Licence Professionnelle
RTIC en PMO

Université Toulouse I

Programmation

Support de cours

C. Hanachi

PLAN[1]

Références.

I Concepts algorithmiques de base.

II Programmation événementielle avec Visual Basic.

III Compléments sur les structures de données : description et manipulation.

IV Réalisation et qualité d'un programme.

Annexes : définitions, diagrammes syntaxiques et exercices.

[1] Ce tome ne contient que les parties I et III ainsi quelques annexes. Pour des raisons pratiques, les autres parties seront distribuées séparément.

Références

• Initiation à l'informatique, Charles Henri-Pierre, Editions Eyrolles, ISBN 2-212-09049-8, 1999.

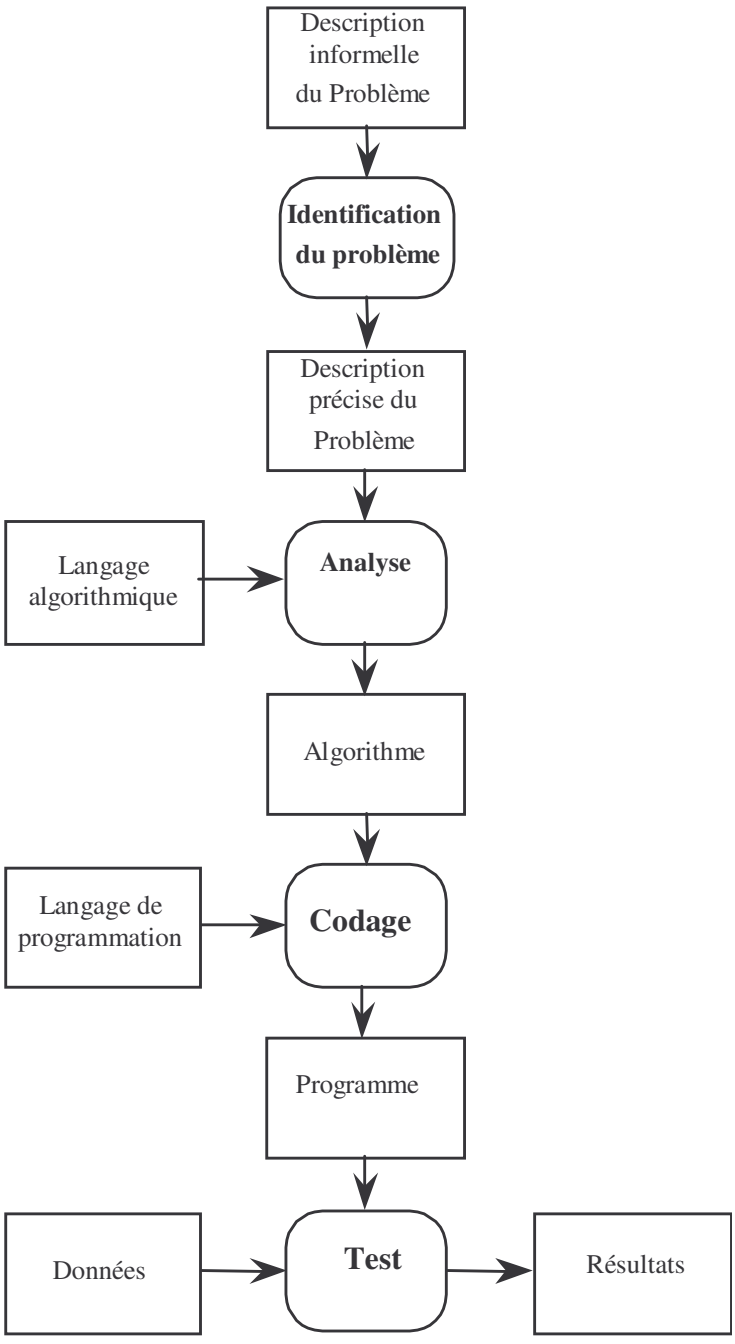
• http://eric.univ-lyon2.fr/~jdarmont/enseignement/algo-prog-deug.html.

• http://helios.univ-reims.fr/UFR/IUT/MP/visualbasic/TdmVB1A.htm

Concepts algorithmiques de base

- Etapes de conception d'un algorithme
- Identification/Spécification d'un problème
- Analyse descendante d'un problème
- Algorithme et programme
- Le Langage Algorithmique
  - Structure d'un Algorithme
  - Quelques structures de données élémentaires
  - Les actions élémentaires
  - Les structures de contrôle
  - Procédures et fonctions
  - Compléments sur les structures de données

Etapes de conception d'un programme



L'identification d’un problème

Objectif : passer d'une description non formelle du problème à traiter, à une description plus précise qui comprend :

• une **reformulation** plus explicite et plus précise du problème comprenant *le choix d'hypothèses* de travail pour lever les éventuelles ambiguïtés de l'énoncé, et la liste *des services* que doit offrir l’application envisagée ;

• la liste **des données** nécessaires à la résolution du problème et la réponse aux questions suivantes : Qui doit les fournir ? Faut-il les contrôler pour vérifier qu'elles sont conformes à ce qui avait été prévu ? Quel type de contrôle faut-il faire ?

