Algorithmique & Programmation

Fahd Karami

fa.karami@uca.ma

Objectif et plan du cours

- **Objectif:**
  - Apprendre les concepts de base de l'algorithmique et de la programmation
  - Etre capable de mettre en œuvre ces concepts pour analyser des problèmes simples et écrire les programmes correspondants
- **Plan:**

I- Généralités (matériel d'un ordinateur, systèmes d'exploitation, langages de programmation, …)

II- **Algorithmique** (affectation, instructions conditionnelles, instructions itératives, fonctions, procédures, …)

III- **Langage C** (un outil de programmation)

I-Généralités

Système Informatique?

Nous ne pouvons pas afficher cette image pour l'instant.

- Ensemble de Techniques du traitement **automatique** de l'**information** au moyen des ordinateurs

  Système informatique = ordinateur + périphériques

- Éléments d'un système informatique

Qu'est-ce qu'un programme d'ordinateur?

- Allumez un ordinateur, vous n'en tirerez rien!!
- Pour le faire marcher il faut lui fournir un programme

  Ordinateur = matériel + programme(s)
- Un programme est une suite d'instructions d'ordinateur
- Une instruction est un **ordre** compris par l'ordinateur et qui lui fait exécuté une **action**, c-à-d une modification de son environnement

Les catégories d'ordres

- Les ordinateurs, quels qu'ils soient, ne sont fondamentalement capables de comprendre que quatre catégories d'ordres (en programmation, on n'emploiera pas le terme d'ordre, mais plutôt celui d'instructions). Ces quatre familles d'instructions sont :
  - Les variables et leurs affectation
  - la lecture / écriture
  - les tests
  - les boucles

Actions d'un ordinateur : Exemple

- Attendre qu'un nombre soit tapé au clavier
- Sortir à l'écran le nombre entré
- Attendre qu'un nombre soit tapé au clavier
- Sortir à l'écran le nombre entré
- Additionner les deux nombres entrés
- Sortir à l'écran le résultat de l'addition

Ces lignes forment un programme d'ordinateur

Langages informatiques

- Un langage informatique est un outil permettant de donner des ordres (**instructions**) à la machine
  - A chaque instruction correspond une action du processeur
- Intérêt : écrire des **programmes** (suite consécutive d'instructions) destinés à effectuer une tache donnée
  - Exemple: un programme de gestion de comptes bancaires
- Contrainte: être compréhensible par la machine

Langage machine

- Langage **binaire**: l'information est exprimée et manipulée sous forme d'une suite de bits
- Un **bit** (*binary digit*) = 0 ou 1 (2 états électriques)
- Une combinaison de 8 bits= 1 **Octet** → $2^8=256$ possibilités qui permettent de coder tous les caractères alphabétiques, numériques, et symboles tels que ?,*,&,…
  - Le code **ASCII** (*American Standard Code for Information Interchange*) donne les correspondances entre les caractères alphanumériques et leurs représentation binaire, Ex. A= 01000001, ?=00111111
- Les opérations logiques et arithmétiques de base (addition, multiplication,…) sont effectuées en binaire

Codage binaire

- Le langage des ordinateurs

- Toutes communications à l'intérieur de l'ordinateur sont faites avec des signaux électriques
  - 0: éteint (absence de signal électrique)
  - 1: allumé (présence de signal électrique)

