Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

IE-0117 Programación Bajo Plataformas Abiertas

Ing. Marco Villalta Fallas - II Ciclo 2017

Laboratorio # 7

C: entrada/salida archivos, memoria dinámica, punteros (Intermedio) y structs

Instrucciones Generales:
Los laboratorios se deben de realizar de manera individual.
El laboratorio debe de entregarse a más tardar el 30 de mayo antes de las 23:59.
Entregue un archivo comprimido que incluya un directorio llamado `informe` con los archivos necesarios para generar el PDF del informe (.tex, imágenes, código, entre otros) y un directorio llamado `src` con los archivos de código fuente que lleven a la solución. En la calificación de este laboratorio se tomará en cuenta el uso del control de versiones git y la documentación en Doxygen. Por complejidad, el valor de este laboratorio equivale a 2 laboratorios.

1. Introducción

El siguiente laboratorio tiene como objetivo comprobar los conocimientos adquiridos durante el curso. Específicamente se evaluará el buen manejo del lenguaje de programación C. Los temas a evaluar son: uso de herramientas de automatización de proyectos (Makefiles, cmake), uso de archivos de encabezado y archivos de implementación (.h y .c respectivamente), funciones, buen manejo de punteros y memoria dinámica, correcta implementación en entrada/salida de archivos, manipulación de structs y por supuesto la funcionalidad del código. Para ello se presenta como ejercicio académico la implementación de una biblioteca en el lenguaje de programación C capaz de proveer la funcionalidad de listas enlazadas.

2. Nota teórica

A continuación se provee un pequeño resumen de la estructura de datos a implementar.

2.1. Listas enlazadas

En la Figura 1 se puede apreciar una visualización de lo que sería la estructura da datos de lista enlazada.

Una lista cuenta con posiciones. Cada una de estas almacena dos datos importantes. El primero corresponde al contenido o valor relevante que se almacena en la posición y el segundo corresponde a un puntero que apunta al siguiente elemento de la lista.

Una ventaja muy grande que tienen las listas es el tiempo que tarda en añadirse un elemento nuevo. Es de orden O(1) ya que basta con tomar el puntero almacenado en la posición anterior, y guardarlo como el puntero siguiente del elemento nuevo, y cambiar el puntero de la posición anterior por la posición de memoria donde se encuentra el elemento nuevo. La Figura 2 muestra el procedimiento de una manera gráfica.

El eliminar elementos tiene una lógica similar a esta, pero en el sentido inverso, asegurando que el elemento anterior al que se va a eliminar, guarde en su puntero la posición de memoria donde se encuentra el elemento posterior al que se va a eliminar.

Con estas consideraciones en mente, y con una lógica simple, se puede construir la biblioteca para manejar listas en C.

