Algorithmique, complexité et graphes.

Chapitre 2 : mécanismes en programmation et conventions de description

Sont décrits ici les mécanismes usuels en programmation et certaines conventions d'écriture que nous utiliserons pour les algorithmes dans la suite du cours.

Commentaires

Tout algorithme commencera par une spécification informelle comportant :

- Entrées
- Sorties
- Ce qu'il doit faire (sans entrer dans les détails)

Format : {commentaires, spécifications} ou /* commentaires, spécifications */

Variables, instructions et blocs

L'algorithme manipule des variables dénotées par des étiquettes et susceptibles de figurer dans les instructions classiques

- **Affectation** : machin ← truc. La flèche dénote l'affectation (on met dans machin la valeur contenue par bidule). On bannira absolument la notation issue du c (machin=truc), qui sera réservée à sa fonction mathématique de comparaison de valeurs. On refusera également les opérateurs d'affectation étendus tels que +=, -=, *=,… qui ne permettent pas de distinguer opération et affectation.
- **Structures de contrôle** : on utilisera
    - Si… alors… sinon…
    - Tant que… faire…
    - Répéter… jusqu'à…
    - Pour… allant de… jusqu'à… faire
- **Blocs :** on délimitera les blocs d'instruction graphiquement avec une barre verticale et/ou une indentation. On pourra utiliser des notations début machin, fin truc afin de lever les éventuelles ambiguïtés (begin/end est accepté).
- **Si… sinon** : attention à l'imbrication des blocs de condition (si) qui peut conduire à des ambiguïtés. On pourra utiliser l'instruction « sinon ne rien faire » pour clarifier les choses.

Booléens

Le type de variable booléen ne contient que deux constantes : VRAI et FAUX. **Certains langages les confondent avec des nombres, ce ne sera pas le cas ici**. Seront acceptées les opérations connues sur les booléens d'un point de vue mathématique :
"4=5" est une expression booléenne en mathématique, dont la valeur est FAUX.

Si B est une variable booléenne et A une variable entière, on pourra écrire : B←A=5, ce qui indique que B reçoit le résultat de la comparaison de la valeur contenue dans A avec l'entier 5.
On pourrait écrire : si A=5 alors B←VRAI sinon B←FAUX

Evaluation des connecteurs logiques

Confrontés à une expression du type *si (A ET B) alors…* où A et B sont des expressions, certains compilateurs évaluent A et B quelle que soit la valeur de A. D'autres fonctionnent de façon "paresseuse" et n'évaluent B que si A est vraie. (c'est vrai également pour OU)

En règle générale, on préfèrera écrire comme si l'évaluation n'était pas paresseuse (il faut donc prendre garde à ce que B puisse être évaluée quelles que soient les circonstances), mais il arrivera qu'on considère l'évaluation comme paresseuse.

Classes, méthodes, types abstraits, procédures et fonctions

En programmation impérative (Algol, Fortran, Pascal, C, Ada, …), un programme est défini par une structure de données et un algorithme.
En programmation objet (Simula, SmallTalk, Objective-C, ObjVlisp, Java, C++, …), les structures de données sont empaquetées avec des outils qui les manipulent : c'est une forme de programmation modulaire. Il s'agit d'un style de programmation qui n'interagit pas fondamentalement avec l'algorithmique sous-jacente.
Nous utiliserons alternativement les deux vues :

- des procédures et fonctions agissant sur des données décrites par des types abstraits
- des classes comportant champs et méthodes

On considèrera que les données sont les champs des classes et les procédures et fonctions sont leurs méthodes : rien ne change pour les algorithmes associés !

Exemple : on représenterait les disques des Tours de Hanoï comme suit

Version objet :	Version impérative :

Disque = classe{
Champs
Diamètre : flottant
Epaisseur : entier
Matière : {bois, fer, verre}
Peint : Booléen
 	
Méthodes
Fonction Volume() : flottant
Procédure Peindre()
}
	Type Disque = enregistrement
	Diamètre : flottant
	Epaisseur : entier
	Matière : {bois, fer, verre}
	Peint : Booléen
	
	Fonction Volume (d: flottant, e: entier) : flottant
	Procédure Peindre (var d : disque)

Procédures : une procédure est un bloc d'instruction exécuté à un moment donné, sur appel de l'algorithme principal. Elle a un statut d'instruction. On précisera systématiquement le passage des paramètres utilisés en entrée et en sortie.
Les variables internes aux procédures ne seront pas toujours déclarées, mais seront toujours considérées comme locales. Les variables globales seront proscrites autant que possible.

Ex : Procédure Ajoute (entrée, sortie S: Sac ; entrée X: Bille) {…}/* S peut être modifiée en sortie, la procédure ajoute la bille X au sac S*/

Fonctions : une fonction rassemble des instructions en passant des paramètres en entrée et en manipulant des variables locales, mais rend un résultat à la sortie.

Ex : Fonction compte (entrée T : Tas ; N : Entier ; y : Elément) : entier /*Connaissant T de taille N, compte les occurrences de Y dans T. La sortie est le résultat rendu par la fonction*/
appel : si compte(T, N, y) = 5 alors…

L'appel ne se fait pas comme une instruction mais en utilisant le résultat de la fonction dans une autre instruction. Elle est du même type que le résultat qu'elle retourne.

Mode de passage des paramètres et pointeurs : voir cours fait en classe !