1

ALGORITHMIQUE

Michel Blache

Version 1.1 du 20 Octobre2022

2

Algorithmique

Votre Animateur

- Michel Blache
- Intervenant pour le Groupe AFIB
- Chef de Projet Infrastructure Airbus, AFI, EFS
- Technologies :
  - ITIL,
  - Linux,
  - Merise, UML,
  - DataBase & SQL
  - AMOA

Iffpa - Poitiers

3

Iffpa - Poitiers

Algorithmique

Parlez-moi de vous

- Employeur
- Fonction
- Utilisation de l'outil informatique
- Besoins
- Attentes

4

Iffpa - Poitiers

Algorithmique

Au programme

- Objectifs du stage
  - comprendre la notion d'algorithme et de programme informatique ;
  - déclarer, manipuler une variable et vous connaîtrez les différents types de données ;
  - manipuler les structures de contrôle (séquence, alternatives) et les structures itératives (boucles) ;
  - déclarer un tableau unidimensionnel ou multidimensionnel statique ou dynamique, le parcourir,
  - rechercher un élément, effectuer un tri ;
  - modulariser un algorithme en écrivant des fonctions.

5

Ifpa - Poitiers

Algorithmique
Organisation de la formation

- Horaires
- Pauses
- Fin de la formation
  - Supports de cours
  - Evaluation de la formation

6

Ifpa - Poitiers

Algorithmique
Pédagogie

- Théorie
- Démonstration du Formateur
- Mise en Application collective puis autonome
- Questions et Réponses

Algorithmique

Avez-vous des questions ?

Algorithmique

Définitions : Qu'est ce qu'un algorithme

- Un algorithme est une suite d'instructions, qui une fois exécutée, conduit à un résultat donné.
  - Si l'algorithme est juste, le résultat correspond au résultat souhaité.
  - Si l'algorithme est faux, le résultat ne correspond pas à celui qui est attendu.
- Un algorithme correspond donc à la description précise de la façon dont on résout un problème donné. Il modélise le comportement d'une solution sous la forme d'une suite d'actions.

Algorithmique

Définitions : Qu'est ce qu'un algorithme

- Comment faire une pizza ?
  - Allumer le four et le placer à 280°
  - Prendre de la pâte et l'étaler sur le support du four déjà fariné
  - Etaler la sauce et placer les ingrédients
  - Etaler du fromage
  - Placer la pizza dans le four chaud pendant 15 mn
  - Retirer la pizza

Algorithmique

Définitions : Qu'est ce qu'un programme informatique

- Un programme est la traduction d'un algorithme dans un langage de programmation.
- Il existe une multitude de langages informatiques:
  - PHP, Java pour le Web
  - C, Python
  - Cobol pour les gros systèmes bancaires
  - SQL et PL/SQL langage de requêtes de BD
  - Etc …

Algorithmique

Algorithmique et programmation

- Apprendre l'algorithmique, c'est apprendre à manier la structure logique d'un programme informatique. Cette dimension est présente quel que soit le langage de programmation.
- La maîtrise de l'algorithmique requiert deux qualités, très complémentaires :
  - L'intuition : raisonnement qui au départ laborieux devient spontané
  - La rigueur : Méthodique et rigoureux

Algorithmique

Convention d'écriture : le pseudo-code

Début

    remplir une casserole d'eau

    la mettre sur le feu

    porter l'eau à ébullition

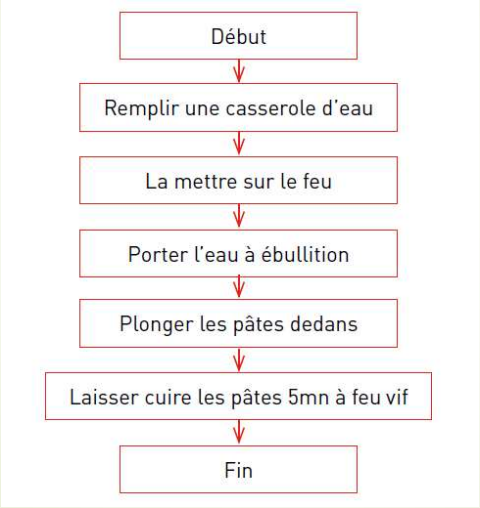
    plonger les pâtes dedans

    laisser cuire les pâtes 5mn à feu vif

Fin

Algorithmique

Convention d'écriture : organigramme



Algorithmique

Les Variables

Afin de stocker provisoirement des données nous utiliserons des variables. Ces données peuvent être de type alphanumérique, numérique, logique.

```
Début
// On initialise la valeur de la somme à 0
somme ← 0
// On ajoute 1 : la valeur de la somme vaut 1
somme ← Somme + 1
// On ajoute 2 : la valeur de la somme vaut 3
somme ← Somme + 2
// On ajoute 3 : la valeur de la somme vaut 6
somme ← Somme + 3
// On ajoute 4 : la valeur de la somme vaut 10
somme ← Somme + 4
// On ajoute 5 : la valeur de la somme vaut 15
somme ← Somme + 5
Fin
```

