Algorithmique : Rappels

I3 - Algorithmique et programmation

Nicolas Delestre - Michel Mainguenaud

Plan. . .

Architecture des Systèmes d'Information

Votre programme s'exécute, mais...

- Connaissez-vous les mécanismes utilisés ?
- Etes vous sûr que le résultat soit juste ?
- Combien de temps devrez vous attendre la fin du calcul ?
- Y a-t-il un moyen pour obtenir le résultat plus vite ?
  - Indépendamment de la machine, du compilateur...
- Qu'est ce qu'un bon ("beau" ?) programme ?

Architecture des Systèmes d'Information

Critères de qualité

Un programme doit être :

1. lisible
2. fiable
3. maintenable
4. réutilisable
5. portable
6. correct (preuve)
7. efficace (complexité)
8. faire face à des contraintes "économiques"

donc...

On a besoin d'une méthodologie

Architecture des Systèmes d'Information

Méthodologie générale

La création d'un programme informatique passe par 4 phases :

1. La spécification (ou analyse)
2. La conception préliminaire (ou conception générale)
3. La conception détaillée
4. Le codage

Spécification 1 / 5

Définition

≪ Ensemble des activités consistant à définir de manière précise, complète et cohérente ce dont l'utilisateur a besoin ≫ (AFNOR)

La spécification possède deux étapes :

1. Reformulation du problème
2. Formalisation du problème et de sa solution

Architecture des Systèmes d'Information

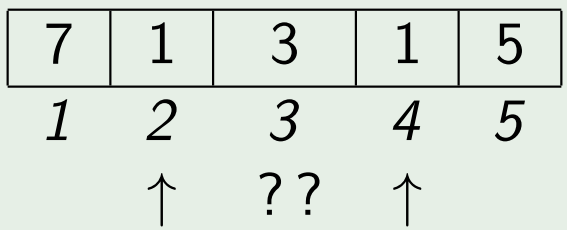
Spécification 2 / 5

Reformulation du problème

- Souvent le problème est "mal posé"

Exemple

Rechercher l'indice du plus petit élément d'une suite



Spécification 3 / 5

Problème ⇒ énoncé

Énoncé = texte où sont définies sans ambiguïté

- L'entrée (données du problème)
- La sortie (résultats recherchés)
- Les relations (éventuelles) entre les données et les résultats

Dans notre exemple

Soit I l'ensemble des indices des éléments égaux au minimum d'une suite.
Déterminer le plus petit élément de I.

Spécification 4 / 5

Formalisation

- Traduction de l'énoncé dans un langage qui pourra être ≪ compris ≫ par tout le monde (humains et ordinateurs)
  - Méthode : Analyse descendante
  - Langage : Graphique (simplification et modification de SADT/IDEF0)

Analyse descendante

- Abstraire
  - Repousser le plus loin possible l'écriture de l'algorithme
- Décomposer
  - Décomposer la résolution en une suite de "sous-problèmes" que l'on considère comme résolus
- Combiner
  - Résoudre le problème par combinaison des abstractions des "sous-problèmes"

Spécification 5 / 5

Langage graphique

- On a besoin d'un langage pour exprimer les activités ainsi que leurs interactions
- Langage graphique :
  - Une boîte *nommée* par résolution de problème avec :
    - à gauche les types des données nécessaires à la résolution du problème (information en entrée)
    - à droite les types des données résultats de la résolution du problème (information en sortie)
  - Lorsqu'une résolution A_0 a besoin d'une autre résolution A_1 (notion de dépendance) la boîte décrivant la résolution A_1 est dessinée en dessous de la boite A_0 et une flèche issue de la boîte A_0 va jusqu'à la boîte de A_1

Un exemple 1 / 3

Problème

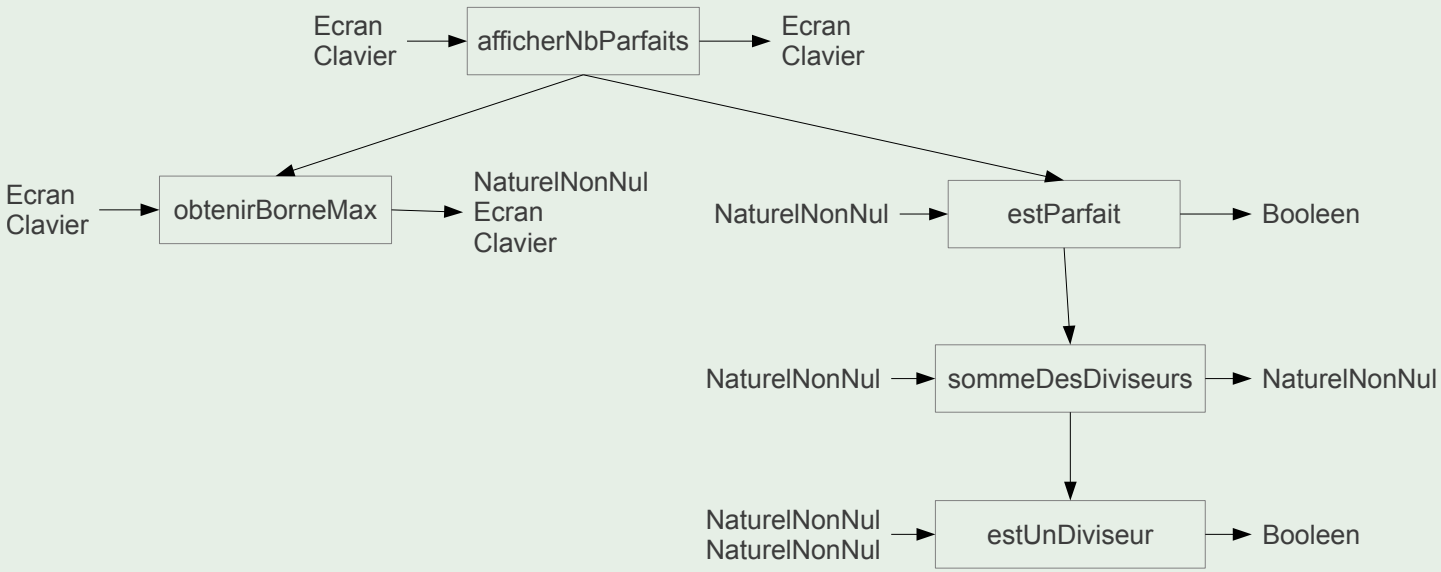
Afficher les nombres parfaits compris entre 1 et un nombre saisi par l'utilisateur.

Énoncé

Afficher les nombres parfaits (nombres égaux à la somme de leurs diviseurs) compris entre 1 et un nombre n (naturel ≥ 1) saisi par l'utilisateur.

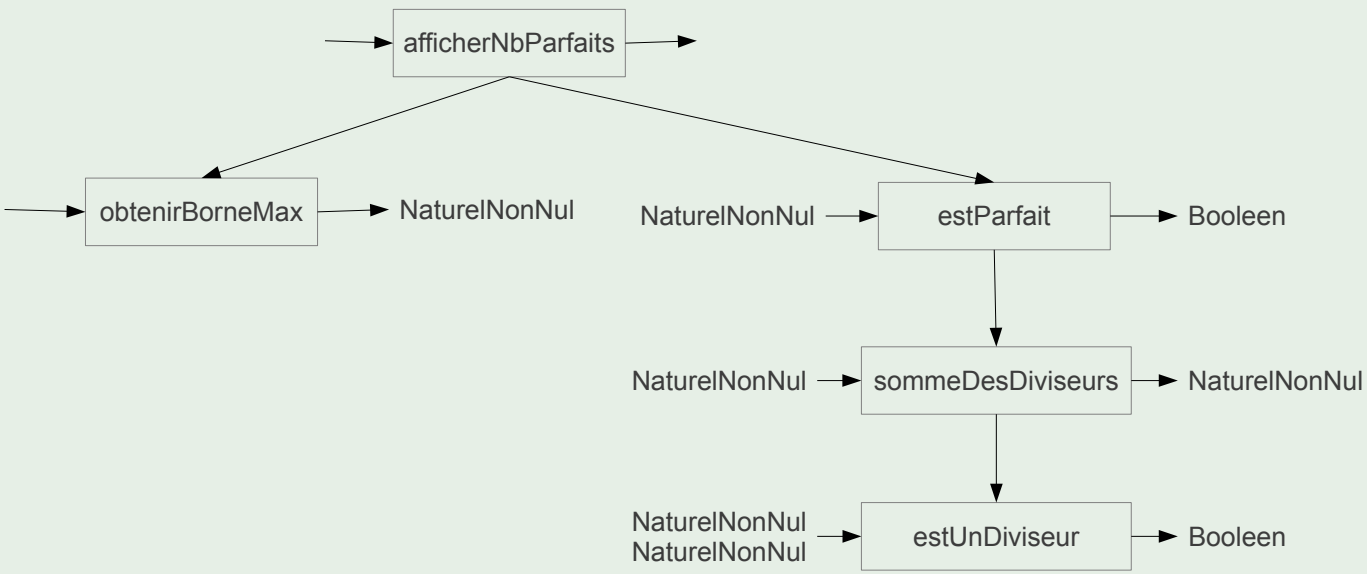
Un exemple 2 / 3

Analyse descendante



Un exemple 3 / 3

Analyse descendante



Conception préliminaire 1 / 5

Définition

≪ Ensemble des activités conduisant à l'élaboration de l'architecture du logiciel ≫ (AFNOR)

- Choix d'un paradigme de programmation
- Traduire les fonctions de l'analyse descendante avec les ≪ outils ≫ du paradigme

Conception préliminaire 2 / 5

Paradigme

≪ . . .style fondamental de programmation informatique qui traite de la manière dont les solutions aux problèmes doivent être formulées . . . ≫ (Wikipédia)

Quelques paradigmes de programmation

- **Programmation impérative**
  - **Programmation structurée**
- Programmation orientée objet (Cf. cours UML-BD en ASI3.2)
- Programmation déclarative
  - Programmation descriptive (Cf. cours Document en ASI4.2)
  - Programmation fonctionnelle
  - Programmation logique (Cf. cours Document en ASI4.2)

Attention

Un langage peut implanter plusieurs paradigmes

Conception préliminaire 3 / 5

Programmation impérative

≪ [. . .]la programmation impérative est un paradigme de programmation qui décrit les opérations en termes d'états du programme et de séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme. ≫ (Wikipédia)

- Une instruction de base : l'affectation (nommée aussi l'assignation)
- Trois organisations d'instructions :
  - Séquentiel
  - Conditionnel
  - Itératif

Conception préliminaire 4 / 5

Programmation structurée

- « . . .sous-ensemble, ou une branche, de la programmation impérative » (Wikipédia)
- Le programme est décomposé en petits sous-programmes appellés procédures ou fonctions

Conception préliminaire 5 / 5

- Dans le paradigme de la programmation structurée les « boîtes » (fonctions mathématiques) de l'analyse vont se traduire en :
  - fonction
  - procédure
- On doit préciser comment s'utilise les fonctions de l'analyse :
  - Signature de fonction et de procédure

Architecture des Systèmes d'Information

Exemple

Fonctions et procédures de la résolution du problème : afficher des nombres parfaits

procédure afficherNombresParfaits **()**
fonction obtenirBorneMax **() : NaturelNonNul**
fonction estParfait **(**n : **NaturelNonNul) : Booleen**
fonction sommeDesDiviseurs **(**n : **NaturelNonNul) : NaturelNonNul**
fonction estUnDiviseur **(**a,b : **NaturelNonNul) : Booleen**

Conception détaillée 1 / 1

Définition

≪ Ensemble des activités consistant à détailler les résultats de la conception préliminaire, tant sur le plan algorithmique que sur celui de la structure des données, jusqu'à un niveau suffisant pour permettre le codage ≫ (AFNOR)

- Besoin d'utiliser un langage formel indépendant du langage informatique qui sera utilisé : Pseudo-code

Architecture des Systèmes d'Information

Exemple

Fonction *sommeDesDiviseurs*

```
fonction sommeDesDiviseurs (n : NaturelNonNul) : NaturelNonNul

    Déclaration  diviseur,somme : NaturelNonNul
debut
    somme ← 1
    pour diviseur ←2 à n div 2 faire
        si estUnDiviseur(n,diviseur) alors
            somme ← somme+diviseur
        finsi
    finpour
    retourner somme
fin
```

Codage

Définition

≪ Activité permettant de traduire le résultat de la conception détaillée en un programme à l'aide d'un langage de programmation donné (AFNOR)

- Il existe des centaines de langages
- Utiliser celui qui est le plus adapté à votre problème
  - Pour nous le problème est d'être pédagogique : Langage Pascal

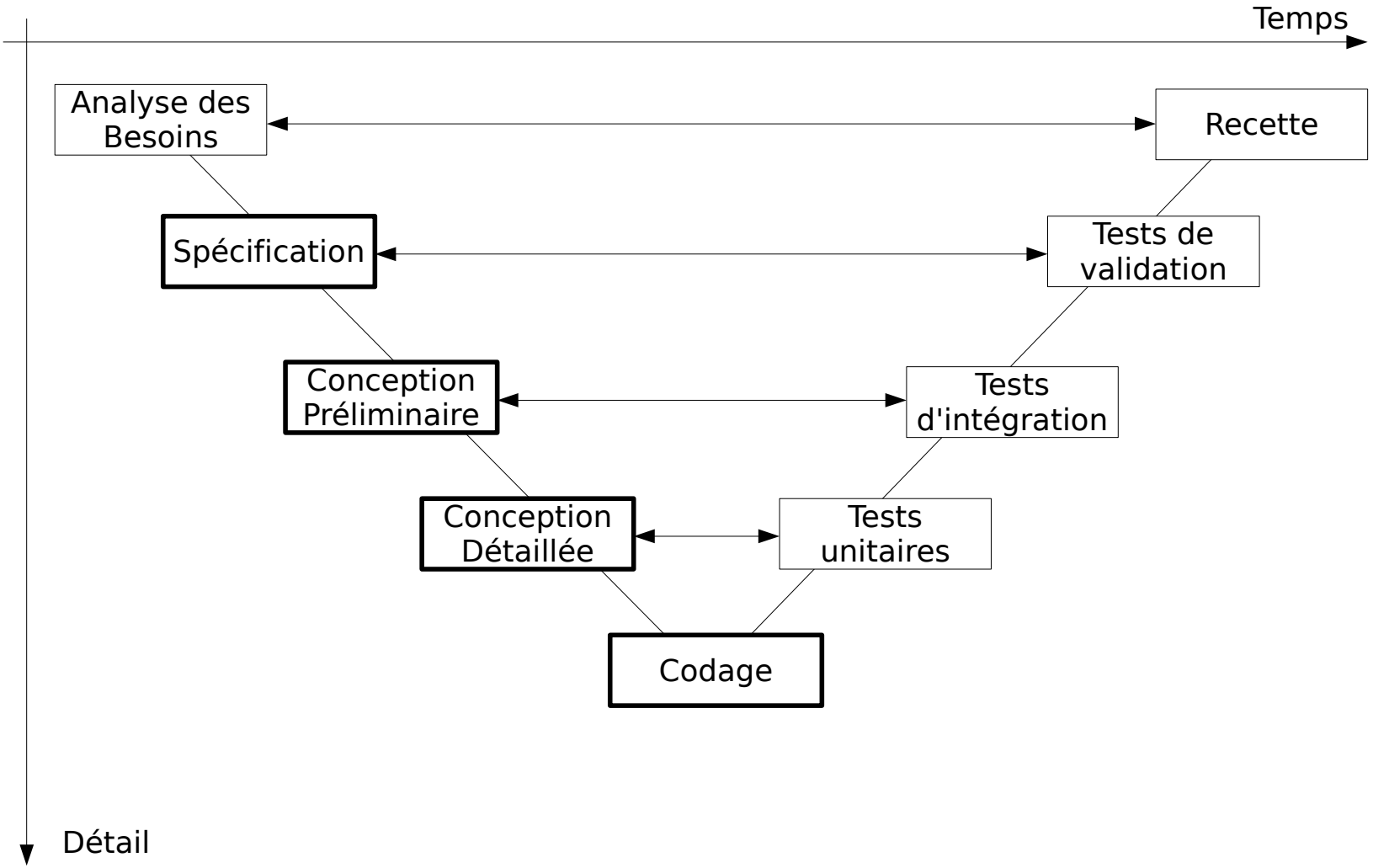
Architecture des Systèmes d'Information

Exemple

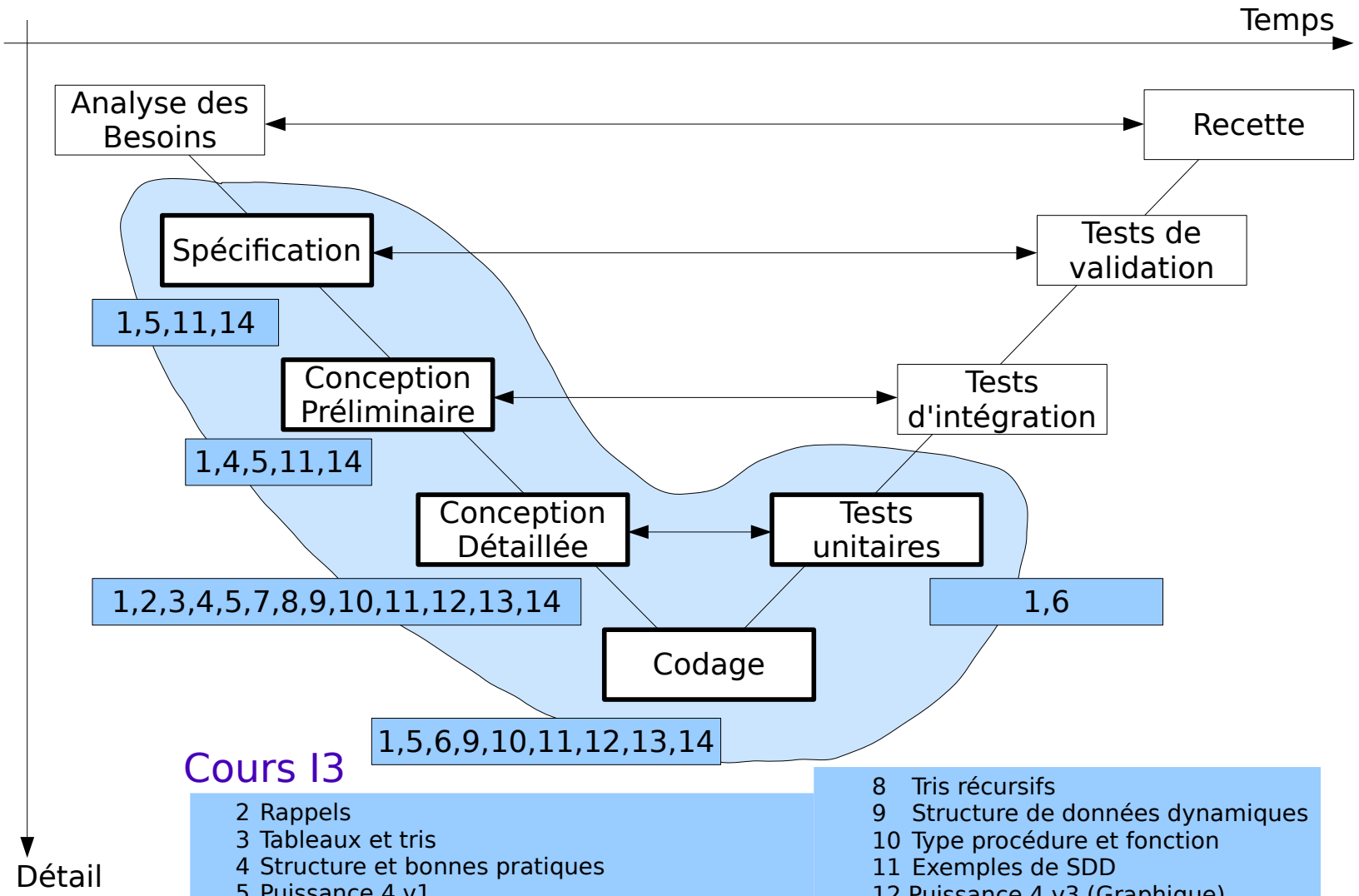
Function *sommeDesDiviseurs*

```
1 function sommeDesDiviseurs(n : Integer) : Integer;
2 var somme,i : Integer;
3 begin
4    somme:=0;
5    for i:=1 to n div 2 do
6       if estUnDiviseur(n,i) then
7          somme:=somme+i;
8    sommeDesDiviseurs:=somme
9 end; { sommeDesDiviseurs }
```

Le cycle en V 1 / 2



Le cycle en V 2 / 2



Cours I3

2 Rappels
3 Tableaux et tris
4 Structure et bonnes pratiques
5 Puissance 4 v1
6 Modularité puissance 4 v2
7 Récursivité
8 Tris récursifs
9 Structure de données dynamiques
10 Type procédure et fonction
11 Exemples de SDD
12 Puissance 4 v3 (Graphique)
13 Fichiers
14 Puissance 4 V4 (IA)

asi
Architecture des Systèmes d'Information

Pseudo-code

Objectifs

Fournir une solution au problème, il faut donc un moyen d'exprimer cette solution à l'aide d'un langage qui doit être :

- formel (pas d'ambiguïtés),
- lisible et concis,
- indépendant de tous langages informatiques
- qui reprend les concepts des langages impérifs :
  - les entités manipulées sont désignées et typées
  - suite d'actions modifiant l'état d'un programme, pour passer :
    - de l'état initial -le problème -
    - à l'état final - la solution au problème-

Architecture des Systèmes d'Information

Types

Objectifs

Classer les objets selon :

- Ensemble des valeurs que peut prendre l'objet
- Ensemble des opérations permises sur ces valeurs

- Tous les types ont un identifiant
- Tous les objets (variables, constantes, fonctions, procédures) ont un type (qui doit être déclaré)
- Toujours utiliser le type le plus en adéquation au problème

Architecture des Systèmes d'Information

Classification des types

On peut classer les types suivant deux points de vue

Prédéfinis / définis explictement

- Prédéfinis : **Booleen**, **Naturel**, **NaturelNonNul**, **Entier**, **Reel**, **ReelPositif**, **ReelPositifNonNul**, **ReelNegatif**, **ReelNegatifNonNul**, **Caractere**, **Chaine de caracteres**
- Définis explicitement (instruction **Type**) : intervalle, énuméré, tableaux, structures

Élémentaires / composites

- Booléen, Naturel, Entier, Réel, Caractère, intervalle, énuméré
- Chaîne de caractères, tableaux, structures

Architecture des Systèmes d'Information

Variables / constantes

Variables

- Entité dont la valeur peut changer au cours de l'exécution du programme (pas le type)
- Abstraction d'un emplacement mémoire de la machine de Von Neumann
- Possibilité de regroupement dans des agrégats : tableau, structure

Toute variable utilisée doit être déclarée

Constante

- Entité qui n'évolue pas
- Deux types de constantes : Constante implicite, Constante explicite
- Une constante a un type et une valeur

Opération, expression

Opération

Une opération est l'association :

- d'un opérateur (symbole, sémantique, arité)
- d'une ou plusieurs opérandes (suivant l'arité de l'opérateur). Les opérandes sont des expressions

Une opération a :

- un type
- une valeur

Expression

Une expression est une constante ou une variable ou une opération. Une expression a :

- un type
- une valeur

Exercice

- Décomposer l'expression : $2.4 * (a + x)$
- Quel est son type (sachant que a est un Naturel et x un Réel ?

Instructions / schémas

Instruction

- Modification de l'état d'un programme (affectation ou appel à des procédures) ou de structuration de l'algorithme (schéma)
- Procédure de base : lire et écrire

Schémas

- Séquentiel
- Conditionnel
- Itératif :
  - déterministe
  - indéterministe

Architecture des Systèmes d'Information

Procédure / Fonction

À partir de l'analyse descendante, on identifie les types des sous-programmes :

- fonction
  - sous programme qui calcule quelque chose (vision mathématique) : retourne une valeur
- procédure
  - sous programme qui modifie l'état du programme

Paramètre formel

Paramètre déclaré dans la signature du sous-programme

Paramètre effectif (argument)

Paramètre utilisé réellement lors de l'invocation du sous-programme

Architecture des Systèmes d'Information

Passage de paramètre 1 / 2

Lorsque l'on appelle un sous-programme on associe un paramètre effectif à un paramètre formel (suivant l'ordre de la déclaration des paramètres formels)

Définition

On nomme passage de paramètre, l'association de valeur qui est utilisée entre le paramètre effectif et le paramètre formel

Passage de paramètre en **Entrée**

Les valeurs du paramètre effectif et du paramètre formel sont les mêmes à l'entrée du sous-programme
En théorie le paramètre formel n'est pas modifié par le sous programme, jamais à gauche de $\leftarrow$

Architecture des Systèmes d'Information

Passage de paramètre 2 / 2

Passage de paramètre en **Sortie**

Les valeurs du paramètre effectif et du paramètre formel sont les mêmes à la sortie du sous-programme

Seules des variables peuvent être utilisées comme paramètre effectif

En théorie le paramètre formel, la première fois qu'il est utilisé, ne peut se trouver qu'à la gauche de ← et doit obligatoirement apparaitre

Passage de paramètre en **Entrée/Sortie**

Les valeurs du paramètre effectif et du paramètre formel sont les mêmes à l'entrée du sous-programme

Les valeurs du paramètre effectif et du paramètre formel sont les mêmes à la sortie du sous-programme

Seules des variables peuvent être utilisées comme paramètre effectif

Architecture des Systèmes d'Information

Passage de paramètre et sous programme

Fonction

Seul le passage de paramètre en Entrée est autorisé pour les fonctions
Aucune information n'est donc ajoutée dans la signature de la fonction

Procédure

Les paramètres admettent les trois types de passage de paramètre, on préfixe donc le paramètre formel dans la signature par :

- **Entrée** ou **E** pour un passage de paramètre en entrée
- **Sortie** ou **S** pour un passage de paramètre en sortie
- **Entrée/Sortie** ou **E/S** pour un passage de paramètre en entrée/sortie

Architecture des Systèmes d'Information

Fonction

fonction *nom de la fonction* **(***[paramètre(s) de la fonction]***) :** *type de la valeur retournée*

⌊**précondition(s)** *préconditions sur les paramètres*

Déclaration *variable locale 1 : type 1 ; . . .*

debut

instructions de la fonction avec au moins une fois l'instruction **retourner**

fin

Procédure

procédure *nom de la procédure* **(**[**E** *paramètre(s) en entrée ;*][**S** *paramètre(s) en sortie ;*][**E/S** *paramètre(s) en entrée/sortie*]**)**

⌊**précondition(s)** *préconditions sur les paramètres*

Déclaration *variable(s) locale(s)*

debut

instructions de la procédure

fin

Algorithmes classiques

L'algorithme c'est comme la cuisine...

- Cela peut demander de l'imagination
- Mais cela demande surtout de la méthode et de la pratique
- Il existe plusieurs types de problème que l'on rencontre constamment, par exemple :
  - Algorithmes de parcours
  - Algorithmes de recherche
  - Algorithmes d'optimisation
- Pour chaque algorithme il existe des algorithmes types qu'il faut adapter au contexte

Algorithme de parcours

Exemples

- Compter le nombre de voyelles dans une chaîne de caractères
- Mettre en majuscule les lettres d'une chaîne de caractères
- Calculer la somme des diviseurs d'un nombre
- Afficher les éléments d'un tableau

Principe

- Initialiser une variable si besoin
- Parcourir l'ensemble des éléments du problème et agir sur chaque élément

Architecture des Systèmes d'Information

Algorithme de recherche

Deux types de recherche

1. Rechercher le premier élément ayant une certaine caractéristique
2. Rechercher un individu qui se caractéristique au regard des autres éléments

Exemples

- Trouver la première voyelle d'une chaîne de caractère (cas 1)
- Trouver le plus petit élément d'un tableau (cas 2)

Principe

1. Parcourir les éléments du problème et s'arrêter dès qu'un élément résout le problème
2. Faire une hypothèse, et pour chaque élément essayer de remettre en cause l'hypothèse

Algorithme d'optimisation

Exemples

- Rendu de monnaie
- Chemin le plus court pour aller d'un point A à un point B

Principe

- De loin les problèmes les plus complexes (souvent des problèmes *NP complet*)
- Quelques algorithmes types (algorithme glouton, A*, simplexe, etc.)

Architecture des Systèmes d'Information

Calcul de complexité

Objectifs

Indépendamment de la machine, du compilateur...
Complexité :

- Taille du problème : n
- Nombre d'opérations significatives : $T(n)$
- Taille mémoire nécessaire : $M(n)$

Notations asymptotiques

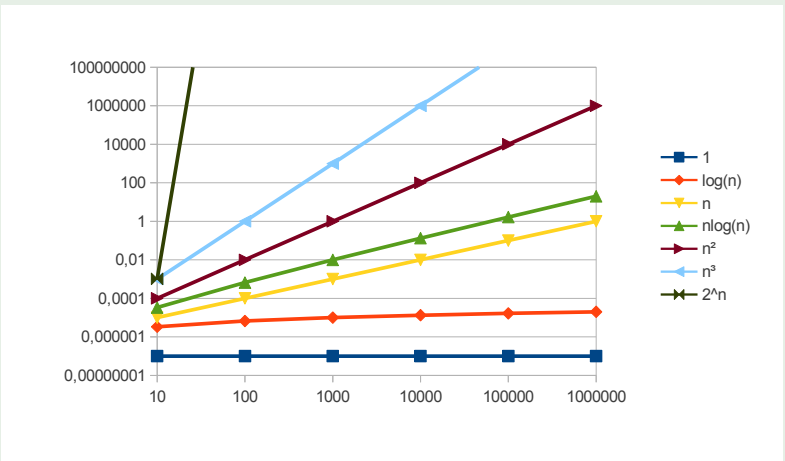
- $f(n) = O(g(n))$: borne asymptotique supérieure (au pire)
- $f(n) = \Omega(g(n))$: borne asymptotique inférieure (au mieux)
- $f(n) = \theta(g(n))$: borne approchée asymptotique (en moyenne)

asi
Architecture des Systèmes d'Information

Exemple de complexité

- 10^6 opérations par seconde
- N = nombre de données à traiter
- C = complexité de l'algorithme de traitement

N x C	1	$log_2 n$	n	$nlog_2 n$	n^2	n^3	2^n
10^2	$< 1\ \mu s$	6,6 μs	0,1 ms	0,66 ms	10 ms	1 s	4.10^6 a
10^3	$< 1\ \mu s$	9,9 μs	1 ms	9,9 ms	1 s	16,6 mn	?
10^4	$< 1\ \mu s$	13,3 μs	10 ms	0,13 s	1,5 mn	11,5 j	?
10^5	$< 1\ \mu s$	16,6 μs	0,1 s	1,66 s	2,7 h	31,7 a	?
10^6	$< 1\ \mu s$	19,9 μs	1 s	19,9 s	11,5 j	31700 a	?



Calcul de complexité 1 / 2

Notation

- Soit A une action (ou instruction), on note :
  - $O(A^o)$ sa complexité dans le pire des cas
  - $\Omega(A^\Omega)$ sa complexité dans le meilleur des cas

Cas d'un schéma séquentiel $(A_1, A_2, \ldots, A_n)$

- Dans le pire des cas : $O(max(A_1^o, A_2^o, \ldots, A_n^o))$
- Dans le meilleur des cas : $\Omega(max(A_1^\Omega, A_2^\Omega, \ldots, A_n^\Omega))$

Schéma conditionnel (C et A_1 ou A_2)

- Dans le pire des cas : $O(max(C^o, A_1^o, A_2^o))$
- Dans le meilleur des cas : $\Omega(max(C^\Omega, min(A_1^\Omega, A_2^\Omega)))$

Architecture des Systèmes d'Information

Calcul de complexité 2 / 2

Schéma itératif déterministe (A)

- Dans le pire des cas : $O(f(n) * A^o)$
- Dans le meilleur des cas : $\Omega(f(n) * A^\Omega)$
  $f(n)$ est le nombre d'itérations

Schéma itératif indéterministe (C et A)

- Dans le pire des cas : $O(f(n) * max(C^o, A^o))$
  $f(n)$ est le nombre d'itérations maximal
- Dans le meilleur des cas : $\Omega(C^\Omega)$ (tant que) ou $\Omega(max(C^\Omega, A^\Omega))$ (répéter jusqu'à ce que)

Complexité des instructions et opérations de base

- Toutes les instructions et opérations de base sont considérées comme étant en $O(1)$ et $\Omega(1)$

ion

Exemple de calcul complexité 1 / 5

estUnDiviseur, $T(n) =$

```
fonction estUnDiviseur (a, b : NaturelNonNul) : Booleen
debut
    retourner a mod b=0
fin
```

$O(1)$ et $\Omega(1)$

Exemple de calcul complexité 2 / 5

sommeDesDiviseurs, $T(n) =$

```
fonction sommeDesDiviseurs (n : NaturelNonNul) : NaturelNonNul
    Déclaration  diviseur, somme : NaturelNonNul
debut
    somme ← 1
    pour diviseur ←2 à n div 2 faire
        si estUnDiviseur(n,diviseur) alors
            somme ← somme+diviseur
        finsi
    finpour
    retourner somme
fin
```

$O(n)$ et $\Omega(n)$

Exemple de calcul complexité 3 / 5

estParfait, $T(n) =$

```
fonction estParfait (n : NaturelNonNul) : Booleen
debut
    retourner n=sommeDesDiviseurs(n)
fin
```

$O(n)$ et $\Omega(n)$

obtenirBorneMax, $T(n) =$

```
fonction obtenirBorneMax () : NaturelNonNul
    Déclaration  resultat : NaturelNonNul
debut
    ecrire("Valeur maximale d'affichage des nombres parfaits")
    lire(resultat)
    retourner resultat
fin
```

$O(1)$ et $\Omega(1)$

Architecture des Systèmes d'Information

Exemple de calcul complexité 4 / 5

afficherNombresParfaits, $T(n) =$

```
procédure afficherNombresParfaits ()
    Déclaration  i,max : NaturelNonNul
debut
    max ← obtenirBorneMax()
    pour i ←1 à max faire
        si estParfait(i) alors
            ecrire(i)
        finsi
    finpour
fin
```

$O(n^2)$ et $\Omega(n^2)$

Exemple de calcul complexité 5 / 5

Sur l'exemple précédent, $T(n) =$

- `estUnDiviseur` : $O(1)$,$\Omega(1)$ et $\theta(1)$
- `sommeDesDiviseurs` : $O(n)$,$\Omega(n)$ et $\theta(n)$
- `estParfait` : $O(n)$,$\Omega(n)$ et $\theta(n)$
- `obtenirBorneMax` : $O(1)$,$\Omega(1)$ et $\theta(1)$
- `afficherNombresParfaits` : $O(n^2)$,$\Omega(n^2)$ et $\theta(n^2)$

Conclusion

Important

Avant de se mettre devant la machine, **on doit écrire les algorithmes ! ! !**

Méthode

1. Analyser le problème
2. Définir un énoncé
3. Faire une analyse descendante
4. À chaque niveau de l'analyse définir les signatures des procédures et/ou fonctions
5. Écrire les algorithmes de chaque procédure et fonction de niveau n de l'analyse en considérant que l'on possède de nouvelles instructions qui résolvent les problèmes du niveau $n-1$