Cours d'algorithmique
Généralités - EISTI - ING 1

Ecole Internationale des Sciences du Traitement de l'Information

Variables

Pourquoi

- Stocker temporairement des valeurs
- Données issues du dique dur, saisies par l'utilisateur, calculées par une autre partie du programme, ...
- Image d'une boite contenant une donnée accessible via une étiquette

Comment : déclaration de variables

- Syntaxe :

```
variables
  nom_variable : type
```

- Types prédéfinis :
  `entier,reel,caractere,chaine,booleen`

Expressions

Définition

Une expression est un ensemble de valeurs reliées par des opérateurs et équivalente à une seule valeur.

Opérateurs

Les différents opérateurs sont liés aux types des valeurs qu'ils manipulent :

- entier : $+, -, *, /, \mathit{div}, \mathit{mod}$
- reel : $+, -, *, /$
- booleen : $\mathit{non}, \mathit{et}, \mathit{ou}$
- chaine : &
- Divers :
  - Parenthèses (gestion des priorités) : $(,)$
  - Comparaison (résultats booléens) : $=, \neq, >, <, \geq, \leq$

En plus des parenthèses des règles de priorités implicites sont liées aux différents opérateurs.

Affectations

Pourquoi

L'affectation permet d'associer une valeur à une variable

Comment : instruction d'affectation

Syntaxe :

```
nom_variable ← expression
```

Remarques

- Seule la partie gauche d'une affectation est modifiée
- L'expression qui est affectée à la variable doit être d'un type compatible avec cette variable
- La dernière affectation d'une variable écrase sa valeur précédente

Variables booléennes

Pourquoi

- Stocker une information pouvant prendre 2 valeurs (en générale opposées) : vrai ou faux
- Permettre une alternative conditionnelle lors d'une suite d'instructions (structures de contrôles)

Opérateurs

et	*V*	*F*
V	*V*	*F*
F	*F*	*F*

ou	*V*	*F*
V	*V*	*V*
F	*V*	*F*

non	*V*	*F*
	F	*V*

Instructions conditionnelles

Pourquoi

Les instructions conditionnelles permettent d'offrir lors du déroulement d'une suite d'instructions, une alternative selon la valeur d'un test booléen (condition).

Comment : Syntaxe

```
si Condition alors
  Instructions
fsi

si Condition alors
  Instructions 1
sinon
  Instructions 2
fsi
```

Instructions conditionnelles imbriquées

Pourquoi

Il est possible d'imbriquer des instruction conditionnelles lorsque le choix dépend de plusieurs critères dépendants les un des autres.

Comment : syntaxe

```
si Condition 1 alors
  Instructions 1
sinonsi Condition 2 alors
  Instructions2
....
sinonsi Condition n alors
  Instructions n
sinon
  Instructions n+1
fsi
```

Assertions

Pourquoi

Améliorer la lisibilité d'un algorithme en ajoutant des expressions logiques (booléennes) toujours vraies décrivant formellement les instructions.

Comment : Syntaxe

```
{expression booleenne}
```

Exemple

```
variables
  temp: entier
  ecrire("Entrez la température de l'eau")
  lire(temp)
  si temp ≤ 0 alors                    {temp ≤0}
    ecrire("C'est de la glace")
  sinonsi temp < 100 alors             {0 < temp < 100}
    ecrire("C'est du liquide")
  sinon                                {temp ≥ 100}
   ecrire("C'est de la vapeur")
  fsi
```

Procédure

Définition :

Suite d'instructions réalisant une certaine tâche, à laquelle on donne un nom pour qu'on puisse l'appeler ultérieurement

Syntaxe :

```
procedure nom(liste de paramètres formels)
Variables locales
Instructions
finprocedure
```

Procédure

Comment

- Procédure principale : corps du programme

```
programme nom
debut
    Instructions
fin
```

- Sousprocédure : suite d'instruction en dehors de la procédure principale
- Appel des sousprocédures explicitement en utilisant leur nom et en fixant les valeurs des paramètres formels (entre parenthèses) qui deviennent des paramètres effectifs

```
nom(liste des paramètres effectifs)
```

Fonctions

Définition

Suite d'instructions nommée (comme les procédures) qui retourne une valeur à la procédure ou fonction appelante.

Syntaxe

```
fonction nom(liste de parametres): type de
    retour
  Variables locales
  Instructions
  retourner ...
finfonction
```

Fonctions prédéfinies

Exemples selon le type

- Numérique :

```
ent(n:reel)     //partie entière
alea(n:entier) //nombre aléatoire entre 0 et n
sqrt(n:entier) //racine carrée de n
sin(x:reel)
cos(x:reel)
```

- Texte :

```
//nombre de caractères:
  longueur(c:chaine)
/sous chaine entre n1 et n2:
  extrait(c:chaine, n1:entier, n2:entier)
//valeur de c dans la table ascii:
  ascii(c:chaine)
//caractere ascii associé à n:
  caractere(n:entier)
```

- ...