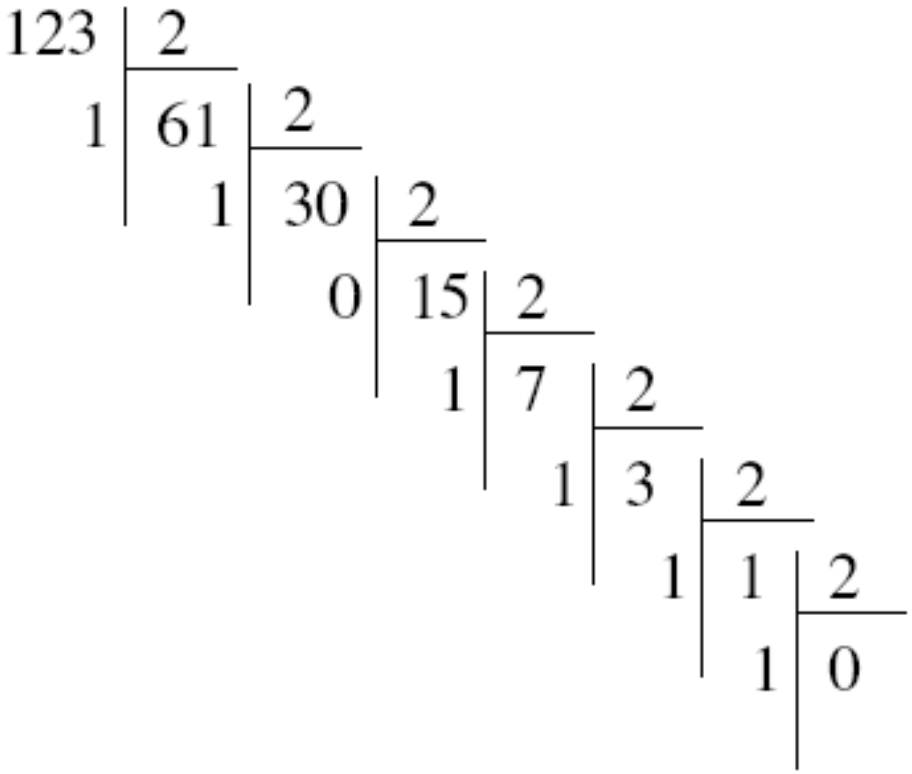
- Un même nombre peut être représenté dans plusieurs bases
  - 123 en base 10 (décimal)
  - 1111011 en base 2 (binaire)
  - 173 en base 8 (octale)
  - 7B en base 16 (hexadécimale)

- De la base 10 à la base 2
  - Il faut diviser le nombre par 2 puis réitérer l'opération en considérant que le nouveau numérateur est l'ancien quotient jusqu'à ce que ce dernier soit nul. La suite inverse des restes représente le nombre binaire

Par exemple 1111011 vaut 123 en base 10 car :

$$1*2^6+1*2^5+1*2^4+1*2^3+0*2^2+1*2^1+1*2^0=123$$

Exemple



- **Ecrire 2014 en base 2?**

- De la base 2 à la base 10
  - Il faut additionner la multiplication du nombre représenté par chaque chiffre avec la puissance de 2 correspondant au rang du chiffre:
- Les opérations élémentaires en base 10 s'appliquent de la même façon en base 2
  - Exemple: Addition, soustraction, multiplication, division

Transcodage binaire/hexadécimal

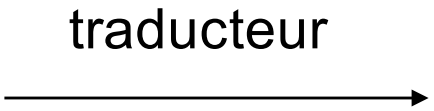
- Un autre système, l'hexadécimal (base 16), est très souvent employé en informatique
  - facilite la représentation des longues séquences de bits
  - représentation :
    - 0 1 2 3 4 5 6 7 8 9 A B C D E F
    - 1011011000010000000110001101001 1 (binaire)
    - 2d8818d3 (hexadécimale)

Langages de Programmation

L'assembleur

- Problème: le langage machine est difficile à comprendre par l'humain
- Idée: trouver un langage compréhensible par l'homme qui sera ensuite converti en langage machine
  - **Assembleur** (1er langage): exprimer les instructions élémentaires de façon symbolique

