

Chapitre 5 : Les Piles et les Files

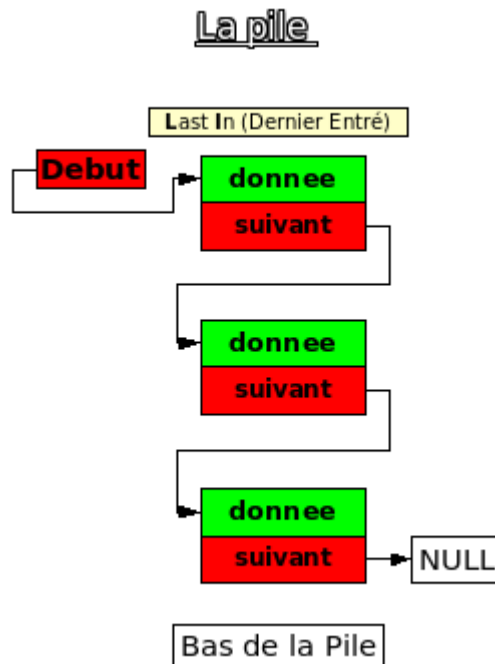
Les Piles :

La pile est une structure de données, qui permet de stocker les données dans l'ordre LIFO (Last In First Out) - en français *Dernier Entré Premier Sorti*).

La récupération des données sera faite dans l'ordre inverse de leur insertion.

L'insertion se faisant toujours au début de la liste, le 1er élément de la liste sera le dernier élément saisi, donc sa position est en haut de la pile.

Ce qui nous intéresse c'est que le dernier élément entré, sera le 1er élément récupéré.



La construction du prototype d'un élément de la pile

Pour définir un élément de la pile le type *struct* sera utilisé.

L'élément de la pile contiendra un champ *donnee* et un pointeur *suivant*.

Le pointeur *suivant* doit être du même type que l'élément, sinon il ne pourra pas pointer vers l'élément.

Le pointeur *suivant* permettra l'accès vers le prochain élément.

```
typedef struct ElementListe {
    char *donnee;
    struct ElementListe *suivant;
}Element;
```

Pour permettre les opérations sur la pile, nous allons sauvegarder certains éléments :

- le premier élément
- le nombre d'éléments

Le 1er élément, qui se trouve en haut de la pile, nous permettra de réaliser l'opération de récupération des données situées en haut de la pile. Pour réaliser cela, une autre structure sera utilisée (ce n'est pas obligatoire, des variables peuvent être utilisées).

Voici sa composition :

```
typedef struct ListeRepere{  
    Element *debut;  
    int taille;  
} Pile;
```

Observation :

Quelque soit la position dans la liste, le pointeur debut pointe toujours vers le 1er élément, qui sera en haut de la pile.

Le champ taille contiendra le nombre d'éléments de la pile, quelque soit l'opération effectuée sur la pile.

Opérations sur les piles

A. Initialisation

Cette opération doit être faite avant toute autre opération sur la pile.

Elle initialise le pointeur debut avec le pointeur NULL, et la taille avec la valeur 0.

La fonction

```
void initialisation (Pile * tas){  
    tas->debut = NULL;  
    tas->taille = 0;  
}
```

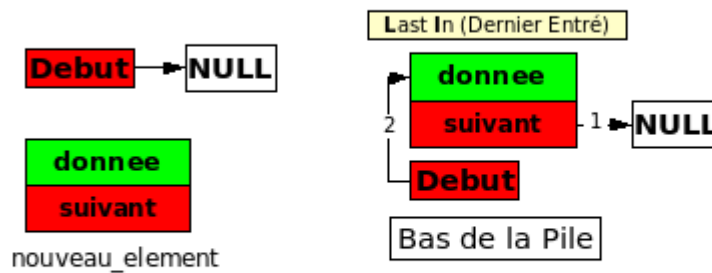
B. Insertion d'un élément dans la pile

Voici l'algorithme d'insertion et de sauvegarde des éléments :

- déclaration d'élément(s) à insérer
- allocation de la mémoire pour le nouvel élément
- remplir le contenu du champ de données
- mettre à jour le pointeur debut vers le 1er élément (le haut de la pile)
- mettre à jour la taille de la pile

La 1ère image montre le début de l'insertion, donc la liste a la *taille* 1 après l'insertion.

