Nombre del grupo: Grupo 1
Componentes: Iker Pintado, Alvaro Rodrigo, Iker Hidalgo

Abstracciones Funcionales:

obtenref: código → ref_código

Descripción: Dado una estructura de código numerado devuelve la referencia a la instrucción inmediatamente siguiente.

Tipo de los argumentos:

código: Estructura numerada de Strings

añadir_inst: código x inst → código

Descripción: Dada una estructura de código numerado y una inst (String), escribe inst en la siguiente línea de la estructura de código.

Tipo de los argumentos:

código: Estructura numerada de Strings
inst: String

añadir_declaraciones: código x listaids x tipo → código

Descripción: Dados una estructura de código numerada, una lista de identificadores, y un tipo de dato, escribe las declaraciones en la estructura de código.

Tipo de los argumentos:

código: Estructura numerada de Strings
listaids: Lista de Strings
tipo: String

añadir: lista x elemento

Descripción: Dados una lista de elementos y un nuevo elemento, añade al inicio de la lista lista el elemento dado.

Tipo de los argumentos:

lista: Lista de elementos
elemento: cualquiera (booleano o int)

añadir_params: código x listaids x clase x tipo → código

Descripción: Dados una estructura de código numerada, una lista de identificadores, una clase, y un tipo de dato, escribe las declaraciones de los parámetros en la estructura de código.

Tipo de los argumentos:

código: Estructura numerada de Strings
listaids: Lista de Strings
clase: val/ref
tipo: String

Nombre del grupo: Grupo 1
Componentes: Iker Pintado, Alvaro Rodrigo, Iker Hidalgo

unir: lista x lista → lista

Descripción: Dadas dos listas de elemento devuelve una lista que contiene todos los elementos de las dos listas iniciales.

Tipo de los argumentos:

lista: Lista de elementos

completar: lista_refs x ref_código

Descripción: Dados una lista de referencias del código y una referencia del código, completa las instrucciones goto que contienen las referencias de la lista con la referencia dada.

Tipo de los argumentos:

lista_refs: Lista de referencias del código.
ref_código: referencia de código

lista_vacia: → lista

Descripción: Inicializa una lista vacía

inilista: elemento → lista

Descripción: Dado un elemento inicializa una lista que contiene dicho elemento

Tipo de los argumentos:

elemento: elemento de cualquier tipo aceptado

nuevo_id: código → string

Descripción: Genera un nuevo identificador para una expresión

Nombre del grupo: Grupo 1
Componentes: Iker Pintado, Alvaro Rodrigo, Iker Hidalgo

Tabla de atributos definidos

Atributos: (L: Léxico, S: Sintetizado)

Símbolo	Nombre	Tipo	L/S	Descripción
id	nom	string	L	Contiene la cadena de caracteres del id.
expresion	nom	string	S	Contiene la cadena de caracteres de la expresión.
	true	lista de enteros	S	Contiene las referencias a los goto's a rellenar si la expresión es cierta, en caso de que la expresión no sea booleana es una lista vacía.
	false	lista de enteros	S	Contiene las referencias a los gotos a rellenar si la expresión es falsa, en caso de que la expresión no sea booleana es una lista vacía.
bloque	exit	lista de enteros	S	Contiene las referencias de los goto's a completar para terminar la sentencia y simular un exit.
	conti	lista de enteros	S	Contiene las referencias de los goto's a completar para terminar la sentencia y simular un continue.
lista_sentencias	exit	lista de enteros	S	Contiene las referencias de los goto's a completar para terminar la sentencia y simular un exit.
	conti	lista de enteros	S	Contiene las referencias de los goto's a completar para terminar la sentencia y simular un continue.
sentencia	exit	lista de enteros	S	Contiene las referencias de los goto's a completar para terminar la sentencia y simular un exit.
	conti	lista de enteros	S	Contiene las referencias de los goto's a completar para terminar la sentencia y simular un continue.
M	ref	num entero	S	Contiene la referencia a una instrucción de código. Se utiliza para completar los goto.
lista_de_ident	inom	lista de strings	S	Contiene la lista de los identificadores
resto_lista_id	inom	lista de strings	S	Contiene la lista de los identificadores

