

Cours d'algorithmique : Structures de données avancées

Mathieu barcikowski
Attaché Temporaire à l'Enseignement et à
la Recherche
Laboratoire SYSCOM
Bat. Mont Blanc

Mail : mathieu.barcikowski@univ-savoie.fr
URL : <http://mathieu.barcikowski.free.fr>

01/09/2006

1

Références

- ❑ Le langage C (Edition CampusPress)
- ❑ Algorithmes et structures de données
génériques (Edition Dunod)
- ❑ [http://www-ipst.u-
strasbg.fr/pat/program/algo.htm](http://www-ipst.u-strasbg.fr/pat/program/algo.htm)
- ❑ [http://www.sciences.univ-
nantes.fr/info/enseignement/deug/info5/](http://www.sciences.univ-nantes.fr/info/enseignement/deug/info5/)
- ❑ ...

01/09/2006

2

Plan

❑ **Algorithmes**

- ❑ Types abstraits de données
- ❑ Pointeurs
- ❑ Listes
- ❑ Piles
- ❑ Files
- ❑ Tris
- ❑ Complexité

01/09/2006

3

Algorithme

- ❑ Algorithme : Suite de calculs obéissant à un
enchaînement déterminé dans un but
donné.
- ❑ Exemple : Algorithme d'Euclide au 8^{ème}
siècle (PGCD)
- ❑ Vision contemporaine : machine capable
d'exécuter des opérations arithmétiques
élémentaires.

01/09/2006

4

Algorithme

- Durant les années 30, formalisation du concept de fonction calculable (Güdel, Church, Turing, Post) :
 - Fonctions récursives
 - λ – calcul
 - Machine de Turing
 - Systèmes de Post
 - Machines à registres
- Théorie de la calculabilité : caractérisation des fonctions (solutions à des problèmes) exprimables (« calculables ») sous forme d'un algorithme.

01/09/2006

5

Plan

- Algorithme
- **Types abstraits de données**
- Pointeurs
- Listes
- Piles
- Files
- Tris
- Complexité

01/09/2006

6

Types abstraits de données > Définition

- Définition d'un algorithme à un niveau aussi conceptuel que possible.
- Abstraction sur les techniques de représentation des données : fonctionnalité avant tout.
- Représentations des données de manière abstraite par l'ensemble de leurs opérations.
- Conception des algorithmes en utilisant les opérations des types abstraits et seulement celles-ci.

01/09/2006

7

Types abstraits de données > Signature d'un type abstrait

- Identificateurs de types identifiant les domaines de valeurs : les sortes.
- Nom du nouveau type.
- Utilisation de sortes prédéfinies/définies
- Liste des opérations de manipulation de sortes.
- Profil de chaque opération : liste des arguments, avec leur type de résultat.

01/09/2006

8

Types abstraits de données > Signature d'un type abstrait > Premier exemple

- Sorte Vecteur
 - Utilise Élément, Entier

- Opérations :
 - lème: Vecteur \otimes Entier \rightarrow Élément
 - Changer-ième : Vecteur \otimes Entier \otimes Élément \rightarrow Vecteur
 - Bornesup : Vecteur \rightarrow Entier
 - Borneinf : Vecteur \rightarrow Entier

01/09/2006

9

Types abstraits de données > Signature d'un type abstrait > Premier exemple

- Si v est un vecteur, i un entier, e un élément :
 - lème (v, i) est un élément
 - Changer-ième (v, i, e) est un vecteur
 - Borneinf (v) est un entier

01/09/2006

10

Types abstraits de données > Signature d'un type abstrait > deuxième exemple

- Sorte Booléen

- Opérations :
 - Vrai: \rightarrow Booléen
 - Faux: \rightarrow Booléen
 - \neg _: \rightarrow Booléen \rightarrow Booléen
 - $_ \wedge _$: \rightarrow Booléen \otimes Booléen \rightarrow Booléen
 - $_ \vee _$: \rightarrow Booléen \otimes Booléen \rightarrow Booléen

- Ces définitions établissent la syntaxe du type abstrait (nom des opérations, types des arguments,...) pas sa sémantique.

01/09/2006

11

Types abstraits de données > Hiérarchies de types abstraits

- Le type abstrait Vecteur fait intervenir les types entier, et élément.
 - On évite de redéfinir ces types lors de leur utilisation.

- On réutilise les types en introduisant la notion suivante :
 - Sorte Vecteur ;
 - Utilise élément, entier ;

- Définition d'une hiérarchie d'utilisation entre les types : le type vecteur est "au-dessus" du type "entier" et "élément".

- Définition :
 - Sorte(s) définie(s) : la ou les sorte(s) correspondant aux noms de sorte nouveau.
 - Sorte(s) prédéfinie(s) : la ou les sortes provenant de sortes utilisés.

