“ Algorithmiques avancée ”

3- Structures de Données Linéaires

Pr. Abdelhay HAQIQ (ahaqiq@esi.ac.ma)

Plan

- **Structures**
- **Piles**
- **Files**
- **Listes chaînées**

Plan

- **Structures**
- Piles
- Files
- Listes chaînées

Structures

- Le langage C possède plusieurs types prédéfinis pour les variables (int, float, char…)
- Les types prédéfinis peuvent être insuffisants à répondre aux besoins de l'utilisateur
  - Il est possible de définir son propre type
  - Il est possible de donner un autre nom à un autre type (prédéfini ou défini par l'utilisateur)

Type structure

- Une structure est un **nouveau type de données** composé de plusieurs champs (ou membres) qui sert à représenter un objet.
- Chaque champ est de type quelconque pouvant être lui aussi une structure
- Le nom d'une structure n'est pas un nom de variable (c'est un nom de type)

```
struct <nom_structure>
{
    <type> <nom_champ1>;
    <type> <nom_champ2>;
     …
    <type> <nom_champN>;
};
```

Type structure

```c
struct personne
{
    int numero;
    char *nom;
    char prenom[20];
};

int main()
{
    struct personne p1;

    p1.numero = 1;
    p1.nom = "HA";
    strcpy(p1.prenom, "Ahmed");

    printf("%d %s %s", p1.numero, p1.nom, p1.prenom);

 return 0;

}
```

Types synonymes

- Les types synonymes donnent un autre nom à un autre type (prédéfini ou défini par l'utilisateur) avec le mot-clé **typedef** dans le but alléger l'écriture du code

```
typedef <type> <nouveau_nom>;
```

- On peut faire un typedef sur un pointeur comme suit :

```
typedef <type*> <type_ptr>;
```

  - Les deux écritures suivantes deviennent équivalentes :

```
type* variable;

type_ptr variable;
```

Type synonyme

```c
typedef int entier;
typedef float point[2];

int main()
{
    entier a = 4;
    point p;

    p[0] = 1.5;
    p[1] = 2.4;

    printf("Entier %d\n", a);
    printf("Point x : %f Point y : %f \n",  p[0],  p[1]);

 return 0;

}
```

Type structure & Type synonyme

```c
typedef struct personne
{
    int numero;
    char *nom;
    char prenom[20];
} personne;

int main()
{
    personne p1;

    p1.numero = 1;
    p1.nom = "HA";
    strcpy(p1.prenom, "Ahmed");

    printf("%d %s %s", p1.numero, p1.nom, p1.prenom);

 return 0;

}
```

Structure nommée

Accès aux champs d'un pointeur d'une structure

- L'accès aux champs d'un pointeur d'une structure s'effectue ainsi :

```
(*<pointeur_structure>).<champ_structure>;

OU

<pointeur_structure>-><champ_structure>;
```

Opérations sur les variables d'une structure

```
int main()
{
    personne p1, p2,*p3;
    p1.numero = 1;
    p1.nom = "Nom1";
    strcpy(p1.prenom,"Prenom1");

    p2 = p1;
    printf("P1: %d %s %s\n", p1.numero, p1.nom, p1.prenom);
    printf("P2: %d %s %s\n", p2.numero, p2.nom, p2.prenom);
 printf("---------------------------\n");
    p2.numero = 2;
    p2.nom = "Nom2";
    printf("P1: %d %s %s\n", p1.numero, p1.nom, p1.prenom);
    printf("P2: %d %s %s\n", p2.numero, p2.nom, p2.prenom);
 printf("---------------------------\n");
     p3 = &p1;
    (*p3).numero = 2;
     p3->nom = "Nom2";
    printf("P1: %d %s %s\n", p1.numero, p1.nom, p1.prenom);
    printf("P3: %d %s %s\n", p3->numero, (*p3).nom, p3->prenom);

 return 0;
}
```

