Université de Yaoundé I
Faculté des Sciences

University of Yaounde I
Faculty of Science

Département d'Informatique

INF231, Rappel Listes chainées, Files et Piles, 26 Septembre 2024

Dr MESSI NGUELÉ Thomas, Senior Lecturer

1 Listes Chainées.

1.1 Pointeurs

Un pointeur est une variable permettant de contenir une adresse mémoire. En algorithmique, un pointeur d'entier est défini par : **var** P^ entier ;

À la création, le pointeur P contient l'adresse NULL. On distingue deux fonctions pour la gestion de la mémoire (pour un pointeur P) :

- **nouveau**(p) : reserve un espace mémoire. On alloue un espace mémoire pour un élément sur lequel pointe P.
- **liberer**(p) : libère l'espace mémoire qui était occupé par l'élément sur lequel pointe P.

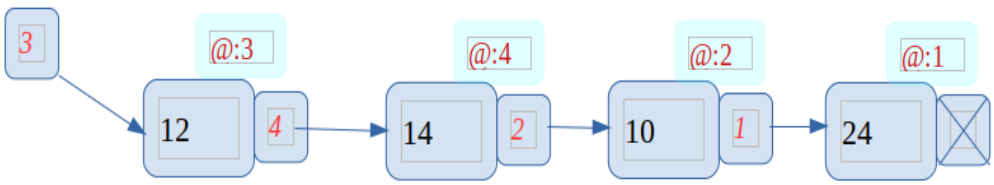
Exemple en algorithmique :

```
var P^ entier;
var i entier;
i <-- 4;
P <-- @i;
ecrire(P^); //affichera la valeur de i qui est 4;
```

1.2 Listes Chainées :

Une liste chainée est une succession de cellules dont la dernière pointe sur une adresse NULL.

Exemple de représentation :



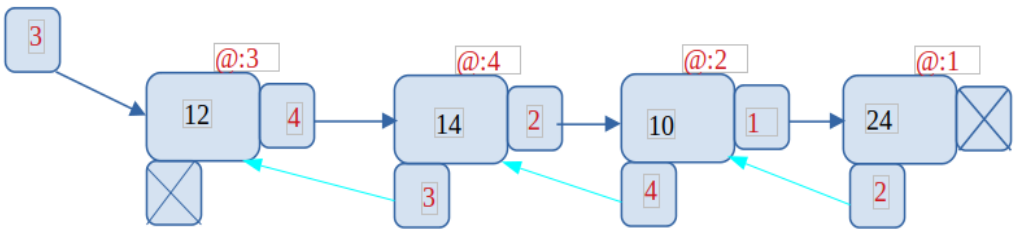
Syntaxe en algorithmique :

```
type liste = ^cellule;
   cellule = enregistrement
               info    : type_info; // il s'agit du type de la donnée stoquée
               suivant : liste;
             fin

var tete, P: liste;
```

1.3 Listes Doublement Chainées :

Il s'agit des listes pouvant être parcourues dans les deux sens.
Exemple de représentation :



Syntaxe en algorithmique :

```
type d_liste = ^cellule;
   cellule = enregistrement
               precedent : d_liste;
               info      : type_info; // il s'agit du type de la donnée stoquée
               suivant   : d_liste;
             fin

var d_tete, d_P: d_liste;
```

1.4 Exercices :

Écrire les algorithmes correspondants aux exercices ci-après (à rendre dans un document pdf), puis un programme C (main.c, fonction.c, fonction.h).

1. **Création d'une liste (par ajout d'éléments)**
   (a) Écrire un algorithme qui crée une liste avec deux entiers.
   (b) Écrire un algorithme qui crée une liste de n éléments entiers entrés par l'utilisateur.
   (c) Écrire un algorithme qui crée une liste d'éléments entrés entiers par l'utilisateur, la liste est achevée lorsque l'utilisateur saisit -1.
2. **Recherche d'éléments dans une liste chainée**
   (a) Écrire une procédure qui recherche une valeur dans une liste chainée et affiche un message à l'utilisateur disant si la valeur est présente ou pas.
   (b) Écrire une procédure qui recherche la dernière occurence d'une valeur dans une liste chainée et affiche un message à l'utilisateur disant si la valeur est présente ou pas, et si elle est présente la position de la dernière occurence.
3. **Suppression d'éléments dans une liste chainée**
   (a) Écrire une procédure qui supprime le premier élément d'une liste chainée.
   (b) Écrire une procédure qui supprime la première occurrence d'une valeur passée en paramètre.
   (c) Écrire une procédure qui supprime toutes les occurrences d'une valeur dans une liste chainée.
4. **Listes doublement chainées.** Écrire une procédure permettant d'afficher une liste doublement chainée :
   (a) dans l'ordre d'enrégistrement,
   (b) dans l'ordre inverse de l'enrégistrement.