01/09/2006

12

Types abstraits de données > Hiérarchies de types abstraits > Définitions & Règles

- Définitions
 - Opération interne : une opération dont le résultat appartient à une sorte définie.
 - Observateur : si au moins un argument est d'une sorte définie et si le résultat est une sorte prédéfinie.
- Règles méthodologiques :
 - Une signature comporte généralement des opérations internes et des observateurs.
 - On ne définit pas de nouvelles opérations sur les sortes prédéfinies.

01/09/2006

13

Types abstraits de données > Type abstrait algébrique

- Donner une sémantique aux constructions introduites dans les signatures (sortes et opérations).
- Définition d'un ensemble d'axiomes décrivant les propriétés des opérations.
- Exemple :
 - Si v est un Vecteur, i un entier et e un élément :
 $\text{Borneinf}(v) \leq i \leq \text{Bornesup}(v) \Rightarrow (\text{lème}(\text{Changer-ième}(v, i, e), i) = e)$

01/09/2006

14

Types abstraits de données > Type abstrait algébrique > Définition

- Spécification algébrique ou axiomatique d'un type abstrait :
 - Une signature et un ensemble d'axiomes
- Deux problèmes essentiels :
 - Consistance : y a-t-il des axiomes contradictoires ?
 - Complétude : a-t-on un nombre suffisant d'axiomes pour décrire les propriétés du type abstrait que l'on voulait spécifier ?
- On applique le critère dit de "complétude suffisante" : Pour toute application d'un observateur à un objet d'une sorte définie, les axiomes doivent permettre de déduire une valeur d'une sorte prédéfinie
 - Observateur (sorte définie, ...)
 - Observateurs (opération interne (...), ...)
- Axiomes définissant le résultat de la composition des observateurs avec toutes les opérations internes.

01/09/2006

15

Types abstraits de données > Type abstrait algébrique > Définition

- Application pertinente des axiomes : les opérations internes doivent rendre des résultats appartenant au domaine de définition des observateurs.
- Le domaine de définition d'une opération partielle est donc défini par une pré-condition.
- Exemple :
 - dans la sorte Vecteur combinaison de Borneinf et Bornesup uniquement avec l'opération interne lème.
 - On ajoute une opération interne définissant les bornes d'un vecteur non initialisé : Vect : Entier \otimes Entier \rightarrow Vecteur

01/09/2006

16

Types abstraits de données > Type abstrait algébrique > Définition

- D'où les axiomes
 - $\text{Borneinf}(\text{Vect}(i, j)) = i$
 - $\text{Bornesup}(\text{Vect}(i, j)) = j$
- Problèmes de la combinaison de leme avec Vect
 - $\text{leme}(\text{Vect}(i, j), k)$
 - Non défini car Vect renvoie un vecteur non initialisé

Écriture du domaine de définition de l'opération leme : utilisation d'une opération indiquant si un élément donné caractérisé par un certain indice ou non été initialisé

- Init: vecteur \otimes entier \rightarrow booléen

01/09/2006

17

Types abstraits de données > Type abstrait algébrique > Définition

- D'où les axiomes :
 - $\text{Init}(\text{Vect}(i, j), k) = \text{faux}$
 - $\text{Borneinf}(v) \leq i \leq \text{Bornesup}(v) \Rightarrow (\text{Init}(\text{Changer-ieme}(v, i, e), i) = \text{vrai})$
 - $\text{Borneinf}(v) \leq i \leq \text{Bornesup}(v) \wedge i \neq j \Rightarrow (\text{Init}(\text{Changer-ieme}(v, i, e), j) = \text{Init}(v, j))$
- Définitions de leme (v, i)
 - leme (v, i) est défini si et seulement si $\text{Borneinf}(v) \leq i \leq \text{Bornesup}(v) \wedge \text{Init}(v, i) = \text{vrai}$

01/09/2006

18

Types abstraits de données > Type abstrait algébrique > Définition

- Définitions des observateurs Bornesup et Borneinf sur le résultat de l'opération interne changer leme :
 - $\text{Borneinf}(\text{Changer-ieme}(v, i, e)) = \text{Borneinf}(v)$
 - $\text{Bornesup}(\text{Changer-ieme}(v, i, e)) = \text{Bornesup}(v)$

01/09/2006

19

Types abstraits de données > Type abstrait algébrique > Synthèse du type vecteur

- Sorte Vecteur
 - Utilise Élément, Entier, booléen
- Opérations :
 - Vect: Entier \otimes Entier \rightarrow Vecteur
 - lème: Vecteur \otimes Entier \rightarrow Élément
 - Changer-ieme: Vecteur \otimes Entier \otimes Élément \rightarrow Vecteur
 - Init: Vecteur \otimes Entier \rightarrow Booléen
 - Bornesup: Vecteur \rightarrow Entier
 - Borneinf: Vecteur \rightarrow Entier

01/09/2006

20

Types abstraits de données > Type abstrait algébrique > Synthèse du type vecteur

□ Avec :

- V : vecteur ;
- i, j, k : entier ;
- e : élément ;

□ Pré-condition :

- $\text{lème}(v, i)$ est défini si et seulement si $\text{Borneinf}(v) \leq i \leq \text{Bornesup}(v) \wedge \text{Init}(v, i) = \text{vrai}$