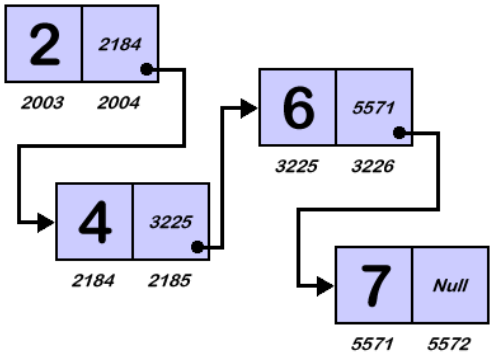

Figura 1: Representación de una lista enlazada

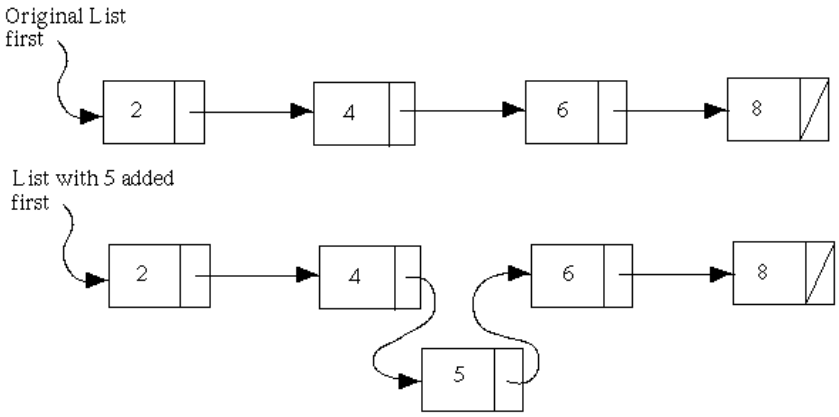

Figura 2: Representación de una lista enlazada

3. Desarrollo

El estudiante deberá de implementar en el lenguaje C una biblioteca capaz de utilizar listas enlazadas y proveer un programa principal en el que se evidencie todas las funcionalidades de la biblioteca de listas.

3.1. Código fuente

La implementación que debe realizar el estudiante consta de los siguiente archivos de código fuente:

1. `lista.h` y `lista.c`: código fuente con el conjunto de funciones que serán llamadas desde el programa principal para proveer las funcionalidades de una lista almacenada en memoria dinámica.
2. `main.c`: código fuente del programa principal encargado de hacer las llamadas a las funciones apropiadas para la correcta manipulación de las listas.

El contenido de los archivos se detalla a continuación.

3.1.1. Archivo lista.h y lista.c

Estos son los archivos donde se realizará la mayor parte de la implementación del código. En el archivo `lista.h` deben de aparecer las siguientes funciones, variables y estructuras de datos:

```
1 typedef struct pos {
2 // data
3 int data;
4 // pointer to next element
```

```
5  struct pos_t* next;
6  } pos_t;
7
8  pos_t* createList(int first_value);
9
10 pos_t* readList(const char* filePath);
11
12 void writeList(pos_t* head, const char* filePath);
13
14 int push_back(pos_t* head, int new_value);
15
16 int push_front(pos_t** head, int new_value);
17
18 int pop_back(pos_t* head);
19
20 int pop_front(pos_t** head);
21
22 int insertElement(pos_t** head, int pos, int new_value);
23
24 void removeElement(pos_t** head, int pos);
25
26
27 int freeList(pos_t* head);
28
29 int getElement(pos_t* head, int index, int* valid);
30
31 int printElement(const int value)
32
33 void sort(pos_t* head, char dir);
34
35 void printList(pos_t* head);
```

- **Tipo de dato `post_t`:** Tipo de dato compuesto. Representa cada elemento individual de la lista y con ellos se construye la lista completa. Almacena un dato entero (`int`) y el puntero al siguiente elemento.
- **Función createList:** Esta función regresa un puntero a una variable del tipo `pos_t` y recibe el entero que se va a almacenar en la primera posición de la lista. Debe reservar memoria solo para un elemento. Ya con este puntero se pueden utilizar las demás funciones de la biblioteca.
- **Función printList:** esta función se encarga de imprimir toda la lista seguida de cambios de línea. Recibe como parámetro únicamente el puntero al primer elemento de la lista.
- **Función freeList:** esta función se encarga de liberar toda la memoria que haya sido localizada para la lista. Recibe como único parámetro el puntero al primer elemento de la lista.
- **Función push_back:** esta función regresa un entero que es 0 en caso de que el elemento se haya podido añadir correctamente al final de la lista, o bien, 1 si no se pudo reservar memoria para el nuevo elemento de la lista. Recibe como parámetro el puntero al primer elemento de la lista, y el nuevo valor que se desea agregar. Debe reservar memoria.
- **Función push_front:** esta función regresa un entero que es 0 en caso de que el elemento se haya podido añadir correctamente al inicio de la lista, o bien, 1 si no se pudo reservar memoria para el nuevo elemento de la lista. Recibe como parámetro el puntero al puntero que a punta al primer elemento de la lista, y el nuevo valor que se desea agregar. Debe reservar memoria.
- **Función pop_back:** esta función elimina la última posición de la lista y además regresa el valor entero que estaba almacenado en dicha posición. Además debe de hacer los cambios necesarios en la lista para

que siga siendo una lista enlazada válida. Recibe como parámetro el puntero al primer elemento de la lista. Debe liberar memoria.

