Algorithmique

— Introduction —

Luc Fabresse
luc.fabresse@imt-nord-europe.fr

version 1.2

Plan

1. Introduction
2. Notions de base d'algorithmique

Qu'est-ce qu'un algorithme ?

Exemple d'un problème de tri

- **Entrée :** suite de nombres $\langle a_1, a_2, \ldots, a_n \rangle$
- **Sortie :** permutation (réorganisation) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ de la suite donnée en entrée, de façon que $a'_1 \leqslant a'_2 \leqslant \ldots \leqslant a'_n$

Définition d'« algorithme » [Al-Khawarizmi,825]

Description d'un processus de calcul :

- fini,
- dont chaque étape est définie complètement et sans ambigiuté,
- permettant de transformer des données prises en entrée en résultats en sortie,
- afin de résoudre un problème clairement spécifié.

Exemple d'algorithme

Le *tri par insertion*

Entrées: Un tableau d'entiers $t = \langle a_1, a_2, \ldots, a_n \rangle$
Sorties: t dont les éléments sont triés par ordre croissant

for $j \leftarrow 2$ **to** n **do** `/* parcours des éléments de t */`
  $cle \leftarrow t[j]$
  $i \leftarrow j-1$ `/*` insère $t[j]$ dans la séquence triée $t[1..j-1]$ `*/`
  while $i > 0$ ***and*** $t[i] > cle$ **do**
    $t[i+1] \leftarrow t[i]$
    $i \leftarrow i-1$
  end
  $t[i+1] \leftarrow cle$
end

Exemple d'instance de problème

31	41	59	26	41	58

$\longrightarrow$

26	31	41	41	58	59

Plan

1. Introduction
2. Notions de base d'algorithmique

Plan

1. Introduction
2. Notions de base d'algorithmique
   - Types de données
   - Manipulation de données
   - Instructions

Définitions

Un « type » est définit par :

- un ensemble de valeurs possibles pour les objets du type
- ensemble d'opérations applicables sur les objets du type
- les propriétés des objets du type
- la représentation physique de cette données (au niveau des bits)

Un « type abstrait de données » est :

- **indépendant** de toute représentation physique en mémoire
- complètement spécifié par les opérations qui lui sont applicables et par les propriétés de ces opérations

Le type abstrait « booleen »

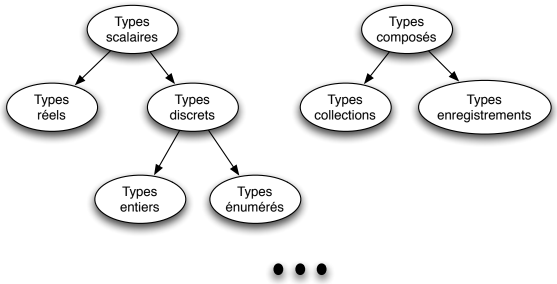
Définition

- Valeurs $= \top, \bot$
- Opérations :
  - $\neg$: $booleen \rightarrow booleen$
  - $\wedge$: $booleen \times booleen \rightarrow booleen$
  - $\vee$: $booleen \times booleen \rightarrow booleen$
- Propriétés : $\forall (a, b) \in booleen^2$
  - $\neg\top = \bot$
  - $\neg\neg a = a$
  - $\top \wedge a = a$
  - $\bot \wedge a = \bot$
  - $a \vee b = \neg(\neg a \wedge \neg b)$

Remarques

Aucune information sur la représentation mémoire des booléens

Familles de types



Types scalaires

- valeurs non décomposables
- quelques opérateurs prédéfinis (affectation, comparaison, ...)

Types scalaires prédéfinis

Le type abstrait « entier » (relatif)

- Valeurs $= \mathbb{Z}$
- Opérateurs :
  - unaires $(+,-)$
  - binaires arithmétiques $(+,-,*,div,mod)$
  - binaires de comparaison $(=,<,>,\leqslant,\ldots)$
- Propriétés : $\forall x \in entier$
  - $x+0=x$
  - $x-x=0$
  - $x*0=0$
  - ...

