



PROCESADORES DE LENGUAJE

Ingeniería Informática
Especialidad de Computación
Tercer curso
Segundo cuatrimestre



Departamento de Informática y Análisis Numérico
Escuela Politécnica Superior de Córdoba
Universidad de Córdoba

Curso académico 2015 - 2016

Hoja de ejercicios de FLEX

1. Sumar

- Codifica un programa en flex que copie el archivo de entrada en uno de salida, sumando 5 a todo número positivo que sea múltiplo de 4.
- Ejemplo

Entrada.txt

```
1 2 3 4
5 6 7 8
9 10 11 12
```

Salida.txt

```
1 2 3 9
5 6 7 13
9 10 11 17
```

- Observación:
 - Se recomienda usar la función `atoi()` de C que transforma una cadena de caracteres en su valor entero.

2. Sumar parametrizado

- Codifica un programa en flex que copie el archivo de entrada en uno de salida, sumando N1 a todo número positivo que sea múltiplo de N2, donde N1 y N2 son dos números pasados como argumentos desde la línea de órdenes.
- Ejemplo: `./sumar_parametrizado.exe Entrada.txt Salida.txt 5 4`

Entrada.txt

```
1 2 3 4
5 6 7 8
```

9 10 11 12

Salida.txt

1	2	3	9
5	6	7	13
9	10	11	17

- Observación:
 - Se recomienda usar la función `atoi()` de C que transforma una cadena de caracteres en su valor entero.

3. Contar

- Elaborar un programa de flex que reciba un fichero de texto y cuente el número de caracteres, palabras y líneas que contiene.

4. Contar apariciones de una palabra

- Codifica un programa flex que reciba un fichero de texto y una palabra y cuente el número veces que aparece dicha palabra en el fichero.

5. Sustitución de una palabra

- Codifica un analizador que reemplace una palabra por otra en un fichero de entrada.
- Ambas palabras, así como el nombre del fichero deberán ser introducidos por el usuario, bien a través de la línea de comandos o cuando el usuario ejecute el programa.

6. Ficheros impresos

- Considérese un fichero de registro que contiene información sobre los ficheros que se han imprimido en una impresora.
- Cada línea tiene la siguiente información
 - usuario fichero fecha hora
- Codifica un analizador léxico que permita
 - Contar cuántos trabajos ha imprimido un usuario
 - El programa recibirá como argumentos
 - el nombre del fichero de registro
 - el nombre del usuario
 - Contar cuántos trabajos ha imprimido un usuario cada día
 - el nombre del fichero de registro
 - el nombre del usuario
 - el día que se desea consultar.

7. Analizador léxico de pseudocódigo

- Codifica un analizador léxico que permita reconocer los componentes léxicos de un programa escrito en pseudocódigo.

- **Palabras reservadas**
 - *inicio, fin, __mod, __o, __y, __no, leer, escribir, si, entonces, si_no, fin_si, mientras, hacer, fin_mientras, repetir, hasta_que, para, desde, hasta, paso, fin_para.*
 - No se distinguirá entre mayúsculas ni minúsculas.
 - Las palabras reservadas no se podrán utilizar como identificadores.

- **IDENTIFICADOR**
 - Características
 - Estarán compuestos por una serie de letras, dígitos y el subrayado;
 - Deben comenzar por una letra,
 - No podrán acabar con el símbolo de subrayado, ni tener dos subrayados consecutivos.
 - No se distinguirá entre mayúsculas ni minúsculas.
 - Ejemplos
 - Identificadores válidos:
dato, dato_1, dato_1_a
 - Identificadores **no** válidos:
dato, dato, dato__1

- **NÚMERO**
 - Se utilizarán números enteros, reales de punto fijo y reales con notación científica.
 - Todos ellos serán tratados conjuntamente como números.

- **CADENA**
 - Estará compuesta por una serie de caracteres delimitados por comillas simples:
'Ejemplo de cadena'
 - Deberá permitir la inclusión de la comilla simple utilizando la barra (\):
'Ejemplo de cadena con \' comillas\' simples'.
 - **Nota:**
 - Las comillas exteriores no formarán parte de la cadena.

- **Operador de asignación**
 - ASIGNACIÓN: :=

- **Operadores aritméticos:**
 - SUMA: +
 - RESTA: -
 - PRODUCTO: *
 - DIVISIÓN: /

- MÓDULO: `__mod`
- POTENCIA: `**`
- **Operador alfanumérico:**
 - CONCATENACIÓN: `||`
- **Operadores relacionales de números y cadenas:**
 - MENOR_QUE: `<`
 - MENOR_IGUAL_QUE: `<=`
 - MAYOR_QUE: `>`
 - MAYOR_IGUAL_QUE: `>=`
 - IGUAL: `==`
 - DISTINTO: `<>`
 - Por ejemplo:
 - Si *A* es una variable numérica y *control* una variable alfanumérica, se pueden generar las siguientes expresiones relacionales:
`(A >= 0)`
`(control <> 'stop')`
- **Operadores lógicos:**
 - DISYUNCIÓN_LÓGICA: `__o`
 - CONJUNCIÓN_LÓGICA: `__y`
 - NEGACIÓN_LÓGICA: `__no`
 - Por ejemplo:
`(A >= 0) __y __no (control <> 'stop')`
- **Comentarios**
 - De varias líneas: delimitados por (`*` y `*`)

`(* ejemplo maravilloso
de comentario
de tres líneas *)`
 - De una línea:
 - Todo lo que siga al carácter `#` hasta el final de la línea.

`# ejemplo espectacular de comentario de una línea`
- **Otro componentes léxicos**
 - FIN_SENTENCIA: `;`
 - Paréntesis
 - Izquierdo: `(`
 - Derecho: `)`

- **Control de errores**
 - El intérprete deberá controlar toda clase de errores:
 - Identificador mal escrito.
 - Números mal escritos.
 - Utilización de símbolos no permitidos.
 - Etc.

- **Prueba**
 - Se deberá comprobar el funcionamiento del analizador léxico usando tres ficheros:
 - Fichero denominado Newton.txt
 - ejemplo_1.txt: fichero original **sin** errores.
 - ejemplo_2.txt: fichero original **con** errores.