Structures de Données: en langage C

Abdellatif HAIR
Université Sultan Moulay Slimane
Faculté des Sciences et Techniques
B.P. 523, Béni-Mellal, MAROC

3. LES PILES ET FILES
représentation dynamique

- PILES : DÉFINITION
- REPRÉSENTATION DES PILES
- OPÉRATIONS DE BASE SUR LES PILES
- FILES : DÉFINITION
- REPRÉSENTATION DES FILES
- OPÉRATIONS DE BASE SUR LES FILES
- APPLICATION

PILES : DÉFINITION

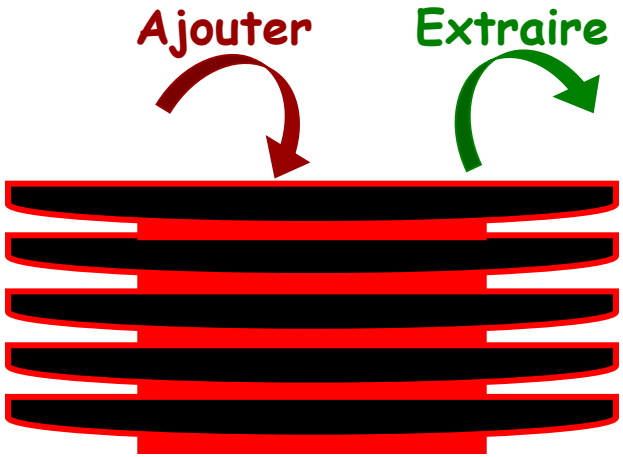
- Les Piles (stack) constituent des structures de données

- Une Pile est une liste linéaire dont une seule extrémité (le sommet) est accessible -visible-

- L'extraction ou l'ajout se font au sommet de la pile

- Une Pile permet de réaliser ce qu'on nomme une LIFO (Last In First Out) : en français *Dernier Entré Premier Sorti*

  - Exemple : une pile d'assiette
    - Lorsqu'on ajoute une assiette en haut de la pile, on retire toujours en premier celle qui se trouve en haut sinon tout le reste s'écroule



REPRÉSENTATION DES PILES

- La Pile en théorie est un objet dynamique (en opposition aux tableaux qui sont des objets statiques). Son état (et surtout sa taille) est variable

- Différentes représentations
  - représentation statique
    - Un tableau, une variable globale indiquant le sommet
    - Un enregistrement avec deux champs
  - représentation dynamique
  - listes chaînées ?

REPRÉSENTATION DES PILES

➔ Structure de la pile

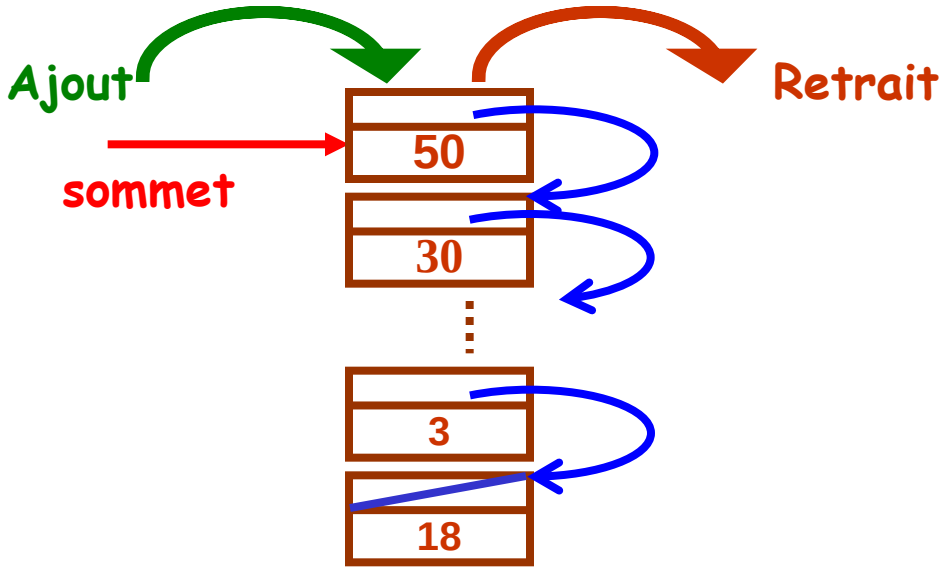
- nous allons créer une pile d'entiers (int)
- notre pile sera basée sur une liste simplement chaînée
- Chaque élément de la pile pointera vers l'élément précédent
- La Pile pointera toujours vers le sommet de la pile

```
typedef struct pile {
        int donnee;  // Donnée que notre pile stockera
 struct pile *precedent;  // Pointeur vers l'élément précédent de la pile
                    } Pile;
```

