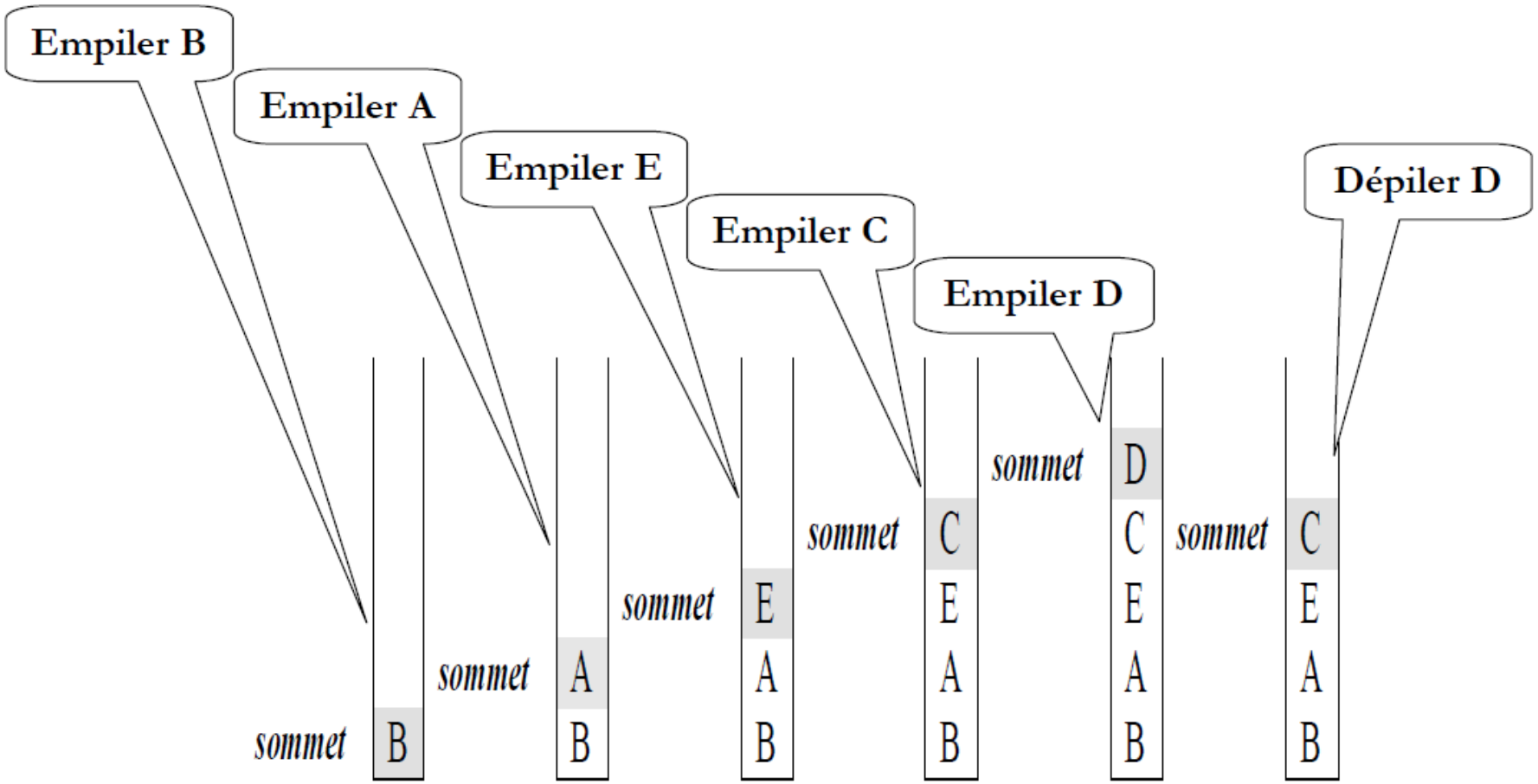
Structures de données linéaires

Piles

Notion de Pile (Stack)

- Les piles sont très utilisées en informatique
- Notion intuitive :
  - pile d'assiettes, pile de dossiers à traiter, ...
- Une pile est une structure linéaire permettant de stocker et de restaurer des données selon un ordre LIFO (Last In, First Out ou « dernier entré, premier sorti »)
- Dans une pile :
  - Les insertions (empilements) et les suppressions (dépilements) sont restreintes à une extrémité appelée sommet de la pile.

