2) Análisis Semántico. El lenguaje S (2 puntos)

S es un lenguaje de programación basado en el uso de subprogramas. El programa S está compuesto por una sección llamada `PERFILES` donde se define un conjunto de perfiles (o cabeceras) de subprogramas seguido de una sección llamada `PROGRAMA` conteniendo una secuencia de instrucciones. El subprograma puede ser una función (devuelve 1 parámetro) o una acción (no devuelve nada). El lenguaje S define dos tipos de datos: tipo entero (`ENTERO`) y tipo vector de enteros (`VECTOR`). Las variables del programa (`PROGRAMA`) se declaran implícitamente: la declaración de la variable se vincula a la ocurrencia de ésta en alguna asignación como variable asignada (p.e. `v` se declara en la asignación `v = [3,2,12,4,5]`). El lenguaje S dispone de 4 tipos de instrucciones: (1) asignación simple, (2) iteración, (3) condicional y (4) llamada a subprograma. Las expresiones enteras incluyen (aparte de constantes y operaciones típicas) las componentes de un vector (p.e. `vector[i]`) y las llamadas a funciones que devuelvan el tipo entero (p.e. `buscar_posicion_menor(v,i)`). Las expresiones de tipo vector se reducen a constantes (p.e. `[3,2,12,4,5]`) y a llamadas a funciones que devuelvan un vector (p.e. `filtrar_componentes_pares(v)`). El lenguaje S predefine una función llamada `ultimo_indice` para calcular la última posición de un vector (p.e. `ultimo_indice(v)`). Los vectores tienen 1 como primera posición. El lenguaje S interpreta los vectores como parámetros de entrada y salida.

Ejemplo de programa S:

```
PERFILES
   FUNCION buscar_posicion_menor(VECTOR,ENTERO) DEVUELVE (ENTERO);
   ACCION intercambiar(VECTOR, ENTERO, ENTERO);
   FUNCION filtrar_componentes_pares(VECTOR) DEVUELVE (VECTOR);
FPERFILES

PROGRAMA
  v = [3,2,12,4,5];
  q = ultimo_indice(v);
  PARA i en 1..q-1 HACER
     pos = buscar_posicion_menor(i,v);
     intercambiar(i,pos);
  FPARA
  v = filtrar_componentes_pares(v);
FPROGRAMA
```

Suponga la siguiente gramática para el lenguaje S:

```
programa: perfiles seccion_principal

perfiles: PERFILES (perfil)* FPERFILES

seccion_principal: PROGRAMA instrucciones FPROGRAMA

perfil: (perfil_funcion | perfil_accion) PUNTOYCOMA

perfil_funcion: FUNCION IDENT PARENTESISABIERTO (parámetros)? PARENTESISCERRADO
                     DEVUELVE PARENTESISABIERTO parametro PARENTESISCERRADO

perfil_accion: ACCION IDENT PARENTESISABIERTO (parámetros)? PARENTESISCERRADO

parametro: ENTERO | VECTOR

parametros: parametro COMA parametros
   | parametro
```

```
instrucciones: instruccion instrucciones | instruccion

instruccion: asignacion PUNTOYCOMA | iteracion
           | condicional | llamada_subprograma PUNTOYCOMA;

asignacion:  IDENT ASIGNACION expresion

iteracion: PARA IDENT EN rango HACER instrucciones FPARA

rango: expresion_entera RANGO expresion_entera

condicional: SI condicion ENTONCES instrucciones (SINO instrucciones)? FSI

llamada_subprograma: IDENT PARENTESISABIERTO (expresiones)? PARENTESISCERRADO
   | ULTIMOINDICE PARENTESISABIERTO expresion_vector PARENTESISCERRADO

expresiones: expresion COMA expresiones
   | expresion

expresion: expresion_entera  | expresion_vector

expresion_entera: expresion_entera (MAS|MENOS|POR|DIVISION) expresion_entera
   | PARENTESISABIERTO expresion_entera PARENTESISCERRADO
   | llamada_subprograma
   | NUMERO
   | variable

expresion_vector: CORCHETEABIERTO (expresiones_enteras)? CORCHETECERRADO
   | llamada_subprograma
   | PARENTESISABIERTO expresion_vector PARENTESISCERRADO
   | IDENT

variable: IDENT CORCHETEABIERTO expresion_entera CORCHETECERRADO
   | IDENT

expresiones_enteras: expresion_entera COMA expresiones_enteras
   | expresion_entera

condicion: condicion (Y|O) condicion
   | PARENTESISABIERTO condicion PARENTESISCERRADO
   | expresion (MAYOR|MENOR|MAYORIGUAL|MENORIGUAL|IGUAL|DISTINTO) expresion
   | NO condicion
```

