Práctica de Compilación I Curso 2010/2011

Fecha de entrega del enunciado: 1/X
Hitos más importantes (entregas parciales):

- analizador léxico: 18/X
- ETDS: 17/XI
- analizador sintáctico: 1/XII (optativo)

Entrega final (escoger una):

- práctica completa (5 o 6): 10/XII
- práctica completa (7 o más): 23/XII

(moodle 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 descendente recursiva a partir de un esquema de traducción dirigida por la sintaxis. El lenguaje de entrada (fuente) al compilador es similar a un subconjunto reducido de Ada y el de salida un código de tres direcciones. En la implementación del traductor puede usarse cualquier lenguaje de programación que soporte la recursividad (podrá reutilizarse código C, JAVA y ADA de prácticas de cursos anteriores).

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 el esquema de construcción de programas "descendente recursivo predictivo" 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.
  - El tratamiento de los errores detectados.
  - 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.".

Con el fin de adaptarse a las diferentes condiciones de matrícula y objetivos de los alumnos que cursan la asignatura se presenta como opcional el desarrollo de aspectos fundamentales del front-end de un compilador para un lenguaje de alto nivel, como:

- Tratamiento de errores sintácticos y semánticos.
- Tratamiento de funciones y llamadas a procedimientos y funciones.
- Tratamiento de operadores lógicos.
- Tratamiento de tipos de datos estructurados.
- Validación de la corrección de un programa respecto a la compatibilidad de tipos, llamadas a procedimientos y funciones, etc.
- Manejo de la tabla de símbolos.

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 y van desde la implementación del proceso de traducción para los constructores básicos hasta la incorporación al traductor de las características más importantes de un front-end (incluyendo gestión de la tabla de símbolos, validación de restricciones semánticas y recuperación de errores sintácticos).

La ampliación del front-end (**opcional** en función de objetivos, y sin tener que seguir un orden) podrá incluir:

1- La modificación de la gramática (y ETDS) para que permita el uso de expresiones booleanas.
2- La comprobación de las restricciones semánticas. Concretamente, el uso correcto de identificadores.
3- La ampliación del lenguaje de entrada de forma que se acepten las llamadas a procedimientos.
4- La gestión de la Tabla de Símbolos (para puntos 3 y 4 anteriores).
5- La recuperación de errores sintácticos utilizando la técnica de panic mode.
6- La ampliación de la gramática de entrada para que permita la declaración y uso de arrays.

2. Evaluación por objetivos

La organización general de la práctica pretende adaptarse a las diversas condiciones previas y objetivos de los alumnos que cursan la asignatura. 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, añadiendo nuevas funcionalidades al traductor, se van añadiendo nuevos niveles de calificación. Como consideración general se recomienda a aquellos alumnos que no soporten una carga lectiva excesiva plantearse al menos los objetivos para un 8.

	Alcance	**Documentación debe incluir:**
Aprobado (5)	Implementación del analizador léxico y sintáctico. Desarrollo del ETDS	(1) = ETDS + casos de prueba
Aprobado (6)	Implementación del traductor especificado en la ETDS	(2) = (1) con nuevos casos de prueba

A continuación señalamos los objetivos opcionales (la puntuación que aparece es la máxima alcanzable, dependiendo de si se cumplen la ampliación necesaria), aunque los alumnos pueden plantear al profesor sus propios objetivos (el orden no es significativo):

	Alcance	**Documentación debe incluir:**
+ 2 puntos	Diseño e implementación de la traducción de expresiones booleanas	(2) + ETDS de booleanas
+ 2 puntos	Diseño e implementación de restricciones semánticas (uso correcto de identificadores).	(2) + descripción de restricciones semánticas + ETDS de restricciones semánticas
+ 2 puntos	Llamadas a procedimientos.	(2) + ETDS para llamadas a procedimientos.
+ 2 puntos	Arrays multidimensionales.	(2) + ETDS para arrays multidimensionales
+ 2 puntos	Diseño e implementación del tratamiento de errores sintácticos	(2) + diseño del tratamiento de errores

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.

[1] Un ejemplo típico NO ACEPTABLE podría ser que en la ETDS se especifique de una forma la traducción de un bucle, y que en la implementación se haga de forma diferente. Esto podría deberse a que la implementación estuviera basada en una ETDS diferente (de otro grupo o año) a la que entrega el grupo.

