

Université Mohammed V
Ecole Nationale Supérieure de l'Informatique et de l'Analyse des Systèmes
Département Génie Logiciel

Structures de données en langage C

Support de TD/TP

Langage C (M—)
Semestre 1 – 2026–2027

2

Tome 2

Structures : modéliser les données et leurs liens

Pr. Youness BOUTYOUR

youness.boutyour@um5.ac.ma

Table des matières

Objectifs	4
Consignes de travail	4
Partie 1 : Structures et types composés	7
Exercice 1 – Point dans le plan	7
Exercice 2 – Étudiant	7
Exercice 3 – Date	7
Exercice 4 – Fraction	7
Exercice 5 – Complexe	7
Exercice 6 – Rectangle	7
Exercice 7 – Classement d'étudiants	8
Exercice 8 – Catalogue de produits	8
Exercice 9 – Adresse imbriquée	8
Exercice 10 – Agenda de rendez-vous	8
Exercice 11 – Matrice encapsulée	8
Exercice 12 – Statistiques d'un tableau	9
Exercice 13 – Mesure avec unité	9
Exercice 14 – Valeur polymorphe	9
Exercice 15 – Registre dynamique	9
Partie 2 : Listes chaînées	12
Exercice 16 – Création d'une liste	12
Exercice 17 – Insertion en tête	12
Exercice 18 – Insertion en queue	12
Exercice 19 – Compter les nœuds	12
Exercice 20 – Somme d'une liste	12
Exercice 21 – Recherche dans une liste	13
Exercice 22 – Suppression en tête	13
Exercice 23 – Suppression par valeur	13
Exercice 24 – Suppression de toutes les occurrences	13
Exercice 25 – Insertion ordonnée	13
Exercice 26 – Copie indépendante	14
Exercice 27 – Concaténation	14

Exercice 28 – Séparer les positifs	14
Exercice 29 – Liste doublement chaînée	14
Exercice 30 – Liste doublement chaînée robuste	15

Partie 3 : Piles et files **18**

Exercice 31 – Pile de caractères	18
Exercice 32 – Pile dynamique	18
Exercice 33 – Parenthèses équilibrées	18
Exercice 34 – Inversion par pile	18
Exercice 35 – Conversion décimal-binaire	18
Exercice 36 – File d'entiers	18
Exercice 37 – File de tâches	19
Exercice 38 – File avec deux piles	19
Exercice 39 – File circulaire	19
Exercice 40 – Palindromes avec une pile et une file	19
Exercice 41 – Évaluation postfixée	19
Exercice 42 – Historique annuler-refaire	20
Exercice 43 – Priorité de tâches	20
Exercice 44 – File de tickets	20
Exercice 45 – Bibliothèque pile-file	20

Partie 4 : Arbres, hachage et graphes **23**

Exercice 46 – Arbre binaire	23
Exercice 47 – Parcours d'arbre	23
Exercice 48 – Compter les nœuds d'un arbre	23
Exercice 49 – Arbre de recherche	23
Exercice 50 – Recherche dans un arbre	23
Exercice 51 – Suppression dans un arbre	24
Exercice 52 – Expression arithmétique en arbre	24
Exercice 53 – Table de hachage simple	24
Exercice 54 – Dictionnaire de mots	24
Exercice 55 – Suppression avec hachage	24
Exercice 56 – Graphe par matrice	25
Exercice 57 – Graphe par listes	25
Exercice 58 – Degrés d'un graphe	25

Exercice 59 – File de parcours	25
Exercice 60 – Projet de carnet de relations	26

Objectifs

- Modéliser des données avec des structures C et des types composés.
- Construire, parcourir et libérer des structures dynamiques.
- Choisir une structure adaptée à une situation algorithmique.
- Vérifier les invariants et traiter les erreurs d'allocation.

Consignes de travail

Créer un fichier C indépendant par exercice. Compiler avec `gcc -std=c11 -Wall -Wextra -Wpedantic`, tester les cas ordinaires et les cas limites, puis libérer toute mémoire allouée dynamiquement.

```
gcc -std=c11 -Wall -Wextra -Wpedantic exercice.c -lm -o exercice
```

1 PARTIE

Structures et types composés

NOTIONS

structures, champs, structures imbriquées, tableaux de structures et unions

COMPÉTENCES

modéliser proprement des objets et contrôler leurs invariants

PROGRESSION

Initiation → application

Structures et types composés

Notions de la partie, code C commenté et tests repères.

