Introduction à l'algorithmique

Mounir T. El Araki

mounir.elarakitantaoui@uic.ac.ma

Plan du cours

- Eléments d'un algorithme
- Les structures alternatives et répétitives
- Les tableaux
- Les fonctions prédéfinis
- Les structures de données
  - Les piles/ Les files / les listes chainées
- Les algorithme de tri et de recherche

Plan du cours

	Dates	Heures	Contenu
S1	13/2	2	Introduction et les structures de base
S2	20/2	4	Les structures alternatives et répétitives
S3	**27/2**		**Pas cours**
S4	6/3	2	Les structures des tableaux
S5	13/3	2	TD 1
S6	**20/3**	**2**	**Contrôle 1**
S7	27/3	2	Les tableaux à plus d'une dimension
S8	**3/4**		**Vacances**
S9	10/4	2	Les fonctions et les fonctions prédéfinis
S10	17/4	2	Les structures de données
S11	**24/4**	**2**	**TD 2**
S12	1/5	2	Contrôle 2
S13	8/5	2	Implémentation des structures de données
S14	15/5	2	Les algorithmes de tri

Cours

- 2 heures par semaine
- 2 Travaux Dirigés
- 2 Contrôles continus (30 %)
- Assiduité & BlackBoard (Participation) (20 %)
- 1 Examen final (50 %)

Plan du cours

- **Eléments d'un algorithme**
- Les structures alternatives et répétitives
- Les tableaux
- Les fonctions prédéfinis
- Les structures de données
  - Les piles/ Les files / les listes chainées
- Les algorithme de tri et de recherche

Introduction

- **INFORMATIQUE**
  - Science du traitement rationnel et automatique de l'information ; l'ensemble des applications de cette science.
  - Qui se rapporte à l'informatique. Système informatique, ensemble des moyens qui permettent de conserver, de traiter et de transmettre l'information.

- **INSTRUCTION**
  - Ordre, indication qu'on donne à quelqu'un pour la conduite d'une affaire ; directive, consigne.
    - Instructions pour la mise en marche d'un appareil.
    - Informatique : Consigne formulée dans un langage de programmation, selon un code.

- **LOGICIEL**
  - Dérivé de logique.
  - Ensemble structurée de programmes remplissant une fonction déterminée, permettant l'accomplissement d'une tâche donnée.

Notion d'algorithme

- Algorithme
  - Exemple;
    - Mode d'emploi
    - Itinéraire routier
    - Recette de cuisine
- Algorithme sert à transmettre un savoir faire
- Algorithme décrit des étapes à suivre pour réaliser un travail
  - Algorithme pour informaticien → recette pour le cuisinier

- Algorithme : préparer une pâte
  - Ingrédients;
    - 250 g de farine
    - 50 g de beurre
    - 1 verre de lait
  - Actions;
    - Début
      - Incorporer le beurre dans la farine
      - Pétrir le mélange jusqu'à ce qu'il soit homogène
      - Ajouter du lait
      - Mélanger
      - Si la pâte est trop sèche, alors ajouter du lait
      - Si la pâte à une bonne consistance, alors laisser reposer une demi heure
      - Faire cuite la pâte
    - Fin

Définition

- Algorithme → Algorismus (latin) → AL KHWARIZMI (mathématicien arabe)
- Un algorithme est une suite ordonnée d'instructions qui indique la démarche à suivre pour résoudre une série de problèmes équivalents.
- Un algorithme est une séquence d'opérations visant à la résolution d'un problème en un temps fini.
- Un algorithme est la description de la méthode de résolution d'un problème quelconque en utilisant des instructions élémentaires
  - Les instructions deviennent compréhensibles par l'ordinateur lors de la traduction de l'algorithme en un programme

Exemples d'algorithmes

- Extrait du mode d'emploi d'un télécopieur concernant l'envoi d'un document.
  1. Insérez le document dans le chargeur automatique.
  2. Composez le numéro de fax du destinataire à l'aide du pavé numérique.
  3. Enfoncez la touche envoi pour lancer l'´emission.

- On cherche à faire dessiner une figure géométrique sur la plage à quelqu'un qui a les yeux bandés.
  - Pour cela, on ne dispose que de 2 commandes orales :
    - avancer de n pas en avant (n est un nombre entier de pas) et
    - tourner à gauche d'un angle (rotation sur place de θ).

  - Faire dessiner un triangle régulier de 10 pas de côté.

Un triangle

Un carré

Un pentagone

Un hexagone

Un octogone

Un Polygone

Algorithmique

- L'algorithmique est la science des algorithmes.
  - L'algorithmique s'intéresse à l'art de construire des algorithmes ainsi qu'à caractériser leur validité, leur robustesse, leur réutilisabilité, leur complexité ou leur efficacité.

- Validité d'un algorithme
  - La validité d'un algorithme est son aptitude à réaliser exactement la tâche pour laquelle il a été conçu.
    - Recherche d'une gare ?
    - (vous allez tout droit jusqu'au prochain carrefour, vous prenez à gauche au carrefour et ensuite la troisième à droite, et vous verrez la gare juste en face de vous.)

- Robustesse d'un algorithme
  - La robustesse d'un algorithme est son aptitude à se protéger de conditions anormales d'utilisation.
  - (les instructions sont elles pour un piéton , ou un automobiliste ? Et si la 3^{ième} à droite était interdit)

Algorithmique

- Réutilisabilité d'un algorithme
  - La réutilisabilité d'un algorithme est son aptitude à être réutilisé pour résoudre des tâches équivalentes à celle pour laquelle il a été conçu.
    - Chemin à la mairie ? (peut on réutiliser le chemin vers la gare ?)

- Complexité d'un algorithme
  - La complexité d'un algorithme est le nombre d'instructions élémentaires à exécuter pour réaliser la tâche pour laquelle il a été conçu.
    - Nombre de pas pour y arriver ?

- Efficacité d'un algorithme
  - L'efficacité d'un algorithme est son aptitude à utiliser de manière optimale les ressources du matériel qui l'exécute
    - N'existerait-il pas un raccourci piétonnier pour arriver plus vite à la gare ?

Définition (Algorithme et programmation)

- Tout problème à programmer doit être résolu sous forme algorithmique d'abord, ensuite converti dans un langage de programmation.

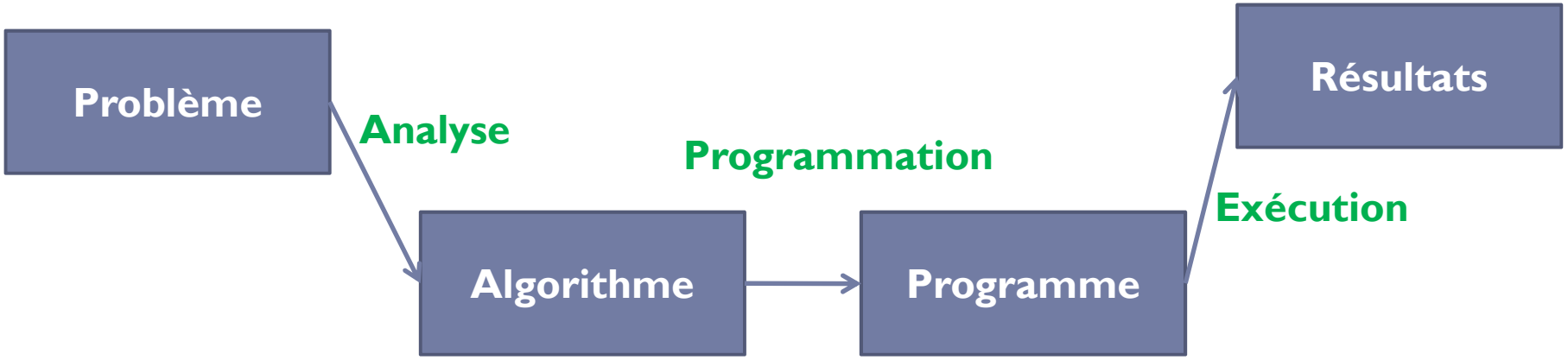
- Algorithme → **indépendant du langage de programmation utilisé**

- Un programme est un enchaînement d'instructions, écrit dans un langage de programmation, exécutées par un ordinateur, traitant un problème et renvoyant les résultats.