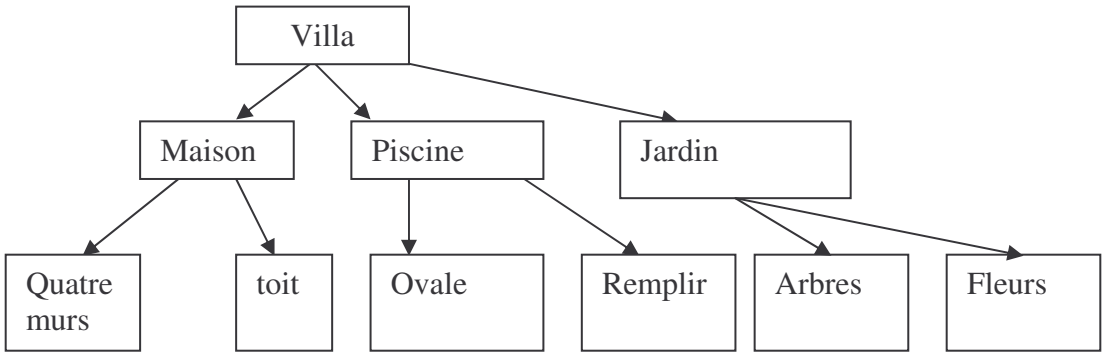
• la liste **des résultats** à fournir, ce qui suppose la réponse à la question suivante : Quelles sont les résultats pertinents à communiquer à l'utilisateur ?

• **l'interface** avec l'utilisateur du programme (ensemble des informations visibles à l'écran) ce qui suppose la réponse aux questions suivantes : Quelles sont les informations que doit fournir l'utilisateur ? Sous quelle forme ? Quelle sont les informations qu'on doit lui communiquer ? Sous quelle forme ? Quelle est l'évolution de l'interface ?

Analyse descendante d'un problème

Définition : « **L'analyse descendante** d'un problème est une démarche systématique qui part d'une expression assez générale du problème à résoudre et le *décompose en taches* plus simples. Chaque tâche peut faire à son tour l'objet d'une telle décomposition. Ce travail d'affinage peut être répété jusqu'à ce que tout ait été exprimé en terme d'actions assez élémentaires pour être traduit directement dans un langage de programmation. » [Meyer].

Exemple : dessiner une villa peut être envisagé progressivement comme le montre ce dessin :



Cette analyse aboutit à la proposition d'une solution sous forme d'algorithmes.

Algorithme

- **Définition *:*** *Un algorithme* est la description d'une action complexe au moyen d'actions élémentaires et de règles de composition de ces actions.

- **Définition** : *Une action* est une opération qui produit un effet prévu en un temps fini.

- **Exemple** : Une recette de cuisine est un algorithme où les actions élémentaires sont des opérations que le cuisinier est censé savoir exécuter ("éplucher les carottes", "mettre le plat au four", etc) et les règles de composition servent à combiner entre elles ces actions élémentaires (exécuter telle action après telle autre, attendre un certain temps avant telle action, répéter telle action un certain temps, etc).

- **Remarque :** En général, il existe plusieurs algorithmes pour résoudre un même problème. Nous privilégierons ici les solutions simples, facile à comprendre.

- Un algorithme destiné à devenir *un programme*, c'est à dire à être traité par un ordinateur (~~et non pas un cuisinier~~), doit être formulé dans *un langage précis*, i.e défini par : un lexique, une syntaxe et une sémantique.

- **Définition** : *Un programme* est un algorithme exprimé dans un *langage de programmation* i.e. compréhensible par une machine.

Le Langage algorithmique

Un langage algorithmique se définit en précisant :

• le type d'objets que peut manipuler l'algorithme *: les structures de données*.

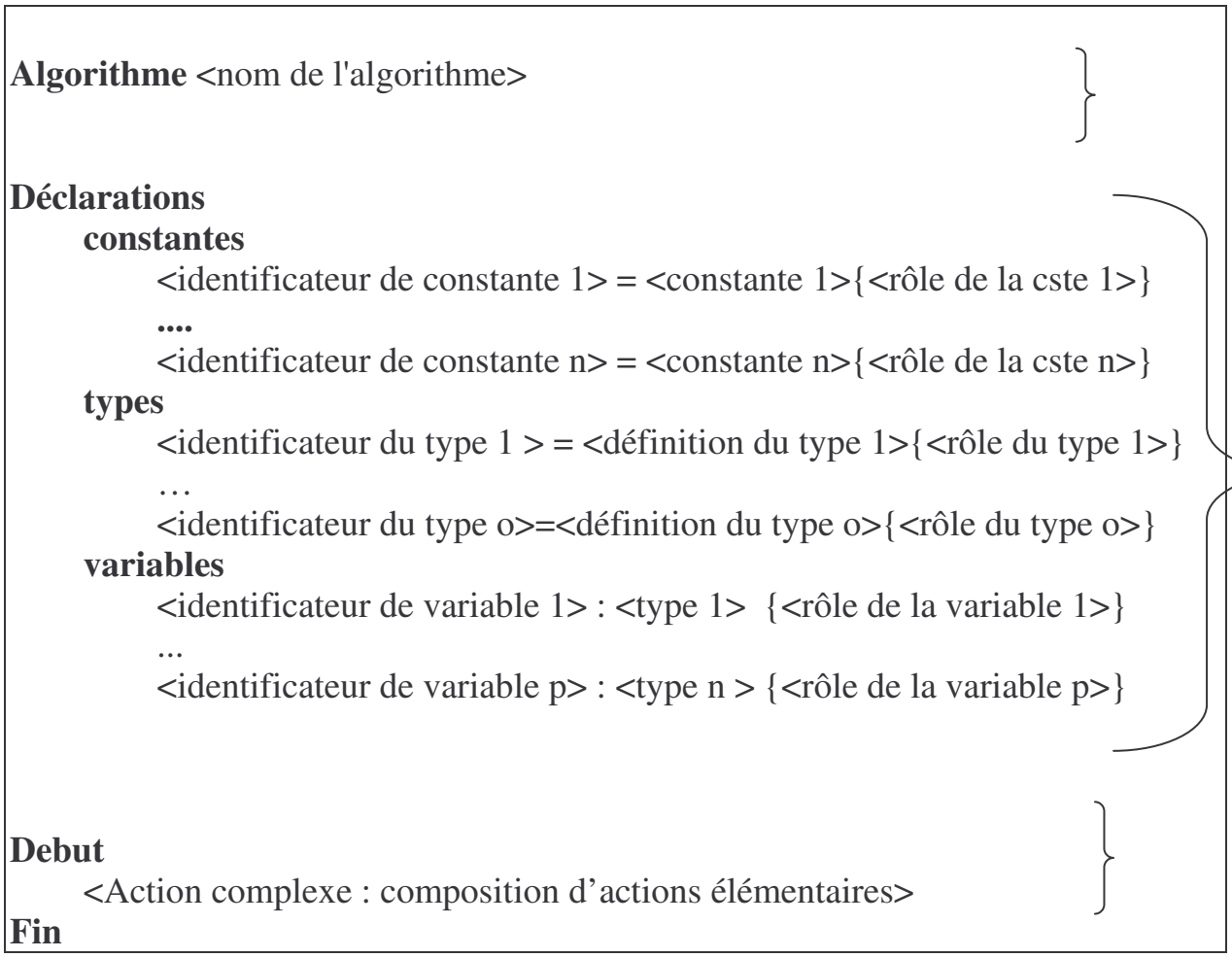
• la manière de décrire ces objets : *déclarations*.

