Piles

Définition et propriétés

- Une pile $\mathcal{P}$ est une structure qui permet de stocker une suite d'éléments de même type et qui possède les propriétés suivantes :
  - l'existence d'un élément particulier appelé Sommet de la pile
  - l'existence pour tout élément, à l'exception de la base, d'un Prédecesseur ;
- Une pile est une liste particulière sur laquelle les opérations d'ajout, de suppression et de consultation ne sont possibles que sur le Sommet
- Le Sommet est le dernier élément ajouté dans la pile, il est aussi le premier élément à en être retiré.

Définition et propriétés

- Quand on ajoute un élément dans une pile, celui-ci devient le sommet de la pile, c'est-à-dire le seul élément accessible.
- Quand on retire un élément de la pile, on retire toujours le sommet, et le dernier élément ajouté avant lui devient alors le sommet de la pile.
- Cette structure est également appelée aussi une liste LIFO (Last In, First Out).
- Empiler : signifie ajouter un élément au sommet de la pile
- Dépiler : signifie supprimer l'élément du sommet de la pile

Description fonctionnelle de la pile

- Soit **Telement** le type de base des éléments de la pile et soit **TPile** un ensemble d'éléments de type **Telement** doté d'une structure de pile .
- Les opérations qu'on peut effectuer sur les piles sont nombreuses, nous ne citons que les plus fondamentales :

- CreerPile
- PileVide
- Hauteur

- Empiler
- Dépiler
- Sommet

Description fonctionnelle de la pile

Nom opération	Domaine	**Nature & Description**
Construction		
CreerPile	→ **TPile**	Création d'une pile vide
Modification		
Empiler	**TPile** x **Telement** → **TPile**	Ajouter un élément
Depiler	**TPile** → **TPile**	Suppression d'un élément
Utilisation		
PileVide	**TPile** → **Booleen**	
HautPile	**TPile** → **Entier**	Hauteur de la pile
Sommet	**TPile** → **Telement**	Consulter le sommet

Description au niveau logique

Une pile est soit vide, soit non vide.

- Une pile vide est une pile qui ne contient aucun élément.
- Les éléments d'une pile non vide ne sont pas obligés d'être contiguës dans la mémoire.

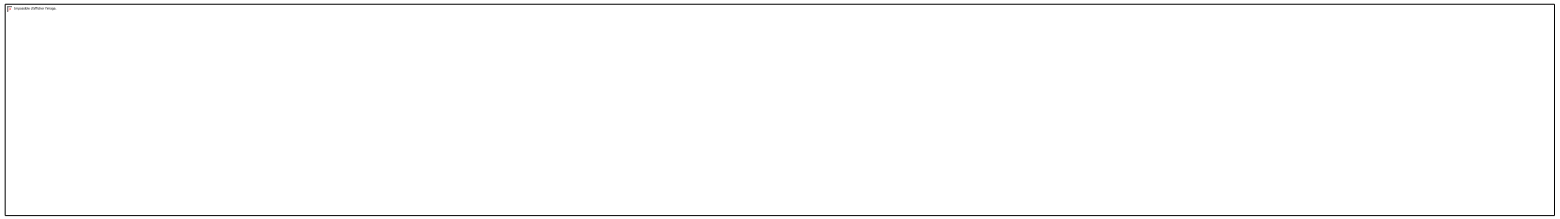
- Pour la représentation non contiguë, nous devons mettre en place un dispositif qui permet à chaque élément de désigner son prédécesseur.

Description au niveau physique
Modélisation contiguë

- Cette méthode utilise un tableau pour implémenter une structure de pile : les éléments de la pile occupent donc des emplacements contiguës dans la mémoire.
- Le tableau des éléments peut être statique ou dynamique.
- Nous avons besoin de connaitre à tout moment la position du sommet dans le tableau d'où la nécessité de stocker l'indice du sommet
- La structure est composée d'un champ entier pour l'indice du sommet et d'un champ tableau.
- L'indice du **Sommet** doit être stocké dans la structure .