Notions essentielles

Déclarer et initialiser.

```
typedef struct { double x; double y; } Point;
```

Un objet peut être initialisé par désignation : `Point p = { .x = 1.0, .y = 2.0 };`. Le point s'écrit `p.x`; un pointeur vers une structure s'écrit `p->x`, équivalent à `(*p).x`.

Modéliser : regrouper dans une structure les données qui décrivent un même objet. Une structure peut contenir une autre structure, par exemple `Personne.adresse.ville`. Un tableau de structures permet de représenter une collection de taille connue : `Etudiant classe[20]`.

Règles de cohérence : documenter les unités, les bornes et les valeurs interdites. Une date doit être valide, une note doit rester dans son intervalle et une quantité ne doit pas devenir négative.

Exemples C commentés

Fonctions et passage d'arguments.

Une structure peut être retournée par valeur. Pour modifier l'objet reçu, transmettre son adresse : `void lire(Point *p);`. Pour éviter une copie inutile, utiliser `const Point *p` lorsque la fonction ne modifie pas l'objet. Séparer les fonctions de lecture, de calcul et d'affichage.

Union étiquetée : une union partage sa mémoire ; un seul membre est valide à la fois. L'étiquette doit être conservée dans un champ séparé : `enum Type { ENTIER, REEL, CARACTERE };`. Ne jamais lire un membre qui ne correspond pas à l'étiquette.

Allocation dynamique :

```
p = malloc(capacite * sizeof *p);
```

Tester le résultat de `malloc`. Pour `realloc`, conserver un pointeur temporaire, mettre à jour la capacité seulement après succès, puis appeler `free` exactement une fois.

Exemple : `Point *p = malloc(sizeof *p);`

```
if (p != NULL) { p->x = 1.0; free(p); }
```

```
// Le test précède toute déréréférence.
```

Partie 1 : Structures et types composés



Exercice 1 – Point dans le plan

Contexte : une application de cartographie manipule la position d'un objet.

Travail demandé : définir `Point` avec deux coordonnées réelles, lire un point et afficher ses coordonnées ainsi que $x^2 + y^2$, le carré de sa distance à l'origine.

À vérifier : tester un point à l'origine et un point avec des coordonnées négatives.



Exercice 2 – Étudiant

Contexte : produire la fiche minimale d'un étudiant.

Travail demandé : définir `Etudiant` avec un nom, un identifiant et une note réelle. Lire une fiche puis afficher les champs sur des lignes alignées.

Contraintes : réserver une capacité suffisante pour le nom et refuser une lecture incomplète.



Exercice 3 – Date

Objectif : comparer deux événements dans un calendrier.

Travail demandé : définir `Date`, lire deux dates annoncées valides et afficher la plus ancienne ou "dates égales". Comparer d'abord l'année, puis le mois, puis le jour.

Cas de test : deux dates d'années différentes, de même année et de même mois.



Exercice 4 – Fraction

Objectif : représenter exactement une quantité rationnelle.

Travail demandé : définir `Fraction`, lire deux fractions, calculer leur somme puis réduire le résultat avec le PGCD. Normaliser le signe en plaçant un dénominateur positif.

Erreur à traiter : refuser un dénominateur nul.



Exercice 5 – Complexe

Contexte : un signal est décrit par un nombre complexe $a + bi$.

Travail demandé : représenter un complexe par deux réels et programmer l'addition, la soustraction et la multiplication. Afficher chaque résultat sous la forme partie réelle et partie imaginaire.

Méthode : une fonction par opération, sans recopier les formules dans `main`.



Exercice 6 – Rectangle

Contexte : une interface graphique doit déplacer un rectangle sans modifier sa taille.

Travail demandé : définir un rectangle par son coin inférieur gauche, sa longueur et sa largeur. Écrire des fonctions pour le périmètre, l'aire et le déplacement de (dx, dy) .

Invariant : longueur et largeur restent positives après chaque opération.



Exercice 7 – Classement d'étudiants

Contexte professionnel : un service RH prépare une liste de candidats.

Données : au plus 20 étudiants, chacun avec nom, identifiant et note.

Travail demandé : lire le tableau, afficher les étudiants ayant une note au moins égale à 10 et calculer la moyenne de la classe. Aucun tri n'est nécessaire.

Sortie attendue : distinguer clairement la liste des admis et la moyenne.



Exercice 8 – Catalogue de produits

Contexte professionnel : un service e-commerce veut connaître la valeur d'une référence en stock.

Travail demandé : représenter un produit par sa référence, son libellé, son prix et sa quantité. Lire le catalogue, rechercher une référence et calculer $\text{prix} \times \text{quantité}$.