Types scalaires prédéfinis

Le type abstrait « entier » (relatif)

- Valeurs $= \mathbb{Z}$
- Opérateurs :
  - unaires $(+,-)$
  - binaires arithmétiques $(+,-,*,div,mod)$
  - binaires de comparaison $(=,<,>,\leqslant,\ldots)$
- Propriétés : $\forall x \in entier$
  - $x+0=x$
  - $x-x=0$
  - $x*0=0$
  - ...

Le type abstrait « caractère »

Valeurs $= [a..z] \cup [A..Z] \cup [0..9]$
...

Le type abstrait « chaîne de caractères »

Opérations : longueur(ch), element(ch,i), début(ch,n), fin(ch,n), concat(ch1,ch2), ...
...

Plan

1 Introduction

2 Notions de base d'algorithmique
- Types de données
- Manipulation de données
- Instructions

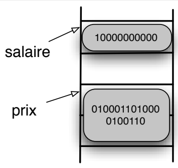
Variable

Variable $= \langle Identificateur, Type, Valeur \rangle$

- *Identificateur* : chaîne de caractères permettant de désigner un espace mémoire
- *Type* : indique comment interpréter le contenu de l'espace mémoire
- *Valeur* : contenu de l'espace mémoire

Exemples

entier Salaire ← 1024
Salaire ← 2048
reel Prix ← 71.3
Prix ← 35.0



Remarques

- À l'exécution, l'*identificateur* et le *type* sont **invariants** contrairement à la *valeur*
- L'*opérateur d'affectation* (←) permet de fixer la valeur d'une variable (écrasant l'ancienne s'il y en avait une)

Constante

Définition

Une variable dont la valeur ne peut être fixée qu'une seule fois

Exemples

constante *reel* PI ← 3.14
constante *reel* TVA ← 0.19

Expressions

Une expression de type T est :

- soit une valeur de type T
- soit une variable de type T
- soit une constante de type T
- soit une combinaison formée à partir de valeurs, de variables, de constantes, d'expressions et d'opérateurs dont le résultat est de type T

Exemples

entier A, B
booleen ma_verite
reel R1
constante *reel* TVA ← 0.19
(A+B*2) / 10
not(ma_verite)
(A<B) or (R1*TVA ≠ 0.0)

Expressions

Une expression de type T est :

- soit une valeur de type T
- soit une variable de type T
- soit une constante de type T
- soit une combinaison formée à partir de valeurs, de variables, de constantes, d'expressions et d'opérateurs dont le résultat est de type T

Exemples

entier A, B
booleen ma_verite
reel R1
constante *reel* TVA ← 0.19
(A+B*2) / 10
not(ma_verite)
(A<B) or (R1*TVA ≠ 0.0)

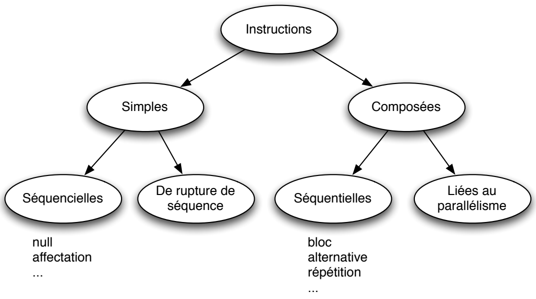
Règle d'or :

Utiliser des parenthèses !

Plan

1. Introduction
2. Notions de base d'algorithmique
   - Types de données
   - Manipulation de données
   - Instructions

Instructions



Remarque

Des instructions s'enchaînent *séquentiellement*, quand la fin d'une instruction déclenche l'exécution de la suivante.

Deux instructions séquentielles simples

`null`

« Ne fait rien ». Aucun effet.

