Les files

https://meet.google.com/bhk-pzsi-yfe

Une **file** est une liste dans laquelle toutes les insertions se font en queue et toutes les suppressions se font en tête. Par analogie avec les files d'attente, on dit que l'élément présent depuis le plus longtemps est le premier. file a une structure "FIFO" (First In, First Out) c'est-à-dire « premier entré premier sorti ».
L'accès à un élément quelconque se fait après le retrait (défilement) de tous les éléments qui le précède. Une

Les files sont très utilisées en informatique :

- Traitement des transactions : réservation des billets,…
- Allocation des ressources
- Appels téléphoniques

Opérations sur les files :

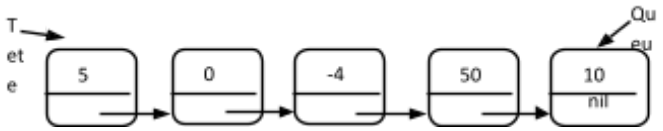
Les opérations sur les files sont les suivantes :
Ajout d'un élément : l'action consistant à ajouter un nouvel élément et le mettre au dernier : ***enfiler***.
Suppression d'un élément : l'action consistant à retirer le premier élément : ***défiler***.
Consultation : consulter le premier élément de la file.

Représentation des files en mémoire :

On peut représenter ou implémenter les files en mémoire de façon contiguë ou chaînée.

1-Représentation chaînée

Remarque : Pour ne pas avoir à parcourir toute la liste au moment d'ajouter un élément en queue, on maintient un pointeur sur le dernier élément.



Déclaration

Type file : ^ objet ; Type objet : Enregistrement inf : <type_elt> ; svt : file ; Fing ;	typedef struct ne *file ; typedef struct ne { int inf ; file svt ; } noeud ;

Allocation mémoire pour un élément de la file

Fonction creer_noeud( ) : file
Début
L : file ;
L<-nil ;
Allouer(L) ;
Si (L=nil) alors
écrire (« problème d'allocation » ;
Fsi ;
 	
Retourne (L) ;
Fin ;
	file creer_noeud( )
	{
	file L= (file)malloc(sizeof(noeud)) ;
	if (L==NULL) {
	printf("erreur d'allocation\n") ;
	exit(-1) ;
	}
	return(L) ;
	}

Ajout d'un élément

Procédure Enfiler(ES/ tete, Que : file, E/ x :<Type_elt>)
Début
p : file
Allouer(p) ;
p^.inf <- x ;
p^.svt <- nil ;
si (tete = nil) alors tete<- p ;
Que <- p ;
sinon Que^.svt <- p ;
Fsi ;
Que <- p ;
Fin ;
	void enfiler (file *tete, file *Que, int x)
	{
	file p;
	p=creer_noeud() ;
	p ->inf = x ;
	p->svt=NULL ;
	if (*tete == NULL)
	{ *tete=p;
	*Que = p ;
	}
	else (*Que) -> svt =p;
	*Que = p ;
	}

Suppression d'un élément

Procédure défiler(ES/ tete, Que : file, s/ x :<Type_elt>)
Début
t : file ; // le test si la file est vide se fait dans le prog
x<- tete^.inf ;
t<- tete ;
tete<- tete^.svt ;
Libérer(t) ;
Si(tete=nil) alors Que <- nil ;
Fin ;
	void defiler(file *tete, file *Que, int *x)
	{
	file t ;
	*x = (*tete) ->inf ;
	t=*tete;
	* tete =(* tete)->svt;
	free(t) ;
	if(*tete==NULL) *Que=NULL ;
	}

Test si la file est vide

Fonction file_vide(E/ t : file) : booléen
Début
Si(t=nil) alors retourne vrai ;
sinon retourne faux ;
Fsi ;
Fin ;
	int file_vide(file t)
	{
	if (t==NULL) return(1);
	else return(0);
	}

Consulter l'élément se trouvant en tête de file

Fonction tete_file(E/ t : file) : <type_ele>
Début
Retourne (t^.inf) ;
Fin ;
	int sommetfile(file t)
	{
	return(t->inf) ;
	}

Exemple :

Début
T, Q : file ;
car : entier ;
 	
Lire(car) ;
Tan que (car ≠ 0)
Faire
Enfiler(T, Q, car) ;
Lire(car) ;
Fait ;
 	
//Affichage
Tan que(T ≠ nil)
Faire
défiler(T, Q, car) ;
Ecrire (car) ;
Fait ;
Fin ;
	int main()
	{
	file tete=NULL ;
	int car;
	
	scanf("%d",&car);
	while(car!=0)
	{
	enfiler(&tete, &Que, car);
	scanf("%d",&car);
	}
	
	printf("\n");
	
	while(tete!=NULL)
	{
	defiler(&tete, &Que, &car);
	printf("%d\n",car);
	}
	return 0;
	}

1-Représentation contigüe

On peut représenter une file par un tableau alloué d'une manière statique. On peut choisir le triplet suivant : (T, Tete, Queue) pour manipuler cette file.