- Un programme représente **la traduction d'un algorithme** à l'aide d'un langage de programmation

Définition (Algorithme et programmation)

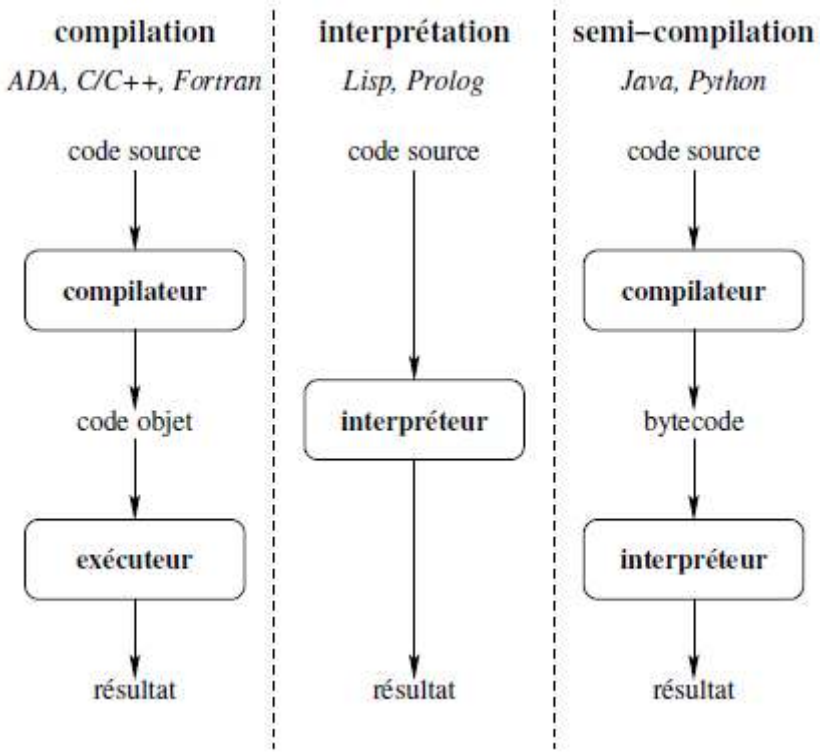
- Le cycle de développement d'un programme (ou une application) informatique:



- Exemple de langage de programmation: Pascal, C, C#, C++, VB, Java, J#

Programmation et compilation

- Compilateur
  - Un compilateur est un programme informatique qui traduit un langage, le langage source, en un autre, appelé le langage cible.
- Interpréteur
  - Un interpréteur est un outil informatique (logiciel ou matériel) ayant pour tâche d'analyser et d'exécuter un programme écrit dans un langage source.
- Langage de programmation
  - Un langage de programmation est un langage informatique, permettant à un humain d'´ecrire un code source qui sera analysé par un ordinateur.



Programmation et compilation

- Compilation
  - La compilation consiste à traduire la totalité du code source en une fois.
  - Le compilateur lit toutes les lignes du programme source et produit une nouvelle suite de codes appelé programme objet (ou code objet).
  - Celui-ci peut être exécuté indépendamment du compilateur et être conservé tel quel dans un fichier (« fichier exécutable»).
  - Les langages Ada, C, C++ et Fortran sont des exemples de langages compilés.

- Interprétation
  - L'interprétation consiste à traduire chaque ligne du programme source en quelques instructions du langage machine, qui sont ensuite directement exécutées au fur et à mesure (« au fil de l'eau »).
  - Aucun programme objet n'est généré.
  - L'interpréteur doit être utilisé chaque fois que l'on veut faire fonctionner le programme.
  - Les langages Lisp et Prolog sont des exemples de langages interprétés.

Programmation et compilation

- Semi compilation
  - Certains langages tentent de combiner les deux techniques afin de retirer le meilleur de chacune. (Langages Java).
  - De tels langages commencent par compiler le code source pour produire un code intermédiaire, similaire à un langage machine (mais pour une machine virtuelle), que l'on appelle **bytecode**, lequel sera ensuite transmis à un interpréteur pour l'exécution finale.
  - Du point de vue de l'ordinateur, le bytecode est facile à interpréter en langage machine.
  - Cette interprétation sera plus rapide que celle d'un code source.

Conventions d'écriture des algorithmes

- Organigrammes
  - Représentation graphique, avec des carrés, des losanges,
    - Représentation quasiment abandonnée.
      - Ce n'est pas pratique quant la logique de l'algorithme devient plus complexe.
      - En plus cette représentation favorise un certain type de programmation, dite non structurée, que l'on tente au contraire d'éviter.

- Pseudo-code
  - Une série de conventions qui ressemble à un langage de programmation authentique dont on aurait évacué la plupart des problèmes de syntaxe.
  - Le pseudo-code, encore une fois, est purement conventionnel ; aucune machine n'est censée le reconnaître.

Structure générale d'un algorithme (Quelques conventions)

- Composé de 3 parties principales

- **L' entête :** cette partie sert à donner un nom à l'algorithme. Elle est précédée par le mot Algorithme;

- **La partie déclarative :** Dans cette partie, on déclare les différents objets que l'algorithme utilise (constantes, variables, etc..)

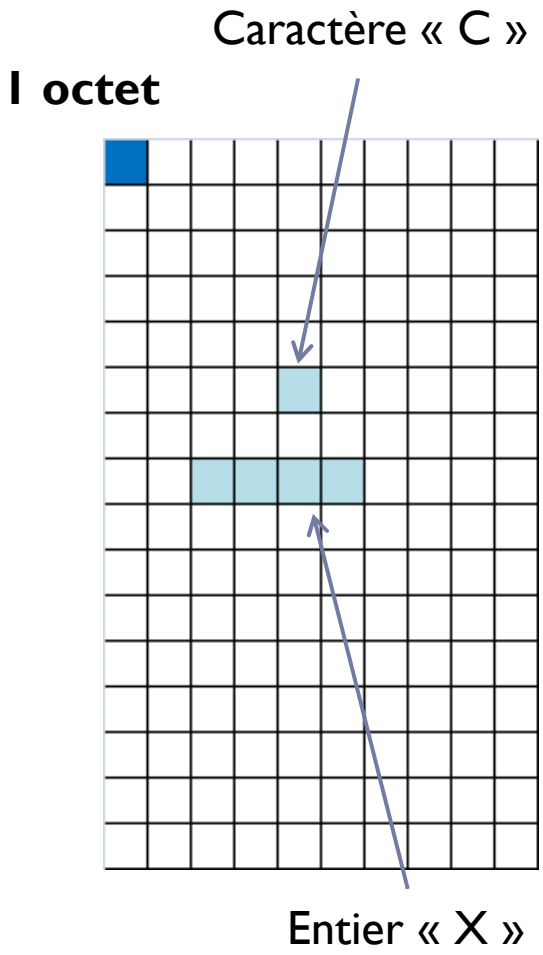
- **Le corps de l'algorithme :** cette partie contient les instructions de l'algorithme. Elle est délimitée par les mots Début et Fin.

Structure générale d'un algorithme (Quelques conventions)

Entête		**Algorithme NomAlgorithme**
Partie déclarative		**Constante** Identificateur **= valeur** **Variable** Identificateur **: type**
Corps de l'algorithme		**Début** Instruction 1 Instruction 2 Instruction 3 Instruction 4 …… Instruction n **Fin**

Les variables et les constantes

- Les données, résultats intermédiaires et résultats finaux sont stockés dans des **cases mémoires**.
- **Variables** correspondent à des espaces mémoires
- Une variable est rangée dans un emplacement mémoire nommé, de taille fixe ou non, prenant au cours du déroulement de l'algorithme un nombre indéfini de valeurs différentes.



Déclaration des variables

- La partie déclaration consiste à énumérer toutes les variables dont on aura besoin au cours de l'algorithme
- Chaque déclaration doit comporter le nom de la variable (identificateur) et son type.

- Syntaxe :
  - **Variable** *identificateur* **: type**

- Exemple :
  - **Variable** *surface* **: réel**
  - **Variable** *i* **: entier**
  - **Variable** *a,b,c,d* **: entier**
  - **Variable** *Nom_Prénom* **: chaîne**
  - **Variable** *Absent* **: logique**

Déclaration des variables

- Un identificateur est un nom donné à une variable, une fonction, etc.
  - Le nom doit commencer par une lettre suivie de lettres et de chiffres (notre convention)
  - Ne doit pas contenir d'espace (notre convention)

- Type de données:
  - Type **Entier :** *sert à manipuler les nombres entiers positifs ou négatifs*
  - Type **Réel :** *sert à manipuler les nombres réels*
  - Type **Caractère :** *sert à manipuler des caractères alphabétiques et numériques, 'a', 'A', 'z', '?', '4', etc*
  - Type **Chaîne :** *sert à manipuler les chaînes de caractères permettant de représenter des mots ou des phrases. Ex: 'Bonjour' etc*
  - Type **Logique (Booléen)** *pour utiliser les expressions logiques, il n'y a que deux valeurs :* ***Vrai*** *et* ***Faux***

