Le Langage Pseudo-Code

Cours d'Algorithmique 1er Semestre (Frédéric Koriche)
IUT Informatique de Lens

UNIVERSITÉ D'ARTOIS

Données

En algorithmique, toute donnée est définie par
- son nom: désigne la donnée dans l'algorithme,
- son type: désigne le domaine de valeurs de la donnée, et
- sa nature: variable (peut changer de valeur) ou constante (ne peut pas changer de valeur).

Types Simples

Type	Domaine
booléen	$\{faux, vrai\}$
caractère	Symboles typographiques
entier	$\mathbb{Z}$
réel	$\mathbb{R}$

Opérateurs

Un opérateur est une *fonction* définie par
- son arité: désigne le nombre de variables d'entrée,
- sa position: l'opérateur peut être préfixe (devant), infixe (milieu) ou postfixe (derrière), et
- son type: désigne le type de ses entrées et celui de sa sortie.

Exemple: L'opérateur d'addition sur les entiers
Opérateur binaire, noté $+$ (position infixe), dont le type est:

$$+ : \textbf{entier} \times \textbf{entier} \rightarrow \textbf{entier}$$

Opérateurs sur les types simples

- Arithmétiques (entiers): toutes les entrées sont des **entiers** et la sortie est un **entier**.

Nom	Symbole
addition	$+$
soustraction	$-$
multiplication	$\times$
division entière	$/$
reste	mod
inversion de signe	$-$

- Arithmétiques (réels): au moins une entrée est un **réel** et la sortie est un **réel**.

Nom	Symbole
addition	$+$
soustraction	$-$
multiplication	$\times$
division	$/$
inversion de signe	$-$

- Comparaisons: les deux entrées sont des **entiers**, **caractères*** ou **réels**. La sortie est un **booléen**.

Nom	Symbole
est égal à	$=$
est plus petit que	$<$
est plus grand que	$>$
est plus petit ou égal à	$\leq$
est plus grand ou égal à	$\geq$

- Logiques: toutes les entrées sont des **booléens** et la sortie est un **booléen**.

Nom	Symbole
conjonction	**et**
disjonction	**ou**
négation	**non**

Expressions

Une expression est une *composition* d'opérations dont l'ordre est spécifié par les parenthèses. Le type d'une expression est donné par le type de sa valeur de sortie

Exemple: Supposons que x, y, z soient des entiers.
- $(x > 0)$ **et** $(y < 0)$ est une expression booléenne
- $(x + y)/z$ est une expression entière

Instructions

Une instruction est une *action* à accomplir par l'algorithme. Les quatre instructions de base sont la déclaration (mémoire), l'assignation (calcul), la lecture (entrées) et l'écriture (sorties).

Instruction	Spécification
Déclaration	*type variable*
Assignation	*variable* $\longleftarrow$ *expression*
Lecture	**lire** *variable*
Ecriture	**écrire** *expression*

Blocs

Un bloc est une séquence d'instructions identifiée par une barre verticale.

Exemple: permutation de valeurs
début
  entier *a*, *b*, *temp*
  lire *a*, *b*
  temp $\longleftarrow$ *a*
  a $\longleftarrow$ *b*
  b $\longleftarrow$ *temp*
  afficher *a*, *b*
fin

L'instruction de test "si alors"

Dans l'instruction **si** *condition* **alors** *bloc*, la condition est une expression booléenne, et le bloc n'est exécuté que si la condition est vraie.

Exemple: valeur absolue
début
  réel x, y
  lire x
  $y \leftarrow x$
  si $y < 0$ **alors**
    $y \leftarrow -y$
  afficher y
fin

L'instruction de test "si alors sinon"

Dans **si** *condition* **alors** *bloc 1* **sinon** *bloc 2*, la condition est une expression booléenne. Le bloc 1 est exécuté si la condition est vraie ; le bloc 2 est exécuté si la condition est fausse.

Exemple: racine carrée
début
  réel x, y
  lire x
  si $x \geq 0$ **alors**
    $y \leftarrow \mathrm{sqrt}(x)$
    afficher y
  sinon
    afficher "Valeur indéfinie"
fin

L'instruction de test "suivant cas"

Dans l'instruction **suivant** *condition* **cas où** v_1 *bloc 1* **cas où** v_2 *bloc 2* ..., la condition est une expression pouvant prendre plusieurs valeurs $v_1, v_2, \ldots$. Selon la valeur de la condition, le bloc du cas correspondant est exécuté.

Exemple: choix de menu
début
  entier menu
  lire menu
  suivant menu **faire**
    cas où 1
      afficher "Menu enfants"
    cas où 2
      afficher "Menu végétarien"
    autres cas
      afficher "Menu standard"
fin

L'instruction de boucle "pour"

L'instruction *pour* est utilisée lorsque le nombre d'itérations est connu à l'avance: elle initialise un compteur, l'incrémente après chaque exécution du bloc d'instructions, et vérifie que le compteur ne dépasse pas la borne supérieure.

Exemple: Somme des entiers de 1 à n
début
  entier n, s, i
  lire n
  $s \leftarrow 0$
  pour i ***de*** 1 ***à*** n **faire**
    $s \leftarrow s + i$
  afficher s
fin

L'instruction de boucle "tant que"

La boucle *tant que* est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance: elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple: Somme des entrées saisies par l'utilisateur (version "tant que")
début
  entier $n \leftarrow 1$, $s \leftarrow 0$
  tant que $n \neq 0$ **faire**
    afficher "Entrer un entier (0 pour arrêter) : "
    lire n
    $s \leftarrow s + n$
  afficher s
fin

L'instruction de boucle "répéter jusqu'à"

La boucle *répéter jusqu'à* est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance, et qu'il faut lancer au moins une exécution du bloc d'instructions. Elle exécute le bloc jusqu'à ce que la condition d'arrêt devienne vraie.

Exemple: Somme des entrées saisies par l'utilisateur (version "répéter jusqu'à")
début
  entier $n, s \leftarrow 0$
  répéter
    lire n
    $s \leftarrow s + n$
  jusqu'à $n = 0$
  afficher s
fin