• l'ensemble des *actions élémentaires* en indiquant leur syntaxe et leur sémantique.

• l'ensemble *des règles de composition* (ou *structure de contrôle*) en indiquant leur syntaxe et leur sémantique.

Structure d'un algorithme (I)

Structure d'un algorithme



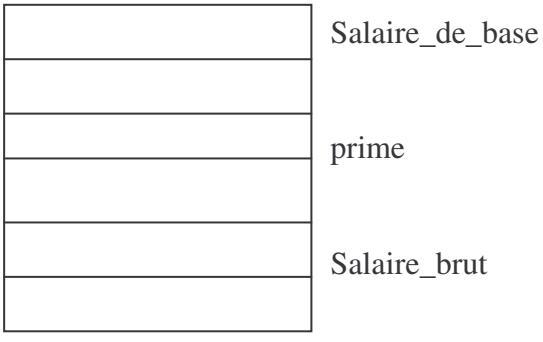
Structure d'un algorithme (II)

```
Algorithme calcul_salaire
{ cet algorithme calcule le salaire brut à partir du salaire de base et de la prime}
Déclarations
    variables
        salaire_de_base : reel
        prime : reel
        salaire_brut : reel
Debut
    {Préparation du traitement : saisie des données}
    écrire ("donnez le salaire de base")
    lire (salaire_de_base)
    écrire ("donnez la prime")
    lire (prime)

    {calcul du salaire brut}
    salaire_brut<-salaire_de_base + prime

    {affichage des résultats}
    écrire ("Le salaire de base étant de", salaire_de_base, "et la prime de", prime, "le salaire brut est donc de ", salaire_brut)
Fin
```

Mémoire



Quelques structures de données élémentaires

- Etapes de conception d'un algorithme
- Identification/Spécification d'un problème
- Analyse descendante d'un problème
- Algorithme et Programme
- Le Langage Algorithmique
    - Structure d'un algorithme
    - **Quelques structures de données élémentaires**
    - Les actions élémentaires
    - Les structures de contrôle
    - Procédures et fonctions

Quelques structures de données élémentaires (I)

Les types

Un type est défini par :
1) un ensemble de valeur, et
2) un ensemble d'opérations applicables aux éléments de cet ensemble.

Exemple :
Le type entier sera défini par :

- un sous-ensemble fini de Z
- et notamment les opérations suivantes : addition, soustraction, division, reste, etc.

Les constantes

Une constante est un objet non modifiable par l'algorithme. Une constante est définie lorsqu'on a précisé *son nom* et *sa valeur* constante.

Exemple : Au lieu de manipuler dans un algorithme la valeur 0.055 correspondant à un taux TVA, il est plus clair d'employer un identifiant pour le désigner, par exemple CodeTva. Cela va permettre :

1) de modifier plus facilement l'algorithme
2) d'indiquer le rôle de cette valeur puisque l'identifiant est plus explicite.

Les variables

Une variable est une objet modifiable par l'algorithme. Elle est définie lorsqu'on a précisé : *son nom(ou identificateur), son type, son rôle*. Par définition, une variable peut prendre plusieurs valeurs au cours de l'exécution d'un algorithme. Mais à un instant donné elle ne peut avoir qu'une valeur.

Quelques structures de données élémentaires (II)

Identificateur : Il faut s'efforcer de choisir des identificateurs explicites c'est à dire en rapport avec ce qu'il représente (on préférera "age" à X si on doit représenter l'age d'un individu). Syntaxiquement, un identificateur peut être décrit par une suite non vide de caractères respectant les conditions suivantes :

1) il ne doit pas contenir de caractères délimiteurs : espaces, parenthèses, guillemets, ...

2) il ne doit pas appartenir au mot réservé du langage ;

3) il doit commencer par une lettre.

Remarques :

• En cours d'exécution, une variable ne peut recevoir que les valeurs associées à son type.

• Le rôle d'une variable est un commentaire facultatif (mais que nous conseillons) qui explicite l'utilisation qui est faite de la variable.

Quelques structures de données élémentaires (III)

- **La partie déclaration d'un algorithme**

Tout algorithme commence par la définition des objets qu'il manipule. Comme dans une recette on met toujours les ingrédients nécessaires en tête de la recette : 1 livre de farine, 1kg de belles tomates, ...

