PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

PROGRAMACIÓN 2

4ta práctica (tipo b)
(Segundo Semestre 2025)

Indicaciones Generales:

- Duración: **110 minutos.**
- No se permite el uso de apuntes de clase, fotocopias ni material impreso.
- No se pueden emplear variables globales, ni objetos (con excepción de los elementos de `iostream`, `iomanip` y `fstream`). No se puede utilizar la clase `string`. Tampoco se podrán emplear las funciones de C que gesten memoria como `malloc`, `realloc`, `memset`, `strdup`, `strtok` o similares, igualmente no se puede emplear cualquier función contenida en las bibliotecas `stdio.h`, `cstdio` o similares y que puedan estar también definidas en otras bibliotecas. No podrá emplear plantillas en este laboratorio.
- Deberá modular correctamente el proyecto en archivos independientes. Las soluciones deberán desarrollarse bajo un estricto diseño descendente. Cada función no debe sobrepasar las 20 líneas de código aproximadamente. El archivo `main.cpp` solo podrá contener la función `main` de cada proyecto y el código contenido en él solo podrá estar conformado por tareas implementadas como funciones. En el archivo `main.cpp` deberá incluir un comentario en el que coloque claramente su nombre y código, de no hacerlo se le descontará 0.5 puntos en la nota final.
- El código comentado no se calificará. De igual manera no se calificará el código de una función si esta función no es llamada en ninguna parte del proyecto o su llamado está comentado.
- Los programas que presenten errores de sintaxis o de concepto se calificarán en base al 40 % de puntaje de la pregunta. Los que no muestren resultados o que estos no sean coherentes en base al 60 %.
- Se tomará en cuenta en la calificación el uso de comentarios relevantes.
- Se les recuerda que, de acuerdo al reglamento disciplinario de nuestra institución, constituye una falta grave copiar del trabajo realizado por otra persona o cometer plagio.
- No se harán excepciones ante cualquier trasgresión de las indicaciones dadas en la prueba.

Puntaje total: 20 puntos

- La unidad de trabajo será `t:\` (Si lo coloca en otra unidad, no se calificará su laboratorio y se le asignará como nota cero). En la unidad de trabajo `t:\` colocará el(los) proyecto(s) solicitado(s).
- Cree allí una carpeta con el nombre "Lab04_2025_2_CO_PA_PN" donde: **CO** indica: Código del alumno, **PA** indica: Primer Apellido del alumno y **PN** indica: primer nombre. De no colocar este requerimiento se le descontará 3 puntos de la nota final.

Cuestionario

- La finalidad principal de este laboratorio es la de reforzar los conceptos contenidos en el capítulo 2 del curso: "Arreglos y punteros". En este laboratorio se trabajará con **punteros genéricos** y **memoria dinámica**.
- Al finalizar la práctica, comprima la carpeta dada en las indicaciones iniciales empleando el programa `Zip` que viene por defecto en el Windows, no se aceptarán los trabajos compactados con otros programas como `RAR`, `WinRAR`, `7zip` o similares.

Para el diseño de su solución, considere que:

- No podrá emplear arreglos estáticos de más de una dimensión.
- No puede manipular un puntero con más de un índice.
- No puede emplear arreglos auxiliares, estáticos o dinámicos, para guardar los datos de los archivos.
- Los archivos solo se pueden leer una vez.

Descripción del caso

Se desea implementar una lista simplemente enlazada genérica que, mediante el uso combinado de punteros genéricos (`void*`) y punteros a funciones, permita realizar las operaciones básicas propias de este tipo abstracto de dato (TAD). Para ello se ha definido el tipo de dato `list` de la siguiente manera:

Código 1: list.h

```
typedef struct {
    int size;
    void *front;
    void *back;
} list;
```

Donde:

- `size`: almacena la cantidad de elementos que tiene la lista en un momento dado.
- `front`: es un puntero genérico que indica el primer nodo de la lista, es decir, el inicio de la secuencia.
- `back`: es un puntero genérico que apunta al último nodo de la lista, lo que facilita agregar elementos al final.

