Algoritmos y Programación II (75.41)

Cátedra Lic. Gustavo Carolo

2° Cuatrimestre 2015

Trabajo Práctico Nº 2: Listas, Pilas y Colas

Fecha de presentación

Fecha de pre-entrega: semana del **21** de **septiembre de 2015**.
Fecha de entrega : semana del **5** de **octubre de 2015**.

El mismo deberá cumplir con todos los requisitos detallados en el reglamento de la cátedra.

Enunciado del TP: Documentador de código

Un documentador, como su nombre lo indica, es un módulo o programa que se encarga de tomar cierta información de uno o más archivos de entrada, los cuales deben contener secuencias específicas de caracteres, que son leídas y convertidas en un nuevo archivo de salida con el proceso de la información leída. De esta forma, si por ejemplo tenemos un archivo fuente de un programa y usamos una secuencia específica que esté contenida dentro de los bloques de comentarios, podríamos guardar información para que ésta sea considerada “documentación”.

Suponga el siguiente caso de ejemplo:

Tenemos un archivo fuente de un programa en lenguaje C, por ejemplo: main.c, con el siguiente contenido:

```
# include <stdio.h>
# include <stdlib.h>

/*
@funcion main
@descr Esta es la funcion principal de mi programa. El trabajo de este
       programa es hacer las cosas mas sencillas para la gente.

       Es importante remarcar que este programa imprime su nombre por pantalla

@autor Diego
@fecha 11/09/2015
@version "1.0"
@param argc parametro de cantidad de argumentos
@param argv parametro de argumentos de invocacion
@return int devuelve el codigo de error
@pre no existen condiciones previas esperadas
@pos el programa retornara 0 si la ejecución fue exitosa o el codigo de error
    correspondiente en otro caso
*/
int main (int argc, char** argv) {
   char nombre[]="documentador";
   printf("El nombre del programa es: %s", nombre);
   return 0;
}
```

Como se puede apreciar en el ejemplo anterior, tenemos varias palabras que no incluímos en el código del programa. En el bloque de comentarios, podemos ver una estructura de comentarios.

```
/*
@funcion main
@descr Esta es la funcion principal de mi programa. El trabajo de este
       programa es hacer las cosas mas sencillas para la gente.

       Es importante remarcar que este programa imprime su nombre por pantalla

@autor Diego
@fecha 11/09/2015
@version "1.0"
@param argc parametro de cantidad de argumentos
@param argv parametro de argumentos de invocacion
@return int devuelve el codigo de error
@pre no existen condiciones previas esperadas
@pos el programa retornara 0 si la ejecución fue exitosa o el codigo de error
    correspondiente en otro caso
*/
```

Esta estructura, contiene líneas formadas de la siguiente forma:
<palabra_reservada> <valor0> <valor1>...<valorn>
<palabra reservada>: es un conjunto de palabras (SET) que utiliza el documentador para interpretar lo que viene a continuación.
<valor0> <valor1>...<valorn>: Son los valores que debe tomar para esa entrada el documentador.

De esta forma, la línea:

```
@funcion main
```

debe poder ser interpretada como:
tipo: *funcion*
valor0: *main*

```
@descr Esta es la función principal de mi programa. El trabajo de este
       programa es hacer las cosas más sencillas para la gente.

       Es importante remarcar que este programa imprime su nombre por pantalla
```

debe poder ser interpretada como:
tipo: *descr*
valor0: *Esta es la función principal de mi programa. El trabajo de este programa es hacer las cosas más sencillas para la gente. Es importante remarcar que este programa imprime su nombre por pantalla.*

Como podemos ver para este ejemplo solo tuvimos la entrada, y una "especificación" de palabras reservadas. Con ello hemos interpretado un archivo, pero hasta ahora no hemos hecho nada más.
Un documentador toma la información y genera una salida en un formato de archivo previamente definido. De esta forma, toma cada línea interpretada y le aplica la transformación necesaria para cumplir con el formato de salida especificado.

Se pide:

1. Implementar un TDA "Documentador" que contenga en su estructura un **analizador lexicográfico y un logger**, cumpliendo con la interfaz que se describe a continuación:

Diseño del tipo de dato del TDA

La implementación del tipo de dato del TDA queda a criterio del grupo, sujeta a la validación por parte del corrector.

Primitivas del TDA

Deben desarrollarse las siguientes primitivas para el TDA, respetando exactamente la interfaz y la funcionalidad indicadas.

DocCrear

```
int DocCrear (TDA_Doc* tda, Logger log);
```

Crea el documentador inicializando su estructura con los datos necesarios.

Precondiciones

Documentador no creado

Postcondiciones

Devuelve RES_OK si pudo crear el documentador, y lo registra en el log en modo INFO. Devuelve RES_ERROR en otro caso, o la constante que se defina pertinentemente y lo registra en el log en modo ERROR.

Parametros

- **tda** el Documentador a crear
- **log** el logger que se usará

DocExtraerDocumentacion

```
int DocExtraerDocumentacion (TDA_Doc* tda, char* arch_entrada, char* arch_salida);
```

Extrae la documentación del archivo de entrada, concatenándola al archivo de salida. No sobreescribe el archivo de salida.

