Algorithmique fonctionnelle - Introduction à Scheme

8 octobre 2009

Scheme, langage interprété et fonctionnel

Scheme, langage interprété et fonctionnel

- Quelques cours...
  - http ://www.ccs.neu.edu/home/dorai/t-y-scheme/t-y-scheme.html
  - http ://www.scheme.com/tspl3/
- Les liens logiciels
  - http ://www.plt-scheme.org/
  - http ://www.scheme.com/

Introduction

Historique

- Dans les années 30, Alonzo Church invente le λ-calcul sur lequel repose le langage LISP, ancètre de SCHEME
- En même temps, Alan Turing invente le modèle des machines de Turing
- Durant la 2e guerre mondiale, les principes de Turing sont mis en oeuvre pour décoder les messages
- Depuis les années 50, les évolutions des ordinateurs ont orienté l'informatique vers une multitude de domaines
  - Etude des langages
  - Système (architecture, réseaux...)
  - Intelligence artificielle
  - Internet
  - ...

Introduction

Programmer, c'est quoi ?

- Ecrire dans un langage spécifique, compréhensible par l'ordinateur, une suite d'instructions traitant des informations
- Adapter un algorithme en pseudo-code à un langage informatique donné
- Démarche :

Introduction

Programmer, c'est quoi ?

- Ecrire dans un langage spécifique, compréhensible par l'ordinateur, une suite d'instructions traitant des informations
- Adapter un algorithme en pseudo-code à un langage informatique donné
- Démarche :
  1. Analyse du problème, décomposition en sous-problèmes

Introduction

Programmer, c'est quoi ?

- Ecrire dans un langage spécifique, compréhensible par l'ordinateur, une suite d'instructions traitant des informations
- Adapter un algorithme en pseudo-code à un langage informatique donné
- Démarche :
  1. Analyse du problème, décomposition en sous-problèmes
  2. Mise en place des algorithmes pour chacun des sous-problèmes

Introduction

Programmer, c'est quoi ?

- Ecrire dans un langage spécifique, compréhensible par l'ordinateur, une suite d'instructions traitant des informations
- Adapter un algorithme en pseudo-code à un langage informatique donné
- Démarche :
  1. Analyse du problème, décomposition en sous-problèmes
  2. Mise en place des algorithmes pour chacun des sous-problèmes
  3. Codage des algorithmes dans le langage donné

Introduction

Programmer, c'est quoi ?

- Ecrire dans un langage spécifique, compréhensible par l'ordinateur, une suite d'instructions traitant des informations
- Adapter un algorithme en pseudo-code à un langage informatique donné
- Démarche :
  1. Analyse du problème, décomposition en sous-problèmes
  2. Mise en place des algorithmes pour chacun des sous-problèmes
  3. Codage des algorithmes dans le langage donné
  4. Tests, corrections, validation

Introduction

Langages

- 1ersProgrammes : langages de bas niveau (ex : assembleur), dépendant de la machine, instructions compréhensibles par le processeur, compliquées à programmer
- Aujourd'hui : langages de haut niveau, indépendants de la machine, plus facile à programmer
  - années 50 : LISP, Fortran
  - années 70, 80 : Basic, Pascal, C, Ada, Prolog
  - années 90 : Java

Introduction

Types de langage

- **compilés :** le programme écrit, un compilateur le traduit dans un langage compréhensible par la machine et produit un exécutable
- **interprétés :** un interprète évalue **à la volée** les différentes expressions à exécuter et renvoie chaque fois la réponse qu'il a calculée

Scheme, langage interprété et fonctionnel

Scheme, langage interprété et fonctionnel

- Simplification de LISP
- **fonctionnel** : écriture de fonctions
- Analogue aux mathématiques : paramètres en entrée, résultat en sortie
- Interprété :
  1. Lecture d'une expression
  2. Evaluation de l'expression
  3. Affichage du résultat

Principes du langage

Principes principaux

- Parenthésé : définit le debut et la fin de toute instruction/déclaration
- Instruction préfixée : le 1^{er}argument est l'opération à effectuer, les arguments suivants sont les opérandes nécessaires à l'opération

Exemples :

(+ 7 5)
(* 4 3)
(/ 36 3)

Exemples

x + y	
a + b + c + d	
x - y + z	
1 + (x mod 2)	
(3 - ln x)/2	
1+2+(4*3)	
racine carrée de 100 !	

