7. 1.2 puntos Dada la siguiente gramática:

programa	→	funciones main
	|	main
funciones	→	funciones funcion
	|	funcion
funcion	→	”function” ident ”()” bloque
main	→	”main” bloque
bloque	→	”{” sentencias ”}”
sentencias	→	sentencias sentencia
	|	sentencia
sentencia	→	ident ”=” expresion ”;”
	|	bloque
expresion	→	expresion ”<” ident ”>” ident
	|	expresion ”?” ident ”:” ident
	|	expresion ”+” expresion
	|	”∗” expresion
	|	”&” expresion
	|	ident
	|	ident ”()”
	|	const
ident	→	ident letra
	|	letra
letra	→	”a” | ”b” | · · · | ”z”
const	→	num ”.” num
num	→	”0” | ”1” | · · · | ”9”

Obtener la tabla de tokens con el máximo nivel de abstracción suponiendo que se va a realizar:

(a) 0.6 puntos Traducción.

Solución:

La tabla de tokens con el máximo nivel de abstracción cuando se realiza el proceso completo de traducción sería la siguiente:

Nº	Token	Patrón
1	FUNCTION	”function”
2	IDENT	[a-z]+
3	PARENTS	”()”
4	MAIN	”main”
5	LLAVEI	”{”
6	LLAVED	”}”
7	ASIGN	”=”
8	PYC	”;”
9	CONST	[0-9]”.”[0-9]
10	OP_REL_MEN	”<”
11	OP_REL_MAY	”>”
12	OP_INTERR	”?”
13	OP_DOSPUNT	”:”
14	OP_MAS	”+”
15	OP_MUL_AMP	”∗” | ”&”

(b) 0.6 puntos Análisis sintáctico.

Solución:

La tabla de tokens con el máximo nivel de abstracción cuando únicamente se va a realizar análisis sintáctico puede abstraerse aún más que la anterior pero no del todo dado que la gramática de partida es libre de contexto (las llaves de los bloques y la posibilidad de la recursividad de su contenido evitan toda posibilidad de gramática regular).

En este caso la abstracción es posible realizarla, desde abajo hasta arriba, hasta abstraer una sentencia de asignación. Dado que una expresión es posible mostrarla mediante un patrón, sería posible concatenarle antes un identificador junto al carácter de la asignación. Desde este punto hasta la primera de las producciones de la gramática lo único que podríamos abstraer es el

concepto de cabecera de la función. Desde ahí hasta el principio no es posible realizar mayor abstracción, por lo que la tabla de tokens quedaría así:

Nº	Token	Patrón
1	CABECERAFUNC	”function{ident}()”
2	MAIN	”main”
3	LLAVEI	”{”
4	LLAVED	”}”
5	ASIGNACION	”{ident}={expresion}”

8. 1.0 puntos Dada la gramática con las producciones siguientes:

1) A → B c d
2) | H d H
3) | F a
4) | ε
5) B → a
6) | ε
7) H → d
8) | ε
9) F → c
10) | ε

Realizar los siguientes cálculos:

(a) 0.5 puntos Rellenar el número o los números de las producciones que deben figurar en las celdas formadas por las parejas de símbolo no terminal y terminal siguientes: (H,d), (A,$) y (A,a) de la tabla de análisis LL(1).

	c	d	a	$
A				
B				
H				
F				

Solución:

La tabla de análisis con las celdas completadas con sus correspondientes producciones quedaría así:

	c	d	a	$
A			$A \rightarrow Bcd$ $A \rightarrow Fa$	$A \rightarrow \varepsilon$
B				
H		$H \rightarrow d$ $H \rightarrow \varepsilon$		
F				

(b) 0.5 puntos Determinar las acciones del estado 0 de la tabla de análisis LR(1).

