University Ferhat Abess-Sétif 1
Faculty of sciences
Computer sciences Departement
Module: ASD3
Level: 2nd-year Computer Science Engineering Students **Dr. Lakehal Abderrahim**

TP 0

Goal: In this TP, we will deepen our understanding of fundamental data structures—linked lists, stacks, and queues—by implementing them in **C++.** We will practice essential operations such as insertion, deletion, and traversal, while also learning to identify and correct common programming errors.

Instructions

Type the source code for each section, write a complete `int main()` to test it, and correct all lexical, syntax, semantic, runtime, and logic errors. Finally, test your program with different inputs and compare the behaviors of linked lists, stacks, and queues.

> **Warning!**
> The use of any AI tool (including, but not limited to, ChatGPT, GitHub Copilot, Bard, or any automated code generator) to produce, complete, modify, or debug code for assignments, labs, projects, or exams in this course is strictly forbidden. Any submission found to have been created or substantially assisted by AI, copy-pasted from external sources, or showing two (or more) substantially similar code versions will be treated as plagiarism and is strictly prohibited. Such violations will result in a zero for the affected work and will be reported to the Academic Integrity Committee; repeated or egregious cases may lead to course failure and further disciplinary action. Claims of misunderstanding or ignorance of this policy will not be accepted as a defense.

A. Music Playlist Manager (Circular Linked List)

Objective:

Simulate a **music playlist system** using a **circular linked list**, where the last node points back to the first, allowing infinite navigation in both directions.

1. Node Structure `SongNode`, each node represents a song with the following information:

- `int id` → Unique song ID,
- `string title` → Title of the song,
- `string artist` → Artist or band name,
- `float duration` → duration in munites,
- `SongNode* next` → Pointer to the next song,
- `SongNode* prev` → Pointer to the previous song,

2. Define the Class `PlayList` (**CircularLinkedList**)

- Create a class `PlayList` with two attributes:
  - `SongNode* head;` → pointer to the first song in the playlist
  - `SongNode* current;` → pointer to the currently playing song
- Add a **constructor** (`PlayList:: PlayList()`) that initializes an empty list (`head = nullptr`, `current = nullptr`)
- Add a **destructor** (`PlayList::~PlayList()`) to free all allocated nodes.

3. Implement the following methods in the class PlayList

- `addSong(int id, string, title, string artist, float duration)` → insert a new song at the end of the playlist,
- `removeSong(string title)` → remove a song by its title,
- `playNext()` → move forward to the next song and print currently playing,
- `playPrev()` → move backward to the previous song and print currently playing,
- `displayPlaylist()` → Display all songs in the playlist in order (loop only once through the circular list),
- `searchSong(string title)` → Search for a song by title and print its details.
- `shufflePlaylist()` (optional, harder) → Rearrange songs randomly while keeping circularity,
- `repeatCurrent()` → Play the current song again without moving forward/backward.

4. Main program (`int main()`): Testing Scenario:

- Create a **circular playlist**.
- Insert at least **6 songs** with real-looking data (title, artist, duration).
- Display the playlist.
- Play 3 songs forward using `playNext()`.
- Play 2 songs backward using `playPrev()`.
- Search for a specific song by title.
- Remove one song and display the updated playlist.
- Use `repeatCurrent()` to replay the same track.
- (Optional) Call `shufflePlaylist()` to mix songs and test navigation again.

University Ferhat Abess-Sétif 1
Faculty of sciences
Computer sciences Departement
Module: ASD3
Level: 2nd-year Computer Science Engineering Students **Dr. Lakehal Abderrahim**

B. Bank Queue System (Queue)

Objective

Simulate a bank waiting line system where customers arrive, wait in a queue, and are served **first-come, first-served (FIFO)** using a **linked list implementation of a queue**.

1. Node Structure `CustomerNode`, each customer is represented as **node** in a linked list:

- `int customerID` → unique identifier for the customer,
- `Node* next` → pointer to the next customer.

2. Define the Class `Queue` (**Queue**)

- Create a class `Queue` with two attributes:
  - `CustomerNode* head;` → first customer in line
  - `CustomerNode* rear;` → last customer in line
- Add a **constructor** (`Queue::Queue()`) that initializes an empty list (`head = nullptr`, `rear = nullptr`)
- Add a **destructor** (`Queue::~Queue()`) to free all allocated nodes.

3. Implement the following methods in the class `Queue`

- `enqueue(int id)` → add at rear a new node with `customerID = id`,
- `dequeue()` → remove the node at the front of the queue,
- `displayQueue()` → traverse and print from front to rear.
- `Peek()` → Show which customer will be served next.

4. Main program (`int main()`): Testing Scenario:

- Create a **Queue** object.
- Add 5 customers (IDs: 101, 102, 103, 104, 105).
  - Queue = [101, 102, 103, 104, 105]
- Serve 2 customers (call `dequeue()` twice).
  - Queue = [103, 104, 105]
- Display remaining customers with `displayQueue()`.