PILES

Motivation

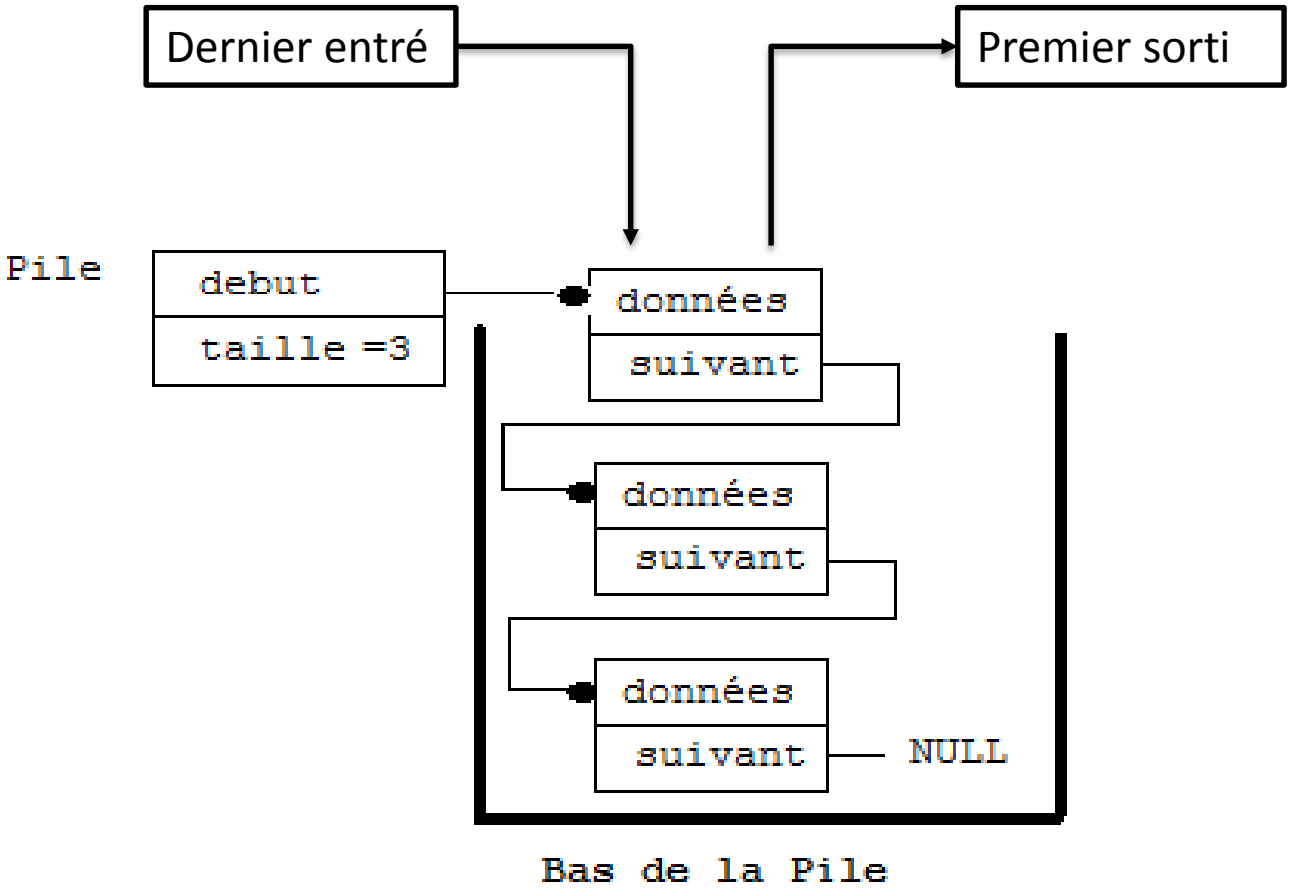
Structure de donnée :

- Pile

Algorithmes

- LIFO (Last In First Out)

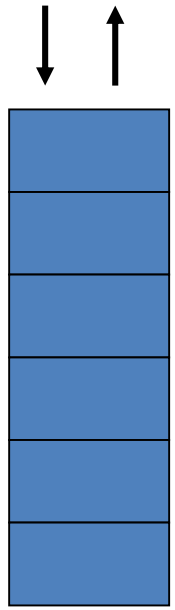
Liste chaînée spécialisée



Liste chaînée spécialisée

Pile, ou Tas (Stack): structure LIFO

`void empiler(Data_t*)` `Data_t* dépiler(void)`



Pile, un TDA

Définition : Une pile est un ensemble dynamique tel que la suppression concerne toujours le dernier élément inséré. Une telle structure est aussi appelé LIFO (last-in, first out).

