Université Toulouse III – Paul sabatier
L2 Informatique

Structures de données

Semestre 4

Toulouse, le 15 janvier 2013

Avant-propos

Suite du module d'algorithmique et programmation, accent sur les structures de données
- Pile
- File
- Arbre
- ...

Heures

- 24h de CTDI
- 26 de TDM

Notation

Contrôle intermédiaire 30%

Contrôle terminal 50%

TP 20%

TP Noté 50%

Devoir ecrit 25%

Devoir TP 25 %

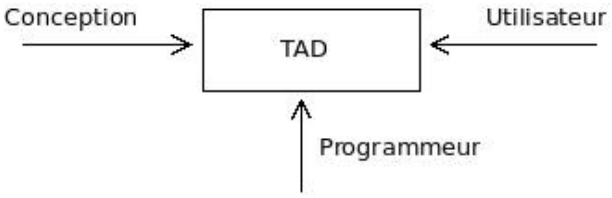
Table des matières

Chapitre 1

Types de données Abstraits (TAD)

C'est une méthode de spécification de structures de données(SD).

C'est utile pour la programmation « En large », c'est-à-dire à plusieurs, pour cela nous sommes obligés de travaillerr sur la communication et l'échange sur le code produit, on utilise pour cela les **spécifications** :
– Les Entrées Sorties du programme [1]
– Les données [2]



Ex Les entiers

Utilisateur : Représentation Interne 1, 2, 3, + ,-, /, +, %

Programmeur : Représentation Externe Entiers « machine » 0000 0011 pour le 3

1.1 Syntaxe des TAD

La syntaxe d'un TAD est répartis en deux étapes :

La signature du TAD [3] Donner les interfaces de la données

La sématique abstraite du TAD [4] Décrire logiquement le fonctionnement de la données.
Une donnée c'est une ou un ensemble de valeurs mais aussi les opérations qui permettent de la manipuler. Cette étape étape nous donne :
– Les limitations de la donnée (préconditions)
– Les descriptions longueur du fonctionnement de chaque opération

1. Vu au S3
2. Nous nous occuperons de cette partie

1.1.1 Signature du TAD `Pile`

Une pile est unee structure de données qui permetn de rassembler des éléments de telle sorte que le dernier élément entré dans la pile soit le premier à en sortir. [5].

Signature de base

Sorte Pile

Utilise Élément, Booleen

Opérations

`creer` →Pile

`empiler` Pile × Element → Pile

`estVide` Pile → Booleen

`sommet` Pile → Pile

`appartient` Pile × Element → Booleen

Signature étandue

Préconditions

- `sommet(p)`⇔ ¬ `estVide(p)`

Axiones

```
/*
 * Avant toute chose on partitionne l'ensemble des opérations en deux sous
    ensembles :
 *    Les constructeurs
 *    Les opérateurs
 *  L'ensemble des constructeurs est necessaire et suffisant pour pouvoir
    gagner n'importe quelle valeur de la donnée
 */

 /* On applique chaque constructeur à chaque opérateur et on decrit logiquement
     ce qui se passe
 */
 estVide(creer()) = true;
 estVide(empiler(p, x)) = false;
 depiler(creer()) = creer();
 depiler(empiler(p, x)) = p;
 sommet(empiler(p, x)) = x;
 appartient(creer(), x) = false;
 appartient(empiler(p, x), y) = (x = y) ∨ appartient(p, y)
```

1.2 Implémentation d'un TAD

1. Implémenter la structure de données
2. Implémenter les opérateurs
3. Séparer l'interface du corps des opérations

   But 1 Permet de modifier les opérations sans remettre en cause la manière d'utiliser le TAD

   But 2 Protéger les données

5. Last In First Out

1.2.1 Implémentation de la structure de données et des opérateurs

Trouver une représentation interne de la structure de données, celle-ci est contrainte par le langage choisi.

Celle-ci peut être statique ou dynamique.

Statique La donnée ne peut plus changer de place ni de taille mémoire ou dynmaique.
- Problème de gaspillage de place
- Avantage de l'efficacité

Dynamique La donnée peut changer de taille ou de place pendant l'execution du programme.
- Pas de gaspillage de place
- Inconvénient de l'efficacité

1.2.1.1 Implémentation statique du TAD `Pile`

- Utilisation d'un tableau
- Utilisation d'un entier donnant le nombre d'éléments rangés dans la pile

```
1  #define N 1000
2
3  struct eltPile {
4    Element Tab[N];
5    int nb;
6  } Pile;
7
8  Pile creer() {
9    Pile p;
10   p.nb = 0;
11
12   return p;
13 }
14
15 Pile empiler(Pile p, Element x) {
16   assert(p.nb < N); // Si la condition est false alors arrête programme
17   p.tab[p.nb] = x;
18   p.nb++;
19
20   return (p);
21 }
22
23 int estVide(Pile p) {
24   return (p.nb == 0);
25 }
26
27 Pile depiler(Pile p) {
28   if(!estVide(p)) {
29     p.nb--;
30   }
31
32   return p;
33 }
34
35 Element sommet(Pile p) {
36   asset(!estVide(p)); // Pas indispensable masi plus robuste
37   return (p.tab[p.nb-1]);
38 }
39
```

```
40 int appartient(Pile p, Element x) {
41   if(estVide(p))
42     return 0;
43
44   if(x == sommet(p)) {
45     return 1;
46   }
47
48   return (appartient(depiler(p), x));
49 }
```

Annexe A

Exercices

A.1 TAD

A.1.1 Suite du TAD `Pile`

1. Implémenter la fonction permettant de remplacer toute les occurrences de l'élément x par l'élément y dans la pile.
2. Implémenter la fonction d'affichage de la Pile.

Rajouter dans le champ des opérations `remplacerOccurence` Pile × Element × Element → Pile

Préconditions rien

Axiones

```
remplacerOccurence(creer(), x, y) = creer();
remplacerOccurence(empiler(p, x), x1, x2) =
    p1 ∧ ∀z (appartient(p1, z) → (z ≠ x1) (empiler(p, x), z') ∧ z' = x1))
```

```
Pile remplacer(Pile pPile, Element pX, Element pY) {
  int i;
  for(i=0 ; i < p.nb ; ++i) {
    if(p.tab[i] == x) {
      p.tab[i] = y;
    }
  }

  return p;
}

void afficher(Pile pPile) {
  while(!estVide(p)) {
    printf(sommet(pPile));
    pPile = depiler(pPile);
  }
}
```