Cas absent : afficher un message explicite si la référence n'existe pas.



Exercice 9 – Adresse imbriquée

Objectif : éviter de mélanger les champs d'une personne et ceux de son adresse.

Travail demandé : définir Adresse, puis Personne contenant une adresse. Lire et afficher le nom, la rue, la ville et le code postal.

Notion visée : accéder à un champ imbriqué avec deux opérateurs point.



Exercice 10 – Agenda de rendez-vous

Contexte professionnel : un cabinet doit filtrer les rendez-vous d'un jour donné.

Travail demandé : représenter un rendez-vous par une date, une heure et une description. Lire les dix entrées, puis afficher uniquement celles qui correspondent à une date recherchée.

Attention : conserver l'ordre de saisie et afficher "aucun rendez-vous" si nécessaire.



Exercice 11 – Matrice encapsulée

Objectif : regrouper une matrice et sa dimension dans un seul objet.

Travail demandé : définir une structure contenant une matrice carrée d'entiers et n . Écrire les fonctions de lecture, d'affichage et de calcul de la trace, c'est-à-dire la somme de la diagonale principale.

Limite : vérifier que n reste dans la capacité du tableau.



Exercice 12 – Statistiques d'un tableau

Contexte : une série de mesures doit conserver ses données et ses statistiques.

Travail demandé : définir une structure contenant un tableau d'entiers, sa taille, son minimum et son maximum. Lire les valeurs puis remplir les champs calculés en un parcours.

Invariant : le minimum et le maximum doivent toujours provenir du tableau.



Exercice 13 – Mesure avec unité

Contexte : une station reçoit une température avec son unité.

Travail demandé : définir une mesure contenant une valeur et une unité parmi C, F et K. Vérifier les températures absolues valides, puis afficher les conversions vers les deux autres unités.

Rappel : $K = C + 273,15$ et $F = 1,8C + 32$.



Exercice 14 – Valeur polymorphe

Objectif : stocker plusieurs types dans une même collection.

Travail demandé : utiliser une union et une étiquette pour représenter un entier, un réel ou un caractère. Lire plusieurs valeurs, conserver leur type et afficher le champ correspondant à l'étiquette.

Règle : ne lire et n'afficher qu'un membre de l'union à la fois.



Exercice 15 – Registre dynamique

Contexte : le nombre d'étudiants n'est pas connu à l'avance.

Travail demandé : créer un tableau dynamique d'étudiants, commencer avec une petite capacité et la doubler lorsqu'elle est pleine. Ajouter jusqu'à une sentinelle, afficher le registre puis libérer le tableau.

Robustesse : utiliser un pointeur temporaire avec `realloc` et traiter un échec d'allocation.

2

PARTIE

Listes chaînées

NOTIONS

nœuds, liens, tête de liste, insertion, suppression, parcours et listes doublement chaînées

COMPÉTENCES

construire des collections dynamiques et préserver les invariants des liens

PROGRESSION

Application → approfondissement

Langage C | CHEATSHEET DE LA PARTIE 2

Listes chaînées

Notions de la partie, code C commenté et tests repères.

Notions essentielles

Nœud simple :

```
typedef struct Noeud { int valeur; struct Noeud
*suivant; } Noeud;
```

La variable `tete` désigne le premier nœud, ou vaut `NULL` si la liste est vide. Le dernier champ `suivant` vaut toujours `NULL`.

Parcourir :

```
for (Noeud *p = tete; p != NULL; p = p->suivant) { ...
}
```

Une liste n'est pas un tableau : l'accès au n -ième élément coûte un parcours. La liste vide est représentée par `tete == NULL`; son dernier nœud pointe vers `NULL`.

Invariant principal : chaque lien doit désigner un nœud valide ou `NULL`; aucun nœud ne doit être perdu lors d'une modification.

Liste doublement chaînée : chaque nœud contient une valeur et deux liens, `precedent` et `suivant`. La tête vérifie `tete->precedent == NULL` et la queue vérifie `queue->suivant == NULL`. Pour deux voisins `a` et `b`, les liens sont réciproques : `a->suivant == b` implique `b->precedent == a`.

Exemples C commentés

Créer et insérer :

```
n = malloc(sizeof *n);
```

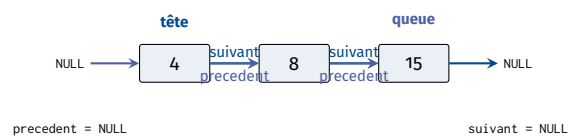
Après vérification, initialiser la valeur et le lien avant de publier le nœud. En tête : `n->suivant = tete`; `tete = n`; En queue, parcourir jusqu'au dernier nœud, ou conserver un pointeur `queue` pour éviter ce parcours.