Opérations :
empiler(x) insère un élément à l'entrée de la pile;
dépiler() retourne et supprime l'élément en entrée de pile;

Applications

La pile d'exécution : les appels des méthodes dans l'exécution d'un programme sont gérés par une pile.

Éditeur de texte : une pile est fournie par les éditeurs de texte évolués qui possèdent le couple d'actions « annuler-répéter ».

Fonctions utiles …

- Définir le TDA
- Ajouter élément (empiler)
- Retirer élément (dépiler)
- Tester si vide

Définition de la pile

- **TDA des nœuds**

```
typedef struct node{
    int valeur;
    struct node *prec;
} node, *Node ;
```

- **TDA de la pile**

```
typedef struct pile {
    Node tete;
    int taille;
} pile, *Pile ;
```

Création d'un node

```
Node createStackNode(int n){
        Node nœud = (Node)malloc(sizeof(node));
        nœud→val = n;
        nœud→prec= NULL;
        return nœud;
}
```

Création de la pile

```
Pile createStack(){
        Pile tas = (Pile)malloc(sizeof(pile));
        tas→tete = NULL;
        tas→taille = 0;
        return tas;
}
```

Insertion : empiler (push)

```
void empiler (Pile * p, int Val) {

        Node tas = malloc (sizeof (node) );

        if(!tas) exit(EXIT_FAILURE); /* Si l'allocation a échouée. */

        tas → valeur = Val;

        (*p) → taille ++;

         tas →prec = (*p)→tete;

         (*p)→tete =  tas ; /* Le pointeur pointe sur le dernier élément. */
}
```

Retrait : dépiler (pop)

```
int depiler (Pile *p) {

        int val;

        Node tmp;

        if(!(*p)→tete) return -1; /* Retourne -1 si la pile est vide. */

        tmp = (*p)→tete→prec;

        val = (*p)→tete→valeur;

        free((*p)→tete); (*p)→tete = tmp;  *p→taille --;
        /* Le pointeur pointe sur le dernier élément.*/

        return val; /* Retourne la valeur soutirée de la pile. */
}
```

Taille de la pile : length

```
int length (Pile p) {
        int n=0;
        while(p →tete) {
                n++;
        p→tete = p→tete→prec;
        }
        return n;
}
```

```
int length (Node p) {
    if (! p)
        return 0 ;
    return 1 +
        length (p →prec);
}
```

Vider la pile : clear

```
void clear (Pile *p) {
        pile *tmp;
        (*p) → taille = 0;
        while ((*p)→tete) {
                tmp = (*p)→tete→prec;
                free((*p) →tete);
                (*p) →tete = tmp;
        }
}
```

Afficher la pile : view

```
void view(Pile p) {

        while(p→tete) {

                printf("%d\n",p→tete→valeur);

                p→tete = p→tete→prec;
        }
}
```

Tester si la pile est vide

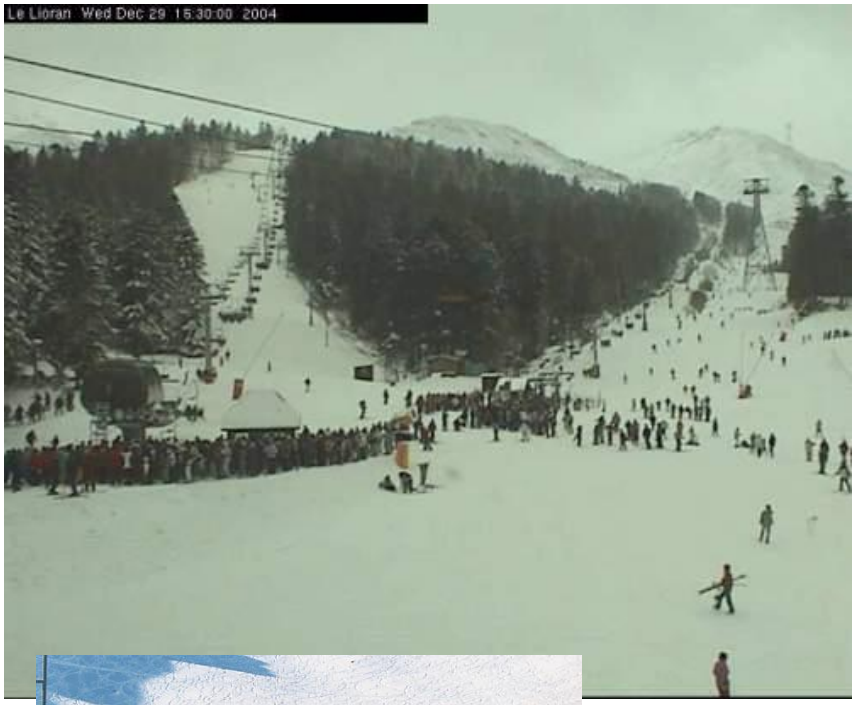
```
int isEmpty (Pile p) {

        return p→taille ;

}
```

