Práctica de Compilación I Curso 2022

Fecha de entrega del enunciado: 4/II
Hitos más importantes (entregas parciales):
- analizador léxico/sintáctico: 27/II
- ETDS: 27/III

Entrega obligatoria (escoger una):
- práctica completa (4 o 5): 1/V
- práctica completa (6 o más): 15/V

(eGela cerrará las entregas a las 23:55 de la fecha relevante)

1. Introducción

La práctica consiste en construir el front-end de un compilador utilizando la técnica de construcción de traductores ascendente, a partir de un esquema de traducción dirigida por la sintaxis. El lenguaje de entrada (fuente) al compilador es un lenguaje de alto nivel, y el de salida un código de tres direcciones. Se recomienda usar en la implementación del traductor C++, Flex y Bison.

Consideraciones generales sobre la práctica.

Esta práctica tiene como objetivos principales los siguientes:

- Manejar con soltura la noción de traducción dirigida por la sintaxis.
- Desarrollar la capacidad para utilizar Flex y Bison en la construcción de traductores.
- Plantear la dificultad que presenta el desarrollo de un compilador respecto a:
  - La especificación de las características de un lenguaje de alto nivel.
  - La calidad del código generado.
  - El volumen de programación que supone la construcción de un compilador.

En resumen, se trata de facilitar la asimilación de una parte importante de los conceptos desarrollados en el temario de la asignatura. En palabras extraídas de la documentación de un grupo que realizó la práctica en un curso anterior: "... hay que reconocer que se aprende mucho de esta asignatura durante la realización de la práctica, y que resulta muy interesante, aunque supone mucho trabajo.".

La parte **básica y obligatoria** de la práctica conlleva el diseño e implementación de un reconocedor para el lenguaje fuente propuesto y el diseño de un ETDS que especifique la traducción de los constructores del lenguaje.
El resto de objetivos de la práctica se definen **de forma opcional**:

1 Implementación del traductor, es decir, la implementación del ETDS obligatorio.
2 La modificación de la gramática (y ETDS) para que acepte una estructura de control nueva.
3 La modificación de la gramática (y ETDS) para que permita el uso de expresiones booleanas.
4 La modificación de la gramática (y ETDS) para que acepte un tipo o estructura de datos nueva.
5 La comprobación de las restricciones semánticas.
6 La ampliación del lenguaje de entrada (y ETDS) para aceptar las llamadas a procedimientos.
7 La ampliación del lenguaje de entrada (y ETDS) para que permita la declaración y uso de arrays multidimensionales.
8 La gestión de la Tabla de Símbolos (para los puntos 4, 5, 6 y 7 anteriores).

2. Evaluación por objetivos

Se parte de la definición de unos objetivos académicos mínimos que marcan el nivel de aprobado en la práctica. Estos serían los recomendados para aquellas personas que pretendan únicamente superar

la asignatura. A partir de los objetivos básicos, se añaden nuevas funcionalidades al traductor, consiguiendo diferentes niveles de calificación en cada paso.

	Alcance	**La documentación debe incluir:**
Mínimo (4)	Implementación del analizador léxico y sintáctico. Desarrollo del ETDS.	(1) = Especificación léxica + ETDS
Aprobado (5)	Implementación del traductor	(2) = (1) + casos de prueba

A continuación señalamos los objetivos opcionales (el orden no es significativo). La puntuación que aparece es la máxima alcanzable, dependiendo del nivel de satisfacción de la ampliación desarrollada. Los alumnos también pueden plantear al profesor sus propios objetivos:

	Alcance	**La documentación debe incluir:**
+ 1 punto	Especificación de una estructura de control nueva, diseño de la traducción e implementación.	(2) + especificación + ETDS de la nueva estructura de control.
+ 1 punto	Especificación de las expresiones booleanas, diseño e implementación de su traducción.	(2) + ETDS de booleanas + documentación pruebas
+ 2 puntos	Especificación de un nuevo tipo de datos o nueva estructura de datos, diseño de la traducción e implementación.	(2) + especificación + ETDS del tipo de datos o estructura de datos + restricciones semánticas.
+ 1 punto	Llamadas a procedimientos (uso correcto de parámetros reales)	(2) + ETDS de llamadas a procedimientos
+ 2 puntos	Arrays multidimensionales.	(2) + ETDS de arrays multidimensionales
+ 2 puntos	Diseño e implementación de restricciones semánticas	(2) + descripción de restricciones semánticas + ETDS de restricciones semánticas

REQUISITO IMPORTANTE:
En cualquier caso, la especificación (contenida en el ETDS y en la especificación de los tokens) DEBE COINCIDIR plenamente con la implementación.[1] En caso contrario, es decir, si se ha implementado de forma diferente a la especificación, se suspenderá la práctica.

