Compilación

Entrega final de la práctica individual

Author:
David Sandström Daparte
dsandstrom001

May 18, 2019

Contents

1 Objetivos entregados

En primer lugar cabe destacar que el presente documento se ha realizado en modo apaisado por facilitar la comprensión del ETDS y de la descripción de los tokens. Para esta práctica se han implementado los siguientes objetivos:

1. Implementación del analizador léxico y sintáctico. Desarrollo del ETDS.
2. Implementación del traductor.
3. Traducción de expresiones booleanas.
4. Implementación de una nueva estructura de control. El bucle for.

2 Autoevaluación

A mi parecer, teniendo en cuenta la corrección del apartado de traducción de expresiones booleanas y la complejidad de la estructura de control añadida, opino que mi nota debería ser un **7,5**.

3 Descripción de los tokens empleados

Nombre del token	Descripción informal	Expresión LEX	Lexemas
TPROGRAM	Es una palabra reservada que se compone por los caracteres p r o g r a m	”program”	program
TIF	Es una palabra reservada que se compone por los caracteres i f	”if”	if
TTHEN	Es una palabra reservada que se compone por los caracteres t h e n	”then”	then
TWHILE	Es una palabra reservada que se compone por los caracteres w h i l e	”while”	while
TFOR	Es una palabra reservada que se compone por los caracteres f o r	”for”	for
TDO	Es una palabra reservada que se compone por los caracteres d o	”do”	do
TUNTIL	Es una palabra reservada que se compone por los caracteres u n t i l	”until”	until
TELSE	Es una palabra reservada que se compone por los caracteres e l s e	”else”	else
TSKIP	Es una palabra reservada que se compone por los caracteres s k i p	”skip”	skip

Nombre del token	Descripción informal	Expresión LEX	Lexemas
TREAD	Es una palabra reservada que se compone por los caracteres r e a d	”read”	read
TPRINT	Es una palabra reservada que se compone por los caracteres p r i n t l n	”println”	println
TVAR	Es una palabra reservada que se compone por los caracteres v a r	”var”	var
TINOUT	Es una palabra reservada que se compone por los caracteres i n o u t	”in out”	in out
TIN	Es una palabra reservada que se compone por los caracteres i n	”in”	in
TOUT	Es una palabra reservada que se compone por los caracteres o u t	”out”	out
TTIPEFLOAT	Es una palabra reservada que se compone por los caracteres f l o a t	”float”	float
TTIPEINTEGER	Es una palabra reservada que se compone por los caracteres i n t e g e r	”integer”	integer
TPROCEDURE	Es una palabra reservada que se compone por los caracteres p r o c e d u r e	”procedure”	procedure
TOR	Es una palabra reservada que se compone por los caracteres o r	”or”	or

Nombre del token	Descripción informal	Expresión LEX	Lexemas
TAND	Es una palabra reservada que se compone por los caracteres a n d	”and”	and
TNOT	Es una palabra reservada que se compone por los caracteres n o t	”not”	not
TEQUAL	Es una palabra reservada que se compone por los caracteres = =	”==”	==
TGREATEQ	Es una palabra reservada que se compone por los caracteres >=	”>=”	>=
TFEWEQ	Es una palabra reservada que se compone por los caracteres <=	”<=”	<=
TNOTEQ	Es una palabra reservada que se compone por los caracteres / =	”/=”	/=
TGREATER	Es una palabra reservada que se compone por el caracter >	”>”	>
TFEWER	Es una palabra reservada que se compone por el caracter <	”<”	<
TASSIG	Es una palabra reservada que se compone por el caracter =	”=”	=
TLBRACE	Es una palabra reservada que se compone por el caracter {	\{	{
TRBRACE	Es una palabra reservada que se compone por el caracter }	\}	}
TLPAREN	Es una palabra reservada que se compone por el caracter (	\(	(
TRPAREN	Es una palabra reservada que se compone por el caracter)	\)	)
TSEMIC	Es una palabra reservada que se compone por el caracter ;	”;”	;

Nombre del token	Descripción informal	Expresión LEX	Lexemas
TCOMMA	Es una palabra reservada que se compone por el caracter ,	”,”	,
TDOUBLEDOT	Es una palabra reservada que se compone por el caracter :	”:”	:
TMUL	Es una palabra reservada que se compone por el caracter *	*	*
TPLUS	Es una palabra reservada que se compone por el caracter +	\+	+
TMINUS	Es una palabra reservada que se compone por el caracter -	\-	-
TDIV	Es una palabra reservada que se compone por el caracter /	\/	/
TINTEGER	Concatenación de dígitos de longitud mayor o igual a uno.	[0-9]+	Acepta: 1. 0 2. 0034 3. 450 4. 010980 No acepta: 1. 0a0 2. 90i90 3. 13.34.67 4. 89w89

Nombre del token	Descripción informal	Expresión LEX	Lexemas
TFLOAT	Los números reales se representan con una secuencia de dígitos (de al menos un dígito), seguido por un punto y una secuencia de dígitos (de al menos un dígito). Posteriormente, pueden incluir el exponente mediante la letra "E" o "e" pudiendo especificar si es positivo o negativo; a esto, le sigue una secuencia de dígitos (de al menos un dígito).	[0-9]+\.[0-9]+([eE][-+]?[0-9]+)?	Acepta: 1. 0.0 2. 00.00e+45 3. 00000010.1e+900 4. 90.01E-0 5. 9.2E10 6. 9.2e10 No acepta: 1. .23 2. 0.45e+ 3. 45.E-4 4. 9.9E