Algorithmique

Les Variables : Définition

Une variable permet de stocker des informations dont la valeur peut changer. Elle possède un nom, un type (le type de l'information stockée), une adresse mémoire et éventuellement une valeur. Les caractéristiques de chaque variable manipulée par le programmeur doivent être déclarées.

Algorithmique

Les Variables : Déclaration de variables

- une variable est caractérisée par :
  - **un nom**. Ce nom obéit à des impératifs changeant selon les langages. Toutefois, une règle absolue est qu'un nom de variable peut comporter des lettres et des chiffres, mais qu'il exclut la plupart des signes de ponctuation, en particulier les espaces. Un nom de variable correct commence également impérativement par une lettre. Quant au nombre maximal de signes pour un nom de variable, il dépend du langage utilise ;
  - **un type de données** (entier, flottant, caractère, …) ;
  - **une adresse mémoire** que la machine attribut elle-même ;
  - éventuellement, **une valeur.**
- La déclaration d'une variable s'effectue donc en précisant chaque caractéristique.

Algorithmique

Les Variables : Types de données (Numérique)

- Type de données:
  - **Numérique**. Le type numérique permet de stocker des valeurs numériques dans une variable.
    - **Le byte**;
    - **L'entier simple;**
    - **L'entier long;**
    - **Le réel simple;**
    - **Le réel double ou réel long**

Algorithmique

Les Variables : Types de données (Numérique)

Type Numérique	Plage d'utilisation
Byte (octet)	0 à 255
Entier simple	-32 768 à 32 767
Entier long	-2 147 483 648 à 2 147 483 647
Réel simple	-3,40 x 1038 à -1,40 x 1045 pour les valeurs négatives 1,40 x 10-45 à 3,40 x 1038 pour les valeurs positives
Réel double	1,79 x 10308 à -4,94 x 10-324 pour les valeurs négatives 4,94 x 10-324 à 1,79 x 10308 pour les valeurs positives

Algorithmique

Les Variables : Types de données (Caractère)

- Type de données:
  - **Caractère**. Les variables peuvent contenir également des caractères et chaînes de caractères qui peuvent être de tout type :
    - des lettres,
    - des signes de ponctuation,
    - d'espace,
    - ou même des chiffres.
    - Le nombre maximal de caractères pouvant être stockés dans une seule variable dépend du langage utilisé.

Algorithmique

Les Variables : Types de données (Caractère)

- Exemples:

caractère mavar

mavar ← 'm'

chaine de caractère mavar

mavar ← 'Bonjour michel'

Algorithmique

Les Variables : Types de données (Booléen)

- Type de données:
  - **Booléen**. On peut représenter ces notions abstraites de VRAI et de FAUX de deux façons :
    - soit avec les valeurs TRUE et FALSE ;
    - soit avec les nombres 0 et 1.

booléen toto, tata
toto ← VRAI
tata ← FAUX
ou bien :
booléen toto, tata
toto ← 1
tata ← 0

Algorithmique

Les Variables : Instruction d'affectation

- Syntaxe et signification:
  - L'instruction d'affectation permet d'attribuer (ou affecter) une valeur à une variable, c'est-à-dire de remplir la boite. En pseudocode, l'instruction d'affectation est symbolisée par une flèche ←.
- L'ordre dans lequel les instructions sont écrites joue un rôle essentiel dans le résultat final.

```
Début
    entier A
    A ← 34
    A ← 12
Fin
```

Algorithmique

Les Variables : Expressions et opérateurs

- Opérateurs arithmétiques:
  - Ce sont les quatre opérations arithmétiques :
  - + : addition
  - - : soustraction
  - * : multiplication
  - / : division.
- Il est possible d'utiliser des paramètres de façon à définir des priorités, de la même façon qu'en mathématiques
- l'opérateur modulo noté % ou Mod : cet opérateur calcule le reste de la division entière par un autre nombre.