Supprimer : mémoriser le nœud à libérer, relier les voisins, puis appeler `free`. Pour supprimer en tête : `ancien = tete`; `tete = tete->suivant`; `free(ancien)`; Toujours traiter la liste vide et la suppression du premier nœud.

Insertion en tête : `n->suivant = tete`; `tete = n`;

// Le nouveau nœud est relié avant de devenir la nouvelle tête.

Liste double : chaque opération met à jour simultanément `precedent` et `suivant`; la tête n'a pas de précédent et la queue n'a pas de suivant. Lors d'une suppression, reconnecter les deux voisins avant de libérer le nœud. Vérifier le parcours tête-queue puis queue-tête. Les cas à tester sont la liste vide, la tête, la queue et un nœud intérieur.



Partie 2 : Listes chaînées



Exercice 16 – Création d'une liste

Objectif : construire la structure mémoire correspondant au schéma.

Travail demandé : créer trois nœuds contenant 12, 7 et 19, relier leurs champs suivant, afficher les valeurs dans l'ordre puis libérer tous les nœuds.

Invariant : la dernière liaison vaut NULL.



Exercice 17 – Insertion en tête

Scénario : les valeurs arrivent les unes après les autres et la nouvelle valeur devient prioritaire.

Travail demandé : lire des entiers jusqu'à la sentinelle 0, insérer chaque valeur en tête, afficher la liste puis libérer sa mémoire.

Question : expliquer pourquoi l'ordre d'affichage est inverse de l'ordre de saisie.



Exercice 18 – Insertion en queue

Contexte : une file d'attente de demandes reçoit un nouvel élément sans modifier l'ordre existant.

Travail demandé : écrire une fonction d'insertion en queue et la tester sur une liste vide puis sur une liste non vide. Dans le schéma, ajouter 25 après 19.



Exercice 19 – Compter les nœuds

Travail demandé : écrire une fonction qui parcourt une liste et retourne son nombre de nœuds.

Tests obligatoires : liste vide, liste à un nœud et liste contenant des doublons.

Contrainte : ne modifier ni les valeurs ni les liens.



Exercice 20 – Somme d'une liste

Objectif : calculer une information sans détruire la collection.

Travail demandé : parcourir la liste, additionner ses valeurs et retourner la somme. Tester les valeurs positives, négatives et une liste vide.

Convention : la somme d'une liste vide vaut 0.



Exercice 21 – Recherche dans une liste

Travail demandé : rechercher la première occurrence d'une valeur et retourner l'adresse de son nœud, ou NULL si elle est absente.

Vérification : afficher la valeur trouvée sans déréférencer NULL, puis tester une valeur en tête, au milieu et absente.



Exercice 22 – Suppression en tête

Objectif : retirer le premier nœud tout en conservant la nouvelle tête.

Travail demandé : écrire une fonction qui traite la liste vide, mémorise l'ancien nœud, avance la tête, libère l'ancien nœud et retourne la nouvelle tête.



Exercice 23 – Suppression par valeur

Travail demandé : supprimer uniquement la première occurrence d'une valeur dans une liste simplement chaînée.

Cas à distinguer : valeur absente, valeur dans la tête, valeur dans un nœud intérieur.

Propriété : les nœuds non supprimés gardent leur ordre.



Exercice 24 – Suppression de toutes les occurrences

Objectif : nettoyer une liste qui peut commencer par plusieurs occurrences consécutives.

Travail demandé : supprimer toutes les occurrences d'une valeur, y compris en tête, puis afficher la liste restante.

Cas limites : toutes les valeurs à supprimer, aucune valeur à supprimer et liste vide.



Exercice 25 – Insertion ordonnée

Donnée : la liste est déjà triée dans l'ordre croissant.

Travail demandé : insérer 10 dans la liste du schéma sans tableau auxiliaire, en conservant

l'ordre et chaque liaison.

Cas à traiter : insertion avant la tête, entre deux nœuds et après la queue.



Exercice 26 – Copie indépendante

Objectif : produire une copie profonde, et non une seconde variable qui partage les mêmes nœuds.

Travail demandé : copier chaque valeur dans un nouveau nœud, modifier ensuite la copie et démontrer que l'originale est inchangée. Libérer les deux listes séparément.

Exercice 27 – Concaténation

Travail demandé : relier la queue de la première liste à la tête de la seconde en évitant de recopier les nœuds.