3. Plan de trabajo

El plan de trabajo depende de los objetivos del grupo. A continuación se describen las tareas asociadas a los objetivos mínimos de la práctica, aquellos necesarios para sacar el cinco. Estas tareas son suficientes para los que se conformen con aprobar la asignatura. 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. Obtención de una gramática a partir de la cual pueda realizarse análisis descendente predictivo. Implementación y pruebas del analizador completo.
4. Desarrollo del ETDS. Para desarrollar el ETDS, sobre la gramática completa que define el lenguaje fuente:
   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.

3.2 Tareas opcionales

Aprobado (6)

- Diseño e implementación del traductor que especifica el ETDS

+ 2 puntos (expresiones booleanas)

- Ampliación del ETDS para recoger la traducción de expresiones booleanas.
- Ampliación del traductor para implementar lo anterior.

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

- Definición de restricciones semánticas sobre uso correcto de identificadores.
- Ampliación del ETDS para comprobar la corrección semántica.
- Especificación funcional de la Tabla de Símbolos.
- Selección de una representación para la Tabla de Símbolos.
- Implementación y prueba de la parte de análisis semántico.

+ 2 puntos (llamadas a procedimientos)

- Ampliación de la gramática y del ETDS para permitir llamadas a procedimientos
- Implementación y prueba de las llamadas a procedimientos.

+ 2 puntos (errores sintácticos)

- Diseño del tratamiento de errores sintácticos
- Implementación del tratamiento de errores sintácticos

+ 2 puntos (arrays multidimensionales)

- Ampliación de la gramática y del ETDS para permitir el tipo *array* muldimensional.
- Implementación y prueba de los arrays.

4. Definición sintáctica del lenguaje de entrada.

programa → **programa id**
declaraciones
decl_de_subprogs
comienzo
lista_de_sentencias
fin ;

declaraciones → **variables** lista_de_ident **:** tipo **;** resto_dec
| ξ

resto_dec → **variables** lista_de_ident **:** tipo **;** resto_dec
| ξ

lista_de_ident → **id** resto_lista_id

resto_lista_id→ **,** id resto_lista_id | ξ

tipo → **entero** | **real**

decl_de_subprogs → decl_de_subprograma decl_de_subprogs
| ξ

decl_de_subprograma →
cabecera declaraciones
comienzo lista_de_sentencias **fin ;**

cabecera →**procedimiento id** argumentos

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

lista_de_param → lista_de_ident **:** clase_par tipo resto_lis_de_param

clase_par →**entrada** | **salida** | **entrada salida**

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 = expresión_simple **;**
| **si** expresion **entonces** lista_de_sentencias **fin si ;**
| **hacer** lista_de_sentencias **mientras** expresión **fin hacer** ;
| **salir si** expresión **;**
| **get (** variable **) ;**
| **put_line (** expresion_simple **) ;**

variable →**id**

expresión → expresión_simple == expresión_simple
| expresión_simple **>** expresión_simple
| expresión_simple **<** expresión_simple
| expresión_simple **>=** expresión_simple
| expresión_simple **<=** expresión_simple
| expresión_simple /= expresión_simple

expresión_simple →expresión_simple + término
| expresión_simple **-** término
| término

término → factor
| término ***** factor
| término / factor

factor → **id**
| **num_entero**
| **num_real**
| **(** expresión_simple **)**

NOTAS:

a. LOS COMENTARIOS empiezan por /* y acaban en */. En medio puede haber cualquier secuencia de caracteres excepto */.

b. La semántica de la instrucción *si* es que cuando la expresión sea verdadera se ejecuta la parte *entonces* y cuando sea falsa no se haga nada.

c. La semántica de la instrucción *hacer…mientras* es la siguiente: la lista de sentencias contenida en el bucle se repite mientras que la condición se cumpla.

d. La semántica de la instrucción *salir si* es la siguiente: si la expresión es verdadera se interrumpe la ejecución del bucle donde esté imbricada esta instrucción y se sale del bucle.