Nombre del token	Descripción informal	Expresión LEX	Lexemas
TIDENTIFIER	Comienza por un carácter alfabético. Posteriormente, puede tener cualquier combinación de caracteres alfanuméricos, entre los cuales se pueden intercalar un guión bajo seguido como mucho. Nunca termina en guión bajo.	[a-zA-Z](_?[a-zA-Z0-9])*	Acepta: 1. Z_9_22_1 2. a_12_t_09 3. o9l_j_fng_df 4. P_P_P_P No acepta: 1. 01234 2. _1_ 3. pki_ 4. i09__2

Nombre del token	Descripción informal	Expresión LEX	Lexemas
COMENTARIO El comentario es un token, no obstante, no tienen un token explícitamente asignado. Se comprueba su corrección léxica, pero no se le asigna un token como tal.	Los comentarios comienzan por la secuencia de caracteres /* y terminan por la secuencia */, pero no pueden contener la secuencia de caracteres */ entre el comienzo y el final del token.	\/*(([*]+[^/*])|[^*])*[*]+\/	Acepta: 1. /*****válido/*/ 2. /*//////////*/ 3. /*****/ 4. /*prueba*/ No acepta: 1. /no válido/ 2. /******/**/ 3. /*/ 4. /*prueba/

4 Gramática empleada

programa → **program id**
bloque;

bloque → { declaraciones
decl_de_subprogs
lista_de_sentencias
}

delcaraciones → **var** lista_de_ident : tipo ; declaraciones
| ξ

lista_de_ident → **id** resto_lista_id

resto_lista_id → **,** **id** resto_lista_id
| ξ

tipo → **integer** | **float**

decl_de_subprogs → decl_de_subprograma decl_de_subprogs
| ξ

decl_de_subprograma → **procedure id** argumentos
bloque;

argumentos → (lista_de_param)
| ξ

lista_de_param → lista_de_ident : clase_par tipo resto_lis_de_param

clase_par → **in** | **out** | **in out**

resto_lis_de_param → ; lista_de_ident : clase_par tipo resto_lis_de_param
| ξ

lista_de_sentencias → sentencia lista_de_sentencias
| ξ

sentencia → variable = expresion ;
| **if** expresion **then** { lista_de_sentencias };
| **while** expresion { lista_de_sentencias };
| **do** { lista_de_sentencias } **until** expresion **else** { lista_de_sentencias };
| **skip if** expresion ;
| **read** (variable) ;
| **println** (expresion) ;

variable → *id*

expresion → expresion == expresion
| expresion > expresion
| expresion < expresion
| expresion >= expresion
| expresion <= expresion
| expresion / = expresion
| expresion + expresion
| expresion − expresion
| expresion ∗ expresion
| expresion / expresion
| **id**
| **num_entero**
| **num_real**
| (expresion)

5 Explicación de los símbolos y sus atributos

En primer lugar cabe destacar que para la resolución de los objetivos opcionales solo se han introducido un total de 4 símbolos terminales:

1. for
2. and

3. or
4. not

Y ninguno de ellos cuenta con ningún atributo. Es por ello por lo que no se mencionan en los dos siguientes apartados.

5.1 Símbolos no terminales

Los símbolos no terminales empleados para la resolución de la práctica son los que a continuación se detallan, así como los atributos de cada uno de ellos:

1. **lista_de_ident**: no terminal que almacena una lista de identificadores. Emplea el siguiente atributo:
   (a) .lnom: almacena todos los identificadores que puedan aparecer en ciertas situaciones.
2. **tipo**: no terminal que almacena los tipos de las variables declaradas. Emplea el siguiente atributo:
   (a) .clase: almacena el tipo de las variables, empleado en este caso para diferenciar las variables de tipo "real" y "entero".
3. **clase_par**: no terminal que almacena el tipo de cada uno de cada uno de los parámetros que se le pasan a una subrutina. Emplea el siguiente atributo:
   (a) .tipo: almacena el tipo de cada parámetro. En caso de que el parámetro sea de tipo "in" tomará el valor "val" y en caso de que el parámetro sea de tipo "out" o "in out" el valor del atributo será "ref".
4. **lista_de_sentencias**: no terminal que almacena un listado de sentencias leídas como parte del programa de entrada. Algunos ejemplos de las sentencias que puede almacenar son: **if**, **while**... Emplea el siguiente atributo:
   (a) .skip: almacena las referencias de todas las instrucciones skip almacenadas en la lista.
5. **sentencia**: no terminal que almacena las sentencias empleadas en el programa a traducir. Emplea el siguiente atributo

   (a) .skip: referencia en la que se encuentra la sentencia goto asociada a la instrucción.

6. **variable**: no terminal que almacena una de las variables del programa. Emplea el siguiente atributo:

   (a) .nombre: nombre de la variable.

7. **expresion**: no terminal que almacena una expresión aritmética o lógica. Emplea los siguientes atributos:

   (a) .true: almacena la referencia en la que se encuentra el goto que habrá de ejecutarse en caso de cumplirse la condición.
   (b) .false: almacena la referencia en la que se encuentra el goto que habrá de ejecutarse en caso de no cumplirse la condición.
   (c) .nombre: valor registrado por la expresión.

8. **M**: no terminal que almacena la referencia de la instrucción actual. Emplea el siguiente atributo:

   (a) .ref: empleado para almacenar la referencia de la instrucción.

5.2 Símbolos terminales

Los símbolos terminales empleados para la resolución de la práctica son los que a continuación se detallan, así como los atributos de cada uno de ellos:

1. **id**: terminal que guarda el identificador de una variable. Emplea el siguiente atributo:

   (a) .nombre: empleado para almacenar el identificador en las situaciones en que sea necesario.

2. **entero**: terminal que guarda un número entero. Emplea el siguiente atributo:

   (a) .nombre: valor del número entero.

3. **real**: terminal que guarda un número real. Emplea el siguiente atributo:

   (a) .nombre: valor del número real.