Algorithmique

Les Variables : Expressions et opérateurs

- Opérateur de comparaison:
  - Ces opérateurs permettent d'effectuer des comparaisons entre des nombres ou des variables de type numérique (entier ou flottant). Ces opérateurs renvoient toujours un résultat de type booléen (VRAI ou FAUX).
  - Ces opérateurs correspondent aux opérateurs de comparaison connus en mathématiques :
  - -- l'opérateur supérieur > , inférieur < , supérieur ou égale >= et inférieur ou égal <=
  - -- l'opérateur d'égalité, noté ==
  - -- l'opérateur de différence noté !=:

Exemple : l'expression (1<2) renvoie le résultat VRAI

Une confusion est très souvent effectuée entre l'opérateur d'égalité == et l'opérateur d'affectation ←, parfois également noté = dans les langages de programmation

Algorithmique

Les Variables : Expressions et opérateurs

- Opérateurs logiques (ou booléens):
  - Les opérateurs logiques permettent d'effectuer des opérations logiques entre des opérandes de type booléen. Ces opérateurs renvoient toujours un résultat de type booléen (vrai ou faux).Ces opérateurs correspondent aux opérateurs de comparaison connus en mathématiques :
  - Soit A et B deux booléens. Il existe 3 types d'opérateurs logiques :
  - ET, noté avec le symbole && : Si A "ET" "B" sont VRAI alors le résultat est VRAI. Si l'une de ces deux variables est fausse (A=FAUX ou B=FAUX) alors le résultat est FAUX.
  - OU, noté avec le symbole || : Si A "OU" B est vrai, alors le résultat est VRAI. Si ces deux variables sont fausses. (A=FAUX et B=FAUX) alors le résultat est FAUX.
  - NON, noté avec le symbole ! : Si A est VRAI alors son inverse est FAUX et inversement, si A est FAUX alors son inverse est VRAI.

Algorithmique

Les Variables : Fonctions prédéfinies

- Il existe plusieurs fonctions prédéfinies communes à tous les langages de programmation. Ces fonctions permettent d'effectuer des opérations arithmétiques (calcul du sinus d'un nombre, calcul du nombre de caractères d'une chaine, …), de réduire la complexité des algorithmes et des programmes traduits et d'améliorer leur lisibilité.
  - Fonction sur les entiers :
    - la fonction **ent**(flottant) qui renvoie la partie entière d'un nombre flottant (réel à virgule).
  - Fonctions sur les flottants :
    - la fonction **abs** (flottant) : renvoie la valeur absolue d'un nombre
    - la fonction **cos**(flottant) : renvoie le cosinus d'un angle
    - la fonction **sin**(flottant) : renvoie le sinus d'un angle
    - la fonction **tan**(flottant) : renvoie la tangente d'un angle

Algorithmique

Les Variables : Fonctions prédéfinies

- Fonctions sur les chaines de caractères:
  - la fonction length(chaine) : renvoie le nombre de caractères d'une chaine.
  - la fonction mid(chaine,n1,n2) : renvoie un extrait de la chaine, commençant au caractère n1 et faisant n2 caractères de long.
  - la fonction left(chaine,n) : renvoie les n caractères les plus à gauche d'une chaine.
  - la fonction right(chaine,n) : renvoie les n caractères les plus à droite d'une chaine.

Algorithmique

Les Entrées-Sorties : Lecture et écriture

- Instruction de lecture au clavier:
  - On référence l'instruction de lecture au clavier par le mot "Lire". Dès que le programme rencontre une instruction "Lire", l'exécution s'interrompt, attendant la frappe d'une valeur au clavier. Dès lors, aussitôt que la touche Entrée (Enter) a été frappée, l'exécution reprend.

```
Exemple :

entier mavar se lit / se traduit par "Je déclare un
entier."

Lire mavar se lit / se traduit par "Je saisis la valeur
de cet entier."
```

Algorithmique

Les Entrées-Sorties : Lecture et écriture

- Instruction d'écriture à l'écran :
  - On référence l'instruction de lecture au clavier par le mot "Écrire".
  - Exemple :

```
Début
    Chaine de caractères nom
    Écrire "Entrez votre nom :"
    Lire nom
    Écrire "Votre nom est :" , nom
Fin
```


Algorithmique

Les structures de contrôle : La séquence

- La séquence permet de contrôler l'enchaînement des actions dans le temps. Nous avons déjà vu cette structure car elle correspond aux mots « Début » et « Fin ». Elle permet d'indiquer quelles sont les actions exécutées par l'algorithme.
  - Exemple :

```
Début
    entier a
    Lire a
    Écrire "Voici la valeur de a :", a
Fin
```