3. Plan de trabajo

A continuación se describen las tareas asociadas a los objetivos mínimos de la práctica. Seguidamente se describen el resto de tareas, relacionadas con el desarrollo completo del traductor. En función de sus objetivos cada grupo deberá realizar una planificación propia. Además de las tareas presentadas, a lo largo del desarrollo de la práctica se realizan otras dos: gestión del grupo y documentación.

3.1 Tareas básicas

1. Familiarización con el enunciado.
2. Diseño, implementación y pruebas del analizador léxico.
3. Implementación y pruebas del analizador sintáctico.
4. Desarrollo del ETDS. :
   - **a.** Estudiar cuáles serían los atributos necesarios para poder definir la traducción.
   - **b.** Introducir las acciones semánticas que definan el proceso de traducción.
   - **c.** Comprobar sobre ejemplos suficientemente completos la corrección del ETDS desarrollado.
5. Pruebas finales y documentación.

[1] Un ejemplo típico NO ACEPTABLE podría ser que en el ETDS se especifique de una forma la traducción, y que en la implementación se haga de forma diferente.

3.2 Tareas opcionales

Aprobado (5)

- Implementación del traductor.
- Inclusión de mensajes de error adecuados. Tratamiento para que no se detenga al detectar el primer error.

+ 1 punto (expresiones booleanas)

- Ampliación del ETDS para recoger la traducción de expresiones booleanas.
- Ampliación del traductor para implementar lo anterior.
- Pruebas: Árbol "a mano" y verificado el resultado con el programa.

+ 1 punto (estructura de control nueva).

- Especificación léxica, sintáctica y semántica de la nueva estructura.
- Ampliación del ETDS para aceptar la nueva estructura de control.
- Pruebas e implementación.
- La puntuación dependerá de la dificultad de traducción de la estructura. Ver AVISOS.

+ 2 puntos (tipo o estructura de datos nueva)

- Especificación léxica, sintáctica y semántica del nuevo tipo o estructura de datos así como de sus operaciones.
- Ampliación del ETDS para aceptar la nueva estructura o tipo de datos.
- Pruebas e implementación.
- La puntuación dependerá de la dificultad de traducción del tipo o estructura de datos.

+ 1 punto (llamadas a procedimientos)

- Ampliación de la gramática y del ETDS para permitir llamadas a procedimientos
- Implementación y pruebas de las llamadas a procedimientos. Incluirá la comprobación del número y tipo de los parámetros, comprobaciones de ámbito, comprobaciones en llamadas por referencia.

+ 2 puntos (arrays multidimensionales)

- Especificación del uso y declaración de tablas multidimensionales.
- Ampliación de la gramática y del ETDS para permitir el tipo *array* muldimensional.
- Implementación y pruebas de traducción de los arrays.

+ 2 puntos (restricciones semánticas, uso correcto de identificadores, etc)

- Definición de las restricciones semánticas.
- Ampliación del ETDS para comprobar la corrección semántica.
- Implementación y pruebas del análisis semántico.

AVISOS:

- Debéis consultar con la profesora antes de añadir una estructura de control nueva, un tipo o estructura de datos nueva. Su puntuación variará en función de su complejidad y completitud. Ideas:
- Añadir un if-then-else no se valora. Sin embargo, hacer una estructura if-then-else con la posibilidad if-then-elseif-elseif…..-else puede alcanzar 1 punto.
- Añadir una instrucción tipo for con comprobaciones semánticas, puede alcanzar 1 punto.
- Añadir una instrucción tipo switch con comprobaciones semánticas, 1 punto.

