Travaux Pratiques N°13

Les Piles

1. Définition

On appelle pile un ensemble de données, de taille variables, éventuellement nul, sur lequel on peut effectuer les opérations suivantes :

créer pile : elle permet de créer une pile vide
vide : elle permet de voir est ce qu'une pile est vide ou non ?
empiler : elle permet d'ajouter un élément de type T à la pile
dépiler : elle permet de supprimer un élément de la pile
dernier : elle retourne(ou rend) le dernier élément empilé et non encore dépilé.

En générale, les opérations applicables sur une structure de donnée sont divisées en trois catégories :

Les opérations de création (souvent une seule opération) : elles permettent de créer une structure SD.

Illustration : SD pile : créer pile

Les opérations de consultation : elles permettent de fournir des renseignements sur une SD

Illustration : SD pile : vide et dernier. Une opération de consultation laisse la SD inchangée.

Les opérations de modification : elles agissent sur la SD. Illustration : empiler et dépiler.

OPERATIONS ILLEGALES :

Certaines opérations définies sur une SD exigent des pré-conditions: on ne peut pas appliquer systématiquement ces opérations sur une SD.

Illustration :

dépiler exige que la pile soit non vide de même
Idem pour l'opération dernier.

2. Propriétés

Les opérations définies sur la SD pile obéissent aux propriétés suivantes :

(P1) creer_pile permet la création d'une pile vide.
(P2) si on ajoute un élément (en utilisant empiler) à une pile alors la pile résultante est non vide.
(P3) un empilement suivi immédiatement d'un dépilement laisse la pile initiale inchangée.
(P4) On récupère les éléments d'une pile dans l'ordre inverse ou on les a mis (empiler).
(P5) Un dépilement suivi immédiatement de l'empilement de l'élément dépilé laisse la pile inchangée.

REMARQUE : Ces propriétés permettent de cerner la sémantique (comportement ou rôle) des opérations applicables sur la SD pile.
En conclusion : la SD pile obéit à la loi LIFO (Last In, First Out) ou encore DRPS (Dernier Rentré, Premier Sortie).

3. Représentation physique d'une Structure de donnée

Pour pouvoir matérialiser (concrétiser, réaliser ou implémenter) une SD on distingue deux types de représentation :

Représentation chaînée : les éléments d'une SD sont placés à des endroits quelconques (bien entendu dans la MC) mais chaînés (ou reliés). Idée : pointeur.

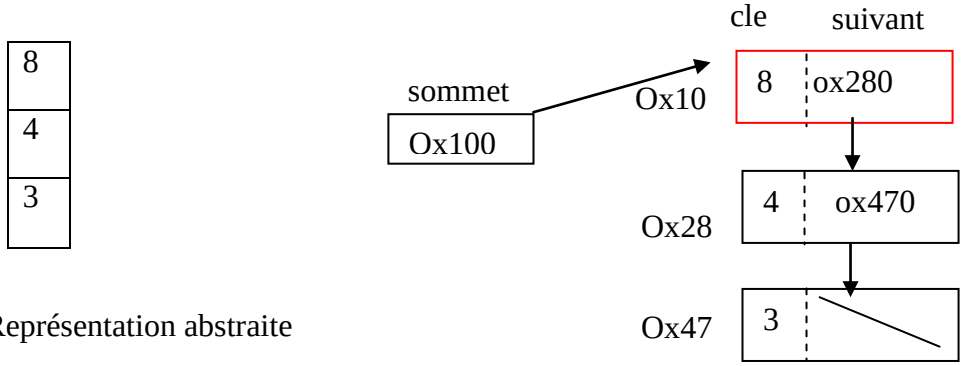
Représentation contiguë : les éléments d'un SD sont rangés (placés) dans un espace contiguë. Idée : tableau.

3.1 Représentation chaînée de la SD pile

Utilisation des pointeurs pour relier explicitement les éléments formant la SD.

La représentation chaînée comporte plusieurs nœuds. Chaque nœud regroupe les champs suivants :

Le champ « cle » permet de mémoriser un élément de la pile.
Le champ « suivant » permet de pointer sur l'élément précédemment empilé.
La variable « sommet » permet de pointer ou de repérer l'élément au sommet de la pile.



