Procesadores de Lenguajes, Grado de Ingeniería Informática

Enunciado de la práctica obligatoria

Jaime Urquiza Fuentes, Emilio López Cano y Ana Gómez Pérez

En este documento se especifica el enunciado de la práctica obligatoria de la asignatura de Procesadores de Lenguajes (Grado de Ingeniería Informática y sus respectivas titulaciones dobles). En concreto, tanto la parte obligatoria como la opcional del analizador léxico, sintáctico y la traducción dirigida por la sintaxis, así como las 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 disponible en la URJC**.
- Especificación ANTLR de la práctica.
- Proyecto de desarrollo completo, comprimido en un archivo zip, 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 de forma que no pueda ser evaluada 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, proporcionando recuperación de errores léxica y sintáctica, completando al menos dos sentencias de control de flujo de la parte opcional, implementando los enlaces entre declaraciones de identificadores y su utilización, e indentando las sentencias incluidas dentro de los bloques correspondientes a las sentencias de control de flujo completadas.
- **sobresaliente** (hasta 9,5), alcanzando el grado de notable y notificando los errores en idioma español de forma detallada (línea, columna y posible causa) 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

- **25 de abril de 2021**. Analizadores léxico y sintáctico. Evaluación ordinaria.
- 30 de mayo de 2021. Práctica completa. Evaluación ordinaria.
- Junio de 2020. Práctica completa. Evaluación extraordinaria.

Especificación de la práctica

La práctica consiste en el diseño e implementación de un **visualizador de código fuente** para un lenguaje de programación. Las especificaciones léxica y sintáctica del lenguaje a procesar se describen en las siguientes secciones. Se permite utilizar herramientas de generación automática estilo ANTLR.

Parte obligatoria

Especificaciones léxicas del lenguaje fuente

Los elementos del lenguaje que aparecen entrecomillados en la gramática (que se muestra en la 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 **`IDENTIFICADOR`** de la gramática, son ristras de símbolos compuestas por letras (del alfabeto inglés, por lo tanto, ni "ñ" ni "Ç" ni vocales acentuadas o con diéresis), dígitos (de base decimal) y guiones bajos "_" (underscore). Empiezan obligatoriamente por una letra o un guion bajo. Ejemplos correctos: `contador`, `contador1`, `acumulador_total_2`, `_contador3`. No se permite utilizar como identificadores las palabras reservadas del lenguaje.

Las **constantes numéricas** pueden ser de **dos tipos**: enteras y reales. Están representadas en la gramática por los símbolos terminales `CONSTENTERO` y `CONSTREAL`, 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** pueden ser de **dos tipos** y se pueden especificar en **dos bases distintas**.

- Bases posibles: decimal y hexadecimal.
  - Las constantes numéricas en **base decimal** (10), estarán compuestas por los dígitos decimales, del 0 al 9, además de los símbolos necesarios en función del tipo al que pertenezcan (enteras o reales).
  - Las constantes numéricas en **base hexadecimal** (16), **siempre comienzan con el símbolo "$"** y estarán compuestas por los dígitos decimales, del 0 al 9 y las letras de la "A" a la "F" (solo en mayúsculas), además de los símbolos necesarios en función del tipo al que pertenezcan (enteras o reales).
- Tipos posibles: enteros y reales.
  - Las constantes numéricas **enteras** son una ristra de dígitos, opcionalmente precedidas de un signo + o -.
  - Las constantes numéricas **reales** son dos ristras de dígitos, opcionalmente precedidas de un signo + o - y separadas por el punto decimal.
- Ejemplos de constantes correctamente escritas:
  - Enteras:
    - Base decimal: +123 , -69 , 45
    - Base hexadecimal: $+123 , $-A6F9 , $FFFF
  - Reales:
    - Base decimal: +123.456 , -0.69 , 45.0
    - Base hexadecimal: $+123.0 , $-E.A6F9 , $0.FFFF

Las **constantes literales**, representadas en la gramática por el símbolo terminal `CONSTLIT`, son ristras de símbolos delimitadas por comillas simples: `'contenido de la constante literal'` o comillas dobles: "`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 algún símbolo delimitador aparezca como contenido existen dos opciones:

- Delimitar con un símbolo cuando se quiera que aparezca el otro como contenido.
- Duplicar el símbolo que se quiere que aparezca en el contenido cuando éste es el usado como delimitador de la constante.