Estado	c	d	a	$	A	B	H	F
0								

Solución:

Estado	c	d	a	$	A	B	H	F
0	$d7/r6$	$d6/r8$	$d5/r10$	$r4$	1	2	3	4

9. 1.2 puntos Dada la siguiente especificación YACC:

```
%token PROGRAM          // "program"
%token PI PD            // "(", ")"
%token BEGIN END        // "begin", "end"
%token IDENT ASIGN      // [a-z]+, "="
%token PYC              // ";"
%token FOR TO DO        // "for", "to", "do"
%token WHILE COMA       // "while", ","
%token TIPO             // "int", "char", "float", "boolean"
%token CONST            // [0-9]+, '[a-z0-9]', [0-9]+.[0-9]+, "true"|"false"

%left OP_MAS_MENOS      // "+", "-"
%left OP_MUL_DIV        // "*", "/"
%nonassoc OP_REL        // "<", "<=", "==", "!=", ">=", ">"
%right OP_NOT           // "!"

%%
programa   : PROGRAM bloque ;
bloque     : BEGIN decl_var sentencias END ;
decl_var   : TIPO lista_var PYC
           | ;
lista_var  : lista_var COMA IDENT
           | IDENT ;
sentencias : sentencias sentencia
           | sentencia ;
sentencia  : IDENT ASIGN expresion PYC
           | FOR IDENT ASIGN expresion TO expresion DO sentencia
           | WHILE expresion DO sentencia ;
expresion  : expresion OP_REL expresion
           | expresion OP_MAS_MENOS expresion
           | expresion OP_MUL_DIV expresion
           | OP_MAS_MENOS expresion %prec OP_NOT
           | OP_NOT expresion
           | IDENT
           | CONST ;

%%
```

Obtener la gramática con atributos para realizar las comprobaciones semánticas sobre tipos. Se debe indicar qué información sería necesaria tener previamente en la tabla de símbolos, estructura de ésta y aquellas funciones que se necesiten para su gestión (no es necesario implementar dichas funciones). Hay que tener en cuenta que el lenguaje no admite coerciones.

Solución:

Al insertar las acciones semánticas que se deben considerar en el problema, tendremos la necesidad de usar una **tabla de símbolos** para alojar las variables que se declaren. Las consideraciones semánticas necesarias son:

- Evitar duplicidad en declaración de variables.
- En sentencia de asignación, el identificador debe estar declarado y ser del mismo tipo que la expresión.
- En sentencia `for`, el identificador debe estar declarado, ser del mismo tipo que la expresión que se le asigna (la primera de ellas) y la segunda expresión también ha de ser del mismo tipo.
- En sentencia `while` la expresión debe ser de tipo lógico (dado que no se menciona la necesidad de evaluar expresiones lógicas de manera forzada en este tipo de sentencias, también se considera correcto evaluar expresiones de cualquier tipo considerando que toda expresión que no sea 0 es verdadera).
- En expresiones, dado que se indica claramente en el enunciado, no admite la posibilidad de adaptación de tipos, por lo tanto se evalúan expresiones que sean concordantes al operador y, donde haya dos expresiones, ambas deben ser del mismo tipo.

La estructura de la tabla de símbolos estará formada por la misma que la propuesta en las prácticas de la asignatura y los métodos de manejo de la misma serán:

- `int TS_InsertaId (char *nombre, tipodato tipo)` Busca en la TS una entrada que sea igual al nombre dado como argumento, en caso de encontrarla devuelve 0 y en otro caso lo inserta como variable asignándole el tipo que también se le aporta como argumento.
- `int TS_ExisteId (char *nombre)` Busca en la TS una entrada que sea igual al nombre dado como argumento, en caso de encontrarla devuelve el valor del campo `tipo` y -1 en otro caso.