Exemples

x + y	(+ x y)
a + b + c + d	(+ a b c d)
x - y + z	(+ (- x y) z)
1 + (x mod 2)	(+ 1 (modulo x 2))
(3 - ln x)/2	(/ (- 3 (log x)) 2)
1+2+(4*3)	(+ 1 2 (* 4 3))
racine carrée de 100 !	(sqrt (factorielle 100))

Principes du langage

Types de base

- Entiers, rationnels, réels, complexes
- Booléens (#t, #f)
- Chaines ("toto")
- Symboles ('toto12, '+)

Principes du langage

Opérateurs

- Entiers : **+, -, *, quotient, remainder, =**, $<$, $<=$
- Rationnels, réels, complexes : **+, -, *, /**
- Booléens : **and, or, not**
- Chaines : **string-length, string-append, string=?**
- Symboles : **equal ?**

Principes du langage

Types de base

- On ne spécifie pas explicitement le type des données, il est fixé par le résultat du calcul, du test
- Nécessite une **rigueur** particulière... ainsi que l'utilisation de **prédicats**

Exemples :

"bonjour", 'bonjour, 1.25
(+ 1/3 1/4)
(sqrt 12), (quotient 5 3), (modulo 5 3)
(integer ? 5), (rational ? 1/2), (not (real ? 8))
(and (< x 8) (> x 2))

Principes du langage

Syntaxe

- Définition de constantes :
  (**define** *maConstante saValeur*)
-

Exemple :

(**define** pi 3.14159)

> pi
3.14159
> (* 2 pi)
6.28318

Principes du langage

Fonctions

- La notation préfixée s'utilise de façon identique pour les fonction :
  $f(x, y) \leftrightarrow (f\ x\ y)$
- Il faut autant de paramètres en entrée que définis dans dans la signature de la fonction

Exemples :

sin(ax+b)	
f(g(x + 1))	

Principes du langage

Syntaxe

- Déclaration d'une fonction :
  (**define** (maFonction param1 param2 param3 . . .)
  (*code de la fonction*))
- Pas d'instruction `retourner` !, le résultat du calcul est directement renvoyé

Exemple :

```
; fonction aire
(define (aire r)
   (* pi r r)
)
```

Principes du langage

Syntaxe

- Déclaration d'une fonction :
  (**define** (maFonction param1 param2 param3 . . .)
  (*code de la fonction*))
- Pas d'instruction `retourner` !, le résultat du calcul est directement renvoyé

Exemple :

Ecrire une fonction qui renvoie la moyenne de deux nombres

Principes du langage

Syntaxe

- Déclaration d'une fonction :
  (**define** (maFonction param1 param2 param3 . . .)
  (*code de la fonction*))
- Pas d'instruction `retourner` !, le résultat du calcul est directement renvoyé

Exemple :

Ecrire une fonction qui renvoie la moyenne de deux variables

```
; fonction moyenne
(define (moyenne x y)
   (/ (+ x y) 2)
)
```

Principes du langage

Évaluation des expressions

L'évaluation d'une expression composée, de la forme : (f a1 a2 ... an) se fait en deux étapes :

1. l'interprète évalue chacun des paramètres
2. il applique la fonction f aux valeurs des arguments et retourne le résultat

Exemples :

- (* 2 3) : l'interprète évalue les nombres 2 et 3 dont les évaluations sont 2 et 3, puis leur applique la fonction + et retourne 7
- (+ (* 2 5) (* 3 4)) : l'interprète évalue d'abord les expressions (* 2 5) et (* 3 4) dont les résultats sont 10 et 12, puis l'expression (+ 10 12) et retourne 22

Principes du langage

Instructions conditionnelles : si... sinon... finsi

- (**if** (*test*) (*action si vrai*) (*action si faux*))

Exemples :

```
(define (parite x)
   (if (= (modulo x 2) 0)
      "pair"
      "impair"
   )
)
```

Ecrire une fonction qui renvoie la valeur absolue d'un nombre

Principes du langage

Instructions conditionnelles : si... sinon... finsi

- (**if** (*test*) (*action si vrai*) (*action si faux*))

Exemples :

```
(define (parite x)
   (if (= (modulo x 2) 0)
      "pair"
      "impair"
   )
)
```

Ecrire une fonction qui renvoie la valeur absolue d'un nombre

```
(define (valeur-absolue x)
   (if (>= x 0)
      x
      (- x)
   )
)
```

Principes du langage

Instruction conditionnelle : si... sinonsi... sinonsi... sinon... finsi

- (**cond** ((*test 1*) action 1)
  ...
  ((*test n*) action n)
  (**else** action par defaut))

Exemples :

Ecrire une fonction mention qui, selon la valeur d'une note, affiche la mention correspondante ou "Recalé" si la note < 10

Principes du langage

Instruction conditionnelle : si... sinonsi... sinonsi... sinon... finsi

- (**cond** ((*test 1*) action 1)
  ...
  ((*test n*) action n)
  (**else** action par defaut))

Exemples :

Ecrire une fonction mention qui, selon la valeur d'une note, affiche la mention correspondante ou "Recalé" si la note < 10

```
(define (mention note)
   (cond ((>= note 16) "TB")
      ((>= note 14) "B")
      ((>= note 12) "AB")
      ((>= note 10) "P")
      (else "Recalé !")
))
```

Principes du langage

Exemples :

Ecrire une fonction qui renvoie le max de 2 nombres
Ecrire une fonction qui renvoie le max de 3 nombres

Principes du langage

Exemples :

Ecrire une fonction qui renvoie le max de 2 nombres

```
(define (valeur-max a b)
   (if (> a b)
      a
      b)
)
```

Ecrire une fonction qui renvoie le max de 3 nombres

```
(define (valeur-max-3 a b c)
   (valeur-max (valeur-max a b) c)
)
```