2 Piles.

2.1 Définition.

Une pile est une liste ou une collection d'élements dans laquelle on ne peut introduire ou enlever un élément qu'à une extrémité appelée **tête de pile** ou **sommet de pile**.

Le dernier élément inséré sera le premier élément à être supprimé ou retiré de la pile : premier arrivé, dernier servi, en anglais Last In, First Out (**LIFO**). La figure 2.1 illustre une pile d'entiers.

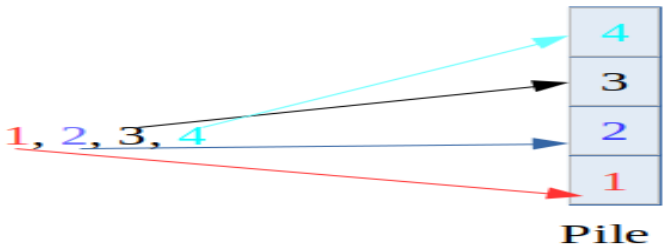


FIGURE 1 – Illustration d'une pile d'entiers

Exemples Courants :
- Pile de livres,
- Pile d'assiètes,
- Pile de cartes
- Autres exemples (donnés par les étudiants).

2.2 Opérations sur une pile.

Une pile est caractérisée par les opérations suivantes :
- **empiler**(P : Pile, elt :type_elt) qui permet d'ajouter un élément en tête de la pile.
- **depiler**(P : Pile) :type_elt qui permet d'enlever l'élément en tête de pile.
- **initialiser**(P :Pile) qui permet d'initialiser la pile (la péprarer à recevoir les données).
- **vider**(P :Pile) qui permet de vider le contenu de la pile.
- **sommet**(P :Pile) :type_elt qui retourne le sommet de la pile.

2.3 Exercices d'application :

Une pile peut être réalisée en stoquant ses éléments dans un tableau ou dans une liste chainée. Écrire en algorithmique puis un programme C permetant de :

1. Réaliser une pile à l'aide d'un tableau.
2. Réaliser une pile à l'aide d'une liste chainée.
3. Qu'est ce qui change fondamentalement d'après vous ?

3 Files :

3.1 Définition :

Une file est une structure de données correspondant à une liste d'éléments ou une collection d'éléments dans laquelle on insère des nouveaux éléments à la fin (queue) et on enlève les éléments au début (tête de file). En d'autres termes, dans une file, le premier élément inséré est aussi le premier élément retiré ; en anglais First In, First Out (**FIFO**). La figure 3.1 illustre une file d'entiers.

Exemples Courants :
- Une file d'attente des clients d'une banque.
- Un autre exemple donné par les étudiants.

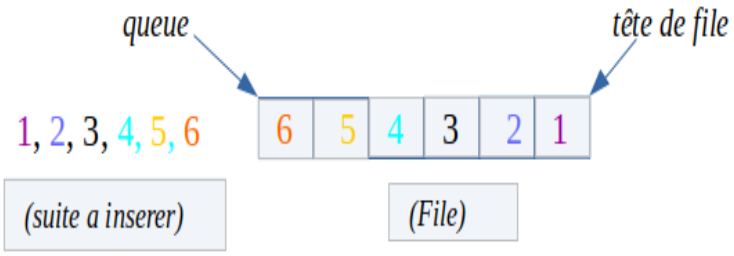


FIGURE 2 – Illustration d'une file d'entiers

3.2 Opération sur une file

Une file est caractérisée par les opérations suivantes :
— **enfiler**(F : File, elt :type_elt) qui permet d'ajouter un élément en queue de la pile.
— **defiler**(F : File) :type_elt qui permet d'enlever l'élément en tête de file.
— **initialiser**(F :File) qui permet d'initialiser la file (la péprarer à recevoir les données).
— **vider**(F :File) qui permet de vider le contenu de la file.

3.3 Exercices d'application :

Tout comme une pile, une file peut être réalisée en stoquant ses éléments dans un tableau ou dans une liste chainée. Écrire en algorithmique puis un programme C permettant de :

1. Réaliser une file à l'aide d'un tableau.
2. Réaliser une file à l'aide d'une liste chainée.
3. Qu'est ce qui change fondamentalement d'après vous ?

