المدرسة العليا للتكنولوجيا – الصويرة
ⵜⵉⵎⵣⴳⵉⵜ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵏ ⵜⵙⵙⵏⵜⵉⵔⵜ - ⵎⵓⵍⵉⵔⴰ
ÉCOLE SUPÉRIEURE DE TECHNOLOGIE - ESSAOUIRA

Département Génie Informatique et Mathématiques
Filière : DUT Informatique Décisionnelle et Science des Données

Structures de données fondamentales en langage C : Atelier pratique

Préparé par :

Fatiha Bendaida

Année universitaire : **2024/2025**

Table des matières

1 Listes chaînées en Langage C

1.1 Objectifs pédagogiques

— Comprendre le fonctionnement d'une liste simplement chaînée
— Implémenter les opérations de base
— Utiliser la mémoire dynamique
— Comparer aux tableaux
— Respecter les conventions de nommage

1.2 Règles de nommage

— Structure de nœud : `Node`
— Champs :
  — `int data`
  — `Node* next`
— Pointeur principal : `head`
— Prototypes imposés :
  — `Node* create_node(int value);`
  — `void insert_at_head(Node** head, int value);`
  — `void insert_at_tail(Node** head, int value);`
  — `void delete_value(Node** head, int target);`
  — `void print_list(Node* head);`
  — `void free_list(Node** head);`

1.3 Déclaration de la structure

```
1    typedef struct Node {
2      int data;
3      struct Node* next;
4    } Node;
5
```

1.4 Travail demandé

a) `create_node`
b) `insert_at_head`
c) `insert_at_tail`
d) `delete_value`
e) `print_list`
f) `free_list`

1.5 Tests

— Ajouter des éléments
— Afficher la liste
— Supprimer un élément
— Libérer la mémoire

2 Les piles (Stack) en Langage C

2.1 Objectifs du TP

— Comprendre et manipuler la structure pile

— Comparer : tableau vs liste chaînée

— Analyser les avantages et contraintes

2.2 Énoncé du TP

1. Implémenter une pile via un tableau
2. Implémenter une pile via une liste chaînée
3. Tester : `push`, `pop`, `top`, `isEmpty`
4. Rédiger un rapport comparatif

2.3 Pile via un tableau

```
1  void initStack(Stack *s);
2  int isEmpty(Stack *s);
3  int isFull(Stack *s);
4  void push(Stack *s, int x);
5  int pop(Stack *s);
6  int peek(Stack *s);
```

2.4 Pile via une liste chaînée

```
1  void initStackList(StackList *s);
2  int isEmptyList(StackList *s);
3  void pushList(StackList *s, int x);
4  int popList(StackList *s);
5  int peekList(StackList *s);
6  void freeStack(StackList *s);
```

2.5 Questions et extensions

— Quelle implémentation est plus efficace ?

— Comment rendre le tableau extensible ?

— Version générique avec `void*`

— Parseur d'expression postfixée

3 Les files (Queue) en Langage C

3.1 Résumé de cours

Une file (*queue*) est une structure de données de type FIFO (*First In, First Out*). Les opérations de base sont :

— `enqueue` - ajoute un élément à la fin

— `dequeue` - retire un élément du début

— `front` - consulte l'élément en tête
— `isEmpty` - vérifie si la file est vide
— `size` - retourne le nombre d'éléments

3.2 Complexité théorique

— Tableau : `enqueue` $O(1)$, `dequeue` $O(n)$
— Liste chaînée : Toutes opérations $O(1)$
— Buffer circulaire : Toutes opérations $O(1)$

3.3 Objectifs du TP

— Comprendre et implémenter la structure de données file
— Comparer trois approches :
  — Tableau avec décalage
  — Liste chaînée
  — Buffer circulaire
— Analyser les compromis performance/mémoire
— Appliquer à un cas concret

3.4 Implémentation requise

Structure commune

```
typedef struct {
  int capacity;
  int size;
  // Autres champs selon l'impl mentation
} Queue;

```

Fonctions obligatoires

```
void initQueue(Queue *q, int capacity);
void enqueue(Queue *q, int value);
int dequeue(Queue *q);
int front(Queue *q);
int isEmpty(Queue *q);
int isFull(Queue *q);
void freeQueue(Queue *q);

```

Questions et extensions

— Comparer les performances
— Implémenter une file prioritaire
— Adapter pour des données génériques (`void*`)
— Simuler un système de gestion de tâches