Nombre del grupo: Grupo2
Componentes: Imanol Conde González, Aitor González González, Victor Ruiz Clavijo, Melisa Fernandez.

Esquema de Traducción Dirigido por la Sintaxis

Abstracciones Funcionales:

añadirInstruccion: código x inst → código

Descripción: Dada una estructura de código numerada 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

añadirDeclaraciones: codigo x lista x tipo -> codigo

Descripción: Por cada nombre de la lista de entrada, empezando por el primero y hasta el último añade una instrucción de esta forma: **tipo nombre ;.** Se verificarían los tipos del lenguaje fuente integer float y se mapearian a sus analogos int real de código intermedio.

Tipo de los argumentos:
lista: Estructura numerada de Strings
tipo: String

añadirParametros: codigo x lista x tipo x clasePar -> codigo

Descripción: Por cada nombre de la lista de entrada, empezando por el primero y hasta el ultimo añade una instrucción dependiendo del tipo, val o ref, llamando a la función añadir declaraciones.

Tipo de los argumentos:
lista: Estructura númerada de Strings
tipo: String con el tipo del lenguaje fuente
clasePar: String

añadir: lista x nombre -> lista

Descripción: Añade el nombre al comienzo de la lista de entrada y devuelve la nueva lista.
Tipo de los argumentos:
lista: Estructura numerada de Strings
nombre: String

inilista: nombre -> lista

Descripción: Crea una lista con el nombre y devuelve la nueva lista.
Tipo de los argumentos:

nombre: String

lista_vacia: void -> lista

Descripción: Crea una lista vacia de enteros y la devuelve.

completarInstrucciones: codigo x lista x etiqueta -> codigo

Descripción: Completa con la marca etiqueta las instrucciones referenciadas por la lista.
Tipo de los argumentos:
codigo: Estructura numerada de strings
lista: Estructura numerada de enteros
etiqueta: String

nuevo_id: void -> identificador

Descripción: devuelve un nuevo identificador de código intermedio: _t1, _t2, _t3…

escribir: codigo -> fichero

Descripcion: dado un array con las instrucciones de codigo escribe dichas instrucciones en fichero, por defecto la salida estandar.

Tipo de los argumentos:
codigo: Estructura numerada de strings

unir: lista1 x lista2 -> lista

Descripcion: dadas dos listas con referencias a intrucciones las une en una unica lista.

Tipo de los argumentos:
lista1: Estructura numerada de enteros
lista2: Estructura numerada de enteros

Atributos: (L: Léxico, S: Sintetizado)

Símbolo	Nombre	Tipo	L/S	Descripción
id	nombre	string	L	Contiene la cadena de caracteres del id.
num_entero	nombre	string	L	Contiene la cadena de caracteres del integer.

num_real	nombre	string	L	Contiene la cadena de caracteres del float.
tipo	clase	string	S	Contiene el tipo de dato de una variable o argumento.
clase_par	nombre	string	S	Contiene el nombre de la clase de parametro que puede ser “val” si es por valor o “ref” si es por referencia.
expresion	nombre	string	S	Contiene la cadena de carácteres de expresión que puede ser el nombre de una variable de código intermedio o el nombre de una variable de código fuente.
lista_de_ident	lnom	lista de string	S	Lista de strings con los nombres de la lista de ids.
resto_lista_ident	lnom	lista de string	S	Lista de strings con los nombres de la lista de ids locales a este no terminal.
sentencia	breaks	lista de integer	S	Contiene la lista de referencias a los goto generados por los breaks que aun no han sido completados.
sentencia	continues	lista de integer	S	Contiene la lista de referencias a los goto generados por los.continues que aun no han sido completados.
expresion	trues	lista de integer	S	Contiene la referencia a las instrucciones goto del caso verdadero.
expresion	falses	lista de integer	S	Contiene la referencia a las instrucciones goto del caso falso.
variable	nombre	string	S	Contiene la cadena de caracteres de variable.
bloque	breaks	lista de integer	S	Contiene las referencias a los gotos.breakssin comletar de la lista de sentencias que contiene.
bloque	continues	lista de integer	S	Contiene las referencias a los gotos.continues sin comletar de la

				lista de sentencias que contiene.
M	ref	integer	S	Contiene la dirección de linea a la que hace referencia esta marca.
N	next	lista de integer	S	Contiene la referencia al goto escrito por este no terminal.