Pregunta 1

Se requiere implementar las siguientes funciones que permiten crear la lista:

- (1 punto) La primera función se llama `new_list` y su propósito es inicializar la lista. Al ejecutarse, debe dejar la cantidad de elementos en cero y establecer tanto el puntero al primer nodo como el puntero al último nodo en nulos, ya que la lista comienza vacía..
- (3 puntos) La segunda función se denomina `push_front` y se utiliza para insertar un nuevo elemento al inicio de la lista. Esta función recibe el valor que se desea agregar y un puntero a una función que permite clonar dicho valor para almacenarlo en la lista. A partir de ello, debe construirse un nuevo nodo que contenga el clon y que se enlace con el nodo que antes ocupaba la primera posición. El puntero al primer nodo debe actualizarse para señalar al nodo recién creado, y en caso de que la lista estuviera vacía también debe actualizarse el puntero al último nodo para que coincida con el mismo. Finalmente, la cantidad de elementos de la lista debe incrementarse en uno.
- (3 puntos) La tercera función recibe el nombre de `push_back` y permite insertar un nuevo elemento al final de la lista. Al igual que la anterior, recibe un valor y un puntero a una función que lo clona. Con estos parámetros debe construirse un nodo cuyo puntero siguiente sea nulo. Si la lista ya tenía elementos, el nodo que antes era el último debe enlazarse con este nuevo. Luego, el puntero al último nodo debe actualizarse para que apunte al nodo recién creado. En el caso de que la lista estuviera vacía, tanto el puntero al primer nodo como el puntero al último deben señalar a este nuevo elemento. Al finalizar, la cantidad de elementos debe incrementarse en uno.

Se le ha compartido un proyecto como soporte a la resolución de este laboratorio. Para probar su solución, ejecute las pruebas `test01`, `test02` y `test03` que se encuentran en la función `main`.

Pregunta 2

Además de las operaciones básicas de inserción, la lista simplemente enlazada genérica requiere funciones que permitan recorrerla de manera sistemática. Para este propósito se implementarán un conjunto de funciones que cumplen el rol de un **iterador**, es decir, un mecanismo que facilita recorrer los elementos de la lista uno por uno, sin exponer directamente los detalles internos de la estructura.

- (1.5 puntos) La primera función se denomina `begin` y se encarga de devolver el punto de inicio del recorrido de la lista. En otras palabras, al ser llamada debe retornar un puntero al primer nodo de la lista, lo cual marca el comienzo de la iteración.
- (0.5 puntos) La segunda función se llama `end` y representa el punto final de la iteración. En este caso, la función devuelve siempre un valor nulo, lo cual indica que no existen más nodos que recorrer en la lista.
- (2 puntos) La tercera función recibe el nombre de `next` y tiene como propósito avanzar en la lista a partir de la posición actual del iterador. Al recibir un puntero a un nodo, debe retornar un puntero al siguiente nodo de la lista, lo que permite continuar con el recorrido.
- (2 puntos) La cuarta función se llama `getValue` y permite obtener el valor almacenado en el nodo señalado por el iterador. De esta manera, dado un nodo actual, la función devuelve el dato contenido en él, sin modificar la posición del recorrido.

Se le ha compartido un proyecto como soporte a la resolución de este laboratorio. Para probar su solución, ejecute las pruebas `test04`, `test05` y `test06` que se encuentran en la función `main`.

Pregunta 3

Para aprovechar las operaciones de recorrido sobre la lista genérica, se requiere implementar funciones que permitan aplicar acciones a todos los elementos y buscar información de manera flexible. Con este fin se implementarán en dos funciones: `foreach` y `findIf`.

- (3.5 puntos) La función `foreach` tiene como propósito recorrer todos los elementos de la lista desde el primero hasta el último y aplicar sobre cada uno de ellos una acción indicada por el programador. Para lograr esto, recibe como parámetro adicional una función que define cómo se debe procesar cada valor, por ejemplo, una función de impresión. Durante el recorrido, la lista es iterada nodo por nodo usando las funciones `begin`, `end` y `next`. En cada iteración se obtiene el valor del nodo actual y se aplica la función recibida. *Se le ha compartido un proyecto como soporte a la resolución de este laboratorio. Para probar su solución, ejecute las pruebas `test07`, `test08` y `test09` que se encuentran en la función `main`.*
- (3.5 puntos) La función `findIf` está orientada a la búsqueda de elementos dentro de la lista. En este caso, además de la lista, se recibe una clave y una función de comparación. El recorrido se realiza de manera similar a la función anterior, avanzando nodo por nodo desde el inicio hasta el final. En cada paso se obtiene el valor del nodo actual y se evalúa la condición de comparación entre dicho valor y la clave proporcionada. Si en algún momento la comparación resulta verdadera, significa que se ha encontrado el elemento buscado y la función debe indicar éxito. Si el recorrido finaliza sin coincidencias, la función debe indicar que no se encontró el valor. *Se le ha compartido un proyecto como soporte a la resolución de este laboratorio. Para probar su solución, ejecute las pruebas `test10`, `test11` y `test12` que se encuentran en la función `main`.*

Profesores del curso: Miguel Guanira, Andrés Melgar, Rony Cueva, Eric Huiza, Erasmo Gómez

Pando, 3 de octubre de 2025