Procesadores de Lenguajes, Grado de Ingeniería Informática

Enunciado de la práctica obligatoria

Jaime Urquiza Fuentes

En este documento se especifica el enunciado de la práctica obligatoria de la asignatura de Procesadores de Lenguajes (Grado de Ingeniería Informática – Campus de Madrid Vicálvaro). También se proporcionan fechas de entrega de las diferentes fases de la práctica.

Práctica obligatoria

Procesadores de Lenguajes

Tabla de contenidos

Introducción

La práctica se podrá realizar en grupos de, como máximo, 3 personas. La puntuación obtenida no depende del número de integrantes del grupo, tampoco tiene por qué ser igual para todos los integrantes.

No se permite la integración de personas en un grupo después de la primera entrega.

Sí se permite la salida de personas de un grupo, dejando claro en una entrevista con el profesor, quién continúa con la práctica y quién no hace la práctica o decide empezar una nueva.

Material de entrega

Una **memoria escrita** en formato electrónico que incluya:

- Una descripción del trabajo realizado, así como cualquier anotación o característica que se desee hacer notar, **sin incluir listados fuente**.
- 8 casos de prueba de los cuales, 4 han de ser correctos y 4 erróneos, de forma que permitan observar el comportamiento del procesador.
- Recordad: ***LO BUENO, SI BREVE, DOS VECES BUENO***

Aplicación informática que implemente la funcionalidad requerida para la entrega correspondiente (léxico, sintáctico o completa):

- Ejecutable de la aplicación, que debe funcionar sobre **plataforma Windows 10 disponible en la URJC**.
- Proyecto de desarrollo completo incluyendo listados fuente de las especificaciones e implementación.
- Los ficheros asociados a los casos de prueba que aparecen en la memoria.

La calidad del material entregado es responsabilidad de los estudiantes. En caso de encontrar una entrega con virus o defectuosa será considerada suspensa.

Calificación

La calificación de la práctica se divide en tres niveles:

- **aprobado** (hasta 5), completando la parte obligatoria.
- **notable** (hasta 7), alcanzando el grado de aprobado, tratando las sentencias de control de flujo if, while-do y repeat-until de la parte opcional y notificando los errores de forma detallada (línea, columna y posible causa).
- **sobresaliente** (hasta 9,5), alcanzando el grado de notable y proporcionando recuperación de errores léxica y sintáctica así como completando toda la parte opcional.

Además se otorgará **medio punto extra** en función de la **calidad de la memoria final**.

Plazos de entrega

- **Marzo-abril de 2019** Analizadores léxico y sintáctico. Evaluación ordinaria.
- Mayo de 2019 Práctica completa. Evaluación ordinaria.
- Junio de 2019 Práctica completa. Evaluación extraordinaria.

Especificación de la práctica

La práctica consiste en el **diseño e implementación de un traductor de programas** escritos en un lenguaje de programación **similar a PASCAL** (de ahora en adelante **lenguaje fuente**), a otro **similar a C** (de ahora en adelante **lenguaje final**) Se permite utilizar herramientas de generación automática estilo Lex/Yacc, CUP, ANTLR, etc.

Parte obligatoria

Especificaciones léxicas del lenguaje fuente

Los elementos del lenguaje que aparecen entrecomillados en la gramática (que se muestra en las especificación sintáctica), deben aparecer **tal cual** (sin las dobles comillas) en cualquier programa correctamente escrito en este lenguaje, el resto de elementos se especifican a continuación.

Los **identificadores**, representados por el símbolo `identifier` de la gramática, son ristras de símbolos compuestas por letras (del alfabeto inglés, por lo tanto ni "ñ" ni "Ç"), dígitos (de base decimal) y guiones bajos "_" (underscore). Empiezan obligatoriamente por una letra. Ejemplos correctos: `contador`, `contador1`, `acumulador_total_2`.