01/09/2006

21

Types abstraits de données > Type abstrait algébrique > Synthèse du type vecteur

□ Axiomes :

- $\text{Borneinf}(v) \leq i \leq \text{Bornesup}(v) \Rightarrow (\text{lème}(\text{Changer-ième}(v, i, e), i) = e)$
- $\text{Borneinf}(v) \leq i \leq \text{Bornesup}(v) \wedge \text{Borneinf}(v) \leq j \leq \text{Bornesup}(v) \wedge i \neq j \Rightarrow (\text{lème}(\text{Changer-ième}(v, i, e), j) = \text{lème}(v, j))$
- $\text{Init}(\text{Vect}(i, j), k) = \text{faux}$
- $\text{Borneinf}(v) \leq i \leq \text{Bornesup}(v) \Rightarrow (\text{Init}(\text{Changer-ième}(v, i, e), i) = \text{vrai})$

01/09/2006

22

Types abstraits de données > Type abstrait algébrique > Synthèse du type vecteur

- $\text{Borneinf}(v) \leq i \leq \text{Bornesup}(v) \wedge i \neq j \Rightarrow (\text{Init}(\text{Changer-ième}(v, i, e), j) = \text{Init}(v, j))$
- $\text{Borneinf}(\text{Vect}(i, j)) = i$
- $\text{Borneinf}(\text{Changer-ième}(v, i, e)) = \text{Borneinf}(v)$
- $\text{Bornesup}(\text{Changer-ième}(v, i, e)) = \text{Bornesup}(v)$

01/09/2006

23

Types abstraits de données > A retenir

- Abstraction du langage et de la représentation machine = un modèle, plusieurs façons d'implémenter.
- Séparation entre opérations internes et observateurs.
- Modélisation de la structure de données par son comportement.
- Prise en compte implicite la gestion des erreurs.

01/09/2006

24

Plan

- Algorithmme
- Types abstraits de données

□ Pointeurs

- Listes
- Piles
- Files
- Tris
- Complexité

01/09/2006

25

Pointeurs > Rappel sur mémoire

- Valeur i = 10
- Adresse i = 0
- Valeur j = 11
- Adresse j = 2
- Valeur c = {x = 10, y = 11}
- Adresse c = 4
- SizeOf(i) = 2
- SizeOf(j) = 2
- SizeOf(c) = 4

Adresse	Valeur
0	Int I (10)
1	
2	Int J (11)
3	
4	COOR c
5	{x = 10, y = 11}
6	
7	

01/09/2006

26

Pointeurs > Utilisation en C

- Opérateur d'adresse : &
 - int i = 0;
 - &i : adresse de i
- Type pointeur :
 - int * point; \pointeur vers un entier i
 - Dans le programme
 - Point : accès à l'adresse
 - *Point : accès à la valeur

01/09/2006

27

Pointeurs > En mémoire

- Valeur i = 10
- Adresse i = 0
- Valeur p = 0
- Adresse p = 500
- Valeur p = Adresse i

Adresse	Valeur
0	Int I (10)
1	
...	...
...	...
...	...
500	Int *p (0)
501	
...	7

01/09/2006

28

Pointeurs > Exemple

- `Int i = 0;`
- `Int *point;` [\\adresse](#) indéfinie

- (hypothèse: adresse de i = 4000)
- `point = &i;` [\\ point](#) pointe sur l'adresse de i
- `I = 4`
- Valeur de point ?
- Valeur de *point?

01/09/2006

29

Pointeurs > Passage de pointeur en paramètre

- déclaration

```
Int passage(Int * x){
    *X = 2; \\utilisation de la valeur de X
    X = 2000; \\utilisation du pointeur vers X
}
```

- Utilisation

```
Int i = 0;
Passage(&i);

Int *point;
Point = malloc(sizeof(int)); \\création de la place mémoire pour un entier
*point = 10;
Passage(point);
```

01/09/2006

30

Pointeurs > Malloc

- adresse indéfinie à la déclaration d'une variable de type pointeur
- Pour l'initialisation :
 - Affectation d'une adresse existante grâce à l'opérateur &
 - Utilisation de la fonction malloc pour créer dynamiquement l'emplacement mémoire pour la structure de données
- Utilisation du malloc
 - `void *malloc (size_t size);`
 - Paramètre : taille de la structure en octets.
 - Exemple : `malloc(1)`, `malloc(sizeof(int))`, `malloc(sizeof(COOR))` ...
 - Renvoie l'adresse de la structure créée dynamiquement

01/09/2006

31

Pointeurs > A retenir

- Accès bas niveau à la mémoire.
- Gestion dynamique de la mémoire permettant la gestion dynamique de structures de données.
- Passage de paramètres par référence (i.e. permet leurs modifications).
- Attention : Toujours utilisé un pointeur après affectation soit par une variable locale déjà initialisée (opérateur &) soit en déclarant dynamiquement une nouvelle zone mémoire (fonction malloc).

01/09/2006

32