

Bases de la programmation

été 2016/17

Dr Łukasz Starzak, MSc(Eng) PhD

Lodz University of Technology
Faculty of Electrical, Electronic, Computer and Control Engineering
Department of Microelectronics and Computer Science
ul. Wólczańska 221/223, bât. B18, bureau 51
<http://neo.dmcs.p.lodz.pl/~starzak> [pl, en]
<http://neo.dmcs.p.lodz.pl/bdp> [fr]

Objectif du cours

- **Pourquoi programmer ?**
 - ◆ L'ordinateur peut fournir la solution :
 - ▶ dans un temps plus court
 - ▶ avec plus de précision et vraisemblance
 - ▶ avec moins d'engagement de la part de l'humain
 - ◆ L'ordinateur est particulièrement utile en génie économique et gestion de même qu'en génie ingénieur
- **Mais il faut savoir comment :**
 - ◆ inventer ce que doit être fait et dans quel ordre
 - ◆ décrire les actions de façon compréhensible pour l'ordinateur
 - ◆ assurer que le résultat soit juste
 - ◆ trouver et éliminer des erreurs
 - ◆ améliorer son programme
 - ◆ coopérer avec d'autres codeurs et utilisateurs
- **Programmation est une matière à apprendre**

Plan du cours magistral

1. Notions de base de la programmation et de l'algorithmique
2. Éléments d'un langage de programmation ;
introduction au langage Matlab et à l'environnement Octave
 - a) Données et leurs types
 - b) Affectation et opérateurs arithmétiques
 - c) Obtention et affichage de données
3. Fonctions et portée de variables
4. Structures de contrôle
 - a) Algorithmes simples du traitement de données
 - b) Opérateurs de relation et logiques
 - c) Alternatives et boucles
 - d) Instructions de terminaison
5. Représentation et stockage de données
6. Problèmes avancés de l'algorithmique et de la programmation
7. Erreurs d'exécution et de programmation

Plan des travaux pratiques

- **1^{re} partie : Programmation en bureautique**
 - ◆ Enseignant : Dr. Małgorzata Napieralska
 - ◆ Durée : 15h = 5 semaines
 - ◆ Langage : HTML – développement de pages web
 - ◆ Langage : Visual Basic – automatisation des outils bureautique
- **2^e partie : Algorithmique en commerce**
 - ◆ Enseignant : Dr. Łukasz Starzak
 - ◆ Durée : 25h = 8 $\frac{1}{3}$ semaines (2h dans les semaines 13 et 14)
 - ◆ Langage : Matlab – calcul numérique
- **Travail personnel** en plus des classes sera nécessaire
 - ◆ c'est prévu dans le programme du cours
 - ◆ c'est normal pour tous les cours en génie informatique
- **Où travailler ?**
 - ◆ on peut accéder aux laboratoires du Département
 - ◆ on va utiliser des environnements gratuits (à l'exception de Microsoft Office) qui peuvent alors être installés sur vos ordinateurs personnels

Évaluation

- **Cours magistral**
 - ♦ interrogation écrite lors du dernier cours ou plus tard selon la décision du groupe
 - ▶ plus tard est conseillé afin d'avoir la possibilité de pratiquer tous les sujets, ce qui permettra de les comprendre mieux
 - ▶ voir la page web pour des repères (lorsque la date est fixée)
 - ♦ 2 interrogations de rattrapage
- **Travaux pratiques – bureautique**
 - ♦ les règles seront communiquées par Dr. Napieralska en TP
- **Travaux pratiques – algorithmique**
 - ♦ travail pendant les cours
 - ▶ montrez les résultats, sinon ils ne seront pas évalués
 - ♦ compte-rendu après que chaque énoncé est accompli
 - ▶ voir la page web pour des directives
- **Note finale**
 - ♦ $0,5 \cdot CM + 0,5 \cdot (15/40 \cdot TP_{bur} + 25/40 \cdot TP_{alg})$

Présence aux cours

- Travaux pratiques
 - ◆ présence obligatoire
 - ◆ 1 absence sans excuse tolérable
- Conférence
 - ◆ présence n'est pas obligatoire
 - ◆ présence n'affecte pas votre note (directement)
- Mais
 - ◆ je ne vais pas répéter en TP ce que j'aurai présenté et expliqué en conférence (ça ne veut pas dire qu'on ne peut pas me demander d'aide)
 - ◆ ne pas venir en CM, c'est déclarer qu'on sait déjà ou bien librement choisir de ne pas savoir alors je ne vais rien expliquer aux personnes absentes à la conférence
 - ◆ c'est pourquoi :
 - ▶ la présence en CM est fortement conseillée
 - ▶ on fera la liste
 - ▶ il faut demander si on ne comprend pas

1. Notions de base de la programmation et de l'algorithmique

Programmation
Langages de programmation
Cycle de vie



Processus de la programmation



- Un **problème** est une question à résoudre par une solution informatique
- Une **instance** du problème est une entrée nécessaire pour calculer une solution du problème
- Un **programme** est une description d'actions qu'un ordinateur doit effectuer pour résoudre un problème donné
 - ◆ un ensemble d'actions
 - ◆ un ordre d'exécution
 - ◆ données [d'entrée] (entrées)
 - ◆ résultats (sorties)

... est le processus de développement (création) de programmes

Programmation : exemple

- Problème
 - ♦ Trouver la température de l'air à une altitude donnée.
- Formule empirique valable dans la troposphère
 - ♦ $t(h) = t(0) - 6,4 \text{ °C/km} \cdot h$
- Instance du problème
 - ♦ Trouver la température à l'altitude de 1 km si elle est de 20 °C près de la surface.
- Programme (strictement : algorithme)
 1. Obtenir les données d'entrée t_0 et h
 2. Multiplier h et 6,4
 3. Mémoriser le résultat
 4. Soustraire le résultat précédent du t_0
 5. Renvoyer le résultat t_h
- Dans la plupart des cas, c'est une réponse au problème et non pas à son instance !
- Le degré de généralisation sera différent selon le cas

Principes méthodologiques

1. Abstraire

- ◆ Retarder le plus longtemps possible le développement du code destiné à l'ordinateur

2. Décomposer

- ◆ Diviser le problème en parties plus facilement abordables

3. Combiner

- ◆ Assembler les solutions partielles pour obtenir la solution globale

Spécification d'un problème

- Un **énoncé** est un texte définissant sans ambiguïté :
 - ♦ l'entrée (données d'entrée)
 - ♦ la sortie (résultats)
 - ♦ relations entre les données et les résultats
- Problèmes **mal posés**
 - ♦ Rechercher la position de l'élément le plus petit d'une suite.
 - ♦ Si l'entrée est $\{ 2, 3, 1, 4, 1, 9 \}$, le résultat c'est... ?
 - ♦ Complément nécessaire : quel doit être le résultat s'il y a plusieurs éléments égaux au minimum de la suite
 - ♦ Il peut y avoir une solution univoque
 - ♦ Sinon il faut demander à l'utilisateur du programme
 - ▶ première position ?
 - ▶ dernière position ?
 - ▶ un ensemble de toutes les positions ?
 - ▶ erreur ?

Précision sur la notion du programme : algorithme



- Un **algorithme** est une description d'un processus de résolution d'un problème défini
- Un **algorithme** est une succession d'actions qui, agissant sur un ensemble de données (entrée) fourniront la solution (sortie) à un problème défini
- Un **programme** est alors un algorithme destiné à l'ordinateur, écrit dans un **langage de programmation**
- La **codification** est le processus de conversion de l'algorithme en **code** (« texte » écrit dans un langage de programmation)
- Le programmeur ne doit pas laisser l'objectif final (le code) l'aveugler

Pseudo-code

- Langage le plus proche possible du langage humain, néanmoins :
 - ♦ simplifié – on n’observe pas toutes les règles de grammaire
 - ♦ formalisé – pas de synonymes, interdit de changer de l’ordre de mots
- Sert à décrire des algorithmes, non pas des programmes
 - ♦ Peut être traduit en langage de programmation quelconque
 - ♦ Certains symboles pourront avoir une signification différente en pseudo-code et dans le langage utilisé
- Symboles généralement reconnus : $+$ $-$ $*$ $/$ $=$ \neq $<$ $>$ etc.
- Symbole de l’affectation \leftarrow :
 - ♦ on écrit : $a \leftarrow 2$
 - ♦ on lit : affecter la variable a avec la valeur 2
 - ♦ ça veut dire : faire que la valeur de la variable a soit 2
- Caractères à signification définie :
 - ♦ $[]$ – élément d’un tableau (matrice, vecteur, chaîne de caractères...)
 - ♦ $()$ – arguments d’une fonction, procédure...
 - ♦ $.$ – composant (champ, méthode...) d’un objet, enregistrement...

