**University of Msila** **2025/2026**
**Department of Mathematics Level.**
**1st year Mathematics**
**Module : ASD2**

**Exercise series N° 3 (Linked Lists)**

**Exercise 1 (TD) :**

1. Declare an integer variable `score` and initialize it to 95.
2. declare a pointer `ptr_score` that points to `score`.
3. Print the memory address of `score` and the value of `score` using both the variable `score` and the pointer `ptr_score`.

**Exercise 2 (TP) :**

1. Declare a float variable `price` and initialize it to 19.99.
2. Declare a pointer `ptr_price` to price.
3. Using `ptr_price`, change the value of price to 25.50.
4. Print the new value of price using the variable `price`.

**Exercise 3 (TD) :**

Given a character array `message` initialized with "Hello", use a pointer `ptr_message` to print each character of the string. Do not use array indexing ([]) after initializing the pointer.

**Exercise 4 (TP) :**

Declare an integer variable `num` with value 10. Declare a pointer `ptr1` that points to num. Then, declare a pointer `ptr2` that points to `ptr1`. Print the value of `num` using `ptr2`.

**Exercise 5 (TD) :**

Write a program that asks the user for the number of integers they want to store. Dynamically allocate an array of that size using `malloc` function. Read the integers from the user into this array and then print them. Finally, free the allocated memory.

**Exercise 6 (TP) :**

Write a function `swap(int *a, int *b)` that takes two integer pointers as arguments and swaps the values they point to. In main, declare two integer variables, initialize them, call swap to exchange their values, and then print the values to verify the swap.

**Exercise 7 (TD) :**

Implement your own version of the strlen function, named `my_strlen`, which takes a `const char *s` as input and returns the length of the string (number of characters before the null terminator). Your implementation must use pointer arithmetic and not array indexing.

**Exercise 8 (TD)**

Define a struct for a singly linked list `node` that stores a char type value and a pointer to the next node. Use typedef to create an alias `CharNode` for this structure.

**Exercise 9 (TP):**

Create a singly linked list with two nodes. The first node should store the integer 5 and the second node 15. Ensure the last node's next pointer is NULL. Print the data of both nodes by traversing the list.

**Exercise 10 (TD):**

Write a function `isEmpty(Node *head)` that returns 1 if the singly linked list is empty (i.e., head is NULL) and 0 otherwise. Test it with an empty list and a non-empty list.

**Exercise 11 (TP):**

Write a function `findMiddle(Node *head)` that takes the head of a singly linked list and returns the data of the middle element. If the list has an even number of elements, return the data of the second middle element. For example, in 1->2->3->4, return 3. Use the fast and slow pointer approach.

**Exercise 12 (TD):**

Write a function `concatenateLists(Node *head1, Node *head2)` that takes the heads of two singly linked lists and appends the second list to the end of the first list. The function should return the head of the concatenated list.

**Exercise 13 (TP):**

Write a function `printList(Node *head)` that takes the head of a singly linked list and prints all its elements. If the list is empty, it should print an appropriate message. Use the Node structure defined previously.

**Exercise 14 (TD):**

Implement a function `insertAtBeginning(Node *head, int newData)` that creates a new node with newData and inserts it at the beginning of the singly linked list. The function should return the new head of the list.

**Exercise 15 (TP):**

Implement a function `insertAtEnd(Node *head, int newData)` that creates a new node with newData and inserts it at the end of the singly linked list. The function should return the head of the list.

**Exercise 16 (TD):**

Implement a function `deleteFromBeginning(Node *head)` that deletes the first node of a singly linked list. The function should return the new head of the list. Handle the case where the list is empty.

**Exercise 17 (TP):**

Write a function `searchList(Node *head, int key)` that takes the head of a singly linked list and an integer key. The function should return 1 if the key is found in the list, and 0 otherwise.

**Exercise 18 (TD):**

Implement a Cfunction `deleteFromEnd(Node *head)` that deletes the last node of a singly linked list. The function should return the new head of the list. Handle cases where the list is empty or has only one node.

**Exercise 19 (TD):**

Define a struct `DoublyNode` for a doubly linked list that stores a double type value, a pointer to the previous node (`prev`), and a pointer to the next node (`next`).

**Exercise 20 (TP):**

Create a doubly linked list with three nodes storing 1.1, 2.2, and 3.3. Print the list in forward direction. Ensure `prev` and `next` pointers are correctly set, and NULL for appropriate ends.