Nombre del grupo: Grupo 1
Componentes: Iker Pintado, Alvaro Rodrigo, Iker Hidalgo

tipo	clase	string	S	Cadena de carácteres que indica el tipo de numero (int o float)
clase_par	tipo	string	S	Cadena de caracteres que indica el tipo de la clase par (val o ref)
num_entero	nom	num entero	L	Contiene el valor numérico (entero)
num_real	nom	num real	L	Contiene el valor numérico (real)
variable	nom	string	S	Cadena de caracteres con el nombre de variable

Nombre del grupo: Grupo 1
Componentes: Iker Pintado, Alvaro Rodrigo, Iker Hidalgo

ETDS:

M → ξ { M.ref :=obtenref(); }

programa → **def main ( ) :** { añadir_inst(proc main) ;}
bloque_ppl { añadir_inst(halt);}

bloque_ppl → decl_bl **{**
decl_de_subprogs
lista_de_sentencias
}

bloque → **{**
lista_de_sentencias
} {bloque.exit := lista_de_sentencias.exit;
bloque.conti := lista_de_sentencias.conti;}

decl_bl → **let** declaraciones **in**
| ξ

declaraciones → declaraciones **;** lista_de_ident **:** tipo
{añadir_declaraciones(lista_de_ident.lnom, tipo.clase)}

| lista_de_ident **:** tipo
{añadir_declaraciones(lista_de_ident.lnom, tipo.clase)}

lista_de_ident → **id** resto_lista_id
{lista_de_ident.lnom := añadir(resto_lista_id.lnom,id.nom);}

resto_lista_id → **,** **id** resto_lista_id
{resto_lista_id.lnom := añadir(resto_lista_id1.lnom,id.nom);}

| ε {resto_lista_id := lista_vacía(); }

tipo → **integer** {tipo.clase := "int";}

| **float** {tipo.clase := "real";}

decl_de_subprogs → decl_de_subprograma decl_de_subprogs
| ξ

decl_de_subprograma →**def id** {añadir_inst(proc || id.nom);}
argumentos : bloque_ppl {añadir_inst(endproc || id.nom);}

Nombre del grupo: Grupo 1
Componentes: Iker Pintado, Alvaro Rodrigo, Iker Hidalgo

argumentos → **(** lista_de_param **)**
| ξ

lista_de_param → lista_de_ident : clase_par tipo
{ añadir_params(lista_de_ident.lnom, clase_par.tipo, tipo.clase); }
resto_lis_de_param

clase_par → **ε** {clase_par.tipo := "val";}
| **&** {clase_par.tipo := "ref";}

resto_lis_de_param → ; lista_de_ident : clase_par tipo
{ añadir_params(lista_de_ident.lnom, clase_par.tipo, tipo.clase); }
resto_lis_de_param

| ξ

lista_de_sentencias → sentencia lista_de_sentencias
{ lista_de_sentencias.exit := unir(sentencia.exit,lista_de_sentencias.exit) ;
lista_de_sentencias.conti := unir(sentencia.conti,lista_de_sentencias.conti); }

| ξ
{lista_de_sentencias.exit := lista_vacía() ;
lista_de_sentencias.conti := lista_vacía();}

sentencia → variable **=** expresion **;**{añadir_inst(variable.nom || := || expresion.nom);
sentencia.exit := lista_vacía() ;
sentencia.conti := lista_vacía(); }

| **if** expresion **:** M bloque M
{completar(expresion.true,M1.ref);
completar(expresion.false,M2.ref);
sentencia.exit := bloque.exit ;
sentencia.conti := bloque.conti;}

| **forever :** M bloque M
{añadir_inst(goto || M1.ref);
completar(bloque.exit, M2.ref + 1);
sentencia.exit := lista_vacía() ;
sentencia.conti := bloque.conti;}

| **while** M expresion **:** M bloque M
{añadir_inst(goto || M1.ref);}
else: bloque M
{completar(expresion.true, M2.ref);
completar(expresion.false, M3.ref + 1);
completar(bloque1.exit, M3.ref +1);

completar(bloque1.conti, M1.ref);
completar(bloque2.exit, M4.ref);
completar(bloque2.conti, M1.ref);
sentencia.exit := lista_vacia() ;
sentencia.conti := lista_vacia() ;}

| break if expresion M**;** {
completar(expresion.false, M1.ref);
sentencia.exit := expresion.true;
sentencia.conti := lista_vacía();}