Algorithmique

Les structures de contrôle : L'alternative

- L'alternative est une structure de contrôle qui permet de contrôler le déroulement des actions d'un algorithme. Cette structure est fondamentalement différente de la structure de type séquence.
  - Exemple :

```
Début
    remplir une casserole d'eau
    la mettre sur le feu
    porter l'eau à ébullition
    plonger les pâtes dedans
    laisser cuire les pâtes 5mn à feu vif
Fin
```


Algorithmique

Les structures de contrôle : L'alternative

- Dans cet algorithme nous ne contrôlons pas les actions qui sont à exécuter si l'eau n'est pas à ébullition au moment où les pâtes sont plongées dans l'eau. Nous aurions pu ainsi indiquer dans l'algorithme « Si l'eau est à ébullition Alors plonger les pâtes dedans Sinon attendre ».

```
Début
    remplir une casserole d'eau
    la mettre sur le feu
    porter l'eau à ébullition
    Si l'eau est à ébullition Alors
        plonger les pâtes dedans
    Sinon
        Attendre
    Fin Si
    laisser cuire les pâtes 5mn à feu vif
Fin
```

Algorithmique

Les structures de contrôle : Structure de l'alternative

- Il y a deux formes possibles à une structure alternative.

```
Cas 1 :
    Si condition Alors
        action 1
        action 2
Fin Si
Cas 2 :
    Si condition Alors
        action 1
    Sinon
        action 2
Fin Si
```


Algorithmique

Les structures de contrôle : Structure de l'alternative

- Une condition est une expression dont la valeur est VRAI ou FAUX. Cela peut donc être (il n'y a que deux possibilités) :
  - une variable de type booléen ;
  - une expression issue d'un test de comparaison ou d'opérations logiques entre plusieurs comparaisons.
- Une condition est une comparaison. Elle est composée de trois éléments :
  - une valeur
  - un opérateur de comparaison
  - une autre valeur.
- Les valeurs peuvent être a priori de n'importe quel type (numériques, caractères…). Mais si l'on veut que la comparaison ait un sens, il faut que les deux valeurs de la comparaison soient du même type !

Algorithmique

Les structures de contrôle : Conditions composées

- Certains problèmes exigent parfois de formuler des conditions qui ne peuvent pas être exprimées sous la forme simple.
- Les conditions composées sont donc formées d'expressions utilisant les opérateurs logiques vus précédemment . Chacune de ces expressions est une condition (renvoyant un booléen VRAI ou FAUX). La condition composée est formée de l'ensemble des conditions.

Algorithmique

Les structures de contrôle : Conditions composées

- Remarque
  - Le ET a le même sens en informatique que dans le langage courant. Pour que « Condition1 ET Condition2 » soit VRAI, il faut impérativement que Condition1 soit VRAI et que Condition2 soit VRAI. Dans tous les autres cas, « Condition1 ET condition2 » sera faux.
  - Il faut se méfier un peu plus du OU. Pour que «Condition1 OU Condition2» soit VRAI, il suffit que Condition1 soit VRAIE ou que Condition2 soit VRAIE. Le point important est que si Condition1 est VRAIE et que Condition2 est VRAIE aussi, Condition1 OU Condition2 reste VRAIE. Le OU informatique ne veut donc pas dire « ou bien ».
  - Enfin, le NON inverse une condition : NON(Condition1) est VRAI si Condition1 est FAUX, et il sera FAUX si Condition1 est VRAI..

Algorithmique

Les structures de contrôle : ET logique /OU logique ?

- Remarque
  - Spontanément, on pense souvent que ET et OU s'excluent mutuellement, au sens où un problème donné s'exprime soit avec un ET, soit avec un OU. Pourtant, ce n'est pas si évident.

```
Si il fait trop chaud ET il ne pleut pas Alors
    Ouvrir la fenêtre
Sinon
    Fermer la fenêtre
Fin Si
```

Cette petite règle pourrait tout aussi bien être formulée comme suit :

```
Si il ne fait pas trop chaud OU il ne pleut pas Alors
    Fermer la fenêtre
Sinon
    Ouvrir la fenêtre
Fin Si
```


Algorithmique

Les structures de contrôle : Structures imbriquées

- Il est possible d'imbriquer différentes structures alternatives entre elles.

```
Début
    remplir une casserole d'eau
    la mettre sur le feu
    porter l'eau à ébullition
    Si l'eau est à ébullition Alors
        plonger les pâtes dedans
        Si il y a du sel dans la cuisine
            mettre du sel
        Sinon
            mettre un bouillon de poule
        Fin Si
    Sinon
        mettre à feu plus vif
        attendre
    Fin Si
    laisser cuire les pâtes 5mn à feu vif
Fin
```