Las **constantes numéricas** pueden ser de **dos tipos**: enteras y reales. Están representadas en la gramática por los símbolos terminales `numeric_integer_const` y `numeric_real_const`, respectivamente. Todas las ristras de dígitos (uno o más dígitos) a las que se hace referencia a continuación se especifican en base decimal:

- Las constantes numéricas **enteras** son una ristra de dígitos, opcionalmente precedida de un signo "+" o "-".
- Las constantes numéricas **reales** pueden opcionalmente ir precedidas de un signo + o – y se pueden expresar de tres formas distintas:
    - Punto fijo: dos ristras de dígitos separadas por el punto decimal.
    - Exponencial: una ristra de dígitos seguida del carácter "`e`" o "`E`", un signo "+" o "-" opcional y otra ristra de dígitos.
    - Mixto: que sería una constante real en punto fijo seguida del carácter "`e`" o "`E`", un signo "+" o "-" opcional y otra ristra de dígitos.

Ejemplos de constantes correctamente escritas:

- Enteras: `+123` , `-690` , `405` , `000078` , `-005` , `+0953`
- Reales:
    - Punto fijo: `+123.456` , `-00.69` , `45.07000`
    - Exponencial: `123E456` , `-64E-77` , `+045e16`, `003E+35`
    - Mixto: `1.23E456` , `-000.64E-77` , `+045.0e16` , `0.03E+35`

Las **constantes literales**, representadas en la gramática por el símbolo terminal `string_const`, son ristras de símbolos entre comillas simples: `'contenido de la constante literal'`. El contenido de las constantes puede ser cualquier carácter que pueda aparecer en el programa fuente. Si se desea que aparezca la comilla simple como contenido, ésta debe ir precedida de otra comilla simple, por ejemplo, el contenido de la constante: `'constante literal con una comilla '' en el contenido'` sería: `constante literal con una comilla ' en el contenido`

Existen dos formatos para los **comentarios de propósito general** dependiendo de cuántas líneas contengan. Ambos formatos de comentarios pueden aparecer antes o después de cualquier elemento del lenguaje:

- Para el caso de una sola línea el formato es: cualquier carácter que pueda aparecer en el código fuente (salvo el salto de línea) entre los símbolos `{` y `}`.
- Para el caso de varias líneas el formato es: cualquier carácter que pueda aparecer en el código fuente entre las parejas de símbolos `(*` y `*)`. Lógicamente, el contenido del comentario no puede tener los caracteres de finalización del mismo.

Especificación sintáctica del lenguaje fuente

Un programa está compuesto por dos partes: la zona de declaraciones (`DCLLIST`) y la zona de sentencias del programa principal (`SENTLIST`).

```
PRG ::= "program" identifier  ";" BLQ "."
BLQ ::= DCLLIST "begin" SENTLIST "end"
DCLLIST ::= Λ | DCLLIST DCL
SENTLIST ::= SENT | SENTLIST SENT
```

La zona de declaraciones es una lista de declaraciones de constantes, variables, procedimientos y/o funciones. Los dos últimos tienen una estructura análoga al programa principal.

```
DCL ::= DEFCTE | DEFVAR | DEFPROC | DEFFUN
DEFCTE ::= "const" CTELIST
CTELIST ::= identifier "=" SIMPVALUE ";"
      | CTELIST identifier "=" SIMPVALUE ";"
SIMPVALUE ::= numeric_integer_const | numeric_real_const
      | string_const

DEFVAR ::= "var" DEFVARLIST ";"
DEFVARLIST ::= VARLIST ":" TBAS
      | DEFVARLIST ";" VARLIST ":" TBAS
VARLIST ::= identifier | identifier "," VARLIST
DEFPROC ::=  "procedure" identifier FORMAL_PARAMLIST ";" BLQ ";"
DEFFUN ::=  "function" identifier FORMAL_PARAMLIST ":" TBAS ";" BLQ ";"
FORMAL_PARAMLIST ::= Λ | "(" FORMAL_PARAM ")"
FORMAL_PARAM ::= VARLIST ":" TBAS
      | VARLIST ":" TBAS ";" FORMAL_PARAM
TBAS ::= "INTEGER" | "REAL"
```