4. Definición sintáctica del lenguaje de entrada

programa → **def main () :**
bloque_ppl

bloque_ppl → decl_bl **{**
decl_de_subprogs
lista_de_sentencias
}

bloque → **{**
lista_de_sentencias
}

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

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

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 → **def id** argumentos **:** bloque_ppl

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

lista_de_param → lista_de_ident **:** clase_par tipo resto_lis_de_param

clase_par → ξ | **&**

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 **:** bloque
| **while** expresion **:** bloque **else :** bloque
| **forever :** bloque
| **break if** expresion ;
| **continue** ;
| **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 **)**

NOTAS:

a. Los tokens *id*, *num_entero* y *num_real* son como los que se pedían en clase (lab. 1 de léxico).

b. LOS COMENTARIOS MULTILÍNEA empiezan por '" y acaban en "'. En medio puede haber cualquier secuencia de caracteres excepto '" y no acepta el carácter #. Pueden aparece en cualquier posición del programa de entrada.

c. El comentario de línea empieza por # y acaba en fin de línea (incluido). En medio no puede aparecer el carácter #.

d. La semántica de la instrucción ***if*** es que cuando la expresión sea verdadera se ejecuta el bloque y cuando sea falsa no se haga nada.

e. La semántica de la instrucción ***while...else…*** es la siguiente: el bloque contenido en el bucle1 se repite mientras la condición se cumpla. En el momento que la condición deja de cumplirse se ejecutará la parte del *else*.

f. La semántica de la instrucción ***forever*** es la siguiente: el bloque contenido en el bucle se repite indefinidamente.

g. La semántica de la instrucción ***continue*** es la siguiente: Es una instrucción que puede aparecer en el interior de un bucle con condición (en este caso *while...else*), en cualquiera de los dos bloques. La ejecución del bucle se interrumpe y va a evaluar la expresión.

h. La semántica de la instrucción ***break if*** es la siguiente: Es una instrucción que puede aparecer en el interior de un bucle. Si la expresión de *break if* es verdadera, la ejecución finaliza y va a evaluar la siguiente instrucción fuera del bloque. Esto es, si está en un *forever* la ejecución irá a la siguiente instrucción al forever; si está en el bloque *while* irá al bloque *else*, y si aparece en la parte *else* la ejecución irá fuera del *while*. Sin embargo, si la expresión de *break if* es falsa, el flujo del programa no cambia y la ejecución del bucle continua normalmente.

5. El lenguaje de salida.

En este apartado se realiza la descripción del código intermedio que se utilizará como lenguaje objeto del traductor. ***No todas las instrucciones descritas han de utilizarse necesariamente en la práctica de este año.***

Una instrucción de código intermedio tiene la forma:

etiqueta: instrucción;

La definición léxica de la etiqueta es **num_entero**.
Las instrucciones pueden ser de asignación, salto, declaración de objetos (procedimientos y variables), llamadas a procedimientos y de entrada/salida.
Los tipos con los que se trabaja son entero y real.

Asignaciones

x	**:=**	**y op$_1$ z;**	**x** es un identificador. **y** & **z** son constantes o identificadores. **op$_1$** es un operador aritmético binario. Ejemplos: a := b * 5; t := c div a;
x	**:=**	**op$_2$ y;**	**op$_2$** es un operador aritmético unario. Ejemplo: a := -b;
x	**:=**	**y**;	
x	**:=**	**y[i];**	**i** desplazamiento entero. **y** es un identificador.
x[i]	**:=**	**y;**	introduce **y** (o el contenido de **y**) en la dirección correspondiente a incrementar en **i** palabras la dirección de **x**. Ejemplos: a[20]:= 10; a[3]:=b;
x	**:=**	**^y;**	Introduce en **x** el contenido de la dirección que se encuentra en **y**. Ejemplo: a := ^b;
^x	**:=**	**y;**	Introduce **y** en la dirección que se encuentra en **x**. Ejemplo: ^b := a;

Saltos incondicionales

goto L;	**L** es una dirección que corresponde a una instrucción del código intermedio. Ejemplo: **goto 50;**

Saltos condicionales

if y oprel z goto L;	Si es cierto **y oprel z** cede el control a la instrucción en la dirección **L**. Ejemplo: **if a > 10 goto 20:;**

Declaraciones

int x;	Una por cada variable del programa principal y en cada procedimiento por cada variable declarada a nivel local como de tipo entero.
real x;	Idem para el tipo real
array_int x,y;	Para un array de y elementos de tipo entero.
array_real x,y;	Para un array de y elementos de tipo real.

Procedimientos

proc x;	Si **x** identifica al procedimiento, **proc x** precederá al código correspondiente al procedimiento.
val_int x; **val_real x;** **val_array_int x,y;** **val_array_real x,y;**	Una por cada variable del procedimiento declarada como parámetro por valor (según su tipo).
ref_int x; **ref_real x;** **ref_array_int x,y;** **ref_array_real x,y;**	Una por cada variable del procedimiento declarada como parámetro por referencia (según su tipo).
endproc x;	**x** identifica al procedimiento.

