Rédigé par : Jimmy Paquereau

Fiche de révisions - Algorithmique

1. Généralités

Algorithme : un algorithme est la description d'une procédure à suivre afin de résoudre un problème donné. Il n'est pas nécessairement linéaire. Il est constitué d'instructions (traitements, conditions, boucles…), est typiquement mis en œuvre au moyen d'un langage de programmation, et est exécuté par un ordinateur.

Instruction : une instruction est un traitement élémentaire (opération, condition...).

Langage de programmation : un langage de programmation est un langage, une langue, permettant de mettre en œuvre des algorithmes. Comme toute langue, un langage de programmation possède une syntaxe, une grammaire et un lexique. Il permet à un ordinateur, au moyen d'un compilateur ou d'un interpréteur, d'exécuter un algorithme. On entend par « exécuter un algorithme » le fait, pour un ordinateur, de réagir à la lecture des instructions d'un algorithme (afficher une fenêtre, effectuer une addition ou une multiplication…).

Algorithmique : l'algorithmique est la branche de l'informatique qui étudie les algorithmes indépendamment de tout langage de programmation.

Variable : formellement, une variable est un espace mémoire alloué par un ordinateur. On attribue à une variable un nom et l'on utilise ce nom. En fait, lorsqu'on manipule une variable, on manipule un espace mémoire (une ou plusieurs cases mémoires). Quand on dit j'ai un ordinateur « 32bits » ou « 64bits », il s'agit, pour faire simple, de la taille de chaque case mémoire. Par manipuler une variable, on entend mettre une donnée dans l'espace mémoire qui lui correspond. Le fait d'attribuer une valeur à une variable s'appelle une **affectation**. Il existe divers types de variables. Il est même possible d'en créer. En algorithmique, on parle facilement de : **entier naturel**, **entier**, **réel**, **booléen**, **caractère**, **chaîne de caractères**.

2. Variables

Pourquoi des types de variables ? On le disait, une variable correspond à un espace mémoire. Or, l'espace mémoire nécessaire pour une donnée varie en fonction du type de la donnée que l'on souhaite manipuler.

Entier naturel : les entiers naturels sont les entiers positifs (on a bien dit positif et non strictement positifs, donc 0 à +∞). On notera que la définition formelle (mathématique) d'un nombre entier n'est pas élémentaire.

Entier ou entier relatif : les entiers relatifs sont les entiers positifs ou négatifs.

Réel : la définition formelle d'un nombre réel n'est pas du tout élémentaire. De façon simplifiée, on retiendra qu'il s'agit des nombres décimaux possédant des chiffres avant la virgule, et potentiellement une infinité après. Informatiquement, il est impossible de stocker une infinité de chiffres. On n'en stocke qu'une partie et l'on perd donc de l'information. Les réels sont stockés usuellement sur 32 ou 64bits (4 ou 8 octets). On parle de nombres à virgule flottante.

Booléen : un booléen prend seulement deux valeurs, vrai/faux en algorithmique, true/false ou 1/0 en pratique.

Caractère : vous savez ce que c'est… Un caractère est typiquement stocké sur 7 bits (codage ASCII) ou 8 bits (codage ASCII étendu). A une représentation visuelle d'un caractère (celle que vous connaissez) est associé un

code. L'ensemble des associations s'appelle un jeu de caractères. Il en existe de multiples : ASCII, UTF8, UNICODE…

Chaîne de caractères : on peut voir une chaîne de caractère comme un petit texte. En règle général, et pour simplifier, il s'agit d'un tableau de caractères. Une chaîne de caractères à un espace mémoire généralement de N octets, avec N = 1 x Nombre de caractères, à savoir 1 octet (8 bits) par caractère.

Tableau : oui, on peut définir des tableaux, des tableaux d'entiers, des tableaux de réels… ! Pas besoin de vous faire un dessin, c'est un tableau. On peut même définir des tableaux de tableaux et ainsi de suite. On parle de tableau à 1, 2, 3, …, N dimensions. On récupère chaque élément au moyen de son ou de ses indexes dans le tableau. En algorithmique, **le premier index est 1**, en programmation c'est généralement 0.

Divers : en pratique, il existe des types ou structures dites élémentaires bien plus complexes (exemple : pile, file, arbres, graphe, tas…). Il est encore possible de construire des types, encore plus complexes.

3. Affectation et expressions arithmétiques

Les expressions arithmétiques sont des instructions élémentaires, des calculs (additions, soustractions…), qui permettent d'affecter à une variable le résultat d'un calcul.

