TRADUCTOR: Parte I ANÁLIZADOR LÉXICO Y SINTÁCTICO

PROCESADORES DE LENGUAJES (GII - URJC) 2019-2020

Marina Fernández - Gonzalo Gómez - Álvaro Justo Rivas

Índice

DESCRIPCIÓN DE LA PRÁCTICA

Se nos ha pedido la codificación de un traductor de código estilo **Fortran** a código tipo **C**, el cual se implementa en el archivo **FortranToC.g4**.

Dicho archivo se estructura en 3 secciones: declaración de variables y constantes, declaración de funciones, y declaración de sentencias. En esta última, además se incluye la implementación de los requisitos opcionales de la práctica.

Cabe destacar la presentación del código siguiendo los estándares de formato y estilo según *Code Conventions for the Java TM Programming Language*[1], con el fin de mantenerlo limpio, organizado, y facilitar su lectura. Además de estar debidamente comentado para simplificar su comprensión.

CASOS DE PRUEBA

A continuación, se mostrarán 8 casos de prueba, cuatro correctos y cuatro incorrectos, para poder analizar el funcionamiento del traductor.

CASOS CORRECTOS

En esta fase, la salida no es demasiado llamativa.

A continuación se muestran las salidas para los cuatro casos de prueba facilitados.

