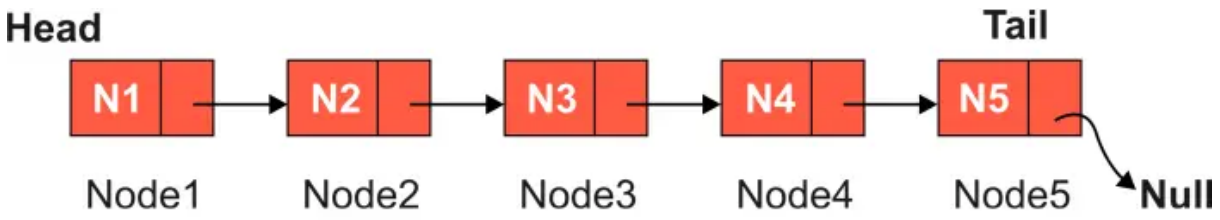
REINVENT ENGINEERING

CENTRALE NANTES

RAPPORT SMP - TP4

Liste chaînée et tableau



Date: 5 Décembre 2025

Auteur: Nabil Khouani - Reda Aouad

Introduction

Ce TP a pour but de créer et de manipuler des structures de données pour gérer un répertoire de personnes. Nous allons implémenter des fonctions permettant d’ajouter, rechercher, afficher et supprimer des informations sur les personnes, comme leur nom, prénom et numéro de téléphone. L'objectif est de comprendre comment ces structures fonctionnent et de comparer leurs performances.

Structure de données

Le fichier type_def.h contient les structures de données que nous allons utiliser.

1. Structure `personne`

La structure `personne` permet de représenter les informations relatives à une personne, à savoir :

- **nom** (type `string`) : Le nom de la personne.
- **prénom** (type `string`) : Le prénom de la personne.
- **numero** (type `string`) : Le numéro de téléphone de la personne.

Cette structure sert à stocker les données essentielles d'une personne dans le répertoire.

2. Structure `elementListe`

La structure `elementListe` représente un élément d'une **liste simplement chaînée**. Chaque élément contient :

- **one** (type `personne`) : Les informations d'une personne (nom, prénom, numéro de téléphone).
- **suivant** (type `elementListe*`) : Un pointeur vers l'élément suivant dans la liste. Si c'est le dernier élément, ce pointeur est nul.

Cette structure permet de relier les éléments entre eux pour former une liste où chaque élément contient une personne et un lien vers l'élément suivant.

3. Structure tableauTrie

La structure `tableauTrie` représente le conteneur principal des données sous forme de tableau statique. Elle contient :

- **tab (tableau de type personne)** : Un tableau de taille fixe (`MAX_TAILLE`) qui stocke directement les informations des personnes.
- **nbElements (type int)** : Un entier qui indique le nombre réel de personnes actuellement présentes dans le tableau.

Cette structure permet de regrouper les données et le compteur d'éléments. Contrairement à la liste, ici toutes les personnes sont stockées les unes à la suite des autres en mémoire, jusqu'à la limite de `MAX_TAILLE`.

Utilitaires

Le fichier utilitaires.cpp contient les fonctions surtout les fonctions sur la création de personnes, leur comparaisons etc. elle contient aussi la fonction pour la création de notre élement Liste.

Fonction `genererPersonne`

Cette fonction génère une structure `personne` contenant des données aléatoires pour une personne. Elle crée un nom, un prénom et un numéro de téléphone au hasard en utilisant des fichiers texte externes. La structure `personne` est ensuite remplie avec ces valeurs et renvoyée.

- **Entrée :** Aucune (fonction autonome).
- **Sortie :** Une structure `personne` avec des données générées aléatoirement.

Fonction `creerElementListe`

Cette fonction crée un nouveau maillon (ou noeud) pour une liste simplement chaînée. Elle alloue dynamiquement un espace mémoire pour un `elementListe`, y place la donnée `personne` passée en argument, et initialise le pointeur `suivant` à `nullptr` (car c'est le dernier élément de la liste au moment de sa création).

- **Entrée :** Un objet de type `personne`.
- **Sortie :** Un pointeur vers un `elementListe` contenant la `personne` donnée et dont le pointeur `suivant` est nul.

Fonction `affichagePersonne`

Cette fonction affiche les informations d'une personne dans la console. Elle prend en argument une structure `personne` et imprime son nom, prénom et numéro de téléphone.

- **Entrée :** Une structure `personne`.
- **Sortie :** Affichage des informations de la personne dans la console.

```
---- Affichage personne ----
Nom: Gougeon Prenom: marek tel: 0170112452
Nom: Pearson Prenom: vladmra tel: 0215895045
Nom: Décarie Prenom: bahj tel: 0790948387
Nom: Filiatrault Prenom: lela tel: 0683268607
```

La fonction affiche bien les personnes que l'on crée.

Fonction `egalitePersonne`

Cette fonction vérifie si deux personnes sont strictement identiques. Elle compare le nom, le prénom et le numéro de téléphone des deux personnes. Si toutes ces valeurs sont égales, la fonction retourne `true`, sinon elle retourne `false`.