Contrat : préciser si la seconde tête est transférée et ne plus la libérer deux fois après la concaténation. Tester les deux listes vides.

Exercice 28 – Séparer les positifs

Objectif : distribuer les valeurs dans deux listes.

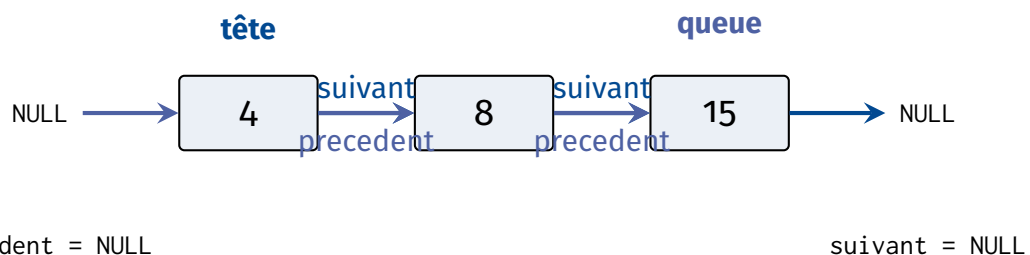
Travail demandé : parcourir une liste et construire une liste négative et une liste non négative. Préserver l'ordre d'apparition dans chacune.

Mémoire : libérer la liste source et les deux listes produites.

Exercice 29 – Liste doublement chaînée

Travail demandé : définir un nœud avec precedent et suivant, insérer en tête et en queue, puis parcourir dans les deux directions.

Invariant : la tête a un précédent nul et la queue a un suivant nul.

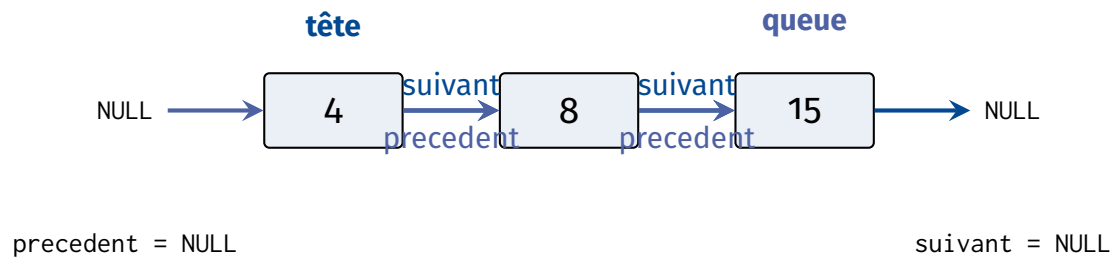




Exercice 30 – Liste doublement chaînée robuste

Objectif : supprimer un nœud intérieur sans casser le parcours inverse.

Travail demandé : supprimer la première occurrence d'une valeur et reconnecter les deux voisins. Vérifier les liens avant et arrière après chaque suppression, puis libérer toute la liste.



3 PARTIE

Piles et files

NOTIONS

pires, files, nœuds dynamiques, invariants LIFO et FIFO, files circulaires

COMPÉTENCES

implémenter des interfaces abstraites et choisir une représentation adaptée

PROGRESSION

Approfondissement → maîtrise

Langage C | CHEATSHEET DE LA PARTIE 3

Piles et files

Notions de la partie, code C commenté et tests repères.

Notions essentielles

Pile – LIFO : le dernier élément empilé est le premier dépilé. Les opérations sont empiler, depiler, sommet et estVide. Avec un tableau, un entier indique le nombre d'éléments ; avec une liste, la tête joue le rôle du sommet.

Contrat : préciser ce que retourne une opération impossible : échec d'empilement si la pile est pleine, ou indication d'absence si elle est vide. Une opération qui échoue ne doit pas modifier la structure.

File – FIFO : le premier élément enfilé est le premier défilé. Une file chaînée maintient tête et queue; dans une file vide, les deux valent NULL.

Exemples C commentés

File circulaire : dans un tableau de capacité N, avancer un indice par $(\text{indice} + 1) \% N$. Le compteur taille distingue les états vide et plein et évite de confondre les indices tête et queue.

Parenthèses : empiler les ouvrantes ; pour chaque fermante, vérifier que la pile n'est pas vide et que son sommet correspond, puis dépiler. À la fin, la pile doit être vide.

Expression postfixée : empiler les opérandes ; à chaque opérateur, dépiler d'abord le second opérande puis le premier, calculer et empiler le résultat. Refuser un jeton inconnu, un opérande manquant, une division par zéro ou une pile finale qui ne contient pas exactement une valeur.