QUEUES OU FILES

Motivation



Structure de donnée :

- File

Algorithmes

- FIFO
  (First In First Out)

Aussi: File à priorité

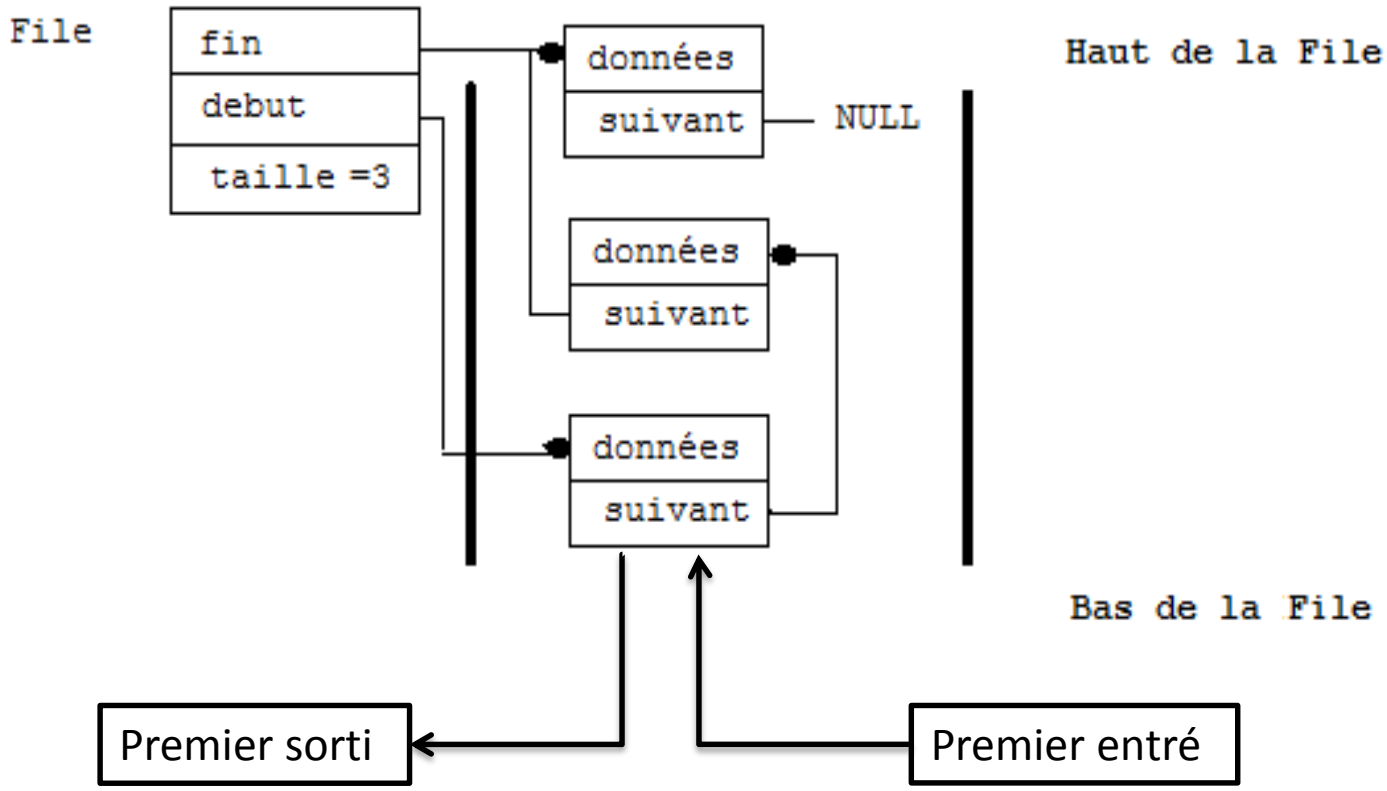
Définition

- **File** (*queue*) = structure de données fondée sur le principe "premier arrive, premier sorti" (**FIFO** : *First In, First Out*). Exemple de file d'attente.
  - Accéder au premier élément et au dernier élément
  - Défiler → premier élément.
  - Enfiler → ajouter élément a la fin.
  - Détecter si elle est vide (et si elle est pleine).