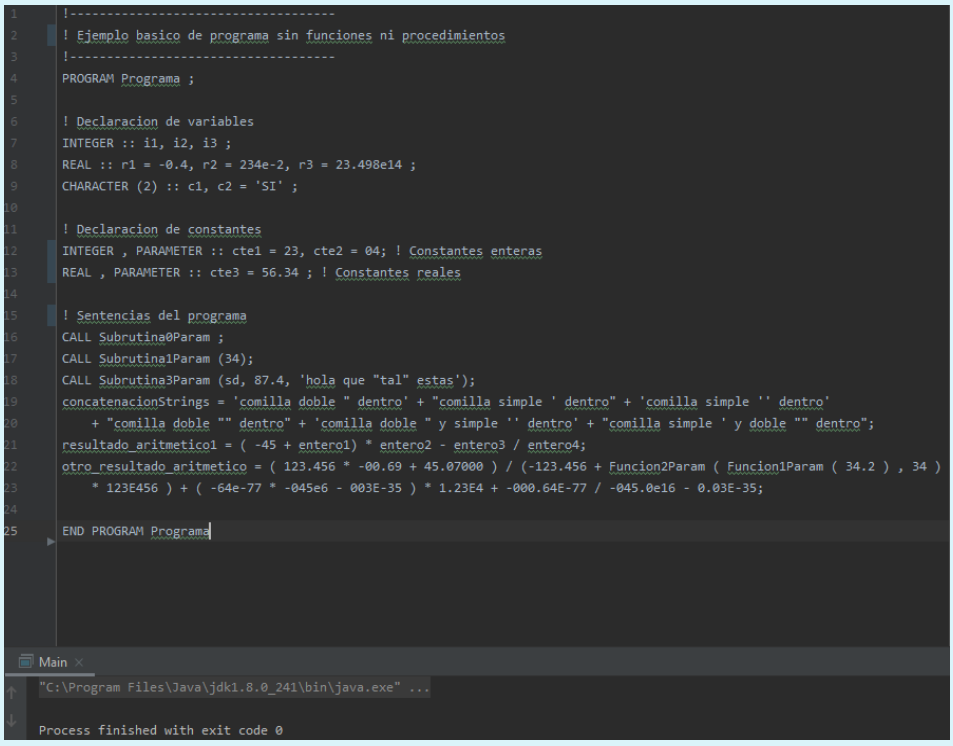


ILUSTRACIÓN 1. EJEMPLO CASO BÁSICO

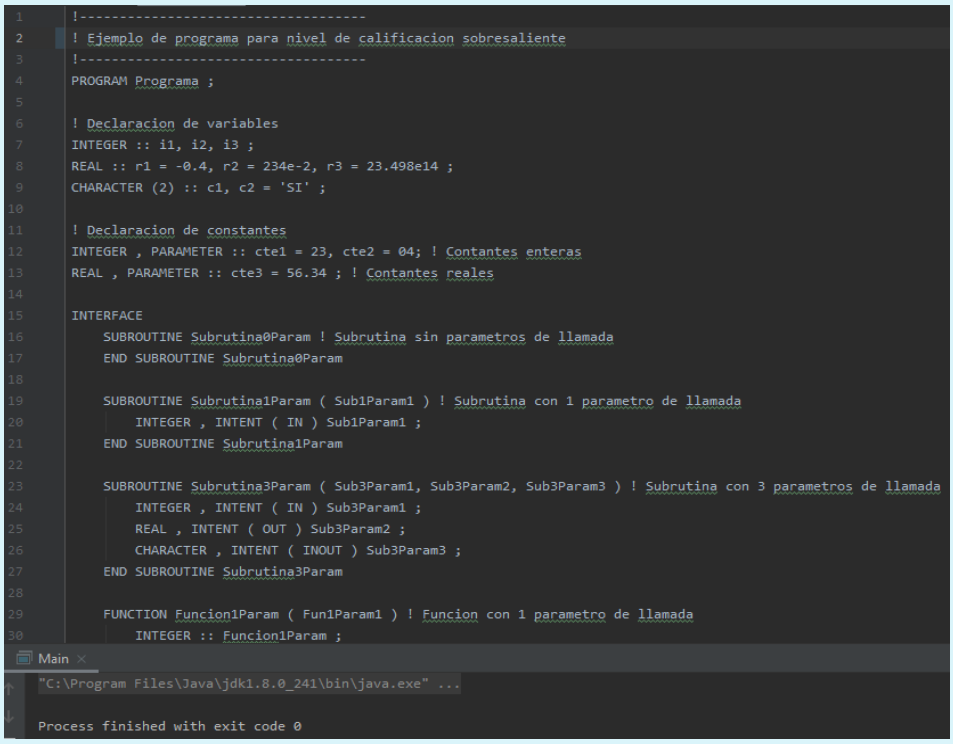
[1] https://www.oracle.com/technetwork/java/codeconvtoc-136057.html

```
!-----------------------------------
! Ejemplo de programa para nivel de calificacion aprobado
!-----------------------------------
PROGRAM Programa ;

! Declaracion de variables
INTEGER :: i1, i2, i3 ;
REAL :: r1 = -0.4, r2 = 234e-2, r3 = 23.498e14 ;
CHARACTER (2) :: c1, c2 = 'SI' ;

! Declaracion de constantes
INTEGER , PARAMETER :: cte1 = 23, cte2 = 04; ! Contantes enteras
REAL , PARAMETER :: cte3 = 56.34 ; ! Contantes reales

INTERFACE
    SUBROUTINE Subrutina0Param ! Subrutina sin parametros de llamada
    END SUBROUTINE Subrutina0Param

    SUBROUTINE Subrutina1Param ( Sub1Param1 ) ! Subrutina con 1 parametro de llamada
        INTEGER , INTENT ( IN ) Sub1Param1 ;
    END SUBROUTINE Subrutina1Param

    SUBROUTINE Subrutina3Param ( Sub3Param1, Sub3Param2, Sub3Param3 ) ! Subrutina con 3 parametros de llamada
        INTEGER , INTENT ( IN ) Sub3Param1 ;
        REAL , INTENT ( OUT ) Sub3Param2 ;
        CHARACTER , INTENT ( INOUT ) Sub3Param3 ;
    END SUBROUTINE Subrutina3Param

    FUNCTION Funcion1Param ( Fun1Param1 ) ! Funcion con 1 parametro de llamada
        INTEGER :: Funcion1Param ;
Main ×
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...

Process finished with exit code 0
```