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 **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).

finproc;

Ejemplo: Sea el procedimiento:

```
procedure ejemplo(var a,b: in integer; c: in out integer) is
        d,e:integer;...
```

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;...
```

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.

prog x; 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:

```
programa ejemplo
        variables a,b,c : entero;
        variables d,e : real;
/* esto es un comentario */

procedimiento sumar (x,y: entrada entero; resul: entrada salida entero)
        variables aux:entero;
        comienzo
          hacer
                aux=a; c=b;
                aux = aux – 1 ;
                c = c+1;
          mientras aux > 0 fin hacer ;
        fin ;

comienzo
        get(a); get(b);
        d= 1/b;
        e= 1/a;
        sumar(a,b,c);  /* los que hagan llamadas a procedimientos */
        c= c*(c*d)+e;
        put_line(c*c);
fin ;
```

Programa equivalente en código intermedio:

```
1:  prog ejemplo;
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: aux:=a;
13: c:=b;
14: _t1:= aux-1;
15:  aux := _t1 ;
16:  _t2 := c+1 ;
17:  c := _t2 ;
18: if aux>0 goto 12;
19: goto 20;
20: finproc;
21: read a;
22: read b;
23: _t3:= 1/b;
24: d:=_t3;
25: _t4:=1/a;
26: e:= _t4;
27: param_val a;
28: param_val b;
29: param_ref c;
30: call sumar;
31: _t5:= c*d;
32:  _t6:=c *_t5;
33: _t7:= _t6+e;
34:  c:=_t7;
35:  _t8:= c*c;
36: write _t8;
37: writeln;
38: 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
*COMPLETA_INST: código X lista_ref_código X ref_código → código
Completa las instrucciones de la lista de referencias (saltos) con la referencia dada.
*OBTEN_REF_CODIGO: 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_IDENT: → 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 implementación de la práctica completa de un curso anterior en C, Ada y Java.

8. Interfaz con el analizador léxico

En tanto no se disponga del analizador léxico simularemos su existencia a través del procedimiento NEXT_TOKEN. NEXT_TOKEN obtiene el siguiente token reconocido en el fichero de entrada dejándolo sobre la variable global LOOKAHEAD, así como la información (atributos) que en su caso tenga asociado.

Más formalmente:

NEXT_TOKEN: fichero → fichero X token

Token es un tipo de datos sobre el que están definidas las operaciones **atr_token**, **tipo_token** y **str_token:**

TIPO_TOKEN: token → ttoken (* tipo de token *)
ATR_TOKEN: token → tatributo
STR_TOKEN: token → CADENA (*de caracteres*)

9. Normas de entrega de la práctica

Aquellas prácticas que no sigan las normas no serán corregidas.

La entrega es doble:
- **fichero único comprimido con toda la práctica, a entregar usando moodle (23:55 del día límite)**
- **la documentación en papel (día siguiente a la fecha límite, en buzón o clase)**

FICHERO UNICO COMPRIMIDO CON TODA LA PRACTICA
La presencia de virus detectables por el software antivirus disponible en la red será motivo para que no se evalúe el funcionamiento del compilador.

El nombre del fichero debe incluir claramente el código de grupo, en qué máquina (PC, SUN, …) y sistema operativo (WINDOWS, LINUX, ...) es ejecutable la práctica.

Contenido necesario del fichero único:
- **documentación (ver mas abajo)**
- **leeme.txt** miembros del grupo, código del grupo, comentarios sobre compilador que se haya usado, así como instrucciones sobre cómo compilar los ficheros en 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.
- **compi.exe.** Código ejecutable del analizador sintáctico o traductor (prácticas en C o Ada).
- **compi.jar.** Fichero independiente del entorno de trabajo en que se haya desarrollado (prácticas en Java).
- **fuente.** Directorio incluyendo los ficheros que contienen el código fuente.
- **ejem$_i$.dat.** Programas de prueba que se proponen para comprobar el funcionamiento del compilador (al principio del programa deben incluirse los comentarios que expliquen el sentido del ejemplo).

DOCUMENTACIÓN
Se agradecerá una correcta redacción y se exigirá el cuidado de aspectos formales (por ejemplo, formato legible, ausencia de faltas de ortografía graves). 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. Estructura general. Autómata
- Estructura general del analizador sintáctico o traductor.
- ETDS
- Casos de prueba. Entrada y, en el caso de notas superiores al 5, salida obtenida.
- En su caso:
  - Diseño del tratamiento de errores sintácticos. Casos de prueba.
  - Gramática ampliada para el tratamiento de booleanas y ETDS actualizado.
  - Restricciones semánticas. Descripción, ETDS actualizado y casos de prueba.
  - Diseño y funcionalidad de la tabla de símbolos.
  - Llamadas a procedimientos. Descripción, ETDS actualizado y casos de prueba.
  - Array multidimensional. Descripción, ETDS actualizado y casos de prueba.
- Opcionalmente:
  Mejoras/especificidades/aspectos a resaltar
  Comentarios/valoración sobre la práctica