CS110 – Fiche de référence du langage algorithmique – Première partie

Le langage algorithmique que nous utilisons est un langage dit *impératif* : l'algorithme est exprimé comme une suite d'actions, réalisées par des *instructions*. Sa syntaxe est proche de celles de langages de programmation classiques comme Pascal et Ada. Il y a trois catégories d'objets dans le langage :

les variables peuvent être pensées comme des "boites" portant un nom et contenant une valeur. La valeur contenue dans la boite peut être utilisée ou modifiée par l'algorithme. L'ensemble des variables constituent l'*environnement* de l'algorithme. Une variable dont on n'a pas fixé la valeur est *indéterminée*.

les instructions représentent une action, elle ont un effet comme afficher un message à l'écran ou modifier l'environnement (changer la valeur d'une variable).

les expressions représentent une valeur. Cette valeur n'est pas forcément immédiatement apparente, elle est calculée par *évaluation* de l'expression.

Structure d'un algorithme

La spécification doit indiquer clairement et précisément, sans ambiguïté, ce que fait l'algorithme (mais pas *comment* il le fait).

```
Algorithme son_nom
(* spécification *)
déclarations
début
   instructions
fin
```

Identificateurs

Divers objets du langage, tels que les variables, sont désignés par leur *nom*. Ce nom, ou *identificateur*, est un mot constitué des caractères alphabétiques courants, des chiffres et des symboles "`-`" et "`_`". Le premier caractère d'un identificateur est forcément une lettre. Les identificateurs doivent être choisis judicieusement pour faciliter la compréhension de l'algorithme.

Exemples

`a, toto12, tri-sélection, ~~ouvert/fermé~~, ~~2ieme_compteur~~`

Valeurs et types

Les valeurs qu'un algorithme peut manipuler sont classées en *types*. La notion de type recouvre plus ou moins la notion mathématique d'ensemble.

entier type des valeurs entières (sous-ensemble de $\mathbb{Z}$)

réel type des valeurs numériques non forcément entières (représentation approchée des nombres réels)

booléen type des valeurs logiques, notées `vrai` et `faux`

car type des caractères alphabétiques, numériques et de ponctuation.

nom	description	littéraux
entier	sous-ens. de $\mathbb{Z}$	1 2 –5 9999
réel	valeurs approchées d'une partie de $\mathbb{R}$	1.0 3.14 –2.67
car	caractères	'a' '6' ' ?'
booléen	valeurs logiques	vrai faux
chaîne	chaîne de caractères	"coucou !"

TAB. 1 – Types de données élémentaires

opérateur	type des opérandes	type du résultat	opération
$-$	1 entier ou réel	idem opérande	négation
$+$ $-$ $\times$ $/$	2 entiers ou 2 réels ou 1 entier et 1 réel	entier si les 2 opérandes le sont, réel sinon	addition soustraction multiplication
		réel	division réelle
div mod	2 entiers	entier	quotient entier reste de la division
$<$ $\leq$ $>$ $\geq$ $=$ $\neq$	2 nombres, 2 caractères ou 2 chaînes	booléen	inférieur inférieur ou égal supérieur supérieur ou égal égal différent
non	1 booléen		négation logique
et ou	2 booléens		conjonction disjonction

TAB. 2 – Opérateurs du langage

Plus forte classe 1	non $-$ (négation)
classe 2	et $\times$ / div mod
classe 3	ou $+$ $-$
Plus faible classe 4	$= \neq < \leq > \geq$

TAB. 3 – Priorité des opérateurs

Expressions

Une expression représente une valeur d'un certain type. Une expression est construite à partir de littéraux du type, de variables ayant ce type et d'opérateurs qui produisent une valeur de ce type. L'évaluation de l'expression se fait suivant les priorités des opérateurs. Les parenthèses peuvent être utilisées pour changer l'ordre d'évaluation.

Variables (déclaration, affectation)

Une variable est un objet contenant une valeur d'un certain type, cette valeur pouvant changer pendant l'exécution de l'algorithme. Toute variable doit être *déclarée* (avec le mot clé `var`) avant d'être utilisée. La déclaration fixe le type de la variable mais pas sa valeur, qui est *indéterminée* (c'est à dire n'importe quoi) avant d'avoir été affectée. Il est donc nécessaire de fixer la valeur d'une variable avant de l'utiliser dans une expression.
Le nom d'une variable représente sa valeur à l'instant considéré sauf s'il apparaît à gauche du symbole de l'affectation $\leftarrow$, qui permet d'affecter une valeur à la variable.
Dans l'instruction `i←i+1` (où `i` est une variable de type `entier`) le deuxième `i` représente la valeur de la variable au moment où l'expression `i+1` est évaluée. Le premier `i` indique que la valeur de l'expression doit être affectée à la variable `i`. Cette instruction a donc pour effet d'augmenter de 1 la valeur de la variable `i`.