- *Autres type de données*
  - *Type* ***Date***
  - *Type* ***Monnaie***

Domaine ou type de données des variables numériques

Type Numérique	Plage
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	$-3,40x10^{38}$ à $-1,40x10^{45}$ pour les valeurs négatives $1,40x10^{-45}$ à $3,40x10^{38}$ pour les valeurs positives
Réel double	$1,79x10^{308}$ à $-4,94x10^{-324}$ pour les valeurs négatives $4,94x10^{-324}$ à $1,79x10^{308}$ pour les valeurs positives

Opérations sur les variables

Type	Opérations possibles	Symbole ou mot clé
Entier	Addition	+
	Soustraction	-
	Multiplication	*
	Division	/
	Division entière	(DIV ou %)
	Modulo (reste de la division entière)	MOD
	x' exposant 'y'	^ ou pow(x,y)
	Comparaisons	<, =, >, <=, >=, <>
Réel	Addition	+
	Soustraction	-
	Multiplication	*
	Division	/
	Exposant	
	Comparaisons	<, =, >, <=, >=, <>
Caractère	Comparaisons	<, =, >, <=, >=, <>
Chaîne	Concaténation	(+ ou &)
	Comparaisons	<, =, >, <=, >=, <>
Booléen	Logiques	ET, OU, NON, Ouex

Priorité des opérateurs

Priorité par ordre croissant	opérateur
	()
	Non
	*, /, DIV, MOD
	+, -
	<, <=, >, >=
	=
	ET
	OU
	←

Les constantes

- Comme la variable, une constante correspond à un emplacement mémoire réservé auquel on accède par le nom qui lui a été attribué, mais dont la valeur stockée ne sera jamais modifiée au cours du programme.

- Syntaxe :
  - **Constante** *Nom_de_la_constante* **= valeur**

- Exemple :
  - **Constante** *PI* **= 3.14**
  - **Constante** *i* **= *1***
  - **Constante** *UneChaîne* **='Introduction à l'algorithmique'**
  - **Constante** *V* **= Vrai**

Instructions de base

- Une instruction de base est une action élémentaire commandant à la machine un calcul, ou une communication avec l'un de ses périphériques d'entrées ou sorties.

- Les instructions de base sont:
  - L'instruction d'affectation
  - L'instruction d'entrée
  - L'instruction de sortie

L'instruction de l'affectation ←

- L'affectation permet d'affecter une valeur à une variable.
- Symbolisée par ← , qui symbolise le sens de l'affectation.

- Syntaxe : (Attention dans un langage de programmation c'est '=')
  - *Variable* ← **Expression**
- L'expression peut être soit:
  - Identificateur;
  - Constante;
  - Expression arithmétique;
  - Expression logique;

  - Une expression est un ensemble de valeurs, reliées par des opérateurs, et équivalent à une seule valeur

L'instruction de l'affectation ←

- variable expression : L'expression peut être n'importe quelle expression évaluable telle qu'une
  - opération logique (x ← Vrai OU Faux ET Non Vrai),
  - une opération arithmétique (x ← 3 + 2*9 - 6*7),
  - un appel de fonction (y ← sin(x)) ou
  - toute autre combinaison evaluable (x ← (x <> y) ET (z + t >= y) OU (sin(x) < 0)).

- L'expression du membre de droite peut faire intervenir la variable du membre de gauche comme dans i ← i + 1 (une incrémentation)

- L'affectation a pour effet de réaliser plusieurs opérations dans la mémoire de l'ordinateur:
  - créer et mémoriser un nom de variable,
  - lui attribuer un type bien déterminée ,
  - créer et mémoriser une valeur particulière,
  - établir un lien (par un système interne de pointeurs) entre le nom de la variable et l'emplacement mémoire de la valeur correspondante.

La sémantique de l'affectation ←

- Une affectation peut être définie en deux étapes
  - Evaluation de l'expression qui se trouve dans la partie droite de l'affectation
  - Placement de cette valeur dans la variable

- Algorithme Calcul
  - (1) Variables A, B, C, D : Entier
  - (2) Début
  - (3) A ← 10
  - (4) B ← 30
  - (5) C ← A + B
  - (6) D ← C * A
  - (7) Fin

		Variables			
		A	B	C	D
N° de lignes	1	?	?	?	?
	2	?	?	?	?
	3	10	?	?	?
	4	10	30	?	?
	5	10	30	40	?
	6	10	30	40	400

Exemples

- **Algorithme Exemple n°1**
  Variable A, B
  Début
  A ← 'C'
  B ← 'A'
  Fin

Algorithme Exemple n°2
Variable A, B
Début
A ← 'C'
B ← A
Fin

- **Algorithme Exemple n°3**
  Variable A
  Début
  A ← 59
  A ← 17
  Fin

Algorithme Exemple n°4
Variable A,
Début
A ← 17
B ← 59
Fin

Autre exemple de l'affectation

- Les variables (numériques) ne sont pas forcément initialisées (à zéro) → leurs valeurs initiales peut être n'importe quoi

- Algorithme Logique
  - (1) Variables A, B, C : Booléen
  - (2) Début
  - (3) A ← Vrai
  - (4) B ← Faux
  - (5) C ← A et B
  - (6) Fin

		Variables		
		A	B	C
N° de lignes	1	?	?	?
	2	?	?	?
	3	VRAI	?	?
	4	VRAI	FAUX	?
	5	VRAI	FAUX	FAUX

Exercices (Affectation)

- Exercices (instruction de l'affectation)

L'instruction d'entrée (Lecture)

- L'instruction d'entrée ou de lecture donne la main à l'utilisateur pour saisir une donnée au clavier.
- La valeur saisie sera affectée à une variable

- Syntaxe :
  - **Lire(** *Identificateur* **)**
- Exemple
  - Lire (A)
  - Lire (A, B, C)

- En réalité, quand un programme rencontre cette instruction, l'exécution s'interrompt et attend que l'utilisateur tape une valeur. Cette valeur est rangée en mémoire dans la variable désignée.

L'instruction de sortie (Ecriture)

- L'instruction de sortie (Ecriture) permet d'afficher des informations à l'écran.

- Syntaxe :
  - **Ecrire (** *Expression* **)**
- Exemple 1
  - Ecrire (A)

- Exemple 1
  - A ← 7
  - Ecrire ('La valeur de A est =', A)

- Avant de lire une variable, il est conseillé d'écrire des libellés à l'écran, afin de prévenir l'utilisateur de qu'il doit taper
  - Sinon l'utilisateur passera son temps à se demander ce que l'ordinateur attend de lui

Les commentaires

- Lorsqu'un algorithme devient long, il est conseillé d'ajouter des lignes de commentaires dans l'algorithme; càd des lignes qui ont pour but de donner des indications sur les instructions effectuées et d'expliquer le fonctionnement du programme (Algorithme) sans que le compilateur ne les prenne en compte.

- Syntaxe :
  - **// Commentaire sur une ligne**
  - **/* Commentaire */**
  - **/* Commentaire**
  - **‘ Commentaire**

- Parfois on utilise les commentaires pour annuler l'action de quelques instructions dans un algorithme ou un programme au lieu de les effacer.

Exercices (Lecture et écriture)

- Exercices (Lecture et écriture)

Exercice (Calcul de Prix TTC)

- Ecrire un algorithme qui permet de saisir le prix HT (PHT) d'un article et de calculer son prix total TTC (PTTC) . TVA = 20%

  - Algorithme Calcul PTTC
    - (1) Variable PHT, PTTC : réel
    - (2) Constante TVA = 0,2
    - (2) Début
    - (3) Ecrire ('Entrez le prix hors taxes:')
    - (4) Lire (PHT)
    - (5) PTTC ← PHT + (PHT * TVA)
    - (6) Ecrire ('Le prix TTC est :', PTTC)
    - (7) Fin

Exercices

- Exercice 1
  - Ecrire un algorithme permettant de calculer la moyenne de deux entiers
- Exercice 2
  - Une grande surface accorde à tous ces clients, une réduction de 3% sur chaque montant d'achat. Ecrire un algorithme permettant de saisir le montant d'achat (MA), de calculer le montant de la remise (R), et le montant à payer (MP).

la vérification **méthodique**, pas à pas, de vos algorithmes représente plus de la moitié du travail à accomplir... et le gage de vos progrès.

