Support de cours Structure de données

Dr. ASSIE Brou Ida,UFHB.

Table des matières

Chapitre 1 : Généralités sur les structures de données

La programmation est utilisée pour résoudre les problèmes qui amènent souvent à analyser, à lire ou à mémoriser des données. Pour cela un algorithme à travers des petites opérations, est utilisé pour effectuer des actions simples. Pour réduire le temps de calcul (c'est-à-dire, le temps pour rendre un résultat) et l'espace en mémoire (le nombre d'informations à stocker pour faire les calculs), il est préférable de les organiser, selon le but à atteindre, dans un objet de type **structure de données**.

Ce chapitre fera l'objet de la généralité sur les structures de données.

I- Intérêt de l'utilisation des structures de données

Problème : les numéros de téléphone peuvent être présentés par nom, par profession (comme les Pages jaunes), par numéro téléphonique (comme les annuaires destinés au télémarketing), par rue ou une combinaison quelconque de ces classements. À chaque usage correspondra une structure d'annuaire appropriée.

En organisant d'une certaine manière les données, on permet un traitement automatique de ces dernières plus efficace et plus rapide.

II- Définition d'une structure de données

Une structure de données est un ensemble organisé de données reliées logiquement pour faciliter leur traitement. C'est un moyen particulier d'organiser les données dans un ordinateur afin qu'elles puissent être utilisées efficacement.

Par exemple, nous pouvons stocker une liste d'éléments ayant le même type de données à l'aide de la structure de données « Tableau ».

1	2	3	4	5	6	7	8	9	10

III- Différents types de structures de données

Il existe différents types de structures de données pour des données différentes ou répondant à des contraintes algorithmiques différentes :

- **Tableaux** : pour stocker des éléments homogènes à des emplacements contigus. La taille d'un tableau doit être fournie avant de stocker des données.
- **Enregistrements** : sont des structures de données dont les éléments peuvent être de type différent. Contrairement aux tableaux qui sont des structures de données dont tous les éléments sont de même type.
- **Fichiers** : sont un ensemble d'enregistrements qui sont tous de même type. Un fichier peut être conservé sur un support externe à la mémoire centrale.
- **Piles et files** : Les piles et les files sont des ensembles dynamiques pour lesquels l'élément à supprimer *via* l'opération SUPPRIMER est défini par la nature intrinsèque de l'ensemble.
- **Listes chainées, Arbres binaires, Graphes, Tas, Hachage, ...**

Chapitre 2 : Enregistrements

Les tableaux sont les structures de données les plus connus. Tous les éléments sont homogènes et dans certaines applications, ces types de données ne répondent pas à nos besoins car nous voulons regrouper des données de différents types différents. Par exemple, le nom et l'âge d'une centaine de personnes. Il est donc utile de pouvoir rassembler des éléments de type différents. On parle alors d'enregistrement ou de structure.

I- Définition d'un enregistrement

Un enregistrement est une structure de données qui permet de rassembler sous un même nom des données de types différents (structure hétérogène).

L'avantage d'un enregistrement est de regrouper des informations sémantiquement liées pour éviter de multiplier les variables et bien faire apparaitre la logique du traitement.

Exemple : Fiche d'un étudiant de UL

- Nom : chaine de caractères
- Age : entier
- Sexe : typé énumératif (masculin ou féminin)
- Matricule : Entier

II- Déclaration d'un enregistrement

La syntaxe générale de la déclaration d'un enregistrement :

Type

nom_Enregistrement = **Enregistrement**

nomduchamp1 : type1;
nomuchamp2 : type2;
...

Fin Enregistrement ;

Exemple :

```
Type
   t_personne =  enregistrement
      nom : chaine de caractères;
      age : entier;
    Fin ;
```

Le type enregistrement peut ensuite être utilisé pour définir des variables comme suit :

```
var
   joueur1, joueur2 : t_personne;
   tabpersonne : tableau [1..100] de t_personne;
```

III- Utilisation d'un type enregistrement

Pour donner une valeur à un "champ" de l'enregistrement, on écrit un point « . » entre le nom de la variable et le nom du champ :

« nomdevariable ». « nomduchamp» ⟵ expression

Pour utiliser la valeur d'un champ dans une expression, c'est pareil, on utilise un « . » entre le nom de la variable et le nom du champ. Par exemple, avec le type et la variable déclarés ci-dessus, on a :

```
Ecrire ("Entrez votre nom.");
Lire( joueur1.nom);
```

IV- Exercice d'application

Dans cet exercice, nous allons reprendre le schéma de l'enregistrement vu en cours et concernant des étudiants.