ETDS: (Poned las acciones de otro color para que destaquen. Diferente al negro.)

programa → **def main ( ) :**
{añadirInstruccion(“proc main ;”)}
bloque_ppl
{añadirInstruccion(“halt ;”);
escribir();}

bloque_ppl → decl_bl **{**
decl_de_subprogs
lista_de_sentencias
}

bloque → **{**
lista_de_sentencias
}
{bloque.breaks= lista_de_sentencias.break;
bloque.continues = lista_de_sentencias.continue}

decl_bl → **let** declaraciones **in**

| ξ

declaraciones → declaraciones **;** lista_de_ident **:** tipo

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

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

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

| ε {resto_lista_id.lnom := {};}

tipo → **integer** {tipo.clase := “integer”}

| **float** {tipo.clase := “float”}

decl_de_subprogs → decl_de_subprograma decl_de_subprogs

| ξ {}

decl_de_subprograma → **def id** {añadirInstruccion(“proc” + id.nombre + “ ;”)}
argumentos : bloque_ppl
{añadirInstruccion(“endproc” + id.nombre + “ ;”)}

argumentos → **(** lista_de_param **)**

| ξ

lista_de_param → lista_de_ident : clase_par tipo
{añadirParametros(lista_de_ident.lnom,
clase_par.nombre,
tipo.clase)}
resto_lis_de_param

clase_par → ε {clase_par.nombre = “val”}
| & {clase_par.nombre = “ref”}

resto_lis_de_param → ; lista_de_ident : clase_par tipo
{añadirParametros(lista_de_ident.lnom, tipo.clase
clase_par.nombre)}
resto_lis_de_param
| ξ

lista_de_sentencias → sentencia lista_de_sentencias

{lista_de_sentencias.breaks:=
unir(lista_de_sentencias1.break,
sentencia.break);
lista_de_sentencias.continues =
unir(lista_de_sentencias1.continue,
sentencia.break);}

| ξ {lista_de_sentencias.breaks= lista_vacia();
lista_de_sentencias.continues = lista_vacia();}

sentencia → variable **=** expresion;
{añadirInstruccion(variable.nombre || := || expresion.nombre);
sentencia.breaks= lista_vacia();
sentencia.continues = lista_vacia();}

| **if** expresion : M bloque M
{completarInstrucciones(expresion.true,M1.ref);
completarInstrucciones(expresion.false, M2.ref);
sentencia.breaks= bloque.break;
sentencia.continues = bloque.continue;}

| **forever :** M bloque M
{completarInstrucciones(bloque.break, M2.ref + 1);
añadirInstruccion(goto || M1.ref);
sentencia.breaks= lista_vacia();
sentencia.continues = lista_vacia();}

| **while** M expresion **:** M bloque N **else:** M bloque M
{completarInstrucciones(expresion.true,M2.ref);
completarInstrucciones(expresion.false, M3.ref);

completarInstrucciones(bloque1.continue, M1.ref);
completarInstrucciones(bloque2.continue, M1.ref);
completarInstrucciones(bloque1.break, M3.ref);
completarInstrucciones(bloque2.break, M4.ref);
completarInstrucciones(N.next, M1.ref);
sentencia.breaks= lista_vacia();
sentencia.continues = lista_vacia();}

|.breaksif expresion M**;**
{ completarInstrucciones(expresion.false, M1.ref);
sentencia.breaks := inilista(expresion.trues);
sentencia.continues = lista_vacia();}

|.continues ;
{ sentencia.continues := inilista(obtenRef());
añadirInstruccion("goto ");
sentencia.breaks= lista_vacia();}

| **read (** variable **) ;**
{añadirInstruccion("read" + variable.nombre + ";")
sentencia.continues = lista_vacia();
sentencia.breaks= lista_vacia();}

| **println (** expresion **) ;**
{añadirInstruccion("write" + expresion.nombre + ";");
añadirInstruccion("writeln ;");
sentencia.continues = lista_vacia();
sentencia.breaks= lista_vacia();}