Exercice (Permutation de deux variables)

- Ecrire un algorithme qui permute les valeurs de deux variables lues aux clavier

  - Algorithme Permutation
    - (1) Variable N_1, N_2, Z : réel
    - (2) // Z variable temporaire qui sera utilisé pour permuter N_1 et N_2
    - (3) Début
    - (3) Ecrire ('Entrez les valeurs de N_1 et N_2:')
    - (4) Lire (N_1,N_2)
    - (5) Z ← N_1
    - (6) N_1 ← N_2
    - (7) N2 ← Z
    - (8) Ecrire ('Les valeurs de N1 et N2 après permutation : N1=', N1,' et N2=' , N2)
    - (7) Fin

Exercice (Evaluation des expressions)

- Donner les valeurs de variables suivantes:

  - a ← 7/2
  - b ← 7 DIV 2
  - c ← 7 MOD 2
  - d ← 't' < 'w'
  - e ← 'Maman' > 'Papa'
  - f ← (5<>2)
  - g ← 'maman' > 'Papa'
  - h ← Non(5=2)
  - i ← (4<6) et (9>2)
  - j ← (2<0) ou (4<>4)
  - k ← 'A' < 'a'

Exercice (Priorité des opérateurs)

- Donner les valeurs des variables suivantes:

- a ← 4* 2 + 5
- b ← 5 + 3*2 -6
- c ← a > b ET 7<>2 OU a < b
- d ← a > b ET 7 =2 OU a < b

Plan du cours

- Eléments d'un algorithme
- **Les structures alternatives et répétitives**
- Les tableaux
- Les fonctions prédéfinis
- Les fichiers
- Les structures de données
  - Les piles/ Les files / les listes chainées
- Les algorithme de tri et de recherche

Structures alternatives

- Traitement séquentiel
- Traitement alternative
  - Exécuter une série d'instruction selon la valeur d'une condition

- Syntaxe :
  - **Si (** *condition* **) Alors**

    Instruction(s) 1
  - **Sinon**

    Instruction(s) 2
  - **FinSi**

- Une condition est une expression logique ou une variable logique évaluée à **Vrai** ou **Faux**.

Exercice

- Ecrire un algorithme qui affiche si un nombre entier saisi au clavier est pair ou impair

```
Algorithme Parité
// Déclaration des variables
Variable n, r : entier
Début
        Ecrire ('Entrez la valeur de n:')
        Lire (n)
        // r est le reste de la division entière de n par 2
        r ← n Mod 2
        Si r=1 Alors
            Ecrire (n,'est impair')
        Sinon
            Ecrire (n,'est pair')
        FinSi
Fin
```

Si (Condition) Alors … Sinon

Bloc 1		Bloc 2

Si Condition Alors
  Instructions (1)
Sinon
  Instructions (2)
FinSi
	Le bloc 1 est équivalent au bloc 2	Si **Non**(Condition) Alors
		  Instructions (2)
		Sinon
		  Instructions (1)
		FinSi

Bloc 1		Bloc 2

Si note>=10 Alors
  Ecrire("Note acceptable")
Sinon
  Ecrire(" Mauvaise Note")
FinSi
	Le bloc 1 est équivalent au bloc 2	Si note< 10 Alors
		  Ecrire("Mauvaise Note")
		Sinon
		  Ecrire(" Note acceptable")
		FinSi

Condition composée

- Ecrire un algorithme qui teste si une note saisie au clavier est comprise entre 0 et 20
- **Algorithme** TestNote
  - // Déclaration des variables
  - Variable Note: réel
  - Message : chaîne
  - **Début**
  - Ecrire ('Entrez la note:')
  - Lire (Note)
  - **Si** Note>=0 et Note<= 20 Alors
  - Message ← 'la note' & Note & 'est correcte'
  - **Sinon**
  - Message ← 'la note' & Note & 'est incorrecte'
  - **FinSi**
  - Ecrire (Message)
  - **Fin**

Exercice (imbrication des conditions)

- Ecrire un algorithme qui demande deux nombres m et n à l'utilisateur et l'informe ensuite si le produit est négatif ou positif. On inclut dans l'algorithme le cas où le produit peut être nul.

```
Algorithme Test_2_Produits
    // Déclaration des variables
    Variable m, n: entier
    Début
                Ecrire ('Entrez deux nombres m et n:')
                Lire (m, n)
                Si m=0 ou n= 0 Alors
                    Ecrire ('Le produit est nul')
                Sinon
                    Si (m<0 et n<0) ou (m>0 et n>0) Alors
                        Ecrire ('Le produit est positif')
                    Sinon
                        Ecrire ('Le produit est négatif')
                    FinSi
                FinSi
    Fin
```

Structure Si … Alors … FinSi

- Cette instruction est utilisée si on veut exécuter une instruction seulement si une condition est vraie et ne rien faire si la condition est fausse

- Syntaxe :
  - **Si (** *condition* **) Alors**

    Instruction(s)
  - **FinSi**

Exercice (Si … Alors …. FinSi)

- Une grande surface accorde à ses clients, une réduction de 2% pour les montants d'achat supérieurs à 1500,00 dhs
- Ecrire un algorithme permettant de saisir le prix total HT (PTHT) et de calculer le montant TTC (PTTC) en prenant en compte la remise et la TVA=20%.

- **Algorithme** Calcul PTTC
  - Constante TVA=0,2
  - Variable PTHT, PTTC : réel
  - **Début**
  - Ecrire ('Entrez le prix total hors taxes:')
  - Lire (PTHT)
  - **Si** PTHT > 1500 Alors
  - PTHT ← (PTHT * 0,98)
  - **FinSi**
  - PTTC ← PTHT + (PTHT * TVA)
  - Ecrire ('Le prix TTC est :', PTTC)
  - **Fin**

Structure Si … Alors …SinonSi… FinSi

- L'instruction qui sera exécutée est l'instruction dont la **condition est vraie**. Si aucune condition n'a la valeur vraie, c'est l'instruction qui suit le **Sinon** qui sera exécutée.

- Syntaxe :

Si (*condition 1*) **Alors**
Instruction1
SinonSi (*condition 2*) **Alors**
Instruction2
SinonSi (*condition 3*) **Alors**
Instruction3
……
Sinon
Instructions (si aucune condition n'est vrai = par défaut)
FinSi

Exercice

- Ecrire un algorithme qui permet d'afficher le maximum parmi deux nombres saisis au clavier

- **Algorithme** Max
  - Variable A, B : réel
  - **Début**
  - Ecrire ('Entrez la valeur de A:')
  - Lire (A)
  - Ecrire ('Entrez la valeur de B:')
  - Lire (B)
  - **Si** A > B alors
  - Ecrire('Le maximum est ', A)
  - **SinonSi** A= B alors
  - Ecrire('égalité')
  - **Sinon**
  - Ecrire('Le maximum est ', B)
  - **FinSi**
  - **Fin**

Exercice (allocations familiales)

- Ecrire un algorithme permettant de calculer le montant des allocations familiales sachant que le montant dépend du nombre d'enfant;
- Si le nombre d'enfants est inférieur ou égale à trois alors les allocations sont 150 dhs par enfant
- Si le nombre d'enfants est strictement supérieur à trois et inférieur ou égale à six alors les allocations sont de:
  - 150 dhs par enfant pour les 3 premiers enfants
  - 38 dhs par enfants pour les suivants
- Si le nombre d'enfants est strictement supérieur à six alors les allocations sont de :
  - 150 dhs par enfant pour les 3 premiers enfants
  - 38 dhs par enfant pour les suivants
  - 0 dhs par enfant pour les suivants

Structure à choix multiples

- Cette structure conditionnelle permet de choisir le traitement à effectuer en fonction de la valeur ou de l'intervalle de valeurs d'une variable ou d'une expression.
- Syntaxe :

Selon *sélecteur* **faire**

Valeur 1 : action(s) 1

Valeur 2 : action(s) 2

Valeur 3 : action(s) 3

…..

Valeur n: action(s) n

Sinon

action(s)

FinSelon

Exemple

- Ecrire un algorithme qui permet d'afficher le mois en toute lettre selon son numéro saisi au clavier

```
Algorithme Mois
    Variable N : Entier
    Début
        Ecrire ('Donner le numéro du mois')
        Lire (N)
        Selon N faire
                1:  Ecrire ('Janvier ')
                2:  Ecrire ('Février')
                3:  Ecrire ('Mars')
                4:  Ecrire ('Avril ')
                5:  Ecrire ('Mai ')
                6:  Ecrire ('Juin ')
                7:  Ecrire ('Juillet ')
                8:  Ecrire ('Aout ')
                9:  Ecrire ('Septembre ')
                10:  Ecrire ('Octobre')
                11:  Ecrire ('Novembre')
                12:  Ecrire ('Décembre')
        Sinon
                Ecrire ('Le numéro saisi est incorrecte ')
        FinSelon
    Fin
```

Structures répétitives

- Exemple:
- Saisi au clavier → question avec une réponse (Oui) ou (Non)
- Mettre en place un contrôle de saisie pour vérifier que les données entrées au clavier correspondent à la logique voulue

- **Algorithme** Contrôle_de_saisie
- Variable Rep : caractère
- **Début**
- Ecrire ('Voulez vous une copie de ce cours ? (O/N)')
- Lire(Rep)
- **Si** Rep **<>** 'O' et Rep **<>** 'N' alors
- Ecrire ('Erreur de saisie, recommencez')
- Lire(Rep)
- **FinSi**
- **Fin**

