Les piles

D.E ZEGOUR

École Supérieure d'Informatique

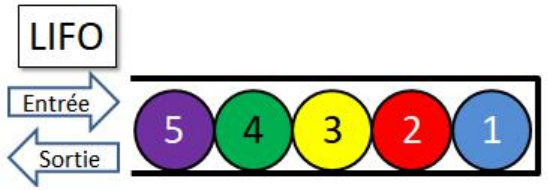
ESI

Les piles

Définition, principe, domaine d'application

Concept parmi les plus utilisés dans la science des ordinateurs

Collection d'éléments
- Tout nouveau élément est inséré à la fin
- Tout retrait se fait également de la fin.



Principe LIFO,
"Last In First Out" // Dernier entré, Premier servi

Utilisée dans le domaine de la compilation :
- résolution de la portée des objets,
- récursivité,
- évaluation d'expressions, etc..

Utilisée pour le parcours des arbres et pour résoudre un grand nombre de problèmes.

Les piles

Machine abstraite

Créerpile(P)	Créer une pile P vide
Empiler(P,Val)	Ajouter Val au sommet de la pile P
Dépiler(P,Val)	Retirer dans Val l'élément en sommet de la pile P
Pilevide((P)	Tester si la pile P est vide
Pilepleine(P)	Tester si la pile P est pleine

Les piles

Implémentation

Statique : au moyen de tableaux

Dynamique : au moyen des listes linéaires chaînées

Les piles

Implémentation : Statique

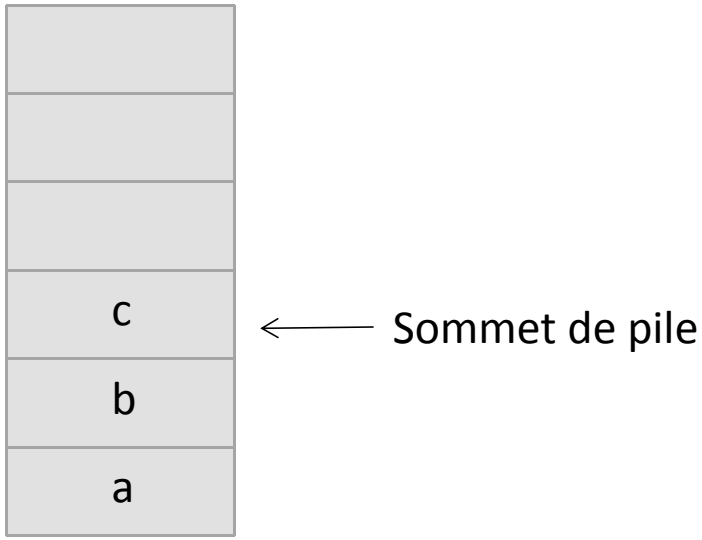


Tableau T

Une pile = (Tableau + Entier)

Entier désigne le sommet

Empilement :
- incrémenter le sommet
- ranger l'élément en sommet de pile

Dépilement:
- récuperer l'élément au sommet
- décrémenter le sommet

Les piles

Implémentation : Statique

```
#define Max 100
#define True  1
#define False 0
typedef int Bool;

struct Pile
  {
    int Som ;
    struct Noeud *Tab[Max];
  }
```

```
void Creerpile( struct Pile *P)
  {
        (*P).Som = 0 ;
  }

 Bool Pilepleine(struct Pile P)
   { return ( P.Som == Max-1 ); }

 Bool Pilevide(struct Pile P)
   { return ( P.Som == 0 ); }
```

Les piles

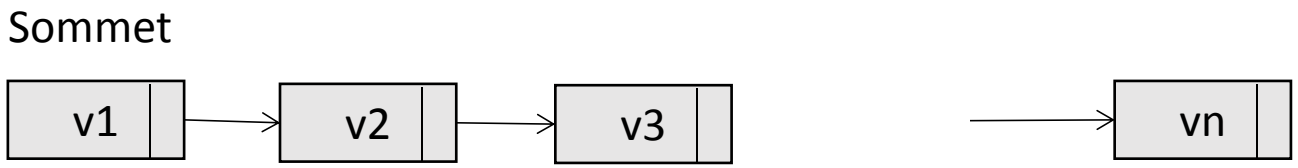
Implémentation : Statique

```
void Empiler ( struct Pile *P, struct Noeud *Val)
  {
   if (  !Pilepleine(*P)  )
    {
      (*P).Som++;
      (*P).Tab[(*P).Som] = Val ;
           }
   else
     fprintf( "Pile satur馥\n"); exit(0) ;
  }
```

```
void Depiler ( struct Pile *P, struct Noeud **Val)
 {
  if (  !Pilevide(*P)  )
   {
     *Val =(*P) .Tab[(*P).Som]  ;
     (*P).Som--;
        }
  else
    fprintf( "Pile vide\n"); exit(0) ;
 }
```