6 Descripción de las abstracciones funcionales empleadas

Las abstracciones funcionales empleadas son las siguientes:

- **anadir_inst(instruccion)**: recibe como entrada el código de una instrucción. Añade esa instrucción al codigo traducido.
- **nuevo_id()**: crea un nuevo id con la forma T_{k+1}, siendo k la cantidad de identificadores que se habian creado hasta el momento. Mediante esta función se obtienen los identificadores temporales empleados para traducir las expresiones.
- **anadir_declaracion(lista, tipo)**: por cada nombre de la lista de entrada, empezando por el primero y hasta el último añade una instrucción de esta forma:

  $$nombre : tipo$$

  Esta función es empleada para procesar todas las declaraciones de variables que pueda contener el código a compilar.
- **anadir(lista, nombre)**: añade el nombre al comienzo de la lista de entrada y retorna la lista con dicha modificación aplicada.
- **inilista(nombre)**: crea una lista con el nombre que recibe como entrada y retorna la lista creada.
- **completar(valor, referencia)**: añade el valor indicado como parámetro de la función (valor) al final de la instrucción cuya referencia se indica mediante el parámetro referencia. Mediante esta función podemos completar las instrucciones "goto", cuya referencia desconozcamos en el momento de añadir la instrucción.
- **lista_vacia()**: retorna una lista vacía.
- **unir(**$lista_1$**,** $lista_2$**)**: retorna la unión de ambas listas.
- **escribir()**: una vez completada la lectura y traducción, vuelca todos los resultados en un fichero de texto.
- **vacio()**: función que retorna el valor *null* o lo que sea necesario asignar como vacío.

7 Esquema de traducción dirigido por la sintaxis

7.1 Significado de la sentencia Skip If

A la sentencia Skip If dentro del *do-until-else* le hemos dotado del siguiente significado: dependiendo del valor que su expresión produzca y donde se encuentre:

1. Si produce el valor **true**:
   - Si se encuentra en la sección do: se pasa a evaluar de nuevo la condición del until, ignorando las instrucciones restantes del bloque do.
   - Si se encuentra en la sección else: no hace nada. Simplemente pasa a ejecutar la siguiente instrucción. Como caso excepcional si se da esta situación además de que la estructura do-until-else se encuentra dentro de un while/for, el skip if nos llevaría al inicio del bucle while/for.
2. Si produce el valor **false** : se ejecuta la siguiente instrucción independientemente de la sección donde se encuentre.

7.2 ETDS básico

$$\textbf{programa} \rightarrow program\ id\ \{anadir\_inst(prog||id.nombre||;);\}\ bloque\ ;\ \{anadir\_inst(halt;); escribir();\}$$

$$\textbf{bloque} \rightarrow \{declaraciones\ decl\_de\_subprogs\ lista\_de\_sentencias\}$$

$$\textbf{declaraciones} \rightarrow var\ lista\_de\_ident\ :\ tipo;\ \{anadir\_declaracion(lista\_de\_ident.lnom,\ tipo.clase);\}\ declaraciones \mid \xi$$

$$\textbf{lista\_de\_ident} \rightarrow id\ resto\_lista\_id\ \{lista\_de\_ident.lnom = anadir(resto\_lista\_id.lnom,\ id.nombre);\}$$

$$\textbf{resto\_lista\_id} \rightarrow ,id\ resto\_lista\_id\ \{resto\_lista\_id.lnom = anadir(resto\_lista\_id_1.lnom,\ id.nombre);\} \mid \xi\ \{resto\_lista\_id.lnom = lista\_vacia();\}$$

$$\textbf{tipo} \rightarrow integer\ \{tipo.clase = "int";\} \mid float\ \{tipo.clase = "real";\}$$

decl_de_subprogs $\rightarrow$ *decl_de_subprograma decl_de_subprogs*
$|\ \xi$

decl_de_subprograma $\rightarrow$ *procedure id* $\{anadir\_inst(proc||id.nombre||;);\}$ *argumentos bloque*; $\{anadir\_inst(endproc;);\}$

argumentos $\rightarrow$ $(lista\_de\_param)$
$|\ \xi$

lista_de_param $\rightarrow$ *lista_de_ident* : *clase_par tipo* $\{anadir\_declaracion(lista\_de\_ident.lnom, clase\_par.tipo||\_||tipo.clase);\}$
resto_lis_de_param

clase_par $\rightarrow$ *in* $\{clase\_par.tipo = "val";\}$
$|$ *out*$\{clase\_par.tipo = "ref";\}$
$|$ *in out* $\{clase\_par.tipo = "ref";\}$

resto_lis_de_param $\rightarrow$; *lista_de_ident* : *clase_par tipo* $\{anadir\_declaracion(lista\_de\_ident.lnom, clase\_par.tipo||\_||tipo.clase);\}$
resto_lis_de_param
$|\ \xi$

lista_de_sentencias $\rightarrow$ *sentencia lista_de_sentencias* $\{lista\_de\_sentencias.skip = unir(sentencia.skip, lista\_de\_sentencias_1.skip);\}$
$|\ \xi$ $\{lista\_de\_sentencias.skip = lista\_vacia();\}$

sentencia $\rightarrow$ $variable = expresion;$ $\left\{\begin{array}{c} anadir\_inst(variable.nombre|| := ||expresion.nombre;); \\ sentencia.skip = lista\_vacia(); \end{array}\right\}$

| **if** $expresion$ **then** $\{M\ lista\_de\_sentencias\}M;$ $\left\{\begin{array}{c} completar(expresion.true, M_1.ref); \\ completar(expresion.false, M_2.ref); \\ sentencia.skip = lista\_de\_sentencias.skip \end{array}\right\}$