La zona de sentencias del programa principal es una lista de sentencias como asignaciones y llamadas a procedimientos:

```
SENT ::= ASIG ";" | PROC_CALL ";"
ASIG ::= ID ":=" EXP
ID := identifier
EXP ::= EXP OP EXP | FACTOR
OP ::= OPARIT
OPARIT ::= "+" | "-" | "*" | "div" | "mod"
FACTOR ::= SIMPVALUE | "(" EXP ")" | identifier SUBPPARAMLIST
SUBPPARAMLIST ::= lambda | "(" EXPLIST ")"
EXPLIST ::= EXP | EXP "," EXPLIST
PROC_CALL ::= identifier SUBPPARAMLIST
```

Especificación de la traducción dirigida por la sintaxis

El objetivo es traducir el código en lenguaje fuente a un código en lenguaje final, cuya gramática proporcionaremos más adelante. El código en lenguaje final estará compuesto por un solo fichero de extensión .c y cuyo nombre será el mismo que el del fichero de entrada. Así, el fichero de prueba "ejemplo.pas" producirá como salida el fichero "ejemplo.c".

Los comentarios que se encuentren en el lenguaje fuente deben ser ignorados (no aparecerán en el lenguaje final). Por otro lado, tanto los identificadores como las constantes numéricas y literales se expresarán en el lenguaje final de forma **idéntica** a como aparecen en el lenguaje fuente. Esto no ocurre de forma completa con otros elementos que se detallan a continuación:

- Palabras reservadas sobre tipos de datos básicos como "`INTEGER`" y "`REAL`", que aparecerán en el lenguaje final como "*`int`*" y "*`float`*" respectivamente.
- Operadores aritméticos. Mientras "+", "−" y "`*`" se escriben de forma idéntica en el lenguaje fuente y el lenguaje final, "`div`" y "`mod`" del lenguaje fuente se escriben en el lenguaje final como "`/`" y "`%`" respectivamente.

La gramática del lenguaje final se presenta a continuación. Un programa está compuesto por dos partes, la declaración de constantes (representada en la gramática por el símbolo *`DEFINES`*) y un conjunto de sucesivas declaraciones de funciones (representadas en la gramática por el símbolo *`PARTES`*).

```
PROGRAM ::= DEFINES PARTES
DEFINES ::= λ | "#define" ident CTES DEFINES
CTES ::= constint | constfloat | constlit
PARTES ::= PART PARTES | PART
PART ::= TYPE RESTPART
RESTPART ::= ident "(" LISTPARAM ")" BLQ
| ident "(" "void" ")" BLQ
BLQ ::= "{" SENTLIST "}"
LISTPARAM ::= LISTPARAM "," TYPE ident | TYPE ident
TYPE ::= "void" | "int" | "float"
```

En el lenguaje final, la declaración de constantes se hace únicamente al comienzo del programa, independientemente de dónde estén en el programa fuente, y según se especifica en la gramática correspondiente en el símbolo *`DEFINES`*. Cada declaración de constante en el lenguaje final aparecerá en una línea nueva. Es decir, no pueden aparecer varias sentencias "`#define`" en la misma línea.

Además, como se ha dicho, en el lenguaje final sólo hay funciones, su declaración se especifica en la gramática correspondiente en el símbolo *`PART`*. Los procedimientos del lenguaje fuente se declararán como funciones del lenguaje final cuyo tipo devuelto es "`void`". Así, en el lenguaje final, la diferencia entre procedimientos y funciones es que los primeros devuelven el tipo "`void`", mientras que las segundas devuelven el tipo "*`int`*" o "*`float`*". Del mismo modo, el uso de los elementos anteriores se especifica en la gramática correspondiente en los símbolos *`SENT`* y *`FACTOR`*. A continuación se muestran unos ejemplos de declaración y uso de estos elementos.