ILUSTRACIÓN 2 EJEMPLO CASO APROBADO SUFICIENTE

```
!-----------------------------------
! Ejemplo de programa para nivel de calificacion notable
!-----------------------------------
PROGRAM Programa ;

! Declaracion de variables
INTEGER :: i1, i2, i3 ;
REAL :: r1 = -0.4, r2 = 234e-2, r3 = 23.498e14 ;
CHARACTER (2) :: c1, c2 = 'SI' ;

! Declaracion de constantes
INTEGER , PARAMETER :: cte1 = 23, cte2 = 04; ! Contantes enteras
REAL , PARAMETER :: cte3 = 56.34 ; ! Contantes reales

INTERFACE
    SUBROUTINE Subrutina0Param ! Subrutina sin parametros de llamada
    END SUBROUTINE Subrutina0Param

    SUBROUTINE Subrutina1Param ( Sub1Param1 ) ! Subrutina con 1 parametro de llamada
        INTEGER , INTENT ( IN ) Sub1Param1 ;
    END SUBROUTINE Subrutina1Param

    SUBROUTINE Subrutina3Param ( Sub3Param1, Sub3Param2, Sub3Param3 ) ! Subrutina con 3 parametros de llamada
        INTEGER , INTENT ( IN ) Sub3Param1 ;
        REAL , INTENT ( OUT ) Sub3Param2 ;
        CHARACTER , INTENT ( INOUT ) Sub3Param3 ;
    END SUBROUTINE Subrutina3Param

    FUNCTION Funcion1Param ( Fun1Param1 ) ! Funcion con 1 parametro de llamada
        INTEGER :: Funcion1Param ;
Main ×
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...

Process finished with exit code 0
```

ILUSTRACIÓN 3. EJEMPLO CASO NOTABLE



```
!-----------------------------------
! Ejemplo de programa para nivel de calificacion sobresaliente
!-----------------------------------
PROGRAM Programa ;

! Declaracion de variables
INTEGER :: i1, i2, i3 ;
REAL :: r1 = -0.4, r2 = 234e-2, r3 = 23.498e14 ;
CHARACTER (2) :: c1, c2 = 'SI' ;

! Declaracion de constantes
INTEGER , PARAMETER :: cte1 = 23, cte2 = 04; ! Contantes enteras
REAL , PARAMETER :: cte3 = 56.34 ; ! Contantes reales

INTERFACE
    SUBROUTINE Subrutina0Param ! Subrutina sin parametros de llamada
    END SUBROUTINE Subrutina0Param

    SUBROUTINE Subrutina1Param ( Sub1Param1 ) ! Subrutina con 1 parametro de llamada
        INTEGER , INTENT ( IN ) Sub1Param1 ;
    END SUBROUTINE Subrutina1Param

    SUBROUTINE Subrutina3Param ( Sub3Param1, Sub3Param2, Sub3Param3 ) ! Subrutina con 3 parametros de llamada
        INTEGER , INTENT ( IN ) Sub3Param1 ;
        REAL , INTENT ( OUT ) Sub3Param2 ;
        CHARACTER , INTENT ( INOUT ) Sub3Param3 ;
    END SUBROUTINE Subrutina3Param

    FUNCTION Funcion1Param ( Fun1Param1 ) ! Funcion con 1 parametro de llamada
        INTEGER :: Funcion1Param ;
```

```
Main
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...

Process finished with exit code 0
```

ILUSTRACIÓN 4. EJEMPLO CASO SOBRESALIENTE

CASOS ERRÓNEOS

A continuación, se mostrarán cuatro ejemplos de la recuperación de errores de nuestro programa.

