FSAB 1402: Informatique 2

Introduction et Concepts de Base

Peter Van Roy
Département d'Ingénierie Informatique, UCL

pvr@info.ucl.ac.be

Transparents inspirés par Christian Schulte et Seif Haridi

Résumé du premier cours

- Organisation du cours
- Résumé du cours
- Introduction aux concepts de base

Organisation du cours

FSAB 1402

- Séances magistrales et séances pratiques
  - Cours magistraux pour introduire les concepts
    - Jeudi 8h30-10h30: BA93
    - Jeudi 10h45-12h45: BA91
  - Séances pratiques pour maîtriser les concepts
    - Mercredi 8h30-10h30 (Bac FSA)
    - Mardi 16h15-18h15 (Bac SINF)
- Evaluation
  - Deux piliers: la pratique (écrire des programmes) et la théorie (définition précise des concepts, sémantique)
  - Interrogation lundi 31 octobre
  - Projet semaines 7-9
  - Examen à la fin

Equipe

- Titulaires
  - Peter Van Roy (pvr@info.ucl.ac.be)
  - Pierre Dupont (pdupont@info.ucl.ac.be)
- Assistants
  - Raphaël Collet (raph@info.ucl.ac.be)
  - Yves Jaradin (yjaradin@info.ucl.ac.be)
  - Boris Mejias (bmc@info.ucl.ac.be)
  - Luis Quesada (luque@info.ucl.ac.be)
- Etudiants/moniteurs
  - Yannick Alsberge (yalsberg@student.fsa.ucl.ac.be)
  - Jean-Baptiste Escoyez (jescoyez@student.fsa.ucl.ac.be)
  - Benoît Frenay (bfrenay@student.fsa.ucl.ac.be)
  - Thibault Helleputte (thellepu@student.fsa.ucl.ac.be)
  - Christophe Klopfert (cklopfer@student.fsa.ucl.ac.be)
  - Sébastien Mouthuy (smouthuy@student.fsa.ucl.ac.be)
  - Ariel Ouziel (aouziel@student.fsa.ucl.ac.be)
  - Damien Saucez (dsaucez@student.fsa.ucl.ac.be)

Matière

- Cours basé sur le livre

  Concepts, Techniques, and Models of Computer Programming, MIT Press, 2004
- Disponibilité
  - A la DUC et au SICI (livre de référence)
  - A la bibliothèque INGI et la BSE (si vous ne voulez pas acheter le livre)
- Informations sur les TPs et des informations supplémentaires disponibles à:

  http://www.info.ucl.ac.be/notes_de_cours/FSAB1402/

Structure des séances magistrales

- Rappel de la dernière séance
- Résumé de ce qu'on va voir
- Contenu
- Résumé de ce qu'on a vu
- Suggestions pour la lecture

Séances pratiques

- Elles commencent la seconde semaine!
- Pour les Bac FSA
  - Semaine 2 et 3: Candix, DAO, IAO
  - Autres semaines: Salles Mercator
- Pour les Bac SINF
  - Semaine 3: (sera affiché)
  - Autres semaines: (sera affiché)

Séances pratiques

- Buts
  - Apprendre la pratique de la programmation
  - Répéter la matière des cours magistraux
  - Répondre aux questions sur la matière
  - Approfondir votre compréhension
- Exercices et problèmes
  - Comprendre les concepts et les mettre en pratique
  - Les encadrants vous aideront à trouver les réponses; ils ne donneront pas les réponses eux-mêmes!
- Une bonne préparation pour l'interro!