Insertion dans la pile vide



(1) nouveau_element->suivant = liste->debut;
(2) liste->debut = nouveau_element;
liste->taille++;

La fonction :

/* empiler (ajouter) un élément dans la pile */

```
int empiler (Pile * tas, char *donnee){  
    Element *nouveau_element;  
    nouveau_element = (Element *) malloc (sizeof (Element)) ;  
    strcpy (nouveau_element->donnee, donnee);  
    nouveau_element->suivant = tas->debut;  
    tas->debut = nouveau_element;  
    tas->taille++;  
}
```

Ôter un élément de la pile

Pour supprimer (ôter ou dépiler) l'élément de la pile, il faut tout simplement supprimer l'élément vers lequel pointe le pointeur debut.

Cette opération ne permet pas de récupérer la donnée en haut de la pile, mais seulement de la supprimer.

Les étapes :

- le pointeur supp_elem contiendra l'adresse du 1er élément
- le pointeur debut pointera vers le 2ème élément (après la suppression du 1er élément, le 2ème sera en haut de la pile)
- la taille de la pile sera décrémentée d'un élément

La fonction

```
int depiler (Pile * tas){  
    Element *supp_element;  
    if (tas->taille == 0)  
        return -1;  
    supp_element = tas->debut;  
    tas->debut = tas->debut->suivant;  
    free (supp_element->donnee);  
    free (supp_element);  
    tas->taille--;  
    return 0;  
}
```

Affichage de la pile

Pour afficher la pile entière, il faut se positionner au début de la pile (le pointeur debut le permettra).
Ensuite, en utilisant le pointeur suivant de chaque élément, la pile est parcourue du 1er vers le dernier élément.
La condition d'arrêt est donnée par la taille de la pile.

La fonction

```
/* affichage de la pile */  
void affiche (Pile * tas){  
    Element *courant;  
    int i;  
    courant = tas->debut;  
    for(i=0;i<tas->taille;++i){  
        printf("\t\t%s\n", courant->donnee);  
        courant = courant->suivant;  
    }  
}
```

Récupération de la donnée en haut de la pile

Pour récupérer la donnée en haut de la pile sans la supprimer, j'ai utilisé une macro.
La macro lit les données en haut de la pile en utilisant le pointeur debut.
#define pile_donnee(tas) tas->debut->donnee

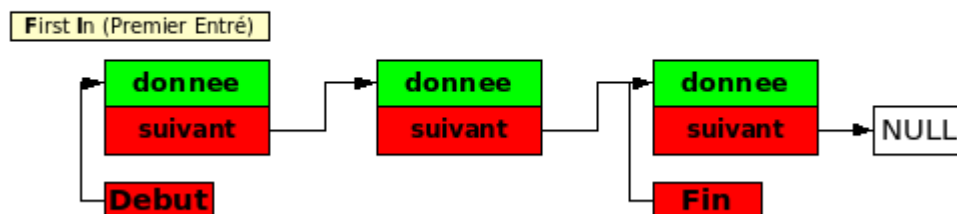
Les Files :

Définition

La file est une structure de données, qui permet de stocker les données dans l'ordre FIFO (First In First Out) - en français *Premier Entré Premier Sorti*).
La récupération des données sera faite dans l'ordre d'insertion.

Pour l'implémentation j'ai choisi une liste simplement chaînée.
L'insertion dans la file se fera dans l'ordre normal, le 1er élément de la file sera le premier élément saisi, donc sa position est au début de la file.

La file



III. La construction du prototype d'un élément de la file

Pour définir un élément de la file le type *struct* sera utilisé.

L'élément de la file contiendra un champ *donnee* et un pointeur suivant.

Le pointeur suivant doit être du même type que l'élément, sinon il ne pourra pas pointer vers l'élément.

Le pointeur suivant permettra l'accès vers le prochain élément.

```
typedef struct ElementListe {  
    char *donnee;  
    struct ElementListe *suivant;  
}Element;
```

Pour avoir le contrôle de la file, il est préférable de sauvegarder certains éléments : le premier élément, le dernier élément, le nombre d'éléments.

Pour réaliser cela, une autre structure sera utilisée (ce n'est pas obligatoire, des variables peuvent être utilisées).

Voici sa composition:

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

IV. Opérations sur les files

A. Initialisation

Prototype de la fonction :

```
void initialisation (File * suite);
```

Cette opération doit être faite avant toute autre opération sur la file.