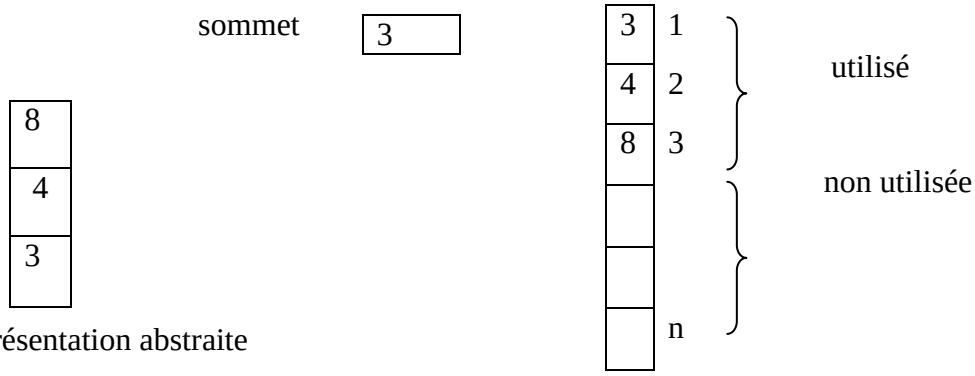
Représentation abstraite

3.2 Représentation contiguë

Pour pouvoir représenter la structure de données Pile d'une façon contiguë, on fait appel à la notion du tableau.
Puisque le nombre d'éléments d'un tableau doit être fixé avant l'utilisation, on aura deux parties :

Partie utilisée : elle est comprise entre 1 et sommet.
Partie non utilisée : elle est comprise entre sommet+1 et n.

Avec sommet est indice compris entre 0 et n.



Représentation abstraite

4. Matérialisation de la SD Pile

Matérialisation de la SD Pile :

Objet abstrait(OA)

Type de données Abstrait (TDA)

- Objet abstrait(OA)➔un seul exemplaire (ici une seul pile)->unique ->implicite.
- Type de données Abstrait (TDA) ➔plusieurs exemplaires->il faut mentionner ou rendre explicite l'exemplaire courant.

4.1 Structure de données Pile comme Objet Abstrait

L'interface pile regroupe des services exportées par cette interface. Chaque service correspond à une opération applicable sur la SD (ici la SD Pile). Et il est fourni sous forme d'un sous-programme

Opérations applicable sur la SD Pile définies
- Opération de création :(proc ou fn)
- Opération de modification : proc
- Opération de consultation : fn

```
/*pile.h est censée être stocké dans le
répertoire courant appelé répertoire de
travail*/
#include<stdlib.h>
#include<assert.h>
/*représentation physique*/
struct element {
  int cle ;
  struct element * suivant ;
} ;

static struct element *sommet ;
/* Pour restreindre la portée de cette variable
à ce fichier, il faut  utiliser le mot réservé
static on dit que la variable sommet n'est pas
destiné à l'exportation */
void creer_pile (void) {
   sommet=NULL ;
}
unsigned vide (void){
   return(sommet==NULL) ;
}
int dernier(void) {
  assert( ! vide()) ;
  return(sommet ->cle) ;
}
void empiler(int info) {
  struct element*p ;
  p=(struct element*)malloc(sizeof(struct
element)) ;
  p->cle=info;
  p->suivant=sommet;
/*mettre à jour sommet*/
  sommet=p;
}
```

```
void depiler (void) {
  struct element* p ;
  assert ( !vide()) ;
  p=sommet ;
  sommet= sommet ->suivant ;
  free(p) ;
}
```

*********Utilisation***********

```
#include<stdio.h>
#include<assert.h>
#include " pile.h "/*on compte utilise des
services offerts par l'interface pile.h*/
void main(void) {
  unsigned i;
  creer_pile();
  assert(vide());
  for(i=1;i<=10;i++) {
    empiler (i);
  }

  assert(!vide());
  for(i=1;i<=10;i++) {
    printf(" %d\n" ,dernier());
    dépiler ();
  }
  assert (vide());
}
```

4.2 Structure de données Pile comme type de donnée abstrait

Dans ce paragraphe, on va concrétiser la SD pile sous forme d'un TDA capable de gérer plusieurs exemplaires de la SD pile et non pas un seul exemplaire (objet abstrait voir 4.1).

