Práctica. Tercera Fase

Desarrollo de constructores de ASTs para Tiny

En esta tercera parte debe realizarse el desarrollo de constructores de ASTs descendentes y ascendentes para **Tiny**, así como implementar un procesamiento sencillo. Para ello, deberá entregarse:

- Una memoria con las siguientes secciones:
  - Portada en la que aparezcan los nombres y apellidos de los integrantes del grupo, y el número de grupo.
  - Especificación de la sintaxis abstracta de **Tiny** mediante la enumeración de las signaturas (cabeceras) de las funciones constructoras de ASTs.
  - Especificación del constructor de ASTs mediante una gramática s-atribuida.
  - Acondicionamiento de dicha especificación para permitir la implementación descendente.
  - Especificación de un procesamiento que imprima los tokens del programa leído, uno en cada línea, pero omitiendo los paréntesis redundantes en las expresiones (es decir, aquellos paréntesis que, eliminados, no cambian el significado de dichas expresiones). Las palabras reservadas se escribirán con todas las letras en minúscula, y entre ángulos (<…>). El fin de fichero se escribirá como <EOF>.
- Una implementación orientada a objetos en Java de la sintaxis abstracta de Tiny, preparada tanto para soportar **programación recursiva**, como para soportar procesamiento mediante el patrón **visitante**.
- Una implementación descendente del constructor de ASTs desarrollada con **javacc**.
- Una implementación ascendente de dicho constructor desarrollada con CUP y **jflex**.
- Tres implementaciones del procesamiento pedido: (i) una utilizando **programación recursiva**; (ii) una utilizando el patrón **intérprete**; (iii) una tercera utilizando el patrón **visitante**.
- Un programa principal que integre ambos constructores, y también los procesamientos desarrollados. Dicho programa vendrá dado por una clase `DomJudge`, situada en el paquete por defecto, que leerá datos por la entrada estándar, y escribirá los resultados por la salida estándar, de acuerdo con las especificaciones dadas en el **apéndice A.**

Fecha límite de entrega: **Viernes 28 de marzo de 2025, a las 11:59 pm**.

Modo de entrega: A través del campus virtual, en un único .zip. Dicho archivo debe contener: (i) un documento PDF `memoria.pdf` con la memoria requerida; (ii) una carpeta `implementacion`, en el interior de la cuál debe incluirse toda la implementación requerida; y (iii) una carpeta `pruebas` con distintos programas de prueba que permitan probar la implementación. La entrega debe ser realizada solamente por un miembro del grupo.

Las implementaciones deberán, además, entregarse a través del juez DomJudge de la asignatura (problema `6 PRUEBA CONS: AST`). Se seguirán los mismos requisitos de organización del .zip que para los analizadores sintácticos (clase `DomJudge` en la raíz de los fuentes, en el paquete por defecto, archivos `ruta_jflex`, `ruta_javacc`, `ruta_cup`, `cup_options` según proceda).

Apéndice A. Comportamiento del programa de prueba

- La entrada esperada por el juez comienza por una línea de selección del método de construcción de ASTs. A continuación, aparece el programa Tiny a procesar.
- La línea de selección del constructor de ASTs contiene uno de los dos siguientes caracteres:
  - `a`: Este carácter indica que el AST debe ser construido mediante el constructor ascendente implementado con cup+jflex.
  - `d`: Este carácter indica que el AST debe ser construido mediante el constructor descendente implementado con javacc.
- Por tanto, el programa principal debe comenzar leyendo el primer carácter del archivo. Dependiendo de este carácter, aplicará uno u otro constructor para construir el AST asociado al programa Tiny que viene a continuación.
- El constructor, aparte de construir el AST, debe imprimir la misma traza que imprimía el correspondiente analizador en la FASE 2. Antes de activar el constructor, el programa principal escribirá una línea con alguno de los siguientes títulos:
  - `CONSTRUCCION AST ASCENDENTE`, si el constructor aplicado ha sido el ascendente.
  - `CONSTRUCCION AST DESCENDENTE`, si el constructor aplicado ha sido el descendente.
- En caso de que la construcción haya tenido éxito (es decir, no se haya interrumpido debido a un error), el programa realizará la impresión del AST aplicando cada uno de los tres métodos de procesamiento

requeridos en esta fase (primero el recursivo, después el basado en el patrón intérprete, por último, el basado en el patrón visitante). Antes de realizar cada impresión, escribirá una línea con un título, indicando el método aplicado:

- `IMPRESION RECURSIVA`, para la impresión llevada a cabo mediante programación recursiva.
- `IMPRESION INTERPRETE`, para la impresión llevada a cabo mediante el patrón intérprete.
- `IMPRESION VISITANTE`, para la impresión llevada a cabo mediante el patrón visitante.