Elle initialise le pointeur *debut* et le pointeur *fin* avec le pointeur *NULL*, et la taille avec la valeur 0.

La fonction

```
void initialisation (File * suite){  
    suite->debut = NULL;  
    suite->fin = NULL;  
    suite->taille = 0;  
}
```

B. Insertion d'un élément dans la file

Voici l'algorithme d'insertion et de sauvegarde des éléments :

- déclaration d'élément(s) à insérer
- allocation de la mémoire pour le nouvel élément

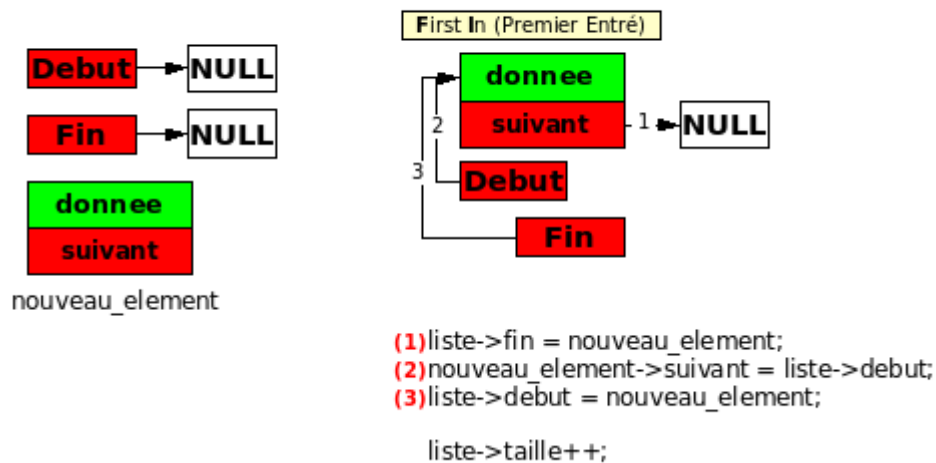
- remplir le contenu du champ de données
- mettre à jour le pointeur debut vers le 1er élément (le début de file)
- mettre à jour le pointeur fin (ça nous servira pour l'insertion vers la fin de la file)
- mettre à jour la taille de la file

Prototype de la fonction :

int enfiler (File * suite, Element * courant, char *donnee);

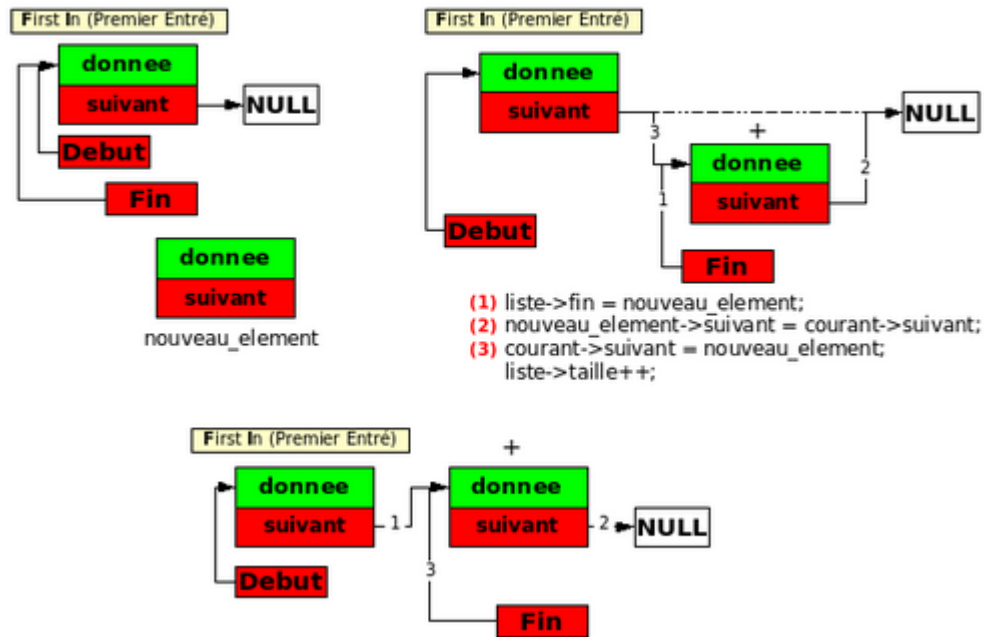
La 1ère image affiche le début de l'insertion, donc la liste a la taille 1 après l'insertion.