Opérateur	:=	+	-	x	/	%
Signification	affectation	addition	soustraction	multiplication	division	modulo

Exemple :

```
UnEntier : entier
UnResultat : entier
// Demande à l'utilisateur de saisir un entier, la valeur saisie est affectée à la variable UnEntier
UnEntier := saisir(''Saisir un entier'')
// Affecte à UnResultat le résultat de l'opération 5 x UnEntier
UnResultat := UnEntier x 5
// Affiche le résultat
afficher(''Le résultat est : '' + UnResultat)
```

Cette ligne est **un commentaire**.

Le « + » ci-dessus représente une addition de chaînes de caractères, ce qui revient à mettre plusieurs chaînes côte-à-côte. Une telle addition de caractères ou chaînes de caractères s'appelle une **concaténation**.

4. Conditions et expressions booléennes

Une expression booléennes est un calcul sur des booléens. Le résultat est booléens, à savoir que le résultat est : « vrai » ou « faux ». Ci-dessous, le résultat des opérations logiques classiques (tables de vérité). On parle d'**opérateurs logiques**.

	A	B	A	B	A	B	A	B
Opération \ Valeurs	FAUX	FAUX	VRAI	FAUX	FAUX	VRAI	VRAI	VRAI
NON A	VRAI		FAUX					
A ET B	FAUX		FAUX		FAUX		VRAI	
A OU B	FAUX		VRAI		VRAI		VRAI	
A OUX B	FAUX		VRAI		VRAI		FAUX	

N.B. : OUX signifie ou exclusif (XOR en anglais). Il existe essentiellement 2 autres opérateurs : NAND et NOR.

Par ailleurs, vous avez à votre disposition les opérateurs de comparaison que vous connaissez depuis bien longtemps à présent :

$$=, \neq, >, <, \leq, \geq$$

Forts de ces opérateurs booléens (opérateurs logiques et opérateurs de comparaison), nous pouvons à présent construire des algorithmes dépendant de conditions. On parle de **branchement conditionnel**. Il s'agit de préciser que l'on effectue un traitement (une série d'instructions), que si une condition est vérifiée, i.e. si une condition est vraie.

SI Expression booléenne 1 **ALORS**
    // Si l'expression booléenne 1 est vraie, je suis ici
    SI Expression booléenne 11 **ALORS**
        …
    SINON SI Expression booléenne 12 **ALORS**
        …
    SINON
        …
    FIN SI
SINON SI Expression booléenne 2 **ALORS**
    // Si l'expression booléenne 1 est fausse et l'expression booléenne 2 est vraie, je suis ici
    …
SINON
    // Si les expressions booléennes 1 et 2 sont fausses, je suis ici
    …
FIN SI

On parle de **si imbriqués** ou de **conditions imbriquées** lorsque l'on a des « si » dans des « si ».
Le « décalage » s'appelle **indentation**.
Commentaires et indentations sont primordiaux pour simplifier la lecture et la compréhension de l'algorithme !

5. Boucles

Les boucles sont appelées **structures itératives**. Il en existe plusieurs. Elles permettent de répéter une série d'instructions soit un certain nombre de fois (potentiellement infini) soit sous condition (tant qu'une condition est vraie ou jusqu'à ce qu'une condition soit vraie).

Boucle POUR : permet de répéter un traitement un certain nombre de fois (potentiellement infini)

i : entier
POUR i **DE** 1 **A** 10 **PAS DE** 1
    …
FIN POUR

Explication : on répète 10 fois la série d'instructions, symbolisée par « … », à savoir on répète ces instructions pour i valant 1, 2, 3, 4, 5, 6, 7, 8, 9 et 10. Le PAS DE 1 signifie qu'on augment i de 1 à chaque tour de boucle. Cette augmentation s'appelle **une incrémentation** si le pas est positif, **une décrémentation** si le pas est négatif.

Boucle TANT QUE : permet de répéter un traitement tant qu'une condition est vraie.