Precondiciones

Documentador creado e inicializado

Postcondiciones

Devuelve RES_OK si pudo extraer la información a documentar del archivo_entrada, escribiéndola en el archivo_salida y lo registra en el log.
Devuelve RES_ERROR en otro caso, o la constante que se defina pertinentemente y lo registra en el log en modo ERROR.

Parametros

- **tda** el Documentador a utilizar para extrae la información que se debe documentar
- **arch_entrada** ruta al archivo de entrada del cual se desea extraer la documentación.
- **arch_salida** ruta al archivo de salida, en el cual se desea escribir la documentación obtenida del archivo de entrada, según el formato especificado.

DocConstruirIndice

```
int DocConstruirIndice (TDA_Doc* tda, char* arch_indice);
```

Emite el índice elaborado a un archivo de salida. El índice debe contener un listado ordenado alfabéticamente que indique claramente en qué archivo se encuentra.

Precondiciones

Documentador creado e inicializado.

Postcondiciones

El archivo de salida “arch_indice” contiene el índice construido en formato html. Devuelve RES_OK en caso de que todo haya funcionado correctamente, o RES_ERROR, o la constante que se defina pertinentemente según sea necesario para el caso.

Parámetros

- **tda** el Documentador
- **arch_indice** la ruta al archivo de salida donde deberá guardarse el índice construido

DocDestruir

```
int DocExtraerDocumentacion (TDA_Doc* tda);
```

Destruye el documentador vaciando previamente su estructura y liberando todos los recursos en uso.

Precondiciones

Documentador creado e inicializado

Postcondiciones

Devuelve RES_OK si pudo destruir.
Devuelve RES_ERROR en otro caso, o la constante que se defina pertinentemente y lo registra en el log en modo ERROR.
En ambos casos registra la actividad en el log según corresponda

Parametros

- **tda** el Documentador

Formato de Salida para la documentación generada: HTML

Tags html a utilizar:

Keyword	Tag
titulo	[valor]
subtitulo	[valor]
funcion	Función: [valor]
descr	<dl> <dt>Descripción</dt> <dd>[valor]</dd> </dl>
author	<dl> <dt>Autor</dt> <dd>[valor]</dd> </dl>
fecha	<dl> <dt>Fecha</dt>

	<dd>[valor]</dd> </dl>
version	<dl> <dt>Version</dt> <dd>[valor]</dd> </dl>
param	<dl> <dt>Parámetro: [nombre]</dt> <dd>[valor]</dd> </dl>
return	<dl> <dt>Retorno</dt> <dd>[valor]</dd> </dl>
pre	<dl> <dt>Pre-Condición</dt> <dd>[valor]</dd> </dl>
pos	<dl> <dt>Post-Condición</dt> <dd>[valor]</dd> </dl>

Cada porción de documentación generada deberá incluirse dentro de un tag de html demarcando donde empieza y donde termina, y el archivo de salida debe incluir la declaración DOCTYPE como se muestra en el siguiente ejemplo.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <META http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
    <div>
        <div>
            <h3>Función: <a name=”main” >main</a></h3>
            <dl>
                <dt>Fecha</dt>
                <dd>19/09/2012</dd>
            </dl>
            <dl>
                <dt>Precondición</dt>
                <dd>El TDA debe estar creado.</dd>
```

```
            </dl>
            <dl>
                <dt>Precondición</dt>
                <dd>El archivo de entrada debe corresponder a una cadena de
                    texto bien formada.</dd>
            </dl>
        </div>
    </div>
</body>
</html>
```

Archivo de configuración:

El archivo de configuración que utilizará el documentador debe contemplar la inclusión de información necesaria para todos sus componenetes.

El programa recibirá los siguientes argumentos.

- Para los archivos de entrada (se usa el directorio): -i directorio_entrada
- Para el archivo de salida: -o output.html
- Para el archivo de log; se abre para append siempre que exista: -l log

Por ejemplo, estas serían invocaciones válidas

```
./tp -i directorio_entrada -o Informe.html -l /tmp/logfile.log
./tp -o Manual_TP.html -i src -l log
```

y estas inválidas