| **while** $M\ expresion\{M\ lista\_de\_sentenciasM\};$ $\left\{\begin{array}{c} completar(expresion.true, M_2.ref); \\ completar(expresion.false, M_3.ref + 1); \\ anadir\_inst(goto||M_1.ref); \\ completar(lista\_sentencias.skip, M_1.ref); \\ sentencia.skip := lista\_vacia(); \end{array}\right\}$

| **do** M { $lista\_de\_sentencias$} **until** $M\ expresion$ **else** M { $lista\_de\_sentencias$} ; $\left\{\begin{array}{c} completar(expresion.true, M_1.ref); \\ completar(expresion.false, M_3.ref); \\ completar(lista\_sentencias_1.skip, M_2.ref); \\ sentencia.skip := lista\_vacia(); \end{array}\right\}$

| **skip if** $expresion\ M;$ $\left\{\begin{array}{c} completar(expresion.false, M.ref); \\ sentencia.skip = expresion.true; \end{array}\right\}$

| **read** $(variable);$ $\left\{\begin{array}{c} anadir\_inst(read||variable.nombre||;); \\ sentencia.skip = lista\_vacia(); \end{array}\right\}$

| **println** $(expresion);$ $\left\{\begin{array}{c} anadir\_inst(write||expresion.nombre||;); \\ anadir\_inst(writeln;); \\ sentencia.skip = lista\_vacia(); \end{array}\right\}$

variable $\rightarrow$ id $\{variable.nombre = id.nombre;\}$

$$\mathbf{M} \rightarrow \xi \ \{M.ref = obtenref();\}$$

$$\mathbf{expresion} \rightarrow expresion == expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| == ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\}$$

$$| \ expresion > \ \ expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| > ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\}$$

$$| \ expresion < \ \ expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| < ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\}$$

$$| \ expresion \geq \ \ expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| \geq ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\}$$

$$| \; expresion \; \leq expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| \leq ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\}$$

$$| \; expresion \; \neq expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| \neq ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\}$$

$$| \; expresion + \; expresion \left\{ \begin{array}{c} expresion.nombre = nuevo\_id(); \\ expresion.true = vacio(); \\ expresion.false = vacio(); \\ anadir\_inst(expresion.nombre|| := ||expresion_1.nombre|| + ||expresion_2.nombre||;); \end{array} \right\}$$

$$| \; expresion - \; expresion \left\{ \begin{array}{c} expresion.nombre = nuevo\_id(); \\ expresion.true = vacio(); \\ expresion.false = vacio(); \\ anadir\_inst(expresion.nombre|| := ||expresion_1.nombre|| - ||expresion_2.nombre||;); \end{array} \right\}$$

$$| \; expresion * \; expresion \left\{ \begin{array}{c} expresion.nombre = nuevo\_id(); \\ expresion.true = vacio(); \\ expresion.false = vacio(); \\ anadir\_inst(expresion.nombre|| := ||expresion_1.nombre|| * ||expresion_2.nombre||;); \end{array} \right\}$$

$$| \; expresion \; / \; \; expresion \left\{ \begin{array}{c} expresion.nombre = nuevo\_id(); \\ expresion.true = vacio(); \\ expresion.false = vacio(); \\ anadir\_inst(expresion.nombre|| := ||expresion_1.nombre||/||expresion_2.nombre||;); \end{array} \right\}$$

$$| \; id \quad \left\{ \begin{array}{c} expresion.nombre = id.nombre; \\ expresion.true = vacio(); \\ expresion.false = vacio(); \end{array} \right\}$$

$$| \; n\_entero \quad \left\{ \begin{array}{c} expresion.nombre = entero.nombre; \\ expresion.true = vacio(); \\ expresion.false = vacio(); \end{array} \right\}$$

$$| \; n\_real \quad \left\{ \begin{array}{c} expresion.nombre = real.nombre; \\ expresion.true = vacio(); \\ expresion.false = vacio(); \end{array} \right\}$$

$$| \; (expresion) \left\{ \begin{array}{c} expresion.nombre = expresion_1.nombre; \\ expresion.true = expresion_1.true; \\ expresion.false = expresion_1.false; \end{array} \right\}$$

7.3 Casos de prueba básicos

Para comprobar el correcto funcionamiento del traductor descrito hasta el momento se han implementado 4 ficheros de pruebas ejecutables mediante la aplicación. Los ficheros:

- prueba1.in
- prueba2.in

Incluyen dos fragmentos de código sin errores para comprobar su correcto funcionamiento y los ficheros:

- pruebaMala1.in
- pruebaMala2.in

Contienen código con errores de forma que se pueda probar el comportamiento de la aplicación ante ficheros con errores.

8 Objetivos extra añadidos.

8.1 Tratamiento de expresiones booleanas.

8.1.1 Fragmento añadido a la gramática

expresion → expresion **or** expresion
| expresion **and** expresion
| **not** expresion

8.1.2 Gramática resultante

La gramática resultante es la descrita con anterioridad en el presente documento con el añadido del subapartado anterior.