Affectation

Notation : `variable` ← `expression`

Effet :

1. évaluation de *expression* qui a pour résultat une valeur *v*
2. modification de la valeur de *variable* qui devient *v*

Conditions :

- `variable` doit être *déclarée* et *visible*
- `expression` doit être *évaluable*
- `expression` et `variable` doivent être du **même** type

Bloc d'instructions

Notation

begin
  `instruction_1`
  `instruction_2`
  ...
  `instruction_n`
end

Effet

Séquence d'instructions inséparables signifie que l'exécution ne peut commencer que par la première instruction du bloc et ne peut se terminer que par la dernière.

Conditions

Aucune

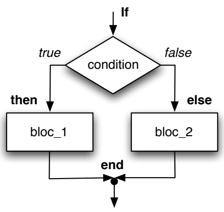
Alternative

Notation

if `condition` **then**
  `instruction_1`
else
  `instruction_2`
end

Effet

1. évaluation de `condition` qui donne une valeur booléenne *v*
2. si *v* est vrai, exécution de `instruction_1` puis passer à l'instruction qui suit le **end** sinon exécution de `instruction_2` puis passer à l'instruction qui suit le **end**.



Conditions

`condition` est une expression évaluable de type **booleen**

Exercice

Problème

Calculer les racines d'un polynome du second degré.

Exercice

Problème

Calculer les racines d'un polynome du second degré.

Algorithme

Entrées: $a, b, c \in reel^3$ coefficients du polynôme $ax^2 + bx + c$
Sorties: $ok \in booleen$ indiquant l'existence de solutions et les solutions $x_1, x_2 \in reel^2$

Exercice

Problème

Calculer les racines d'un polynome du second degré.

Algorithme

Entrées: $a, b, c \in reel^3$ coefficients du polynôme $ax^2 + bx + c$
Sorties: $ok \in booleen$ indiquant l'existence de solutions et les solutions $x_1, x_2 \in reel^2$

$delta \leftarrow (b * b) - (4 * a * c)$
if $delta \leqslant 0$ **then**
  $ok \leftarrow vrai$
  if $delta \leq 0$ **then**
    $x_1 \leftarrow (-b + sqrt(delta))/(2 * a)$
    $x_2 \leftarrow (-b - sqrt(delta))/(2 * a)$
  else
    $x_1 \leftarrow -b/(2 * a)$
    $x_2 \leftarrow x_1$
  end
else
  $ok \leftarrow faux$
end

Répétition *pour*

Notation

for *compteur* ← *valeurInitiale* **to** *valeurFinale* **do**
    `instruction`
end

Effet

1. *compteur* est une variable de type `entier` dont la valeur est fixée à *valeurInitiale*
2. si la valeur de *compteur* est inférieur à *valeurFinale* alors `instruction` est exécutée sinon l'instruction après le **end** est exécutée
3. la valeur de *compteur* est **incrémentée** de 1
4. retour à l'étape 2

Exemple

sum, *i* : **entier**
sum ← 0
for *i* ← 1 **to** 10 **do**
    sum ← *sum* + *i*
end

$$\sum_{i=1}^{10} i = 1+2+\ldots+10$$

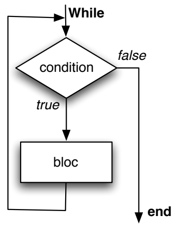
Répétition *tant que*

Notation

while `condition` **do**
    `instruction`
end

Condition

`condition` est une expression évaluable de type **booleen**



Répétition *tant que*

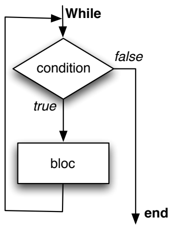
Notation

while `condition` **do**
    `instruction`
end

Condition

`condition` est une expression évaluable de type **booleen**

Attention aux boucles infinies !



Fin sur les instructions

A savoir :

- D'autres instructions existent
- Des variantes existent

Dans la suite

Les instructions du langage C