5	0	-4	50	10	…… ……				

1) Déclaration Constante max 100 Type File = Enregistrement T : tableau [max] <Type_Elt> ; Tete, Queue : entier ; Fing ; f : File ;	**2) Initialisation** Fonction InitFile ( ) : File Début f : File f.Tete <-1; f.Queue <-0; retourner (f); Fin	**3) Consultation de la tête de file** Fonction TeteFile (E/ f : File) : <TypeElt> Début retourner (f.T[f.Tete]) ; Fin

4) Test si la file est vide Fonction FileVide (E/ f : File) : booléen Debut si (f.Queue < f.Tete) alors retourner (vrai) ; sinon retourner (faux) ; fsi ;	**5) Test si la file est pleine** Fonction FilePleine (E/ f : File) : booléen Début si (f.Queue=max) alors retourner (vrai) ; sinon retourner (faux) ; fsi ;

Fin ;	Fin ;

6) Ajout d'un element	**7) Suppression d'un élément**

Procedure Enfiler (E/S f: File, E/ x : <TypeElt>)
Début
  f.Queue <- f.Queue+1 ;
  f.T[f.Queue] <- x ;
Fin
	Procedure Défiler (E/S f: File, E/S x : <TypeElt>)
	Début
	  x <- f.T[f.Tete] ;
	  f.Tete <- f.Tete+1 ;
	Fin

Remarque : A tout moment le nombre d'éléments dans la file est F.queue- F.tête + 1.

Exercice 1 : réécrire les actions précédentes à l'aide du triplet suivant : (T, tete, nb_elt)

Exercice 2 : Etant donné N entiers strictement positifs. On veut afficher les nombres parfait de cet ensemble en suivant les étapes suivantes (voir les figures 1,2 et 3):

1. Donner la définition de la structure donnée par la figure 3
2. Initialiser les deux tableaux : Tete et Queue (voir figure 1)
3. Créer pour chaque élément du tableau un enregistrement avec la valeur 1 représentant le premier diviseur. Mettre l'adresse de cet élément dans le tableau Tete et dans le tableau Queue (voir figure 2)
4. Construire la liste des diviseurs pour chaque valeur du tableau (voir figure 3)
5. Afficher les nombres parfaits de cet ensemble.

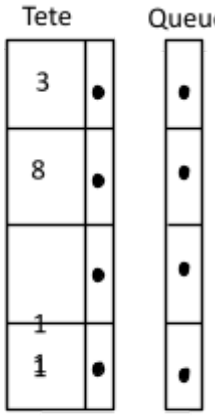


Figure 1

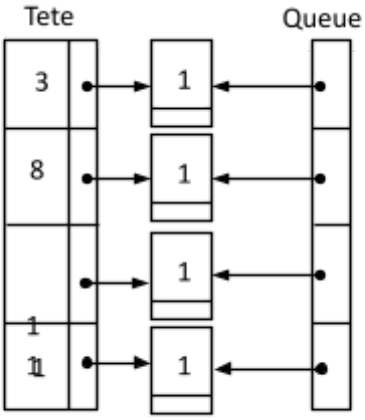


Figure 2

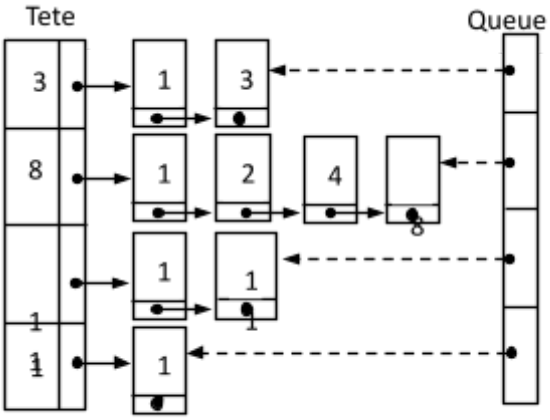


Figure 3

Solution :

1/

```
Type file : ^ objet ;
Type objet : Enregistrement
     inf : entier;
     svt : file ;
 Fing ;

Queue : tab [100] file ;
Tete : tab [100] objet ;
```

2/
```
i, x : entier ;
Pour (i<- 1 à 100)
  Faire
    Lire (Tete[i].inf) ;
    Tete[i].svt <- nil ;
    Queue[i] <- nil
  Fait ;
```

3/
```
i : entier ;
Pour (i<- 1 à 100)
  Faire
    Enfiler (Tete[i].svt, Queue[i],  1);
  Fait ;
```

4/
```
I, j : entier ;
Pour (i<- 1 à 100)
  Faire
    Pour (j <- 2 à (Tete[i].inf)/2)
      Faire
        Si (Tete[i].inf mod j = 0)  alors  Enfiler (Tete[i].svt, Queue[i],  j);
        Fsi ;
      Fait ;
      Enfiler (Tete[i].svt, Queue[i],  Tete[i].inf);
  Fait ;
```

5/
```
I, som, x, val : entier ;

Pour (i<- 1 à 100)
  Faire
    som <- 0 ;
    val <-  tete_file(Tete[i].inf) ;
    Tant que (file_vide(Tete[i].svt) =faux )
      Faire
        Défiler (Tete[i].svt, Queue[i],  x) ;
        som <- som + x ;
      Fait ;
      som <- som - val ;
      si ( som = val) alors ecrire(val, « est un nombre parfait ») ; fsi ;
  Fait ;
```