programa → **program id**
bloque;

bloque → { declaraciones
decl_de_subprogs
lista_de_sentencias
}

delcaraciones → **var** lista_de_ident : tipo ; declaraciones
| ξ

lista_de_ident → **id** resto_lista_id

resto_lista_id → **, id** resto_lista_id
| ξ

tipo → **integer** | **float**

decl_de_subprogs → decl_de_subprograma decl_de_subprogs
| ξ
decl_de_subprograma → **procedure id** argumentos
bloque;

argumentos → (lista_de_param)
| ξ

lista_de_param → lista_de_ident : clase_par tipo resto_lis_de_param

clase_par → **in** | **out** | **in out**

resto_lis_de_param → ; lista_de_ident : clase_par tipo resto_lis_de_param
| ξ

lista_de_sentencias → sentencia lista_de_sentencias
| ξ

sentencia → variable = expresion ;
| **if** expresion **then** { lista_de_sentencias };
| **while** expresion { lista_de_sentencias };
| **do** { lista_de_sentencias } **until** expresion **else** { lista_de_sentencias };
| **skip if** expresion ;
| **read** (variable) ;
| **println** (expresion) ;

variable → *id*

expresion → expresion **or** expresion
| expresion **and** expresion
| **not** expresion
| expresion == expresion
| expresion > expresion
| expresion < expresion
| expresion >= expresion
| expresion <= expresion
| expresion / = expresion
| expresion + expresion
| expresion − expresion
| expresion ∗ expresion
| expresion / expresion
| **id**
| **num_entero**
| **num_real**
| (expresion)

8.1.3 Fragmento añadido al ETDS

Partiendo del ETDS previamente expuesto, ha sido necesario realizar el siguiente añadido al mismo con el fin de poder procesar las expresiones booleanas:

$$\textbf{expresion} \rightarrow expresion\ \textbf{or}\quad M\ expresion \left\{ \begin{array}{c} completar(expresion_1.false, M.ref); \\ expresion.true = unir(expresion_1.true, expresion_2.true); \\ expresion.false = expresion_2.false; \end{array} \right\}$$

$$\mid expresion\ \textbf{and}\ M\ expresion \left\{ \begin{array}{c} completar(expresion_1.true, M.ref); \\ expresion.true = expresion_2.true; \\ expresion.false = unir(expresion_1.false, expresion_2.false); \end{array} \right\}$$

$$\mid \textbf{not}\ expresion\ \left\{ \begin{array}{l} expresion.true = expresion_1.false; \\ expresion.false = expresion_1.true; \end{array} \right\}$$

8.1.4 ETDS resultante

El ETDS resultante consiste en el ETDS básico, con el añadido mencionado en el subapartado anterior:

$$\begin{aligned}
\textbf{programa} &\rightarrow program\ id\ \{anadir_inst(prog||id.nombre||;);\} \\
&\quad bloque\ ;\ \{anadir_inst(halt;); escribir();\} \\
\\
\textbf{bloque} &\rightarrow \{declaraciones\ decl_de_subprogs\ lista_de_sentencias\} \\
\\
\textbf{declaraciones} &\rightarrow var\ lista_de_ident\ :\ tipo;\ \{anadir_declaracion(lista_de_ident.lnom,\ tipo.clase);\}\ declaraciones \\
&\quad | \ \xi \\
\\
\textbf{lista_de_ident} &\rightarrow id\ resto_lista_id\ \{lista_de_ident.lnom = anadir(resto_lista_id.lnom,\ id.nombre);\} \\
\\
\textbf{resto_lista_id} &\rightarrow ,id\ resto_lista_id\ \{resto_lista_id.lnom = anadir(resto_lista_id_1.lnom,\ id.nombre);\} \\
&\quad | \ \xi\ \{resto_lista_id.lnom = lista_vacia();\} \\
\\
\textbf{tipo} &\rightarrow integer\ \{tipo.clase = "int";\} \\
&\quad | \ float\ \{tipo.clase = "real";\}
\end{aligned}$$

$$\textbf{decl\_de\_subprogs} \rightarrow decl\_de\_subprograma\ decl\_de\_subprogs \mid \xi$$

$$\textbf{decl\_de\_subprograma} \rightarrow procedure\ id\ \{anadir\_inst(proc||id.nombre||;);\}\ argumentos\ bloque;\{anadir\_inst(endproc;);\}$$

$$\textbf{argumentos} \rightarrow (lista\_de\_param) \mid \xi$$

$$\textbf{lista\_de\_param} \rightarrow lista\_de\_ident\ :\ clase\_par\ tipo\ \{anadir\_declaracion(lista\_de\_ident.lnom, clase\_par.tipo||\_||tipo.clase);\}\ resto\_lis\_de\_param$$

$$\textbf{clase\_par} \rightarrow in\ \{clase\_par.tipo = "val";\} \mid out\{clase\_par.tipo = "ref";\} \mid in\ out\ \{clase\_par.tipo = "ref";\}$$

$$\textbf{resto\_lis\_de\_param} \rightarrow ;\ lista\_de\_ident\ :\ clase\_par\ tipo\ \{anadir\_declaracion(lista\_de\_ident.lnom, clase\_par.tipo||\_||tipo.clase);\}\ resto\_lis\_de\_param \mid \xi$$

$$\textbf{lista\_de\_sentencias} \rightarrow sentencia\ lista\_de\_sentencias\ \{lista\_de\_sentencias.skip = unir(sentencia.skip, lista\_de\_sentencias_1.skip);\} \mid \xi\ \{lista\_de\_sentencias.skip = lista\_vacia();\}$$

sentencia $\rightarrow$ $variable = expresion;$ $\left\{\begin{array}{c} anadir\_inst(variable.nombre|| := ||expresion.nombre;); \\ sentencia.skip = lista\_vacia(); \end{array}\right\}$

| **if** $expresion$ **then** $\{M\ lista\_de\_sentencias\}M;$ $\left\{\begin{array}{c} completar(expresion.true, M_1.ref); \\ completar(expresion.false, M_2.ref); \\ sentencia.skip = lista\_de\_sentencias.skip \end{array}\right\}$

| **while** $M\ expresion\{M\ lista\_de\_sentenciasM\};$ $\left\{\begin{array}{c} completar(expresion.true, M_2.ref); \\ completar(expresion.false, M_3.ref + 1); \\ anadir\_inst(goto||M_1.ref); \\ completar(lista\_sentencias.skip, M_1.ref); \\ sentencia.skip := lista\_vacia(); \end{array}\right\}$

