Conception d'un programme structuré

I2 - Algorithmique et Programmation structurée

Julien Saunier, Alexandre Pauchet, Pierrick Tranouez

I2 - Algorithmique et programmation structurée

conditionnelles

variables

De la programmation impérative (I1) ...

types de données

boucles

analyse descendante

procédures

... à la programmation structurée (I2)

fonctions

entrées/sorties

fichiers

\+ des compléments de programmation (tableaux, enregistrements, . . .)

Déroulement du cours

- 8 CM
- 16 TD/TP (1 par semaine + 2 TPs supplémentaires en fin d'année)
- Soutien : 10 jeudi après midi (voir l'edt, début fin février)
- Évaluation
  - 30 % IS, 40 % DS
  - 30 % Contrôle continu : QCMs surprises, 1 TP noté

Plan

Méthodes de développement

Méthodologie générale

La création d'un programme informatique suit 4 phases :

1. La spécification des besoins
   - Analyse et reformulation du problème posé
   - Production : **Cahier des charges**
2. La conception globale
   - Définition de l'architecture globale du programme
   - Production : **Analyse descendante**
3. La conception détaillée
   - Écriture des algorithmes de chaque sous-partie du programme
   - Production : **Algorithmes en pseudo-code**
4. L'implémentation
   - Transcription dans un langage de programmation
   - Production : **Programmes**

Spécification des besoins

- Formaliser l'ensemble des fonctionnalités attendues :
- *Une spécification est un ensemble explicite d'exigences à satisfaire par un matériau, produit ou service.*

- Non Ambigu
- Accord entre le client et le fournisseur
- Contient une description fonctionnelle et/ou technique des exigences

Conception globale

Formalisation

- 1^{ere} étape de traduction de l'énoncé vers un ensemble de sous-problèmes

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 ≫

Conception détaillée

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)

Pseudo-code

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

- formel, lisible et concis,
- indépendant des langages informatiques,
- qui reprend les concepts de la programmation structurée.

Implémentation

Définition

Écriture dans un langage de programmation

- Respect du paradigme choisi lors de la conception globale
- Le choix du langage dépend de l'objectif. Par exemple
  - C, C++ : proximité du système, rapidité d'exécution
  - Java, Python : développement web, portabilité, science des données
  - Lisp, Prolog : programmation logique, Intelligence Artificielle
  - ADA : temps réel
  - Pascal : pédagogie
- Si la conception détaillée est bien faite, l'implémentation est une "simple" traduction

Cycles de développement 1 / 5

Définition

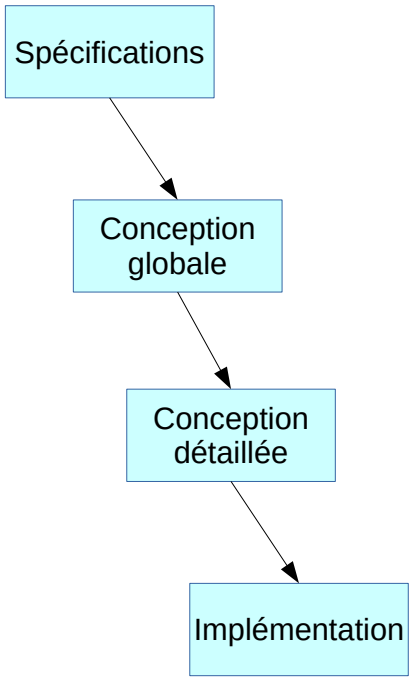
Un cycle de développement logiciel structure le déroulement des phases de développement.

- Méthodologie (décrivant le processus et non la production)
- Différentes méthodologies existent :
  - Modèle en cascade
  - Cycle en V
  - Cycle en spirale
  - Méthodes agiles
  - ...