Partie interface : pile.h :

```
/******* Partie interface : pile.h *******/
/*représentation physique*/
struct element {
  int cle ;
  struct element*suivant ;
} ;
/*opérations  ou services exporté*/
struct element*creer_pile (void) ;
unsigned vide (struct element*) ;
int dernier (struct element*) ;
void empiler (int, struct element**) ;
void depiler (struct element**) ;
/****** implémentation : Pile .c *******/
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "pile.h "
/*realisation des operations exportées par
pile.h*/
struct element* creer_pile (void) {
  return NULL;
}
unsigned vide (struct element*p) {
  return(p==NULL);
}
int dernier (struct element*p) {
  assert( !vide(p)) ;
  return(p->cle) ;
}
void empiler (int info, struct element** p) {
  struct element*q ;
  q=(struct element*) malloc(sizeof(struct
element)) ;
    q->cle=info;
    q->suivant=*p;
  /*mise à jour*/
  *p=q;
}
```

```
void depiler(struct element **p) {
struct element *q;
assert(!vide(*p));
q=*p ;
*p=(*p) -> suivant ;
free(q) ;
}
/*******Partie utilisation :test.c********/
#include<stdio.h>
#include " pile.h "
void main (void) {
  struct element*p1;
  struct element*p2;
  unsigned i;
  p1=creer_pile();
  for (i=1;i<=10;i++) {
     empiler(i,&p1);
  }
  p2=creer_pile();
  for (i=11;i<=20;i++) {
    empiler(i,&p2);
  }
/*affichage de p1 et p2*/
  for (i=1 ;i<=10 ;i++) {

printf(" %d\t%d\n" ,dernier(p1),dernier(p2));
    depiler(&p1);
    depiler(&p2);
}
/*en principe p1 et p2 sont vides*/
  if (vide(p1)&&vide(p2))
    printf("quelle joie!!");
  else
    printf("problème!?!?");
}
```

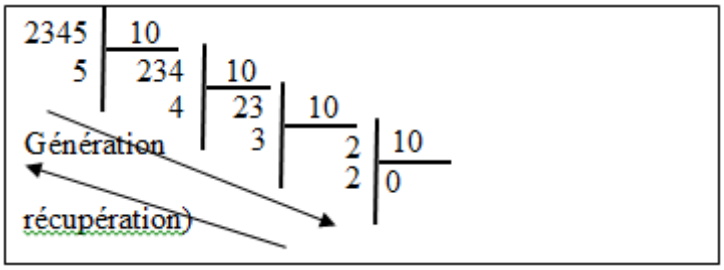
5. Exercices d'application

Exercice 1 :
Enrichir l'objet abstrait PILE, concrétisé par une représentation chainée, en intégrant les opérations suivantes :

- **nb_element** : renvoie le nombre d'éléments de la pile.
- **remplace_sommet** : change le sommet de la pile. Elle exige que la pile soit non vide.
- **effacer** : efface tous les éléments de la pile.

Exercice 2 :
Ecrire un programme en C permettant de lire une expression avec parenthèses supposée valide (nombre des parenthèses ouvrantes=nombre de parenthèses fermantes) et d'afficher toutes les sous-expressions entre parenthèses en commençant par la plus interne. Par exemple, si l'expression soumise au programme est :
a+ (b-(c*d)+8.14)-(d*k) alors le programme demandé doit afficher :
(c*d)
(b-(c*d) +8.14)
(d*k)

Exercice 3 :
Ecrire un programme C qui permet de lire un entier et d'afficher tous les chiffres qui le composent.
Exemple :
Si le nombre proposé est 2345 alors le programme souhaité doit afficher dans l'ordre : 2, 3, 4, 5
Solution
Idée : divisons entiers successives



=>utiliser la SD pile