Boucle TantQue ….. Faire

- Permet de répéter un traitement tant que la condition est vraie
- Syntaxe :

  TantQue *Condition* **faire**

  Instruction(s)

  FinTantQue

- **Algorithme** Contrôle_de_saisie
- Variable Rep : caractère
- **Début**
- Ecrire (‘Voulez vous une copie de ce cours ? (O/N)’)
- Lire(Rep)
- **TantQue (**Rep **<>** ‘O’ et Rep **<>** ‘N’) **Faire**
- Ecrire (‘Erreur de saisie,’)
- Ecrire (‘Voulez vous une copie de ce cours ? (O/N)’)
- Lire(Rep)
- **FinTantQue**
- **Fin**

Boucle TantQue ..... Faire
(une boucle infinie)

- **Algorithme** Boucle_infinie
- Variable I: Entier
- **Début**
- I ← 3
- **TantQue (**I <= 10**) Faire**
- Ecrire (‘Bonjour’)
- **FinTantQue**
- **Fin**

Boucle **Pour …Jusqu'à….. Faire**

- Permet de répéter une liste d'instructions un nombre connu de fois
- Syntaxe :

Pour *Compteur* ← *valeur_initiale* **jusqu'à** *valeur_finale* **faire**

Instruction(s)

FinPour

- **Algorithme** multiplication_par_8

- Variable k: entier
- **Début**
- **Pour** k ← 0 **jusqu'à** 10 **Faire**
- Ecrire ('8*', k, '=', 8*k)
- Ecrire ('\')
- **FinPour**
- **Fin**

Exemple

- Ecrire un algorithme qui permet de calculer la somme des dix premiers nombres entiers

- Algorithme Somme
  - Variable S, I: Entier
  - **Début**
  - S ← 0
  - Pour i ← 1 jusqu'à 10 Faire
  - S ← S + i
  - FinPour
  - Ecrire ('La somme des dix premiers entiers est:', S)
  - **Fin**

Boucle **Pour** …**Jusqu'à**….. **Faire** (**Pas**)

- Permet de répéter une liste d'instructions un nombre connu de fois
- Syntaxe :

  Pour *Compteur* ← *VI* **jusqu'à** *VF* **pas** Valeur_Pas **Faire**
  Instruction(s)
  FinPour

- **Algorithme** multiplication_par_8

- Variable k: entier
- **Début**
- **Pour** k← 0 **jusqu'à** 10 **Pas** 1 **Faire**
- Ecrire ('8*', k, '=', 8*k)
- Ecrire ('\')
- **FinPour**
- **Fin**

Exemple

- Ecrire un algorithme qui permet de saisir un nombre entier et qui calcul la somme des entiers pairs jusqu'à ce nombre. Par exemple, si l'on saisi 10, le programme doit calculer: 0 + 2 +4 + 6 + 8 + 10 = 30

Algorithme Somme

- Variable S, I, N: Entier
- **Début**
- Ecrire('Entrer la valeur de N:')
- Lire (N)
- S ← 0
- **Pour** i← 1 **jusqu'à** N **Pas** 2 **Faire**
- S ← S + i
- **FinPour**
- Ecrire ('La somme des dix premiers entiers est:', S)
- **Fin**

Boucle **Répéter…Jusqu'à**

- Permet de répéter une liste d'instructions jusqu'à ce qu'une condition soit vraie
- Syntaxe :

Répéter

Instruction(s)

***Jusqu'à* condition**

Exemple

- En utilisant la boucle Répéter …. Jusqu'à, écrire un algorithme qui calcule la somme des N premiers nombres entiers. On suppose que N est strictement positif. Par exemple, Si N=6, le programme doit calculer : 1+2+3+4+5+6=21

- Algorithme Somme
  - Variable S, I, N: Entier
  - **Début**
  - Ecrire('Entrer la valeur de N (strictement positif):')
  - Lire (N)
  - S ← 0
  - i ← 1
  - Répéter
  - S ← S + i
  - **i ← i + 1**
  - Jusqu'à (i >= N)
  - Ecrire ('La somme des',N,' premiers entiers est:', S)
  - **Fin**

Remarque (boucles)

- Les boucles 'Répéter' et 'TantQue' sont utilisées lorsqu'on ne sait pas au départ combien de fois il faudra exécuter les instructions au sein de la boucle.

- On utilise la boucle 'Pour' quand l'on connait le nombre d'itérations à l'avance.

- La boucle 'Tant Que' est différente de la boucle 'Répéter' puisque cette dernière est exécutée au moins une fois

- La condition d'arrêt de la boucle 'Répéter' est la négation de la condition de poursuite de la boucle 'Tant Que'

Exercices

- Exercices (instructions alternatives et répétitives)

Introduction

- Ecrire un algorithme permettant de saisir cinq notes et de les afficher après avoir multiplié toutes les notes par trois

  - Algorithme Note
    - Variable N_1,N_2,N_3,N_4,N_5: Entier
    - **Début**
    - Ecrire('Entrer la valeur de la 1ière Note:')
    - Lire (N_1)
    - Ecrire('Entrer la valeur de la 2ième Note:')
    - Lire (N2)
    - Ecrire('Entrer la valeur de la 3ième Note:')
    - Lire (N_3)
    - Ecrire('Entrer la valeur de la 4ième Note:')
    - Lire (N_4)
    - Ecrire('Entrer la valeur de la 5ième Note:')
    - Lire (N_5)
    - Ecrire('La note 1 multiplié par 3 est:', N_1*3)
    - Ecrire('La note 1 multiplié par 3 est:', N_2*3)
    - Ecrire('La note 1 multiplié par 3 est:', N_3*3)
    - Ecrire('La note 1 multiplié par 3 est:', N_4*3)
    - Ecrire('La note 1 multiplié par 3 est:', N_5*3)
    - **FIN**

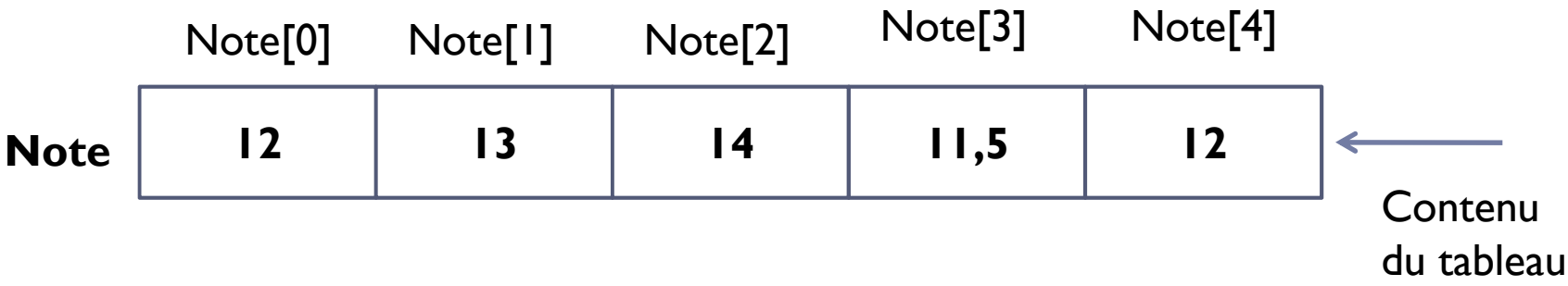
Plan du cours

- Eléments d'un algorithme
- Les structures alternatives et répétitives
- **Les tableaux**
- Les fonctions prédéfinis
- Les fichiers
- Les structures de données
  - Les piles/ Les files / les listes chainées
- Les algorithme de tri et de recherche

Définition

- Un Tableau est une suite d'éléments de même type. Il utilise plusieurs cases mémoire à l'aide d'un seul nom.

- Comme toutes les cases ont un seul nom, elles se différencient par un numéro ou un indice.



Déclaration

- La déclaration d'un tableau permet d'associer à un nom une zone mémoire composée d'un certain nombres de cases mémoires de même type.

- Syntaxe :

- ***Variable identificateur : tableau [indice_min .. Indice_max] de type***
- ***Ou***
- ***Variable identificateur: tableau[taille] de type***

- Notes:
  - Le premier élément d'un tableau porte l'indice zéro (par convention)
  - La valeur d'un entier doit être un nombre entier
  - La valeur d'un indice doit être inférieure ou égale au nombre d'éléments du tableau.
  - (Tab[20] → tab[21])

- Variable Note : Tableau[30] de réels
- Variable Note : Tableau[0 .. 29] de réels

Exemple

- Déclarer deux tableaux nommés A et B composé chacun d'eux de 10 éléments de type chaîne.

  - Variable A, B : tableau[0 .. 9] de chaîne

  - Lorsqu'on déclare un tableau, on déclare aussi de façon implicite toutes les variables indicées qui le constituent.

Utilisation

- Les éléments d'un tableau sont des variables indicées qui s'utilisent exactement comme n'importe quelles autres variables classiques.
  - Elles peuvent faire l'objet d'une affectation,
  - figurer dans une expression arithmétique,
  - dans une comparaison,
  - elles peuvent être affichées et saisies, etc
- Utilisation via le nom du tableau et l'indice (Note[i])

- Remarque: Il ne faut pas confondre l'indice d'un élément d'un tableau avec son contenu.
  - Par exemple, la 4^{ième} maison de la rue (tableau) n'a pas forcément 4 habitants.

Exemple

- L'instruction suivante affecte à la variable X la valeur du premier élément du tableau Note:
- X ← Note[0]

- L'instruction suivante affiche le 5^{ième} élément du tableau Note:
- Ecrire(Note[4])

- L'instruction suivante affecte une valeur introduite par l'utilisateur à l'élément trois du tableau Note:
- Lire(Note[2])

Parcours complet d'un tableau

- Ecrire un algorithme permettant de saisir 30 notes et de les afficher après avoir multiplié toutes ces notes par un coefficient fourni par l'utilisateur.

```
Algorithme Tableau_Note
    Variable Note : tableau[30] de réels
             Coeff, i : entier
    Début
        Ecrire('Entrer le coefficient:')
        Lire (Coeff)
        // Remplissage du tableau Note
        Pour i← 1 jusqu'à 30 faire
            Ecrire('Entrer la valeur de la Note:')
            Lire (Note[i-1])
        FinPour
        // Affichage des notes * Coeff
        Pour i← 1 jusqu'à 30 faire
            Ecrire(Note[i-1] * Coeff)
        FinPour
    FIN
```

Exercice

- Ecrire un algorithme qui calcul la somme des éléments d'un tableau de 100 réels.

```
Algorithme Somme_Tableau
    Constante M=100
    Variable T: tableau[M] de réels
              N i : Entier
              S : Réel
    Début
        Ecrire('Entrer la taille de tableau:')
        Lire (N)
        Si (N>M)
            N ← M
        FinSi
        // Remplissage du tableau
        Pour i← 1 jusqu'à N faire
            Ecrire('Entrer la valeur de l'élément ', i, 'du tableau')
            Lire (T[i-1])
        FinPour
        // Calcul de la somme des éléménts du tableau
        Pour i← 1 jusqu'à N faire
            S ← S + T[i-1]
        FinPour
        Ecrire ('La somme des éléments du tableau est :', S)
    FIN
```

Tableau à deux dimensions

- Un étudiant a plusieurs notes (une note pour chaque matière). Pour 4 étudiants nous aurons.

	Etudiant 1	Etudiant 2	Etudiant 3	Etudiant 4
Informatique	12	14	12,5	10
Comptabilité	9	10	8,5	9,5
Mathématiques	15	12	14	13

- Tableau à 3 lignes et de 4 colonnes

Déclaration

- Syntaxe :
- ***Variable identificateur : tableau [indice_min_lignes .. Indice_max_lignes, indice_min_colonnes .. Indice_max_colonnes] de type***
- ***Ou***
- ***Variable identificateur: tableau[nb_lignes, nb_colonnes] de type***
- Variable Note : Tableau[3,4] de réels
- Variable Note : Tableau[0 .. 2, 0..3] de réels

- Utilisation:
- L'instruction suivante affecte à la variable X la valeur du premier élément du tableau Note:
  - X ← Note[0,0]

Utilisation

- Pour accéder à un élément de la matrice (2 dimensions), il faut préciser, entre crochets, les indices de la case contenant cet élément.
- Exemple:
  - L'instruction suivante affecte la variable X à la valeur du premier élément du tableau Note: X ← Note[0,0]

Utilisation

- Exercice:
- Un algorithme permettant la saisie des notes d'une classe de 30 étudiants en 5 matières
  - Algorithme Notes
    - Constante N=30, M=5
    - Variable note: tableau[1..N, 1.. M] de réels
    - i, j : entier
    - **Début**
    - **//Remplissage du tableau note**
    - Pour i ← 1 jusqu'à N faire
    - Pour i ← 1 jusqu'à N faire
    - Ecrire('Entrer la note de l'étudiant', i, 'dans la matière', j)
    - Lire (note[i-1,j-1])
    - FinPour
    - FinPour
    - **Fin**

Tableau dynamique

- Les tableaux définis jusqu'ici sont statiques.
  - Il faut qu'au moment de l'écriture du programme, le programmeur décide de la taille maximale que pourra atteindre le tableau.
  - Dans plusieurs cas, on ne peut pas savoir la taille au départ.
- Les tableaux dynamiques sont des tableaux dont la taille n'est définie que lors de l'exécution
  - Affecter une taille vide au départ

Déclaration de tableau dynamique

- Syntaxe :
- ***Variable identificateur : tableau [] de type***
- Syntaxe :
- ***Redimensionner identificateur[N]***

- Variable t: Tableau[] d'entiers
- Redimensionner t[11]

Exercices

1. En utilisant les tableaux, écrire un algorithme qui permet la saisie d'une liste de n moyennes réelles et d'afficher le nombre des moyennes supérieures ou égales à 10. On suppose que n<=100.
2. Ecrire un algorithme qui permute les éléments d'un tableau de 8 éléments en plaçant le dernier éléments en premier, deuxièmes en avant dernier et ainsi de suite.
3. Ecrire un algorithme qui permet la saisie des moyennes d'une classe de n étudiants et affiche le maximum.

Plan du cours

- Eléments d'un algorithme
- Les structures alternatives et répétitives
- Les tableaux
- **Les procédures et les fonctions**
- Les fichiers
- Les structures de données
  - Les piles/ Les files / les listes chainées
- Les algorithme de tri et de recherche

Introduction

- Certains traitements sont complexes à effectuer par un algorithme simple.
  - Calcul du cosinus d'un angle → formule complexe
- Tous les langages de programmation ont un certain nombre de fonctions qui permettent de connaitre immédiatement le résultat d'une fonction.
- Ensemble de fonctions prédéfinies permettant :
  - Conversion de type de données
  - Calcul mathématiques
  - Manipulation des dates/heures
  - Manipulation de chaînes de caractères
  - …

Fonctions de chaînes de caractères

- Une chaîne est une séquence de caractère dont la longueur correspond au nombre de caractère qu'elle contient.
- Syntaxe longueur d'une chaîne:
  - Longueur(ch)
- Exemple longueur d'une chaîne:
  - Longueur('bonjour') → 7
- Syntaxe concaténation de chaîne:
  - Concat(ch1, ch2)
- Exemple de concaténation de chaîne:
  - Concat('bonjour', ' Monsieur') → 'Bonjour Monsieur'
- Syntaxe copie d'une chaîne:
  - Copie(ch, position, n)
- Exemple copie d'une chaîne:
  - Copie('bonjour', 4,4) → 'jour'
- Syntaxe de comparaison de chaînes:
  - Comp(ch1, ch 2)
- Syntaxe de recherche dans une chaîne:
  - Recherche(ch1, ch 2)

Procédures et fonctions

- Conception d' un algorithme → peut prendre une taille et une complexité croissante.
- En plus, des séquences d'instructions peuvent se répéter à plusieurs endroits.
- Découpages de l'algorithme en plusieurs petites parties
  - Plusieurs procédures ou fonctions

Réutilisabilité

- Dans la pratique, on ne souhaite pas recopier 2 fois le même code d'autant plus si celui-ci nécessite de très nombreuses lignes de code.
- Pour améliorer **la réutilisabilité** de l'algorithme autrement que par un « copier-coller », une solution sera l'encapsulation du code à répéter au sein d'une « fonction »

- **Réutilisabilité d'un algorithme**
  - La réutilisabilité d'un algorithme est son aptitude à être réutilisé pour résoudre des tâches équivalentes à celle pour laquelle il a été conçu.

Structuration du problème

- L'approche efficace d'un problème complexe consiste souvent à le décomposer en plusieurs sous-problèmes plus simples qui seront étudiés séparément.
  - Ces sous problème peuvent être éventuellement être eux-mêmes décomposés à leur tour
  - Le concepteur définira et utilisera ses propres « fonctions» pour réaliser la structuration d'un problème en sous problèmes .
- On divise le problème en sous-problèmes pour mieux le contrôler (diviser pour régner).

Exemple de structuration

- Un nombre x peut être représenté en base b par un triplet [s, m, p] tel que
  - $x = (-1)^s \cdot m \cdot b^p$
- où s représente le signe de x, m sa mantisse et p son exposant (p comme puissance) où :
- – signe s : s = 1 si x < 0 et s = 0 si $x \geq 0 \leq$
- – mantisse m : m dans [1, b[si $x \neq 0$ et m = 0 si x = 0
- – exposant p : p dans [min, max]

- Ainsi, le codage de x = −9.75 en base b = 2 s'effectuera en 4 étapes :

1. coder le signe de x : x = −9.75 < 0) s = 1
2. coder la partie entière de |x| : 9 = $(1001)_2$
3. coder la partie fractionnaire de |x| : 0.75 = $(0.11)_2$
4. et coder |x| en notation scientifique normalisée : m dans [1, 2[
   - $(1001)_2 + (0.11)_2 = (1001.11)_2 = (1.00111)_2 \cdot 2^3 = (1.00111)_2 \cdot 2^{(11)2}$
   - $x = (-1)^1 \cdot (1.00111)^2 \cdot 2^{(11)2}$ donc $s = (1)_2$, $m = (1.00111)_2$ et $p = (11)_2$.

Encapsulation

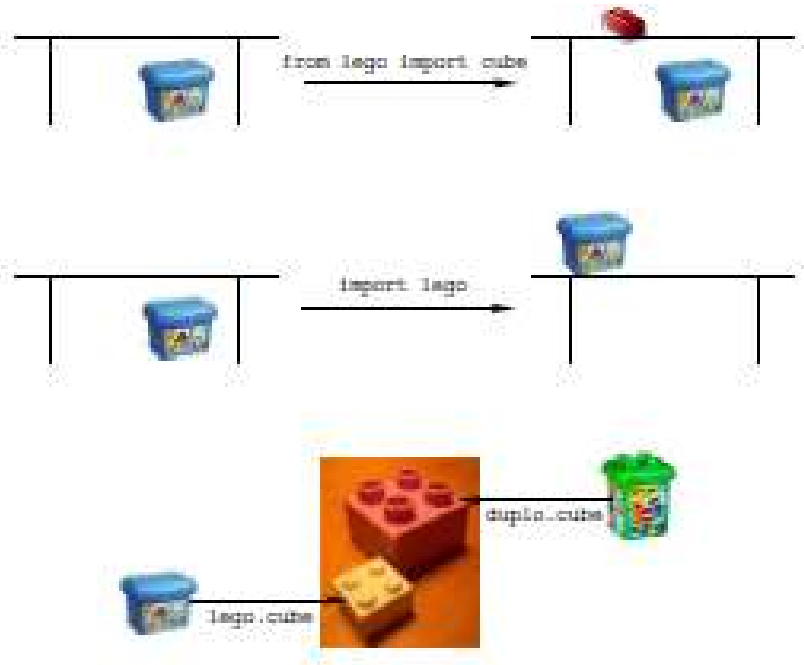
- L'encapsulation est l'action de mettre une chose dans une autre : on peut considérer que cette chose est mise dans une *« capsule » comme on conditionne* un médicament dans une enveloppe soluble (la capsule).
- Ici, il s'agit d'encapsuler des instructions dans une fonction ou une procédure

Fonctions et Procédures

- Une fonction est un bloc d'instructions nommé et paramétré, réalisant une certaine tâche. Elle admet zéro, un ou plusieurs paramètres et renvoie toujours un résultat.
- Une procédure est un bloc d'instructions nommé et paramétré, réalisant une certaine tâche. Elle admet zéro, un ou plusieurs paramètres et ne renvoie pas de résultat.

Procédures et fonctions intégrées

- Les procédures et les fonctions intégrées à un langage sont relativement peu nombreuses : ce sont seulement celles qui sont susceptibles d'être utilisées très fréquemment
- Les autres fonctions sont regroupées dans des fichiers séparés que l'on appelle des modules.
- Les modules sont des fichiers qui regroupent des ensembles de fonctions. Souvent on regroupe dans un même module des ensembles de fonctions apparentées que l'on appelle des bibliothèques.
- Pour pouvoir utiliser ces fonctions, il faut importer le module correspondant.



Procédure : déclaration

- Syntaxe :
- ***Procédure nom_procédure(liste de paramètres)***
- ***Variables identificateurs : type***
- ***Début***
- ***Instructions***
- ***FinProc***

- Liste de paramètres → paramètres formels (avec des valeurs inconnues lors de la création de la procédure)

Exemple procédure et appel

- Procédure Etoiles()
  - Variable i: Entier
  - **Début**
  - Pour i ← 1 jusqu'à 15 faire
  - Ecrire('*')
  - FinPour
  - Ecrire('\n')
  - **FinProc**

  - **Syntaxe** :
  - Nom_Proc(liste de paramères)
  - Exemple: Etoiles()

Exemple d'utilisation de la procédure

- Algorithme carré_étoiles

  - Variable j: Entier

  - // Déclaration de la procédure Etoiles()
  - Procédure Etoiles()
  - …..
  - FinProc

  - //Algorithme Principale
  - **Début**
  - Pour j ← 1 jusqu'à 15 faire
  - //Appel de la procédure Etoiles
  - **Etoiles()**
  - FinPour
  - **Fin**

  - L'algorithme commence l' exécution de sa partie principale
  - Dans un algorithme avec des procédures
    - La définition de la procédure doit apparaître avant la partie principale
    - L'appel à la procédure (ou la fonction) doit se faire au sein de l'algorithme principale ou de procédures ou fonctions déclarées après.

Paramétrage

- Un algorithme est une suite ordonnée d'instructions qui indique la démarche à suivre pour résoudre une série de problèmes équivalents.
- Dans ce contexte, c'est le paramétrage de l'algorithme qui lui donnera cette capacité recherchée de résoudre des problèmes équivalents

- Paramètre d'entrée
  - Les paramètres d'entrée d'une fonction sont les arguments de la fonction qui sont nécessaires pour effectuer le traitement associé `a la fonction.
- Paramètre de sortie
  - Les paramètres de sortie d'une fonction sont les résultats retournés par la fonction après avoir effectué le traitement associé à la fonction.

Passage de paramètres

- Les échanges d'informations entre une procédure et le sous algorithme appelant se font par l'intermédiaire de paramètres.

- Dans la définition d'une fonction, la liste des paramètres d'entrée spécifie les informations à transmettre comme arguments lors de l'appel de la fonction.
  - Les arguments effectivement utilisés doivent être fournis dans le même ordre que celui des paramètres d'entrée correspondants : le premier argument sera affecté au premier paramètre d'entrée, le second argument sera affecté au second paramètre d'entrée, et ainsi de suite.

Définition	**L'appel**
Procédure Etoiles(n: entier) ….. FinProc	x ← 5 Etoiles(x)

- Les paramètres introduits dans la définition sont appelés les paramètres formels (par exemple n dans la définition de la procédure Etoile) ;
- Les paramètres passés en arguments sont appelés les paramètres effectifs (par exemple x dans l'appel Etoile(x).)

Passage de paramètres

- Il existe deux types de passages de paramètres qui permettent des usages différents :

- **Passage par valeur**: Le paramètre formel reçoit uniquement une copie de la valeur du paramètre effectif. La valeur du paramètre effectif ne sera jamais modifiée.

- **Passage par référence ou par adresse**: La procédure utilise l'adresse du paramètre effectif. Lorsqu'on utilise l'adresse du paramètre, on accède directement à son contenu. La valeur de la variable effectif sera donc modifiée.

Exemple de passage par valeur

- Algorithme passage_par_valeur

  - Variable N: Entier

  - // Déclaration de la procédure P1
  - Procédure P1(A : entier)
  - Début
  - A ← A*2
  - Ecrire(A)
  - FinProc

  - //Algorithme Principale
  - **Début**
  - N← 5
  - P1(N)
  - Ecrire(N)
  - **Fin**

Exemple de passage par référence ou par adresse

- Algorithme passage_par_référence

  - Variable N: Entier

  - // Déclaration de la procédure P1
  - Procédure P1(**VAR** A : entier)
  - Début
  - A ← A*2
  - Ecrire(A)
  - FinProc

  - //Algorithme Principale
  - **Début**
  - N← 5
  - P1(N)
  - Ecrire(N)
  - **Fin**

Fonction: déclaration

- Les fonctions sont des sous algorithmes admettant des paramètres et retournant (une seule valeur) de type simple qui peut apparaître dans une expression, dans une comparaison, à la droite d'une affectation, etc.