| **do** M { $lista\_de\_sentencias$} **until** $M\ expresion$ **else** M { $lista\_de\_sentencias$} ; $\left\{\begin{array}{c} completar(expresion.true, M_1.ref); \\ completar(expresion.false, M_3.ref); \\ completar(lista\_sentencias_1.skip, M_2.ref); \\ sentencia.skip := lista\_vacia(); \end{array}\right\}$

| **skip if** $expresion\ M;$ $\left\{\begin{array}{c} completar(expresion.false, M.ref); \\ sentencia.skip = expresion.true; \end{array}\right\}$

| **read** $(variable);$ $\left\{\begin{array}{c} anadir\_inst(read||variable.nombre||;); \\ sentencia.skip = lista\_vacia(); \end{array}\right\}$

| **println** $(expresion);$ $\left\{\begin{array}{c} anadir\_inst(write||expresion.nombre||;); \\ anadir\_inst(writeln;); \\ sentencia.skip = lista\_vacia(); \end{array}\right\}$

variable $\rightarrow$ id $\{variable.nombre = id.nombre;\}$

$$\mathbf{M} \rightarrow \xi\ \{M.ref = obtenref();\}$$

$$\mathbf{expresion} \rightarrow expresion\ \mathbf{or}\quad M\ expresion \left\{ \begin{array}{c} completar(expresion_1.false, M.ref); \\ expresion.true = unir(expresion_1.true, expresion_2.true); \\ expresion.false = expresion_2.false; \end{array} \right\}$$

$$\mid expresion\ \mathbf{and}\ M\ expresion \left\{ \begin{array}{c} completar(expresion_1.true, M.ref); \\ expresion.true = expresion_2.true; \\ expresion.false = unir(expresion_1.false, expresion_2.false); \end{array} \right\}$$

$$\mid \mathbf{not}\ expresion \quad \left\{ \begin{array}{l} expresion.true = expresion_1.false; \\ expresion.false = expresion_1.true; \end{array} \right\}$$

$$\mid expresion == expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| == ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\}$$

$$\mid expresion > \quad expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| > ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\}$$

$$
\begin{aligned}
&| \; expresion < \; expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| < ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\} \\
&| \; expresion \geq \; expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| \geq ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\} \\
&| \; expresion \; \leq expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| \leq ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\} \\
&| \; expresion \; \neq expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| \neq ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\} \\
&| \; expresion + \; expresion \left\{ \begin{array}{c} expresion.nombre = nuevo\_id(); \\ expresion.true = vacio(); \\ expresion.false = vacio(); \\ anadir\_inst(expresion.nombre|| := ||expresion_1.nombre|| + ||expresion_2.nombre||;); \end{array} \right\}
\end{aligned}
$$

$$| \; expresion - expresion \left\{ \begin{array}{c} expresion.nombre = nuevo\_id(); \\ expresion.true = vacio(); \\ expresion.false = vacio(); \\ anadir\_inst(expresion.nombre|| := ||expresion_1.nombre|| - ||expresion_2.nombre||;); \end{array} \right\}$$

$$| \; expresion * \; expresion \left\{ \begin{array}{c} expresion.nombre = nuevo\_id(); \\ expresion.true = vacio(); \\ expresion.false = vacio(); \\ anadir\_inst(expresion.nombre|| := ||expresion_1.nombre|| * ||expresion_2.nombre||;); \end{array} \right\}$$

$$| \; expresion / \; expresion \left\{ \begin{array}{c} expresion.nombre = nuevo\_id(); \\ expresion.true = vacio(); \\ expresion.false = vacio(); \\ anadir\_inst(expresion.nombre|| := ||expresion_1.nombre||/||expresion_2.nombre||;); \end{array} \right\}$$

$$| \; id \quad \left\{ \begin{array}{c} expresion.nombre = id.nombre; \\ expresion.true = vacio(); \\ expresion.false = vacio(); \end{array} \right\}$$

$$| \; n\_entero \quad \left\{ \begin{array}{c} expresion.nombre = entero.nombre; \\ expresion.true = vacio(); \\ expresion.false = vacio(); \end{array} \right\}$$

$$| \; n\_real \quad \left\{ \begin{array}{c} expresion.nombre = real.nombre; \\ expresion.true = vacio(); \\ expresion.false = vacio(); \end{array} \right\}$$

$$| \; (expresion) \quad \left\{ \begin{array}{c} expresion.nombre = expresion_1.nombre; \\ expresion.true = expresion_1.true; \\ expresion.false = expresion_1.false; \end{array} \right\}$$

8.1.5 Casos de prueba

Los casos de prueba para comprobar el correcto funcionamiento de la implementación de la traducción de expresiones booleanas puede encontrarse en dos ficheros:

1. pruebaBooleanos.in: este fichero contiene una serie de pruebas correctas para comprobar su buen funcionamiento.
2. pruebaBooleanosMala.in: este fichero contiene pruebas con fallos para asegurar que la implementación realizada los detecta.