Dans un algorithme la déclaration se fait selon la syntaxe suivante :

Déclarations

constante

<identificateur de constante 1> = <constante 1>{<rôle de la constante 1>}
....
<identificateur de constante n> = <constante n>{<rôle de la constante n>}

types

<identificateur du type 1 > = <définition du type 1>{<rôle du type 1>}
…
<identificateur du type o>=<définition du type o>{<rôle du type o>}

variable

<identificateur de variable 1> : <type 1> {<rôle de la variable 1>}
...
<identificateur de variable p> : <type n > {<rôle de la variable p>}

Exemple :

Déclarations

constante

Taux_Tva = 0.055 {représente le taux tva utilisé dans le contexte du problème posé}

variable

Quantité : entier {indique le nombre d'articles commandés}
Prix_HT: réel {indique le prix unitaire hors taxes de chaque article}

Quelques structures de données élémentaires (IV)

- **Les types prédéfinis élémentaires**

Ce sont des types que l'on retrouve dans la plupart des langages. La machine sait les représenter et offre des fonctions pour les manipuler.

Entier : C'est un sous-ensemble fini de Z (par exemple de [-32768, 32767]). Les opérations possibles sur les entiers sont définies par :
+, -, *, div(division entière), ^(puissance), mod(RESTE), abs(valeur absolue), sqr(carré) …

Réel : ensemble des valeurs numériques non forcément entières que la machine sait représenter. Les opérations possibles sur les réels sont définies par : +, -, *, /, ^, abs (valeur absolue), int (partie entière), sqr (carré), sqrt (racine carrée).

Chaîne de Caractères : ensemble des caractères que la machine reconnaît (jeu de caractères fait de lettres, de chiffre, ...). On convient de distinguer les valeurs de type caractère en les délimitant par des guillemets : "lulu", "12 rue des lois", etc. Nous verrons plus tard les fonctions qui s'appliquent sur les variables de ce type.

Booléen : ensemble des valeurs "vrai" et "faux." Les opérations possibles sur les booléens sont définies par : et, ou, non

Il existe également des types composés à partir de ces types élémentaires. Nous les introduirons plus tard.

Les actions élémentaires (I)

- **L'affectation**

L'affectation permet de donner une valeur à une variable

Syntaxe : **<variable> <- <expression>**
Sémantique:
1) calcule la valeur de <expression>
2) donne cette valeur à <variable>
Une expression peut être une constante, une variable ou une opération faisant intervenir constantes, variables, opérateurs, et/ou fonctions. La valeur de l'expression doit être du même type que la variable.

Exemples :
nom <- "dupont"
salaire_de_base <- 12000
prime <- 2500
salaire_brut <- salaire_de_base + prime
(Si ces trois premières actions sont exécutées successivement salaire_brut vaudra 14500)
C<- 20
C<- C+1 (incrémentation de C)
C<- C-1 (décrémentation de C)
incorporable<- (age >18) et (sexe ='m')
(incorporable est un booléen)

L'affectation est une opération qui se fait en mémoire : aucun écho de cette opération n'est visible à l'écran.

Les actions élémentaires (II)

Une expression est évaluée de gauche à droite mais en **tenant compte de priorités**. Les opérateurs possibles sont indiqués ci-dessous, de la priorité la plus haute à la priorité la plus faible :

- Les opérations unaires (n'impliquant qu'un opérande): + , - , NOT
- ensuite: * , / , DIV , MOD , AND
- ensuite: + , - , OR
- et enfin: = , <> , < , > , <= , >=

Les parenthèses peuvent être utilisées pour lever les ambiguïtés.
Il y a deux types d'expression :

• les **expressions arithmétiques** dont le résultat de l'évaluation est un nombre.
Exemple : 4 * (2 + X), 4*2+X

• les **expressions logiques** (ou booléennes) dont le résultat de l'évaluation est "vrai" ou "faux" c'est à dire un booléen.
Exemple : (age >= 16) et (age <= 65) et (non salarié)
ou age est de type numérique et salarié de type booléen.

Les actions élémentaires (III)

- **L'écriture**

L'écriture permet à la machine d'afficher des informations à l'écran.

syntaxe : écrire (<expression 1>, ..., <expression n>)
sémantique :
1) calcule les valeurs de <expression 1>,..., <expression n>
2) affiche à l'écran ces n valeurs les unes à la suite des autres.

L'écrire permet à la machine de s'exprimer sur l'écran.

Exemples : supposons que les variables salaire_de_base, prime, salaire_brut aient respectivement les valeurs 12000, 2500, 14500
écrire (prime)
-> affiche **2500**

écrire ("bonjour")
-> affiche **bonjour**

écrire (3)
-> affiche **3**

écrire (salaire_de_base + prime)
-> affiche **14500**

écrire ("voici le salaire brut ", salaire_brut)
-> affiche **voici le salaire brut 14500**

écrire ("Le salaire de base étant de", salaire_de_base, "et la prime de", prime, "le salaire brut est de ", salaire_brut)
->affiche **Le salaire de base étant de 12500 et la prime de 2500 le salaire net est de 14500**

Les actions élémentaires (IV)

- **La lecture**

La lecture permet à la machine d'acquérir des données communiquées par l'utilisateur.

syntaxe : lire (<variable 1>, ..., <variable n>)
sémantique :
1) affiche un point d'interrogation et
2) s'arrête et attend que n valeurs soient frappées au clavier par l'utilisateur. Les variables recevront dans l'ordre les valeurs frappées.

Exemple :
lire (nom)
-> un ? S'affiche à l'écran et attend que l'utilisateur frappe une valeur. La valeur frappée est affectée à la variable nom. Le lire permet donc à l'utilisateur de s'exprimer.

lire (A, B)
-> un ? s'affiche à l'écran et attend que l'utilisateur frappe deux valeurs.