Nous avons au plus 500 étudiants et pour chaque étudiant nous voulons avoir les informations suivantes :

- Le matricule,
- le nom et prénom,
- le groupe,
- les notes en tp, en projet, sa moyenne et son rang.

1- Quelle est la structure de données convenable pour ce problème ?
2- Construire 2 modules : le premier qui permet de rentrer toutes les informations concernant un étudiant et toutes ses notes en algorithmique, et le second (MOYALGO) qui calcule sa moyenne.

Chapitre 3 : Fichiers

Les fichiers sont organisés sur le disque, sur la disquette ou sur tout autre support de stockage selon une arborescence logique. Pour désigner un fichier, il faut préciser le chemin complet qui permet l'accès à ce fichier.

I- Définition et déclaration d'un fichier

Un fichier (file en anglais) est un ensemble structuré de données de même type stocké en général sur un support externe (disquette, disque dur, disque optique, flash disque, ...). C'est une collection d'informations structurées en une unité d'accès appelée article ou enregistrements qui sont tous de même type.

Un fichier est donc considéré comme un empilement d'enregistrements.

Il existe deux types de fichiers :

- les **fichiers de données** : ce sont des fichiers formatés (écrits dans un format de données spécifique).
- les **fichiers textes** sont écrits au format texte.

II- Déclaration du type fichier

Type « NomEnregistrement » = enregistrement

champ 1 : type1
champ 2 : type2
.....
champ n : type n

fin enregistrement

« NomFichier » = fichier de « NomEnregistrement »

III- Organisation des fichiers

L'organisation caractérise les modes d'implémentation des informations ou des enregistrements dans le fichier et fournit les propriétés d'accès.

Au niveau de l'organisation des fichiers, on a :

- L'**organisation séquentielle** : Les enregistrements sont stockés sur le support suivant l'ordre dans lequel ils sont entrés.
- L'**organisation relative** : le fichier se compose d'un certain nombre d'enregistrements de même taille et identifiés par un numéro d'ordre.

Les propriétés d'accès aux fichiers sont :

- L'**accès séquentiel** : Pour lire une information particulière, il faut lire toutes les informations situées avant.
- L'**accès direct** : Pour lire une information particulière, il faut accéder à l'information à partir de son numéro d'ordre.

IV- Traitement ou opérations sur les fichiers

Lors du traitement d'un fichier, l'algorithme doit assurer le contrôle de ce fichier à l'aide d'une primitive d'ouverture de fichier, soit en lecture, soit en écriture, soit en lecture/écriture.

Les opérations avec les objets de type fichier sont :

- **Association ou assignation** : Avant d'utiliser un fichier, il faut associer son nom logique au nom physique par la commande « Associer ». La syntaxe est la suivante :

  Associer (Nom_logique, nom_physique)

  NB : « Nom_logique » correspond à la variable désignant le fichier et « nom_physique », le chemin d'accès au fichier.

  Ouverture d'un fichier en lecture/écriture

La syntaxe de lecture/écriture d'un fichier est

Ouvrir(F)

L'exécution de cette instruction permet une utilisation du fichier en écriture et ou lecture.

- **Fermeture d'un fichier**

A la fin de traitement du fichier, l'algorithme doit fermer le fichier F en exécutant l'instruction suivante :

Fermer(F)

V- Exercice d'application de traitement d'un fichier

1- en lecture

F: fichier
val : article
lire(F, val)

L'exécution de cette instruction permet d'introduire la valeur d'un ensemble de F dans la variable val.

2- en écriture

écrire(F, val)

L'exécution de cette instruction permet d'introduire la valeur de val dans le fichier F.

3- Traitement de fin de ficher

Fin(F)

Cette instruction est une fonction booléenne qui est fausse quand aucune action de lecture est exécutée. Elle garde cette valeur tant que les exécutions successives de lire(F, val) rencontrent les articles F. Elle prend la valeur vraie dès que l'exécution de lire(F, val) rencontre la marque de fin de fichier.