Choix de représentation : tableau pour une capacité bornée et accès compact ; liste pour une taille variable, au prix d'une allocation par élément.

Pile tableau :

```
if (sommet == capacite) return 0;
pile[(*sommet)++] = valeur; // empiler
valeur = pile[-(*sommet)]; // dépiler
```

Partie 3 : Piles et files



Exercice 31 – Pile de caractères

Objectif : implémenter l'interface LIFO avec un tableau de capacité fixe.

Travail demandé : fournir `empiler`, `depiler`, `sommet` et `estVide`. Signaler une pile pleine ou vide sans accéder hors du tableau.

Tests : empiler trois caractères, dépiler puis vérifier l'ordre inverse.



Exercice 32 – Pile dynamique

Contexte : la taille des commandes n'est pas connue à l'avance.

Travail demandé : représenter la pile d'entiers par des nœuds alloués dynamiquement. Traiter un échec de `malloc`, afficher le sommet et libérer tous les nœuds.

Invariant : le sommet est toujours la tête de la liste.



Exercice 33 – Parenthèses équilibrées

Rappel : chaque parenthèse ouvrante doit être fermée par la bonne parenthèse, dans l'ordre inverse d'ouverture.

Travail demandé : utiliser une pile de caractères, ignorer les autres caractères et décider si l'expression est équilibrée.

Exemples : `(a+[b])` est valide ; `([ )]` ne l'est pas.



Exercice 34 – Inversion par pile

Scénario : afficher une ligne à l'envers sans utiliser une fonction de chaîne inversée.

Travail demandé : lire une ligne avec `fgets`, empiler ses caractères utiles puis les dépiler pour produire l'inversion.

Cas limite : ligne vide et ligne contenant des espaces.



Exercice 35 – Conversion décimal-binaire

Rappel : les restes successifs de divisions par 2 sont obtenus du bit de poids faible vers le bit de poids fort.

Travail demandé : convertir un entier naturel en binaire en empilant les restes puis en les dépilant. Traiter séparément le cas 0.



Exercice 36 – File d'entiers

Objectif : construire une file FIFO avec une liste chaînée.

Travail demandé : maintenir explicitement les pointeurs *tete* et *queue*, écrire l'enfilement et le défilement, puis tester les transitions vide/non vide.

Invariant : une file vide possède deux pointeurs nuls.



Exercice 37 – File de tâches

Contexte professionnel : un ordonnanceur de serveur reçoit des tâches avec un identifiant et une durée.

Travail demandé : enfiler plusieurs tâches, les traiter dans l'ordre d'arrivée et afficher la durée cumulée après chaque traitement.

Règle : aucune tâche ne doit être traitée avant celles arrivées précédemment.



Exercice 38 – File avec deux piles

Objectif : simuler FIFO à partir de deux structures LIFO.

Travail demandé : utiliser une pile d'entrée et une pile de sortie. Ne transférer les éléments que lorsque la pile de sortie est vide, puis expliquer pourquoi le plus ancien devient accessible en premier.



Exercice 39 – File circulaire

Travail demandé : implémenter une file dans un tableau de capacité fixe avec une tête, une queue et un compteur d'occupation.

Rappel : l'indice suivant est $(i + 1) \bmod N$. Tester les passages de la queue de tableau vers l'indice 0, ainsi que les états vide et plein.



Exercice 40 – Palindromes avec une pile et une file

Rappel : un palindrome se lit de la même manière dans les deux sens.

Travail demandé : lire un mot, empiler chaque caractère et l'enfiler simultanément, puis comparer les sorties de la pile et de la file.

Convention : travailler sur les caractères réellement saisis, sans ignorer les doublons.



Exercice 41 – Évaluation postfixée

Rappel : dans une expression postfixée, l'opérateur suit ses opérandes : $3\ 5\ +\ 2\ *$ représente $(3 + 5) * 2$.

Travail demandé : parcourir les jetons, empiler les entiers, dépiler le second puis le premier opérande à chaque opérateur et empiler le résultat.

Erreurs : opérande manquant, opérateur inconnu, division par zéro ou pile non réduite à une seule valeur.



Exercice 42 – Historique annuler-refaire

Contexte professionnel : un éditeur de code conserve les actions récentes de l'utilisateur.

Travail demandé : simuler deux piles : une pile des actions annulables et une pile des actions rétablissables. Après une nouvelle écriture, vider la pile refaire ; après annulation, déplacer l'action dans l'autre pile.