SE PIDE:

Una solución para un analizador semántico que decida si cada llamada a un subprograma declarado en el perfil es correcta. Una llamada a subprograma declarado en perfil es correcta si y sólo si (a) el número de parámetros reales en la llamada coincide con el de parámetros formales en el perfil y (b) los tipos de los parámetros reales en la llamada emparejan secuencialmente con los declarados en el perfil. Por ejemplo, la ejecución del analizador semántico sobre el programa de ejemplo emitirá los siguientes mensajes por pantalla:

```
buscar_posicion_menor(i,v)   error en los tipos de parámetros
intercambiar(i,pos)          error en el número de parámetros
filtrar_componentes_pares(v) correcto
```

OBJETIVO

Construir un analizador semántico que detecte llamadas incorrectas a subprogramas.
Una llamada a subprograma es correcta si y sólo si
(a) el nombre del subprograma llamado coincide con el nombre de algún subprograma en algún perfil,
(b) el número de parámetros reales en la llamada coincide con el de parámetros formales en el perfil y
(c) los tipos de los parámetros reales en la llamada emparejan secuencialmente con los declarados en el perfil.

DECISIONES:
----

1) Memorizar los perfiles de los subprogramas. Distinguiremos entre perfiles de funciones y perfiles de acciones.

memoria_funciones:

nombre	parametros entrada	parametro salida
buscar_posicion_menor	VECTOR,ENTERO	ENTERO
filtrar_componentes_pares	VECTOR	VECTOR

memoria_acciones:

nombre	parametros entrada
intercambiar	VECTOR,ENTERO,ENTERO

2) Para cada ocurrencia de una llamada a subprograma en una expresión o instrucción debe comprobarse su corrección. Una llamada a subprograma es incorrecta si y sólo si el subprograma llamado no está declarado en los perfiles o, si lo está, no coinciden los parámetros reales con los parámetros formales en tipo o en número.
Por ejemplo:
La llamada buscar_posicion_menor(i,v) es incorrecta si el tipo de i es ENTERO y el de v es VECTOR y el perfil de buscar_posicion_menor es (VECTOR,ENTERO).
Como se observa, no coincide en tipos.

3) Los parámetros reales en las llamadas a subprogramas son expresiones, por tanto, el analizador debe calcular el tipo de las expresiones.
El tipo de una expresión puede ser: entero, vector o error.
Una expresión es errónea si y sólo si no es de tipo entero o de tipo vector.

4) Las expresiones pueden contener variables. El tipado dinámico de S obligará a actualizar el tipo de la variable asignada al procesar asignaciones.
Para calcular el tipo de una expresión será necesario memorizar el tipo de las variables durante el procesamiento del programa. Las variables usadas en expresiones que no ocurren en memoria_variables se consideran de tipo error.

memoria_variables:

nombre	tipo
v	VECTOR
q	ENTERO
...	...

```
GRAMATICA ATRIBUIDA:
--------------------
(global) memoria_funciones, memoria_acciones, memoria_variables

programa: perfiles seccion_principal

perfiles: PERFILES (perfil)* FPERFILES

seccion_principal: PROGRAMA instrucciones FPROGRAMA

perfil: (perfil_funcion | perfil_accion) PUNTOYCOMA

//Decisión 1
perfil_funcion: FUNCION IDENT PARENTESISABIERTO (params_entrada=parametros)?
PARENTESISCERRADO
                     DEVUELVE PARENTESISABIERTO param_salida=parametro
PARENTESISCERRADO
       { almacenar en memoria_funciones el perfil (IDENT,params_entrada,param_salida) }

//Decisión 1
perfil_accion: ACCION IDENT PARENTESISABIERTO (params_entrada=parametros)?
PARENTESISCERRADO
       { almacenar en memoria_acciones el perfil (IDENT,params_entrada) }

parametro dev param: ENTERO {param=entero} | VECTOR {param=vector};

parametros dev params: param=parametro {añadir param a params}
                       COMA aux=parametros {añadir cada elemento de aux a params en
orden}
   | parametro {añadir param a params}

instrucciones: instruccion instrucciones
   | instruccion

instruccion: asignacion PUNTOYCOMA | iteracion | condicional | llamada_subprograma
PUNTOYCOMA

//Decisión 4
asignacion:  IDENT ASIGNACION t=expresion
      { actualizar el tipo de IDENT en memoria_variables con t }

iteracion: PARA IDENT { actualizar el tipo de IDENT en memoria_variables con entero }
             EN rango HACER instrucciones FPARA

rango: expresion_entera RANGO expresion_entera

condicional: SI condicion ENTONCES instrucciones (SINO instrucciones)? FSI

//Decisión 2
llamada_subprograma dev tipo:
     IDENT PARENTESISABIERTO (tipos=expresiones)? PARENTESISCERRADO
     { si IDENT no es ni una accion ni una funcion entonces tipo = error
       sino
          si IDENT es una funcion entonces
             si los parametros de entrada de IDENT emparejan con tipos entonces
                tipo = parametro salida de IDENT en memoria_funciones
             sino
                tipo = error
                emitir mensaje de error en consecuencia
             fsi
          sino
```

```
              si los parametros de entrada de IDENT emparejan con tipos entonces
                 tipo = error
              sino
                 tipo = error
                 emitir mensaje de error en consecuencia
              fsi
           fsi
        fsi }

   | ULTIMOINDICE PARENTESISABIERTO tipo1=expresion_vector PARENTESISCERRADO
       { si tipo1 es igual a error entonces tipo = error
         sino tipo = entero
         fsi }

expresiones dev tipos: tipo=expresion {añadir tipo a tipos} COMA aux=expresiones
        {añadir cada elemento de aux a tipos en orden}
   | tipo=expresion {añadir tipo a tipos}

expresion dev tipo: tipo=expresion_entera
         | tipo=expresion_vector

//Decisión 3
expresion_entera dev tipo:
     tipo1=expresion_entera (MAS|MENOS|POR|DIVISION) tipo2=expresion_entera
     { si tipo1 y tipo2 son iguales a entero entonces tipo = entero
       sino tipo = error
       fsi }
   | PARENTESISABIERTO tipo=expresion_entera PARENTESISCERRADO
   | tipo=llamada_subprograma
   | NUMERO {tipo = entero}
   | tipo=variable

//Decisión 3
expresion_vector dev tipo:
     CORCHETEABIERTO (tipos=expresiones_enteras)? CORCHETECERRADO
     { si algún elemento en tipos es igual a error entonces tipo = error
       sino tipo = lista
       fsi }
   | PARENTESISABIERTO tipo=expresion_vector PARENTESISCERRADO
   | tipo = llamada_subprograma
   | IDENT { si IDENT no está en memoria_variables entonces tipo = error
             sino tipo = consultar en el tipo de IDENT en memoria_variables
             fsi }

//Decisión 3
variable dev tipo: IDENT CORCHETEABIERTO tipo1=expresion_entera CORCHETECERRADO
    { si IDENT no está en memoria_variables entonces tipo = error
      sino
         si tipo1 es igual a entero entonces tipo = entero
         sino
            tipo = error
         fsi
      fsi }
   | IDENT { si IDENT no está en memoria_variables entonces tipo = error
             en otro caso tipo = consultar en el tipo de IDENT en memoria_variables }

expresiones_enteras dev tipos: tipo=expresion_entera {añadir tipo a tipos}
     COMA aux=expresiones_enteras {añadir cada elemento de aux a tipos en orden}
   | tipo=expresion_entera {añadir tipo a tipos}
```

```
condicion: condicion (Y|O) condicion
   | PARENTESISABIERTO condicion PARENTESISCERRADO
   | expresion (MAYOR|MENOR|MAYORIGUAL|MENORIGUAL|IGUAL|DISTINTO) expresion
   | NO condicion
```

3) ANTLR (1 punto)

Programar como método Visitor en ANTLR4/Java la siguiente regla de una gramática atribuida:

```
      expresion dev tipo: tipo=expresion_entera
           | tipo=expresion_vector
```

```
@Override
public String visitExpresion(Anasint2.ExpresionContext ctx) {
    String tipo=null;
    if (ctx.expresion_entera()!=null)
       tipo=visitExpresion_entera(ctx.expresion_entera());
    else tipo=visitExpresion_vector(ctx.expresion_vector());
    return tipo;
}
```