Constante	Valor
`'comilla doble " dentro'`	`comilla doble " dentro`
`"comilla simple ' dentro"`	`comilla simple ' dentro`
`'comilla simple '' dentro'`	`comilla simple ' dentro`
`"comilla doble "" dentro"`	`comilla doble " dentro`
`'comilla doble " y simple '' dentro'`	`comilla doble " y simple ' dentro`
`"comilla simple ' y doble "" dentro"`	`comilla simple ' y doble " dentro`

Los comentarios son de dos tipos, de línea o multilínea:

- El formato de los **comentarios de línea** es: "%%" (dos símbolos de porcentaje) seguidos de cualquier carácter que pueda aparecer en el código fuente salvo el fin de línea, que marca su final. Este tipo de comentarios pueden aparecer en cualquier punto del código fuente.
- El formato de los **comentarios multilínea** es: cualquier carácter que pueda aparecer en el código fuente, fin de línea incluido, encerrado entre las dos parejas de símbolos "%-" y "-%". Este tipo de comentarios pueden aparecer en cualquier punto del código fuente.

Especificación sintáctica del lenguaje fuente

Un programa está compuesto por diferentes partes que coinciden con declaraciones de funciones y/o procedimientos (`part`).

```
program ::= part program | part
part ::= "funcion" type restpart | "procedimiento" restpart
restpart ::= IDENTIFICADOR "(" listparam ")" blq
  | IDENTIFICADOR "(" ")" blq
listparam ::= listparam "," type IDENTIFICADOR | type IDENTIFICADOR
type ::= "entero" | "real" | "caracter"
blq ::= "inicio" sentlist "fin"
```

Cada declaración de función y/o procedimiento contiene, además de la cabecera de declaración, un conjunto de sentencias (`sent`) que pueden ser de tres tipos: declaraciones variables, sentencias de asignación y llamadas a procedimientos.

```
sentlist ::= sentlist sent | sent
sent ::= type lid ";" | IDENTIFICADOR asig exp ";" | "return" exp ";"
  | IDENTIFICADOR "(" lid ")" ";" | IDENTIFICADOR "(" ")" ";"
lid ::= IDENTIFICADOR | IDENTIFICADOR "," lid
asig ::= "=" |  "+=" | "-=" | "*=" | "/="
exp ::= exp op exp | IDENTIFICADOR "(" lid ")" | "(" exp ")"
  | IDENTIFICADOR | CONSTENTERO | CONSTREAL | CONSTLIT
op ::= "+" | "-" | "*" | "/"
```

Tanto en esta parte como en la parte opcional hay que asegurarse de que la gramática usada con el generador correspondiente, **ANTLR** es **LL(1)**.

Especificación de la traducción dirigida por la sintaxis

El objetivo es visualizar de forma más manejable el código recibido en el fichero de entrada. La visualización se hará produciendo una página web con la estructura descrita a continuación.

La página web resultante estará compuesta por un solo fichero de extensión .html y cuyo nombre será el mismo que el fichero de entrada al que se le añade la extensión .html. Así, el fichero de prueba "ejemplo.x" producirá como salida el fichero "ejemplo.x.html". La página web tendrá como título (elemento HTML `<TITLE>...</TITLE>`) el nombre del fichero procesado.

Lo primero que aparecerá será un título de primer nivel (elemento HTML `<H1>...</H1>`) con el texto "Programa: XXXXX", donde XXXXX es de nuevo el nombre del fichero procesado. A continuación, se mostrará el listado de funciones y procedimientos, con el encabezado de tipo segundo nivel (elemento HTML `<H2>...</H2>`) "Funciones y procedimientos" seguido de una lista no enumerada (elemento HTML `<UL>...</UL>`) donde cada ítem (elemento HTML `<LI>...</LI>`) será la cabecera de cada función o procedimiento existente en el fichero de entrada, enlazada (elemento HTML `<A HREF="...">...</A>`) al código de la correspondiente función o procedimiento. Esta cabecera estará compuesta por:

- Tipo devuelto, nombre y listado de parámetros (con tipo y nombre) entre paréntesis, en el caso de **funciones**.
- Nombre y listado de parámetros (con tipo y nombre) entre paréntesis, en el caso de **procedimientos**.