Entrada fuente	Salida en lenguaje final

{declaración de función CON parámetros}
FUNCTION fun1 (a:INTEGER ; b:REAL) : INTEGER
BEGIN
…
proc2; {llamada de procedimiento}
END;
{declaración de función SIN parámetros}
FUNCTION fun2 : REAL
BEGIN
…
proc1(1.3 , -4); {llamada de procedimiento}
END;
{declaración de procedimiento CON parámetros}
PROCEDURE proc1 (c:REAL ; d:INTEGER)
BEGIN
…
valor := fun1(1 , 1.0); {llamada de funcion}
END
{declaración de procedimiento SIN parámetros}
PROCEDURE proc2
BEGIN
…
valor := fun2; {llamada de funcion}
END
	int fun1 (int a , float b)
	{
	…
	proc2();
	}
	
	float fun2 (void)
	{
	…
	proc1(1.3 , -4);
	}
	
	void proc1 (float c , int d)
	{
	…
	valor = fun(1 , 1.0);
	}
	
	void proc2 (void)
	{
	…
	valor = fun2();
	}

Nótese que según la gramática fuente, en la declaración de procedimientos y funciones, es posible compactar la declaración de parámetros de la siguiente forma:

```
procedure procVariosParam(a, b : INTEGER; c : REAL)
```

Sin embargo, para la parte obligatoria de la práctica no hace falta contemplar esta posibilidad, por ello solo se tratarán declaraciones de parámetros sin compactar, el caso equivalente al anterior sería:

```
procedure procVariosParam(a : INTEGER; b : INTEGER; c : REAL)
```

En el lenguaje fuente, las sentencias del programa principal lo conforman aquellas sentencias dentro del último bloque BEGIN-END declarado, aquel terminado con ".". En el lenguaje final el programa principal se declarará como un procedimiento cuyo nombre es "main" que además no tendrá parámetros formales. Así, las sentencias del programa principal del lenguaje fuente irán dentro de este procedimiento en el lenguaje final.

```
void main ( void )
{
      …
}
```

Pasamos a ver la parte de la gramática del lenguaje final correspondiente al contenido de funciones y procedimientos. Dentro de las funciones y procedimientos se pueden encontrar sentencias de declaración de variables, asignación, retorno y llamadas a funciones y procedimientos.

```
SENTLIST ::= SENTLIST SENT | SENT
SENT ::= TYPE LID ";" | ident "=" EXP ";" | ident "(" LEXP ")" ";"
| ident "(" ")" ";" | "return" EXP ";"
LID ::= ident | LID "," ident
LEXP ::= EXP | LEXP "," EXP
EXP ::= EXP OP EXP | FACTOR
OP ::= "+" | "-" | "*" | "/" | "%"
FACTOR ::= ident "(" LEXP ")" | ident "(" ")"
| "(" EXP ")" | ident | CTES
```

Las variables declaradas dentro del programa principal en el lenguaje fuente aparecerán al comienzo del procedimiento "main" del lenguaje final. Todas las funciones y procedimientos declarados dentro del programa principal en el lenguaje fuente aparecerán como funciones y procedimientos independientes en el lenguaje final, siempre declarados antes del procedimiento "main". Finalmente, las variables declaradas en cada función o procedimiento del lenguaje fuente aparecerán al comienzo dentro de su correspondiente función o procedimiento del lenguaje final.

En el lenguaje fuente, para devolver un valor como resultado de una función se usan sentencias de asignación. Estas sentencias tienen una particularidad: el identificador en la parte izquierda de la asignación es el nombre de la función dentro de la que están ubicadas. En el lenguaje final serán sustituidas por sentencias de retorno que comienzan por la palabra reservada "*return*" seguida de la expresión existente en la parte derecha de la sentencia de asignación del lenguaje fuente.

Entrada fuente	Salida en lenguaje final

`FUNCTION fun ( … ) : INTEGER`
`BEGIN`
`  …`
`  fun := a + 4; {valor resultante de fun}`
`END`
	`int fun ( … )`
	`{`
	`  …`
	`  return a + 4;`
	`}`

Para terminar con la parte obligatoria, el código en lenguaje final generado tendrá la siguiente propiedad en cuanto a tabulación se refiere: todas las sentencias entre los símbolos "{" y "}", se colocarán en un nivel de tabulación mayor (indentación a la derecha) que estos.