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

expresion → expresion **==** expresión
{expresion.nombre = "";
expresion.trues = inilista(obtenref());
expresion.falses = inilista(obtenref() + 1);
añadirInstruccion(if || expresion1.nombre || = ||expresion2 .nombre || goto);
añadirInstruccion(goto)}
| expresion **>** expresion
{expresion.nombre = "";
expresion.trues = inilista(obtenref());
expresion.falses = inilista(obtenref() + 1);
añadirInstruccion(if || expresion1.nombre || > ||expresion2 .nombre || goto);
añadirInstruccion(goto)}
| expresion **<** expresion
{expresion.nombre = "";
expresion.trues = inilista(obtenref());

```
  expresion.falses = inilista(obtenref() + 1);
 añadirInstruccion(if || expresion1.nombre || < ||expresion2 .nombre || goto );
 añadirInstruccion(goto)}
   | expresion >= expresion
{expresion.nombre = “”;
expresion.trues = inilista(obtenref());
  expresion.falses = inilista(obtenref() + 1);
 añadirInstruccion(if || expresion1.nombre || >= ||expresion2 .nombre || goto );
 añadirInstruccion(goto)}
   | expresion <= expresion
{expresion.nombre = “”;
  expresion.trues = inilista(obtenref());
  expresion.falses = inilista(obtenref() + 1);
 añadirInstruccion(if || expresion1.nombre || <= ||expresion2 .nombre || goto );
 añadirInstruccion(goto)}
   | expresion /= expresion
{expresion.nombre = “”;
 expresion.trues = inilista(obtenref());
  expresion.falses = inilista(obtenref() + 1);
 añadirInstruccion(if || expresion1.nombre || /= ||expresion2 .nombre || goto );
 añadirInstruccion(goto)}
   | expresion + expresion
     {expresion.trues = lista_vacia();
     expresion.falses = lista_vacia();
     expresion.nombre = nuevo_id();
      añadirInstruccion(expresion.nombre || := || expresion1.nombre || + ||
      expresion2.nombre);}

   | expresion - expresion
     {expresion.trues = lista_vacia();
     expresion.falses = lista_vacia();
     expresion.nombre = nuevo_id();
      añadirInstruccion(expresion.nombre || := || expresion1.nombre || - ||
      expresion2.nombre);}

   | expresion * expresion
     {expresion.trues = lista_vacia();
     expresion.falses = lista_vacia();
     expresion.nombre = nuevo_id();
      añadirInstruccion(expresion.nombre || := || expresion1.nombre || * ||
      expresion2.nombre);}

   | expresion / expresion
     {expresion.trues = lista_vacia();
     expresion.falses = lista_vacia();
     expresion.nombre = nuevo_id();
      añadirInstruccion(expresion.nombre || := || expresion1.nombre || / ||
```

```
                    expresion2.nombre);}

                | variable
                  {expresion.trues = lista_vacia();
                  expresion.falses = lista_vacia();
                   expresion.nombre := variable.nombre}
                | num_entero
                  {expresion.trues = lista_vacia();
                  expresion.falses = lista_vacia();
                   expresion.nombre := num_entero.nombre}
                | num_real
                  {expresion.trues = lista_vacia();
                  expresion.falses = lista_vacia();
                  expresion.nombre := num_real.nombre}
              | ( expresion )
                  {expresion.trues := expresion1.true;
                  expresion.falses := expresion1.false;
                  expresion.nombre := expresión1.nombre}

M → ξ {M.ref := obtenref();}

N → ξ {N.next := inilista(obtenref());
       añadirInstruccion("goto");}
```

Para comprobar que el ETDS funciona correctamente hemos realizado pruebas con dos códigos fuente realizando el arbol y obteniendo así el codigo intermedio:
https://drive.google.com/file/d/1KkC4w6YAFnFtVSUp9erL557yhqar0ash/view?usp=sharing