Passage de paramètres

Types de paramètres

- Paramètres en entrée (lecture) : E
- Paramètres en sortie (écriture) : S
- Paramètres en entrée-sortie (lecture-écriture) : ES

Exemple

```
procedure divisionEntiere(E a:entier,E b:entier, S
    quotient:entier, S reste: entier)
procedure inversion(ES s:chaine)
```

Portée des variables

Variables locales

- Déclarées au sein d'une procédure ou d'une fonction
- Visibles uniquement par cette procédure/fonction

Variables globales

- Alternative aux arguments pour communiquer entre procédures
- Visibles par tout le programme

Procédures d'entrées sorties

Lecture

Entrer des valeurs externes (ex : clavier).

Ecriture

Communiquer des valeurs vers l'extérieur (ex : écran).

Syntaxe :

```
//écriture d'une chaîne de caractères:
ecrire(E ch:chaine)

//écriture d'une chaîne de caractères et retour à
    la ligne:
ecrirenl(E ch:chaine)

//lecture d'un variable (procédure polymorphe):
lire(S val:?)
```

Ici, *val* est une variable d'un type quelconque (caractere,chaine,entier, reel, booleen) contenant la valeur externe produite par l'utilisateur.

Instruction d'itération

Comment ?

Syntaxe d'une boucle "tant que" :

```
tant que Condition faire
  Instructions
ftq
```

Exemple : saisie d'une réponse

```
variables rep:chaine
ecrire("Entrer une réponse")
lire(rep)
tant que (rep ≠ "Oui" et rep ≠ "Non") faire
  ecrire("Entrer une réponse")
  lire(rep)
ftq
Suite Instructions
```

Boucles pour compter

Pourquoi

Dans de nombreux cas, une boucle sert principalement à compter, c'est à dire à effectuer la suite d'instructions un nombre bien défini de fois. On pourra utiliser dans ce cas une structure particulière adéquate : la boucle "pour".

Syntaxe

```
pour Compteur ← Initial à Final pas ValeurDuPas
  Instructions
fpour
```

- Initial contient la valeur initiale du compteur
- Final contient la valeur finale du compteur
- ValeurDuPas contient la valeur de l'incrémentation du compteur à chaque boucle

Notion d'invariant de boucle

Définition

Un invariant de boucle est une assertion particulière associée à une boucle. Il s'agit donc d'une formule booléenne toujours vraie quelque soit l'itération de la boucle.

Pourquoi

L'apport essentiel de l'invariant de boucle est de permettre une démonstration formelle du résultat produit par une boucle.

Invariant de boucle

Exemple : division euclidienne

```
B ← b
R ← a
Q ← 0
{a = B * Q + R}
tant que R ≥ B faire {(a = B * Q + R) ∧ (R ≥ B)}
  R ← R − B
  Q ← Q + 1
ftq
{(a = B * Q + R) ∧ (R < B)}
```

Preuve de l'invariant

- Conditions initiales : $a = b * 0 + a = a$
- Soit R', B', Q' les valeur modifiées par la boucle de R, B, Q :
  - $R' = R - B$ et $Q' = Q + 1$
  - donc $B' * Q' + R' = B * (Q + 1) + R - B = B * Q + B + R - B = B * Q + R$
  - de plus, $R - B$ diminue strictement (si $B \neq 0$)
- En sortie de boucle, on a $a = B * Q + R$ et $R < B$ □

Invariant de boucle

Exemple : calcul de puissance

```
A ← a
N ← n
R ← 1
{A^N * R = a^n}
tant que N > 0 faire
   si pair(N) alors
      A ← A*A
      N ← N/2
      {(A^N * R = a^n)}
   sinon
      R ← R*A
      N ← N–1
      {(A^N * R = a^n)}
   fsi
ftq
{(A^N * R = a^n) ∧ (N = 0)}  donc  {R = a^n}
```

Notion de coût

Définition

Le coût d'un algorithme peut être déterminé par le nombre d'instructions nécessaires pour le dérouler.

- Calcul très simple pour les instructions séquentielles (comptage)
- Calcul assez simple pour les instructions conditionnelles (on considère le pire des cas)
- Calcul plus compliqué pour les instructions itératives (il faut évaluer le nombre de boucles dans le pire des cas) :
  - Simple en cas de boucle "pour" : valeur du compteur
  - Nécessite un peu de réflexion pour une boucle "tant que" : nombre d'étapes avant que la condition soit vraie
  - Peut être délicat à calculer pour les boucles imbriquées : dépend du degré d'imbrication et de l'indépendance des conditions d'arrêt des boucles.

Coût d'un algorithme

Exemple : division euclidienne

```
B ← b    // 1 instruction
R ← a    // 1 instruction
Q ← 0    // 1 instruction
{a = B * Q + R}
tant que R ≥ B faire                    // 1 instruction
   {(a = B * Q + R) ∧ (R >= B)}
   R ← R − B                            // 2 instruction
   Q ← Q + 1                            // 2 instruction
ftq                                     // Q boucles
{(a = B * Q + R) ∧ (R < B)}  // Total : 5 * Q + 3
```

Le coût ne doit dépendre que des paramètres d'entrée de l'algorithme, ici a et b. On exprime donc Q en fonction de a et b à la fin de l'algorithme :

- D'après l'assertion finale, $a = B * Q + R$ et $R < B$
- Donc $Q = (a - R)/B < (a - B)/B = (a - b)/b$
- Le coût est donc majoré par $5 * (a - b)/b + 3$