- **Entrée :** Deux structures `personne` à comparer.
- **Sortie :** `true` si les deux personnes sont identiques, `false` sinon.

```
cout << egalitePersonne(firstone, secondone) << endl; // Test de la fonction d'égalité
```

En testant cette fonction me retourne bien false ou 0 car Gougeon et Pearson ne sont pas les mêmes noms.

Fonction `comparerPersonne`

Cette fonction compare deux personnes pour déterminer laquelle vient avant l'autre dans un ordre alphabétique. L'ordre est défini ainsi :

1. Comparaison des noms.
2. Si les noms sont égaux, comparaison des prénoms.
3. Si les prénoms sont aussi égaux, comparaison des numéros de téléphone.

La fonction retourne `true` si la première personne (`one`) doit venir avant la seconde (`two`), sinon elle retourne `false`.

- **Entrée :** Deux structures `personne` à comparer.
- **Sortie :** `true` si la première personne vient avant la seconde, `false` sinon.

```
cout << comparerPersonne(firstone, secondone) << endl; // Test de la fonction de comparaison
```

Cette fonction retourne true ou 1 car la première personne (Gougeon) est avant la deuxième personne (Pearson).

Répertoires

Fonction `ajouter`

Cette fonction permet d'ajouter une personne dans une liste simplement chaînée tout en maintenant l'ordre alphabétique. Elle parcourt la liste pour trouver la bonne position d'insertion, et insère la personne au début, au milieu ou à la fin de la liste.

- **Entrée :** Une structure `personne` à ajouter et un pointeur vers le début de la liste.

- **Sortie :** Un pointeur vers le début de la liste après l'insertion de la personne.

Pour la tester nous affichons le nom lié a one lié a ma liste myList. Le std::cout montre bien les noms prenom et numero des personnes. Nous pouvons aussi la tester avec l'affichage des listes.

Fonction `ajouterTab`

Cette fonction ajoute une personne dans un tableau trié tout en maintenant l'ordre alphabétique. Elle cherche la position où insérer la personne, décale les éléments suivants vers la droite, puis insère la nouvelle personne à la bonne position.

- **Entrée :** Une structure `personne` à ajouter et un pointeur vers le tableau trié.

- **Sortie :** Le tableau trié mis à jour avec la nouvelle personne insérée.

Pour la tester nous affichons le nom prenom et numero de tableauTrie.tab[0]. Le std::cout montre bien les noms prenom et numero des personnes. Nous pouvons aussi la tester avec l'affichage des tableauTrié.

Fonction `afficher`

Cette fonction affiche le contenu d'une liste simplement chaînée en parcourant chaque élément et en affichant les informations de la personne contenue dans chaque maillon.

- **Entrée :** Un pointeur vers le début de la liste.

- **Sortie :** Affichage des informations de chaque personne dans la console.

```
Nom: Décarie Prenom: bahj tel: 0790948387
Nom: Gougeon Prenom: marek tel: 0170112452
Nom: Pearson Prenom: vladmra tel: 0215895045
```

Fonction `afficherTab`

Cette fonction affiche le contenu d'un tableau trié en parcourant chaque case remplie et en affichant les informations de chaque personne.

- **Entrée :** Un pointeur vers le tableau trié.

- **Sortie :** Affichage des informations de chaque personne dans le tableau dans la console.

```
DEBUG: Nombre d'elements trouves : 4
Nom: Décarie Prenom: bahj tel: 0790948387
Nom: Filiatrault Prenom: lela tel: 0683268607
Nom: Gougeon Prenom: marek tel: 0170112452
Nom: Pearson Prenom: vladmra tel: 0215895045
```

Fonction `rechercher`

Cette fonction effectue une recherche linéaire dans la liste simplement chaînée pour trouver une personne donnée. Elle parcourt la liste maillon par maillon jusqu'à trouver la personne recherchée.

- **Entrée :** Une structure `personne` à rechercher et un pointeur vers le début de la liste.

- **Sortie :** L'index de la personne dans la liste si trouvée, ou `-1` si non trouvée.

Le prénom pearson est dans notre tableau bien a la 3 eme position donc à l'indice 2.

```
---- indice du prenom 2 ----
2
```

Fonction `rechercherTab`

Cette fonction effectue une recherche linéaire dans le tableau trié pour trouver une personne donnée. Elle compare chaque élément du tableau avec la personne recherchée jusqu'à la trouver.

- **Entrée :** Une structure `personne` à rechercher et un pointeur vers le tableau trié.

- **Sortie :** L'index de la personne dans le tableau si trouvée, ou `-1` si non trouvée.

Dans cette fonction nous cherchons le 4eme nom, Filiatrault. Il nous dis que il est a l'indice 1 donc le 2 eme nom dans l'ordre alphabétique.

```
1
```

Fonction `supprimer`

Cette fonction supprime un élément de la liste chaînée. Elle parcourt la liste pour trouver l'élément à supprimer, puis le retire en ajustant les pointeurs des éléments voisins.

