uc3m

Procesadores del Lenguaje

Curso 2020-2021

Familiarización con el análisis sintáctico

1 Introducción

Esta segunda fase de laboratorio arranca con la familiarización de las herramientas que permiten representar gramáticas básicas para construir analizadores sintácticos, partiendo del ejemplo de la sesión anterior. Se partirá del ejemplo de la calculadora para hacer extensiones del lenguaje, utilizando las herramientas JFlex y CUP para obtener las modificaciones de los analizadores léxicos y sintácticos iniciales.

2 Especificación

En las sesiones anteriores hemos visto como implementar reconocedores léxicos de expresiones regulares para capturar los símbolos necesarios del lenguaje de una calculadora que procese expresiones matemáticas con números enteros y reales, con posibilidad de agrupar con paréntesis.

A continuación, se solicita modificar la especificación de los lenguajes utilizados en el escáner (JFlex) y parser (CUP) para realizer las siguientes tareas.

Tareas a realizar

Reconocedor Léxico

A: Aceptar e ignorar comentarios, que deben ser tratados –e ignorados– por completo en el análisis léxico. Los comentarios comenzarán con los caracteres '/*' y se extienden hasta la aparición de '*/', pudiéndose extender a varias líneas separadas. Un ejemplo sería:

B. Incorporar números reales (con notación científica habitual) y números hexadecimales (comienzan por la secuencia "0x" o "0X")

```
% Operaciones
5.3+.1+1e-1 <!-- esta expresion debe resultar 5.5 -->
5.3+0X11 <!-- esta expresion debe resultar 22.3 -->
```

Reconocedor Sintáctico

A. Incorporar nuevos símbolos de la gramática y las acciones para incorporar en la gramática funciones científicas y calcular el resultado: **exp**(), **log** (), **cos**(), **sin**()

B. Eliminar las reglas de precedencia de la gramática y ver el efecto en el analizador sintáctico. Modificar la gramática para evitar la necesidad de utilizar reglas de precedencia de la solución, incorporando los símbolos no terminales necesarios. Introducir el operador "-"de modificador de signo sin necesidad de utilizar precedencia.

C. Incorporación de un array de variables en memoria, de manera que los operadores participantes en las operaciones podrán ser los números literales y variables de memoria denominadas como "MEM[i]", siendo i un número entero del 0 al 99. Por tanto, existirán dos

tipos de sentencias: expresiones matemáticas y asignaciones. Las asignaciones comienzan por el nombre de variable, "MEM[i]", seguido del signo igual '=' y terminado con una expresión matemática normal. El valor por defecto de estas variables es cero, por lo que puede ser usadas en una expresión sin haber aparecido con anterioridad en la parte izquierda de una asignación.

D (opcional). Incorporar los operadores de preincremento (++MEM[i]), postincremento (MEM[i]++), predecremento (--MEM[i]), postdecremento (MEM[i]--) sobre las variables de memoria, con el mismo uso que en el lenguaje Java

A continuación se incluye código de muestra tomado de varios de los archivos de ejemplo proporcionados:

```
/* Operaciones */
5 * +3 - 80/10 /* esta expresion debe resultar 7 */
10 / 5 * 2 /* esta expresion debe resultar 4 */
- 3 *4 /* esta expresion debe resultar -12 */
5 * (4-3) / (8 - (4-1)) /* esta expresion debe resultar 1 */
MEM[1] = 5
MEM[2] = (-(7*MEM[1]+-6)*-1 /* MEM[2] = 41 */
(MEM[2]-MEM[1]/2 /* esta expresion debe resultar 18 */
MEM[0] = 1
MEM[1] = 1
MEM[2] = ++MEM[0]+1;
MEM[3] = MEM[1]+++1;
MEM[0] /* esta expresion debe resultar 2 */
MEM[1] /* esta expresion debe resultar 2 */
MEM[2] /* esta expresion debe resultar 3 */
MEM[3] /* esta expresion debe resultar 2 */
```

3 Entrega

Cada pareja debe entregar todo el contenido de su práctica en un único archivo comprimido –preferiblemente en formato ZIP–. El nombre del comprimido debe ser "pl_p2_grupo_XX", (pe-ele_pe-uno) donde XX son los primeros apellidos de los miembros del grupo.
Al abrir el archivo comprimido, debe verse un único directorio con el mismo nombre que el comprimido. Este directorio debe contener un proyecto eclipse completo y funcional, siguiendo la estructura del código de base. El código fuente del proyecto incluirá al menos los archivos con la especificación léxica y sintátctica del scanner y parser que satisfagan los requisitos del enunciado.
NOTAS:

- ***Se permite añadir nuevas clases Java si se considera necesario, y deberán incluirse en la entrega.***
- ***Todos los proyectos son compilados por el corrector, de forma que NO se deben realizar modificaciones a los ficheros generados por JFlex y CUP, puesto que al compilar de nuevo estos cambios se perderán.***

Se adjuntará una breve memoria con consideraciones para el corrector. El formato deber ser TXT, DOC o PDF.
Además, cada grupo debe entregar ficheros de prueba:
• "grupo_XX_pruebaOK.txt": contendrá una entrada compuesta por código bien formado, que producirá una salida correcta del programa.

Los dos criterios que primarán en la corrección son:
• Funcionalidad. Todo debe funcionar sin errores y devolver la salida esperada.
• Buen uso de JFlex/CUP, que demuestre que se domina la herramienta (¡conviene leer los manuales!).

Consideración final

Es muy importante respetar de forma estricta el formato de entrega. De no hacerlo, se corre el riesgo de perder puntos de calificación.
Cada grupo responderá de forma totalmente responsable del contenido de su práctica. Esto implica que los autores deben conocer en profundidad todo el material creado por ellos.
El corrector podrá convocar al grupo para responder a cuestiones sobre cualquier aspecto de la memoria o código entregados.