Horaire

- Attention: l'horaire du cours a quelques petites irrégularités
  - Le site Web du cours contiendra toujours les données exactes
  - Il y a deux semaines sans cours; il y a une semaine avec deux cours

Suggestions de lecture pour le premier cours

- Transparents sur le site Web du cours
- Dans le livre
  - Chapitre 1 (1.1, 1.2, 1.3) - Introduction aux concepts de base
  - Chapitre 2 (2.1) - Définition des langages de programmation
  - Chapitre 2 (2.3) - Langage de base
  - Chapitre 2 (2.4.1) - Concepts de base (identificateurs, variables, environnement, portée, fonctions et procédures)

Environnement de programmation

- http://www.mozart-oz.org/
  - Langage: Oz
  - Système: Mozart
  - Éditeur: Emacs
  - Système interactif

- Disponible sur l'infrastructure FSA, sous Linux et Windows
- Vous pouvez l'installer sur vos ordinateurs personnels
- Si vous avez un ordinateur portable, vous pouvez déjà installer Mozart avant les séances pratiques

Des commentaires et suggestions sont bienvenus!

- Sur la structure ou le contenu du cours
- N'hésitez pas à me contacter ou contacter un encadrant
- Si vous voulez me contacter, prenez un rendezvous s'il vous plaît!
  - Par email: pvr@info.ucl.ac.be
  - Par téléphone: 010 47 83 74

Questions et freins!

- Posez des questions quand ce n'est pas clair
  - Pour répéter une explication
  - Pour donner une meilleure explication
  - Pour donner un exemple
- Dites-moi quand je vais trop vite!
- Dites-moi quand je vais trop lentement!

Résumé du cours

Objectif du cours

Donner une introduction à la discipline de la programmation, en cinq thèmes:

1. La programmation fonctionnelle (3 semaines)
   - Un programme est une fonction mathématique
2. Les techniques de programmation (3 semaines)
   - La complexité calculatoire
   - Les algorithmes sur les listes et les arbres
3. La sémantique formelle des langages (1 semaine)
   - On ne peut pas maîtriser ce qu'on ne comprend pas
4. L'abstraction de données (4 semaines)
   - Partitionner un problème pour maîtriser la complexité
   - Les deux approches: objets et types abstraits
   - Le langage Java
5. La concurrence et les systèmes multi-agents (1 semaine)

Il y a beaucoup de manières de programmer un ordinateur!

- Programmation déclarative
  - Programmation fonctionnelle ou programmation logique
- Programmation concurrente
  - Par envoi de messages ou par données partagées
- Programmation avec état
- Programmation orientée objet
- Programmation par composants logiciels

Modèles de programmation (“paradigmes”)

- Mettre ensemble
  - Des types de données et leurs opérations
  - Un langage pour écrire des programmes
- Chaque modèle/paradigme permet d’autres techniques de programmation
  - Les paradigmes sont complémentaires
- Le terme “paradigme de programmation”
  - Très utilisé dans le monde commercial; attention au sens (buzzword)!

Langages de programmation

- Différents langages soutiennent différents modèles/paradigmes
  - Java: programmation orientée objet
  - Haskell: programmation fonctionnelle
  - Erlang: programmation concurrente pour systèmes fiables
  - Prolog: programmation logique
  - …
- Nous voudrions étudier plusieurs paradigmes!
- Est-ce qu'on doit étudier plusieurs langages?
  - Nouvelle syntaxe…
  - Nouvelle sémantique…
  - Nouveau logiciel…

La solution pragmatique…

- Un seul langage de programmation
  - Qui soutient plusieurs modèles de programmation
  - Parfois appelé un langage "multi-paradigme"
- Notre choix est Oz
  - Soutient ce qu'on voit dans le cours, et plus encore
- L'accent sera mis sur
  - Les modèles de programmation!
  - Les techniques et les concepts!
  - **Pas** le langage en soi!

Comment présenter plusieurs modèles de programmation?

- Basé sur un langage noyau
  - Un langage simple
  - Un petit nombre de concepts **significatifs**
  - Buts: simple, minimaliste
- Langage plus riche au dessus du langage noyau
  - Exprimé en le traduisant vers le langage noyau
  - But: soutenir la programmation pratique

Approche incrémentale

- Commencer par un langage noyau simple
  - Programmation fonctionnelle
- Ajouter des concepts
  - Pour obtenir les autres modèles de programmation
  - Très peu de concepts!
  - Très peu à comprendre!
- En FSAB1402 nous ne verrons que quelques modèles, à cause de la taille limitée du cours
  - Les autres modèles peuvent être vus dans d'autres cours (par exemple, en INGI2131 ou INGI2365) ou en lisant plus loin dans le livre

Les modèles que vous connaissez!