REPRÉSENTATION DES PILES

➔ Structure de la pile

◆ Pour avoir une idée de ce à quoi ressemblera notre pile, voici un schéma qui pourra vous aider à mieux visualiser



◆ Chaque case représente un élément de la pile. Les cases sont en quelque sorte *emboîtées* les unes sur les autres

◆ Le pointeur sommet devra toujours pointer vers le sommet de la pile

OPÉRATIONS DE BASE SUR LES PILES

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

- le premier élément : se trouve en haut de la pile et permet de réaliser l'opération de récupération et l'ajout des données (sommet)
- le nombre d'éléments : (taille)

◆ On suppose que la pile est déclarée de la façon suivante :

```
typedef struct pile {
        char *donnee;
 struct pile *precedent;
                        } Pile;
/* les variables globales*/
  Pile *sommet;
   int taille;
```

OPÉRATIONS DE BASE SUR LES PILES

➔ Initialisation

Prototype de la fonction : void initialisation ();

- Cette opération doit être faite avant toute autre opération sur la Pile.
- Elle initialise le pointeur sommet avec le pointeur NULL, et la *taille* avec la valeur 0.

Fonction

```
void initialisation () {
                    sommet = NULL;
                    taille = 0;
}
```

OPÉRATIONS DE BASE SUR LES PILES

➔ Ajout d'un nouvel élément

◆ L'ajout d'un nouvel élément se fera à la fin de la Pile

Prototype de la fonction :

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

```
int pile_push(char *donnee);
```

Algorithme de la fonction :

- 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 sommet (le haut de la pile)
- mettre à jour la taille de la pile

OPÉRATIONS DE BASE SUR LES PILES

➔ Ajout d'un nouvel élément

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

```
int pile_push (char *donnee) {
Pile *element;
if ((element = (Pile *) malloc(sizeof(Pile))) ==NULL) return -1;
if ((element->donnee=(char *)malloc(50*sizeof(char)))==NULL) return -1;
strcpy (element->donnee, donnee);
element->precedent = sommet;
sommet = element;
taille++;
return 0;
}
```

OPÉRATIONS DE BASE SUR LES PILES

➔ Retrait d'un élément

◆ L'élément qui sera retiré de la pile sera le dernier élément que l'on a ajouté : LIFO (l'élément se trouve au sommet de la pile)

Prototype de la fonction :

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

```
int pile_pop();
```

Algorithme de la fonction :

- le pointeur supp_elem contiendra l'adresse du pointeur sommet
- le pointeur sommet pointera vers l'élément précédent du pointeur sommet
- la taille de la pile sera décrémentée d'un élément

OPÉRATIONS DE BASE SUR LES PILES

➔ Retrait d'un élément

Fonction /*dépiler (supprimer) un élément de la pile*/.

```
int pile_pop ( ) {
Pile *supp_element;
if (taille == 0)    return -1;
supp_element = sommet;
sommet = sommet->precedent;
free (supp_element->donnee);
free (supp_element);
taille--;
return 0;
}
```

OPÉRATIONS DE BASE SUR LES PILES

➔ Affichage de la pile

- il faut se positionner au sommet de la pile
- En utilisant le pointeur precedent de chaque élément, la pile est parcourue du haut vers le bas
- La condition d'arrêt est donnée par la taille de la pile

```
affiche () {
Pile *courant;  int i;
courant = sommet;
for(i=0;i<taille;++i) { printf("   %s\n", courant->donnee);
                        courant = courant->precedent;
                      }
}
```