Exercices : traiter les exercices 1 à 5 (annexes : niveau 1).

Les structures de contrôle (I)

Les règles de composition d'actions sont plus connues sous le nom de structures de contrôle.
Ces structures ne modifient pas la valeur des variables, mais permettent à la machine, lors de l'exécution du programme, d'ordonner l'exécution des instructions.
Le regroupement de plusieurs actions élémentaires a un certain effet et constitue à ce titre aussi une action que l'on appelle action complexe. Dans la suite, lorsqu'on ne précise pas la nature de l'action c'est qu'il peut s'agir indifféremment de l'une ou de l'autre.

a) La séquence

Syntaxe :
Soient A et B deux actions et considérons la séquence suivante :

A
B

Sémantique :
Cette séquence constitue une action dont l'effet est obtenu par l'exécution successive de A et de B. L'ordre d'écriture est utilisé pour exprimer un ordre temporel : exécution de A et ensuite exécution de B.

Ceci se généralise à plusieurs actions. L'exécution de la séquence A,B,C s'écrira :

A
B
C

Les structures de contrôle (II)

b) L'alternative :
Certaines actions sont à effectuer uniquement dans certaine condition.

Syntaxe

Soient une condition C (expression logique), une action A et une action B

Si C **Alors**
A
Sinon
B
Finsi

Sémantique
1) on calcule C
2) Si C est vérifiée on exécute A et dans le cas contraire on exécute l'action B.

Ceci est sémantiquement équivalent à :

Si non C **Alors**
B
Sinon
A
Finsi

Exemple :
Si x > y **alors**
max<-x
sinon
max<-y
finsi
Cette action a pour effet de donner à max le plus grand des deux nombres x et y.

Exercice : calculer le maximum de trois nombres.

Les structures de contrôle (III)

Remarque : Soit l'algorithme suivant :

```
Si   C          Alors
     A
Sinon
     B
     D
Finsi
E
```

Lorsque cette structure alternative est rencontrée, il y a exécution de manière exclusive de A ou de (B et D). La suite de l'algorithme, en l'occurrence ici E, se poursuit de manière commune. Il est donc nécessaire de marquer par "finsi" la fin de la structure alternative pour éviter toute ambiguïté.

Cas particulier:
Un cas fréquent est celui où une des deux actions A ou B est vide. La notation devient alors :

```
Si C       Alors
     A
finsi
```

ou bien

```
Si non C Alors
     B
finsi
```

Exercices : traiter les exercices 6 et 7 (annexes : niveau 1).

Les structures de contrôle (IV)

c) L'itération

Une même action doit quelques fois être répéter plusieurs fois. L'itération va permettre d'exprimer ce type de traitement. Une itération est arrêtée lorsqu'une condition est vérifiée L'itération peut prendre plusieurs forme :

Cas où le nombre d'itération est non déterminé

Syntaxe

Soient C une condition de continuation, A une action

Tant Que C

A

Fin Tant Que

Sémantique

1) calculer C
2) si C est faux, l'itération s'arrête là sinon on exécute A
3) après l'exécution de A, on reprend en 1).

Remarques :

• Toutes les fois qu'on écrit un schéma d'itération, il faut s'assurer que la condition d'arrêt sera bien vérifiée après un nombre fini d'opération. Pour cela, il faut faire en sorte que cette condition puisse évoluer.
• L'état initial de la (n+1)ième exécution de A est l'état final de sa nième exécution.
• A la fin de l'itération, la condition C a la valeur faux
• Si C est faux dés le départ, A ne sera jamais exécutée.

Les structures de contrôle (V)

Une autre forme d'itération est la suivante :

Syntaxe
Soient C une condition d'arrêt, A une action

Répéter
A
Jusqu'à C

Sémantique
1) exécuter l'action
2) après exécution de A, si la condition C est vérifiée, l'itération s'arrête là sinon on reprend en 1).

Exemple :

```
Répéter
    écrire ("voulez-vous continuer (oui/non) ?" )
    lire rep
Jusqu'à (rep="oui") ou (rep="non")
```

Remarque :
• l'exécution de A est toujours exécutée au moins une fois
• Ce schéma d'itération peut s'exprimer à l'aide du "tant que" :

```
A;
Tant Que C

    A
Fin Tant Que
```

Les structures de contrôle (VI)

Cas où le nombre d'itération est déterminé

Syntaxe
Soient v une variable de type entier, et e1 et e2 deux expressions de type entier respectant la condition suivante : e1 < e2.

Pour v **allant de** e1 **à** e2
    A
Fin Pour

Sémantique
v<-e1
Tant Que v< e2
    A
    v<-v+1
Fin Tant Que

Remarques
• la boucle Pour s'exécute (e2- e1+1) fois
• le nombre de répétition est déterminée par les valeurs de e1 et e2 au début de l'exécution de la boucle, et non pas par A.
• les valeurs de e1 et de e2 ne doivent pas être modifiées à l'intérieur de la boucle.