| continue;{sentencia.conti := inilista(obtenref());
sentencia.exit := lista_vacia();
añadir_inst(goto); }

| **read (** variable **) ;** {añadir_inst(read || variable.nom);
sentencia.exit := lista_vacía() ;
sentencia.cont := lista_vacía();}

| **println (** expresion **) ;** { añadir_inst(write || expresion.nom) ;
añadir_inst(writeln) ;
sentencia.exit := lista_vacía() ;
sentencia.cont := lista_vacía();}

variable → **id** {variable.nom := id.nom}

expresion → expresion **==** expresion{expresion.nom := “”;
expresion.true := inilista(obtenref());
expresion.false := inilista(obtenref()+1);
añadir_inst(if || expresion1.nom || == || expresion2.nom || goto);
añadir_inst(goto); }

| expresion **>** expresion{expresion.nom := “”;
expresion.true := inilista(obtenref());
expresion.false := inilista(obtenref()+1);
añadir_inst(if || expresion1.nom || > || expresion2.nom || goto);
añadir_inst(goto);}

| expresion **<** expresion{expresion.nom := “”;
expresion.true := inilista(obtenref());
expresion.false := inilista(obtenref()+1);
añadir_inst(if || expresion1.nom || < || expresion2.nom || goto);
añadir_inst(goto);}

| expresion **>=** expresion{expresion.nom := “”;
expresion.true := inilista(obtenref());
expresion.false := inilista(obtenref()+1);
añadir_inst(if || expresion1.nom || >= || expresion2.nom || goto);
añadir_inst(goto);}

```
  | expresion <= expresion{expresion.nom := nuevo_id();
  expresion.true := inilista(obtenref());
expresion.false := inilista(obtenref()+1);
añadir_inst(if || expresion1.nom || <= || expresion2.nom || goto);
añadir_inst(goto); }

  | expresion /= expresion  {expresion.nom := “”;
  expresion.true := inilista(obtenref());
expresion.false := inilista(obtenref()+1);
añadir_inst(if || expresion1.nom || /= || expresion2.nom || goto);
añadir_inst(goto); }

  | expresion + expresion   {expresion.nom := nuevo_id();
añadir_inst(expresion.nom || := || expresion1.nom || + || expresion2.nom);
expresion.true := lista_vacía();
expresion.false := lista_vacía(); }

  | expresion - expresion    {expresion.nom := nuevo_id();
añadir_inst(expresion.nom || := || expresion1.nom || - || expresion2.nom);
expresion.true := lista_vacía();
expresion.false := lista_vacía(); }

  | expresion * expresion    {expresion.nom := nuevo_id();
añadir_inst(expresion.nom || := || expresion1.nom || * || expresion2.nom);
expresion.true := lista_vacía();
expresion.false := lista_vacía(); }

  | expresion / expresion    {expresion.nom := nuevo_id();
añadir_inst(expresion.nom || := || expresion1.nom || / || expresion2.nom);
expresion.true := lista_vacía();
expresion.false := lista_vacía(); }

  | variable       {expresion.nom := variable.nom;
expresion.true := lista_vacía();
expresion.false := lista_vacía();}

  | num_entero   {expresion.nom := num_entero.nom;
expresion.true := lista_vacía();
expresion.false := lista_vacía();}

  | num_real       {expresion.nom := num_real.nom;
expresion.true := lista_vacía();
expresion.false := lista_vacía();}
  | ( expresion )   {expresion.nom := expresion1.nom;
expresion.true := expresion1.true;
expresion.false := expresion1.false;}
```