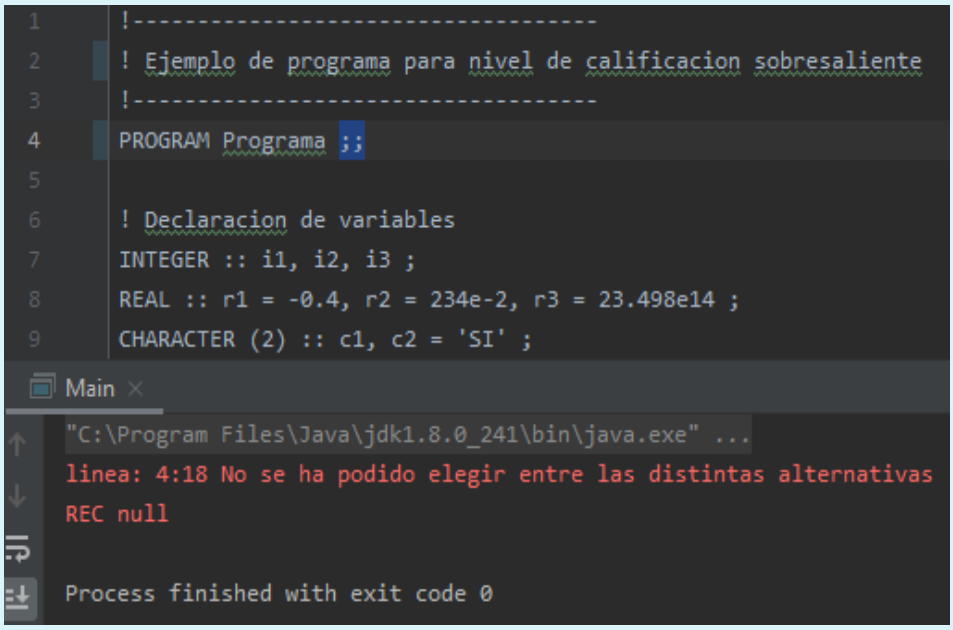


ILUSTRACIÓN 5. EJEMPLO DE DECISIÓN DE ESTRATEGIA.

AQUÍ PODEMOS VER COMO EL PROGRAMA DETECTA LOS DOS ';' AL COMIENZO DEL MISMO Y LO NOTIFICA INDICANDO QUE HAY AMBIGÜEDAD AL INTENTAR ELEGIR UNA ESTRATEGIA A SEGUIR.

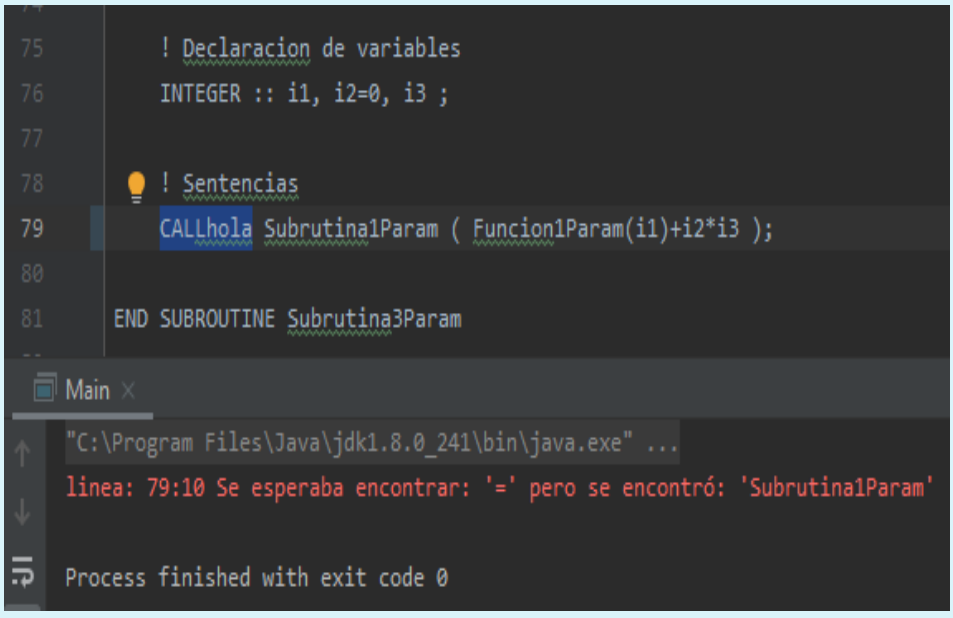


ILUSTRACIÓN 6. EJEMPLO ELEMENTO INTRUSO EN VEZ DE PALABRA RESERVADA.

