Práctica 3
Introducción al Manejo de *Bison*

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

Tareas

1. Estudia el capítulo 1 del libro *flex & bison* (páginas 5 a 15).
2. Lee la introducción de esta práctica y realiza los ejercicios propuestos.
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 día anterior al comienzo de la Práctica 4.

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

Se recuerda especialmente lo siguiente:

- Verifica que todos tus ficheros fuente (*.l*) contienen como comentario en sus primeras líneas el NIA y el nombre del autor. Todos los programas deberán estar debidamente documentados.
- Verifica que los ficheros que vas a presentar compilan y ejecutan correctamente en hendrix.
- Crea un fichero comprimido .zip llamado

  `niaPr3.zip`

  donde *nia* es el identificador personal. Este fichero comprimido **debe contener exclusivamente** el fichero con la memoria en formato *PDF*, los ficheros fuente con tu código (*.l* de *Flex*), y los de prueba (*.txt* de texto). No usar subdirectorios.
- Utiliza el enlace a una tarea de moodle disponible en la sección "Prácticas de Laboratorio" para entregar el fichero `niaPr3.zip`

Introducción

El objetivo principal de esta práctica es familiarizarse con la herramienta de creación de analizadores sintácticos *Bison* y entender cómo utilizarla en conjunción con *Flex*. Para ello se propone un ejercicio guiado para realizar una calculadora sencilla de números positivos (basada en los ejemplos recogidos en el capítulo 1 del libro *flex & bison*) y la realización de variantes de dicha calculadora además de otros reconocedores sencillos.

Antes de realizar el ejercicio guiado debes ser capaz de responder a las siguientes preguntas tras la lectura del capítulo 1 del libro *flex & bison*:

- ¿Qué es un token?
- ¿Qué son $$, $1 y $2? ¿Qué es `yylval`?
- ¿Qué es el "start symbol"? ¿Cómo se define en bison?
- ¿Cómo escribirías en bison la siguiente gramática?:
  $S \rightarrow \ aS \mid T$
  $T \rightarrow \ bTb \mid \epsilon$

Ejercicio 1

Observa el contenido de los ficheros *calcOrig.l* y *calcOrig.y* que encontrarás al final de este pdf y que puedes descargar de moodle.

Indica a qué gramática corresponde *calcOrig.y*

Compila dichos fuentes ejecutando los siguientes comandos:

```
bison -yd calcOrig.y
flex calcOrig.l
gcc lex.yy.c y.tab.c -lfl -o calcOrig
```

¿Qué mensajes has recibido de bison? El objetivo de estos fuentes es implementar una calculadora sencilla de enteros positivos, con las operaciones de suma, resta, multiplicación y división con la precedencia habitual.

Prueba esta calculadora con distintas expresiones aritméticas, ¿para cuáles da error?

Observa el código alternativo en *calcMejor.l* y *calcMejor.y*. ¿Qué diferencias observas? ¿Cómo funciona en los casos que fallaban antes? ¿Por qué?

Ejercicio 2

El objetivo de este ejercicio es realizar una serie de mejoras sobre la calculadora de enteros positivos con fuentes *calcMejor.l* y *calcMejor.y*. Cada mejora se realizará sobre *calcMejor*, no siendo acumulativas.

El nombre de los fuentes resultantes debe ser `ej21.l ej21.y ej22.l ej22.y` etc.

1. Modificar la calculadora para que acepte enteros en decimal o en binario. Específicamente, una cadena que sólo tenga 0's y 1's y termine en la letra "b" (sin espacios en blanco en medio) debe ser interpretada como un entero en binario. Por ejemplo, al ejecutar la calculadora debe ocurrir lo siguiente:

   `hendrix: ./ej21`

   10 + 3

   =13

   10b + 3

   =5

   Para esto sólo deberías modificar el scanner flex.

2. Modificar la calculadora para que todas las líneas de entrada terminen con ";" o bien ";b". Si la línea termina en ";b" el resultado se debe escribir en binario, si termina en ";" se escribe en decimal como antes.