Description au niveau physique : Modélisation contiguë statique

- il est nécessaire d'utiliser Nmax la hauteur maximale de la pile. Pour un tableaux statique, Nmax sera une constante



Contiguë statique

```
struct Pile {
  Telement P[Nmax];
  int iSommet  ;
};

typedef struct Pile TPile ;
```

Primitives sur les piles
Modélisation contiguë statique

- La fonction Creerpile n'est pas nécessaire dans le cas de tableau statique (mais iSommet doit être initialisé par -1)

```
int PileVide(TPile pile){
   return(pile.iSommet<0);
}
```

```
int HautPile(TPile pile) {
  return pile.iSommet+1;
};
```

```
int PilePleine(TPile pile){
  return(pile.iSommet==Nmax-1);
}
```

```
Telement Sommet(TPile pile){
if (!PileVide(pile))
      return pile.P[pile.iSommet]);
}
```

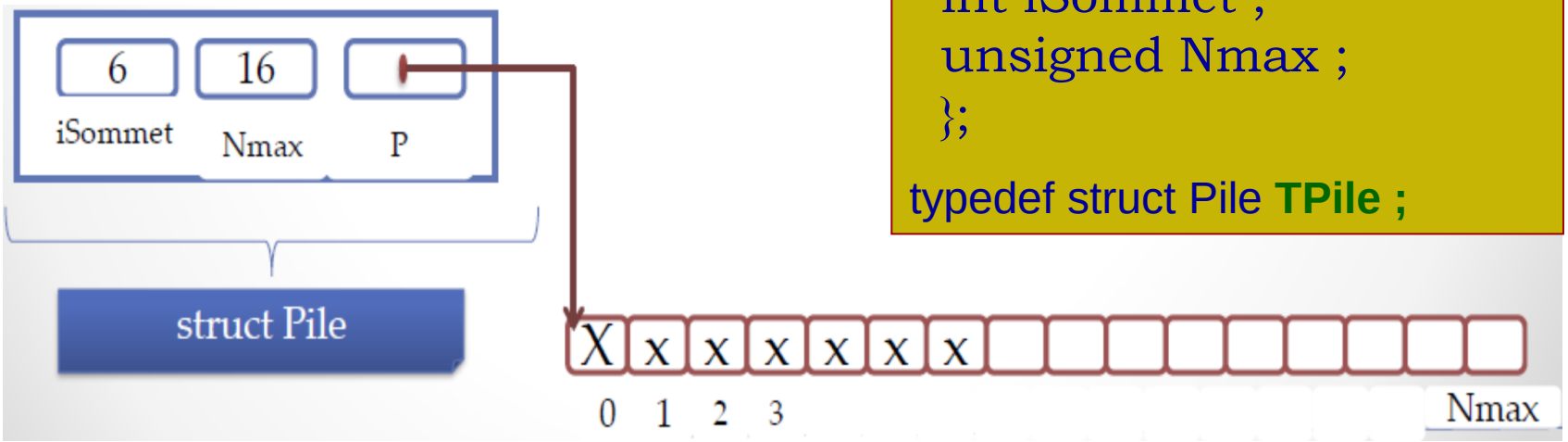
Primitives sur les piles
Modélisation contiguë statique

```
int Empiler(TPile*pPile, Telement e)
{
   if(PilePleine(*pPile))
              return 0;
  else{
        pPile->iSommet++;
        pPile->P[pPile-> iSommet] = e;
        return 1 ;
        }
};
```

```
Telement Depiler(TPile*pPile)
{
if (! PileVide(*pPile))
  {
     pPile->iSommet--;
    return pPile->P[pPile->iSommet+1];
  }
};
```

Description au niveau physique : Modélisation contiguë dynamique

- il est nécessaire d'utiliser Nmax: la hauteur maximale de la pile.




```
struct Pile {
  Telement *P;
  int iSommet ;
  unsigned Nmax ;
  };
typedef struct Pile TPile ;
```

Primitives sur les piles
Modélisation contiguë dynamique

- La fonction **CreerPile** est nécessaire dans ce cas de même que la fonction **liberePile**

```
Void CreerPile(unsigned capacite, TPile*pPile)
{ pPile->Nmax=capacite;
   pPile->P=malloc(Nmax*sizeof(Telement));
   pPile->iSommet=-1;
  return;
}
```

```
Void LibererPile(TPile*pPile)
{ pPile->Nmax=0;
     free(pPile->P)
     pPile->iSommet=-1;
       return;
}
```

Primitives sur les piles
Modélisation contiguë dynamique

```
intPileVide(TPilepile){
   return(pile.iSommet<0);
}
```

```
int HautPile(TPilepile) {
  return pile.iSommet+1;
};
```

```
intPilePleine(TPilepile){
  return(pile.iSommet==Nmax-1);
}
```

```
TelementSommet(TPilepile){
if (!PileVide(pile))
      returnpile.P[pile.iSommet]);
}
```