4 Éléments de correction :

4.1 Liste chainée avec deux éléments :

```
type liste = ^cellule;
   cellule = enregistrement
               info    : entier;
               suivant : liste;
             fin

 Algorithme: liste_de_2_elements
    var tete, P: liste;
 Debut
    tete <-- null;

    nouveau(P);
    ecrire("Entrer la première valeur de la liste");
    lire(P^.info);
    P^.suivant <-- tete;
          tete <-- P;

    nouveau(P);
    ecrire("Entrer la deuxième valeur de la liste");
    lire(P^.info);
    P^.suivant <-- tete;
          tete <-- P;
 Fin
```

4.2 Liste chainée avec n éléments :

```
type liste = ^cellule;
   cellule = enregistrement
               info    : entier;
               suivant : liste;
             fin

 Algorithme: liste_de_n_elements
    var tete, P: liste;
        i, n: entier;
 Debut
    ecrire("Entrer le nombre d'éléments de la liste");
    lire(n);
    tete <-- null;

    Pour i allant de 1 à n faire
        nouveau(P);
        ecrire("Entrer l'élément ", i);
        lire(P^.info);
        P^.suivant <-- tete;
              tete <-- P;
    FinPour
 Fin
```

4.3 Recherche d'une valeur dans une liste chainée :

```
type liste = ^cellule;
   cellule = enregistrement
               info    : entier;
               suivant : liste;
             fin

 Fonction recherche(tete: liste, val: entier):booleen
    var P: liste; trouve:booleen;
 Debut
    P    <-- tete;
    Tantque ((P<>null) et (P^.info<> val)) faire
        P    <-- P^.suivant;
    FinTantque
    Si (P = null) alors
        trouve <-- faux;
    Sinon
        trouve <-- vrai;
    FinSi
    retourner trouve;
 Fin
```

4.4 Affichage d'éléments dans une liste doublement chainée :

```
type d_liste = ^cellule;
   cellule = enregistrement
               precedent : d_liste;
```

```
              info      : entier;
              suivant   : d_liste;
            fin

var d_tete, d_P: d_liste;

 Fonction affiche_liste(tete: liste):booleen
    var P, Q: liste;
 Debut
    P    <-- tete;
    Tantque ((P<>null) faire
        ecrire(P^.info, " ");
        Q   <-- P;
        P    <-- P^.suivant;
    FinTantque
    Tantque ((Q<>null) faire
        ecrire(Q^.info, " ");
        Q  <-- Q^.precedent;
    FinTantque

 Fin
```

4.5 Suppression du premier élément d'une liste :

```
type liste = ^cellule;
   cellule = enregistrement
              info    : entier;
              suivant : liste;
            fin

 Procedure supprime_tete(var tete: liste)
    var P: liste;
 Debut
    Si (tete<>null) alors
        P    <-- tete;
        tete <-- P^.suivant;
        liberer(P);
    Sinon
        ecrire("La liste est vide");
    FinSi
 Fin
```

4.6 Implémentation d'une file d'entiers à l'aide d'un tableau :

```
File = enregistrement
           tableau Tab[N]: entier;
           tete : entier;
           queue : entier;
           longueur_file: entier;
       fin
 Fonction enfiler(f: File, a: entier):File
 Debut
    Si (file_pleine(f)<>vrai) alors
```

```
        f.Tab[f.queue] <-- a;
        f.queue <-- f.queue + 1;
    Sinon
        ecrire("La file est pleine");
    FinSi
    retourner f;
Fin

Fonction defiler( var f: File):entier
var val_retour: entier
Debut
    Si (file_vide(f)<>vrai) alors
        val_retour <-- f.Tab[f.tete];
        f.tete <-- f.tete + 1;
    Sinon
        ecrire("La file est vide");
        val_retour <-- -1;
    FinSi
    retourner val_retour;
Fin

Fonction file_pleine(f: File):booleen
var val_retour: booleen
Debut
    Si (f.queue = (f.longueur_file +1)) alors
        val_retour <-- vrai;
    Sinon
        val_retour <-- faux;;
    FinSi
    retourner val_retour;
Fin

Fonction file_vide(f: File):booleen
var val_retour: booleen
Debut
    Si (f.queue = f.tete) alors
        val_retour <-- vrai;
    Sinon
        val_retour <-- faux;
    FinSi
    retourner val_retour;
Fin

Fonction init_file(f: File, N: entier):file
var val_retour: booleen
Debut
    f.longueur_file <-- N;
    f.tete <-- 1;
    f.queue <-- 1;
    retourner f;
Fin
```