   `hendrix: ./ej22`

   10 + 3;

   =13

   10 + 3;b

   =1101

3. Modificar la calculadora para que permita una única variable acumulador. Para ello necesitamos permitir asignaciones a esta variable y referencias a la misma. La variable se llamará `acum` y las asignaciones serán de la forma "`acum`:= expresión", por ejemplo la calculadora funciona como sigue

   `hendrix: ./ej23`

   acum:= 5

   acum

   =5

   acum:= 3*2

acum

=6

acum+4

=10

acum:=8+acum

acum

=14

Ejercicio 3

Implementa un analizador sintáctico para la siguiente gramática:

$$\boxed{\begin{array}{l} S \to CxS \mid \epsilon \\ B \to xCy \mid xC \\ C \to xBx \mid z \end{array}}$$

Para las entradas válidas, el programa no ha de mostrar nada por pantalla y para las que no pertenezcan al lenguaje debe mostrar por pantalla ***"syntax error"***.

¿Qué lenguaje reconoce?

```
/* calcOrig.l fichero para la practica 3 de Teoria de la Computacion */
%{
#include "y.tab.h"
%}
%%
"+"      {return(ADD);}
"-"      {return(SUB);}
"*"      {return(MUL);}
"/"      {return(DIV);}
"("      {return(OP);}
")"      {return(CP);}
[0-9]+   {yylval = atoi(yytext); return(NUMBER);}
\n       {return(EOL);}
[ \t]    {/* ignorar espacios */}
.        {return(yytext[0]); /* caracter inesperado */}
%%
```

```
/* calcOrig.y fichero para la practica 3 de Teoria de la Computacion */
%{
#include <stdio.h>
%}
%token NUMBER ADD SUB MUL DIV EOL CP OP
%start calclist
%%

calclist : /* nada */
        | calclist exp EOL { printf("=%d\n", $2); }
        ;
exp :   factor
        | exp ADD factor { $$ = $1 + $3; }
        | exp SUB factor { $$ = $1 - $3; }
        ;
factor :        term
                | exp MUL factor { $$ = $1 * $3; }
                | exp DIV factor { $$ = $1 / $3; }
                ;

term :  NUMBER { printf("number=%d\n", $1); }
        | OP exp CP { $$ = $2; }
        ;
%%
int yyerror(char* s) {
   printf("\n%s\n", s);
   return 0;
}
int main() {
  yyparse();
}
```

```
/* calcMejor.l fichero para la practica 3 de Teoria de la Computacion */
%{
#include "y.tab.h"
%}
%%
"+"      {return(ADD);}
"-"      {return(SUB);}
"*"      {return(MUL);}
"/"      {return(DIV);}
"("      {return(OP);}
")"      {return(CP);}
[0-9]+   {yylval = atoi(yytext); return(NUMBER);}
\n       {return(EOL);}
[ \t]    {/* ignorar espacios */}
.        {return(yytext[0]); /* caracter inesperado */}
%%
```

```
/* calcMejor.y fichero para la practica 3 de Teoria de la Computacion
*/
%{
#include <stdio.h>
%}
%token NUMBER EOL CP OP
%start calclist
%token ADD SUB
%token MUL DIV
%%

calclist : /* nada */
        | calclist exp EOL { printf("=%d\n", $2); }
        ;
exp :   factor
        | exp ADD factor { $$ = $1 + $3; }
        | exp SUB factor { $$ = $1 - $3; }
        ;
factor :        factor MUL factorsimple { $$ = $1 * $3; }
                | factor DIV factorsimple { $$ = $1 / $3; }
                | factorsimple
                ;
factorsimple :  OP exp CP { $$ = $2; }
                | NUMBER
                ;
%%
int yyerror(char* s) {
   printf("\n%s\n", s);
   return 0;
}
int main() {
  yyparse();
}
```