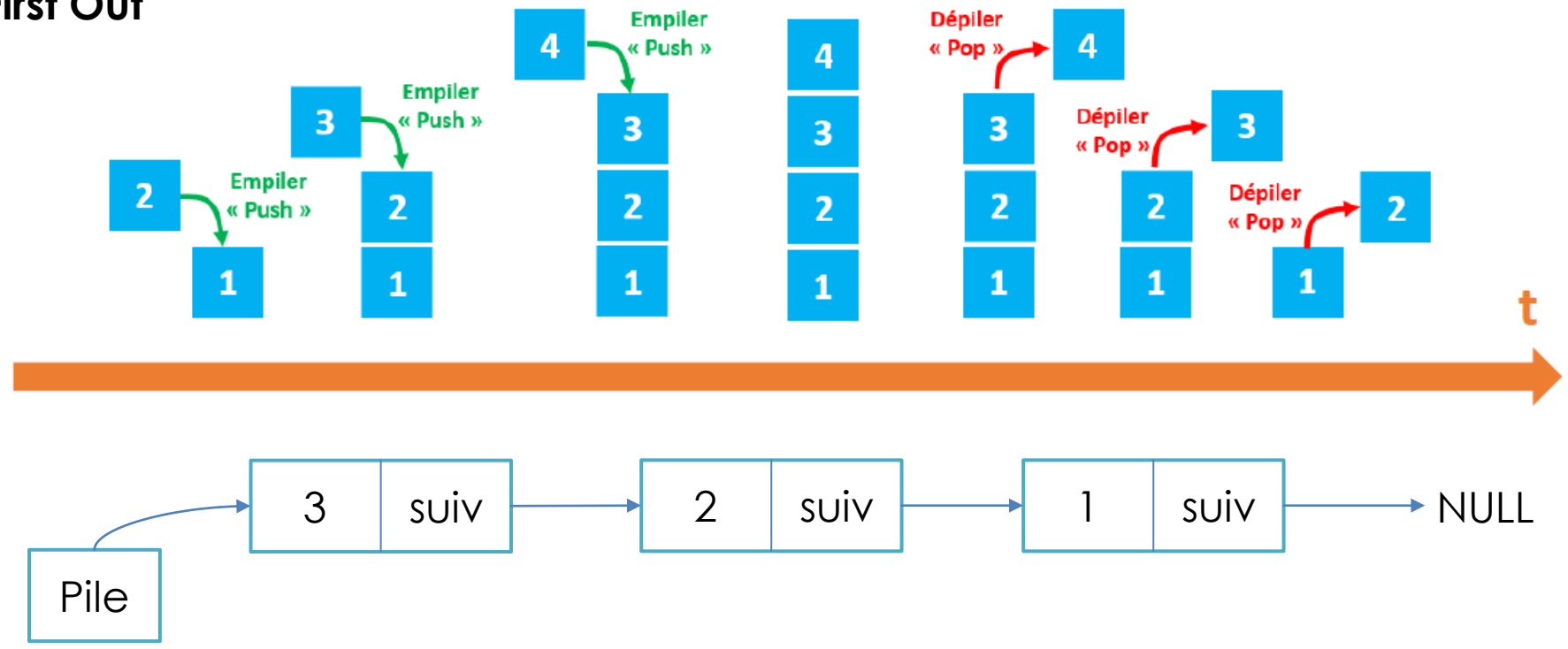
```
P1 : 1 Nom1 Prenom1
P2 : 1 Nom1 Prenom1

P1 : 1 Nom1 Prenom1
P2 : 2 Nom2 Prenom1

P1 : 2 Nom2 Prenom1
P3 : 2 Nom2 Prenom1
```

Plan

- Structures
- **Piles**
- **Files**
- **Listes chaînées**

Piles

Last In First Out



Pile

```c
typedef int Element;  /* les éléments sont des int */

//Pile Chainee
typedef struct cellule {
 Element valeur;
 struct cellule *suivant;
} Cellule;

typedef Cellule* Pile;

// Déclaration des fonctions gérant la pile
Pile init_pile();
Pile empiler( Pile p, Element e );
Pile depiler( Pile p );
Element sommet( Pile p );
int est_vide(Pile p);
```

```c
// Création d'une cellule
Cellule * pc;
Cellule * pc=(Cellule *)malloc(sizeof(Cellule));
// OU
Pile pc;
Pile pc=(Pile)malloc(sizeof(Cellule));
```

