

Chapitre : Les Files d'attente

Introduction

Une file d'attente est une structure de donnée complémentaire qui fonctionne avec le principe du FIFO (FIRST IN FIRST OUT). Les files d'attente sont utilisées dans les applications de gestion de réservation et/ou de gestion priorité des processus. Une file d'attente fonctionne essentiellement sur la base de primitives. Les primitives associées sont réalisées soit avec des pointeurs soit avec des tableaux.

I - Les primitives associées aux files d'attente

Ces primitives sont des méthodes qui permettent d'appliquer des opérations spécifiques à une ou des files d'attente. Elles sont:

- **initFile()**
- **fileVide()**
- **filePleine()**
- **Enfiler()**
- **Defiler()**

1) La primitive initFile():

Elle permet d'initialiser les arguments, d'une file d'attente, passés en paramètres.

2) La primitive fileVide():

Elle reçoit les arguments d'une file d'attente puis renvoie vrai si la file d'attente est vide et faux dans le cas contraire.

3) La primitive filePleine():

Elle reçoit les arguments d'une file d'attente puis renvoie vrai si la file d'attente est pleine et faux dans le cas contraire

4) La primitive Enfiler() ou AjouterFile():

Elle reçoit les arguments d'une file et une valeur puis ajoute la valeur dans la file tout en respectant le principe du FIFO (Toute valeur est ajoutée en queue de file si la file n'est pas vide sinon elle est placée en Tête de file).

NB : La primitive Enfiler() ne peut être utilisée que si la file n'est pas pleine.

5) La primitive defiler() ou EnleverFile():

Elle reçoit les paramètres d'une file d'attente puis extrait la valeur située en Tête de file afin de la sauvegarder dans une variable pour d'éventuels traitements.

Nb : La file ne peut être défilée que si elle n'est pas vide.

Remarque générale : L'implémentation d'une file d'attente est possible en utilisant soit les pointeurs (sous forme de liste dynamique) soit en utilisant les tableaux (vecteur c'est à dire sous forme de liste contiguë)

II - Implémentation d'une file d'attente en utilisant les pointeurs

Une file d'attente réalisée sous forme de pointeur a la même structure que la liste monodirectionnelle sauf que la file d'attente a deux (2) voies d'accès qui sont Tête et Queue. Tête contient l'adresse du premier élément et Queue contient l'adresse du dernier élément.

A. Déclaration de la file d'attente Syntaxe :

```
Type nomFileAttente = ↑ Structure
Debut
```

```
    info(s) : Type(s)
    SUIV : nomFileAttente
```

```
Fin
Var Tete, Queue: nomFileAttente
```

Exemple 1 : File d'attente d'entiers réalisée sous forme de pointeur

```
Type FileEntier = ↑Structure
```

```
Debut
```

```
    info : entier    suiv :
```

```
FileEntier
```

```
Fin
```

```
Var Tete,Queue : FileEntier
```

Exemple 2 : File d'attente de processus réalisée sous forme de pointeur. Processus (id, nom, etat, taille)

```
Typne FileProcessus = ↑Structure  
Debut  
    id : entier  
    nom,etat:chaîne  
    taille:reel  
    suiv : FileProcessus  
Fin  
Var Tete,Queue : FileProcessus  
2e Méthode :  
    Type PROCESSUS = Structure  
    Debut  
        id : entier  
        nom,etat:chaîne  
        taille:reel  
    Fin  
Typne FileProcessus = ↑Structure  
Debut  
    Info : PROCESSUS  
    suiv : FileProcessus  
Fin  
Var Tete,Queue : FileProcessus
```

B. Réalisation des primitives en utilisant les pointeurs

Une file d'attente réalisée avec les pointeurs ressemble à une liste monodirectionnelle c'est à dire elle a la même description que la liste monodirectionnelle. La file d'attente peut disposer de plusieurs champs informations mais elle a exactement un champ pointeur. La file d'attente a 2 voies d'accès qui sont Tête et Queue.

Tête contient l'adresse du premier élément de la file

Queue contient l'adresse du dernier élément de la file.

La file d'attente a des maillons chaines du premier au dernier

Le traitement des maillons se fait en respectant le principe du FIFO.

1- **La primitive initFile()**

Pour initialiser une file d'attente réalisée avec les pointeurs, il faut affecter à Tête et à Queue la valeur NIL.

Exercice d'application:

soit une file d'attente d'entiers realisee avec les pointeurs, ecrire un module qui realise la primitive initFile().

Solution

```
Type File = ↑Structure
```

```
Debut
```

```
    info:entier
```

```
    suiv:File
```

```
Fin
```

```
Var Tete,Queue:File
```

```
Procédure initFile(D/R Tete,Queue:File)
```

```
Debut
```

```
    Tete <- Nil
```

```
    Queue <- Nil
```

```
Fin
```

2- La primitive fileVide()

Elle reçoit les arguments d'une file puis renvoie VARI si Tête = Queue = Nil et FAUX dans le cas contraire.

NB: Une file qui n'a pas de premier n'a pas non plus de dernier et vice versa.

Exercice d'application:

Soit une file d'attente d'entiers réalisée avec les pointeurs, écrire un module qui réalise la primitive fileVide().