8.2 Estructura de control for

8.2.1 Fragmento añadido a la gramática

sentencia → **for** id **from** expresion **to** expresion **do** { lista_de_sentencias };

8.2.2 Gramática resultante

Esta gramática consiste en la resultante de la implementación de la traducción de expresiones booleanas con el añadido del subapartado anterior:

programa → **program id**
bloque;

bloque → { declaraciones
decl_de_subprogs
lista_de_sentencias
}

delcaraciones → **var** lista_de_ident : tipo ; declaraciones
| ξ

lista_de_ident → **id** resto_lista_id

resto_lista_id → **,** **id** resto_lista_id
| ξ

tipo → **integer** | **float**

decl_de_subprogs → decl_de_subprograma decl_de_subprogs
| ξ
decl_de_subprograma → **procedure** **id** argumentos
bloque;

argumentos → (lista_de_param)
| ξ

lista_de_param → lista_de_ident : clase_par tipo resto_lis_de_param

clase_par → **in** | **out** | **in out**

resto_lis_de_param → ; lista_de_ident : clase_par tipo resto_lis_de_param
| ξ

lista_de_sentencias → sentencia lista_de_sentencias
| ξ

sentencia → variable = expresion ;
| **if** expresion **then** { lista_de_sentencias };
| **for** id **from** expresion **to** expresion **do** { lista_de_sentencias };
| **while** expresion { lista_de_sentencias };
| **do** { lista_de_sentencias } **until** expresion **else** { lista_de_sentencias };
| **skip if** expresion ;
| **read** (variable) ;
| **println** (expresion) ;

variable → *id*

expresion → expresion **or** expresion
| expresion **and** expresion
| **not** expresion
| expresion == expresion
| expresion > expresion
| expresion < expresion
| expresion >= expresion
| expresion <= expresion
| expresion / = expresion
| expresion + expresion
| expresion − expresion
| expresion ∗ expresion
| expresion / expresion
| **id**
| **num_entero**
| **num_real**
| (expresion)

8.2.3 Fragmento añadido al ETDS

$$\textbf{sentencia} \rightarrow \textbf{for}\ \text{id}\ \textbf{from}\ \text{expresion}\ \textbf{to}\ \text{expresion}\ \textbf{do}\ \text{M}\ \{ \begin{Bmatrix} anadir\_inst(id.nombre|| := ||expresion_1.nombre||;); \\ anadir\_inst(if|| := ||id.nombre|| > ||expresion_2.nombre||goto); \end{Bmatrix}$$
$$lista\_de\_sentencias\ \}; \begin{Bmatrix} anadir\_inst(id.nombre|| := ||id.nombre|| + 1||;); \\ anadir\_inst(goto||M.ref + 2||;); \\ completar(obtenref(), M.ref + 2); \end{Bmatrix}$$

8.2.4 ETDS resultante

El ETDS se compone del ETDS resultante del añadido de las expresiones booleanas, con el añadido mencionado en el subapartado anterior:

programa $\rightarrow$ $program\ id\ \{anadir\_inst(prog||id.nombre||;);\}$
$bloque\ ;\ \{anadir\_inst(halt;);escribir();\}$

bloque $\rightarrow$ $\{declaraciones\ decl\_de\_subprogs\ lista\_de\_sentencias\}$

declaraciones $\rightarrow$ $var\ lista\_de\_ident\ :\ tipo;\ \{anadir\_declaracion(lista\_de\_ident.lnom,\ tipo.clase);\}\ declaraciones$
$| \xi$

lista_de_ident $\rightarrow$ $id\ resto\_lista\_id\ \{lista\_de\_ident.lnom = anadir(resto\_lista\_id.lnom,\ id.nombre);\}$

resto_lista_id $\rightarrow$ $,id\ resto\_lista\_id\ \{resto\_lista\_id.lnom = anadir(resto\_lista\_id_1.lnom,\ id.nombre);\}$
$| \xi\ \{resto\_lista\_id.lnom = lista\_vacia();\}$

tipo $\rightarrow$ $integer\ \{tipo.clase = "int";\}$
$| float\ \{tipo.clase = "real";\}$

$$\begin{aligned}
\textbf{decl_de_subprogs} &\rightarrow decl_de_subprograma\ decl_de_subprogs \\
&\mid \xi \\
\\
\textbf{decl_de_subprograma} &\rightarrow procedure\ id\ \{anadir_inst(proc||id.nombre||;);\}\ argumentos\ bloque;\{anadir_inst(endproc;);\} \\
\\
\textbf{argumentos} &\rightarrow (lista_de_param) \\
&\mid \xi \\
\\
\textbf{lista_de_param} &\rightarrow lista_de_ident\ :\ clase_par\ tipo\ \{anadir_declaracion(lista_de_ident.lnom, clase_par.tipo||_||tipo.clase);\} \\
&\quad resto_lis_de_param \\
\\
\textbf{clase_par} &\rightarrow in\ \{clase_par.tipo = "val";\} \\
&\mid out\{clase_par.tipo = "ref";\} \\
&\mid in\ out\ \{clase_par.tipo = "ref";\} \\
\\
\textbf{resto_lis_de_param} &\rightarrow ;\ lista_de_ident\ :\ clase_par\ tipo\ \{anadir_declaracion(lista_de_ident.lnom, clase_par.tipo||_||tipo.clase);\} \\
&\quad resto_lis_de_param \\
&\mid \xi \\
\\
\textbf{lista_de_sentencias} &\rightarrow sentencia\ lista_de_sentencias\ \{lista_de_sentencias.skip = unir(sentencia.skip, lista_de_sentencias_1.skip);\} \\
&\mid \xi\ \{lista_de_sentencias.skip = lista_vacia();\}
\end{aligned}$$

sentencia $\rightarrow$ $variable = expresion;$ $\left\{\begin{array}{c} anadir\_inst(variable.nombre|| := ||expresion.nombre;); \\ sentencia.skip = lista\_vacia(); \end{array}\right\}$

| **if** $expresion$ **then** $\{M\ lista\_de\_sentencias\}M;$ $\left\{\begin{array}{c} completar(expresion.true, M_1.ref); \\ completar(expresion.false, M_2.ref); \\ sentencia.skip = lista\_de\_sentencias.skip \end{array}\right\}$

| **for** id **from** expresion **to** expresion **do** M { $\left\{\begin{array}{c} anadir\_inst(id.nombre|| := ||expresion_1.nombre||;); \\ anadir\_inst(if|| := ||id.nombre|| > ||expresion_2.nombre||goto); \end{array}\right\}$

