Algoritmos y Programación II (75.41) Cátedra Lic. Gustavo Carolo 1° Cuatrimestre 2005	***Trabajo Práctico Nº 2*** ***Listas, pilas y colas***

Fecha de presentación: Martes: Miércoles:

Breve introducción teórica sobre Filtros y tubos

Los filtros y tubos (pipes and filters, en inglés) son una forma de dividir un problema en tareas más simples que se ejecutan de manera sequencial. Esta solución consiste en dos tipos de componentes: filtros y tubos. Los filtros realizan el procesamiento y los tubos realizan la comunicación entre ellos. El resultado va a ser la composición del procesamiento de cada uno de los filtros.

Los filtros son independiente entre sí y no conocen su posición en la cadena. Cada filtro realiza una sola tarea simple. Con la composición de varios filtros puede resolverse un problema complejo.

Ejemplo

Problema: un programa que lea un archivo y devuelva otro con la 10 palabras que ocurrieron más veces.
Solución: descomponer el programa en los siguientes filtros:

- La entrada lee el archivo y pone en el tubo las palabras que va encontrando.
- Filtro 1: ordena las palabras que recibe.
- Filtro 2: cuenta las palabras iguales consecutivas y emite <cantidad de ocurrencias con ceros adelante>;<palabra>. Ej: 0023;perro – 0015;gato ...
- Filtro 3: igual que el 1. (tener en cuenta que en este caso ordena por cantidad de ocurrencias).
- Filtro 4: separa el string que recibe según un separador (";") y devuelve el segundo campo.
- Filtro 5: deja pasar las primeras 10 palabras, el resto las ignora.
- Salida: escribe las palabras que recibe en un archivo.

Cada filtro realiza una tarea simple. Además, muchos de estos filtros pueden reutilizarse y reordenarse para lograr otros objetivos.
Para pensar: ¿que cambiaría para hacer que el programa devuelva las 10 palabras que aparecen MENOS veces?

Enunciado del TP:

1) Implementar un TDA "TEJECUTOR_T_F" (T: tubos / F: filtros) que permita la ejecución de un sistema de filtros y tubos. El TDA será independiente de los filtros que vaya a ejecutar. Además se encargará de generar la entrada en el primer tubo, leyendo un archivo de texto de entrada y metiendo las **palabras** que vaya encontrando una a una en el primer tubo. También se encargará de capturar las palabras a la salida del último tubo e imprimirlas en el archivo de salida.

Primitivas:

```
// Los tubos se implementan con colas de strings
typedef TCola TTubo;

// Prototipo de función que deben implementar los filtros
/* Pre: entrada y salida creados.
        datos_filtro apunta a una estructura conocida por la función (puede ser
nulo)
        eof indica si hay fin del archivo, es decir que las palabras que están en
el tubo de entrada son todas las que quedan.
   Post: Ejecuta el procesamiento.
        Devuelve TRUE si eof == TRUE y procesó todas las palabras de la entrada o
si es seguro que no va a emitir más palabras, sino devuelve FALSE.
        Si pudo generar un resultado, lo emite en el tubo de salida y devuelve el
control al ejecutor. La función de filtro devuelve a lo sumo un resultado en cada
ejecución.
*/
typedef int TProcesarFn( void * datos_filtro, TTubo entrada, TTubo salida, int
eof );

typedef struct {
      TProcesarFn * fnProceso;
      void * datos_filtro;
} TFilter;

/**
 * Crea el ejecutor.
 * @pre ejecutor no creado
 * @pre archivo_entrada abierto para lectura.
 * @pre archivo_salida abierto para escritura.
 * @pre filtros es una lista creada y no vacía de TFilter.
 * @pos ejecutor creado.
 */
void PR_Crear( TEjecutor_T_F * ejecutor, FILE * archivo_entrada, TLista filtros,
               FILE * archivo_salida );
/**
 * Ejecuta la cadena de tubos y filtros y genera la salida.
 * @pre ejecutor creado
 * @pos Se generó la salida y no puede volver a ejecutarse.
 */
void PR_Ejecutar( TEjecutor_T_F * ejecutor );

/**
 * Destruye el procesador.
 * @pre ejecutor creado
 * @pos ejecutor destruido.
 */
void PR_Destruir( TProcesador * procesador );
```

2) Implementar los siguientes filtros, cumpliendo la interfaz requerida por el ejecutor.

- Pasar a mayúsculas
- Pasar a minúsculas
- Ordenar
- Eliminar repeticiones consecutivas

- Filtrar determinadas palabras
- Pasar solo los primeros N
- Pasar solo los últimos N
- Contar consecutivos (emite 000X;palabra)
- Extraer n-esimo campo.

3) Implementar un programa que reciba un archivo de entrada, uno de salida y otro de definición de los filtros a utilizar.

```
ejecutar_t_f <entrada> <salida> <definicion>
```

El archivo de definición indicará los filtros a utilizar y en qué order y tendrá el siguiente formato:
FILTRO1 <parámetros>
FILTRO2 <parámetros>
FILTRO3 <parámetros>
...

Ej:
```
MAYUSCULAS
FILTRAR a,la,el,y,de,que
ORDENAR ASC
CONTAR_CONSEC
PRIMEROS 10
```

Los filtros y sus parámetros:

Filtro	Clave	Parámetros	Observaciones
Ordenar	ORDENAR	ASC ó DESC	
Pasar a mayúsculas	MAYUSCULAS	No tiene	
Filtrar determinadas palabras	FILTRAR	palabra1,palabra2,palabra3,...,palabraN	
Pasar los primeros N	PRIMEROS	cantidad	
Pasar los últimos N	ULTIMOS	cantidad	
Contar consecutivas	CONTAR_CONSEC	No tiene	
Extraer n-esimo campo	EXTRAER	<número de campo>	El primer campo es el 1

Aclaración: Se consideran palabras a sucesiones de caracteres desde a-z, A-Z y 0-9. El resto de los caracteres son ignorados y solo sirven para separar palabras.

Aclaración 2: Todos los filtros reciben y emiten strings.

Aclaración 3: Puede considerarse que las palabras no tienen más de 64 caracteres.