Procesadores de Lenguaje

2015/2016

Práctica 2

Analizador Léxico de hahaScript

Héctor Garbisu Arocha
Grado en Ingeniería Informática
Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

Índice

Introducción

Para esta fase del desarrollo, usaremos la herramienta Flex, que permite generar un analizador léxico en lenguaje C a partir de un fichero de definición con una sintaxis mucho más sencilla y clara. El procedimiento a seguir es el siguiente:

```
flex hahascriptlex.l
gcc -o alex lex.yy.c -lfl
alex fichero_prueba [>fichero_salida]
```

Makefile

Para generar más rápidamente el analizador léxico uso Make con un Makefile muy sencillo:

```
all: lex cc

lex:
	flex hahascriptLex.l

cc:
	gcc -g -Wall lex.yy.c -o alex -lfl

clean:
	-rm lex.yy.c
	-rm alex
```

Fichero Lex

El fichero comienza con una serie de #define para poder utilizar códigos pasados como enteros por yylex(). Para poder entender mejor la salida de la ejecución del analizador sobre un fichero de prueba, en el main() se define una tabla de decodificación, para que además del identificador numérico del componente, salga un identificador textual más fácil de leer.

Las expresiones regulares con las que se identifica a los tokens se encuentran más o menos por la mitad del fichero. Durante este proceso se han forzado más limitaciones al lenguaje. Entre ellas:

- Los identificadores sólo pueden empezar por una letra minúscula que no sea 'a' ni 'h'
- Los elementos situados a la derecha de un punto y coma (;) solo serán tratados como comentarios
- Las instrucciones dentro de bucles estarán indicadas con o mayúscula seguido de un espacio

```
%{
#define UNKNOWN 256

#define INT     257
#define FLOAT   258
#define CHAR    259
#define STRING  260

#define FUNC    261
#define LOOP    262
#define LOOPE   263
#define APAR    264
#define CPAR    265
#define ABLO    266
#define CBLO    267
#define COND    268
#define ELSE    269
#define ASIG    270
#define MAYORQUE 271
#define MAYORIGU 272
#define INCR    273
#define SUMA    274
#define MULTI   275
#define POTEN   276
#define DECRE   277
#define RESTA   278
#define DIVIS   279
#define LOGAR   280
#define RETURN  281
#define BREAK   282

#define COMILLA 283
#define PUNTOCOMA 284

#define IDEN    285

int numline =1;
void error(char* );
%}

digit           [0-9]
letter               [a-zA-Z]
lcnaoh               [b-gi-z]
notAorH              [b-gi-zB-GI-Z]
espacios        [ \t]
bobb            [-_]

%%
\n                          { numline++;      }
```

```
"a"                  { return INCR;    }
"aa"                 { return SUMA;    }
"aaa"                { return MULTI;   }
"aaaa"               { return POTEN;   }
"h"                  { return DECRE;   }
"hh"                 { return RESTA;   }
"hhh"                { return DIVIS;   }
"hhhh"               { return LOGAR;   }
"\"".*"\""                 { return STRING;  }
"\""                 { return COMILLA; }
"aHa"                { return ASIG;    }
"XDD"                { return MAYORQUE;}
"XD"                 { return MAYORIGU;}
"haha"               { return FUNC;    }
"ahah"               { return RETURN;  }
L                    { return LOOP;    }
"O"                  { return LOOPE;   }
"ha"                 { return APAR;    }
"ah"                 { return CPAR;    }
"HA"                 { return ABLO;    }
"AH"                 { return CBLO;    }
"Ha"                 { return COND;    }
"aH"                 { return ELSE;    }
"ROFL"               { return BREAK;   }
";".*$                     { /*comentario*/  }

{lcnaoh}({digit}|{letter}|{bobb})*  { return IDEN; }
{digit}+              { return INT; }
{digit}*"."{digit}*   { return FLOAT; }
\'{letter}|{digit}\'  { return CHAR; }
{espacios}+             { /*NOP*/ }
.                    { printf("Unknown character [%c]\n",yytext[0]);
                       return UNKNOWN;    }
%%
int main (int argc, char** argv){
     const char* identifiers[40];
     identifiers[0] = "UNKNOWN";
     identifiers[1] = "INT";
     identifiers[2] = "FLOAT";
     identifiers[3] = "CHAR";
     identifiers[4] = "STRING";
     identifiers[5] = "FUNC";
     identifiers[6] = "LOOP";
     identifiers[7] = "LOOPE";
     identifiers[8] = "APAR";
     identifiers[9] = "CPAR";
     identifiers[10] = "ABLO";
     identifiers[11] = "CBLO";
     identifiers[12] = "COND";
```

```
	identifiers[13] = "ELSE";
	identifiers[14] = "ASIG";
	identifiers[15] = "MAYORQUE";
	identifiers[16] = "MAYORIGU";
	identifiers[17] = "INCR";
	identifiers[18] = "SUMA";
	identifiers[19] = "MULTI";
	identifiers[20] = "POTEN";
	identifiers[21] = "DECRE";
	identifiers[22] = "RESTA";
	identifiers[23] = "DIVIS";
	identifiers[24] = "LOGAR";
	identifiers[25] = "RETURN";
	identifiers[26] = "BREAK";
	identifiers[27] = "COMILLA";
	identifiers[28] = "PUNTOCOMA";
	identifiers[29] = "IDEN";
	identifiers[30] = "";
	identifiers[31] = "";
	identifiers[32] = "";
	identifiers[33] = "";
	identifiers[34] = "";
	identifiers[35] = "";
	identifiers[36] = "";
	identifiers[37] = "";
	identifiers[38] = "";
	identifiers[39] = "";
	int s;
	if(argc>1){
		yyin = fopen(argv[1],"r");
		if(yyin==NULL)
			printf("\aError abrieno el fichero.\n");
		else{
			printf("%s :\n", argv[1]);
			s = yylex();
			while(s!=0){
				printf("%d %s \t%s\n", s, identifiers[s-256],
yytext);
				s = yylex();
			}
		}
	}
	printf("\n%d lines\n",numline);
	printf("end of file\n");
	return 0;
}
int yywrap(void){
	return 1;
}
```