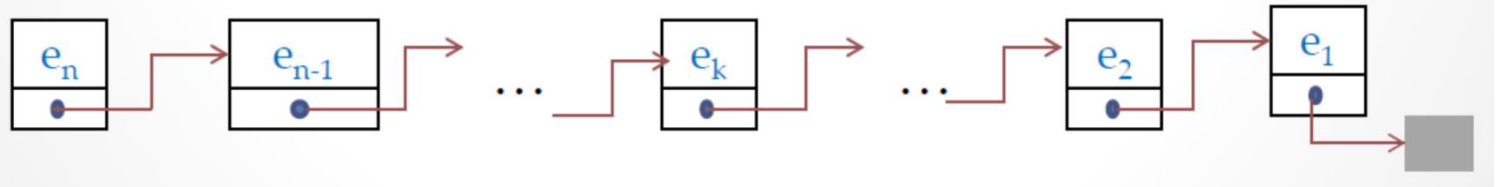
Primitives sur les piles
Modélisation contiguë dynamique

```
int Empiler(TPile*pPile, Telemente)
{
   if(PilePleine(*pPile))
              return 0;
  else{
        pPile->iSommet++;
        pPile->P[pPile-> iSommet] = e;
        return 0 ;
        }
};
```

```
Telement Depiler(TPile*pPile)
{
 if (! PileVide(*pPile))
  {
     pPile->iSommet--;
    return pPile->P[pPile->iSommet+1];
} ;
```

Description au niveau physique Modélisation non contiguë

- Les éléments d'une pile non vide ne sont pas contiguës dans la mémoire.
- Nous devons donc mettre en place un dispositif qui permet à chaque élément de désigner son suivant.
- Nous utilisons le notion de **maillon**: Un **maillon** étant une **structure** qui contient deux parties: la valeur de l'élément et un indicateur (pointeur ou indice) sur le maillon suivant.



Description au niveau physique
Modélisation chaînée

Un maillon est une structure comportant deux champs:

1. L'élément : du type Telement(int, double,…),
2. Un pointeur vers le maillon suivant.

```
struct maillon {
      Telement e  ;
   struct maillon *pSuivant ;
   };
typedef struct maillon Tmaillon;
```

La pile est déterminée par l'adresse de son sommet : donc c'est une structure contenant un seul champ

```
struct Pile {
          Tmaillon *pSommet ;
              };
typedef struct Pile TPile;
```

Primitives sur les piles
Modélisation chaînée (non contiguë)

```
CreerPile(TPile*pPile) {
   pPile->pSommet=NULL;
  return;
}
```

```
int PileVide(TPile pile){
   return(pile.pSommet== NULL);
}
```

```
Telement Sommet(TPile pile){
if (!PileVide(pile))
    return (pile.pSommet)->e;
};
```

```
int HautPile(TPile pile) {
if(PileVide(pile))
    return 0;
int h=1;
Tmaillon*m=pile.pSommet;
while(m->pSuivant!=NULL)
 {
  m=m->pSuivant;
   h++;
  }
  return h;
};
```

Primitives sur les piles
Modélisation chaînée (non contiguë)

```
Tmaillon*CreerMaillon( Telement valeur){
        Tmaillon*pm= malloc(sizeof(Tmaillon));
        if (pm!=NULL) {
                pm->e = valeur;
                pm->pSuivant= NULL;
         }
        return pm;
}
```

```
int Empiler(TPile*pPile, Telement valeur) {
        Tmaillon*pm=CreerMaillon(valeur);
        if (pm==NULL) return 0;
        pm->pSuivant=pPile->pSommet;
        pPile->pSommet=pm;
        return 1;
}
```

Primitives sur les piles
Modélisation chaînée (non contiguë)

```
Telement Depiler(TPile*pPile){
        Telement valeur;
        Tmaillon*AncienSommet;
        if (!PileVide(*pPile)) {
                valeur= pPile-> pSommet->e;
                AncienSommet= pPile->pSommet;
                pPile->pSommet= AncienSommet->pSuivant;
                AncienSommet->pSuivant= NULL;
                free(AncienSommet);
                return valeur;
        }
} ;
```