TANT QUE Expression booléenne **FAIRE**
    …
FIN TANT QUE

Ou, afin d'exécuter les traitements « … » au moins une fois :

FAIRE
…
TANT QUE Expression booléenne

6. Algorithme

On vous demande d'écrire un algorithme en lui donnant un nom et en adoptant une syntaxe similaire à la suivante *(Attention ! L'algorithme ci-dessous n'est pas forcément élémentaire. Retenez avant tout la syntaxe !)* :

```
Algorithme CalculerPGCD
Variables
        NB1 : entier
        NB2 : entier
        A, B, D, R, PGCD : entier
DEBUT
        // On demande à l'utilisateur de saisir 2 nombres
        NB1 := saisir(''Saisir un nombre :'')
        NB2 := saisir(''Saisir un deuxième nombre : '')
        A := NB1
        B := NB2
        // Il faut que A et B soient > 0 (on ne sait pas diviser par 0, du moins chez les entiers)
        SI A > 0 ET B > 0 ALORS
                // Si B plus grand que A, on intervertit A et B
                SI B > A ALORS
                        D := B
                        B := A
                        A := D
                FIN SI
                // On effectue des divisions euclidiennes successives (tant que le reste n'est pas nul)
                R := B
                TANT QUE R > 0 FAIRE
                        PGCD := R
                        D := A / B
                        R := A - C x B
                        A := B
                        B := R
                FIN TANT QUE
                // On affiche le PGCD, à savoir le dernier reste non nul
                afficher(''Le PGCD de '' + NB1 + '' et '' + NB2 + '' est '' + PGCD)
        SINON
                afficher(''Chez les entiers, on ne sait pas diviser par 0'')
        FIN SI
FIN
```

Rappel : PGCD (Plus Grand Commun Diviseur). Egalement, comprenez qu'un algorithme ne se lit pas comme une BD. Il faut parfois le lire sereinement et prendre le temps de réfléchir pour le comprendre. Ne soyez pas apeurés !

7. Procédures et fonctions

Procédures et fonctions sont des algorithmes réutilisables, à savoir des algorithmes que l'on peut utiliser dans d'autres algorithmes. Procédures comme fonctions ont un nom. **Une fonction retourne un unique résultat**. **Une procédure ne retourne pas de résultat** (*Mea culpa* : ceci n'est pas tout à fait exact, mais l'on s'en contentera). En fait, nous avons déjà utilisé une procédure et une fonction :
- **afficher(UnTexte : chaîne de caractères)** est une procédure. Elle affiche une chaîne de caractères **passée en paramètre**. UnTexte est une variable appelée **paramètre**.
- **saisir(UnMessage : chaîne de caractères) : entier** est une fonction qui affiche le message UnMessage (message passé en paramètre) et retourne, en l'occurrence, l'entier saisi par l'utilisateur. Ici, entier est le **type de retour**.

```
FONCTION Max(a : entier, b : entier) : entier
        SI a > b ALORS
                RETOURNER a
        FIN SI
        RETOURNER b
FIN FONCTION
```

N.B. : une fonction s'arrête normalement dès que le retour est effectué.

```
PROCEDURE DireMin(a : entier, b : entier)
        SI a > b ALORS
                afficher(''Le min de '' + a + '' et '' + b + '' est '' + b)
        SINON
                afficher(''Le min de '' + a + '' et '' + b + '' est '' + a)
        FIN SI
FIN PROCEDURE

Algorithme DireMinMax
Variables
        A, B, C : entier
DEBUT
        A := saisir(''Saisir un nombre : '')
        B := saisir(''Saisir un autre nombre : '')
        C := Max(A, B)
        Afficher(''Le max de '' + A + '' et '' + B + '' est '' + C)
        DireMin(A, B)
FIN
```

8. Complément sur les expressions booléennes

Comme les additions et multiplications usuelles (sur les entiers, réel…), il y a des priorités de calcul chez les booléens. En logique, le « ET » est noté « . » (multiplication) et le OU est noté « + » (addition). Les opérateurs « ET » et « OU » sont associatifs. Le « ET » est distributifs sur le « OU » et inversement. Le « ET » est prioritaire par rapport au « OU ». Le « NON » est prioritaire par rapport à « ET » et à « OU ». Explicitons.

Disons que A, B et C sont des booléens, à savoir ils valent vrai ou faux, alors :

Associativité :
A ET (B ET C) = (A ET B) ET C
A OU (B OU C) = (A OU B) OU C

Distributivité :

A ET (B OU C) = (A ET B) OU (A ET C) = A ET B OU A ET C (comme le « ET » est prioritaire)

A OU (B ET C) = (A OU B) ET (A OU C) ≠ A OU B ET A OU C (puisque le « ET » est prioritaire)

Priorité du non :

NON A ET B = (NON A) ET B ≠ NON (A ET B) (puisque le « NON » est prioritaire)