Pruebas realizadas

He utilizado cuatro ficheros de prueba en esta práctica para probar las siguientes características del lenguaje:

Fichero	**Debe reconocer:**
prueba_funcion.hhs	Función, Identificador, Bloque, Print, Retorno, Comentario
prueba_funcion_args.hhs	Paso de variable
prueba_funcion_recursiva.hhs	Llamada a función, Operador MULT, Operador RESTA, Operador MAYORIGUAL
prueba_bucle.hhs	Asignación de valores, Bucles, Operación BREAK

A continuación se muestra el contenido de cada uno de los ficheros hahaScript (hhs) y el resultado del análisis.

prueba_funcion.hhs

```
;funcion sin args
 haha dame10 HA
      "toma 10";
    ahah 10;
AH
```

→

```
pruebas/prueba_funcion.hhs :
261 FUNC      haha
285 IDEN      dame10
266 ABLO      HA
260 STRING  "toma 10"
281 RETURN          ahah
257 INT       10
267 CBLO      AH

5 lines
end of file
```

prueba_funcion_args.hhs

```
haha dameX x HA
     "toma x";
     ahah x;
AH
```

→

```
pruebas/prueba_funcion_args.hhs :
261 FUNC     haha
285 IDEN     dameX
285 IDEN     x
266 ABLO     HA
260 STRING   "toma x"
281 RETURN          ahah
285 IDEN     x
267 CBLO     AH

4 lines
end of file
```

prueba_funcion_recursiva.hhs

```
haha factorial num_input HA
     Ha ha 1 XD num_input ah
     HA
                ahah 1; si num_input<=1, el resultado es 1
     AH
     ahah num_input aaa ha factorial ha num_input hh 1 ah ah;
AH
```

→

```
pruebas/prueba_funcion_recursiva.hhs :
261 FUNC     haha
285 IDEN     factorial
285 IDEN     num_input
266 ABLO     HA
268 COND     Ha
264 APAR     ha
257 INT      1
272 MAYORIGU     XD
285 IDEN     num_input
265 CPAR     ah
266 ABLO     HA
281 RETURN       ahah
257 INT      1
267 CBLO     AH
281 RETURN       ahah
285 IDEN     num_input
275 MULTI    aaa
264 APAR     ha
285 IDEN     factorial
264 APAR     ha
285 IDEN     num_input
278 RESTA    hh
257 INT      1
265 CPAR     ah
265 CPAR     ah
267 CBLO     AH

7 lines
end of file
```

prueba_bucle.hhs

```
Ha ha 1 XDD n ah HA
     "0";
AH
"0\n1"; cuac
p1 aHa 0;
p2 aHa 1;
L
O i aHa 1;
O i a;
O Ha ha n XDD i ah HA
O    ROFL;
O AH
O "\p1+\p2";
O vaux aHa p1;
O p1 aHa p2;
O p2 aHa vaux aa p2;
L
```

→

```
pruebas/prueba_bucle.hhs :
268 COND    Ha
264 APAR    ha
257 INT     1
271 MAYORQUE    XDD
285 IDEN    n
265 CPAR    ah
266 ABLO    HA
260 STRING  "0"
267 CBLO    AH
260 STRING  "0\n1"
285 IDEN    p1
270 ASIG    aHa
257 INT     0
285 IDEN    p2
270 ASIG    aHa
257 INT     1
262 LOOP    L
263 LOOPE   O
285 IDEN    i
270 ASIG    aHa
257 INT     1
263 LOOPE   O
285 IDEN    i
273 INCR    a
263 LOOPE   O
268 COND    Ha
264 APAR    ha
285 IDEN    n
```

```
271 MAYORQUE     XDD
285 IDEN      i
265 CPAR      ah
266 ABLO      HA
263 LOOPE     O
282 BREAK     ROFL
263 LOOPE     O
267 CBLO      AH
263 LOOPE     O
260 STRING    "\p1+\p2"
263 LOOPE     O
285 IDEN      vaux
270 ASIG      aHa
285 IDEN      p1
263 LOOPE     O
285 IDEN      p1
270 ASIG      aHa
285 IDEN      p2
263 LOOPE     O
285 IDEN      p2
270 ASIG      aHa
285 IDEN      vaux
274 SUMA      aa
285 IDEN      p2
262 LOOP      L

17 lines
end of file
```