Exercice 43 – Priorité de tâches

Contexte : un centre de support possède trois files FIFO : haute, normale et basse.

Travail demandé : enfiler chaque tâche dans la file correspondant à sa priorité et toujours traiter la première file non vide.

Test : vérifier qu'une tâche basse attend tant qu'une tâche haute ou normale existe.



Exercice 44 – File de tickets

Contexte : un guichet distribue des tickets à des clients.

Travail demandé : créer une file dynamique de tickets contenant numéro, nom et catégorie. Servir le premier client, afficher les tickets restants et libérer les chaînes et les nœuds.



Exercice 45 – Bibliothèque pile-file

Objectif : produire une petite bibliothèque réutilisable.

Travail demandé : définir des fonctions génériques de pile et de file de pointeurs, puis les tester avec des chaînes allouées dynamiquement.

Contrat : préciser qui possède les pointeurs et qui les libère après le dépilement ou le défilement.

4 PARTIE

Arbres, hachage et graphes

NOTIONS

arbres binaires, arbres de recherche, tables de hachage, graphes et représentations

COMPÉTENCES

modéliser des relations hiérarchiques ou réseau et contrôler les pointeurs associés

PROGRESSION

Maîtrise → projet

Arbres, hachage et graphes

Notions de la partie, code C commenté et tests repères.

Notions essentielles

Arbre binaire :

```
struct Noeud { int cle; struct Noeud *gauche, *droit;
};
```

Un arbre vide vaut NULL; une feuille possède deux fils nuls. Les parcours sont préfixe (racine–gauche–droit), infixe (gauche–racine–droit) et postfixe (gauche–droit–racine). La récursivité suit naturellement la définition : traiter le nœud puis appliquer la même fonction à ses sous-arbres.

Arbre binaire de recherche : toute clé du sous-arbre gauche est plus petite et toute clé du sous-arbre droit est plus grande. Le parcours infixe fournit donc les clés dans l'ordre croissant.

Exemples C commentés

Insertion, recherche et suppression : comparer la clé avec le nœud courant puis descendre d'un seul côté. Lors d'une suppression, distinguer la feuille, le nœud à un fils et le nœud à deux fils ; dans ce dernier cas, remplacer la clé par son successeur ou son prédécesseur avant de supprimer ce nœud. Libérer récursivement les sous-arbres avant le nœud.

Hachage : calculer $\text{indice} = \text{cle} \% \text{capacite}$; une collision est placée dans la liste chaînée de la case. Pour une chaîne, conserver le lien précédent lors d'une suppression et tester les clés une à une.

Graphe : une matrice d'adjacence consomme $O(n^2)$ cases ; les listes d'adjacence conviennent mieux à un graphe peu dense. Pour un graphe non orienté, la matrice est symétrique et chaque arête contribue deux fois à la somme des degrés.

BFS : initialiser *visite*, marquer un sommet au moment de son enfilement, puis traiter la file jusqu'à ce qu'elle soit vide.

Recherche ABR :

```
if (cle == n->cle) return n;
if (cle < n->cle) return chercher(n->gauche, cle);
return chercher(n->droit, cle); // un seul sous-arbre
```

Partie 4 : Arbres, hachage et graphes

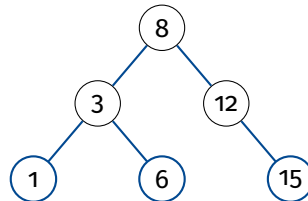


Exercice 46 – Arbre binaire

Objectif : représenter une hiérarchie de nœuds.

Travail demandé : définir un nœud contenant une valeur entière et deux pointeurs fils, construire l'arbre du schéma puis afficher son parcours préfixe.

Rappel : un pointeur fils nul représente un sous-arbre vide.



Exercice 47 – Parcours d'arbre

Travail demandé : construire un arbre binaire connu puis écrire les parcours préfixe, infixe et postfixe. Afficher un parcours par ligne.

À expliquer : indiquer à quel moment la racine est visitée dans chacun des trois parcours et traiter l'arbre vide.



Exercice 48 – Compter les nœuds d'un arbre

Objectif : extraire trois mesures récursives.

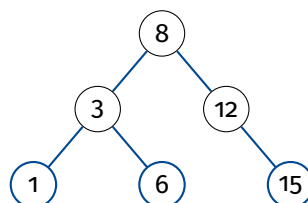
Travail demandé : retourner le nombre de nœuds, le nombre de feuilles et la hauteur. Utiliser la définition hauteur(arbre vide) = 0 et préciser la convention retenue pour une feuille.