Cycles de développement 2 / 5

Modèle en cascade

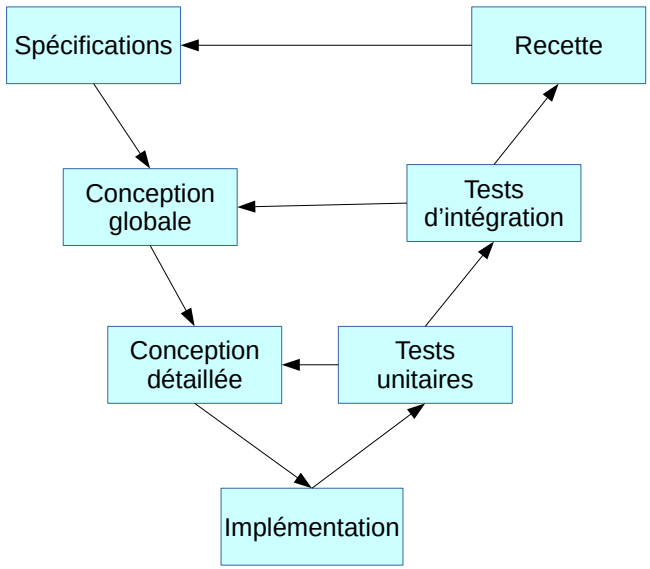
Les phases traditionnelles sont réalisées simplement les unes après les autres.



Cycles de développement 3 / 5

Cycle en V

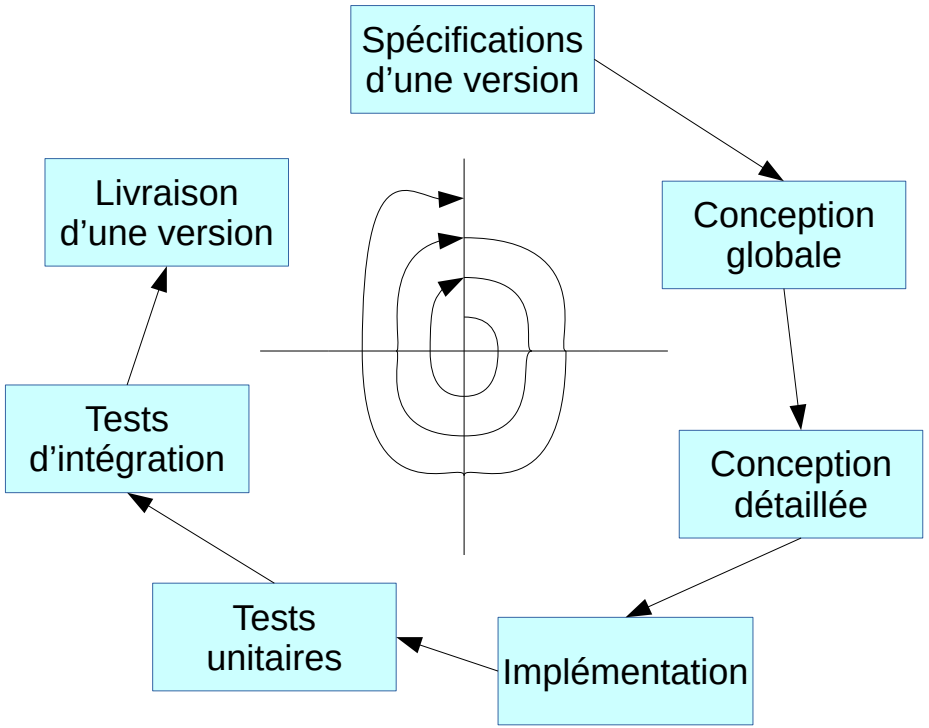
Le développement se réalise en cascade mais est suivi de tests unitaires, d'intégration et de validation, permettant un retour vers les phases de conception et d'analyse.



Cycles de développement 4 / 5

Cycle en spirale

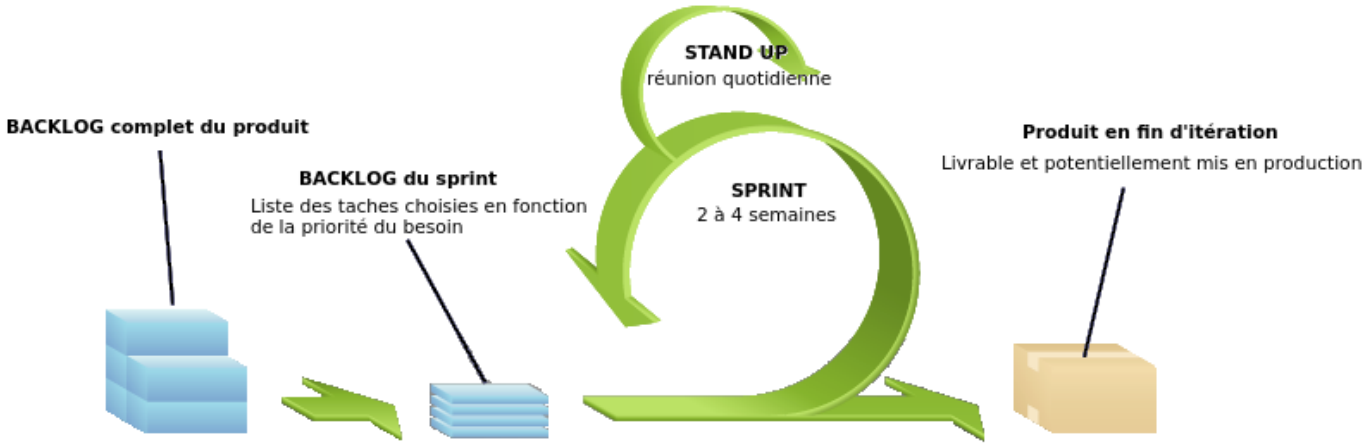
Le développement se réalise selon un cycle en cascade ou en V mais par le développement de plusieurs versions successives.