$lista\_de\_sentencias$ }; $\left\{\begin{array}{c} anadir\_inst(id.nombre|| := ||id.nombre|| + 1||;); \\ anadir\_inst(goto||M.ref + 2||;); \\ completar(obtenref(), M.ref + 2); \end{array}\right\}$

| **while** $M\ expresion\{M\ lista\_de\_sentenciasM\};$ $\left\{\begin{array}{c} completar(expresion.true, M_2.ref); \\ completar(expresion.false, M_3.ref + 1); \\ anadir\_inst(goto||M_1.ref); \\ completar(lista\_sentencias.skip, M_1.ref); \\ sentencia.skip := lista\_vacia(); \end{array}\right\}$

| **do** M { $lista\_de\_sentencias$} **until** $M\ expresion$ **else** M { $lista\_de\_sentencias$} ; $\left\{\begin{array}{c} completar(expresion.true, M_1.ref); \\ completar(expresion.false, M_3.ref); \\ completar(lista\_sentencias_1.skip, M_2.ref); \\ sentencia.skip := lista\_vacia(); \end{array}\right\}$

| **skip if** $expresion\ M;$ $\left\{\begin{array}{c} completar(expresion.false, M.ref); \\ sentencia.skip = expresion.true; \end{array}\right\}$

$$| \textbf{ read } (variable);\ \left\{\begin{array}{c} anadir\_inst(read||variable.nombre||;); \\ sentencia.skip = lista\_vacia(); \end{array}\right\}$$

$$| \textbf{ println } (expresion);\ \left\{\begin{array}{c} anadir\_inst(write||expresion.nombre||;); \\ anadir\_inst(writeln;); \\ sentencia.skip = lista\_vacia(); \end{array}\right\}$$

$$\textbf{variable} \rightarrow id\ \{variable.nombre = id.nombre;\}$$

$$\textbf{M} \rightarrow \xi\ \{M.ref = obtenref();\}$$

$$\textbf{expresion} \rightarrow expresion\ \textbf{or}\quad M\ expresion \left\{ \begin{array}{c} completar(expresion_1.false, M.ref); \\ expresion.true = unir(expresion_1.true, expresion_2.true); \\ expresion.false = expresion_2.false; \end{array} \right\}$$

$$\mid expresion\ \textbf{and}\ M\ expresion \left\{ \begin{array}{c} completar(expresion_1.true, M.ref); \\ expresion.true = expresion_2.true; \\ expresion.false = unir(expresion_1.false, expresion_2.false); \end{array} \right\}$$

$$\mid \textbf{not}\ expresion \quad \left\{ \begin{array}{l} expresion.true = expresion_1.false; \\ expresion.false = expresion_1.true; \end{array} \right\}$$

$$\mid expresion == expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| == ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\}$$

$$\mid expresion > \quad expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| > ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\}$$

$$\mid expresion < \quad expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| < ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\}$$

$$
\begin{aligned}
&| \; expresion \geq \; expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| \geq ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\} \\
&| \; expresion \; \leq expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| \leq ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\} \\
&| \; expresion \; \neq expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| \neq ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\} \\
&| \; expresion + \; expresion \left\{ \begin{array}{c} expresion.nombre = nuevo\_id(); \\ expresion.true = vacio(); \\ expresion.false = vacio(); \\ anadir\_inst(expresion.nombre|| := ||expresion_1.nombre|| + ||expresion_2.nombre||;); \end{array} \right\}
\end{aligned}
$$

$$| \; expresion - expresion \left\{ \begin{array}{c} expresion.nombre = nuevo\_id(); \\ expresion.true = vacio(); \\ expresion.false = vacio(); \\ anadir\_inst(expresion.nombre|| := ||expresion_1.nombre|| - ||expresion_2.nombre||;); \end{array} \right\}$$

$$| \; expresion * \; expresion \left\{ \begin{array}{c} expresion.nombre = nuevo\_id(); \\ expresion.true = vacio(); \\ expresion.false = vacio(); \\ anadir\_inst(expresion.nombre|| := ||expresion_1.nombre|| * ||expresion_2.nombre||;); \end{array} \right\}$$

$$| \; expresion / \; expresion \left\{ \begin{array}{c} expresion.nombre = nuevo\_id(); \\ expresion.true = vacio(); \\ expresion.false = vacio(); \\ anadir\_inst(expresion.nombre|| := ||expresion_1.nombre||/||expresion_2.nombre||;); \end{array} \right\}$$

$$| \; id \quad \left\{ \begin{array}{c} expresion.nombre = id.nombre; \\ expresion.true = vacio(); \\ expresion.false = vacio(); \end{array} \right\}$$

$$| \; n\_entero \quad \left\{ \begin{array}{c} expresion.nombre = entero.nombre; \\ expresion.true = vacio(); \\ expresion.false = vacio(); \end{array} \right\}$$

$$| \; n\_real \quad \left\{ \begin{array}{c} expresion.nombre = real.nombre; \\ expresion.true = vacio(); \\ expresion.false = vacio(); \end{array} \right\}$$

$$| \; (expresion) \quad \left\{ \begin{array}{c} expresion.nombre = expresion_1.nombre; \\ expresion.true = expresion_1.true; \\ expresion.false = expresion_1.false; \end{array} \right\}$$

8.2.5 Casos de prueba

Los casos de prueba para comprobar el correcto funcionamiento de la implementación de la traducción de la nueva estructura de control:

1. **pruebaFor.in**: este fichero contiene una serie de pruebas correctas para comprobar su buen funcionamiento.
2. **pruebaForMala.in**: este fichero contiene pruebas contiene fallos para asegurar que la implementación realizada los detecta.