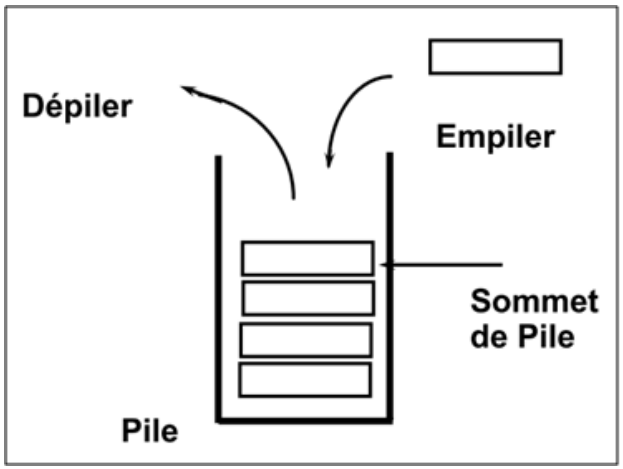
Exemple de pile



Opérations de la Pile

- Les opérations autorisées pour la pile sont :

1. consulter le dernier élément de la pile (le sommet)$\Longrightarrow$ TOP
2. tester si la pile est vide
3. empiler un élément, le mettre au sommet de la pile $\Longrightarrow$ PUSH
4. dépiler un élément (par le sommet) $\Longrightarrow$ POP



Type Abstrait Pile

Type Pile
Utilise Élément, Booléen
Opérations
pile_vide : $\rightarrow$ Pile
est_vide : Pile $\rightarrow$ Booléen
empiler : Pile $\times$ Élément $\rightarrow$ Pile
dépiler : Pile $\rightarrow$ Pile
sommet : Pile $\rightarrow$ Élément
Préconditions
dépiler(p) est-défini-ssi est_vide(p) = faux
sommet(p) est-défini-ssi est_vide(p) = faux
Axiomes
Soit, e : Element, p : Pile
est_vide(pile_vide) = vrai
est_vide(empiler(p,e)) = faux
dépiler(empiler(p,e)) = p
sommet(empiler(p,e)) = e