Langages de programmation

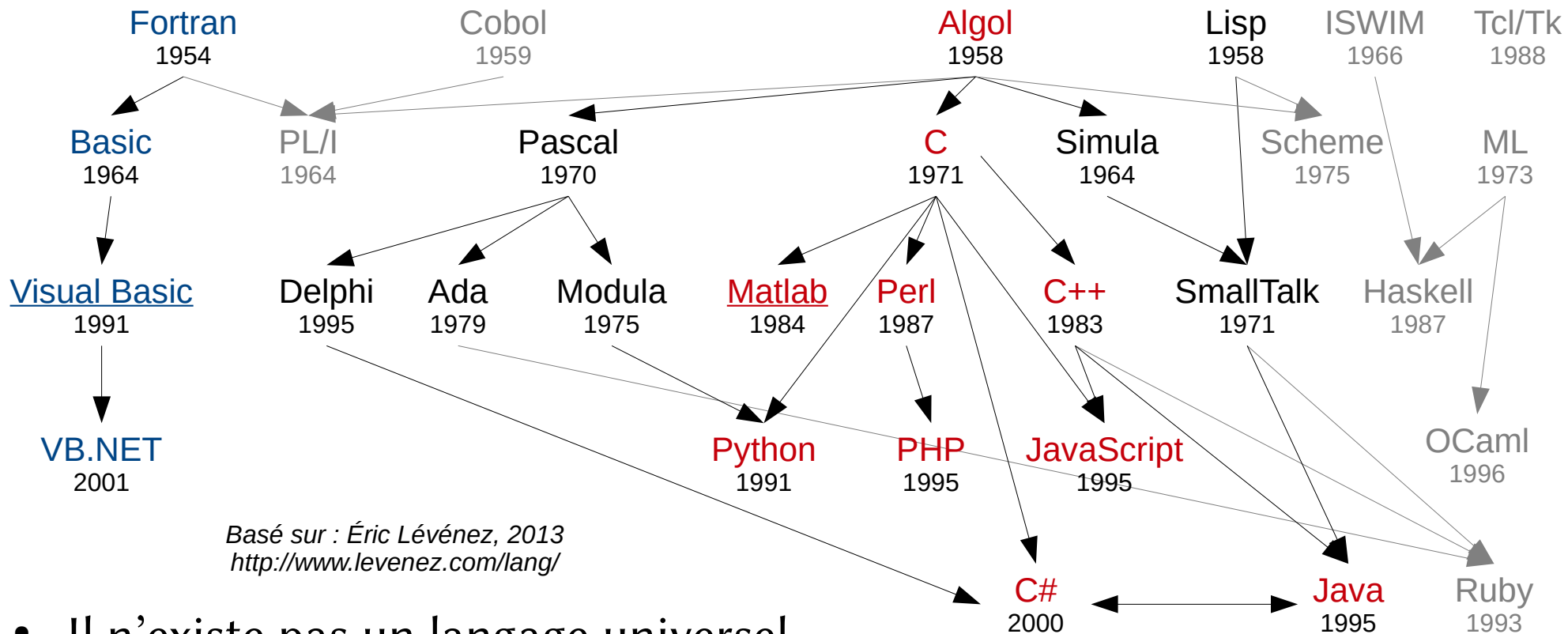
- **Bas-niveau**

- ◆ **1^{ère} génération** – codes machine
- ◆ **2^e génération** – assembleurs
- ◆ spécifiques pour chaque processeur
- ◆ directement compréhensibles (codes machine) ou littéralement traduisibles (assembleurs) pour le processeur
- ◆ alors directement exécutables par le processeur
- ◆ très difficiles (plus faciles les 2G) à comprendre par l'humain
- ◆ aujourd'hui rarement utilisés en génie informatique (plutôt en électronique)
- ◆ mais toujours utilisés par les processeurs eux-mêmes

- **Haut-niveau**

- ◆ plus proches du langage humain et du pseudo-code
- ◆ alors plus abordables pour l'humain
- ◆ incompréhensibles pour les processeurs
- ◆ ne dépendent pas du processeur
- ◆ il existe des traducteurs capables d'en produire un code machine
- ◆ **3^e génération** : plus formels et exigeants mais efficaces – exécution plus rapide et fiable
- ◆ **4^e génération** : faciles et abordables même pour des non-professionnels – programmation plus rapide

Généalogie simplifiée des langages haut-niveau 3G



- Il n'existe pas un langage universel

- ◆ Fortran : calcul numérique (mathématiques, contrôle)
- ◆ **Visual Basic** : programmes simples « macros » p. ex. en bureautique
- ◆ C, Python : programmes simples, fonctions de base, les systèmes Linux, MacOS
- ◆ C++ : projets complexes, le système Windows
- ◆ C#, Java : programmes modérément complexes avec interfaces graphiques
- ◆ PHP, Java, JavaScript : programmation réseaux (internet, sites web...)
- ◆ **Matlab** : créé strictement pour ce logiciel donc calcul numérique (4G)

Popularité des différents langages haut-niveau

Source : R. King, IEEE Spectrum, 2011
<http://spectrum.ieee.org/at-work/tech-careers/the-top-10-programming-languages>