Instructions d'entrée/sortie

`lire` permet de lire au clavier une valeur et de l'affecter à la variable passée en paramètre.
`écrire` permet d'afficher à l'écran les valeurs des expressions passées en paramètres.

Exemples

```
...
var i : entier
début
    lire(i)                 ⇐ 12
    écrire("i+1=", i+1)     ⇒ i+1=13
    ...
```

Constructions conditionnelle et alternative

condition est une expression booléenne. Ces constructions peuvent être imbriquées, il est alors judicieux de bien utiliser l'indentation.

Instruction conditionnelle

```
si condition alors
        instructions
fin si
```

condition est évaluée. Si elle est vraie, les *instructions* sont exécutées avant de passer à la suite. Si elle est fausse, on passe directement à la suite.

Instruction alternative

```
si condition alors
        instructions1
sinon
        instructions2
fin si
```

condition est évaluée. Si elle est vraie, les *instructions1* sont exécutées avant de passer à la suite. Si elle est fausse, les *instructions2* sont exécutées avant de passer à la suite.

Construction de choix

expression est une expression de type `entier` ou `car`.

```
selon expression faire
    valeur1 : instructions1
    valeur2 : instructions2
    ...
    autrement : instructionsn
fin selon
...
```

L'expression est tout d'abord évaluée. Si sa valeur est une des valeurs prévues, alors les instructions correspondantes sont exécutées. Sinon les instructions associées au cas `autrement` sont exécutées. Le cas `autrement` est facultatif.

Constructions itératives ("boucles")

Les boucles permettent de répéter une liste d'instructions tant qu'une certaine condition est vraie, ou jusqu'à ce qu'une certaine condition devienne vraie, ou un certain nombre de fois. Aucune de ces boucles ne peut être interrompue autrement que par le fonctionnement normal (condition devenant fausse dans `tant que`, devenant vraie dans `répéter`, nombre d'itérations déterminé dans `pour`).
On dispose de trois constructions itératives :

```
tant que condition faire
    instructions
fin tant que
```

La condition, qui est une expression booléenne, est évaluée. Si elle est fausse, on passe à l'instruction suivant le `fin tant que`, sinon les *instructions* sont exécutées, la condition est de nouveau évaluée etc.

```
répéter
    instructions
jusqu'à condition
```

Les *instructions* sont exécutées. La *condition* est évaluée. Si elle est vraie, on passe à la suite, sinon les *instructions* sont de nouveau exécutées, la *condition* évaluée etc.

```
pour cpt de début à fin faire
    instructions
fin pour
```

cpt (la *variable de boucle*) est une variable de type `entier`. Les expressions entières *début* et *fin* sont évaluées. Les *instructions* sont exécutées *fin* − *début* + 1 fois (ou 0 si négatif), avec *cpt* valant *début*, *début*+1, ... *fin*. Les *instructions* ne doivent pas modifier la valeur de *cpt*. Pour la variante montrée en exemple ci-dessous la valeur de *cpt* est décrémentée à chaque itération.

Exemples

```
tant que i<n faire
    i ← i+1
    écrire("i=",i)
fin tant que
```

```
répéter
    lire(n)
jusqu'à n ≥ 0
```

```
pour i décroissant de n à 1 faire
    f ← f×i
fin pour
```

Commentaires

Les *commentaires* sont des textes libres encadrés par `(*` et `*)`. Ils ne sont pas destinés au processeur, qui les ignore, mais à une personne qui lit l'algorithme. On les utilise pour donner la *spécification* de l'algorithme, mais aussi pour en faciliter la lecture et la compréhension. Ils peuvent être placés au début de l'algorithme (spécification), entre les instructions (pour donner une propriété vraie à cet endroit, voir exemple ci-dessous), à côté des déclarations de variables (pour indiquer à quoi va servir une variable). Un commentaire ne doit pas paraphraser le texte de l'algorithme (c'est à dire répéter ce qui est déjà évident à la lecture) auquel cas il alourdit le texte de l'algorithme au lieu de le clarifier.

Exemples

```
...
p ← 0
pour i de 1 à n faire
    p ← p + m
    (* p = m × i *)
fin pour
(* p = m × n *)
...
```