**Grupo Miércoles 17:00-19:00 semanas A**
**– Práctica 3 –**
**Autor:** Víctor Marteles Martínez
**NIP:928927**
**Autor:** Javier Martínez Virto
**NIP:930853**

## Ejercicio 1:

1. Resumen
El archivo “calcOrig.y” se corresponde a una gramática que describe expresiones aritméticas básicas.

Al compilar los fuentes, recibimos 16 “warnings” reduce/reduce. A pesar de ello, podemos compilar. Para operaciones muy simples la calculadora funciona bien, sin embargo, si avanzamos en nuestras pruebas observamos que la calculadora la prioridad de las operaciones va de derecha a izquierda en vez de izquierda a derecha, como debería ser. Por ejemplo, la operación “16/8*2” debería ser 4, ya que 16/8=2 y 2*2=4; pero, la calculadora devuelve 1, ya que 8*2=16 y 16/16=1. Además, se produce un error de sintaxis (“syntax error”) al intentar realizar sumas o restas con más de dos operandos de manera consecutiva.

Para solucionar estos fallos en las operaciones, en los archivos “calcMejor” se modifica la producción del “factor” en la calculadora (utilizando una regla “factorsimple” en lugar de “term”), de modo que admita operaciones recursivas.

2. Pruebas
hendrix: ./calcOrig

| ENTRADA | SALIDA |
|---|---|
| 8+8 | number=8<br>number=8<br>=16 |
| 5*3 | number=5<br>number=3<br>=15 |
| 15/3 | number=15<br>number=3<br>=5 |
| 16/8*2 | number=16<br>number=8<br>number=2<br>=1 |
| 81/9/9 | number=81<br>number=9<br>number=9<br>=81 |
| 7+7+7 | |

| | |
|---|---|
| | number=7<br>number=7<br>number=7<br><br>syntax error |

**Ejercicio 2:**

**2.1**

1. Resumen

Partiendo de “calcMejor.l”, en “ej21.l” se añade el token “BASE” para poder interpretar “b=”. Además, si se consume “ [0-9]+"b" ”, el número seguido de la “b” se interpretará en la base anteriormente establecida (o si no ha establecido ninguna, en base 10). Si un número no va seguido de una “b”, se interpretará en base 10, base predeterminada.

En “ej21.y”, se declaran los tokens nuevos y se pone la regla de que la base debe ser un número del 2 al 10.

2. Pruebas

hendrix: ./ej21

| Entrada: | Salida |
|---|---|
| b=2<br>10+3<br><br>10b+3<br><br>(8/2+16)*5<br><br>b=2<br>(8/2+16)*5<br><br>(1000b/10b+16)*101b<br><br>b=6<br>(8/2+24b)*5 | <br><br>=13<br><br>=5<br><br>=100<br><br>=100<br><br>=100<br><br>=100 |

**2.2**

1. Resumen.

Partiendo de “calcMejor.l”, en “ej22.l” se añade el token “BASE” para poder interpretar “b=” (como en “ej21.l”) y los tokens “PCOMA” y “PCOMAB” para interpretar “;” y “;b”.

En “ej22.l”, se añaden los nuevos tokens y nuevas reglas y modificaciones a “calclist”. Ahora, para la salida en decimal se debe poner un “;” al final de la línea (calclist exp PCOMA EOL). Asimismo, se establece la regla de que la base debe ser un número del 2 al 10. Por último, se añade que si en vez de en “;”, la línea acaba en “;b” (calclist exp PCOMAB EOL), el resultado de la operación debe aparecer siguiendo la base establecida anteriormente por “b=”.

2. Pruebas

hendrix: ./ej22

| Entrada: | Salida |
|---|---|
| 10+3;<br><br>b=2<br>10+3;b<br><br>(86+4-8)/(22-20);<br><br>b=2<br>(86+4-8)/(22-20);b<br><br>(86+4-8)/(22-20);<br><br>b=9<br>(86+4-8)/(22-20);b<br><br>b=5<br>(86+4-8)/(22-20);b | =13<br><br>=1101<br><br>=41<br><br>=101001<br><br>=41<br><br>=45<br><br>=131 |

## 2.3

1. Resumen.

Partiendo de “calcMejor.l”, en “ej23.l”, si se consume un número en hexadecimal seguido de una “H” (0-9A-F]+"H") se pasará a base 10 y se devolverá como el token “NUMBER” para poder operar con los demás números de la expresión.

En “ej23.y” no hay ningún cambio ya que la conversión de hexadecimal a decimal es realizada por Flex.

2. Pruebas

hendrix: ./ej23

| Entrada: | Salida |
|---|---|

| | |
|---|---|
| 10+3<br>A0H+3<br>78/39+777/111<br>4EH/39+309H/111<br>78/27H+777/6FH<br>4EH/27H+309H/6FH | <br>=13<br>=163<br>=9<br>=9<br>=9<br>=9 |

## 2.4

1. Resumen.
Partiendo de “calcMejor.l”, en “ej24.l”, se añaden los tokens “PCOMA” Y “PCOMAH” para poder interpretar “;” y “;H”.

En “ej24.y”, se añaden los nuevos tokens y nuevas reglas y modificaciones a “calclist”. Ahora, para la salida en decimal se debe poner un “;” al final de la línea (calclist exp PCOMA EOL). Además, si la línea acaba con un “PCOMAH” (calclist exp PCOMAH EOL), se imprimirá el resultado en hexadecimal gracias a “ { printf("=%X\n", $2); } “.

2. Pruebas
hendrix: ./ej24

| Entrada: | Salida |
|---|---|
| 10+3;<br>10+3;H<br>(978-102)/2;<br>(978-102)/2;H<br>54*12/2;<br>54*12/2;H<br>(5+15)/(12/6);<br>(5+15)/(12/6);H | <br>=13<br>=D<br>=438<br>=1B6<br>=324<br>=144<br>=10<br>=A |

**Ejercicio 3:**

1. Resumen

Nos piden un analizador sintáctico del lenguaje :

$$S \rightarrow CxS \mid \varepsilon$$
$$B \rightarrow xCyzy \mid xC$$
$$C \rightarrow xBx \mid z$$

Para el que añadimos en byson los TOKENs: ‘X’, ‘Y’, ‘Z’ y ‘EOL’ para detectar las x, y, z y los \n respectivamente. Además, se añade el estado inicial ‘s’, correspondiente al ‘S’ del lenguaje del enunciado. Todo lo que no sean esos caracteres, menos los espacios que son ignorados, se pasan a bison como caracter inesperado para que por terminal se escriba syntax error, para posteriormente finalizar el programa.

2. Pruebas

| ENTRADA | SALIDA |
|---|---|
| zx | |
| zxzx | |
| xxzyzyxx | |
| xxzxx | |
| xxxxzyzyxyzyxx | |
| xx | |
| | syntax error |
| xxyxx | |
| | syntax error |
| xzxz | |
| | syntax error |