ADD A, 4
LOAD B
MOV A, OUT



langage machine

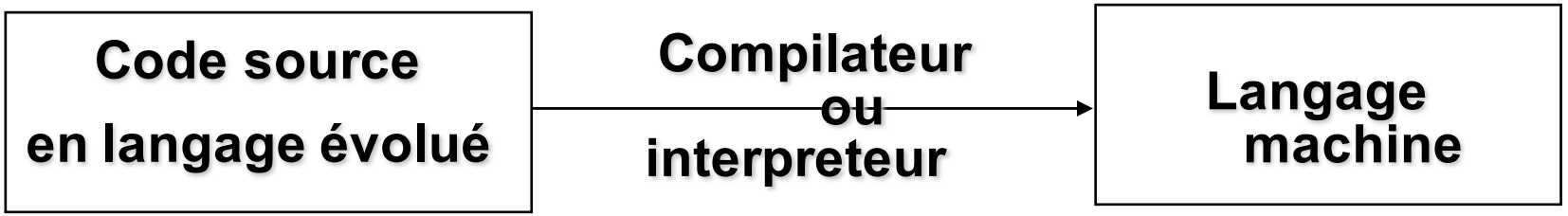
  - +: déjà plus accessible que le langage machine
  - -: dépend du type de la machine (n'est pas ***portable***)
  - -: pas assez efficace pour développer des applications complexes

Apparition des langages évolués

$\Rightarrow$

Langages haut niveau

- Intérêts multiples pour le haut niveau:
  - Proche du langage humain «anglais» (compréhensible)
  - Permet une plus grande portabilité (indépendant du matériel)
  - Manipulation de données et d'expressions complexes (réels, objets, a*b/c, …)
- Nécessité d'un traducteur (compilateur/interpréteur), exécution plus ou moins lente selon le traducteur



Compilateur/interpréteur

- Compilateur: traduire le programme entier une fois pour toutes

  - + plus rapide à l'exécution
  - + sécurité du code source
  - - il faut recompiler à chaque modification

- Interpréteur: traduire au fur et à mesure les instructions du programme à chaque exécution

  - + exécution instantanée appréciable pour les débutants
  - - exécution lente par rapport à la compilation

Langages de programmation:

- Deux types de langages:
  - Langages procéduraux : sont à base de procédures. Une procédure est une portion de programme écrit en langage de haut niveau qui accomplit une tâche spécifique nécessaire au programme.
  - Langages orientés objets : sont des langages non procéduraux dans lesquels les éléments du programme sont considérés comme des objets qui peuvent s'échanger des messages.

- Choix d'un langage?

Principaux Langages de programmation:

Pascal	Blaise PASCAL, mathématicien et inventeur de la première machine à calculer 1971	Langage compilé et structuré, dérivé d'ALGOL. c'est un langage de développement standard pour les micro-ordinateurs.
C	C'est une version améliorée du langage de programmation B du Bell Laboratory, créé en 1972	Langage de programmation structuré et compilé, très largement employé car ses programmes peuvent facilement se transférer d'un type d'ordinateur à un autre.
Maple	Nasa 1980 de SUN	couvrir tous les domaines D'application formel
Java	Microsystems 1990	Ce langage connaît un succès qu'aucun autre langage n'avait encore connu.

Etapes de réalisation d'un programme

Enoncé du problème

↓ *Spécification*

Cahier des charges

↓ *Analyse*

Algorithme

↓ *Traduction en langage*

Programme source

↓ *Compilation*

Programme exécutable

↓ *Tests et modifications*

Version finale et résultats

La réalisation de programmes passe par l'écriture d'algorithmes

$\Rightarrow$ D'où l'intérêt de l'**Algorithmique**

Pourquoi apprendre l'algorithmique pour apprendre à programmer ?

- Un algorithme est une description complète et détaillée des actions à effectuer et de leur séquencement pour arriver à un résultat donné
  - Intérêt: séparation analyse/codage (pas de préoccupation de syntaxe) l'algorithmique exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage.
  - Qualités: **exact** (fournit le résultat souhaité), **efficace** (temps d'exécution, mémoire occupée), **clair** (compréhensible), **général** (traite le plus grand nombre de cas possibles), …
- Pour prendre une image, si un programme était une dissertation, l'algorithmique serait le plan, une fois mis de côté la rédaction et l'orthographe. Mieux faire d'abord le plan et rédiger ensuite que l'inverse…

Représentation d'un algorithme

Historiquement, deux façons pour représenter un algorithme:

- **L'Organigramme:** représentation graphique avec des symboles (carrés, losanges, etc.)
  - offre une vue d'ensemble de l'algorithme
  - représentation quasiment abandonnée aujourd'hui

