Fondements de l'informatique
Structures de données et algorithmes

1

Frédéric Boulanger

Février 2014

Objectifs de ce cours

Concevoir des solutions logicielles

- Identifier les problèmes
- Spécifier une solution
- Mettre en œuvre les algorithmes classiques
- Évaluer les performances d'une réalisation
- Programmer dans un langage à objets (Java)
- Vérifier que le résultat fait :
  1. ce qu'on avait spécifié
  2. ce qu'on veut qu'il fasse

Objectifs de ce cours

Fondements de l'informatique

- Comment spécifier une solution ?
- Comment évaluer les performances ?
- Ordre de grandeur de la complexité.
- Vérification : test et preuve ($\rightarrow$ cours de GL)

Objectifs de ce cours

Structures de données et algorithmes

- Structures de données classiques (listes, arbres, graphes)
- Algorithmes usuels (classements, backtracking...)
- Idiomes classiques de la programmation par objets

Structure du cours

- 22 créneaux de 1h30
- dont au moins 6h de pratique dirigée (si possible sur machines)
  - spécifications algébriques
  - complexité des algorithmes
  - arbres binaires
  - flots dans un graphe, algorithmes approchés
- 9h de travaux pratiques
  - rappels de Java (vu en MLP) : listes chaînées
  - héritage : dérivation formelle d'expressions arithmétiques
  - backtracking : coloriage optimal d'un graphe

Lien avec les autres cours d'informatique

- Modèles et Langages de Programmation — bases
  - programmation élémentaire
  - structures de contrôle
- Génie Logiciel — modéliser
  - analyse des problèmes
  - modélisation objet
  - architectures classiques des logiciels
- Architecture des Systèmes Informatiques — briques matérielles
  - Architecture d'un ordinateur
  - Fonctionnement d'un processeur
  - Comment le matériel permet au logiciel de fonctionner
  - Systèmes d'exploittaion
- Systèmes d'Information — infrastructure
  - Bases de données
  - Réseaux de communication
  - Architecture client-serveur
  - Parallélisme, distribution

Concepts clefs de ce cours

Spécification

Description de ce que doit faire un système
sans indiquer comment il le fait.

- une spécification exprime les propriétés du système (le *quoi*)
- une réalisation du système remplit la fonctionnalité
  d'une certaine façon (le *comment*)
- plusieurs réalisations d'un système peuvent correspondre
  à la même spécification
- une spécification permet de vérifier
  qu'une réalisation est correcte
- différentes réalisations peuvent avoir des propriétés
  non fonctionnelles différentes (coût, performance, robustesse)

Concepts clefs de ce cours

Spécification

Description de ce que doit faire un système
sans indiquer comment il le fait.

Exemple

Véhicule permettant de transporter deux personnes sur la route

Concepts clefs de ce cours

Spécification

Description de ce que doit faire un système
sans indiquer comment il le fait.

Exemple

Véhicule permettant de transporter deux personnes sur la route

Concepts clefs de ce cours

Spécification

Description de ce que doit faire un système
sans indiquer comment il le fait.

Exemple

Véhicule permettant de transporter deux personnes sur la route

Concepts clefs de ce cours

Spécification

Description de ce que doit faire un système
sans indiquer comment il le fait.

Exemple

Véhicule permettant de transporter deux personnes sur la route

Concepts clefs de ce cours

Spécification

Description de ce que doit faire un système
sans indiquer comment il le fait.

Exemple

Véhicule permettant de transporter deux personnes sur la route

Concepts clefs de ce cours

Spécification

Description de ce que doit faire un système
sans indiquer comment il le fait.

Exemple

Véhicule permettant de transporter deux personnes sur la route

Concepts clefs de ce cours

Algorithme

Suite d'opérations à effectuer en nombre fini sur une structure de données pour résoudre un problème.

- certaines opérations (alternative, boucle) contrôlent l'exécution de l'algorithme.
- les opérations utilisées par un algorithme dépendent de la structure de données à laquelle il s'applique.

Concepts clefs de ce cours

Exemple

Calcul du PDCD de a et b :

```
tant_que a ≠ b
   si a > b
   alors a ← a − b
   sinon b ← b − a
   fin_si
fin_tant_que
resultat a
```

Concepts clefs de ce cours

Exemple

Calcul du PDCD de *a* et *b* :

```
tant_que a ≠ b
   si a > b
   alors a ← a − b
   sinon b ← b − a
   fin_si
fin_tant_que
resultat a
```

Opérations

- type de données = entiers
- comparaison d'entiers
- soustractions d'entiers
- affectation de variables

Concepts clefs de ce cours

Exemple

Calcul du PDCD de a et b :

```
pgcd ← 1
pour i allant_de 2 à a
   si (i divise a) et (i divise b) et (i > pgcd)
   alors pgcd ← i
   fin_si
fin_pour
resultat pgcd
```

Concepts clefs de ce cours

Exemple

Calcul du PDCD de *a* et *b* :

```
pgcd ← 1
pour i allant_de 2 à a
   si (i divise a) et (i divise b) et (i > pgcd)
   alors pgcd ← i
   fin_si
fin_pour
resultat pgcd
```

Opérations

- type de données = entiers
- comparaison d'entiers
- division d'entiers
- affectation de variables

Concepts clefs de ce cours

Algorithmes : questions

- un algorithme est-il correct ?
  preuve de programme : cours de Génie logiciel
- un algorithme est-il performant ?
  complexité des algorithmes (dans ce cours)

Concepts clefs de ce cours

Réalisation d'un algorithme : programmation

```
import sys
def pgcd(a, b):
 while (a != b):
  if a > b:
   a = a - b
  else:
   b = b - a
 return a

a = int(sys.stdin.readline())
b = int(sys.stdin.readline())
print(pgcd(a,b))
```

Concepts clefs de ce cours

Réalisation d'un algorithme : programmation

```
import java.io.*;
class PGCD {
 public static int pgcd(int a, int b) {
  while (a != b) {
   if (a > b) {
    a = a - b;
   } else {
    b = b - a;
   }
  }
  return a;
 }

 public static void main(String args[]) throws IOException {
  BufferedReader buf = new BufferedReader(
                          new InputStreamReader(System.in));
  int a = Integer.parseInt(buf.readLine());
  int b = Integer.parseInt(buf.readLine());
  System.out.println(pgcd(a, b));
 }
}
```

Specification, Algorithme, Programme

- un langage de programmation impose de prendre en compte des détails qui perturbent la recherche d'une solution
- un algorithme correspond à une solution, mais à quel problème ?
- une spécification indique ce que doit faire un système
- l'expression des exigences pose le problème à résoudre

Specification, Algorithme, Programme

- un langage de programmation impose de prendre en compte des détails qui perturbent la recherche d'une solution
- un algorithme correspond à une solution, mais à quel problème ?
- une spécification indique ce que doit faire un système
- l'expression des exigences pose le problème à résoudre

Methodologie

1. définir les besoins (exigences)
2. specifier une solution
3. choisir un algorithme
4. programmer dans un langage
5. vérifier que le programme est conforme à la spécification
6. valider le système par rapport aux exigences

Spécifications algébriques

Les mathématiques au service de l'informatique