A continuación, aparecerá el código de todas las funciones y procedimientos. El formato de cada uno de ellos será el siguiente:

- En primer lugar, aparecerá una barra horizontal (elemento HTML `<HR/>`), a continuación, un "ancla" (elemento HTML `<A NAME="xxx">`, donde xxx será un nombre único que identifique a la función) que sirva como destino de los enlaces al código de la función o procedimiento, y finalmente la cabecera tal y como está en el fichero de entrada con los estilos descritos a continuación.
- Todas las sentencias entre las palabras reservadas "inicio" y "fin" aparecerán con un nivel de indentación mayor. Para ello se puede utilizar la etiqueta `<DIV style="text-indent: Xcm"> ... </DIV>` que indentaría todo su contenido X centímetros.
- Cada sentencia se mostrará en una línea distinta. Entendiendo como sentencia la cabecera, las palabras reservadas "inicio" y "fin" así como las construcciones sintácticas del símbolo gramatical SENT. El elemento `DIV` de HTML incluye de forma implícita un salto de línea. Para incluir un salto de línea de forma explícita se puede usar el elemento HTML `<BR/>`.
- Los siguientes elementos tendrán un estilo especial, previamente definido en la cabecera:
  - constantes: `<SPAN CLASS="cte">...</SPAN>`
  - palabras reservadas: `<SPAN CLASS="palres">...</SPAN>`
  - identificadores: `<SPAN CLASS="ident">...</SPAN>`
- Después de cada función o procedimiento aparecerán dos enlaces (elemento HTML `<A HREF="...">...</A>`), uno al comienzo del código de la función o procedimiento actual, y otro al comienzo de la página web.

Nota sobre HTML: un enlace a un ancla definida con el nombre `"pepito"` *debe especificar el destino con el carácter "#" delante,* `"#pepito"`. *Así el enlace a un ancla definida como* `<A NAME="pepito">` *será de la forma* `<A HREF="#pepito">Texto del enlace</A>`

Parte opcional

En esta parte se explican las ampliaciones de la parte obligatoria, estas ampliaciones consistirán en nuevos tokens a reconocer que, junto con nuevas construcciones sintácticas, habrá que integrar en la gramática de la parte obligatoria. Ambos, los nuevos tokens y las nuevas construcciones sintácticas, provocarán también nuevos requisitos de traducción dirigida por la sintaxis. Todo esto se describe en esta sección. De forma general, las ampliaciones en la gramática se describirán aumentando las producciones existentes e incluyendo otras nuevas. Así la producción:

```
A ::= … | D
```

Significa que al símbolo A ya existente en la gramática se le añade (este es el significado de los ...) una nueva producción A ::= D, manteniendo todas las que tenía en la gramática de la parte obligatoria.

Especificación léxica y sintáctica del lenguaje fuente

La ampliación de la parte sintáctica consiste en añadir las siguientes sentencias de control del flujo de ejecución.

```
sent ::= "bifurcacion" "(" lcond ")" "entonces" blq "sino" blq
  | "buclepara" "(" IDENTIFICADOR asig exp ";" lcond ";" IDENTIFICADOR asig
       exp ")" blq
  | "buclemientras" "(" lcond ")" blq
  | "bucle" blq "hasta" "(" lcond ")"
  | blq

lcond ::= lcond opl lcond | cond | "no" cond
cond ::= exp opr exp | "cierto" | "falso"
opl ::= "y" | "o"
opr::= "==" | "<>" | "<" | ">" | ">=" | "<="
```

Especificación de la traducción dirigida por la sintaxis

Las sentencias incluidas en los bloques (símbolo `blq` de la gramática) deben tener un nivel mayor de indentación que las palabras reservadas "inicio" y "fin" que delimitan ese bloque.

Las variables, procedimientos y funciones utilizadas en el código deben estar enlazadas (elemento HTML `<A HREF="...">...</A>`) con su correspondiente declaración:

- Cabecera en caso de tratarse de una función, procedimiento o parámetro.
- Sentencia de declaración en caso de tratarse de una variable declarada dentro del código de una función o procedimiento.

Por lo tanto, para la utilización de los identificadores habrá que sustituir el estilo "ident" usado en la parte obligatoria por el elemento de enlace descrito anteriormente, pero sólo en su utilización. En la declaración de los identificadores habrá que incluir un "ancla" (elemento HTML `<A NAME="xxx">`) que sirva como destino de los enlaces, junto con el estilo de los identificadores `<SPAN CLASS="ident">...</SPAN>` usado hasta ahora.

En caso de que aparezca una función o procedimiento con nombre "Principal", debe comprobarse que sólo existe una:

- Si sólo existe una, cuando se genere el código HTML, su código aparecerá el primero después del listado inicial de funciones y procedimientos.
- Si no, habrá que comunicar un error y no generar ningún código HTML como salida del programa.