PODEMOS OBSERVAR CÓMO INDICA CORRECTAMENTE LA POSICIÓN DEL FALLO.

```
SUBROUTINE Subrutina1Param ( Sub1Param1 ) ! Subrutina con 1 parametro de llamada
    INTEGER , INTENT ( IN ) Sub1Param1

    ! Declaracion de variables
    REAL :: r1, r2=0.2, r3 ;

    ! Sentencias
    CALL Subrutina0Param;
    r1 = Sub1Param1;
END SUBROUTINE Subrutina1Param
```

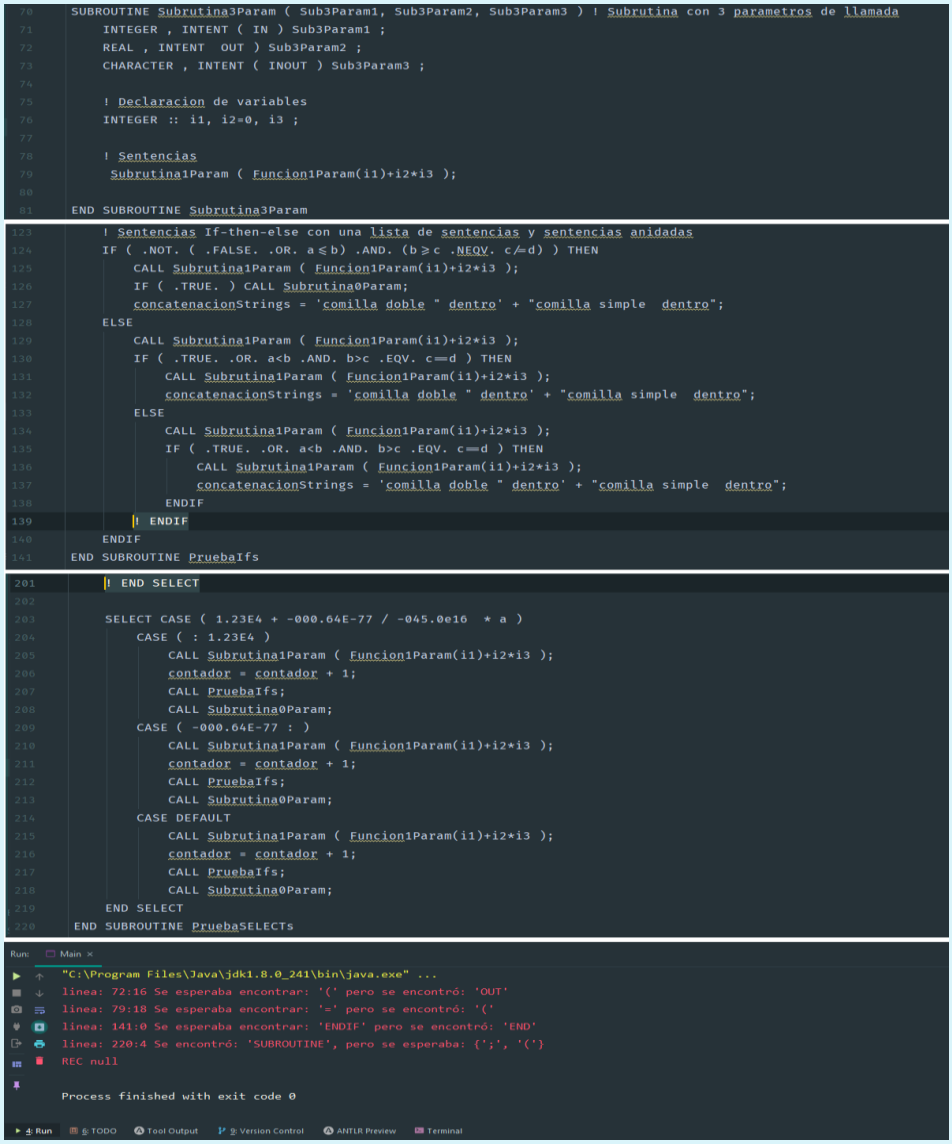
```
SUBROUTINE Subrutina3Param ( Sub3Param1, Sub3Param2, Sub3Param3 ) ! Subrutina con 3 parametros de llamada
    INTEGER , INTENT ( IN ) Sub3Param1 ;
    REAL , INTENT  OUT ) Sub3Param2 ;
    CHARACTER , INTENT ( INOUT ) Sub3Param3 ;

    ! Declaracion de variables
    INTEGER :: i1, i2=0, i3 ;

    ! Sentencias
    CALL Subrutina1Param ( Funcion1Param(i1)+i2*i3 );

END SUBROUTINE Subrutina3Param
```

```
FUNCTION Funcion2Param ( Fun2Param1, Fun2Param2 ) ! Funcion con 2 parametros de llamada
    REAL :: Funcion2Param ;
    INTEGER , INTENT ( IN ) Fun2Param1 ;
    CHARACTER (25) , INTENT ( IN ) Fun2Param2 ;
    ! Declaracion de variables
    REAL :: r1, r2=0.2, r3 ;

    ! Sentencias
     Subrutina0Param;
    Funcion2Param = Sub1Param1;
END FUNCTION Funcion2Param
```

```
Run:  Main ×
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
linea: 63:1 Se esperaba encontrar: ';' pero se encontró: 'REAL'
linea: 72:16 Se esperaba encontrar: '(' pero se encontró: 'OUT'
linea: 103:17 Se encontró: ';', pero se esperaba: '='
REC null

Process finished with exit code 0

4: Run  6: TODO  Tool Output  9: Version Control  ANTLR Preview  Terminal
```