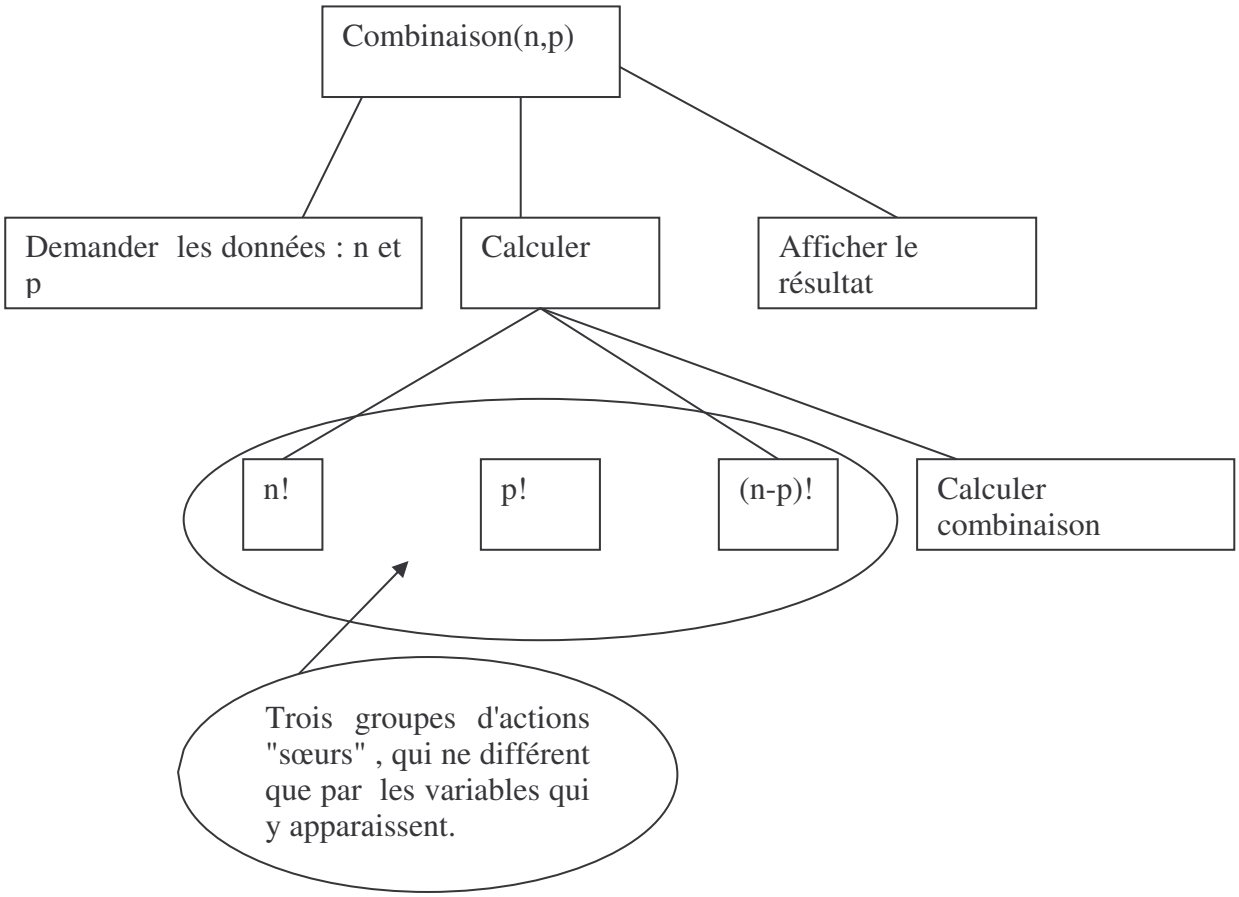
Exercices : traiter les exercices 8 à 11 (annexes : niveau 1).

Procédures et Fonctions

Exemple justifiant l'intérêt des procédures et des fonctions.

Ecrire un Algorithme qui calcule C(n,p) :le nombre de combinaisons de p objets pris parmi n (n>=p>0)

C(n,p) = (**n!**) div (**(n-p)!** * **p!**)



Définition des sous-programmes

Définition : Un sous-programme est un groupe d'actions aux caractéristiques suivantes :

- il est créé à l'initiative du programmeur,
- il réalise une tâche précise,
- il possède un nom et éventuellement des paramètres,
- une fois déclaré(défini), il peut être utilisé à volonté comme les actions prédéfinis (écrire, lire, affectation, abs,).
- Il peut prendre la forme d'une fonction ou d'une procédure.

Différence entre Procédure et fonctions :

- une fonction retourne une valeur alors qu'une procédure peut réaliser des traitements divers (calculer, afficher, affecter des valeurs à une ou plusieurs variables, etc.).
- Se définissent et s'utilisent différemment (cf. Exemple plus loin).

Structure et exécution d'un algorithme avec sous-programmes

```
ALGORITHME calcul_combinaison
{algorithme permettant de calculer C(n,p) =n!/( (n-p)!*p!) Où n et p sont
demandées à l'utilisateur}

Déclarations {déclarations globales}
        Variables
        n, p : entier {variables demandées à l'utilisateur}
        Factn, Factp, Factnmoinsp, combinaison : entier {variables recueillant les
différents résultats}

{déclarations des procédures et/ou fonctions}
Procédure factorielle (D x:entier , R f: résultat)        ⟶   entête
{procédure qui calcule la factorielle de x et qui l'affecte à f}
Déclarations
        Variable I :entier {déclarations locales}
Début                                                          ⎫
      f<-1                                                     ⎪
      Pour I allant de 1 à x                                   ⎪
                                                               ⎪  ⟶   corps de la procédure
                f<-f*I                                         ⎪
      Fin Pour                                                 ⎪
Fin procédure                                                  ⎭

DEBUT
      {lecture de n et p}
      Ecrire ("donnez les valeurs de n et p, n>=p")
      Lire(n,p)

      {calcul des différentes factorielles}                    ⎫
      Factorielle (n, factn)                                   ⎪
      Factorielle (p, factp)                                   ⎪
      Factorielle (n-p, factnmoinsp)                           ⎭
        Combinaison<-factn /( factp * factnmoinsp)

      {affichage des résulats}
      Ecrire ("le résultat est" ,  factn/(factp*factnmoinsp))
FIN
```

3 Appels à la procédure factorielle

Déclaration et appel d'une procédure

- **Déclaration :**

Procedure <NOM> (<liste des paramètres formels>)
<Déclarations locales>
Debut
    <Actions>
Fin Procedure

- **Liste des paramètres formels**

(<statut> <variable> : <type>, <statut> <variable> : <type>, …)

<statut> : D pour donnée ,
R pour résultat,
ou D/R pour Donnée/Résultat.

