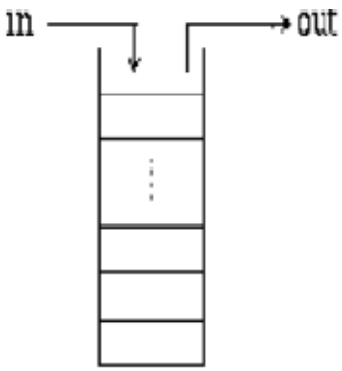
PILE &FILE

PILE



Ex : pile d'assiettes, pile de dossiers à traiter, …
Une pile est une structure linéaire permettant de stocker et de restaurer des données en n'autorisant que 4 opérations:
1. consulter le dernier élément de la pile (le sommet de la pile)
2. tester si la pile est vide
3. empiler un élément, le mettre au sommet de la pile ==> PUSH
4. dépiler un élément (enlever l'élément au sommet) ==> POP

Implémentation de la structure Pile à l'aide d'une liste chaînée

1. Définition du type Pile

```
typedef int Element; /* les éléments sont des int */
typedef struct cellule {
Element valeur;
struct cellule *suivant;
} Cellule;
typedef Cellule *Pile;
```

2. Déclaration des fonctions gérant la pile

```
Pile pile_vide ();
Pile empiler ( Pile p, Element e );
Pile depiler ( Pile p );
Element sommet ( Pile p );
int est_vide ( Pile p );
```

3. Définition des fonctions gérant la pile

```
Pile pile_vide(void) {
return NULL;}
```

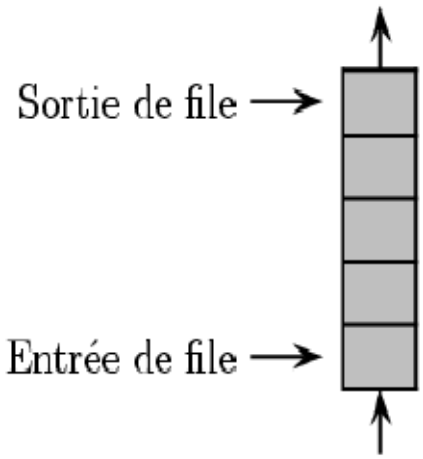
```
int est_vide(Pile p) {
return (p == NULL);
}

Pile empiler(Pile p, Element e) {
Cellule * pc;
pc=(Cellule*)malloc(sizeof Cellule));
pc->valeur=e;
pc->suivant=p;
return pc;
}

Pile depiler(Pile p) {
Cellule * pc = p;
if (est_vide(p)) {
printf("Erreur: pile vide!\n");
exit(-1);
}
p=p->suivant;
free(pc);
return p;
}

Element sommet(Pile p) {
/* pré-condition: pile non vide ! */
if (est_vide(p)) {
printf("Erreur: pile vide !\n");
exit(-1);
}
return p->valeur;
}
```

4. Exemple d'utilisation

```
int main () {
Pile p = pile_vide();
p = empiler(p,50);
p = empiler(p,5);
p = empiler(p,20);
p = empiler(p,10);
printf("%d au sommet après empilement de 50, 5, 20 et 10\n", sommet(p));
p = depiler(p);
p = depiler(p);
printf("%d au sommet après dépilement de 10 et 20\n", sommet(p));
return 0;
}
```

File

La file est modifiée à ses deux bouts.



Une file est une structure de données dynamique dans laquelle on insère des nouveaux éléments à la fin (queue) et où on enlève des éléments au début (tête de file).
Quatre opérations de base:
1. consulter le premier élément de la file
2. tester si la file est vide
3. enfiler un nouvel élément: le mettre en dernier
4. défiler un élément, le premier (le supprimer)

Implémentation de la structure File à l'aide d'une liste chaînée

1. Définition du type File

```
typedef int Element; /* les éléments sont des int */
typedef struct cellule {
Element valeur;
struct cellule *suivant;
} Cellule;
typedef struct {
struct cellule *tete;
struct cellule *queue;
} File;
```

2.Déclaration des fonctions gérant la file

```
File file_vide ();
File enfiler ( File f, Element e );
File defiler ( File f );
Element tete ( File f );
int est_vide ( File f );
```

3. Définition des fonctions gérant la file

```
File file_vide() {
File f;
f.tete=f.queue=NULL;
return f;
}

int est_vide(File f) {
return (f.tete==NULL)&&(f.queue==NULL); }

Element tete(File f) {
/* pré-condition: file non vide ! */
if (est_vide(f)) {
printf("Erreur: file vide !\n");
exit(-1);
}
return (f.tete)->valeur;
}

File enfiler(File f, Element e) {
Cellule * pc;
pc=(Cellule *)malloc(sizeof(Cellule));
pc->valeur=e;
pc->suivant=NULL;
if (est_vide(f)
f.tete=f.queue=pc;
else f.queue=(f.queue)->suivant=pc;
return f;
}

File defiler(File f) {
Cellule * pc;
if (est_vide(p)) {
printf("Erreur: file vide !\n");
exit(-1);
}
pc=f.tete;
f.tete=(f.tete)->suivant;
free(pc);
if (f.tete==NULL) f.queue=NULL;
return f;
}
```