ILUSTRACIÓN 7. EJEMPLO DE DETECCIÓN MÚLTIPLE Y RECUPERACIÓN (3 CASOS).

AQUÍ PODEMOS OBSERVAR CÓMO EL ANALIZADOR DETECTA CORRECTAMENTE LA FILA Y LA COLUMNA DE LOS SÍMBOLOS QUE FALTAN O QUE SE ENCUENTRAN EN ESTADO ERRÓNEO. CONCRETAMENTE SE APRECIA LA RECUPERACIÓN DE ERRORES UBICADOS EN DISTINTAS SUBRUTINAS.



```
SUBROUTINE Subrutina3Param ( Sub3Param1, Sub3Param2, Sub3Param3 ) ! Subrutina con 3 parametros de llamada
    INTEGER , INTENT ( IN ) Sub3Param1 ;
    REAL , INTENT  OUT ) Sub3Param2 ;
    CHARACTER , INTENT ( INOUT ) Sub3Param3 ;

    ! Declaracion de variables
    INTEGER :: i1, i2=0, i3 ;

    ! Sentencias
     Subrutina1Param ( Funcion1Param(i1)+i2*i3 );

END SUBROUTINE Subrutina3Param
```

```
    ! Sentencias If-then-else con una lista de sentencias y sentencias anidadas
    IF ( .NOT. ( .FALSE. .OR. a≤b) .AND. (b≥c .NEQV. c≠d) ) THEN
        CALL Subrutina1Param ( Funcion1Param(i1)+i2*i3 );
        IF ( .TRUE. ) CALL Subrutina0Param;
        concatenacionStrings = 'comilla doble " dentro' + "comilla simple  dentro";
    ELSE
        CALL Subrutina1Param ( Funcion1Param(i1)+i2*i3 );
        IF ( .TRUE. .OR. a<b .AND. b>c .EQV. c==d ) THEN
            CALL Subrutina1Param ( Funcion1Param(i1)+i2*i3 );
            concatenacionStrings = 'comilla doble " dentro' + "comilla simple  dentro";
        ELSE
            CALL Subrutina1Param ( Funcion1Param(i1)+i2*i3 );
            IF ( .TRUE. .OR. a<b .AND. b>c .EQV. c==d ) THEN
                CALL Subrutina1Param ( Funcion1Param(i1)+i2*i3 );
                concatenacionStrings = 'comilla doble " dentro' + "comilla simple  dentro";
            ENDIF
        ! ENDIF
    ENDIF
END SUBROUTINE PruebaIfs
```

```
    ! END SELECT

    SELECT CASE ( 1.23E4 + -000.64E-77 / -045.0e16  * a )
        CASE ( : 1.23E4 )
            CALL Subrutina1Param ( Funcion1Param(i1)+i2*i3 );
            contador = contador + 1;
            CALL PruebaIfs;
            CALL Subrutina0Param;
        CASE ( -000.64E-77 : )
            CALL Subrutina1Param ( Funcion1Param(i1)+i2*i3 );
            contador = contador + 1;
            CALL PruebaIfs;
            CALL Subrutina0Param;
        CASE DEFAULT
            CALL Subrutina1Param ( Funcion1Param(i1)+i2*i3 );
            contador = contador + 1;
            CALL PruebaIfs;
            CALL Subrutina0Param;
    END SELECT
END SUBROUTINE PruebaSELECTs
```

```
Run:  Main ×
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
linea: 72:16 Se esperaba encontrar: '(' pero se encontró: 'OUT'
linea: 79:18 Se esperaba encontrar: '=' pero se encontró: '('
linea: 141:0 Se esperaba encontrar: 'ENDIF' pero se encontró: 'END'
linea: 220:4 Se encontró: 'SUBROUTINE', pero se esperaba: {';', '('}
REC null

Process finished with exit code 0

▶ 4: Run  6: TODO  Tool Output  9: Version Control  ANTLR Preview  Terminal
```