De esta forma, la gramática con atributos que resuelve el problema quedaría así:

```
......
%%
programa   : PROGRAM bloque ;
bloque     : BEGIN decl_var sentencias END ;

// Aquí se puede solucionar insertando una acción en mitad de la regla.

decl_var   : TIPO { tmp= $1.atrib ; } lista_var PYC
           | ;
//-------------------------------------------------------------------------------------------------------
lista_var  : lista_var COMA IDENT
//-------------------------------------------------------------------------------------------------------
                                     { if (!TS_InsertaId ($3.nombre, tmp))
                                          printf ("Error semántico:
                                                   Identificador duplicado\n");
                                     }
//-------------------------------------------------------------------------------------------------------
           | IDENT
//-------------------------------------------------------------------------------------------------------
                                     { if (!TS_InsertaId ($1.nombre, tmp))
                                          printf ("Error semántico:
                                                   Identificador duplicado\n");
                                     }
sentencias : sentencias sentencia
           | sentencia ;
//-------------------------------------------------------------------------------------------------------
sentencia  : IDENT ASIGN expresion PYC
//-------------------------------------------------------------------------------------------------------
                                     { if ((tipo=TS_ExisteId ($1.nombre))!=-1)
                                          printf ("Error semántico:
                                                   Identificador no declarado en asignación\n");
                                       else if (tipo!=$3.tipo)
                                          printf ("Error semántico:
                                                   Tipos incompatibles en asignación\n");
                                     }
//-------------------------------------------------------------------------------------------------------
           | FOR IDENT ASIGN expresion TO expresion DO sentencia
//-------------------------------------------------------------------------------------------------------
                                     { if ((tipo=TS_ExisteId ($1.nombre))!=-1)
                                          printf ("Error semántico:
                                                   Identificador no declarado en sentencia for\n");
                                       else if (tipo != $4.tipo || tipo != $6.tipo)
                                          printf ("Error semántico:
                                                   Tipos incompatibles en expresiones
                                                   de sentencia for\n");
                                     }
//-------------------------------------------------------------------------------------------------------
           | WHILE expresion DO sentencia
//-------------------------------------------------------------------------------------------------------
                                     { // Si admitimos únicamente expresiones lógicas:
                                       if ($2.tipo!=BOOLEANA)
                                          printf ("Error semántico:
                                                   Tipo no booleano en sentencia while\n");
                                       // Si admitimos cualquier tipo de expresión, no habría que
                                       // añadir regla semántica.
                                     }
//-------------------------------------------------------------------------------------------------------
expresion  : expresion OP_REL expresion
//-------------------------------------------------------------------------------------------------------
                                     { if ($1.tipo != $3.tipo)
                                       {
                                          printf ("Error semántico:
                                                   Tipos incompatibles en expresión relacional\n");
                                          $$.tipo= INCORRECTO ;
```

```
                                        }
                                        else
                                           $$.tipo= $1.tipo ;
                                     }
//-------------------------------------------------------------------------------------------------------
          | expresion OP_MAS_MENOS expresion
//-------------------------------------------------------------------------------------------------------
                                     { if (($1.tipo==$3.tipo) && (($1.tipo==ENTERO) || ($1.tipo==REAL)))
                                           $$.tipo= $1.tipo ;
                                        else
                                        {
                                           printf ("Error semántico:
                                                    Tipos incompatibles en expresión aritmética\n");
                                           $$.tipo= INCORRECTO ;
                                        }
                                     }
//-------------------------------------------------------------------------------------------------------
          | expresion OP_MUL_DIV expresion
//-------------------------------------------------------------------------------------------------------
                                     { if (($1.tipo==$3.tipo) && (($1.tipo==ENTERO) || ($1.tipo==REAL)))
                                           $$.tipo= $1.tipo ;
                                        else
                                        {
                                           printf ("Error semántico:
                                                    Tipos incompatibles en expresión aritmética\n");
                                           $$.tipo= INCORRECTO ;
                                        }
                                     }
//-------------------------------------------------------------------------------------------------------
          | OP_MAS_MENOS expresion %prec OP_NOT
//-------------------------------------------------------------------------------------------------------
                                     { if (($1.tipo==ENTERO) || ($1.tipo==REAL))
                                           $$.tipo= $1.tipo ;
                                        else
                                        {
                                           printf ("Error semántico:
                                                    Tipo incompatible en expresión aritmética\n");
                                           $$.tipo= INCORRECTO ;
                                        }
                                     }
//-------------------------------------------------------------------------------------------------------
          | OP_NOT expresion
//-------------------------------------------------------------------------------------------------------
                                     { if ($1.tipo==BOOLEANO)
                                           $$.tipo= $1.tipo ;
                                        else
                                        {
                                           printf ("Error semántico:
                                                    Tipo incompatible en expresión lógica\n");
                                           $$.tipo= INCORRECTO ;
                                        }
                                     }
//-------------------------------------------------------------------------------------------------------
          | IDENT
//-------------------------------------------------------------------------------------------------------
                                     { if (!(tipo=TS_ExisteId ($1.nombre)))
                                        {
                                           printf ("Error semántico:
                                                    Identificador no declarado en asignación\n");
                                           $$.tipo= INCORRECTO ;
                                        }
                                        else
                                           $$.tipo= tipo ;
                                     }
//-------------------------------------------------------------------------------------------------------
          | CONST
//-------------------------------------------------------------------------------------------------------
                                     { $$.tipo= $1.atrib ; }
%%
........
```