- **Appel d'une procédure**

<Nom> (liste des paramètres effectifs)

<Nom> (<parmètre1>, <paramètre2>, …)

Remarque : Il doit y avoir compatibilité en type et en nombre entre les paramètres effectifs et les paramètres formels. Un paramètre effectif doit avoir le même type que le paramètre formel auquel il correspond.

La notion de fonction

- **Définition** : Une fonction est un sous-programme qui retourne une valeur. Toute procédure qui comporte un seul paramètre résultat peut être remplacée par une fonction.

- **Déclaration d'une fonction**

Fonction <nom de la fonction > (liste de paramètres données) : <type de la fonction>
<déclarations locales>
Debut
Actions
<nom de la fonction > <- <expression>
Fin Fonction

- **Exemple :**

Fonction Factorielle (D x:entier) **:entier** } entête avec le type de la fonction

Variable I :entier {déclarations locales}
f:entier {résultat }

Début
f<-1
Pour I allant de 1 à x
f<-f*I
FinPour

Factorielle<-f
Fin Fonction

} corps de la fonction

- **Appel d'une fonction**
S'utilise comme une valeur :
Ecrire (fact(3))
Y<-fact(4)
~~Fact(3)~~

Avantages des sous-programmes

- Lisibilité des algorithmes ;
- réutilisation de groupes d'actions (briques de base) ;
- rend compte de l'analyse descendante qui a permis d'aboutir au programme.
- facilite la mise au point du programme.

Exercices : traiter les exercices 1 à 9 (annexes : niveau 2).

Compléments sur les structures de données

Compléments sur les structures de données : tableaux, structures

- Une variable est définie par un identifiant et un type. A chaque variable déclarée va être associé un certain nombre d'emplacements mémoire dépendant de son type.

- La richesse d'un langage se mesure notamment à son pouvoir d'expression en terme de types de données.

- Plus on a de pouvoir d'expression en terme de types et plus les traitements sont simples à exprimer.

- Types scalaires : entier, réel, booléen, caractère, …

- **Types structurés : composés d'éléments de nature :**
  - Identique : tableau, chaîne.
  - différente : structure.

Définition des tableaux

Définition

Intuition : lorsque les données sont nombreuses et de même nature, pour éviter de multiplier le nombre de variables il est plus commode de les regrouper dans une seule variable identifiée par un nom, avec un ou plusieurs index assurant l'accès individuel aux données.

Exemple : un tableau de fruits.

Fruits

Mandarine	Melon	Figue	Ananas	Kiwi	Raisin
1	2	3	4	5	6

- La taille est 6
- Chacun des fruits a un rang.
- Accès séquentiel ou direct sont possibles.
- Le 4ème fruit se note "fruits(4)" et sa valeur est "Ananas".

Définition : Un tableau est un groupe d'objets de même nature dont le nombre maximum est connu : c'est la taille du tableau.

Exemple :

- tableau de 10 entiers
- tableau de 10 caractères (chaîne de 10 caractères)
- tableau de 30 booléens.

Déclaration des tableaux

Une variable de type tableau est définie par :
- Un identificateur
- L'intervalle des indices
- Le type de ces éléments

Exemple :

Variable fruits : **tableau** (1..6) de chaîne
Variable jeu : tableau (1..10, 1..10) de booleen

Variable temperature : **tableau** (1..7) de réel
Ou mieux
Constante nbjour=7
Variable temperature : tableau (1..nbjour) de réel

L'utilisation des tableaux

- Chaque élément d'une variable T de type tableau se comporte comme une variable de nom : **T(indice de l'élément, …)**
- Les opérations possibles sur chaque élément sont toutes celles du type auquel il appartient.

Exemple :

Constante nbjour = 7
Variable temperature : **tableau** (1..nbjour) de réel
Variable I : entier
Temperature(1)= 21
Temperature(2)=temperature(1)
I=4
Temperature(I+1)=temperature(I+2)
~~Temperature(8)~~ débordement
~~Temperature(I)="A"~~ conflit de type
Temperature(I)= X valable si 1<=I<=7 et X de type réel.

Quelques algorithmes

```
{lectures des valeurs d'un tableau}
Pour I allant de 1 à nbjour
    Ecrire ("donnez la", I, "ème valeur")
    Lire (temperature(I))
Fin Pour
```

```
{Calcul de la moyenne des valeurs d'un tableau}
Somme <-0
Pour I allant de 1 à nbjour
    Somme <-somme + temperature (I)
Fin Pour
Moyenne <-somme/nbjour
```

L'utilisation des tableaux

```
{Vérifier la présence d'une valeur : zéro par exemple}
variable trouve : booléen

I=1
trouve <-faux
Tant Que (I<=nbjour)  et (trouve = faux)
        Si temperature (i)=0 Alors
                        trouve<-vrai
        Finsi
          I<-I+1
Fin Tant Que
Si trouve alors
        Ecrire ("la valeur zero est présente")
Sinon
        Ecrire("la valeur zero n'est pas présente")
Finsi
```

{**home work** : afficher le contenu d'un tableau, calculer le plus grand élément, donnez la position d'une valeur donnée dans un tableau, trier un tableau}

Le type structure

Définition

Lorsqu'une entité doit contenir différentes informations - de structure éventuellement différente- mais qui forment une entité cohérente, il est intéressant de les regrouper dans une variable de type structure.