- Vous connaissez tous le langage Java, qui soutient
  - La programmation avec état
  - La programmation orientée objet
- C'est clair que ces deux modèles sont importants!

Pourquoi les autres modèles?

- Deux nouveaux modèles qu'on verra dans ce cours
  - Programmation déclarative (fonctionnelle)
  - Programmation concurrente (multi-agent) avec dataflow
- D'autres modèles pas vu dans ce cours
  - Programmation concurrente par envoi de messages ou par données partagées
  - Programmation logique (déterministe et nondéterministe)
  - Programmation par contraintes
  - Programmation par composants
  - …
- Est-ce que tous ces modèles sont importants?
  - Bien sûr!
  - Ils sont importants dans beaucoup de cas, par exemple pour les systèmes **complexes**, les **systèmes multi-agents**, les systèmes à **grande taille**, les systèmes à **haute disponibilité**, etc.
  - On reviendra sur ce point plusieurs fois dans le cours

Notre premier modèle

- La programmation **déclarative**
  - Deux formes: programmation fonctionnelle et programmation logique
  - On peut considérer la programmation fonctionnelle comme la base de la plupart des autres modèles
    - On regardera uniquement la programmation fonctionnelle
- Approche
  - Introduction informelle aux concepts et techniques importants, avec exemples interactifs
  - Introduction au langage noyau
  - Sémantique formelle basée sur le langage noyau
  - Étude des techniques de programmation, surtout la récursion (sur entiers et sur listes) et la complexité calculatoire

La programmation déclarative

- L'idéal de la programmation déclarative
  - Dire uniquement **ce que** vous voulez calculez
  - Laissez l'ordinateur trouvez **comment** le calculer
- De façon plus pragmatique
  - Demandez plus de soutien de l'ordinateur
  - Libérez le programmeur d'une partie du travail

Propriétés du modèle déclaratif

- Un programme est une fonction ou une relation au sens mathématique
  - Un calcul est l'évaluation d'une fonction ou une relation sur des arguments qui sont des structures de données
- Très utilisé
  - Langages fonctionnels: LISP, Scheme, ML, Haskell, …
  - Langages logiques (relationnels): Prolog, Mercury, …
  - Langages de représentation: XML, XSL, …
- Programmation "sans état"
  - Aucune mise à jour des structures de données!
  - Permet la simplicité

L'utilité du modèle déclaratif

- Propriété clé: "Un programme qui marche aujourd'hui, marchera demain"
  - Les fonctions ne changent pas de comportement, les variables ne changent pas d'affectation
- Un style de programmation qui est à encourager dans tous les langages
  - "Stateless server" dans un contexte client/server
  - "Stateless component"
- Pour comprendre ce style, nous utilisons le modèle fonctionnel!
  - Tous les programmes écrits dans ce modèle sont ipso facto déclaratif: une excellent manière de l'apprendre

Introduction aux Concepts de Base

Un langage de programmation

- Réalise un modèle de programmation
- Peut décrire des programmes
  - Avec des **instructions**
  - Pour calculer avec des **valeurs**
- Regardons de plus près
  - instructions
  - valeurs

Système interactif

declare

X = 1234 * 5678

{Browse X}

- Sélectionner la région dans le buffer Emacs
- Donner la région sélectionnée au système
  - La région est compilée
  - La région compilée est exécutée
- Système interactif: à utiliser comme une calculatrice
- Essayez vous-même après ce cours!