Parte opcional

Especificación sintáctica del lenguaje fuente

Sentencias de control de flujo

Las sentencias de control de flujo se basan en la comprobación de expresiones condicionales, cuya gramática es:

```
EXPCOND ::= EXPCOND OPLOG EXPCOND | FACTORCOND
OPLOG ::= "or" | "and"
FACTORCOND ::= EXP OPCOMP EXP | "(" EXP ")" | "not" FACTORCOND
OPCOMP ::= "<" | ">" | "<=" | ">=" | "="
```

La gramática del lenguaje fuente para las sentencias de control de flujo es:

```
SENT ::= ...
       | "if" EXPCOND "then" BLQ "else" BLQ
       | "while" EXPCOND "do" BLQ
       | "repeat" BLQ "until" EXPCOND ";"
       | "for" identifier ":=" EXP INC EXP "do" BLQ
INC ::= "to" | "downto"
```

Distinguir entre librerías y programas

Si el programa fuente comienza por la palabra reservada "**UNIT**" en vez de "**PROGRAM**" significa que es una librería. La gramática sería la siguiente:

```
PRG ::= … | "UNIT" identifier ";" DCLLIST "."
```

Especificación de la traducción dirigida por la sintaxis

Sentencias de control de flujo

Como se explicaba anteriormente, las sentencias de control de flujo se basan en la comprobación de expresiones condicionales. La gramática del lenguaje final de estas expresiones es:

```
LCOND ::= LCOND OPL LCOND | COND | "!" COND
OPL ::= "||" | "&&"
COND ::= EXP OPR EXP
OPR::= "==" | "<" | ">" | ">=" | "<="
```

Nótese que existen diferencias con respecto al lenguaje fuente. Así los elementos del lenguaje fuente "**or**", "**and**", "**not**" y "**=**" se corresponden respectivamente con los siguientes del lenguaje final: "**||**", "**&&**", "**!**" y "**==**". La gramática del lenguaje final para las sentencias de control de flujo es:

```
SENT ::= …
       | "if" "(" LCOND ")" BLQ "else" BLQ
       | "while" "(" LCOND ")" BLQ
       | "do" BLQ "until" "(" LCOND ")"
       | "for" "(" ident "=" EXP ";" LCOND ";" ident "=" EXP ")" BLQ
```

Mientras que las tres primeras sentencias tienen una traducción directa, la sentencia `for` debe traducirse como se explica a continuación. Como se puede ver en la gramática del lenguaje fuente, esta sentencia solo permite decrementos o incrementos unitarios, especificado mediante el símbolo "`INC`" de la gramática. La traducción deberá seguir el siguiente esquema:

Entrada fuente	Salida en lenguaje final

FOR *cont* := exp1 TO exp2 DO
BEGIN
Sentencias-BLQ
END
	for(*cont*=exp1; *cont*<exp2+1; *cont*=*cont*+1)
	{
	Sentencias-BLQ
	}

FOR *cont* := exp1 DOWNTO exp2 DO
BEGIN
Sentencias-BLQ
END
	for(*cont*=exp1; *cont*>exp2-1; *cont*=*cont*-1)
	{
	Sentencias-BLQ
	}

La variable `cont` representa al contador del bucle, cuyos valores inicial y final serán los representados por `exp1` y `exp2`. Finalmente, el incremento ("`TO`") o decremento ("`DOWNTO`") se traducirán por los operadores "`<`" o "`>`" de la condición de parada en el lenguaje final, así como las sentencias de actualización de incremento o decremento.

Distinguir entre librerías y programas

Como se dijo anteriormente, si el programa fuente comienza por la palabra reservada "**UNIT**" en vez de "**PROGRAM**" significa que es una librería. En el lenguaje final, una librería implica que **no hay ninguna función con identificador "main"**. Tan solo será necesario que el programa en el lenguaje final comience por la siguiente construcción: `// Librería: ident` Donde `ident` es el identificador que acompaña a la palabra "`UNIT`" en el programa fuente.

Mejora de la generación de declaraciones de funciones y procedimientos

Para la parte optativa sí es necesario considerar la posibilidad de compactar la declaración de parámetros de los procedimientos y funciones del lenguaje fuente. Como se puede ver en la gramática del lenguaje final, en dicho lenguaje no existe esta posibilidad, por lo tanto habrá que traducir esas declaraciones de forma análoga al siguiente ejemplo:

Entrada fuente

```
… procedure procVariosParam(a, b : INTEGER; c : REAL) …
```

Salida en lenguaje final

```
… void procVariosParam( int a, int b, float c) …
```