico en lex.

3.7. Compilación y ejecución del código

Como precondición debemos de tener en el mismo directorio todos los archivos necesarios para la ejecución del programa, es decir, ***.l**, ***.h** y ***.c**

Para ejecutar nuestro código, escribimos los siguientes comandos:

1- lex *"nombrePrograma"*.l
   Tras ejecutar el comando generera el archivo lex.yy.c
2- cc "nombrePrograma".c
3- ./a.out < *"nombrePrueba"*.txt

   En la entrega de la practica habrá dos ficheros de prueba, *prueba.txt* será un código correcto en el lenguaje de la gramática, la segunda prueba es *prueba_con_errores.txt* que tiene errores léxicos y sintácticos.