Système interactif: Qu'est-ce qui se passe?

declare

X = 1234 * 5678

{Browse X}

- **Déclarer** ("créer") une variable **désignée** par X
- **Affecter** à la variable la valeur 7006652
  - Obtenu en faisant le calcul 1234*5678
- **Appeler** la procédure Browse avec l'argument désignée par X
  - Fait apparaître une fenêtre qui montre 7006652

Variables

- Des raccourcis pour des valeurs
- Peuvent être affectées une fois au plus
  - Variables mathématiques
  - (Note: l'affectation multiple est un autre concept; on la verra plus loin dans le cours sous le nom de "cellule")
- Sont dynamiquement typées (type connu **pendant** l'exécution)
  - En Java elles sont statiquement typées: type connu **avant** l'exécution (à la compilation)
- Attention: il y a **deux** concepts cachés ici!
  - L'identificateur: le nom que vous tapez sur le clavier, c'est une chaine de caractères qui commence avec une majuscule
    Var, A, X123, OnlyIfFirstIsCapital
  - La variable en mémoire: une partie de la mémoire du système

Déclaration d'une variable

declare

X = …

- **declare** est une instruction ("statement")
  - Crée une nouvelle variable en mémoire
  - Fait le lien entre l'identificateur X et la variable en mémoire
- Troisième concept: l'environnement
  - Une fonction des identificateurs vers les variables
  - Fait la correspondance entre chaque identificateur et une variable (et donc sa valeur aussi)
  - Le même identificateur peut correspondre à différentes variables en différents endroits du programme!

Affectation

declare

X = 42

- L'affectation prend une variable en mémoire et la lie avec une valeur
- Dans le texte d'un programme, l'affectation est exprimée avec les identificateurs!
- Après l'affectation X=42, la variable qui correspond à l'identificateur X sera liée à 42

Affectation unique

- Une variable ne peut être affectée qu'à une seule valeur
  - On dit: **variable à affectation unique**
  - Pourquoi? Parce que nous sommes en modèle déclaratif!
- Affectation incompatible: signale une erreur

  X = 43
- Affectation compatible: ne rien faire

  X = 42

La redéclaration d'une variable (en fait: d'un identificateur!)

```
declare
X = 42
declare
X = 11
```

- Un identificateur peut être redéclaré
  - Ça marche parce qu'il s'agit des variables en mémoire différentes! Les deux occurrences de l'identificateur correspondent à des variables en mémoire différentes.
- L'environnement interactif ne gardera que la dernière variable
  - Ici, X correspondera à 11

La portée d'une occurrence d'un identificateur

```
local
   X
in
   X = 42  {Browse X}
   local
      X
   in
      X = 11  {Browse X}
   end
   {Browse X}
end
```

- L'instruction
  local X **in** <stmt> **end**
  déclare X qui existera entre **in** et **end**
- La portée d'une occurrence d'un identificateur est la partie d'un programme pour laquelle cet identificateur correspond à la même variable en mémoire.
- (Si la portée est déterminée par une inspection du code d'un programme, elle s'appelle portée lexicale ou portée statique.)
- Pourquoi il n'y a pas de conflit entre X=42 et X=11?
- Le troisième Browse affichera quoi?

Structures de données (valeurs)

- Structures de données simples
  - Entiers 42, ~1, 0
    Notez: "~" pour entier négatif (!)
  - Virgule flottante 1.01, 3.14, 0.5, ~3.2
  - Atomes (constantes) foo, 'Paul', nil
- Structures de données composées
  - Listes
  - Tuples, enregistrements ("records")
- Dans ce cours on utilisera principalement les entiers et les listes

Fonctions

- Définir une fonction
  - Donner une instruction qui définit ce que doit faire la fonction
- Appeler une fonction
  - Utiliser la fonction pour faire un calcul selon sa définition
  - On dit aussi: **appliquer** une fonction

Notre première fonction

- Pour calculer la négation d'un entier
  - Prend un argument: l'entier
  - Rend une valeur: la négation de l'entier
- En notation mathématique:

$$\text{minus}: \begin{cases} \text{Integer} \rightarrow \text{Integer} \\ n \rightarrow \sim n \end{cases}$$