Déclaration

```
Déclarations
    Type membre
        nom : chaine
        prenom : chaine
        datenaissance: date
        cotisation : numerique
    Fin

Variable president, tresorier, secretaire : membre
Variable conseil_administration : tableau (1 à 10) de membre
```

Utilisation

```
tresorier=president{manipulation globale}
secretaire.nom="riri" {manipulation d'un élément}
conseil_administration(1).nom="fifi"
```

Exercices : traiter l'exercice 10 (annexes : niveau 2).

ANNEXES

Exercices
Niveau I

Exercice 1 : Ces instructions sont-elles correctes ?
I+1 <-I
Ecrire "les valeurs de I et J sont I, J"
I<-R
L<- (I > J) et R
lire "bonjour", A," au revoir "
L<-(I>J) ou (R > Q)

I,J sont des entiers, R un réel, et L un booléen.

Exercice 2 : Quelles est la nature des expressions suivantes :
A+B*C
(A-18) et (B>20)
(A=18) et (B > 20)
(A-18) et (B>20)
(L > 4) et B
(A=18) et (B > 20) ou (non L)

A et B sont des entiers, L un booléen. Dites s'il s'agit d'expressions logiques, d'expressions arithmétiques. S'il s'agit d'une expression mal formée dites pourquoi.

Exercice 3 : Soient A et B deux notes obtenues par un étudiant, et C1 et C2 les coefficients de chacune de ces notes. Ecrire l'expression logique qui détermine si l'étudiant à la moyenne sur l'ensemble de ces deux notes.

Exercice 4 : Concevoir et écrire un algorithme qui calcule le prix TTC d'un lot de produit dont l'utilisateur donnera le prix hors taxe et la quantité unitaire. On fixera la TVA à 0.055.
Quelles modifications sont nécessaires pour que l'algorithme soit valable quelque soit le taux TVA du produit ?

Exercice 5 : Concevoir et écrire un algorithme qui permute les valeurs de deux variables.

Exercice 6 : concevoir et écrire un algorithme qui affiche la mention (AJOURNEE, PASSABLE, ASSEZ BIEN, BIEN, TRES BIEN) qui correspond à une note donnée.

Exercice 7 : concevoir et écrire un algorithme qui permet de résoudre une équation du second degré. On se limitera aux solutions réelles.

Exercice 8 : Concevoir et écrire un algorithme qui calcule la somme des n premiers entiers positifs. Si n vaut 4 l'algorithme doit afficher 10.

Exercice 9 : Concevoir et écrire un algorithme qui réalise la multiplication de nombre a et b sans utiliser la multiplication.

Exercice 10 : Concevoir et écrire un algorithme qui affiche la liste des diviseurs d'un nombre entier.

Exercice 11 : Une personne se pesant une fois par jour aimerait connaître à la fin de la semaine : son poids moyen sur la semaine, son plus fort et son plus faible poids de la semaine.

Exercices
Niveau 2

Exercice 1:
Les tarifs d'affranchissement d'une lettre sont les suivants :

En-dessous de 20 g	: 3 Francs.
A partir de 20 g, mais en dessous de 50g	: 6 Francs.
A partir de 50 g	: 8 Francs.

Ecrire la fonction qui délivre le tarif pour un poids donné.

Exercice 2 :
Ecrire une procédure qui permet de permuter la valeur de deux variables a et b.

Exercice 3 :
Soient n et p deux entiers. Ecrire une fonction qui détermine si p est diviseur de n.

Exercice 4 :
Ecrire une fonction booléenne qui détermine si un nombre entier positif est premier.

Exercice 5 :
Ecrire une procédure qui affiche la liste des nombres premiers inférieur à un nombre positif n.

Exercice 6 :
Ecrire une fonction qui retourne le PGCD de deux nombres entiers positifs.

Exercice 7 :
Ecrire une fonction qui retourne le nombre de combinaison de p éléments pris parmi n. On se basera sur une fonction qui calcule la factorielle.

Exercice 8 :
Réaliser une procédure qui transforme un temps T exprimé en secondes en heures, minutes et secondes.

Exercice 9 : Soit k un entier positif. k est dit "doublement premier" si k et k+2 sont premiers. Ainsi 3 est "doublement premier" car 3 et 5 (=3+2) sont premiers. 5 et 11 sont également "doublement premiers". Ecrivez un sous-programme qui affiche la liste des nombres doublement premiers inférieurs ou égaux à un nombre donné N. Si N vaut 13 votre sous-programme doit afficher : 1,3,5,11.

Exercice 10 :
Considérons un tableau de N températures correspondant aux températures moyennes mensuelles d'une ville donnée. Produire les procédures et/ou fonctions permettant de :
- calculer la moyenne des températures ;
- afficher la température la plus haute ;
- afficher la température la plus basse et le mois correspondant ;
- trier le tableau dans l'ordre croissant.

Si l'on souhaite gérer plusieurs villes, quels changements sont nécessaires ?

QUELQUES DEFINITIONS

Définition : *Une machine* (un processeur) est un dispositif capable :
1) d'exécuter des actions élémentaires(instructions) appartenant à un répertoire bien défini et caractéristiques de la machine,
2) éventuellement, d'exécuter des actions complexes par combinaison d'actions élémentaires selon les règles de composition d'actions caractéristiques de la machine.
Elle se définit par :
1) son jeu d'instructions (ce qu'elle sait faire)
2) le langage qu'il faut utiliser pour lui transmettre les ordres.

Définition : *Un programme* est un algorithme exprimé dans un langage compréhensible par une machine.