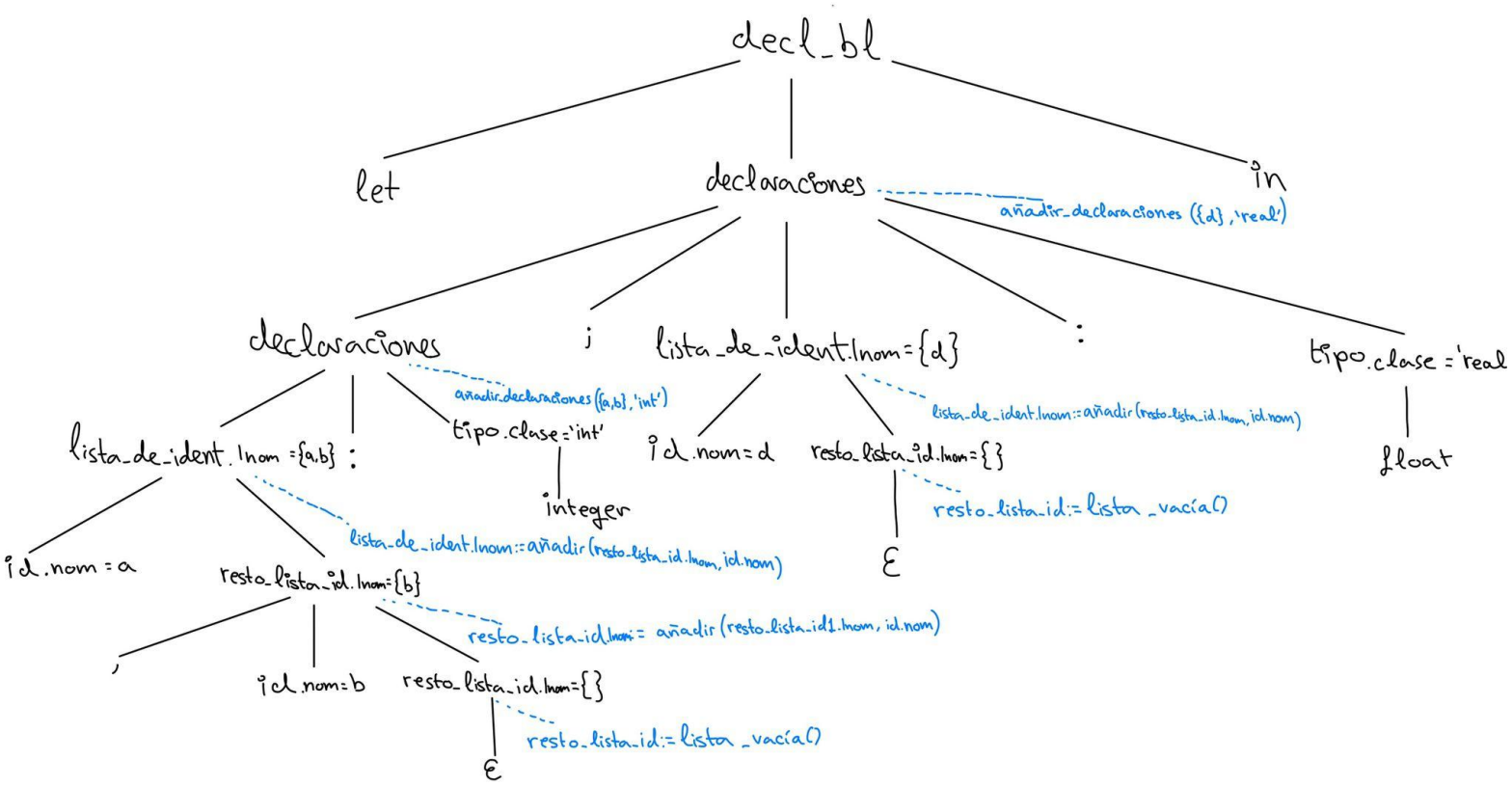
Nombre del grupo: Grupo 1
Componentes: Iker Pintado, Alvaro Rodrigo, Iker Hidalgo

Prueba 1: Árbol decorado y resultado

Programa:

.
.
.
let a,b: integer; d: float in{
.
.
.

Árbol decorado (mejor calidad al final del documento):



Código intermedio generado:

1. .
2. int a;
3. int b;
4. real d;
5. .
6. .
7. .