Solution

```
Type File = ↑Structure  
Debut  
    info:entier  
    suiv:File  
Fin  
Var Tete,Queue:File  
Fonction fileVide(Donnees Tete,Queue:File):Booleen  
Debut  
    Si Tete = Nil et Queue = Nil Alors  
        retourner Vrai  
    Sinon  
        retourner Faux  
    FinSi  
Fin
```

3- La primitive filePleine()

Une file réalisée avec les pointeurs est pleine si toutes les ressources mémoires sont occupées. Dans ce cas, la primitive renvoie VRAI sinon elle renvoie FAUX.

Exercice d'application:

Soit une file d'attente d'entiers réalisée avec les pointeurs, écrire un module qui réalise la primitive filePleine().

Solution

Type File = ↑Structure

Debut

info:entier

suiv:File

Fin

Var Tete,Queue:File

Fonction filePleine(Donnees Tete,Queue:File):Booleen

var p:File

Debut

Allouer(p)

Si p = Nil Alors

Ecrire "Toutes les ressources memoires sont occupees"

retourner Vrai

Sinon

Liberer(p)

retourner Faux

FinSi

Fin

4- La primitive enfiler() ou ajouterFile()

Elle reçoit les arguments d'une file et une valeur à ajouter dans la file. La primitive ajoute la valeur dans la file si la file n'est pas pleine. L'ajout se fait toujours en Queue de file si la file n'est pas vide sinon l'élément ajouté devient premier et dernier.

Exercice d'application :

Soit une file d'attente d'entiers réalisée avec les pointeurs, écrire un module qui reçoit les paramètres de la file et une valeur à ajouter puis réalise la primitive enfiler().

Solution

```
Type File = ↑Structure
```

```
Debut
```

```
    info:entier
```

```
    suiv:File
```

```
Fin
```

```
Var Tete,Queue:File
```

```
procedure enfiler(Donnee Val:Entier D/R Tete,Queue:File)
```

```
    Var pval:File
```

```
Debut
```

```
Si filePleine(Tete,Queue)=Vrai Alors
```

```
    Ecrire "Impossible d'ajouter car la file est pleine"
```

```
Sinon
```

```
    Allouer(pval)
```

```
    pval↑.info <- Val
```

```
    pval↑.suiv <- Nil
```

```
    SI fileVide(Tete,Queue)=Nil Alors
```

```
        Tete <- pval
```

```
        Queue <- pval
```

```
    Sinon
```

```
        Queue↑.suiv <- pval
```

```
        Queue <- pval
```

FinSi

FinSi

Fin

5- **La primitive defiler() ou enleverFile()**

Elle reçoit les arguments d'une file et extrait la valeur située en Tête de file pour la sauvegarder dans une variable pour d'éventuels traitements. Toute valeur extraite est automatiquement supprimée de la file. L'extraction ne peut se faire que si la file n'est pas vide.

Exercice d'application:

Soit une file d'attente d'entiers réalisée avec les pointeurs, écrire un module qui reçoit les paramètres de la file puis réalise la primitive defiler().

Solution **Type File = ↑Structure**

Debut

info:ntier

suiv:File

Fin

Var Tete,Queue:File

Procédure defiler(D/R Tete,Queue:File Resultat val:entier)

var pval:File

Debut

Si fileVide(Tete,Queue) = Vrai Alors

Ecrire "Impossible d'extraire car la file est vide"

Sinon pval <- Tete

val <- Tete↑.info

Si Tete = Queue Alors //File composee d'un element

initFile(Tete,Queue)

Sinon

Tete <- Tete↑.suiv

FinSi


```
        Liberer(pval)
    FinSi
Fin
```

Exercice d'application 1 :

Soit un tableau de 150 produits, écrire un module qui transfère dans une file d'attente les produits de catégorie Alimentaire. Produit (code, nom, categorie, prix unitaire, quantite)

```
Const N = 150 Type
Produit = Structure
Debut
    code, nom, categorie: chaine
    prixUnitaire: reel
    quantite: entier
Fin
Type Tab = Tableau[1..N] Produit
Type FileProduit = ↑Structure
Debut
    info : Produit
    suiv : FileProduit
Fin
Var Tete, Queue: FileProduit
Var T: Tab
Procedure CreationFile(Donnees T: Tab N: Entier
                        Resultats Tete, Queue: FileProduit)
var i: entier
Debut
    initFile(Tete, Queue)
    Pour i allant de 1 a N Faire
        Si T[i].categorie = "Alimentaire" Alors
            Enfiler(T[i], Tete, Queue)
```

```
        FinSi
    FinPour
Fin
```

Exercice application 2:

Soit une file d'attente de processus, écrire un module qui transfère dans un tableau les processus qui sont éligibles. Un processus est caractérisé par son id, son nom, son état (elu, éligible, bloqué) et sa taille.

```
Const N = 100 Type Processus = Structure
Debut
    id:entier
    nom:chaine
    etat:"Elu","Eligible","Bloqué"
    taille:reel
Fin
Type TabProcessus = Tableau[1..N]Processus
Type FileProcessus = ↑Structure
Debut
    info:Processus
    suiv:FileProcessus
Fin
var T:Tab
var Tete,Queue:FileProcessus
```

**Procedure Application(Donnees Tete,Queue:FileProcessus
Resultats T:Tab N:Entier)**

var p:Processus var i:entier

Debut

i<- 0

TantQue(FileVide(Tete,Queue)=Faux) Faire

Defiler(Tete,Queue,p)

Si p.etat="Eligible" Alors

i <- i + 1

T[i] <- p

FinSi

FinTantQue

N <- i

Fin