ILUSTRACIÓN 8. EJEMPLO DE DETECCIÓN MÚLTIPLE Y RECUPERACIÓN (4 CASOS).

AQUÍ PODEMOS OBSERVAR CÓMO NUESTRO PROGRAMA REACCIONA ANTE VARIOS FALLOS, EN ESTE CASO LOS 2 PRIMEROS MUESTRAN CÓMO EL ANALIZADOR SE RECUPERA AL ENCONTRAR UN ERROR EN LA MISMA FUNCIÓN Y SIGUE PROCESANDO LA ENTRADA. EN ÚLTIMO LUGAR SE ENCUENTRAN OTROS 2 ERRORES QUE NO SE HABÍAN EXPUESTO ANTERIORMENTE, EN LOS QUE FALTAN PALABRAS RESERVADAS DEL LENGUAJE Y EL ANALIZADOR DEVUELVE UNA PREDICCIÓN RAZONABLE SOBRE LOS POSIBLES ELEMENTOS QUE PODRÍAN ESTAR UBICADOS EN SU LUGAR.

DIFICULTADES

En esta parte la carga de trabajo ha estado bastante marcada por el enunciado, puesto que indicaba de forma muy concreta qué teníamos que hacer y el resultado que debíamos obtener, salvo en la parte de recuperación y detección de errores, cuya implementación ha sido muy difícil.

Para llevar a cabo esta parte, ha sido necesario analizar y comprender el capítulo 9 de la guía oficial de ANTLR en profundidad (dicho libro se nos proporcionaba en los recursos de la asignatura). Para esta parte, hemos creado 2 clases auxiliares, *CustomErrorStrategy* y *CustomErrorListener*.

En el *Listener* aparece la función de syntaxError que es la encargada de notificar al usuario de forma correcta (especificando línea y columna) el error cuando se ha detectado.

Nos encontramos con la necesidad de idear una nueva estrategia frente a los errores para modificar el comportamiento que tiene por defecto ANTLR (*ErrorStrategy*), en este caso ANTLR ya trae la recuperación de errores implementada pero para mostrarlo de forma adecuada al usuario y adaptarlo a la práctica hemos realizado algunos cambios significativos.

En el capítulo 9 del libro aparece explicada la forma en la que se debe modificar la recuperación de errores, redefiniendo las operaciones de *recoverInLine*, *sync* y *recover*. Al redefinir estas funciones hemos conseguido que se notifiquen errores en distintas subrutinas de la entrada con bastante robustez, y en la misma subrutina en algunos casos concretos (Ilustración 8).

A demás, hemos redefinido todas las funciones que traía la estrategia por defecto de ANTLR (*DefaultErrorStrategy*) las cuales notificaban algún mensaje al usuario para conseguir así que se notificaran en castellano.

Por último hemos añadido estas 2 funcionalidades al analizador sintáctico en el método main de la aplicación, cambiando la estrategia y el listener por defecto con los que trabaja el analizador sintáctico de ANTLR por los personalizados.

CONCLUSIÓN

En conclusión, todos estamos de acuerdo en que esta primera parte ha sido bastante interesante y útil, con la que hemos aprendido muchos conceptos nuevos sobre el funcionamiento de los compiladores que usamos a diario y de los procesadores de lenguajes en general.

Esperamos haber conseguido una arquitectura del analizador léxico y sintáctico lo suficientemente robusta como para facilitarnos la implementación de la fase de síntesis (traductor dirigido por la sintaxis) que corresponde con la próxima parte del desarrollo de este procesador de lenguajes.