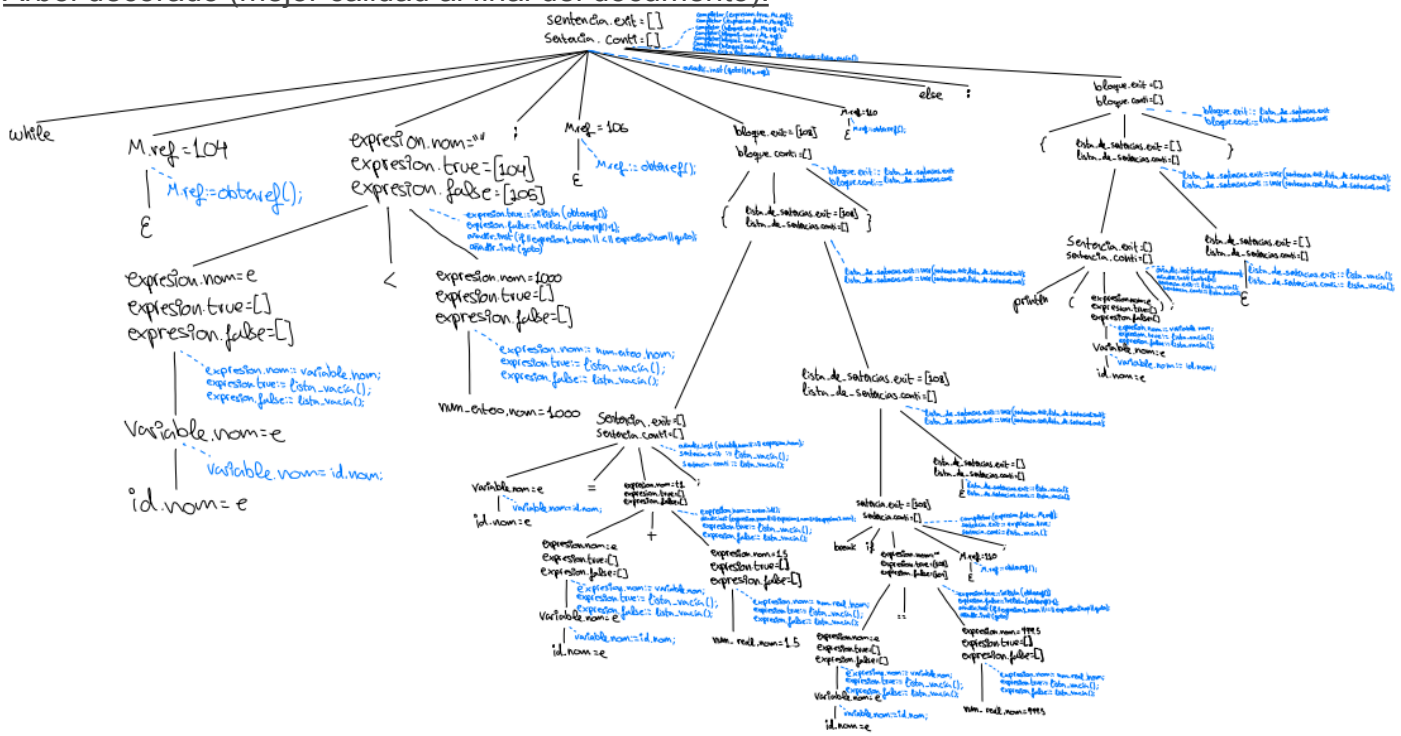
Nombre del grupo: Grupo 1
Componentes: Iker Pintado, Alvaro Rodrigo, Iker Hidalgo

Prueba 2: Árbol decorado y resultado

Programa:

```
.
.
.
  while e < 1000 :{
     e = e +1.5
     break if e == 999.5;
  } else :{
     println(e);
  }
.
.
.
```