Algorithmique

Les Boucles

- La notion de boucle
  - La notion de boucle est une notion fondamentale de la programmation. En effet, beaucoup de traitements informatiques sont répétitifs.
  - Par exemple, la création d'un agenda électronique nécessite de saisir un nom, prénom et un numéro de téléphone autant de fois qu'il y a de personnes dans l'agenda.
  - Dans de tels cas, la solution n'est pas d'écrire un programme qui comporte autant d'instructions de saisie qu'il y a de personnes, mais de faire répéter par le programme le jeu d'instructions nécessaires à la saisie d'une seule personne. Pour ce faire, le programmeur utilise des instructions spécifiques appelées structures de répétition ou boucles qui permettent de déterminer la ou les instruction(s) à répéter.

Algorithmique

Les Boucles

- Considérons l'algorithme suivant permettant de faire la somme des nombres entiers de 1 à 100.

```
Début
    entier somme ← 0
    somme ← somme + 1
    somme ← somme + 2
    …
    somme ← somme + 100
Fin
```

Algorithmique

Les Boucles

- Nous voyons ici que l'instruction somme ← somme + x avec x appartenant à l'intervalle [1, 100] se répète 100 fois. L'algorithme précédent peut alors se réécrire de la façon suivante :

```
Début
    entier somme ← 0
    entier i ← 1
    Répéter
        somme ← somme + i
        i ← i + 1
    Tant que i <= 100
Fin
```


Algorithmique

Les Boucles : La boucle « répéter »

- La boucle « Répéter » est une structure répétitive, dont les instructions sont exécutées avant même de tester la condition d'exécution de la boucle. La syntaxe d'une telle structure est la suivante :

```
Répéter
    action 1
    action 2
    …
    action N
Tant que condition
```

Algorithmique

Les Boucles : La boucle « répéter »

- Principe de fonctionnement
- Les actions situées à l'intérieur de la boucle sont exécutées tant que la condition est vraie.
- Les actions sont exécutées au moins une fois, puisque la condition est examinée en fin de boucle, après exécution des actions
- Une action modifiant le résultat de la condition est toujours placée à l'intérieur de la boucle de façon à stopper les répétitions au moment souhaité.
- Si la condition reste toujours vraie, les actions de la boucle sont répétées à l'infini, ce qui est une erreur de programmation. On parle alors de boucle infinie.

Algorithmique

Les Boucles : La boucle « répéter »

```
Début
    flottant note, total, moyenne
    entier nbNote
    caractère rep
    nbNote ← 0
    total ← 0
    moyenne ← 0
    Répéter
        Écrire "Indiquez la note"
        Lire note
        total ← total + note
        nbNote ← nbNote + 1
        Écrire "Voulez-vous continuer ? o/n"
        Lire rep
    Tant que (rep == 'o')
    moyenne ← total / nbNote
    Écrire "La moyenne est de :", moyenne
Fin
```

Algorithmique

Les Boucles : La boucle « tant que »

- La boucle « Tant que » est analogue à la boucle « Répéter » mais la décision de poursuivre la répétition s'effectue en début de boucle.

```
Tant que condition Faire
    action 1
    action 2
    …
    action N
Fin Tant que
```


Algorithmique

Les Boucles : La boucle « tant que »

```
Début
      flottant note, total, moyenne
      entier nbNote
      caractère rep
      nbNote ← 0
      total ← 0
      moyenne ← 0
      rep ← 'o'
      Tant que rep =='o' Faire
            Écrire "Indiquez la note"
            Lire note
            total ← total + note
            nbNote ← nbNote + 1
            Écrire "Voulez-vous continuer ? o/n"
            Lire rep
      Fin Tant que
      moyenne ← total / nbNote
      Écrire "La moyenne est de :", moyenne
Fin
```

Algorithmique

Les Boucles : La boucle « pour »

- La structure « Pour » permet d'écrire des boucles quand on connait à l'avance le nombre d'itérations à exécuter. Elle est équivalente à la boucle « Tant que ».

```
entier i
Pour i allant de valDebut à valFin avec i ← i+pas Faire
      action 1
      action 2
      ...
      action N
Fin Pour
```


Algorithmique

Les Boucles : La boucle « pour »

- Pas d'incrément :
  - Il est possible d'utiliser une valeur de pas différente de 1.
  - Par exemple, si nous souhaitons faire la somme des 100 premiers entiers impairs, il nous faudra compter de la façon suivante : 1, 3, 5, … 99, donc de 2 en 2.
  - A chaque passe, la valeur du compteur devra être augmentée de 2.

```
Début
   entier somme ← 0
   entier i
   Pour i allant de 1 à 100 avec i ← i+2 Faire
      somme ← somme + i
   Fin Pour
Fin
```