- **Le pseudo-code:** représentation textuelle avec une série de conventions ressemblant à un langage de programmation (sans les problèmes de syntaxe)
  - plus pratique pour écrire un algorithme
  - représentation largement utilisée

Exemple de pseudo code

- Problème du tri
  - **Entrée:** une séquence de *n* nombres (*a*1, : : : ,*an*)
  - **Sortie:** une permutation (*a*1';: : : ;*an'*) de la séquence d'entrée: a1'<a2'<….<an'
  - Exemple : (31;41;59;26;41;58) → (26;31;41;41;58;59)

Décrire une manière de résoudre ce problème?

Exemple de pseudo code

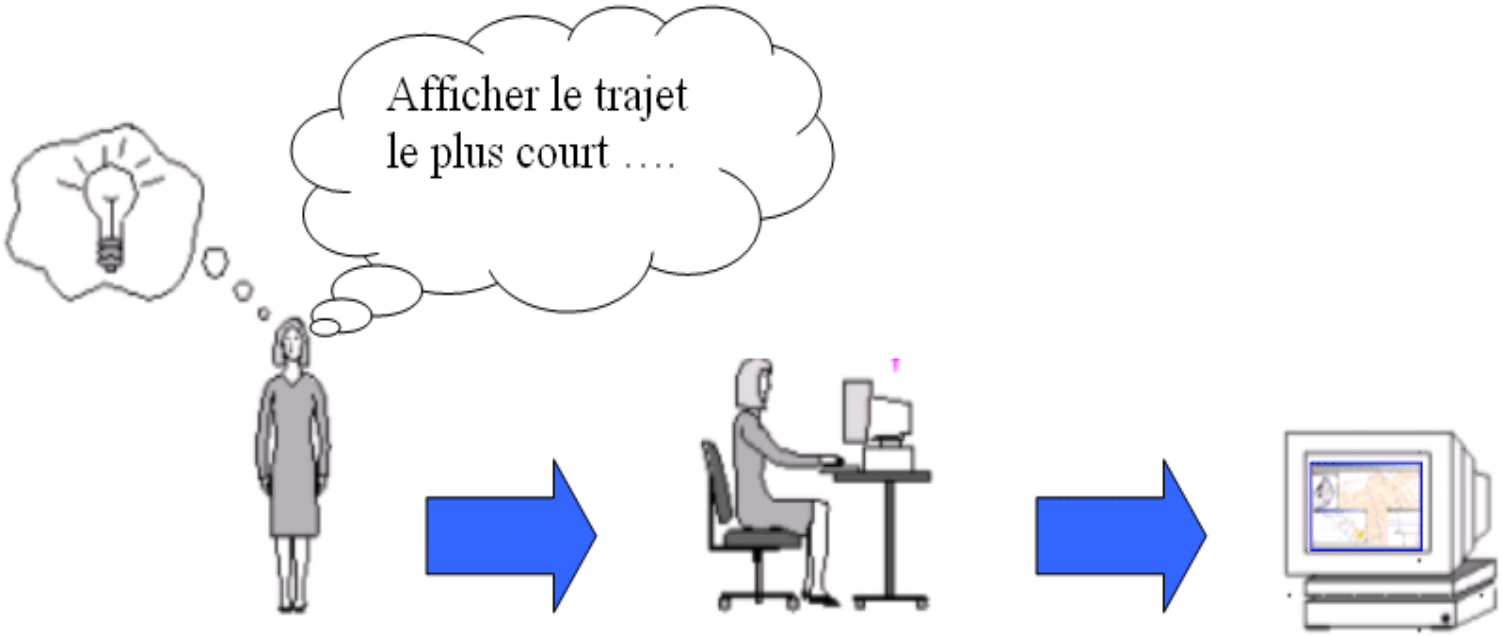
- Problème du tri
  - **Entrée:** une séquence de *n* nombres (*a*1, : : : ,*an*)
  - **Sortie:** une permutation (*a*1’; : : : ;*an’*) de la séquence d’entrée: a1’<a2’<….<an’
  - Exemple : (31;41;59;26;41;58) → (26;31;41;41;58;59)

```
TRI-INSERTION(A)
1  pour j ← 2 à A.longeur faire
2      clé ← A[j]
3      ▷ Insertion de A[j] dans la séquence triée A[1..j − 1].
4      i ← j − 1
5      tant que i > 0 et A[i] > clé faire
6          A[i + 1] ← A[i]
7          i ← i − 1
8      A[i + 1] ← clé
```