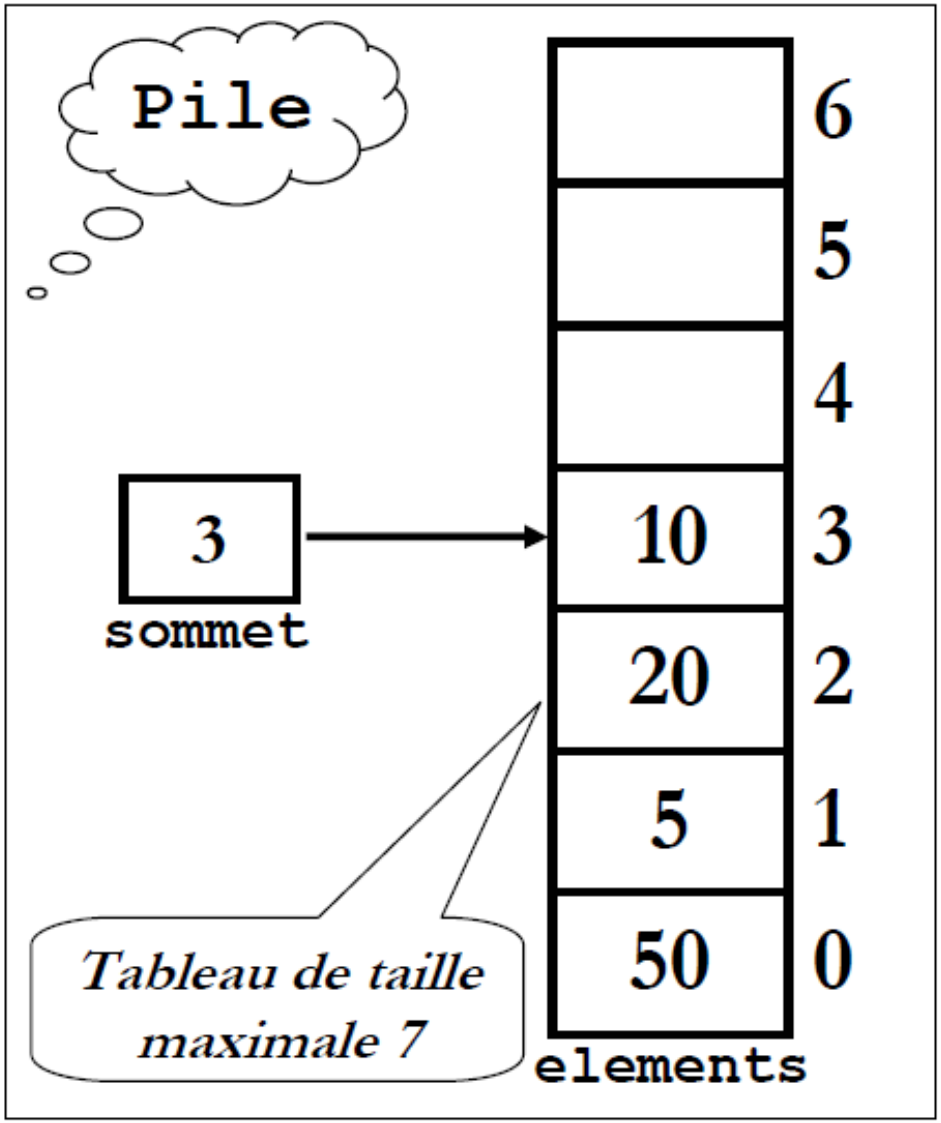
Opérations sur une Pile

- pile_vide : $\rightarrow$ Pile
  - opération d'initialisation ; la pile créée est vide
- est_vide : Pile $\rightarrow$ Booléen
  - teste si pile vide ou non
- sommet : Pile $\rightarrow$ Élément
  - permet de consulter l'élément situé au sommet ; n'a pas de sens si pile vide
- empiler : Pile $\times$ Élément $\rightarrow$ Pile
  - ajoute un élément dans la pile
- dépiler : Pile $\rightarrow$ Pile
  - enlève l'élément situé au sommet de la pile ; n'a pas de sens si pile vide

Représentation d'une Pile

- Représentation contiguë (par tableau) :
  - Les éléments de la pile sont rangés dans un tableau
  - Un entier représente la position du sommet de la pile
- Représentation chaînée (par pointeurs) :
  - Les éléments de la pile sont chaînés entre eux
  - Un pointeur sur le premier élément désigne la pile et représente le sommet de cette pile
  - Une pile vide est représentée par le pointeur NULL

Pile Contiguë (Contiguous Stack)



Spécification d'une Pile Contiguë

```
/* fichier "Tpile.h" */
#ifndef _PILE_TABLEAU
#define _PILE_TABLEAU
#include "Booleen.h"
// Définition du type Pile (implémentée par un tableau)
#define MAX_PILE 7 /* taille maximale d'une pile */
typedef int Element; /* les éléments sont des int */
typedef struct {
 Element elements[MAX_PILE]; /* les éléments de la pile */
 int sommet; /* position du sommet */
} Pile;
// Déclaration des fonctions gérant la pile
Pile pile_vide ();
Pile empiler (Pile p, Element e);
Pile depiler (Pile p);
Element sommet (Pile p) ;
Booleen est_vide (Pile p);
#endif
```

Réalisation d'une Pile Contiguë (1)

```
/* fichier "Tpile.c" */
#include "Tpile.h"
// Définition des fonctions gérant la pile
// initialiser une nouvelle pile
Pile pile_vide() {
 Pile p;
 p.sommet = -1;
 return p;
}
// tester si la pile est vide
Booleen est_vide(Pile p) {
 if (p.sommet == -1) return vrai;
 return faux;
}
// Valeur du sommet de pile
Element sommet(Pile p) {
/* pré-condition : pile non vide ! */
 if (est_vide(p)) {
  printf("Erreur: pile vide !\n"); exit(-1);}
 return (p.elements)[p.sommet];}
```