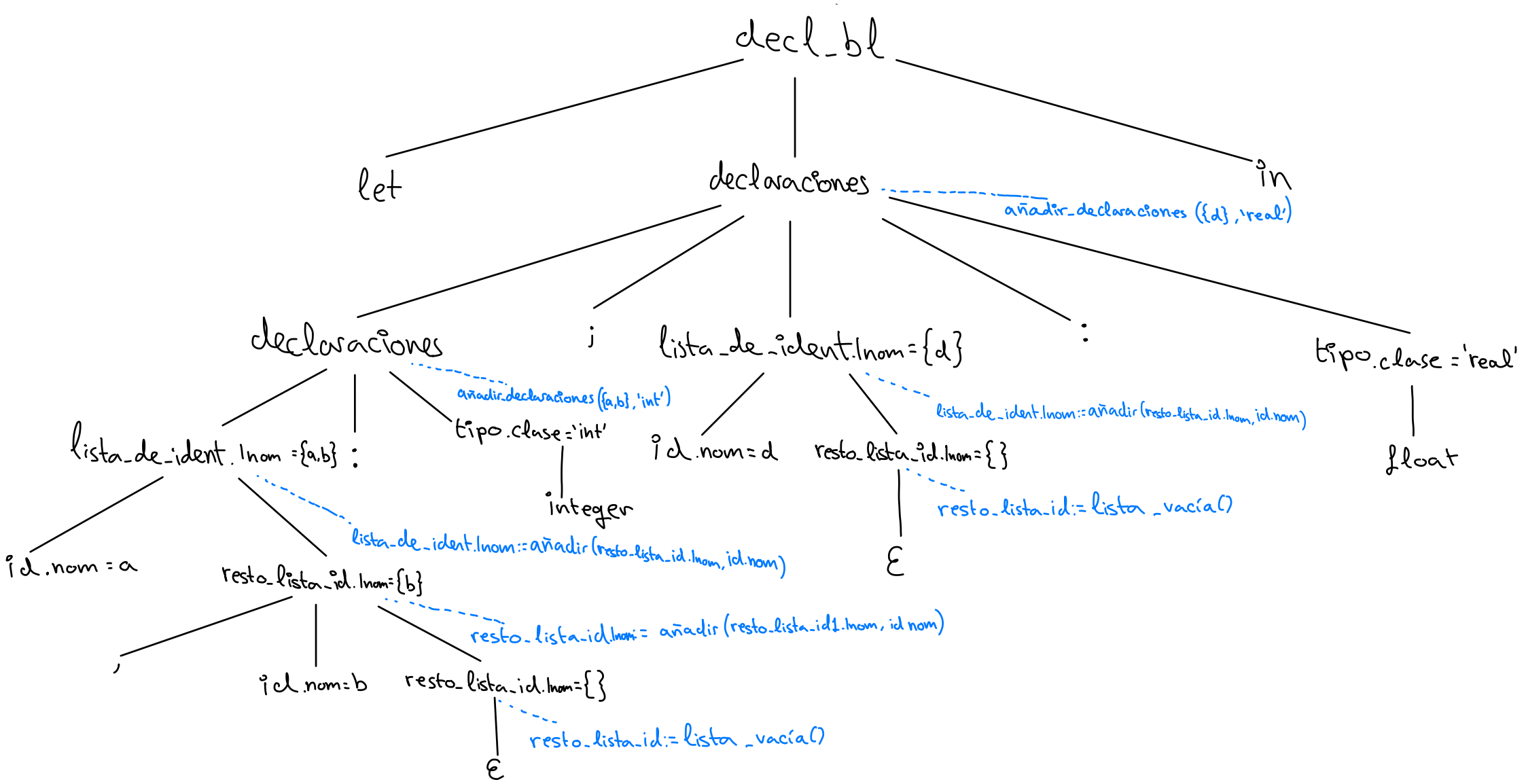
Árbol decorado (mejor calidad al final del documento):