Exercice 49 – Arbre de recherche

Rappel : dans un ABR, les clés à gauche sont plus petites et celles à droite plus grandes.

Travail demandé : insérer 8, 3, 12, 1, 6 et 15, puis afficher le parcours infixe. Il doit produire une séquence croissante ; comparer cette séquence avec le schéma.





Exercice 50 – Recherche dans un arbre

Travail demandé : rechercher une clé dans un arbre binaire de recherche avec une fonction récursive puis une fonction itérative. Retourner un booléen et compter, si possible, le nombre de comparaisons.

Cas de test : clé à la racine, clé dans une feuille et clé absente.



Exercice 51 – Suppression dans un arbre

Objectif : préserver la propriété d'ABR après suppression.

Travail demandé : traiter séparément une feuille, un nœud avec un seul fils et un nœud avec deux fils en le remplaçant par son successeur ou son prédécesseur. Afficher l'infixe avant et après, puis libérer l'arbre.



Exercice 52 – Expression arithmétique en arbre

Rappel : les feuilles représentent des nombres et les nœuds internes des opérateurs.

Travail demandé : construire l'arbre de $(3+5)*2$, puis l'évaluer récursivement. Vérifier que les sous-arbres sont évalués avant l'opérateur qui les combine.



Exercice 53 – Table de hachage simple

Contexte professionnel : un service d'authentification veut retrouver rapidement des identifiants entiers.

Travail demandé : créer une table de capacité fixe, calculer l'indice par modulo et résoudre les collisions par une liste chaînée dans chaque case. Implémenter insertion et recherche.

Test : choisir des clés produisant volontairement la même case.



Exercice 54 – Dictionnaire de mots

Objectif : compter les occurrences sans tableau de taille égale au nombre de mots possibles.

Travail demandé : associer chaque mot à un compteur dans une table de hachage chaînée. Lorsqu'un mot existe, incrémenter son compteur ; sinon créer une entrée. Afficher les mots et leurs fréquences.



Exercice 55 – Suppression avec hachage

Travail demandé : ajouter la suppression d'une clé dans une case chaînée. Tester la suppression du premier nœud, d'un nœud intérieur, du dernier nœud et d'une clé absente.

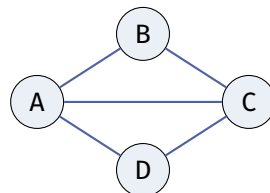
Invariant : les autres entrées restent accessibles après chaque suppression.



Exercice 56 – Graphe par matrice

Donnée : le graphe non orienté du schéma comporte les sommets A, B, C et D.

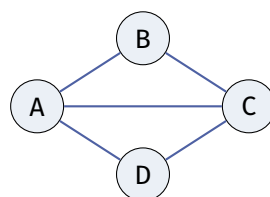
Travail demandé : construire sa matrice d'adjacence, ajouter les arêtes et afficher les voisins de chaque sommet. Pour une arête non orientée, les deux cases symétriques doivent être égales.



Exercice 57 – Graphe par listes

Contexte : le graphe représente un réseau de dépendances entre services. Représenter le même graphe par une liste d'adjacence par sommet. Afficher chaque voisin puis comparer le nombre de cases utilisées avec une matrice complète.

Conclusion attendue : expliquer pourquoi les listes sont adaptées aux graphes peu denses.



Exercice 58 – Degrés d'un graphe

Rappel : le degré d'un sommet non orienté est le nombre de ses voisins.

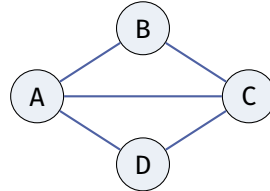
Travail demandé : parcourir chaque ligne de la matrice, calculer le degré correspondant et signaler les sommets isolés. Vérifier que la somme des degrés vaut deux fois le nombre d'arêtes.



Exercice 59 – File de parcours

Objectif : visiter un graphe en largeur, niveau par niveau.

Travail demandé : utiliser les listes d'adjacence et une file pour partir de A. Marquer un sommet dès son enfilement afin de ne pas l'ajouter plusieurs fois, puis afficher l'ordre de visite.



Exercice 60 – Projet de carnet de relations

Mini-projet professionnel : construire un annuaire interne de personnes et de relations entre équipes.

Fonctionnalités : ajouter une personne dans une table de hachage, créer une relation dans les listes de voisins, rechercher une personne et afficher ses voisins.

Robustesse : refuser un doublon, une relation vers une personne inconnue et toute allocation échouée ; libérer toutes les structures à la fin.