Algorithmique

Les Boucles : La boucle « pour »

- Décrémentation :
  - Il est également possible de compter à l'envers : 99, 97, .. ,3,1
  - Ceci revient à compter de 2 en 2 mais en partant de valFin pour descendre jusqu'à valDebut en retranchant la valeur 2 au compteur.

```
Début
    entier somme ← 0
    entier i
    Pour i allant de 99 à 1 avec i ← i-2 Faire
    somme ← somme + i
    Fin Pour
Fin
```


Algorithmique

Les Boucles : La boucle « pour »

```
Début
    flottant note, total, moyenne
    entier i, nbNote
    total ← 0
    moyenne ← 0
    Écrire "Indiquez le nombre de notes à saisir"
    Lire nbNote
    Pour i allant de 1 à nbNote avec i← i+1 Faire
        Écrire "Indiquez la note"
        Lire note
        total ← total + note
    Fin Pour
    moyenne ← total / nbNote
    Écrire "La moyenne est de :", moyenne
Fin
```

Algorithmique

Les Tableaux : Définition

- Un ensemble de valeurs portant le même nom de variable et repérées par un nombre, s'appelle un tableau.
- Le nombre qui, au sein d'un tableau, sert à repérer chaque valeur s'appelle l'indice.
- Chaque fois que l'on doit désigner un élément du tableau, on fait figurer le nom du tableau, suivi de l'indice de l'élément, entre crochets.

Algorithmique

Les Tableaux : Déclaration d'un tableau

- un tableau se déclare de la façon suivante :
  - en utilisant le mot Tableau ;
  - en précisant le type d'éléments (entier, réel, booléen, ...) ;
  - en spécifiant le nom du tableau ;
  - en précisant le nombre d'éléments du tableau entre les crochets.

```
Exemples
    Déclaration d'un tableau nommé tab1 de 100 flottants
    Tableau de flottants tab1[100]
    Déclaration d'un tableau nommé Note de 12 entiers
    Tableau d'entiers Note[12]
```

Algorithmique

Les Tableaux : Parcours d'un tableau

- L'un des avantages des tableaux est de pouvoir utiliser une boucle afin de pouvoir les parcourir, soit en vue :
  - d'affecter et/ou d'afficher une valeur à chaque élément ;
  - d'effectuer des opérations arithmétiques ou de comparaison entre les valeurs des éléments.

```
Début
    Tableau d'entier Notes[12]
    Entier Moy
    Entier somme ← 0
    Pour i allant de 0 à 11 avec i ← i+1 Faire
        Somme ← Somme + Notes[i]
    Fin Pour
    Moy ← Somme / 12
Fin
```


Algorithmique

Les Tableaux : Tableaux Dynamiques

- Il arrive que l'on ne connaisse pas à l'avance le nombre d'éléments que devra comporter un tableau. Nous avons la possibilité de déclarer le tableau sans préciser au départ son nombre d'éléments. Exemple : Tableau de flottants tab1[ ]

```
Début
    Entier nb
    Tableau d'entiers Notes[]
    entier somme ← 0
    entier moy
    Écrire « Combien y a-t-il de notes à saisir »
    Lire nb
    Pour i allant de 0 à nb-1 avec i←i+1 Faire
        Écrire « Saisir note : «
        Lire Notes[i]
    Fin Pour
    Pour i allant de 0 à nb-1 avec i←i+1 Faire
        somme ← somme + Note[i]
    Fin Pour
    moy ← somme / nb
Fin
```

Algorithmique

Les Tableaux : Recherche d'un élément

- Pour rechercher un élément dans un tableau, il suffit de parcourir le tableau et de comparer la valeur recherchée avec la valeur de l'élément courant. Le résultat de cette recherche ne peut prendre que deux formes :
  - VRAI : la valeur recherchée est bien présente dans le tableau ;
  - FAUX : la valeur recherchée n'est pas présente dans le tableau.

```
Début
    Tableau d'entiers Notes[ 5]
    Entier valCherchee ← 20
    booléen trouve ← FAUX
    // Parcours du tableau
    Pour i allant de 0 à 4 avec i←i+1 Faire
        Si (Notes[i]==valCherchee)
            trouve ← VRAI
        Fin Si
    Fin Pour
Fin
```


Algorithmique

Les Tableaux : Tri par sélection

- La technique du tri par sélection est la suivante : on met en bonne position l'élément numéro 1, c'est-à-dire le plus petit. Puis on met en bonne position l'élément suivant. Et ainsi de suite jusqu'au dernier.
- Par exemple, si l'on part du tableau suivant :