Insertion dans la file vide



Dans la file, l'élément à récupérer c'est le 1er entré. Pour cela, l'insertion se fera toujours à la fin de la file. Il s'agit de l'ordre normal de l'insertion (1er, 2ème, 3ème etc.).

Insertion dans la file



La fonction

```
int enfiler (File * suite, Element * courant, char *donnee){
    Element *nouveau_element;
    if ((nouveau_element = (Element *) malloc (sizeof (Element))) == NULL)
        return -1;
    if ((nouveau_element->donnee = (char *) malloc (50 * sizeof (char)))
        == NULL)
        return -1;
    strcpy (nouveau_element->donnee, donnee);

    if(courant == NULL){
        if(suite->taille == 0)
            suite->fin = nouveau_element;
        nouveau_element->suivant = suite->debut;
        suite->debut = nouveau_element;
    }else {
        if(courant->suivant == NULL)
            suite->fin = nouveau_element;
        nouveau_element->suivant = courant->suivant;
        courant->suivant = nouveau_element;
    }
    suite->taille++;
    return 0;
}
```

C. Oter un élément de la file

Pour supprimer (ôter) l'élément de la file, il faut tout simplement supprimer l'élément vers lequel pointe le pointeur debut. Cette opération ne permet pas de récupérer la donnée au début de la file (la première donnée), mais seulement de la supprimer.

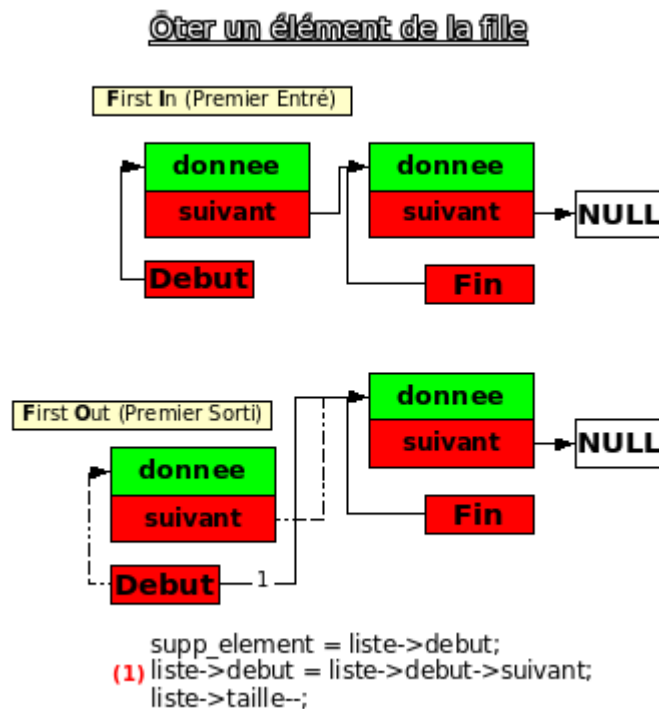
Prototype de la fonction :

```
int de_filer (File * suite);
```

La fonction renvoie -1 en cas d'échec sinon elle renvoie 0.

étapes :

- le pointeur `supp_elem` contiendra l'adresse du 1er élément
- le pointeur `debut` pointera vers le 2ème élément (après la suppression du 1er élément, le 2ème sera au début de la file)
- la taille de la file sera décrémentée d'un élément



La fonction

```
int de_filer (File * suite){  
    Element *supp_element;  
    if (suite->taille == 0)  
        return -1;  
    supp_element = suite->debut;  
    suite->debut = suite->debut->suivant;  
    free (supp_element->donnee);  
    free (supp_element);  
    suite->taille--;  
    return 0;  
}
```

D. Affichage de la file

Pour afficher la file entière, il faut se positionner au début de la file (le pointeur

debut le permettra).

Ensuite en utilisant le pointeur suivant de chaque élément, la file est parcourue du 1er vers le dernier élément.

La condition d'arrêt est donnée par la *taille* de la file.