- Syntaxe :
- ***Fonction nom_Fonction(liste de paramètres) : type***
- ***Variables identificateurs : type***
- ***Début***
- ***Instructions***
- ***Retourner Expression (type)***
- ***FinProc***

Exemple de fonction Max

```
// Déclaration de la fonction Max
Fonction Max(X: réel, Y: réel) : réel
Début
    Si (X > Y) alors
        Retourner X
    Sinon
        Retourner Y
    FinSi
FinFonction
```

Appel d'une fonction

- A la différence d'une procédure, la fonction retourne une valeur. L'appel d'une fonction pourra donc être utilisé dans une instruction.
- Syntaxe :
- ***Nom_Fonction*(*liste de paramètres*)**

```
Algorithme Appel_Fonction_Max/
Variables A,B,M: réel
// Déclaration de la fonction Max
Fonction Max(X: réel,Y: réel) : réel
..
FinFonction

// Algorithme principale
Début
     Ecrire( 'Donnez la valeur de A')
     Lire(A)
     Ecrire( 'Donnez la valeur de B')
    Lire(B)
     M ← Max(A,B) // appel à la fonction Max
     Ecrire(' Le plus grand de ces deux nombres est: ', M)
FIN
```

Portée des variables

- La portée d'une variable désigne le domaine de visibilité de cette variable.
  - Une variable peut être déclarée dans deux emplacements distincts.
- Une variable déclarée dans la partie déclaration de l'algorithme principal est appelée **variable globale**.
  - Elle est accessible de n'importe où dans l'algorithme, même depuis les procédures et les fonctions?
  - Elle existe pendant toute la durée de vie du programme.
- Une variable déclarée à l'intérieur d'une procédure (ou une fonction) est dite **variable locale**.
  - Elle n'est accessible qu'au sein de la procédure où elle est déclarée.
  - La durée de vie d'une variable locale est limitée à la durée d'exécution de la procédure.

Exemple de la portée d'une variable

- ***Algorithme Algorithme Portée***
- ***Variables X, Y: Entier***

- Procédure P1()
- Variable A: entier
- Début
- ……
- FinProc

A une variable locale
Visible uniquement à
L'intérieur de la procédure

- // Algorithme principale
- Début
- …..
- FIN

X et Y sont des
Variables globales,
Visibles dans tout
L'algorithme

Les variables globales
Sont à éviter pour la
Maintenance des
programmes

La récursivité

- Une procédure (ou une fonction) est dite récursive si elle s'appelle elle-même
- Exemple :Fonction factorielle

```
Fonction Fact(n: entier) : entier
Début
   Si n> 1 alors
      Retourner (fact(n-1)*n)
   Sinon
      Retourner 1
   Finsi
FinFonction
```

- Toute procédure ou fonction récursive comporte une instruction (ou un bloc d'instructions) nommée « point terminal » permettant de sortir de la procédure ou de la fonction.

Avantages des procédures et des fonctions

- Les procédures ou fonctions permettent de ne pas répéter plusieurs fois une même séquence d'instructions au sein du programme (algorithme)
- La mise au point du programme est plus rapide en utilisant des procédures et des fonctions. En effet, elle peut être réalisée en dehors du contexte du programme.
- Une procédure peut être intégrée à un autre programme, ou elle peut être rangée dans une bibliothèque d'outils pour être utilisée par n'importe quel programme.
- **Mieux diviser pour régner.**

Plan du cours

- Eléments d'un algorithme
- Les structures alternatives et répétitives
- Les tableaux
- Les fonctions prédéfinis
- Les fichiers
- Les structures de données
  - Les piles/ Les files / les listes chainées
- **Les algorithme de tri et de recherche**

Introduction

- Le traitement de données pose le problème de leur classement **(tri).**
  - Différentes méthodes ont été développées
- Un algorithme de tri sert à ordonner un ensemble d'éléments.
  - Facilite l'accès à l'information et la recherche d'un élément.
- Les algorithmes présentent des performances différentes
  - En termes de temps d'exécution
  - En termes d'espace mémoire

Tri par échange (Algorithme de tri)

- Algorithme peu performant
- Très simple à comprendre

- Principe:
  - Prendre chaque élément d'un tableau et le comparer à tous ses suivants, lorsque l'ordre n'est pas respecté les deux éléments sont permutés.

Tri par échange

- Algorithme Tri par échange
  - Variable T tableau[1..N] d'entiers
  - i, j, N, échange : entier
  - **Début**
  - **Pour** i ← 0 **jusqu'à** N-2 **faire**
  - **Pour** j ← i+1 **jusqu'à** N-1 **faire**
  - **Si** (T[i]>T[j]) **alors**
  - échange ← T[i]
  - T[i] ← T[j]
  - T[j] ← échange
  - **FinSi**
  - **FinPour**
  - **FinPour**
  - **Fin**
  -

Simulation tri par échange

	T[0]	T[1]	T[2]	T[3]	T[4]	T[5]
	7	4	1	2	9	5
	1ière itération : i=0					
j=1	4	7	1	2	9	5
j=2	1	7	4	2	9	5
j=3	1	7	4	2	9	5
j=4	1	7	4	2	9	5
j=5	1	7	4	2	9	5
	itération : i=1					
j=2	1	4	7	2	9	5
j=3	1	2	7	4	9	5
j=4	1	2	7	4	9	5
j=5	1	2	7	4	9	5
	itération : i=2					
j=3	1	2	4	7	9	5
j=4	1	2	4	7	9	5
j=5	1	2	4	7	9	5
	itération : i=3					
j=4	1	2	4	7	9	5
j=5	1	2	4	5	9	7
	itération : i=4					
j=5	1	2	4	5	7	9

Tri à bulle (Bubble Sort)

- Principe:
  - Comparer successivement les éléments adjacents, et les permuter si le premier élément est supérieur au second. L'opération est relancée jusqu'à ce qu'il n'y ait plus de permutation possible (tous les nombres sont triés)

Tri à bulle (Bubble Sort)

- Algorithme Tri à bulle
  - Variable T tableau[1..N] d'entiers
  - i, N, échange : entier
  - Permute : Booleen
  - **Début**
  - **Répéter**
  - Permute ← **Faux**
  - **Pour** i ← 0 **jusqu'à** N-2 **faire**
  - **Si** (T[i]>T[i+1]) **alors**
  - // permuter les deux éléments
  - échange ← T[i]
  - T[i] ← T[i+1]
  - T[i+1] ← échange
  - Permute ← Vrai
  - **FinSi**
  - **FinPour**
  - **Jusqu'à (**Permute **= Faux)**
  - **Fin**

Simulation tri à bulle

	T[0]	T[1]	T[2]	T[3]	T[4]	T[5]	
	7	4	1	2	9	5	
i=0	7	4	1	2	9	5	Permute =Vrai
i=1	4	7	1	2	9	5	
i=2	4	1	7	2	9	5	
i=3	4	1	2	7	9	5	
i=4	4	1	2	7	9	5	
	4	1	2	7	5	9	Permute =Vrai
Permute =Faux							
i=0	4	1	2	7	5	9	Permute =Vrai
i=1	1	4	2	7	5	9	
i=2	1	2	4	7	5	9	
i=3	1	2	4	7	5	9	Permute =Vrai
i=4	1	2	4	5	7	9	
	1	2	4	5	7	9	Permute =Vrai
Permute =Faux							
i=0	1	2	4	5	7	9	Permute =Faux
i=1	1	2	4	5	7	9	Permute =Faux
i=2	1	2	4	5	7	9	Permute =Faux
i=3	1	2	4	5	7	9	Permute =Faux
i=4	1	2	4	5	7	9	Permute =Faux

Tri par insertion

- Le tri par insertion est un tri naturel
  - Tri utilisé de façon intuitive
- Principe:
  - On considère que le tableau est divisé en deux parties de tailles variables; une dans laquelle les données sont triées et l'autre contenant les données à trier.
  - Initialement, la partie trié du tableau contient uniquement le premier élément. La partie non triée contient les autres éléments (le reste du tableau).
  - On prend un à un les éléments de la partie non triée du tableau et on **l'insère** en bon endroit dans la partie triée du tableau en déplaçant tous les éléments qui le précède.

Tri par insertion

- Algorithme Tri par insertion
  - Variable T tableau[1..N] d'entiers
  - i, j ,N, échange : entier
  - **Début**
  - **Pour** i ← 1 **jusqu'à** N-1 **Faire**
  - Echange ← T[i]
  - j ← i -1
  - **TantQue** (j >=0) **ET** (échange< T[j]) **Faire**
  - T[j+1] ← T[j]
  - j ← j - 1
  - **FinTantQue**
  - T[j+1]← échange
  - **FinPour**
  - **Fin**

Simulation tri par insertion

T[0]	T[1]	T[2]	T[3]	T[4]	T[5]
7	4	1	2	9	5

						Echange
7	4	1	2	9	5	4
4	7	1	2	9	5	1
1	4	7	2	9	5	2
1	2	4	7	9	5	9
1	2	4	7	9	5	5
1	2	4	5	7	9	