Initialisation et Création Pile

```
Pile init_pile()
{
    return NULL;
}

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

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

    return pc;
}
```

Suppression Elément Pile

```
Pile depiler(Pile p)
{
    /* pre-condition: pile non vide ! */
    Cellule * pc = p;
    if (est_vide(p))
    {
        printf("Erreur: pile vide !\n");
        return NULL;
    }
    p=p->suivant;
    free(pc);
    return p;
}
```

Affichage Pile

```c
void afficher (Cellule * p)
{
    printf("%p\n", p);
    while(p)
    {
        printf("%d \t", p->valeur);
        p =p->suivant;
        printf("%p\n", p);
    }
    printf("%p\n", p);
}
```

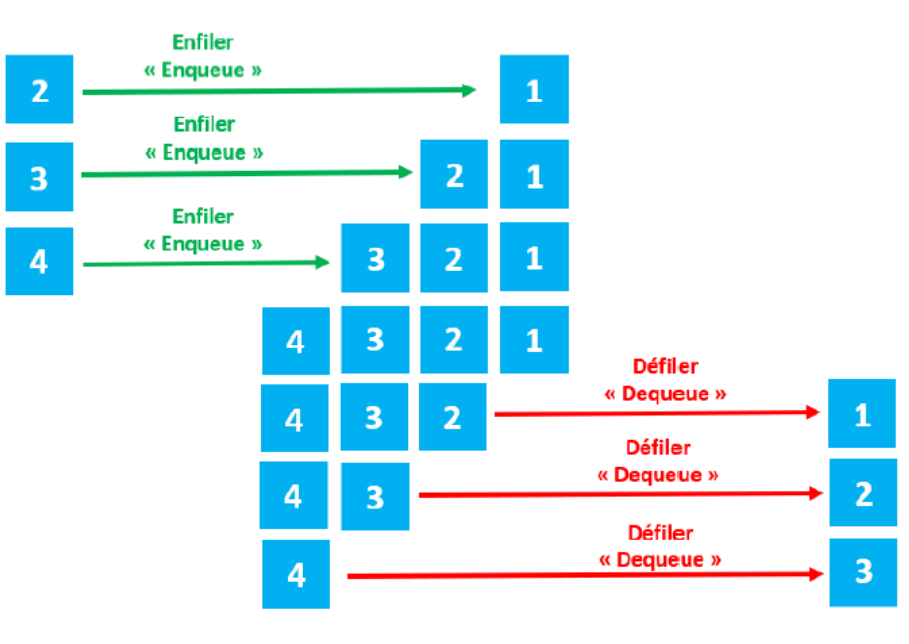
```c
Element sommet(Pile p)
{
    /* pré-condition: pile non vide ! */
    if (est_vide(p))
    {
        printf("Erreur: pile vide !\n");
        return NULL;
    }
    return p->valeur;
}
```

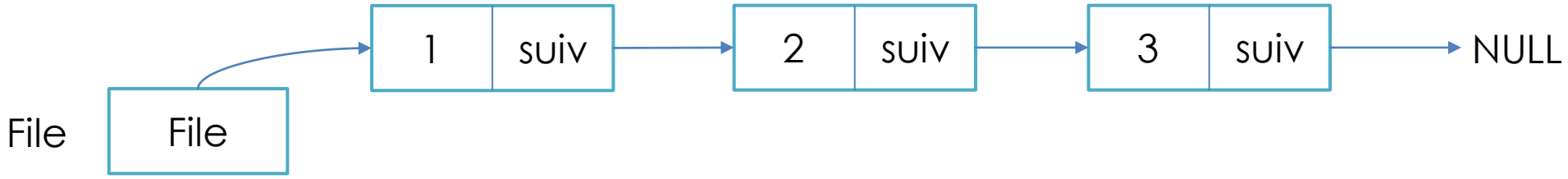
Main Pile

```c
int main()
{
    Pile p = init_pile();

    p = empiler(p,50);
    p = empiler(p,5);
    p = empiler(p,20);
    p = empiler(p,10);

    printf("%d au sommet apres empilement de 50, 5, 20 et 10\n", sommet(p));

    p = depiler(p);
    p = depiler(p);

    printf("%d au sommet apres depilement de 10 et 20\n", sommet(p));
    return 0;
}
```