La fonction

```
void affiche(File *suite){
    Element *courant;
    int i;
    courant = suite->debut;

    for(i=0;i<suite->taille;++i){
        printf(" %s ", courant->donnee);
        courant = courant->suivant;
    }
}
```

E. Récupération de la donnée au début de la file

Pour récupérer la donnée au début de la file sans la supprimer, j'ai utilisé une macro. La macro lit les données au début de la file en utilisant le pointeur debut.

```
#define file_donnee(suite) suite->debut->donnee
```

V. Exemple complet

file.h

```
/* *****\
 *   file.h   *
 \******/
typedef struct ElementListe{
    char *donnee;
    struct ElementListe *suivant;
} Element;

typedef struct ListeRepere{
    Element *debut;
    Element *fin;
    int taille;
} File;

/* initialisation */
void initialisation (File * suite);

/* ENFILER*/
int enfiler (File * suite, Element * courant, char *donnee);

/* DE_FILER*/
int de_filer (File * suite);
```

```

/* FirstInFirstOut */
#define file_donnee(suite) suite->debut->donnee

/* Affiche la file */
void affiche(File *suite);

```

file_function.h

```

/*****\

* file_function.h      *
\*****/

void initialisation (File * suite){
    suite->debut = NULL;
    suite->fin = NULL;
    suite->taille = 0;
}

/* enfiler (ajouter) un élément dans la file */
int enfiler (File * suite, Element * courant, char *donnee){
    Element *nouveau_element;
    if ((nouveau_element = (Element *) malloc (sizeof (Element))) == NULL)
        return -1;
    if ((nouveau_element->donnee = (char *) malloc (50 * sizeof (char)))
        == NULL)
        return -1;
    strcpy (nouveau_element->donnee, donnee);

    if(courant == NULL){
        if(suite->taille == 0)
            suite->fin = nouveau_element;
        nouveau_element->suivant = suite->debut;
        suite->debut = nouveau_element;
    }else {
        if(courant->suivant == NULL)
            suite->fin = nouveau_element;
        nouveau_element->suivant = courant->suivant;
        courant->suivant = nouveau_element;
    }
    suite->taille++;
    return 0;
}

/* de_filer (supprimer) un élément de la file */
int de_filer (File * suite){
    Element *supp_element;
    if (suite->taille == 0)
        return -1;
    supp_element = suite->debut;
    suite->debut = suite->debut->suivant;
    free (supp_element->donnee);
    free (supp_element);
    suite->taille--;
    return 0;
}

```

```

/* affichage de la file */
void affiche(File *suite){
    Element *courant;
    int i;
    courant = suite->debut;

    for(i=0;i<suite->taille;++i){
        printf(" %s ", courant->donnee);
        courant = courant->suivant;
    }
}

```

file.c

```

/*****\

*    file.c    *
\*****/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include "file.h"
#include "file_function.h"

int main ()
{
    File *suite;
    char *nom;
    if ((suite = (File *) malloc (sizeof (File))) == NULL)
        return -1;
    if ((nom = (char *) malloc (50 * sizeof (char))) == NULL)
        return -1;
    initialisation (suite);

    printf ("Entrez un mot : ");
    scanf ("%s", nom);
    enfiler (suite, suite->fin, nom);
    printf ("La file (%d éléments)\n",suite->taille);
    printf("\nDébut de la FILE [ ");
    affiche (suite);    /*le premier entré sera affiché */
    printf(" ] Fin de la FILE\n\n");

    printf ("Entrez un mot : ");
    scanf ("%s", nom);
    enfiler (suite, suite->fin, nom);
    printf ("La file (%d éléments)\n",suite->taille);
    printf("\nDébut de la FILE [ ");
    affiche (suite);    /*le premier entré sera affiché */
    printf(" ] Fin de la FILE\n\n");

    printf ("Entrez un mot : ");
    scanf ("%s", nom);
    enfiler (suite, suite->fin, nom);
    printf ("La file (%d éléments)\n",suite->taille);
    printf("\nDébut de la FILE [ ");
    affiche (suite);    /*le premier entré sera affiché */
    printf(" ] Fin de la FILE\n\n");
}

```

```
printf ("\nLe premier entré (FirstInFirstOut) [ %s ] sera supprimé",
        file_donnee(suite));
printf ("\nLe premier entré est supprime\n");
de_filer (suite);          /* suppression de premier element entre */
printf ("La file (%d éléments): \n",suite->taille);
printf("\nDébut de la FILE [ ");
affiche (suite);
printf(" ] Fin de la FILE\n\n");

return 0;
}
```