UNIVERSIDAD DE MÁLAGA

Procesadores de Lenguajes

PARCIAL 3 (2024)

Apartado 1. (2 punto) Implementar mediante JFlex y CUP un analizador sintáctico que permita evaluar expresiones aritméticas con números enteros. A continuación se muestra un ejemplo de programa en este lenguaje:

```
a = 5;
b = 6;
c = a + b;
a = c * b;
write( a );
write( b );
write( c );
```

El lenguaje tendrá las siguientes características:

- Las variables se asignarán con el operador =. No es necesario definir las variable previamente.
- Utilizar una variable a la que no se ha dado ningún valor tendrá valor 0.
- Los nombres de variables tienen que comenzar por una letra (mayúscula o minúscula) y seguir con cualquier combinación de letras, dígitos y carácter subrayado.
- Los operadores válidos son +, -, *, /, menos unario y paréntesis (con el mismo significado y preferencia que en el lenguaje C++).
- El lenguaje cuenta con el procedimiento `write`. Este procedimiento acepta un valor (variable, número o expresión). El método `write` mostrará este valor por pantalla seguido de un salto de línea en la pantalla.
- El programa se deberá poder ejecutar con uno o dos parámetros. El primer parámetro corresponderá al fichero con el programa a interpretar y el segundo parámetro que será opcional corresponderá al fichero donde se guardará la salida del programa. Si solo se ejecutase con un parámetro entonces mostraría la salida por consola. Por tanto, en Unix, cualquiera de las siguientes ejecuciones del programa serían válidas:

```
java -cp /usr/share/java/java-cup-0.11b-runtime.jar:. CalPro entrada.cal
java -cp /usr/share/java/java-cup-0.11b-runtime.jar:. CalPro entrada.cal salida.out
```

Apartado 2. (4 puntos) Añadir al lenguaje el operador ternario alternativa `if <expresión booleana> then <valor1> else <valor2>`. Que devuelve `valor1` si la `<expresión booleana>` es verdadera y devuelve el `valor2` si la `<expresión booleana>` es falsa (de forma similar al mismo operador ? de C++). Los operadores relacionales que implementarán las expresiones booleanas son (==, <, y >). Y los operadores lógicos que implementará el intérprete son (`not`, `and` y `or`). La precedencia de los operadores relacionales y lógicos será similar a la de C++. Por ejemplo, el siguiente código sería un ejemplo de uso del operador ternario:

```
a = 5;
b = 6;
c = if a > 3 then  b else a + 3;
d = if (a < 3) and b > 5 then 5  else if c > 6 then 4 else 5;
write( a );
```

```
write( b );
write( c );
write( d );
```

Apartado 3. (4 puntos) Añadir al lenguaje el procedimiento `print`. Este procedimiento acepta una lista de valores separados por comas. Cada valor puede ser una expresión numérica o una cadena. La función `print` mostrará cada uno de los valores de la lista por pantalla y terminará escribiendo un salto de línea en la pantalla. Las cadenas que aparecen como parámetros del procedimiento `print` no pueden ocupar más de una línea y no contendrán caracteres escapados (como, por ejemplo, \n). Por ejemplo, el siguiente código sería un ejemplo de uso del procedimiento `print`:

```
a = 5;
b = 7;
print( “Resultado: (“, a, a+b, b, “)” );
```

NOTAS:

- **NO ES OBLIGATORIO USAR LOS FICHEROS DE AYUDA.** Es decir, estos ficheros se proporcionan como ayuda para facilitar la implementación, pero no es imprescindible usarlos. También se pueden modificar si hace falta, como cada cual lo crea conveniente, a fin de cumplir con las especificaciones del enunciado.
- Como resultado de este ejercicio deben enviarse TODOS los ficheros necesarios para la compilación del compilador comprimidos en un fichero denominado **CalPro.zip**, (tenga cuidado de no incluir ficheros innecesarios que puedan dificultar la compilación automática). Para obtener el fichero comprimido puede usar la siguiente instrucción: `zip CalPro.zip CalPro.flex CalPro.cup CalPro.java`
- La corrección de este ejercicio se hace en función de varios casos de prueba independientes, similares, pero NO IGUALES, a los que se incluyen en el fichero comprimido. La correcta compilación de los casos de prueba que se dan en el fichero **CalPro-init.zip** es condición necesaria, pero no suficiente para superar las pruebas.