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 y sus respectivas titulaciones dobles). En concreto, tanto la parte obligatoria como la opcional del analizador léxico y sintáctico, 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 (.jar), que debe funcionar sobre **plataforma Windows disponible en la URJC**.
- Especificación ANTLR de la práctica.
- 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 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 del flujo de ejecución (considerando su anidamiento).
- **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

- 15 de abril de 2023. Analizadores léxico y sintáctico. Evaluación ordinaria.
- 21 de mayo de 2023. Práctica completa. Evaluación ordinaria.
- **3 julio de 2022**. 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 similar a C. 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 los elementos se especifican a continuación.

Los **identificadores**, representados por el símbolo `IDENTIFIER` de la gramática, son ristras de símbolos compuestas por letras minúsculas (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. No pueden estar formados únicamente por guiones bajos. Ejemplos correctos: `contador`, `contador1`, `acumulador_total_2`, `_contador3`. No se permite utilizar como identificadores las palabras reservadas del lenguaje.

Un caso especial de identificadores son las **definiciones de constantes**, representados por el símbolo `CONST_DEF_IDENTIFIER` en la gramática, son un caso especial de `IDENTIFIER` que cumplen las mismas reglas más una: las letras deben ser mayúsculas. Ejemplos correctos: `CONTADOR`, `CONTADOR1`, `ACUMULADOR_2`, `_CONTADOR3`.

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.
  - Punto inicial: un punto seguido de una ristra de dígitos.
  - 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 o punto inicial 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`
  - Punto inicial: `+.456`, `-.69`, `.07000`
  - Exponencial: `123E456`, `-64E-77`, `+045e16`, `003E+35`

- Mixto: `1.23E456 , -000.64E-77 , -.64E-77, +045.0e16 , 0.03E+35, .03E+35`

Las **constantes literales**, representadas en la gramática por el símbolo terminal `STRING_CONST`, 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.
- Insertar una barra backslash (\ ) seguida del símbolo que se desea que aparezca en el contenido cuando éste es el usado como delimitador de la constante.

A continuación, se ofrecen algunos ejemplos:

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 " o \" y simple \' dentro'`	`comilla doble " o " y simple ' dentro`
`"comilla simple ' o \' y doble \" dentro"`	`comilla simple ' o ' y doble " dentro`

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 detrás de una doble barra //. El comentario termina cuando se encuentra un salto de línea.
- 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 tres partes: la zona de declaraciones de constantes y variables (`dcllist`), zona de declaración e implementación de funciones (`funlist`) y la zona de sentencias del programa principal (`sentlist`).

```
program ::= dcllist funlist sentlist
dcllist ::= Λ | dcllist dcl
funlist ::= Λ | funlist funcdef
sentlist::= mainhead "{" code "}"
```

La zona de declaraciones es una lista de declaraciones de constantes.

```
dcl ::= ctelist | varlist
ctelist ::= "#define" CONST_DEF_IDENTIFIER simpvalue "\n" | ctelist
"#define"  CONST_DEF_IDENTIFIER simpvalue "\n"
simpvalue ::= NUMERIC_INTEGER_CONST | NUMERIC_REAL_CONST
      | STRING_CONST
varlist ::= vardef ";"| varlist vardef ";"
vardef::= tbas IDENTIFIER ";" | tbas IDENTIFIER "=" simpvalue ";"
tbas ::= "integer" | "float" | "string" | tvoid
```

```
tvoid ::= "void"
```

La zona de implementación de funciones es una lista de implementaciones de funciones con una estructura análoga al programa principal.

```
funcdef::=  funchead "{" code "}"
funchead::= tbas IDENTIFIER  "(" typedef ")"
typedef ::= ʎ | typedef tbas IDENTIFIER
```

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

```
mainhead ::= tvoid "Main"  "(" typedef ")"
code ::= ʎ | code sent
sent ::= asig ";" | funccall ";"
asig ::= IDENTIFIER "=" exp
exp ::= exp op exp | factor
op ::= "+" | "-" | "*" | "DIV" | "MOD"
factor ::= simpvalue | "(" exp ")" | funccall
funccall ::= IDENTIFIER subpparamlist
subpparamlist ::= ʎ | "(" explist ")"
explist ::= exp | exp "," explist
```

Tanto en esta parte como en la parte opcional hay que asegurarse de que la gramática usada con **ANTLR**, es LL(1) y se especifica en **notación BNF**.