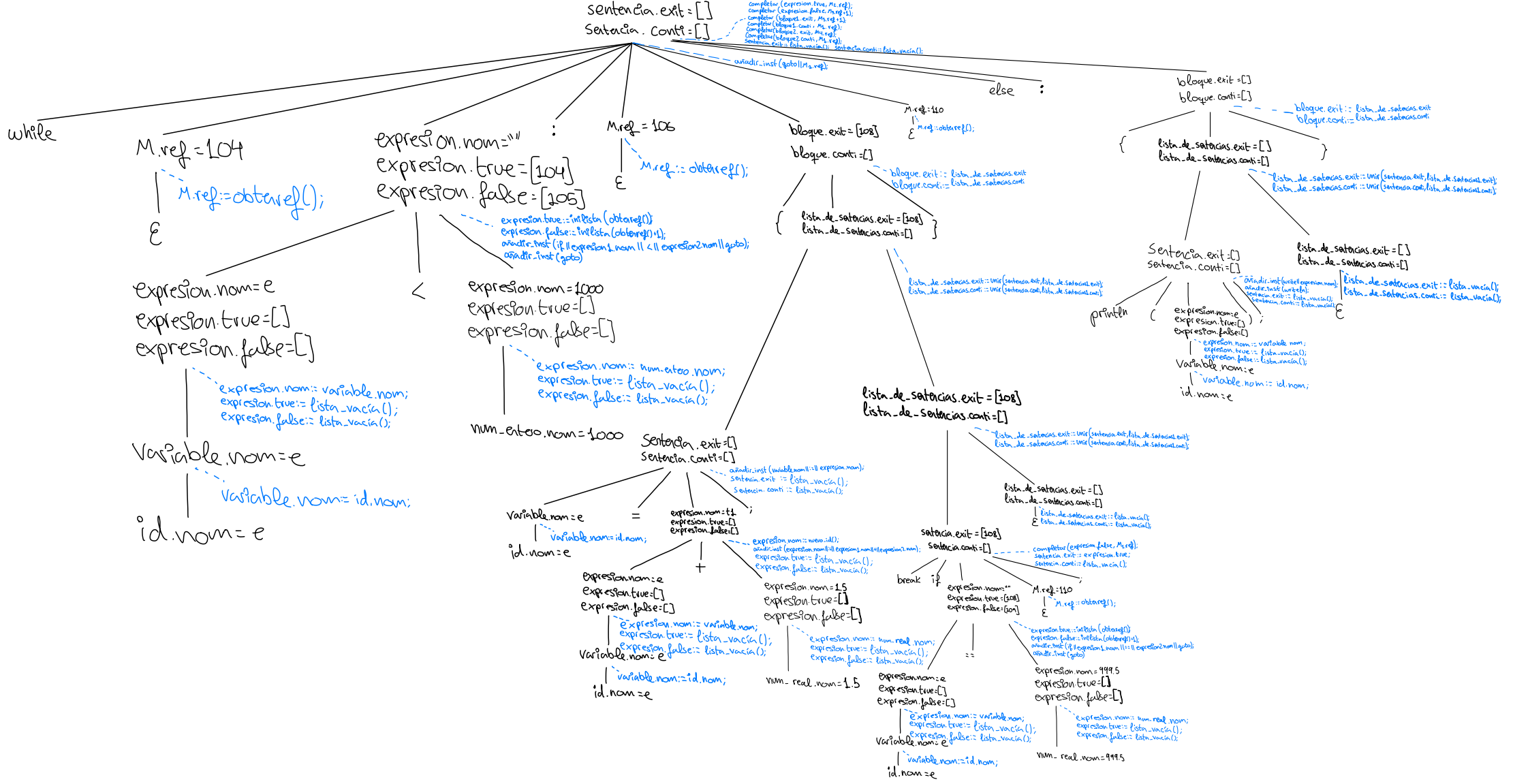


Código intermedio generado:

```
101.  .
102.  .
103.  .
104.  if e <1000 goto 106;
105.  goto 111;
106.  t1 := e + 1.5;
107.  e := t1;
108.  if e == 999.5 goto 111;
```

```
109.  goto 110
110.  goto 104
111.  write e;
112.  writeln;
113.  .
114.  .
115.  .
```


decl_bl
let
declaraciones
añadir_declaraciones ({d}, 'real')
in
declaraciones
;
lista_de_ident.lnom={d}
:
tipo.clase = 'real'
añadir_declaraciones ({a,b}, 'int')
tipo.clase='int'
lista_de_ident.lnom:=añadir(resto_lista_id.lnom, id.nom)
lista_de_ident.lnom = {a,b}
:
id.nom=d
resto_lista_id.lnom={}
float
integer
lista_de_ident.lnom:=añadir(resto_lista_id.lnom, id.nom)
resto_lista_id:= lista_vacía()
id.nom=a
resto_lista_id.lnom={b}
ε
,
resto_lista_id.lnom:= añadir(resto_lista_id1.lnom, id.nom)
id.nom=b
resto_lista_id.lnom={}
resto_lista_id:= lista_vacía()
ε