4- Parcours des éléments d'un fichier

Parcourir un fichier consiste à accéder à chaque article ou élément du fichier une et une seule fois. Les procédures générales de parcours d'un fichier sont:

procédure parcours1(F: fichier)

var val: article

début

ouvrir(F)

lire(F, val)

tant que non(Fin(F)) faire

lire(F, val)

fin tant que

fermeture(F)

fin

ou bien

procédure parcours2(F: fichier)

var val: article

début

ouvrir(F)

répéter

lire(F, val)

jusqu'à fin(F)

fermer(F)

fin

Exemple1:

procédure creer_fichier_Etudiant(F: F_Etudiant)

var val: article

début

ouvrir(F)

répéter

lire(val.code)

lire(val.nom)

lire(val.prenom)

lire(val.note)

si val.code≠' ' alors

écrire(F, val)

fin s i

jusqu'à val.code=' '

fermer(F)

fin

Exemple2:

Ecrire une fonction qui permet de calculer la taille d'un fichier

fonction taille_Fichier(F: fichier): entier

var val: Etudiant

cp: entier

début

ouvrir(F)

cp ← 0

lire(F, val)

tant que nonfin(F) faire

cp ← cp+1

lire(F, val)

fin tant que

taille_Fichier ← cp

fermer(F)

fin

VI- Exercices

Exercice d'applications : On veut informatiser la gestion d'une bibliothèque en sauvegardant dans un fichier nommée « Livres.dat » tous les livres. Chaque livre est caractérisé par :

- Titre : chaîne (30)
- Nom auteur : chaîne (30)
- Date d'édition : chaîne (10)
- Nombre de pages : entier

Écrire un programme qui permet de :

- Saisir et enregistrer les livres. La saisie se termine en répondant « N » à la question « Continuer (O/N) ? »
- Afficher tous les livres

Chapitre 4 : Piles et files

A-PILES

Une pile est une structure de données qui stocke de manière ordonnée des éléments, mais rend accessible uniquement un seul d'entre eux, appelé le **sommet de la pile**. Quant on ajoute un élément à la pile, celui-ci devient le sommet de la pile. Quant on retire un élément de la pile, on retire toujours le sommet, et le dernier élément ajouté avant lui devient alors le sommet de la pile. Autrement dit, le dernier élément ajouté dans la pile est le premier élément à en être retiré. On dit que la pile met en œuvre le principe LIFO (*Last In, First Out*).

Il existe deux (02) manières de représenter une pile :

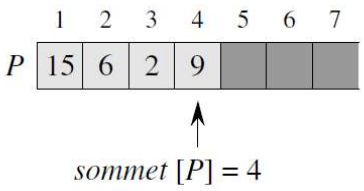
- Un tableau,
- Une liste chainée (Cours structures de données avancées).

I- Modélisation par un tableau

L'implémentation d'une structure de données pile peut se faire par un tableau, c'est-à-dire, elle consiste à utiliser un tableau pour représenter la pile.

Avec un tableau *P*, il est donc possible d'implémenter une pile d'au plus *n* éléments si la taille de celui-ci vaut *n*. Le tableau possède un attribut appelé *sommet*[*P*] qui indexe l'élément le plus récemment inséré. La pile est constituée des éléments *P*[1], ..., P[*sommet*[*P*]], où *P*[1] est l'élément situé à la base de la pile et *P*[*sommet*[*P*]] est l'élément situé au sommet.

La figure suivante représente une pile par cette modélisation :



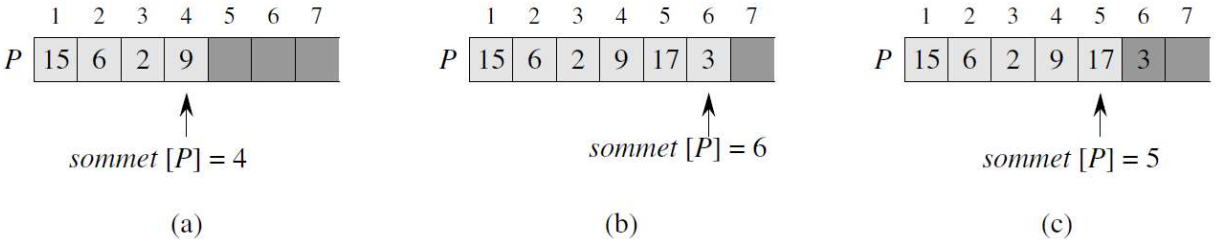
Il s'agit de l'implémentation d'une pile *P* via un tableau. Les éléments de la pile apparaissent uniquement aux positions en gris clair. La pile *P* contient 4 éléments dont l'élément sommet est 9.

