Práctica 4
Manejo avanzado de *Bison*

Elvira Mayordomo, Jorge Bernad, Mónica Hernández, José Manuel Colom

Tareas

1. Estudia la información sobre trazas en el *Capítulo 8 - Depurando Su Analizador* del manual de Bison (`bison-es-1.27.pdf`).
2. Realiza el ejercicio propuesto.
3. Elabora la memoria de la práctica y entrégala junto con los ficheros fuente según el Procedimiento de Entrega de Prácticas explicado en la Introducción a las Prácticas de la Asignatura. La fecha tope de entrega será hasta el 19 de enero de 2018 (incluido).

Nota: El incumplimiento de las normas de entrega se reflejará en la calificación de la práctica.

Se recuerda especialmente lo siguiente:

- Crea un directorio que contenga exclusivamente el fichero con la memoria en formato *PDF*, los ficheros fuente con tu código *.l* de *Flex*, los ficheros fuente con tu código *.y* de *Bison*, y los de prueba (*.txt* de texto). No usar subdirectorios.
- Accede al directorio con tus ficheros y ejecuta el comando

```
zip nipPr4.zip *.*
```

  donde *nip* es el identificador personal.
- En caso de que el fichero resultante tenga un tamaño mayor de 1024 KB deberás dividir el fichero `nipPr4.zip` en varios ficheros de tamaño 1024KB con el comando de linux

```
split -b 1m nipPr4.zip nipPr4.zip.
```

Introducción

El objetivo de esta cuarta práctica es implementar en *Flex* y *Bison* un parser para compilar ficheros de texto escritos en el lenguaje *CMMS* (C muy muy simple) que se describe a continuación. El resultado de la compilación será mostrar por pantalla una línea de texto con el mensaje "`Syntax error`", en caso de que el código no pertenezca al lenguaje *CMMS*, y si pertenece, no se mostrará nada.

El lenguaje *CMMS*

El lenguaje *CMMS* consiste en: cuatro palabras reservadas, `float`, `if`, `do`, `while`; cuatro operadores artiméticos, '+','−','∗', '/'; dos operadores lógicos, '&&', '||'; dos operadores de comparación '>', '=='; el operador de asignación '='; y los delimitadores ';' '(' ')' '{' '}'.

Un programa de *CMMS* consiste en una lista de cero o más instrucciones de los siguientes tipos:

- Instrucción tipo declaración: una instrucción de tipo declaración empezará por la palabra reservada `float`, a continuación aparecerá el nombre de un identificador de variable, el carácter '=' (asignación), y una constante numérica u otro identificador finalizado por ';'. Un identificador de variable será cualquier secuencia de caracteres que empieza por una letra mayúscula o minúscula seguida de letras mayúsculas o minúsculas y dígitos. Una constante numérica será cualquier secuencia de uno o más dígitos, o dos secuencias de dígitos separados por el caracter '.' donde al menos una de las dos secuencias debe tener un dígito o más. Ejemplos válidos: "`float` $v = 56.3$;" "`float` $v = v1$;" Ejemplos no válidos: "`float` v;" "`float` $2v = 3$;". Será necesario que los identificadores de variables que aparezcan a la derecha de la asignación de una instrucción de tipo declaración hayan sido declarados previamente en otra instrucción de este mismo tipo en su parte izquierda. Esto es, para que "'`float` $v = v1$;" sea correcta, es necesario que previamente aparezca una instrucción estilo "`float` $v1 = 6.2$;".

- Instrucción de operación: estas instrucciones se componen de un identificador de variable seguido del carácter '=' y de una lista con uno o más identificadores de variable/constantes numéricas separadas por operadores aritméticos finalizados por ';'. Ejemplos validos: "$v = a$;" "$v = a + 3 * 7$;" Ejemplos no válidos: "$v = (a + 7) * 3$;" "$2 = v$;" "`int` $v = a + 2$;" "$v = a$". Es necesario que los identificadores de variables que aparezca en este tipo de instrucciones hayan sido declaradas previamente en la parte izquierda de la asignación de una instrucción de tipo declaración. Supondremos que en todos nuestros programas se declararán como mucho 10 variables.

- Instrucción tipo if: este tipo de instrucciones empiezan por la palabra reservada `if` seguida de: el carácter '('; una condición; el carácter ')'; el carácter '{'; y una lista de cero o más instrucciones finalizadas por el carácter '}'. Una condición en la instrucción if es una lista de una o más condiciones simples separadas por uno de los operadores lógicos, siendo una condición simple dos identificadores de variable/constantes numéricas separadas por un operador de comparación. Ejemplos válidos: "`if` $(a > 3)\{\}$" "`if` $(a > 3 \&\& 3 > 1 || b == c)\{$ `float` $a = 3;\}$" Ejemplos no válidos "`if` $((a > 3)\&\&(b == c))\{\}$".

- Instrucción tipo do..while: se componen de: la palabra reservada `do` seguida del carácter '{'; una lista de cero o más instrucciones finalizadas por el carácter '}'; la palabra reservada `while`; un paréntesis '(' y una condición finalizada por ')'.

Pueden aparecer **espacios en blanco y saltos de línea** en cualquier lugar del código, excepto dentro de una palabra reservada, nombre de variable o constante numérica.

También se admiten **comentarios** entre '/*' y '*/' que empiecen y terminen en cualquier lugar del código, excepto dentro de una palabra reservada, nombre de variable o constante numérica. Los comentarios pueden terminar con '*/' o con el final de fichero.

Si aparece cualquier otro símbolo no especificado en este enunciado fuera de un comentario, el resultado será "Syntax error".

Tarea a realizar

Construye un fuente flex *comp.l* y un fuente bison *comp.y* cuya compilación produzca un ejecutable que realice las siguientes acciones para un fichero de entrada dado: mostrar por pantalla una línea de texto con el mensaje "Syntax error", en caso de que el código no pertenezca al lenguaje *CMMS*, y si pertenece, no se mostrará nada.

Nota:

- Para hacer trazas de la ejecución de *Bison* se pueden utilizar acciones dentro de las reglas, con la misma estructura que ya se explicó para *Flex*. Aunque estas acciones sirven para muchas más cosas, se pueden utilizara para localizar problemas en la gramática. Por ejemplo:
  $S: \ ELEMENTOS\ FIN\ \{printf(\text{"}Regla\ S: ELEMENTOS\ FIN\backslash n\text{"});\}$
  Esto escribirá por pantalla una traza del análisis realizado por la gramática. Observad que la acción está al final de la regla. También se pueden poner acciones entre los tokens de una regla, por ejemplo:
  $S: \ \{printf(\text{"}Antes\ S: ELEMENTOS\ FIN\backslash n\text{"});\}\ ELEMENTOS\ FIN$
  pero en este caso, Bison generará automáticamente un conjunto de reglas nuevo que puede llevar a que aparezcan conflictos.

- En esta práctica se valorarán especialmente las pruebas realizadas, debiéndose entregar los ficheros correspondientes a las mismas.