Réalisation d'une Pile Contiguë (2)

```
 // ajout d'un élément
Pile empiler(Pile p, Element e) {
 if (p.sommet >= MAX_PILE-1) {
  printf("Erreur : pile pleine !\n");
  exit(-1);
 }
 (p.sommet)++;
 (p.elements)[p.sommet] = e;
 return p;
}
// enlever un élément
Pile depiler(Pile p) {
/* pré-condition : pile non vide !*/
 if (est_vide(p)) {
  printf("Erreur: pile vide !\n");
  exit(-1);}
 p.sommet--;
 return p;
}
```

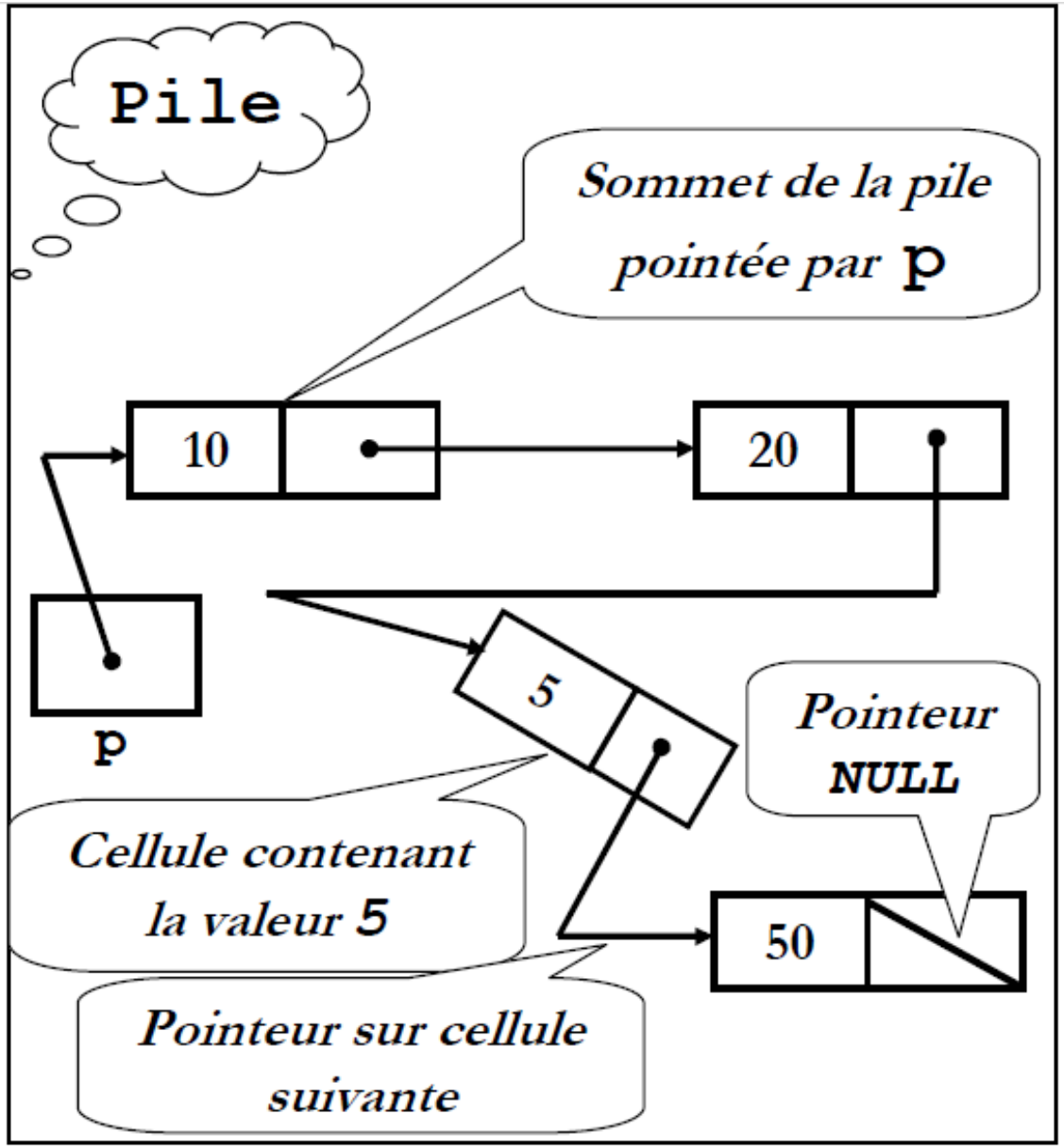
Exemple d'utilisation d'une Pile Contiguë

```c
/* fichier "UTpile.c" */
#include <stdio.h>
#include "Tpile.h"

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;
}
```