45	122	12	3	21	78	64	53	89	28	84	46

- On commence par rechercher, parmi les 12 valeurs, quel est le plus petit élément, et où il se trouve. On l'identifie en quatrième position (c'est le nombre 3), et on l'échange alors avec le premier élément (le nombre 45) :

3	122	12	45	21	78	64	53	89	28	84	46

Algorithmique

Les Tableaux : Tri par sélection

```
Début
      // Première boucle : on parcourt les éléments du tableau
      Pour i allant de 0 à 10 avec i←i+1 Faire
            // Seconde boucle : on recherche l'indice de l'élément qui possède la plus petite valeur
            entier iMin ← i
            Pour j allant de i+1 à 11 avec j←j+1 Faire
                  Si tab[j] < tab[iMin] Alors
                        iMin ← j
                  Fin Si
            Fin Pour
            // Si la valeur de l'indice i est plus grande que celle de l'indice iMin on intervertit les
valeurs
            Si tab[i] > tab[iMin] Alors
                  // On met tab[i] dans une variable temporaire
                  tampon ← tab[i]
                  tab[i] ← tab[iMin]
                  tab[iMin] ← tampon
            Fin Si
      Fin Pour
Fin
```


Algorithmique

Les Tableaux : Tri à bulle

- L'idée de départ du tri à bulles consiste à se dire qu'un tableau trié en ordre croissant, c'est un tableau dans lequel tout élément est plus petit que celui qui le suit.

45	122	12	3	21	78	64	53	89	28	84	46

- On commence par comparer le premier élément (le nombre 45) avec le suivant (le nombre 122). Comme l'ordre est le bon, on passe à l'élément d'après.
- On compare alors le deuxième élément (le nombre 122) avec le troisième (le nombre 12). Comme l'ordre n'est pas le bon alors on permute les deux éléments. Nous obtenons.

45	12	122	3	21	78	64	53	89	28	84	46

Algorithmique

Les Tableaux : Tri à bulle

```
Début
    booléen permutation
    // Au départ on suppose que le tableau n'est pas trié
    // et donc qu'il y a des permutations à faire
    permutation ← VRAI
    // Première boucle : on continue le processus
    // tant qu'il y a des permutations
    Tant que (permutation == VRAI) Faire
        // On met la valeur de permutation à FAUX à fin de
        // s'arrêter s'il n'y a pas de permutation
        permutation ← FAUX
        // Seconde boucle : on parcourt le tableau
```


Algorithmique

Les Tableaux : Tri à bulle

```
        // Seconde boucle : on parcourt le tableau
        Pour i allant de 0 à 11 avec i←i+1 Faire
            Si (tab[i] > tab[i+1]) Alors
                // Il y a une permutation
                permutation ← VRAI
                // On intervertit les valeurs
                tampon ← tab[i]
                tab[i] ← tab[i+1]
                tab[i+1] ← tampon
            Fin Si
        Fin Pour
    Fin Tant que
Fin
```

Algorithmique

Les Tableaux : Tableaux à deux dimensions

- L'informatique nous offre la possibilité de déclarer des tableaux dans lesquels les valeurs ne sont pas repérées par une seule, mais par deux coordonnées.
- Un tel tableau se déclare par exemple ainsi :

```
Tableau d'entier Cases[5][5]
```


Algorithmique

Les Tableaux : Tableaux à deux dimensions

```
Début
    Tableau d'entiers X[2][3]
    entier i, j, val
    val ← 1
    Pour i allant de 0 à 1 avec i←i+1 Faire
        Pour j allant de 0 à 2 avec i←i+1 Faire
            X[i][j] ← Val
            Val ← Val + 1
        Fin Pour
    Fin Pour
    Pour i allant de 0 à 1 avec i←i+1 Faire
        Pour j allant de 0 à 2 avec i←i+1 Faire
            Écrire X[i][j]
        Fin Pour
    Fin Pour
Fin
```

Algorithmique

Les Fonctions: Problématique

- Une application, surtout si elle est longue, a toutes les chances de devoir procéder aux mêmes traitements, ou à des traitements similaires, à plusieurs endroits de son déroulement. Par exemple, la saisie d'une réponse par oui ou par non (et le contrôle qu'elle implique) peut être répétée dix fois, à des moments différents de la même application..