- Aparte de imprimir cada elemento, de acuerdo con las indicaciones dadas en el enunciado, el procesamiento deberá acompañar alguno de estos elementos con información de su vínculo (fila, columna) en el texto fuente. Más concretamente, se mostrará esta información para:
  - El identificador en las declaraciones de tipo y de variable.
  - Los identificadores en las definiciones de tipo.
  - El identificador en las declaraciones de procedimiento.
  - El identificador en los parámetros formales.
  - Los campos en las definiciones de tipos *struct*.
  - El entero que representa la dimensión en tipos arrays (en este caso, la información se pondrá junto al corchete de cierre:]).
  - El identificador del procedimiento invocado en una instrucción `call`.
  - Los operadores de las expresiones binarias y unarias (niveles 0 a 5).
  - El corchete de apertura en una expresión de indexación de un elemento de un array.
  - El identificador de campo en una expresión de acceso a campo de registro.
  - El operador ^ en una expresión de acceso a objeto apuntado.
  - Los literales en expresiones básicas.
  - Los identificadores en expresiones básicas.
- Cada vínculo al texto fuente se escribe como `$f:`*num fila*`,c:` *num col*`$`
- **Ejemplos**

Fila entrada	Texto fuente	Impresión	Fila entrada	Texto fuente	Impresión

3	`int[9] x;`	`<int>`
		`[`
		`9`
		`]$f:3,c:5$`
		`x`**`$f:3,c:8$`**
		`;`
			12	`5 + 6.0;`	`@`
					`5`**`$f:12,c:3$`**
					`+`**`$f:12,c:5$`**
					`6.0`**`$f:12,c:7$`**
					`;`

5	`type struct {`
	`  int x,`
	`  int y`
	`} z;`
		`<struct>`
		`{`
		`<int>`
		`x`**`$f:6,c:7$`**
		`,`
		`<int>`
		`y`**`$f:7,c:7$`**
		`}`
		`z`**`$f:8,c:3$`**
		`;`
			18	`@ x[56];`	`@`
					`x`**`$f:18,c:3$`**
					`[`**`$f:18,c:4$`**
					`56`**`$f:18,c:5$`**
					`]`
					`;`

9	`proc x(int i) {}`	`<proc>`
		`x`**`$f:9,c:6$`**
		`(`
		`<int>`
		`i`**`$f:9,c:12$`**
		`)`
		`{`
		`}`
			19	`@ x.y;`	`@`
					`x`**`$f:19,c:3$`**
					`.`
					`y`**`$f:19,c:5$`**
					`;`

20	`@ 5^;`	`@`
		`5$f:20,c:3$`
		`^$f:20,c:4$`
		`;`
			21	`call p()`	`<call>`
					`p$f:21,c:6$`
					`(`
					`)`

- **Un ejemplo completo**

Entrada	Salida

a
## Expresion con parentesis redundantes 	
{
@ ((((((((((not x) / (- 7)) and x) + 8) < 9) = 20)))))
}
	CONSTRUCCION AST ASCENDENTE
	{
	@
	(
	(
	(
	(
	(
	(
	(
	(
	(
	(
	<not>
	x
	)
	/
	(
	-
	7
	)
	)
	<and>
	x
	)
	+
	8
	)
	<
	9
	)
	=
	20
	)
	)
	)
	)
	)
	}
	<EOF>
	IMPRESION RECURSIVA
	{
	@
	<not>$f:4,c:14$
	x$f:4,c:18$
	/$f:4,c:21$
	-$f:4,c:24$
	7$f:4,c:26$
	<and>$f:4,c:30$
	x$f:4,c:34$
	+$f:4,c:37$
	8$f:4,c:39$
	<$f:4,c:42$
	9$f:4,c:44$
	=$f:4,c:47$
	20$f:4,c:49$
	}
	IMPRESION INTERPRETE
	{
	@
	<not>$f:4,c:14$
	x$f:4,c:18$
	/$f:4,c:21$
	-$f:4,c:24$
	7$f:4,c:26$
	<and>$f:4,c:30$
	x$f:4,c:34$
	+$f:4,c:37$
	8$f:4,c:39$
	<$f:4,c:42$
	9$f:4,c:44$
	=$f:4,c:47$
	20$f:4,c:49$
	}
	IMPRESION VISITANTE
	{
	@
	<not>$f:4,c:14$
	x$f:4,c:18$
	/$f:4,c:21$
	-$f:4,c:24$
	7$f:4,c:26$
	<and>$f:4,c:30$
	x$f:4,c:34$
	+$f:4,c:37$
	8$f:4,c:39$
	<$f:4,c:42$
	9$f:4,c:44$
	=$f:4,c:47$
	20$f:4,c:49$
	}

La traza emitida por el constructor durante la construcción del AST es la misma que la que emitía el correspondiente analizador sintáctico en la fase 2. En particular, si hay error, se señala, y el proceso termina (no se realiza impresión alguna):

Entrada	Salida

a
{
## error: falta @ 	
x = 5
}
	CONSTRUCCION AST ASCENDENTE
	{
	x
	ERROR_SINTACTICO