II- Opérations sur une pile

Le type PILE sera utilisé pour représenter une pile au sens général sans se soucier de sa modélisation. Les opérations sur une pile sont :

- INITIALISERPILE(*P*)
- PILEVIDE(*P*)
- EMPILER(*P, x*)
- DÉPILER(*P*)

La figure suivante présente ces opérations :



➢ **Interprétation** : **(a)** La pile *P* contient 4 éléments. L'élément sommet est 9.
(b) L'état de la pile *P* après les appels EMPILER(*P,* 17) et EMPILER(*P,* 3).
(c) L'état de la pile *P* après que l'appel DÉPILER(*P*) a retourné 3, qui est l'élément le plus récemment empilé.

1. Déclaration d'une pile

Une pile est un enregistrement à 2 cases : un tableau avec une taille fixe et un indice entier qui pointe la dernière valeur ajoutée à la pile (sommet). La syntaxe est la suivante :

```
Type
    Tab = Tableau de N TypeElement; /* TypeElement est le type des données
    enregistrées dans  la pile*/
      P = Enregistrement
        T : Tab ;
        Sommet : Entier
      Fin ;
```

2. InitialiserPile (P)

L'opération INITIALISERPILE(*P*) permet d'initialiser la pile, c'est-à-dire, fait en sorte que la pile soit vide. L'algorithme suivant permet d'implémenter cette opération :

```
Procedure INITIALISERPILE (P :Pile)
DEBUT
    sommet[P] = 0 ;
FIN
```

3. PileVide (P)

L'opération PILEVIDE(*P*) est de type booléen et permet d'indiquer si la pile est vide. L'algorithme suivant permet d'implémenter cette opération :

```
Fonction PILEVIDE(P : Pile) : booléen
    Si sommet[P] = 0 Alors
        retourner VRAI
    Sinon
        retourner FAUX
Fin
```

4. Empiler(*P*,x)

Cette opération est de type booléen et permet d'empiler l'élément x au sommet de la pile P. VRAI est retournée si l'opération a réussi.

L'algorithme suivant permet d'implémenter cette opération :

```
Procedure EMPILER(P, x)

Debut

    sommet[P] ← sommet[P] + 1

    P[sommet[P]] ← x

Fin
```

5. Depiler(*P*)

Cette opération est de type booléen et permet de dépiler l'élément au sommet de la pile P. Cela revient à récupérer la valeur de l'élément au sommet avant de le supprimer de cette dernière. VRAI est retournée si la pile n'est pas vide.

L'algorithme suivant permet d'implémenter cette opération :

```
DEPILER(P)
  Si PILE-VIDE(P) Alors
      erreur « débordement négatif »
  Sinon
     sommet[P] ← sommet[P] – 1
     retourner P[sommet[P] + 1]
```

B- Files

Une file est une structure de données qui stocke de manière ordonnée des éléments, mais rend accessible uniquement un seul d'entre eux, appelé la **tête de la file**. La file comporte une **tête** et une **queue.** Quant on ajoute un élément à la file, c'est à la queue de la file que celui-ci est ajouté. Quant on retire un élément de la file, on retire toujours la tête de la file. Autrement dit, le premier élément ajouté dans la file est le premier élément à en être retiré. On dit que la pile met en oeuvre le principe FIFO (*First In, First Out*).

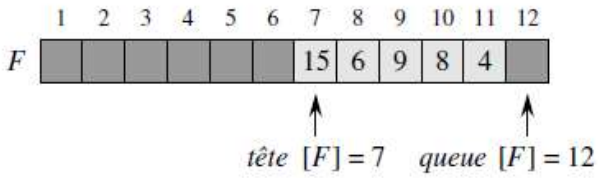
Il existe deux (02) manières de représenter une file :

- Un tableau,
- Une liste chainée.