sentencia.exit = []
sentencia.cont = []
completar(expresion.true, M2.ref);
completar(expresion.false, M3.ref+1);
completar(bloque1.exit, M3.ref+1);
completar(bloque1.cont, M4.ref);
completar(bloque2.exit, M4.ref);
completar(bloque2.cont, M4.ref);
sentencia.exit = lista_vacia(); sentencia.cont = lista_vacia();
añadir_inst(goto||M4.ref)
while
M.ref = 104
M.ref := obtenref();
ε
expresion.nom = ""
expresion.true = [104]
expresion.false = [105]
expresion.true := initLista(obtenref());
expresion.false := tailLista(obtenref()+1);
añadir_inst(if || expresion1.nom || < || expresion2.nom || goto);
añadir_inst(goto)
:
M.ref = 106
M.ref := obtenref();
ε
bloque.exit = [108]
bloque.cont = []
bloque.exit := lista_de_sentencias.exit
bloque.cont := lista_de_sentencias.cont
{
lista_de_sentencias.exit = [108]
lista_de_sentencias.cont = []
}
lista_de_sentencias.exit := unir(sentencia.exit, lista_de_sentencias.exit);
lista_de_sentencias.cont := unir(sentencia.cont, lista_de_sentencias.cont);
else
:
M.ref = 110
M.ref := obtenref();
ε
bloque.exit = []
bloque.cont = []
bloque.exit := lista_de_sentencias.exit
bloque.cont := lista_de_sentencias.cont
{
lista_de_sentencias.exit = []
lista_de_sentencias.cont = []
}
lista_de_sentencias.exit := unir(sentencia.exit, lista_de_sentencias.exit);
lista_de_sentencias.cont := unir(sentencia.cont, lista_de_sentencias.cont);
expresion.nom = e
expresion.true = []
expresion.false = []
expresion.nom := variable.nom;
expresion.true := lista_vacia();
expresion.false := lista_vacia();
variable.nom = e
variable.nom := id.nom;
id.nom = e
<
expresion.nom = 1000
expresion.true = []
expresion.false = []
expresion.nom := num_entero.nom;
expresion.true := lista_vacia();
expresion.false := lista_vacia();
num_entero.nom = 1000
sentencia.exit = []
sentencia.cont = []
añadir_inst(variable.nom || := || expresion.nom);
sentencia.exit := lista_vacia();
sentencia.cont := lista_vacia();
variable.nom = e
variable.nom := id.nom;
id.nom = e
=
expresion.nom = t1
expresion.true = []
expresion.false = []
;
expresion.nom := nuevo.id();
añadir_inst(expresion.nom || := || expresion1.nom || + || expresion2.nom);
expresion.true := lista_vacia();
expresion.false := lista_vacia();
expresion.nom = e
expresion.true = []
expresion.false = []
expresion.nom := variable.nom;
expresion.true := lista_vacia();
expresion.false := lista_vacia();
variable.nom = e
variable.nom := id.nom;
id.nom = e
+
expresion.nom = 1.5
expresion.true = []
expresion.false = []
expresion.nom := num_real.nom;
expresion.true := lista_vacia();
expresion.false := lista_vacia();
num_real.nom = 1.5
println
(
expresion.nom = e
expresion.true = []
expresion.false = []
)
;
expresion.nom := variable.nom;
expresion.true := lista_vacia();
expresion.false := lista_vacia();
variable.nom = e
variable.nom := id.nom;
id.nom = e
sentencia.exit = []
sentencia.cont = []
añadir_inst(println || expresion.nom);
añadir_inst(writeln);
sentencia.exit := lista_vacia();
sentencia.cont := lista_vacia();
lista_de_sentencias.exit = []
lista_de_sentencias.cont = []
lista_de_sentencias.exit := lista_vacia();
lista_de_sentencias.cont := lista_vacia();
ε
lista_de_sentencias.exit = [108]
lista_de_sentencias.cont = []
lista_de_sentencias.exit := unir(sentencia.exit, lista_de_sentencias.exit);
lista_de_sentencias.cont := unir(sentencia.cont, lista_de_sentencias.cont);
lista_de_sentencias.exit = []
lista_de_sentencias.cont = []
ε
lista_de_sentencias.exit := lista_vacia();
lista_de_sentencias.cont := lista_vacia();
sentencia.exit = [108]
sentencia.cont = []
completar(expresion.false, M1.ref);
sentencia.exit := expresion.true;
sentencia.cont := lista_vacia();
break
if
expresion.nom = ""
expresion.true = [108]
expresion.false = [107]
==
expresion.nom = e
expresion.true = []
expresion.false = []
expresion.nom := variable.nom;
expresion.true := lista_vacia();
expresion.false := lista_vacia();
variable.nom = e
variable.nom := id.nom;
id.nom = e
M.ref = 110
M.ref := obtenref();
ε
expresion.true := initLista(obtenref());
expresion.false := tailLista(obtenref()+1);
añadir_inst(if || expresion1.nom || != || expresion2.nom || goto);
añadir_inst(goto)
expresion.nom = 999.5
expresion.true = []
expresion.false = []
expresion.nom := num_real.nom;
expresion.true := lista_vacia();
expresion.false := lista_vacia();
num_real.nom = 999.5
;