Les piles

Implémentation : Dynamique



Comme une liste linéaire chaînée

La pile est définie par le sommet : Tete de liste

Enfilement : au debut de la liste
Défilement : au début de la liste

Les piles

Implémentation : Dynamique

```
typedef int bool ;
#define True 1
#define False 0

/** Implémentation **\: PILE DE ENTIERS**/
typedef int Typeelem_Pi ;
typedef struct Maillon_Pi * Pointeur_Pi ;
typedef   Pointeur_Pi  Typepile_Pi  ;

struct Maillon_Pi
 {
   Typeelem_Pi  Val ;
   Pointeur_Pi Suiv ;
 } ;
```

```
void Creerpile_Pi( Pointeur_Pi *P )
   { *P = NULL ; }

 bool Pilevide_Pi ( Pointeur_Pi P )
   { return  (P == NULL ); }
```

Les piles

Implémentation : Dynamique

```
void Empiler_Pi ( Pointeur_Pi *P,  Typeelem_Pi Val )
  {
    Pointeur_Pi Q;

    Q = (struct Maillon_Pi *) malloc( sizeof( struct Maillon_Pi))   ;
    Q->Val = Val ;
    Q->Suiv = *P;
    *P = Q;
  }
```

```
void Depiler_Pi ( Pointeur_Pi *P,  Typeelem_Pi *Val )
 {
   Pointeur_Pi Sauv;

   if (! Pilevide_Pi (*P) )
    {
      *Val = (*P)->Val ;
      Sauv = *P;
      *P = (*P)->Suiv;
      free(Sauv);
    }
   else printf ("%s \n", "Pile vide");
 }
```

Les piles

Application : Expressions arithmétiques

Soit à évaluer les expressions suivantes:
(a +b) * (c-d)
(a+b+c) / (a+b-c)
(-(a+b) * (c-d)) + (a*b)

Problèmes
Combien utiliser de variables temporaires?
Comment les gérer?

Solution algorithmique : difficile

Technique de compilation:
- Transformer l'expression en Postfixée (Phase sémantique)
- Utiliser une pile pour l'évaluer

(a +b) * (c-d) → ab+cd-*

(a+b+c) / (a+b-c) → ab+c+ab+c-/

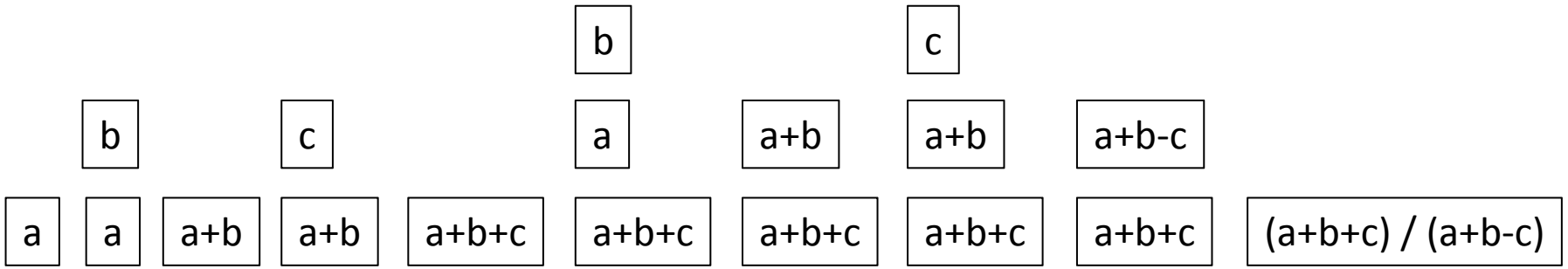
(-(a+b) * (c-d)) + (a*b) → ab+!cd-*ab*+

Notation polonaise postfixée

Les piles

Application

Evaluer a b + c + a b + c - /



Evolution de la pile

Les piles

Application : Expressions arithmétiques (Algorithme)

L'expression est une chaine de caractères terminée par '#'

La chaine de caractères est lue caractère par caractère

Un caractère est soit un opérande (nom de variable)
ou un opérateur

Val (C) : valeur associée à la variable C

Operande(C) : prédicat vrai si C est un opérande, Faux sinon

Binaire (C) : prédicat vrai si C est un opérateur binaire, faux sinon

Oper : fonction retournant le résultat de l'opération C sur V et V2

Oper2 : fonction retournant le résultat de l'opération C sur V

```
Lire(C)
Creerpile(P)
Tq C  <> '#'
   Si Operande()
      Empiler(P, Val(C))
   Sinon
      Si  Binaire(C)
         Depiler(P, V)
         Depiler(P, V2)
         Empiler(P, Oper(C, V, V2))
      Sinon
         Depiler(P, V)
         Empiler(P, Oper2(C, V))
      Fsi
    Fsi
    Lire(C)
Ftq
```