Ejemplo: Sea el procedimiento:

```
def ejemplo( a,b: integer; c: & integer)
        let d,e:integer in ...
```

Su traducción será la siguiente:

```
10: proc ejemplo;
11: val_int a;
12: val_int b;
13: ref_int c;
14: int d;
15: int e;...
40: endproc ejemplo;
```

Llamadas a procedimientos:

param_ref x; Una por cada parámetro actual por referencia en las llamadas al procedimiento.

param_val x; Una por cada parámetro actual por valor en las llamadas al procedimiento.

call x; **x** nombre de un procedimiento (para el que existirá su correspondiente **proc x**).

Ejemplo: Sea la llamada ejemplo(a+b,12,c)
Su traducción sería:

```
40: t1 := a+b;
41: param_val t1;
42: param_val 12;
43: param_ref c;
44: call ejemplo;
```

Entrada/Salida

read x;

write y; **y** puede ser un nombre, una cte. numérica o una secuencia de caracteres. No escribe fin de línea.

writeln; Escribe exclusivamente un fin de línea.

Instrucción nula

nop; No hace nada.

Principio y final del programa.

proc main; Será la cabecera del programa.

halt; Cuando se alcanza esta instrucción termina la ejecución del programa.

Ejemplo de traducción

NOTA: La **declaración de procedimientos** se ha de tratar para todas la notas.

La **llamada a procedimientos** es opcional**.**

Programa fuente:

```
    def main ():
    let a,b,c : integer; d,e : float in {
    ''' esto es un comentario '''
    def sumar (x,y: integer; resul: & integer):
    let aux, vueltas: integer in {
      aux = y; resul = x;
      if resul < 1000: {
         vueltas = 0;
         while aux == 0: {
            resul = resul + 1;
            break if resul > 100000;
            aux = aux - 1;
            vueltas = vueltas + 1;
         } else: {
            if resul < 0: { continue; }
            println(vueltas);
         } # acaba while-else
         println(resul);
      } # acaba if
      println(vueltas);
  } # acaba def sumar -let
  read(a); read(b);
  d = 1/b;
  sumar(a,b,c); ''' los que hagan llamadas a procedimientos '''
  c = c*(c*d)+e;
  println(c);
} # acaba def main -let
```

Programa equivalente en código intermedio:

```
 1: proc main ;
 2: int a ;
 3: int b ;
 4: int c ;
 5: real d ;
 6: real e ;
 7: proc sumar ;
 8: val_int x ;
 9: val_int y ;
10: ref_int resul ;
11: int aux ;
12: int vueltas ;
13: aux := y ;
14: resul := x ;
15: if resul < 1000 goto 17 ;
16: goto 36 ;
17: vueltas := 0 ;
18: if aux = 0 goto 20 ;
19: goto 29 ;
20: _t1 := resul + 1 ;
21: resul := _t1 ;
22: if resul > 100000 goto 29 ;
23: goto 24 ;
24: _t2 := aux - 1 ;
25: aux := _t2 ;
26: _t3 := vueltas + 1 ;
27: vueltas := _t3 ;
28: goto 18 ;
29: if resul < 0 goto 31 ;
30: goto 32 ;
31: goto 18 ;
32: write vueltas ;
33: writeln ;
34: write resul ;
35: writeln ;
36: write vueltas ;
37: writeln ;
38: endproc sumar ;
39: read a ;
40: read b ;
41: _t4 := 1 / b ;
42: d := _t4 ;
43: param_val a ;
44: param_val b ;
45: param_ref c ;
46: call sumar ;
47: _t5 := c * d ;
48: _t6 := c * _t5 ;
49: _t7 := _t6 + e ;
50: c := _t7 ;
51: write c ;
52: writeln ;
53: halt ;
```

6. Cómo obtener el código.

Para obtener el código de salida se hace necesaria **una estructura de datos en la que podamos ir acumulando las instrucciones generadas**. A continuación se presenta una posible especificación funcional de este tipo de datos.

INICIALIZA_CODIGO: → código
AÑADIR_INST : código X inst → código
*COMPLETAR: código X lista_ref_código X ref_código → código
Completa las instrucciones de la lista de referencias (saltos) con la referencia dada.
*OBTENREF: código → ref_código
Devuelve la referencia de la instrucción que va a ser añadida a continuación.
ESCRIBIR_CODIGO: código X fichero → fichero

Alguna operación más con la que habrá que contar.