Files

First In First Out

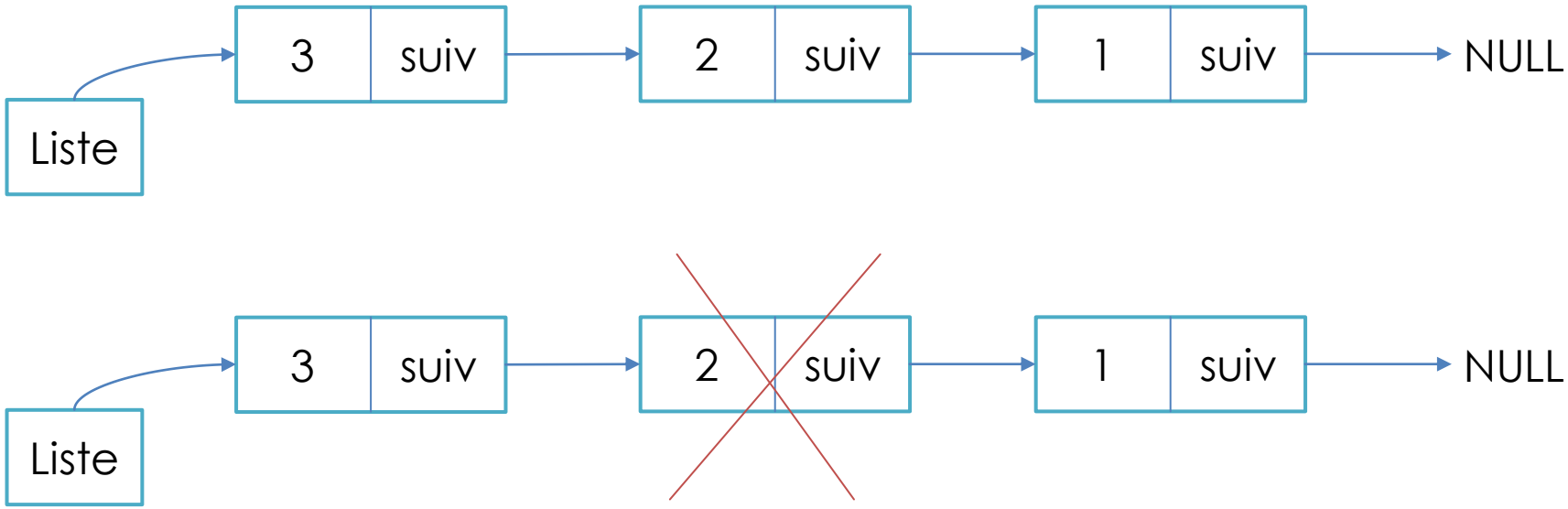




Files

```c
typedef int Element; /* les éléments sont des int */

typedef struct cellule
{
    Element valeur;
    struct cellule* suivant;
} Cellule;


typedef Cellule* File;

// Déclaration des fonctions gérant la file
File init_file ();
File enfiler ( File f, Element e );
File defiler ( File f );
Element sommet ( File f );
int est_vide ( File f );
```

Liste chainée



Liste chainée

```
typedef int Element; /* les éléments sont des int */

typedef struct cellule
{
    Element valeur;
    struct cellule* suivant;
} Cellule;


typedef Cellule* Liste;

// Déclaration des fonctions gérant la liste
Liste init_liste();
Liste inserer( Liste l, Element e );
Liste supprimer( Liste l, Element e );
afficher( Liste l );
int est_vide ( Liste l );
```