La définition de la fonction Minus

```
declare
fun {Minus X}
   ~X
end
```

- L'identificateur Minus sera lié à la fonction
  - Déclarer une variable qui est liée à l'identificateur Minus
  - Affecter cette variable à la fonction en mémoire
- La portée de l'argument X est le corps de la fonction

L'application de la fonction Minus

```
declare
Y = {Minus ~42}
{Browse Y}
```

- Y est affecté à la valeur calculée par l'application de Minus à l'argument ~42

Syntaxe

- Définition d'une fonction

  fun {*Identificateur Arguments*}
  Corps de la fonction
  end

- Appel d'une fonction

  X = {*Identificateur Arguments*}

Fonction de maximum

- Calculer le maximum de deux entiers
  - Prend deux arguments: entiers
  - Rend une valeur: l'entier le plus grand
- En notation mathématique:

$$\max: \begin{cases} \text{Integer} \times \text{Integer} \rightarrow \text{Integer} \\ n, m \rightarrow n, & n > m \\ \quad\;\; m, & \text{otherwise} \end{cases}$$

Définition de la fonction Max

```
declare
fun {Max X Y}
   if X>Y then X
   else Y
   end
end
```

- Nouvelle instruction: conditionnel (if-then-else)
- Le conditionnel renvoie un résultat

Application de la fonction Max

```
declare
X = {Max 42 11}
Y = {Max X 102}
{Browse Y}
```


Maintenant le minimum

- Une possibilité: couper et coller
  - Répéter ce qu'on a fait pour Max
- Mieux: la **composition** de fonctions
  - Reutiliser ce qu'on a fait avant
  - C'est une bonne idée de reutiliser des fonctions compliquées
- Pour le minimum de deux entiers:

  min(*n*,*m*) = ~max(~*n*,~*m*)

Définition de la fonction Min

```
declare
fun {Min X Y}
   {Minus  {Max  {Minus X}
                 {Minus Y}}}
end
```


Définition de la fonction Min (avec ~)

```
declare
fun {Min X Y}
   ~{Max ~X ~Y}
end
```

Définition inductive d'une fonction

- Fonction de factorielle *n*!
  - La définition inductive est

    0! = 1

    n! = *n* * ((*n*-1)!)
  - Programmer ceci en tant que fonction Fact
- Comment procéder?
  - Utiliser l'application **récursive** de Fact!

Définition de la fonction Fact

```
fun {Fact N}
   if N==0 then     % Test d'égalité
      1
   else
      N * {Fact N-1} % Appel récursif
   end
end
```

Récursion

- Structure générale
  - Cas de base
  - Cas récursif
- Pour un nombre naturel *n*, souvent:
  - Cas de base: *n* est zéro
  - Cas récursif: *n* est différent de zéro
    n est plus grand que zéro
- Beaucoup plus: la semaine prochaine!
  - La récursion c'est la manière de faire une boucle en modèle déclaratif

Une fonction est un cas particulier d'une procédure

- Le vrai concept de base est la **procédure**
- Une fonction est une procédure avec un argument en plus, qui est utilisée pour renvoyer le résultat
- Z={Max X Y} *(fonction Max)*
  est équivalent à:
  {Max X Y Z} *(procédure Max, avec un argument de plus)*

Résumé

- Système interactif
- “Variables”
  - Déclaration
  - Affectation unique
  - Identificateur
  - Variable en mémoire
  - Environnement
  - Portée d’une occurrence d’un identificateur
- Structures de données
  - Entiers
- Fonctions
  - Définition
  - Appel (application)
- Récursion

A la prochaine fois!

- Les séances pratiques
  - Mercredi 8h30-10h30 pour Bac FSA
  - Mardi 16h15-18h15 pour Bac SINF
  - La semaine prochaine, j’attends de vous que vous vous familiarisez avec la plate-forme Mozart et son environnement
- Vous pouvez déjà regarder l’énoncé du premier TP sur le Web
  - Je vous conseille d’installer Mozart chez vous! C’est facile!