Algorithmique :

I. Préambule : pourquoi l'algorithmique :

1. pour concevoir des programmes indépendants du langage utilisé.
2. Quand on a compris les principes de l'algorithmique, on peut s'adapter à tous les langages existants.

II. Définition : écrire un algorithme, c'est décrire dans un langage **compréhensible**, les actions successives permettant de résoudre un problème.

Exemple1 : Ecrire l'algorithme pour « changer une roue de ma voiture ».

Exemple 2 : Ecrire l'algorithme pour « calculer la factorielle d'un entier ».

Exemple 3 : Ecrire l'algorithme pour «convertir une écriture binaire en écriture décimale».

Exemple 4 : Ecrire l'algorithme pour «convertir une écriture décimale en écriture binaire».

NB : la science qui étudie les algorithmes s'appelle l'informatique algorithmique.

Il existe plusieurs approches différentes pour écrire des algorithmes :

- graphique : les algorigrammes. (utilisable pour des programmes très simple, transposable facilement en assembleur, Basic, Cobol,...langage non structuré)

- Structurée (fonctionnelle) : on utilise un **pseudo langage de description d'algorithme**. (Utilisable pour des programmes complexes, facilement transposables dans un langage de programmation structurée pe « C », Pascal, ...)

- Orienté objet : un programme est vu comme un ensemble **d'objet** qui interagissent entre eux. Les objets de même comportement sont regroupé en **classes**. (Utilisable pour des programmes très complexes, facilement transposables dans des langages orientés objets pe C++, Java, C#, …). A envisager dès l'étude informelle du problème avec des outils tel que UML.

III. Variable : case (mémoire) contenant une valeur qui a une signification bien défini dans le cadre du problème à résoudre. On la désigne par un nom qui en général correspond à sa signification.
Comme en mathématique il faudra la **définir** de manière plus ou moins précise.

Exemple : déclaration d'une variable entière appelé i.

1. en pseudo langage algo : « x : entier »
2. en c : « int i; »
3. en Pascal : « i : integer ; »

Définitions :

1. Le fait de modifier sa valeur s'appelle **affectation.**
2. Le fait de comparer deux variables s'appelle une **comparaison. (plus grand, plus petit, égale à).**

NB 1: on doit utiliser deux signes différents pour l'affectation et la comparaison. Attention en mathématiques et dans notre cerveau, on utilise le même signe et un même mot (=) ce qui introduit une ambiguïté (donc risque d'erreur !).

Exemple : en C : Affectation « = » ; comparaison « = = »
en Pascal : affectation « := » comparaison « = »
en pseudo langage algo : pour l'affectation« ← » ; comparaison « = ».

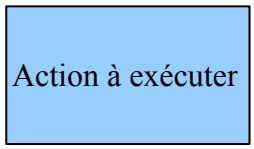
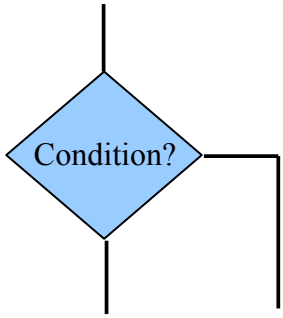
NB 2 : Comment comparer deux réels ?

Soit a une variable réelle :
début
a:réel
entrer le nombre a;
si (a=0) alors
  Afficher « erreur »;
fin
Cet algorithme est-il correct ?

IV. Algorigramme : schéma graphique représentant un algorithme.

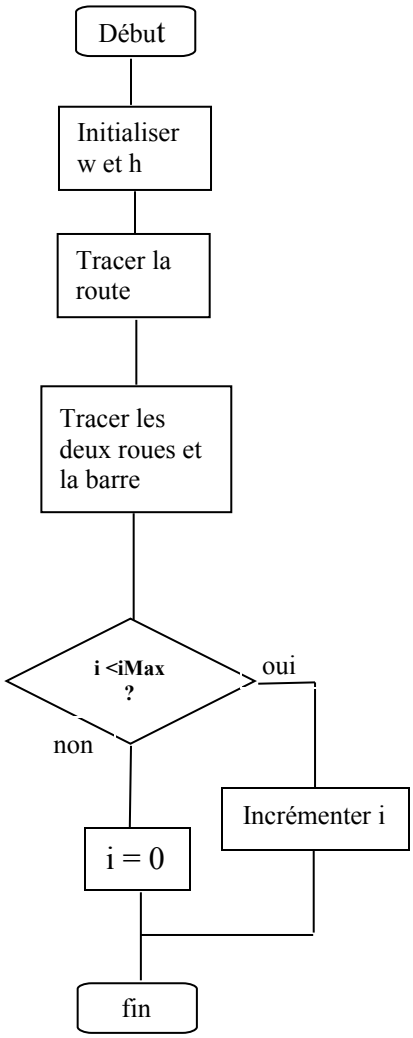
Avantages : concept simple,

Inconvénients : inadapté à un problème compliqué, ressemble rapidement à un paquet de spaghetti mal cuites.

Symbole	Désignation	Commentaire
	Début, Fin, Etiquette	
	Symbole géneral Bloc d'instructions	
	Test et Branchement Suivant la réponse à une question la condition est réalisée ou non.	

Exemple : dans une animation écrite en java dessin d'un petit vélo.

Algorigramme	Extrait du programme

	public void paint(Graphics g) {
	// dessin d'un vélo
	
	int w = getSize().width;
	int h = getSize().height;
	
	g.drawLine(0, R - 3 , w, h + R -3); // diagonal
	
	g.drawArc(i * w / iMax,
	i * h / iMax,
	R, R, 0, 360); // roue 1
	g.drawArc((i + 10) * w / iMax, (i + 10) * h / iMax,
	R, R, 0, 360); // roue 2
	g.drawLine(i * w / iMax +R/2 , i * h / iMax + R/2,
	(i + 10) * w / iMax + R/2,
	(i + 10) * h / iMax + R/2); // Barre
	
	if (i < iMax)
	
	i++;
	else
	i = 0;
	
	}

V. Algorithme structurée :

a) Action : cela peut être :
- opération à exécuter sur et avec des variables.
- une structure quelconque. (c) d),...)

Exemple : i ← i+1

b) Blocs Séquentiels : suite d'actions séparées, regroupées entre un **début** et une **fin**. Les actions sont effectuées les unes après les autres.
Ne contient pas de structure conditionnelles (c) , d), ...)
Exemple :

```
début
s ← s + i
i ← i+1
fin
```

c) Structures conditionnelles : « Si *condition*…. alors faire …. Sinon faire …. ».

```
Si Condition alors faire
    ...
    sinon faire
    ...
```

NB condition = résultat d'un test.
Exemple :

Si (i<iMax) alors faire i ← i + 1; sinon faire i ← 0;	Si (i<iMax) alors i ← i + 1 sinon i ← 0 finSi
Si (i<iMax) alors i ← i + 1 sinon i ← 0 Fsi	On souligne les mots d'algo pour les distinguer des noms de variables.

d) Structure de choix :

« Suivant valeur faire :
Cas Constante1 : Action1;
Cas Constante2 : Action2;
par défaut : ActionPardéfaut;
FinSuivant; »

Exemple :

```
// Car : caractère
suivant Car faire :
        cas 'O' :
                enregistrer()
                fermer()
        cas 'N' :
                fermer()
        defaut :
                // ne rien faire
finsuivant
```

e) Structures itératives :

- « Pour i allant d'un Minimum à maximum avec incrément de N faire ….»

exemple : 2 notations rencontrées.

pour index de 1 à n faire début S ← S + n fin	pour index de 1 à n faire S ← S + n finPour

- « répéter faire …. Jusqu'à ce que la *condition*…. Soit réalisée » on exécute au moins une fois l'action.

Exemple :

répéter S ← S + i i ← i + 1 jusqu'à i >= n	faire S ← S + i i ← i + 1 jusqu'à i >= n

- « tant que *condition* réalisée faire …. » On peut ne jamais exécuter l'Action.

Exemple :

tant que i < n faire début S ← S + i i ← i + 1 fin	tant que i < n faire S ← S + i i ← i + 1 finTQ

f) emboîtements de structures : un bloc séquentiel peut être vu comme une seule action

Exemple : Calcul d'une factorielle :

```
début
n : entier  // déclaration de variable

répéter

entrer n

si n > 0 alors
        début   // n est un entier et a une valeur positive
        i : entier
        aux : entier   // variable de calcul
        aux ← 1
        pour i de 1 à n faire
                début
                aux ← aux * i
                fin
        afficher aux
        fin

jusqu'à n <0

fin
```

g) Fonction : quand un bloc séquentiel est utilisé plusieurs fois, on peut le désigner par un nom appelé **fonction** et le remplacer par ce nom à chaque fois que l'on veut l'utiliser.
Souvent une fonction **reçoit** des paramètres et **retourne (renvoie)** une valeur.

Exemple : dans le programme précédent , on crée la fonction factorielle :

```
Fonction Factorielle( n : entier ) : entier
// n est un entier et a une valeur positive
début
i : entier
aux : entier   // variable de calcul
aux ← 1
pour i de 1 à n faire
      début
      aux ← aux * i
      fin
retourner aux
fin
```

```
Fonction Factorielle( n : entier ) : entier
// n est un entier et a une valeur positive
var
i : entier
aux : entier   // variable de calcul

début
aux ← 1
pour i de 1 à n faire
      début
      aux ← aux * i
      fin
retourner aux
fin
```

Le programme devient :

```
début
n : entier  // déclaration de variable ( cf paragraphe IV)

répéter

entrer( n)      // fonction entrer un caractère

si n > 0 alors Afficher( Factorielle ( n))
// appel de la fonction  avec le paramètre n


jusqu'à n <0
fin
```

Exercice :

1) écrire l'algorithme d'un programme qui permet de calculer la somme des carrés des premiers entiers précédents un nombre n.
On créera une fonction qui calcule le carré d'un entier.
Le programme nous demande un nombre, affiche le résultat jusqu'à ce que l'on donne un nombre négatif.

2) que se passe-t-il pour n = 100 ?
3) Implémenter à l'aide java ou C#.

NB : pour l'entrée du nombre n on utilisera la ligne de code :

```
n = Convert.ToInt32( Console.ReadLine());
```

TSI09 6/10

Exercice :

1) Ecrire l'algorithme d'une Fonction qui calcule les racines d'un polynôme du second degré.
2) Ecrire l'algorithme d'un programme de démonstration de cette fonction.
3) Implémentation C#.

TSI2009

VI. Type des variables :

En mathématique, nous utilisons différents objets :

- Entier.
- Rationnel.
- Complexe.
- Le point.
- La Droite.
- ...

De la même manière, en informatique, nous manipulons des données de natures différentes. La nature d'une donnée contenu dans une variable est appelée **type** de la variable.
Type de bases (ou élémentaires):

- Entier :
- Réel : en fait des décimaux
- Booléen : vrai ou faux (true, false)
- caractère :
- Chaîne de caractères :

le 18/9/08 TSI2008

Le fait de donnée un nom et un type à une variable s'appelle une **déclaration**.

NB 1 : avec certains langages, la déclaration peut-être implicite.
NB 2 : avec le pseudo-langage algorithmique, les déclarations des variables d'une fonction peuvent regroupées avant l'étiquette **début** sous l'étiquette **var**

Une **constante** est une donnée que l'on ne peut pas modifier.
Une **variable** est une donnée que l'on peut modifier.

Exemple :
Constante PI = 3.14; // constante transcendante pi à 2 décimales

var

taille:Réel // variable représentant le diamètre du cercle

nom:chaîne // chaîne de caractères contenant le nom

début

// corps du programme

fin

VII. Les Opérateurs : (à complèter)

1. liste des opérateurs du pseudo-langage:

   (a) Les opérateurs unaires : +, -, non

   (b) Les opérateurs binaires : +, - , *, /, modulo, et, ou, ou exclusif

   (c) Les opérateurs binaires de comparaison : =, ≠ , >, <, ≤, >=
   Type du résultat : booléen.

   (d) Les opérateurs binaires d'affection : ←

Le type du résultat d'une opération dépend du type des opérandes et de l'opération.

2. Priorité des opérateurs : c'est la priorité habituellement utilisée en mathématiques.

3. Avec C#

Unaire		+, -, !, ++x, --x, x++, x--
Binaire	Arithmétique - multiplication	*, /, %
	Arithmétique - addition	+, -
	Décalage	<<, >>
	Relationnel et test de type	<, >, <=, >=,
	Égalité	==, !=
	Logique binaire	&, |
	Conditionnel, par ordre de priorité	&&, ||,
Assignation		=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
Ternaire		? :

4. L-Value :

Quand on réalise un calcul, on utilise un opérateur d'affectation (noté <- ou =).

Par exemple : S <- S + n;

C'est un opérateur binaire non commutatif :

en effet écrire a <- b , ce n'est pas la même chose que écrire b <- a.

L'opérande de gauche s'appelle une L-value (pour left) car cela ne peut pas être n'importe quoi.

Par exemple : on n'a pas le droit d'écrire a+b <- 3 , ni 3 <- b,

VIII. Les types de données complexes :

1) Structure : Une structure permet de regrouper plusieurs variables qui peuvent être de types différents.

Exemple :

// Définition d'un point :

Structure unPointEcran **c'est**
X**:Entier**
Y**:Entier**
FinStructure

// définition d'une date :

Structure uneDate **c'est**
JJ**:Entier**
MM**:Entier**
AAAA:**Entier**
FinStructure

Exercice : donnée la structure de donnée permettant de décrire une entrée d'un répertoire téléphonique avec le nom, le prénom, le téléphone

uneEntrée

2) Tableau :

a) Définition :

Un tableau permet de regrouper plusieurs variables de **même** types. Chaque variable est repérée par un numéro appelé **index**.

Exemple :

Tableau ListeEntierJusquaN **composé de** N+1 **entier;**

Exercice :

Déclarer un tableau pouvant contenir une chaîne de caractère de taille maximale 255.

TSI le 22/09/08

NB : un tableau peut avoir plusieurs dimensions.

Exemple : **Tableau** Matrice composé de 3 Réels **par** 3 Réels
ou
Tableau Matrice composé de 3 tableau composés de 3 Réels

Exercice : déclarer un tableau représentant une table de jeu de morpion.

b) Utilisation :

Accès au tableau : on donne le nom du tableau et l'index de la variable à laquelle on veut accéder.

Exemple : ListeEntierJusquaN[2] ← -11 // place le nombre -11 à la troisième place du tableau.

Placer le texte « dupond » dans la chaîne de caractères nom

nom[0] ← 'd'
nom[1] ← 'u'
nom[2] ← 'p'
nom[3] ← 'o'
nom[4] ← 'n'
nom[5] ← 'd'
nom[6] ← FinDeChaîne

Exercice : donner l'algorithme permettant de d'initialiser le jeu de morpion.

NB 1 : Certains langages commencent à numéroter à partir de 0 d'autres à partir de 1 d'autres par une valeur arbitraire.
NB 2 : l'accès à des variables de tableaux à plusieurs dimensions peut poser problème.

3) Type de données :

Quand on utilise plusieurs fois une **structure**, on peut un nouveau **type de donnée.**

Exemple :

TypeDeDonnée Date
Structure
JJ**:Entier**
MM**:Entier**
AAAA:**Entier**
FinStructure
FinTypeDeDonnée

// on pourra déclarer une variable en utilisant ce nouveau type de donnée

DateDuJour : Date ;

Début
// comme avec les sgbd JJ est un champs ou un enregistrement
// JJ est un champs de la varibale dateDuJour qui est une structure de type Date
dateDuJour.JJ <- 20 ;
dateDuJour.MM <- 10 ;
dateDuJour.AAAA <- 2009 ;

fin

Exercice :

- décrire un type de donnée représentant les entrées dans un répertoire.
- Déclarer une structure de donnée représentant un répertoire.(on se servira d'un tableau d'entrée (100 maximum) et d'un entier indiquant le nombre d'entrées valides dans le tableau.)

IX. Retour sur les fonctions :

Passage d'un paramètre en entrée (on dit par **Valeur**): la fonction peut lire la valeur d'entrée mais ne peut pas modifier le paramètre.
Passage d'un paramètre en entrée et sortie (on dit par **Référence ou par adresse)** : la fonction peut lire la valeur du paramètre mais aussi écrire une valeur dans le paramètre.
Passage d'un paramètre en sortie : la fonction peut écrire une valeur dans le paramètre mais n'utilise pas sa valeur d'entrée.

On adoptera la notation utilisée dans les exemples suivant :

Fonction Majuscule(Entrée caractère cEntrée; Sortie caractère cSortie):rien
// lit la valeur de cEntrée, calcule la majuscule de cEntrée place le résultat dans cSortie
Début

... à compléter

retourner rien;
Fin

Fonction Majuscule(EntréeSortie caractère c):rien
// lit la valeur de c calcule la majuscule de c place le résultat dans c
Début

... à compléter

retourner rien;
Fin

Fonction Majuscule(Entrée caractère c):caractère
// lit la valeur de c, calcule la majuscule de c et retourne le résultat
Début

... à compléter

retourner c;
Fin

NB1 : En C , le passage par référence n'existe pas. Pour palier cette absence, on passe par valeur l'**adresse** des variables au moyen de deux types de données spéciaux appelés « pointeurs » et par adresse (en C++). (cf exemple)
NB2 : le passage par valeur peut poser problème.
NB3 : une **procédure** est une fonction qui ne retourne aucune valeur. (cette notion n'existe pas explicitement dans tous les langages).

NB4 : dans la machine que ce passe-t-il lors de l'appel d'un sous-programme :

adresse, registres, pile, tas, ...

X. Application :

1. Avec le répertoire décrit précédemment, écrire l'algorithme de deux fonctions :
   a) Insertion d'une entrée dans le répertoire.
   b) Afficher la liste des entrées : nom, prénom, adresse.

2. Ecrire un programme de démonstration dans le langage C en mode Console effectuant le travail suivant :

   a) insertion de 4 entrées avec les données de votre choix. (pe : dupond, henry, 0188888888, …)
   b) Affichage de la liste.