Pile Chaînée



Spécification d'une Pile Chaînée

```
/* fichier "Cpile.h" */
#ifndef _PILE_CHAINEE
#define _PILE_CHAINEE
#include "Booleen.h"
// Définition du type Pile (implémentée par pointeurs)
typedef int Element; /* les éléments sont des int */
typedef struct cellule {
 Element valeur;
 struct cellule *suivant;
} Cellule;
typedef Cellule *Pile;
// Déclaration des fonctions gérant la pile
Pile pile_vide ();
Pile empiler (Pile p, Element e);
Pile depiler (Pile p);
Element sommet (Pile p);
Booleen est_vide (Pile p);
#endif
```

Réalisation d'une Pile Chaînée (1)

```c
/* fichier "Cpile.c" */
#include "Cpile.h"
Pile pile_vide(void) {
 return NULL;
}
Booleen est_vide(Pile p) {
 return (p == NULL);
}
Element sommet(Pile p) {
/* pré-condition: pile non vide ! */
 if (est_vide(p)) {
   printf("Erreur: pile vide !\n");
   exit(-1);
 }
 return p->valeur;
}
```

Réalisation d'une Pile Chaînée (2)

```
Pile empiler(Pile p, Element e) {
 Cellule * pc;
 pc=(Cellule *)malloc(sizeof(Cellule));
 pc->valeur=e;
 pc->suivant=p;
 p=pc;
 return p;
}
Pile depiler(Pile p) {/* pré-condition: pile non vide ! */
 if (est_vide(p)) {
   printf("Erreur: pile vide !\n");
   exit(-1);
 }
 Cellule * pc = p;
 p=p->suivant;
 free(pc);
 return p;
}
```

Exemple d'utilisation d'une Pile Chaînée

```c
/* fichier "UCpile.c" */
#include <stdio.h>
#include "Cpile.h"

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;
}
```

Exemples d'application d'une Pile

- Vérification du bon équilibrage d'une expression parenthèsée :
  - Pour vérifier qu'une expression parenthèsée est équilibrée, à chaque rencontre d'une parenthèse ouvrante on l'empile et à chaque rencontre d'une parenthèse fermante on dépile ;
- Evaluation des expressions arithmétiques postfixées (expressions en notation polonaise inverse) :
  - Pour évaluer une telle expression, on applique chaque opérateur aux deux opérandes qui le précédent. Il suffit d'utiliser une pile dans laquelle les opérandes sont empilés, alors que les opérateurs dépilent deux éléments, effectuent l'opération et empilent le résultat. Par exemple, l'expression postfixée 2 3 5 * + 1 – s'évalue comme suit : ((2 (3 5 *) +) 1 –) = 16 ;
- Conversion d'une expression en notation infixe (parenthèsée) en notation postfixée

Exemples d'application d'une Pile

- Gestion par le compilateur des appels de fonctions :
  - les paramètres, l'adresse de retour et les variables locales sont stockées dans la pile de l'application
- Mémorisation des appels de procédures imbriquées au cours de l'exécution d'un programme, et en particulier les appels des procédures récursives ;
- Parcours « en profondeur » de structures d'arbres (voir chapitre arbres) ;