Implantation

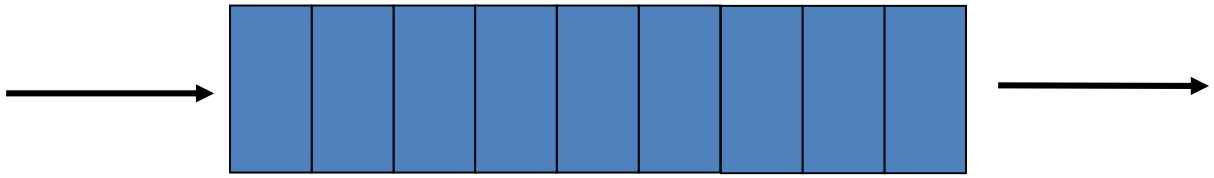
- Il est possible d'implanter des files en C en utilisant des
  - tableaux,
  - listes simplement chaînées ou
  - listes doublement chaînées.

Liste chaînée spécialisée



Liste chaînée spécialisée

File, ou queue : structure FIFO



```
void enfiler(Data_t*)
```

```
Data_t* défiler(void)
```

File un T.D.A.

Définition : Une file est un ensemble dynamique tel que les insertions se font d'un coté (l'entrée de file) et les suppressions de l'autre coté (la sortie de file).

Opérations :
enfiler(x) ajoute un élément en entrée de file;
défiler() supprime l'élément situé en sortie de file.
détecter si elle est vide (éventuellement pleine)

Applications

Les files d'attentes pour les systèmes de réservations, d'inscriptions, d'accès à des ressources…

Définition des TAD

```
typedef struct Element{
        int val;
        struct Element * suiv;
} Element, *PElement ;
```

```
typedef struct file{
        Element *debut;
        Element *fin;
        int taille;
} file, *File ;
```

Création d'un nœud

```
PElement createFileElement(int n){
        PElement nœud = (PElement)malloc(sizeof(Element));
        nœud→val = n;
        nœud→suiv= NULL;
        return nœud;
}
```

Création de la file

```
File createFile(){
        File queue= (File)malloc(sizeof(file));
        queue→debut= NULL;
        queue→fin= NULL;
        queue→taille = 0;
        return queue;
}
```

Fonction *enfiler*

```
void enfiler (File * f, int val){
        PElement el = (PElement) malloc (sizeof(Element));
        if (!el)
                exit(EXIT_FAILURE);
        el→val =val;
        el→suiv = (*f)→debut;
        (*f)→debut = el;
        (*f)→taille ++;
        if ((*f)→taille == 1)  (*f)→fin = el;
}
```

Fonction défiler

```
int defiler (File *f){
        if (!(*f)→fin) return -1; //(*f)→taille = 0
        int val = (*f)→fin→val;
        if ((*f)→taille == 1){
                        (*f)→fin = NULL; free((*f)→debut) ;
        }
        (*f)→taille --;
        else {      PElement q, p = (*f)→debut;
                        while (p && p→suiv) {q =p; p = p→suiv;}
                        (*f)→fin = q; q→suiv = NULL; free(p);
      return val;
}
```

Taille de la file : length

```
int length (File f) {
        int n=0;
        while(f→debut) {
                n++;
        f →debut = f→debut→suiv;
        }
        return n;
}
```

```
int length (PElement f) {
    if (!f)
        return 0 ;
   return 1 + length (f→suiv);
}
```

Vider la file : clear

```
void clear (File *f) {
        file *tmp;
        (*f) → taille = 0;
        (*f) → fin = NULL;
        while ((*f)→debut) {
                tmp = (*f)→debut→suiv;
                free((*f)→debut);
                (*f)→debut = tmp;
        }
}
```

Afficher la file : view

```
void view(File f) {

        while(f→debut) {

                printf("%d\n",f→debut→val);

                f→debut = f→debut→suiv;
        }
}
```

Tester si la file est vide

```
int isEmpty (File f) {

        return f→taille ;

}
```