```
./tp archivodeentrada.txt
./tp /tmp/logfile.log -i sources
./tp -icl archivodeentrada.txt configuracion.txt log
./tp -l /tmp/logfile.log miArchivoDeEntrada.txt
```

Incluya además la posibilidad de invocar al programa con `-h` o `--help` y que imprima una breve ayuda para la invocación demostrando la sintaxis.

Nota: El programa deberá realizar el chequeo de todas las pre condiciones de las primitivas del TDA que utilice y deberá mostrar mensajes acordes ante las situaciones de error que se presenten.

Aclaraciones varias:

El núcleo del trabajo puede resumirse en lo siguiente:

- leer archivos de texto que cuenten con la sintaxis esperada, ignorando todo lo que no esté demarcado por los delimitadores de contenido
- validar la sintaxis y semántica del texto leído
- transformar los valores, según corresponda, en los diferentes tags html indicados en la tabla precedente, e imprimirlos en el archivo de salida.

Sin embargo, deben tenerse en cuenta cuestiones de implementación que se requieren como parte obligatoria de la solución.

Sobre el índice:

Cada vez que se procesa un bloque de documentación, el documentador deberá guardar un registro de ese bloque, para la posterior confección del índice. Dicho índice tendrá una entrada

por cada bloque procesado, identificado por el nombre de función que declare con el tag @funcion.

Finalmente, al invocar a la primitiva de construcción del índice, se emitirá en el archivo de salida una lista de items, con enlaces (hipervínculos) a la documentación generada.

Para recopilar la información que contendrá el índice, utilice una Lista Simple (implementación provista por la cátedra). Mantenga los elementos de la lista ordenados por el nombre de la función que identifica al bloque. La información que debe guardar, además del nombre del bloque, es el archivo de documentación donde se imprimió la salida correspondiente a ese bloque. En resumen, use una lista que contenga, al menos, el nombre de cada función documentada, y la ruta al archivo de salida en el cuál se documentó.

Para imprimir el índice, utilice los tags html: ul y li. Vea el siguiente ejemplo de archivo de índice.

```
<h1>&Iacute;ndice</h1>
<hl />
<ul>
    <li><a href=”doc.html#miFuncion01”>miFuncion01</a></li>
    <li><a href=”doc.html#miFuncion02”>miFuncion02</a></li>
    <li><a href=”doc.html#miFuncion03”>miFuncion03</a></li>
    <li><a href=”doc.html#main”>main</a></li>
</ul>
```

El archivo de índice se generará bajo el nombre del archivo de salida o output, intercalándole un interfijo “idx” antes de la extensión, separado por puntos. Algunos ejemplos

- -o doc.html -> doc.idx.html
- -o mi.nombre.de.archivo.con.puntos.html -> mi.nombre.de.archivo.con.puntos.idx.html
- -o miarchivosinextension -> idx.miarchivosinextension

Pre-Entrega

Para la preentrega de la primer semana, deberán presentarse los siguientes ítems, y cualquier agregado puntual que el ayudante a cargo determine conveniente:

- Divisón de tareas estimada para el grupo, con algún soporte de tipo gráfico y una prosa explicativa que complete la información del gráfico.
- Un diagrama de entidades donde se muestre gráficamente cuáles son las partes que constituyen la aplicación (programa de aplicación, documentador, etc), y las conexiones entre ellas.
- Archivo de encabezados del TDA_Doc como se utilizará (documentador.h)
- Un modelo de archivo de cada uno de los archivo de configuración. Esto no significa que deberán explicitarse por extensión, sino un ejemplo de entre 5 y 10 líneas donde se pueda entender el formato.

Entrega

Para la entrega, deberá presentarse el código fuente del programa de aplicación completo, subido al sistema de compilación automática, enviado por e-mail, o de la manera que el ayudante a cargo del grupo indique. Junto con el código fuente, se deberá enviar al menos dos escenarios de prueba con los que el grupo haya probado la aplicación, conteniendo cada uno de estos un juego de archivos de configuración completo, un archivo de entrada y un archivo más conteniendo la salida esperada. No se aceptarán diskettes para esto.
Además deberá presentarse un informe que incluya toda la documentación presentada en la preentrega a excepción de los archivos de cabeceras. Se deberá incluir además la distribución real de las tareas, un listado detallado de las consideraciones que se debieron tomar para resolver el trabajo práctico y un instructivo de cómo usar el programa una vez compilado (use de ejemplo una manpage de algún comando de Linux, por ejemplo: `man grep`; puede ver esta página en unix-help[1]; podrá encontrar más man pages en kernel-man-pages[2]).

Enlaces

Repositorio
Log - https://subversion.assembla.com/svn/algo2_primitivas/src/Primitivas en uso/src/
cabeceras: log.h
fuente: log.c
Lista Simple - https://subversion.assembla.com/svn/algo2_primitivas/src/Primitivas en uso/src/
cabeceras: straight_list.h
fuente: straight_list.c

E-Book
El lenguaje C - K&R - http://zanasi.chem.unisa.it/download/C.pdf
Idem anterior - http://www.utnianos.com.ar/foro/attachment.php?aid=4254

Ejemplo de man page
[1] - http://unixhelp.ed.ac.uk/CGI/man-cgi?grep
[2] - http://www.kernel.org/doc/man-pages/

Herramientas conocidas
Diagramación: Gliffy - http://www.gliffy.com/
IDE: Eclipse - http://www.eclipse.org/ -
downloads:http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers-includes-incubating-components/indigosr2
IDE: CodeBlocks - http://www.codeblocks.org/ -
downloads: http://www.codeblocks.org/downloads/binaries
p/windows: http://prdownload.berlios.de/codeblocks/codeblocks-10.05mingw-setup.exe

Información útil
http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html
http://www.acm.uiuc.edu/webmonkeys/book/c_guide/
http://www.cplusplus.com/reference/
http://en.cppreference.com/w/