Cuando se analiza el proceso de generación de código se descubre la necesidad de obtener nombres de identificadores que no puedan ser confundidos con otros que puedan aparecer en el programa fuente. Esto se puede realizar por medio de esta operación:

NUEVO_ID: → identificador

7. ¿Qué material se proporciona?

Además del enunciado se proporcionarán los siguientes componentes:

a. Un fichero conteniendo la gramática que define el lenguaje fuente (este enunciado)
b. Una solución para la implementación de la Tabla de Símbolos

9. Normas de entrega de la práctica

Aquellas prácticas que no sigan las normas no serán corregidas.

Entregable obligatorio:

- Fichero único comprimido con toda la práctica, a entregar usando e-gela.

Contenidos obligatorios del fichero

- **documentación** (ver mas abajo)
- **leeme.txt** nombre del grupo, miembros del grupo, instrucciones sobre cómo compilar los ficheros en el directorio **fuente** y cómo ejecutar el traductor (parámetros, etc.). Si siguiendo los pasos descritos no se consigue compilar y ejecutar el traductor, no será corregida la práctica.
- **fuente.** Directorio incluyendo los ficheros que contienen el código fuente:
  - Todos los fuentes escritos en Flex, Bison y C++
  - Fichero **Makefile** y carpeta de **pruebas** para poder ser ejecutado
- **Pruebas.** Una carpeta con los ficheros de pruebas que se proponen para comprobar el funcionamiento del compilador:
  - **PruebaBuena$_i$.dat.** Programa de prueba sin errores. Al principio del programa deben incluirse los comentarios que expliquen el sentido del ejemplo.
  - **PruebaMala$_i$.dat.** Programa de prueba con errores. Al principio del programa deben incluirse los comentarios que expliquen el sentido del ejemplo.

DOCUMENTACIÓN

La documentación debe tener una redacción correcta y se exigirá el cuidado de aspectos formales (por ejemplo, formato legible, ausencia de faltas de ortografía). NO INCLUIR CÓDIGO FUENTE.

Formato:

- Portada: Identificador del grupo, nombre y apellidos de los componentes, **un e-mail OFICIAL (ikasle.ehu.es)**.
- Índice.
- Páginas numeradas (en cada página aparecerá el identificador del grupo)

Contenidos:

- Introducción. Explicando qué es lo que se ha hecho (objetivos entregados).
- Autoevaluación. Señalando **razonadamente** la nota que corresponde a la práctica.
- Descripción por fases y miembros del grupo del esfuerzo invertido (en horas).
- Análisis léxico. Especificación de los tokens (expresiones regulares). **Autómata único**.
  - Casos de prueba (solo cuando se entregue el analizador léxico).
- Especificación del proceso de traducción (**EL ORDEN ES SIGNIFICATIVO**)

  1. **Gramática**

  2. **Definición de atributos. Por cada terminal y no-terminal (si tiene atributos):**
     - Nombre del atributo
     - Tipo de información que contiene. Por ejemplo:
       * referencia de código o lista de referencias de código
       * nombre de una variable
       * nombre o lista de nombres de identificador
       * enumerado (y sus valores posibles)
       * ...
     - Tipo de atributo (léxico, sintetizado)

  3. **Descripción de las abstracciones funcionales utilizadas.** Descripción de las funciones definidas para que la descripción del ETDS sea más compacta:
     - nombre
     - número y tipos de los argumentos
     - resultado
     - descripción breve de su funcionalidad y para qué se usa en el ETDS

  4. **ETDS. Es necesario entregar un nuevo ETDS por cada objetivo implementado.**

- Casos de prueba. **Entrada** y, en el caso de notas superiores al 4, **salida** obtenida.
- **En su caso**:
  - Gramática (léxico, sintaxis, semántica) ampliada para el tratamiento de una nueva estructura de control, ETDS actualizado y casos de prueba.
  - Gramática ampliada para el tratamiento de booleanas, ETDS actualizado y casos de prueba verficados.
  - Diseño y funcionalidad de la tabla de símbolos.
  - Restricciones semánticas. Descripción, ETDS actualizado y casos de prueba.
  - Gramática ampliada para el tratamiento de llamadas a procedimientos, ETDS actualizado y casos de prueba.
  - Gramática (léxico, sintaxis, semántica) ampliada para el tratamiento de un nuevo tipo de datos, ETDS actualizado y casos de prueba.
  - Gramática ampliada para el tratamiento de arrays multidimensionales, ETDS actualizado y casos de prueba.
- **Opcionalmente**:
  - Mejoras/especificidades/aspectos a resaltar
  - Comentarios/valoración sobre la práctica