Cycles de développement 5 / 5

Méthodes "agiles" (ex : SCRUM)

Découpage d'un projet en boîtes de temps, nommées « sprints » (durée : entre quelques heures et un mois).



Une itération selon la méthode Scrum

Conception globale

Rappel : Méthodologie générale de développement

La création d'un programme informatique suit 4 phases :

1. La spécification des besoins
   - Analyse et reformulation du problème posé
   - Production : **Cahier des charges**
2. **La conception globale**
   - Définition de l'architecture globale du programme
   - Production : **Analyse descendante**
3. La conception détaillée
   - Écriture des algorithmes de chaque sous-partie du programme
   - Production : **Algorithmes en pseudo-code**
4. L'implémentation
   - Transcription dans un langage de programmation
   - Production : **Programmes**

Conception globale

Formalisation

- Traduction de l'énoncé dans un langage non ambigu
  - Méthode : Analyse descendante
  - Langage : Graphique (simplification et modification de SADT/IDEF0)

Analyse descendante

- Abstraire
- Décomposer
- Combiner

SADT 1 / 3

Structured Analysis and Design Technic

Méthode d'analyse développée par SoftTech [a] pour la description graphique d'un système complexe.
Utilisée pour la description de systèmes automatisés de différentes natures

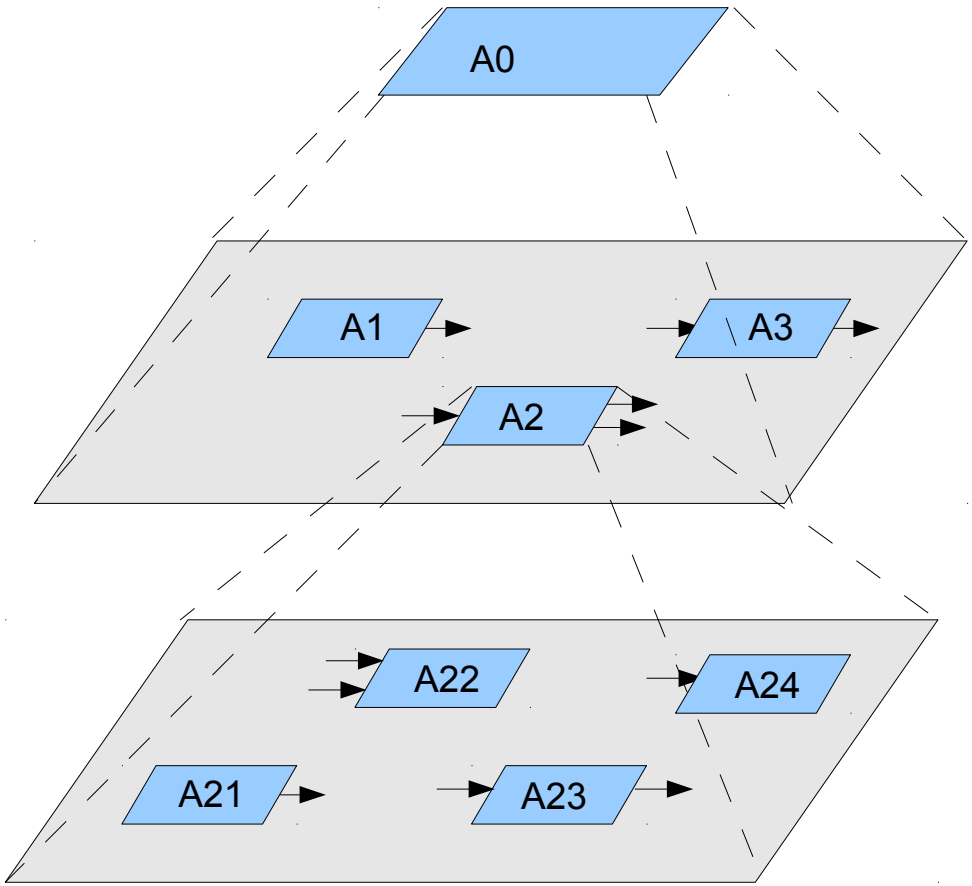
- en automatique
- pour des chaînes de production
- en télécommunications

a. États-Unis, 1977

... en informatique

Représentation d'un système d'information par les flux d'informations entre les composantes du système.

SADT 2 / 3

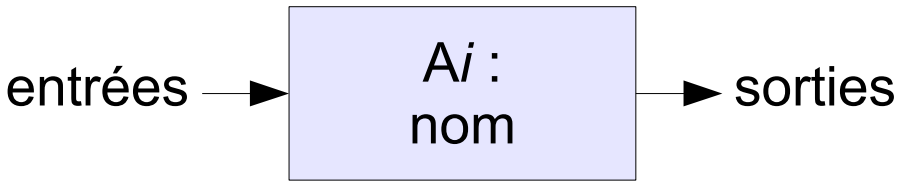


SADT 3 / 3

Diagramme d'analyse fonctionnelle descendante

Chaque boîte représente un sous-programme défini par

- son nom
  / !\ explicite : décrit ce que fait le programme
- ses entrées et sorties



Rappel : gestion des Entrées/Sorties

Entrées/Sorties

Pour traiter des informations, un programme a besoin de prendre des données en **entrée** et d'en fournir en **sortie**.
3 modes d'entrées/sorties sont possibles :

- En "dur" dans le programme
  $\longrightarrow$ *ne fonctionne que pour ces valeurs précises*
- Lecture et écriture (terminal/écran ; fichier(s))
  $\longrightarrow$ *nécessite des interactions utilisateur et/ou fichiers*
- **Formelle**
  $\longrightarrow$ générique mais devant être combiné avec d'autres programmes fournissant les entrées ou exploitant les sorties

Paramètres d'entrée/sortie 1 / 2

Rôles

Paramètres d'entrée :

- valeurs nécessaires à l'exécution d'un algorithme

Paramètres de sortie :

- servent à stocker les valeurs produites par un algorithme

Paramètres d'entrée/sortie :

- Valeur initiale nécessaire au fonctionnement de l'algorithme
- et potentiellement modifiée par l'algorithme

Paramètres d'entrée/sortie 2 / 2

Exemples

Quelles sont les entrées, sorties, entrées/sorties d'

- un algorithme calculant une valeur absolue
- un algorithme vérifiant si un nombre est premier
- un algorithme calculant le coup suivant d'une partie d'échecs

Exemple : Le mot le plus long 1 / 1

Énoncé

Concevoir un jeu du mot le plus long à deux joueurs et un arbitre.

Problèmes identifiés

- Comment tirer 9 lettres au hasard ?
- Comment tirer 1 lettre au hasard ?
- Comment vérifier que les mots soient corrects ?
- Comment mettre à jour les scores des joueurs ?

Programmation structurée

Structure d'un programme

- un programme est composé de **sous-programmes**
- l'exécution débute par un sous-programme particulier appelé le **programme principal**
- l'exécution se déroule par des **appels** successifs des sous-programmes
- les sous-programmes interagissent par le passage de données en entrée et en sortie

2 types de sous-programmes

- les fonctions
- les procédures (*cf. prochain cours*)

Lignes directrices

Comment décomposer un programme ?

- Décomposer assez... mais pas trop
- Critères :
  - longueur du code
  - réutilisabilité potentielle
  - appréhension de la solution
- Carottes $\neq$ Chaussettes
  - Ne pas mélanger des éléments fonctionnellement différents
    - Saisie et calcul
    - calcul et affichage
    - ...
- Elements SADT :
  - Caractérisation : nom, E/S
  - Hiérarchie = appel = besoin fonctionnel

Limites

Problèmes

- Division exacte des tâches inconnue
  - Necessité de noms et E/S explicites, e.g :
    - LettreauHasard() : Char
    - lettresAuHasard() : String
  - Noms de variable également
  - Rôle de la documentation
    - Commentaires
    - Documentation (cf. cours ultérieur)
- Nécessite de construire la résolution de tâche en parallèle
  - Niveau 1 : en général division par étapes
  - Niveaux 2 et inférieurs : liés à la résolution pratique (*i.e.* algorithmique du sous-problème)
    - Mutualisable entre sous-problèmes

Un nouvel exemple : les nombres parfaits 1 / 2

Problème

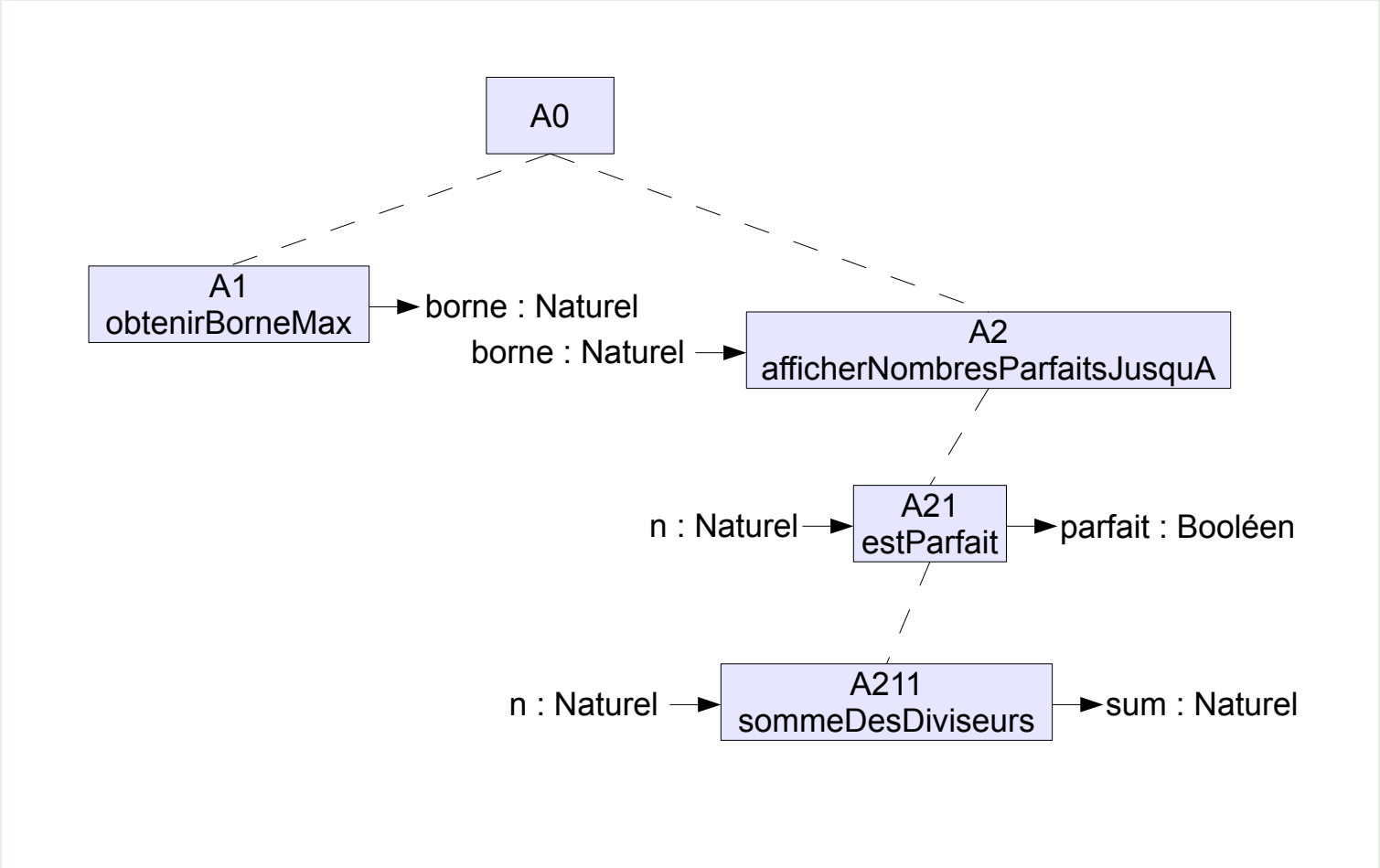
Afficher les nombres parfaits compris entre 1 et un nombre saisi par l'utilisateur.

Énoncé

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

Un nouvel exemple : les nombres parfaits 2 / 2

Analyse descendante



Conception détaillée

Rappel : Méthodologie générale de développement

La création d'un programme informatique suit 4 phases :

1. La spécification des besoins
   - Analyse et reformulation du problème posé
   - Production : **Cahier des charges**
2. La conception globale
   - Définition de l'architecture globale du programme
   - Production : **Analyse descendante**
3. **La conception détaillée**
   - Écriture des algorithmes de chaque sous-partie du programme
   - Production : **Algorithmes en pseudo-code**
4. L'implémentation
   - Transcription dans un langage de programmation
   - Production : **Programme(s)**

Pseudo-code pour la conception détaillée

Objectif(s)

- Abstraction du langage de programmation
- Focalisation sur l'algorithmique
- Formalisation non ambiguë

Formalisme

- type de données et déclaration de variables
- opérateurs
- instructions conditionnelles
- instructions itératives
- définition des entrées/sorties

Déclarations

Structure générale

Le terme **Déclaration** introduit les variables locales de l'algorithme en définissant leur identifiant et leur type de donnée.
Il apparaît avant les termes **debut** et **fin** qui encadrent les instructions de l'algorithme.

Champ déclaration

Plusieurs variables peuvent être définies dans la partie déclaration.
Si elles sont de mêmes types les identifiants des variables sont séparées par une virgule.
Le type de donnée est indiqué après le symbole :
Le point-virgule après un type de donnée permet de déclarer à la suite des variables d'un autre type.

Types de données

Rappel

Un type caractérise :

- l'ensemble des valeurs que peut prendre une variable
- l'ensemble des opérations que l'on peut effectuer sur une variable

Types en pseudo-code

- Types textuels : **Caractere**, **Chaine de caracteres**
- Types personnalisés : **Intervalle**, **Enumération**
- Types structurés : **Ensemble**, **Tableau**
- Types numériques : **Naturel**, **Entier**, **Reel**
- Autre type : **Booleen**

Types personnalisés / structurés

Définition de types

Les types personnalisés et structurés sont à définir par un type spécifique dans le champ de déclaration avec le terme **Type**.

Exemples :

```
Type Jour = {Lundi, Mardi, Mercredi, Jeudi, Vendredi,
Samedi, Dimanche}
Type Mois = 1..12
```

Opérateurs

Affectation

L'instruction d'affectation attribue une valeur à une variable. Sa syntaxe est :

`<variable>` ← `<valeur>`

Opérateurs de calcul

- non, et, ou, ouExclusif pour les booléens
- +, -, *, / pour les naturels, entiers, réels
- div, mod pour les naturels et entiers
- + pour les chaînes de caractères (et caractères par transtypage)
- $=$, $\neq$, $>$, $<$, $\leq$, $\geq$ pour les comparaisons

Exemple d'algorithme en pseudo-code

Exemple : Différence entre dates

Déclaration:
Type Mois = {Janvier, Fevrier, Mars, Avril, Mai, Juin, Juillet, Aout, Septembre, Octobre, Novembre, Decembre} ;
mois1, mois2 : **Mois** ; an1, an2, difference : **Entier**
debut
    mois1 ← Decembre
    an1 ← 2000
    mois2 ← Septembre
    an2 ← 2011
    difference ← an2 - an1
    difference ← difference * 12
    difference ← difference + ((ord mois2) - (ord mois1))
    ecrire('Il y a ', difference,' mois entre les 2 dates')
fin

L'instruction si ... alors ... sinon 1 / 2

si ... alors ... sinon ...

L'instruction `si alors sinon` permet d'associer des instructions à une condition vraie et d'autres instructions si la condition est fausse.
Sa syntaxe est :

si <*expression booléenne*> **alors**
    <*bloc d'instructions exécutées si l'expression est vrai*>
sinon
    <*bloc d'instructions exécutées si l'expression est fausse*>
finsi

L'instruction si ... alors ... sinon 2 / 2

abs

Déclaration: unEntier, laValeurAbsolue : **Entier**
debut
    lire(unEntier)
    si unEntier $\geq$ 0 **alors**
        laValeurAbsolue $\leftarrow$ unEntier
    sinon
        laValeurAbsolue $\leftarrow$ -unEntier
    finsi
    ecrire(laValeurAbsolue)
fin

L'instruction cas où 1 / 2

cas où ... vaut ...

L'instruction cas où est une instruction conditionnelle portant sur plusieurs valeurs.
Sa syntaxe est :

cas où v **vaut**
$v1$: $action_1$
$v2_1, v2_2, \ldots, v2_m$: $action_2$
...
vn : $action_n$
autre : *action*
fincas

- vi sont des **constantes** de type **scalaire** (entier, naturel, enuméré, ou caractère)
- $action_i$ est exécutée si v = vi (on quitte ensuite l'instruction cas)
- *action* est exécutée si $\forall$ i, v $\neq$ vi

L'instruction cas où 2 / 2

moisA30Jours ?

Déclaration: mois : **Entier** ; resultat : **Booleen**
debut
    lire(mois)
    cas où mois **vaut**
        4,6,9,11 : resultat ← Vrai
        autre : resultat ← Faux
    fincas
    ecrire(resultat)
fin

L'instruction itérative déterministe

Instruction pour

- Il existe une seule instruction permettant de faire des boucles déterministes, c'est l'instruction `pour`
- Sa syntaxe est :

  pour *identifiant d'une variable de type scalaire* ←*valeur de début* **à** *valeur de fin*
  faire
  instructions à exécuter à chaque boucle
  finpour

- dans ce cas la variable utilisée prend successivement les valeurs comprises entre valeur de début et valeur de fin

Pas de l'itération déterministe

Instruction pour

- le **pas** de l'instruction donne la valeur ajoutée au compteur après chaque itération
- cette valeur peut être négative, dans ce cas valeur de début $>$ valeur de fin
- Syntaxe :

pour *identifiant d'une variable de type scalaire* $\leftarrow$*valeur de début* **à** *valeur maximale* **pas de** *valeur du pas* **faire**
    instructions à exécuter à chaque boucle
finpour

Exemple

nombre premier

Déclaration: i, n : **Naturel** ; premier : **Booleen**
debut
    lire(n)
    premier $\leftarrow$ Vrai
    si (n $\neq$ 2) et (n mod 2) = 0 **alors**
        premier $\leftarrow$ Faux
    sinon
        pour i $\leftarrow$3 **à** $\sqrt{n}$ **pas de** 2 **faire**
            si (n mod i) = 0 **alors**
                premier $\leftarrow$ Faux
            finsi
        finpour
    finsi
fin

Les instructions itératives indéterministes 1 / 2

- Il existe deux instructions permettant de faire des boucles indéterministes :
  - L'instruction **tant que** :
    tant que *expression booléenne* **faire**
    instructions
    fintantque
    - qui signifie que tant que l'expression booléenne est vraie on exécute les instructions
  - L'instruction **répéter jusqu'à ce que** :
    repeter
    instructions
    jusqu'a ce que *expression booléenne*
    - qui signifie que les instructions sont exécutées jusqu'à ce que l'expression booléenne soit vraie

Les instructions itératives indéterministes 2 / 2

Calculer le pgcd(a,b), avec a et b Naturel non nul, à l'aide de l'algorithme d'euclide

Déclaration: a,b,c,reste,pgcd :**Naturel**
debut
    lire(a,b)
    si (b > a) **alors**
        c ← a
        a ← b
        b ← c
    finsi
    repeter
        reste ← a mod b
        a ← b
        b ← reste
    jusqu'a ce que reste=0
    pgcd ← a
fin

Les fonctions

On déclare une fonction de la façon suivante :
fonction *nom de la fonction* **(***paramètre(s) d'entrée de la fonction***) :** *types des valeurs retournées*

Déclaration *variable(s) locale(s)*

debut

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

retourner *valeur*

fin

Modifier l'exemple précédent pour que le calcul du pgcd soit une fonction.

Solution

Calculer le pgcd(a,b), avec a et b Naturel non nul, à l'aide de l'algorithme d'euclide

fonction pgcd (a, b : **Naturel) : Naturel**

Déclaration c,reste :**Naturel**

debut

si (b > a) **alors**

c ← a

a ← b

b ← c

finsi

repeter

reste ← a mod b

a ← b

b ← reste

jusqu'a ce que reste=0

retourner a

fin