- **Función pop_front:** esta función elimina la primera posición de la lista y además regresa el valor entero que estaba almacenado en dicha posición. Además debe de hacer los cambios necesarios en la lista para que siga siendo una lista enlazada válida. Recibe como parámetro el puntero al puntero que a punta al primer elemento de la lista. Debe liberar memoria.
- **Función insertElement:** esta función agrega un elemento a la lista. Recibe como parámetros el puntero al puntero que a punta al primer elemento de la lista, la posición donde desea insertar el nuevo elemento y el valor entero que desea agregar. Regresa 0 en caso de haber agregado correctamente el elemento, o 1 en caso de no poder localizar la memoria necesaria. Debe reservar memoria.
- **Función removeElement:** esta función elimina un elemento de la lista. Recibe como parámetros el puntero al puntero que a punta al primer elemento de la lista y la posición de la lista que desea eliminar. Regresa el valor del elemento eliminado o -1 en caso de haber un error. Debe liberar memoria.
- **Función getElement:** esta función se encarga de regresar el valor de un elemento específico de la lista. Recibe como parámetro el puntero al primer elemento de la lista, la posición que desea obtener y un puntero a int donde se almacenará un 0 en caso de que el valor retornado sea valido o 1 en caso contrario. Un valor no será válido en caso de que el usuario solicite una posición que no existe en la lista.
- **Función sort:** esta función se encarga de ordenar la lista. Lo realiza de manera ascendente si el `char` recibido como parámetro es 'a', o de manera descendente en caso de que el char recibido sea 'd'. Deja la lista intacta en caso de recibir cualquier otra letra. Además recibe como parámetro el puntero al primer elemento de la lista.
- **Función readList:** Esta función regresa el puntero al primer elemento de la lista creada a partir del archivo binario ubicado en la ruta especificado por el parámetro filePath. El archivo antes mencionado contiene una lista de números enteros separados por cambios de línea.
- **Función writeList:** Esta función escribe una lista en un archivo de texto plano. Recibe como parámetro el puntero al primer elemento de la lista y la ruta del archivo donde se quiere guardar la lista. Cada elemento de la lista se escribirá separado por un cambio de línea en el archivo de salida.

  Puntos extra:
  Implemente la función printList usando el puntero a función de printElement donde:

- **Función printElement:** esta función se encarga de imprimir en consola un número entero seguido de un espacio, con cambios de línea. Recibe como parámetro el número entero que desea imprimir.

Nota: ya se proveen los archivos `lista.h` y `lista.c` en el sitio web del curso, el estudiante debe concentrarse únicamente en la implementación de cada una de las funciones dentro del archivo `lista.c` y no debe de modificar nada en el `lista.h`.

3.1.2. Archivo main.c

Este archivo debe incluir el header de `lista.h` y únicamente debe de hacer llamados a las funciones de la biblioteca creada para demostrar su funcionalidad de manejar listas de manera adecuada. Debe tener al menos una declaración de una variable tipo pos_t.

3.2. Estructura del programa

La organización del código fuente debe seguir la estructura definida en clase (directorio raíz con los directorios `src`, `include` y `build`), así como un archivo Makefile o CMakeLists.txt para la compilación automatizada del código fuente. Si no se cuenta con alguno de estos dos medios para compilar el trabajo, o la compilación falla, se asignará una nota de cero.

3.3. Informe final del laboratorio

Finalmente, luego de concluir con la implementación, debe de realizar un informe técnico con los detalles de la elaboración del código fuente, así como el programa principal que evidencie todas las funcionalidades de la biblioteca implementada. Las funcionalidades que no se encuentren debidamente explicadas en el informe, serán penalizadas en la nota final del laboratorio. A los estudiantes que no presenten el informe escrito se les asignará una nota de cero.