I- Modélisation par un tableau

L'implémentation d'une structure de données file peut se faire par un tableau, c'est-à-dire, elle consiste à utiliser un tableau pour représenter la file. L'ajout d'un élément se fait à la suite du dernier élément du tableau. Le retrait d'un élément de la file se fera en enlevant le premier élément du tableau. Il faut donc deux indices pour ce tableau, le premier qui indique le premier

élément de la file et le deuxième qui indique la fin de la file. La figure suivante représente une file d'attente par cette modélisation :



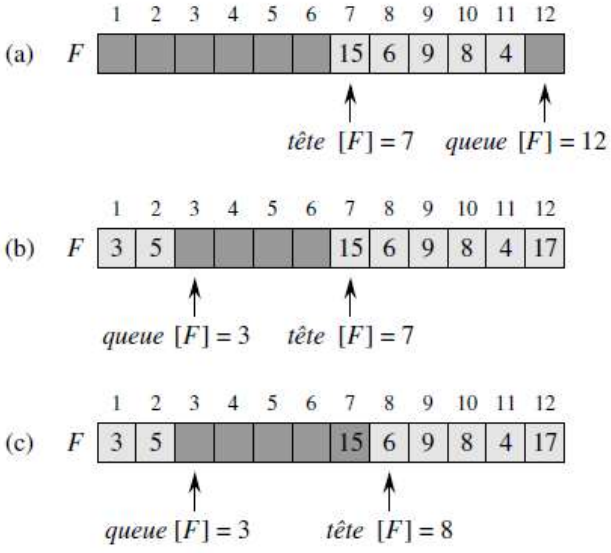
La file implémentée à l'aide d'un tableau $F[1 . . 12]$ contient 5 éléments, aux emplacements $F[7 . . 11]$. Les éléments de la file apparaissent uniquement aux positions en gris clair.

II- Opérations sur une file

Le type FILE sera utilisé pour représenter une file au sens général sans se soucier de sa modélisation. Les opérations sur une pile sont :

- INITIALISERFILE(*F*)
- FILEVIDE(*F*)
- TAILLEFILE(F)
- ENFILER(F, *x*)
- DÉFILER(F)

La figure suivante présente ces opérations :



- **Interpretation** : **(a)** La file contient 5 éléments, aux emplacements *F*[7 . . 11].
  (b) La configuration de la file après les appels ENFILER(*F,* 17), ENFILER(*F,* 3) et ENFILER(*F,* 5).
  (c) La configuration de la file après l'appel DÉFILER(*F*) qui retourne la valeur de clé 15 précédemment en tête de file. La nouvelle tête à la clé 6.

6. Déclaration d'une file

```
Type
    Tab = Tableau de N TypeEleement ; /* TypeElement est le type des données
    enregistrées dans la file*/
    F = Enregistrement
      T : Tab ;
       Debut, Queue : Entier
    Fin ;
```

7. InitialiserFile (F)

L'opération INITIALISERFILE(*F*) permet d'initialiser la file, c'est-à-dire, elle fait en sorte que la file soit vide :

```
INITIALISERFILE(F)
Debut
        tete[F] =0 ;
        queue[F]=0 ;
Fin
```

8. FileVide (F)

L'opération FILEVIDE(F) est de type booléen et permet d'indiquer si la file est vide. L'algorithme suivant permet d'implementer cette opération :

```
FONCTION FILEVIDE(P) : BOOLEEN
DEBUT
SI tete[F] = 0 ALORS
    retourner(VRAI)
SINON
    retourner(FAUX)
FIN
```

9. TailleFile(F)

La fonction *Taille*(f) permet de calculer la taille de la FILE.

```
FONCTION TAILLEFILE(F) : ENTIER
DEBUT
SI FileVide (F) ALORS
    Retourner 0
SINON
    SI tete[F] ≤ queue[F] ALORS
        retourner (queue[F] - queue[F]+1)
    SINON
        retourner (queue[F] + queue[F]+1)
    FIN
FIN
```

10. EnFiler(*F*,x)

Cette opération est de type booléen et permet d'enfiler l'élément x à la queue de la file F si celle-ci n'est pas pleine.

11. DeFiler(F)

Cette opération permet de défiler le premier élément de la tête de la file si celle-ci n'est pas vide.