Algorithmique

Les Fonctions: Problématique

```
Début
    Chaîne de caractères Rep1, Rep2
    Écrire "Êtes-vous marié ? Oui / Non"
    Lire Rep1
    Tant que Rep1 != "Oui" && Rep1 != "Non" Faire
        Écrire "Êtes-vous marié ? Oui / Non"
        Lire Rep1
    Fin Tant que
    Écrire "Avez-vous des enfants ? Oui / Non"
    Lire Rep2
    Tant que Rep12!= "Oui" && Rep2 != "Non" Faire
        Écrire "Avez-vous des enfants ? Oui / Non"
        Lire Rep1
    Fin Tant que
Fin
```

Algorithmique

Les Fonctions: Création d'une fonction

- Nous allons donc créer une fonction dont le rôle sera de renvoyer la réponse (oui ou non) de l'utilisateur. Ce mot de « fonction », en l'occurrence, ne doit pas nous surprendre : nous avons étudié précédemment des fonctions fournies avec le langage, et nous avons vu que le but d'une fonction était de renvoyer une valeur (par exemple la fonction modulo notée Mod ou %). Eh bien, c'est exactement la même chose ici, sauf que c'est nous qui allons créer notre propre fonction, que nous appellerons RepOuiNon

Algorithmique

Les Fonctions: Création d'une fonction

```
Fonction RepOuiNon()
// retourne chaîne de caractères
Début
   Chaîne de caractères Rep
   Rep ← «»
   Tant que Rep != "Oui" && Rep !="Non" Faire
      Écrire "Tapez Oui ou Non"
      Lire Rep
   Fin Tant Que
Renvoyer Rep
Fin
```

Algorithmique

Les Fonctions: Création d'une fonction

- On remarque au passage l'apparition d'un nouveau mot-clé : Renvoyer, qui indique quelle valeur doit prendre la fonction lorsqu'elle est utilisée par le programme.

```
Début
    Chaîne de caractères Rep1, Rep2
    Écrire "Êtes-vous marié ? Oui / Non"
    Rep1 ← RepOuiNon()
    Écrire "Avez-vous des enfants ? Oui / Non"
    Rep2 ← RepOuiNon()
Fin
```


Algorithmique

Les Fonctions: Passage d'arguments

- Reprenons l'exemple qui précède et analysons-le. Nous écrivons un message à l'écran, puis appelons la fonction RepOuiNon pour poser une question ; puis, un peu plus loin, on écrit un autre message à l'écran, et on appelle de nouveau la fonction pour poser la même question, etc. C'est une démarche acceptable, mais qui peut encore être améliorée : puisque avant chaque question, on doit écrire un message, autant que cette écriture du message figure directement dans la fonction appelée. Cela implique deux choses :
  - lorsqu'on appelle la fonction, on doit lui préciser quel message elle doit afficher avant de lire la réponse,
  - la fonction doit être « prévenue » qu'elle recevra un message, et être capable de le récupérer pour l'afficher.
- En algorithmique, on dira que le message devient un argument de la fonction.

Algorithmique

Les Fonctions: Passage d'arguments

- La fonction sera dorénavant déclarée comme suit :

```
Fonction RepOuiNon(message) : retourne chaîne de caractères
Début
    Chaîne de caractères Rep
    Écrire message
    TantQue Rep != "Oui" && Rep != "Non" Faire
        Écrire «Tapez Oui ou Non» Lire Truc Rep
    FinTantQue
    Renvoyer Rep
Fin
Rep1 ← RepOuiNon(«Etes-vous marié ?»)
Rep2 ← RepOuiNon(«Avez-vous des enfants ?»)
```


Algorithmique

Les Fonctions: Variable locale et variable globale

- L'existence des fonctions , et des paramètres, pose le problème de la « durée de vie » des variables, et donc ce qu'on appelle leur portée.
  - locale. Cela signifie que la variable est déclarée dans la fonction et disparaît (et sa valeur avec) à la fin de la fonction où elle a été créée. Cette variable n'est pas accessible dans l'algorithme principal ou dans les autres fonctions .
  - globale. Ce qui signifie qu'une telle variable est déclarée à l'extérieur de toute fonction (c'est-à-dire dans l'algorithme principal). Les variables globales sont visibles par toutes les fonctions.

Algorithmique

Les Fonctions: Variable locale et variable globale

- Comment choisir de déclarer une variable locale ou globale ?
- C'est très simple : les variables globales consomment énormément de ressources en mémoire.
- En conséquence, le principe qui doit présider au choix entre variables locales et globales doit être celui de l'économie de moyens : on ne déclare comme publiques que les variables qui doivent absolument l'être. Et chaque fois que possible, lorsqu'on crée une procédure, on utilise le passage de paramètres plutôt que des variables globales.