Algorithmique

Notions et instructions de base

Identifier le rôle d'un algorithme



Les catégories d'ordres

- les ordinateurs, quels qu'ils soient, ne sont fondamentalement capables de comprendre que quatre catégories d'ordres (en programmation, on n'emploiera pas le terme d'ordre, mais plutôt celui d'instructions). Ces quatre familles d'instructions sont :
  - Les variables et leurs affectation
  - la lecture / écriture
  - les tests
  - les boucles

Notion de variable

- Dans les langages de programmation une **variable** sert à stocker la valeur d'une donnée

- Une variable désigne en fait un emplacement mémoire dont le contenu peut changer au cours d'un programme (d'où le nom variable)

- Règle : Les variables doivent être **déclarées** avant d'être utilisées, elle doivent être caractérisées par :
  - un nom (**Identificateur**)
  - un **type** (entier, réel, caractère, chaîne de caractères, …)

Choix des identificateurs (1)

Le choix des noms de variables est soumis à quelques règles qui varient selon le langage, mais en général:

- Un nom doit commencer par une lettre alphabétique
  exemple valide: A1 **exemple invalide: 1A**
- Doit être constitué uniquement de lettres, de chiffres et du soulignement _ (Eviter les caractères de ponctuation et les espaces)
  valides: SMI2007, SMI_2007 **invalides: SMI 2007, SMI-2007, SMI;2007**
- doit être différent des mots réservés du langage (par exemple en Java: **int, float, else, switch, case, default, for, main, return**, …)
- La longueur du nom doit être inférieure à la taille maximale spécifiée par le langage utilisé

Choix des identificateurs (2)

Conseil: pour la lisibilité du code choisir des noms significatifs qui décrivent les données manipulées

exemples: TotalVentes2006, Prix_TTC, Prix_HT

Remarque: en pseudo-code algorithmique, on va respecter les règles citées, même si on est libre dans la syntaxe

Types des variables

Le type d'une variable détermine l'ensemble des valeurs qu'elle peut prendre, les types offerts par la plus part des langages sont:

- Type numérique (entier ou réel)
  - **Byte** (codé sur 1 octet): de 0 à 255
  - **Entier court** (codé sur 2 octets) : -32 768 à 32 767
  - **Entier long** (codé sur 4 ou 8 octets)
  - **Réel simple précision** (codé sur 4 octets)
  - **Réel double précision** (codé sur 8 octets)
- Type logique ou booléen: deux valeurs VRAI ou FAUX
- Type caractère: lettres majuscules, minuscules, chiffres, symboles, …

  exemples: 'A', 'a', '1', '?', …
- Type chaîne de caractère: toute suite de caractères, **exemples: " Nom, Prénom", "code postale: 1000", …**

Déclaration des variables

- Rappel: toute variable utilisée dans un programme doit avoir fait l'objet d'une déclaration préalable
- En pseudo-code, on va adopter la forme suivante pour la déclaration de variables

variables liste d'identificateurs : type

- Exemple:

Variables i, j,k : entier
x, y : réel
OK: booléen
ch1, ch2 : chaîne de caractères

- Remarque: pour le type numérique on va se limiter aux entiers et réels sans considérer les sous types

L'instruction d'affectation

- **l'affectation** consiste à attribuer une valeur à une variable (ça consiste en fait à remplir où à modifier le contenu d'une zone mémoire)
- En pseudo-code, l'affectation se note avec le signe ←

  Var← e : attribue la valeur de e à la variable Var

  - e peut être une valeur, une autre variable ou une expression
  - Var et e doivent être de même type ou de types compatibles
  - l'affectation ne modifie que ce qui est à gauche de la flèche