OPÉRATIONS DE BASE SUR LES PILES

➔ Vidage de la pile

- Il s'agit d'une fonction permettant d'effacer la Pile

Prototype de la fonction : `pile_clear( );`

Algorithme de la fonction

Tant que le sommet n'est pas nul
effacer l'élément le plus haut de la pile

Fonction

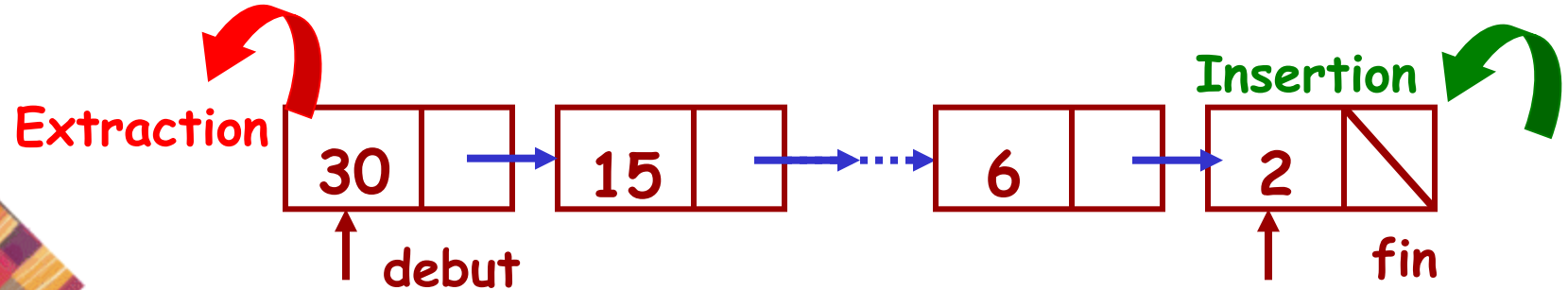
```
pile_clear() { while (sommet != NULL) pile_pop(); }
```

FILES : DEFINITION

- Identiquement aux piles, cette structure est basée sur les listes simplement chaînées

- 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*

- L'ajout d'un élément se fera à la fin (queue, arrière) de la liste (File) et le retrait d'un élément se fera au début (avant, tête) de la liste (File)



REPRESENTATION DES FILES

- La récupération (extraction) des données sera faite dans l'ordre d'insertion. Cependant, on pointera sur la base de la file (sur le premier élément de la file)
- L'ajout (insertion) des données sera faite à partir de la fin de la file. On pointera alors sur la fin de la file
- nous utilisons un pointeur vers l'élément suivant et non plus vers l'élément précédent

```
typedef struct file {
        char *donnee; /* Donnée que notre file stockera */
    struct file *suivant; /* Pointeur vers l'élément suivant de la file
        } File;
```

REPRESENTATION DES FILES

◆ Pour avoir le contrôle de la file, il est préférable de sauvegarder trois éléments :

- le premier élément : se trouve au début de la file et il permet de réaliser l'opération de suppression des données File *debut
- le dernier élément : se trouve à la fin de la file et il permet de réaliser l'opération d'insertion des données File *fin
- le nombre d'éléments : int taille

◆ On suppose que la File est déclarée de la façon suivante :

```
typedef struct file {
                char *donnee;
            struct file *suivant;
                        } File;
/* les variables globales*/
   File *debut, *fin;
    int taille;
```

OPÉRATIONS DE BASE SUR LES FILES

➔ Initialisation

Prototype de la fonction : void initialisation ();

- Cette opération doit être faite avant toute autre opération sur la File.
- Elle initialise le pointeur *debut* et *fin* avec le pointeur NULL, et la *taille* avec la valeur 0

Fonction

```
void initialisation () {
                debut = NULL;
                  fin = NULL;
               taille = 0;
}
```

OPÉRATIONS DE BASE SUR LES FILES

➔ Insertion d'un nouvel élément

Prototype de la fonction :

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

```
int file_push(char *donnee);
```

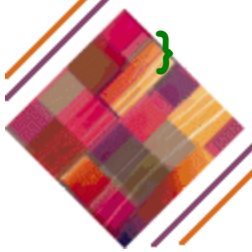
Algorithme de la fonction :

- déclaration d'élément(s) à insérer
- allocation de la mémoire pour le nouvel élément
- remplir le contenu du champ de donnée
- si la file est vide le pointeur debut pointera vers le nouvel élément (-insertion dans une file vide-)
- sinon le pointeur suivant du dernier élément (fin) pointera vers le nouvel élément (-insertion vers la fin dans la file non vide-)
- mettre à jour le pointeur fin
- mettre à jour la taille de la file

OPÉRATIONS DE BASE SUR LES FILES

➔ Insertion d'un nouvel élément

```
int file_push(char *donnee) {
File *element;
if ((element=(File *) malloc (sizeof(File))) ==NULL) return -1;
if ((element->donnee=(char *)malloc (50*sizeof(char)))==NULL) return -1;
strcpy (element->donnee, donnee);
element->suivant = NULL;
if (taille == 0) debut = element;
else fin->suivant = element;
fin = element;
taille++;
return 0;
}
```



OPÉRATIONS DE BASE SUR LES FILES

➔ Suppression d'un élément

- Pour supprimer 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, mais seulement de la supprimer.

Algorithme de la fonction :

- le pointeur supp_elem contiendra l'adresse du 1er élément (debut)
- le pointeur debut pointera vers le 2ème élément (après la suppression du 1er élément, le 2ème sera au début de la file)
- mettre à jour le pointeur fin dans le cas où la file contient 1 seul élément
- la taille de la file sera décrémentée d'un élément

OPÉRATIONS DE BASE SUR LES FILES

➔ Suppression d'un élément

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

```
int file_pop () {
File *supp_element;
if (taille == 0)    return -1;
supp_element = debut;
debut = debut->suivant;
if (debut == NULL) fin=NULL;      // if (taille ==1) fin=NULL;
free (supp_element->donnee);
free (supp_element);
taille--;
return 0;
}
```

OPÉRATIONS DE BASE SUR LES FILES

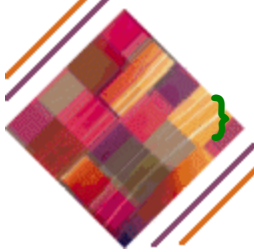
➔ Affichage de la file

Algorithme de la fonction

- il faut se positionner au debut de la file
- En utilisant le pointeur suivant de chaque élément, la file est parcourue du debut vers la fin
- La condition d'arrêt est donnée par la taille de la file

Fonction

```
void affiche () {
File *courant;  int i;
courant = debut;
for(i=0;i<taille;++i) { printf("   %s\n", courant->donnee);
                        courant = courant->suivant;  }
}
```



OPÉRATIONS DE BASE SUR LES FILES

➔ Vidage de la file

- Il s'agit d'une fonction permettant d'effacer la File

Algorithme de la fonction

Tant que le pointeur debut n'est pas NULL
Effacer le premier élément de la file

Fonction

```
file_clear() { while (debut != NULL) file_pop(); }
```

APPLICATION

Exercice :

Ecrire le programme C qui crée une file dynamique d'entiers FD et de le charger par des nombres positifs. Changer la file FD en mettant en tête les nombres pairs dans le même ordre et ensuite les nombres impairs en ordre inverse.

Par exemple

si FD=(3,4,2,9,8,5,1,7,6) alors FD=(4,2,8,6,7,1,5,9,3)

APPLICATION

```
# include <stdio.h>
# define max 50

typedef struct file {
            int donnee ;
          struct file * suivant ;
              } File;

     /* les variables globales*/
      File *debut, *fin;
      int taille;
```

APPLICATION

```
int file_push(int donnee) {
File *nouvel_element;
if ((nouvel_element=(File *) malloc (sizeof(File))) ==NULL) return -1;
nouvel_element->donnee=donnee;
nouvel_element->suivant = NULL;
if (taille == 0)  debut = nouvel_element;
else fin->suivant = nouvel_element;
fin = nouvel_element;
taille++;
return 0;
}
```

APPLICATION

```
int file_pop() {
File *supp_element;
int info;
if (taille == 0) {printf("file vide \n");   return -1; }
supp_element = debut;
debut = debut->suivant;
if  (taille == 1) fin=NULL;
info=supp_element->donnee;
free (supp_element);
taille--;
return info;
}
```

APPLICATION

```
void affiche () {
File *courant;  int i;
if (taille==0) printf("la file est vide \n");
else {  courant = debut;
        for(i=1; i<=taille; i++) {
                printf("   %d\n", courant->donnee);
                courant = courant->suivant;
         }
}
}

void initialiser() {
    taille=0; fin=NULL; debut=NULL;
}
```

APPLICATION

```
Ordre_file(){
    int i, j, s, k, ct, tab[max];
    if (taille==0)  printf("la file est vide \n ");
    else {    i=1; ct=taille ; k=-1;
              while(i<=ct) {
                   s=file_pop();
                   if (s%2==0) file_push(s); else { k++; tab[k]=s; }
                    i++;
                }
              for (j=k; j>=0; j--) file_push(tab[j]);
     }
}
```

APPLICATION

```
main(){
initialiser();
file_push(1); file_push(2); file_push(3); file_push(4);
file_push(5); file_push(6); file_push(7);
Ordre_file() ;
affiche();
system("pause");
}
```