- **Entrée :** Une structure `personne` à supprimer et un pointeur vers le début de la liste.

- **Sortie :** Un pointeur vers le début de la liste après la suppression de l'élément.

On voit que l'on supprime bien le nom 2 Pearson.

```
---- affiche la liste supprimer el 2 ---
Nom: Décarie Prenom: bahj tel: 0790948387
Nom: Gougeon Prenom: marek tel: 0170112452
```

Fonction `supprimerTab`

Cette fonction supprime un élément du tableau trié. Elle cherche l'index de l'élément à supprimer, décale tous les éléments suivants vers la gauche pour combler le vide, puis réduit la taille du tableau.

- **Entrée :** Une structure `personne` à supprimer et un pointeur vers le tableau trié.

- **Sortie :** Le tableau trié mis à jour après la suppression de l'élément.

Dans ce test nous cherchons a supprimé le nom 4 filiatrault.

```
---- Tableau supprimé ----
DEBUG: Nombre d'elements trouves : 3
Nom: Décarie Prenom: bahj tel: 0790948387
Nom: Gougeon Prenom: marek tel: 0170112452
Nom: Pearson Prenom: vladmra tel: 0215895045
```

Test et temps d’éxécution

Pour valider nos choix d'implémentation, nous avons comparé les performances de la **Liste Chaînée** et du **Tableau Trié** (statique). Les mesures ont été effectuées à l'aide de la fonction `clock()` de la librairie `<ctime>`, sur un jeu de données de **1000 personnes**.

Voici les temps d'exécution relevés lors des tests :

```
Generation des 1000 personnes...

TESTS LISTE CHAINEE
Temps AJOUT (1000) : 0.015335 s
Temps AFFICHAGE : 1e-06 s (desactive)
Temps RECHERCHE (100) : 0.003908 s
Temps SUPPRESSION (100) : 0.004055 s

TESTS TABLEAU TRIE
Temps AJOUT (1000) : 0.028815 s
Temps AFFICHAGE : 1e-06 s (desactive)
Temps RECHERCHE (100) : 0.003592 s
Temps SUPPRESSION (100) : 0.005723 s
```

1. Création du répertoire (Ajout)

L'ajout de 1000 personnes est nettement plus rapide avec la **liste chaînée** (0.015s) qu'avec le tableau trié (0.028s).

- **Explication :** Bien que la liste nécessite une allocation dynamique (`new`) pour chaque élément, l'insertion consiste simplement à modifier des pointeurs. À l'inverse, pour maintenir le **tableau trié**, chaque

insertion nécessite de décaler physiquement tous les éléments suivants en mémoire (décalage vers la droite). Sur 1000 insertions, ces opérations de copie mémoire (`O(N)`) deviennent coûteuses.

2. Affichage

Le temps d'affichage est négligeable (`1e-06`) car la fonction d'affichage console (`cout`) a été désactivée pour ne pas fausser les mesures. L'affichage est une opération coûteuse en Entrée/Sortie (I/O) qui masquerait la performance réelle des algorithmes de parcours.

3. Recherche

Pour la recherche de 100 personnes, le **tableau** (0.0035s) est légèrement plus performant que la liste (0.0039s).

- **Explication :** Bien que les deux algorithmes effectuent une recherche linéaire (complexité `O(N)`), le tableau bénéficie de la **contiguïté mémoire**. Le processeur peut précharger les données (cache CPU), rendant le parcours séquentiel d'un tableau plus rapide que le saut de pointeur en pointeur dispersé dans le tas (heap) de la liste chaînée.

4. Suppression

La suppression est plus efficace sur la **liste chaînée** (0.0040s) que sur le tableau (0.0057s).

- **Explication :** La suppression implique deux étapes : trouver l'élément, puis l'enlever.

    - Pour la **liste** : Une fois l'élément trouvé, la suppression est instantanée (`O(1)`), il suffit de rediriger le pointeur précédent vers le suivant et de libérer la mémoire.
    - Pour le **tableau** : Comme pour l'ajout, supprimer un élément au milieu du tableau oblige à décaler tous les éléments suivants vers la gauche pour combler le vide. Ce coût de déplacement pénalise la performance globale.

5. Différences

- **Tableau :** Très performant pour la lecture et la stabilité mémoire, mais souffre de lenteurs dès qu'il faut modifier sa structure (ajout/suppression) à cause des décalages. Il a aussi une limite de taille fixe (`MAX_TAILLE`).
- **Liste :** Plus flexible et plus rapide pour les modifications fréquentes, mais consomme plus de mémoire (stockage des pointeurs) et est légèrement plus lente à parcourir.

Conclusion

En réalisant ce TP, nous avons appris à utiliser des structures de données pour gérer des informations de manière efficace. Les tests réalisés ont permis de comparer les performances des différentes opérations (ajout, recherche, suppression) et de mieux comprendre l’impact des choix de structures de données sur ces performances.