- **Ex valides:** **i ←1;** **j ←i;** **k ←i+j;**
  x ←10.3 ; **OK ←FAUX;** **ch1 ←"SMI";**
  ch2 ←ch1 ; **x ←4;** **x ←j;**

  (voir la déclaration des variables dans le transparent précédent)
- **non valides:** **i ←10.3;** **OK ←"SMI";** **j ←x;**

Quelques remarques

- Beaucoup de langages de programmation (C/C++, Java, …) utilisent le signe égal = pour l'affectation ←. Attention aux confusions:
  - L'affectation n'est pas commutative :A←B est différente de B←A
  - L' affectation est différente d'une équation mathématique :
    - A←A+1 a un sens en langages de programmation
    - A+1←2 n'est pas possible en langages de programmation et n'est pas équivalente à A←1
- Certains langages donnent des valeurs par défaut aux variables déclarées. Pour éviter tout problème il est préférable **d'initialiser les variables** déclarées

Exercices simples sur l'affectation (1)

Donnez les valeurs des variables A, B et C après exécution des instructions suivantes ?

Algorithme Test_Var

Variables A, B, C: **Entiers**

Début

A ← 3;

B ← 7;

A ← B;

B ← A+5;

C ← A + B;

C ← B – A;

Fin

Exercices simples sur l'affectation (2)

Donnez les valeurs des variables A et B après exécution des instructions suivantes ?

Algorithme Test_Permut

Variables A, B : **Entier**
Début
A ← 1;
B ← 2;
A ← B;
B ← A;
Fin

Les deux dernières instructions permettent-elles d'échanger les valeurs de A et B ?

Exercices simples sur l'affectation (3)

Ecrire un algorithme permettant d'échanger les valeurs de deux variables A et B?

Expressions et opérateurs

- Une **expression** peut être une valeur, une variable ou une opération constituée de variables reliées par des **opérateurs**

  exemples: 1, b, a*2, a+ 3*b-c, …

- L'évaluation de l'expression fournit une valeur unique qui est le résultat de l'opération

- Les **opérateurs** dépendent du type de l'opération, ils peuvent être :

  - **des opérateurs arithmétiques: +, -, *, /, % (modulo), ^ (puissance)**
  - **des opérateurs logiques: NON, OU, ET**
  - **des opérateurs relationnels: =, , <, >, <=, >=**
  - **des opérateurs sur les chaînes: & (concaténation)**

- Une expression est évaluée de gauche à droite mais en tenant compte de **priorités** ≠

Priorité des opérateurs

- Pour les opérateurs arithmétiques donnés ci-dessus, l'ordre de priorité est le suivant (du plus prioritaire au moins prioritaire) :
  - ^ : (élévation à la puissance)
  - * , / (multiplication, division)
  - % (modulo)
  - + , - (addition, soustraction)

  exemple: 2 + 3 * 7 vaut 23

- En cas de besoin (ou de doute), on utilise les parenthèses pour indiquer les opérations à effectuer en priorité
- **exemple: (2 + 3) * 7 vaut 35**

Les instructions d'entrées-sorties: lecture et écriture (1)

- Les instructions de lecture et d'écriture permettent à la machine de communiquer avec l'utilisateur
- La **lecture** permet d'entrer des donnés à partir du clavier
  - En pseudo-code, on note: **Lire (var)**

    la machine met la valeur entrée au clavier dans la zone mémoire nommée var
  - Remarque: Le programme s'arrête lorsqu'il rencontre une instruction Lire et ne se poursuit qu'après la frappe d'une valeur au clavier et de la touche Entrée

Les instructions d'entrées-sorties: lecture et écriture (2)

- **L'écriture** permet d'afficher des résultats à l'écran (ou de les écrire dans un fichier)
  - En pseudo-code, on note: **Ecrire (var)**

    la machine affiche le contenu de la zone mémoire var
  - Conseil: Avant de lire une variable, il est fortement conseillé d'écrire des messages à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper