UNIVERSITÉ PARIS SUD

M1 Physique et application

Programmation et données numériques

Cours 16h: Guillaume Roux
TDs 20h: François Orieux, Aurélien Grabsch et Guillaume Roux

Programme du cours

- Cours 1-2-3 : Programmation (4 TDs)
  - architecture des ordinateurs
  - langages de programmation, introduction au langage Python
  - programmation orientée objet
- Cours 4-5-6 : Information et données (2 TDs)
  - numérisation de l'information: texte, signaux, images
  - produire, stocker, échanger et représenter l'information
- Cours 7-8 : Modélisation des données (4 TDs)
  - ajustement des données, optimisation
  - modélisation par apprentissage
- En parallèle: cours de traitement du signal (cours/TD/TP)
  - *TPs également en langage Python*

Qu'est ce que l'informatique?

- Informatique :
  traitement automatisé de l'information
  - Machine
  - Information
  - Algorithmique
  - Langage




- Mise en œuvre pratique
  - Matériel, hardware
  - Logiciels, software
  - Technologie de l'information et de la communication

Architecture d'un ordinateur

Qu'est qu'un ordinateur?

- À l'origine: un calculateur

- Aujourd'hui: machine qui manipule de l'information/données
  - stocke, produit, modifie de l'information (texte, son, image, vidéo)
  - permet de créer, *développer des logiciels* effectuant eux-mêmes de nouvelles tâches
  - *communique* avec des utilisateurs et d'autres machines (interface graphique, réseaux, internet, « cloud »)
  - est capable d'*apprendre automatiquement*, de contrôler d'autres machines: intelligence artificielle
- une structure universelle et « abstraite », ultra-polyvalente
- des réalisations de plus en plus diverses:
  portables, supercalculateurs, smartphone, tablette

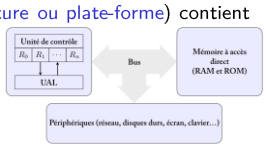
Une définition abstraite

- Machine de Turing (1936)
  ruban avec symboles, mémoire
  tête de lecture/écriture, table d'action
  => *peut exécuter n'importe quel algorithme*

- un ordinateur est la réalisation concrète d'une machine de Turing universelle, c'est-à-dire une machine traitant des informations et capable en principe de prendre comme *donnée d'entrée n'importe quel algorithme et de l'exécuter*
- ordinateur = machine programmable
- Un « système embarqué » n'est pas vraiment un ordinateur car il n'exécute que des programmes spécialisés même s'il pourrait en faire plus: lecteur DVD, microcontrôleurs...

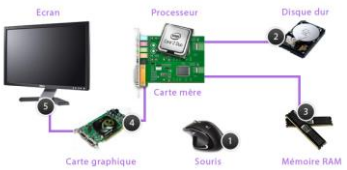
Architecture de von Neumann (1945)

- La réalisation physique (architecture ou plate-forme) contient
  - Mémoire vive
  - Processeur
  - Périphériques
  - Bus: assure les communications

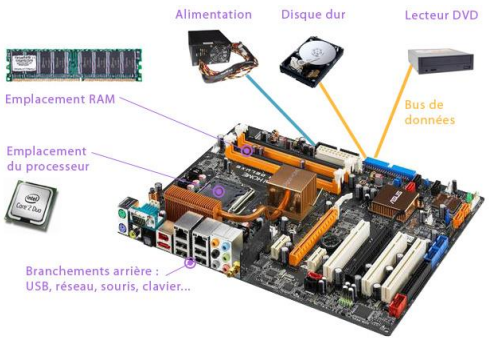


- Mémoire vive: stockage de l'information manipulée par l'algorithme, y compris celle représentant l'algorithme
  - ensemble de cases contenant des mots (64bits), inerte et volatile
  - chaque case a une adresse (un entier)
  - on peut lire/écrire directement dans n'importe quelle case (RAM = Random Access Memory), il n'y a pas de hiérarchisation
- Processeur:
  - va chercher les *instructions* et les *données* en mémoire par l'unité de contrôle
  - effectue les opérations successivement dans l'unité de calcul

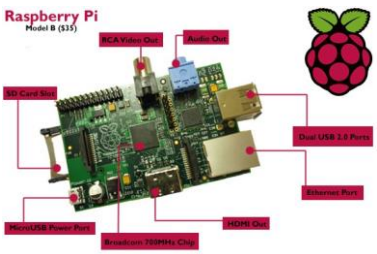
L'unité centrale et les périphériques



La carte mère



L'ordinateur le plus simple: Raspberry pi

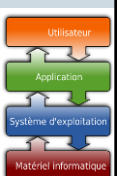


L'algorithmique

- Science qui consiste à proposer une méthode, une recette, un programme en vue de réaliser une tâche, résoudre un problème

- Se développe depuis des millénaires
  - divisions
  - extraction de racine carrée...
  - métiers à tisser
  - trier des données
  - trouver le chemin qui minimise la distance parcourue par le facteur
  - moteur de recherche google

Le système d'exploitation



- Problématique: il faut pouvoir
  - gérer plusieurs périphériques en parallèle
  - communiquer avec l'utilisateur, lancer un programme
  - faire tourner plusieurs programmes en même temps
- L'OS est chargé en mémoire vive par le BIOS qui assure le démarrage de la carte mère
- L'OS gère
  - Les utilisateurs (interface graphique et droits d'accès)
  - L'organisation du disque dur (système de fichiers)
  - Le multitâche
  - Les périphériques
- Deux principales familles:
  - Unix: ordinateurs personnels, serveurs, supercalculateurs, les Box, Ubuntu, Mac OS
  - Windows: ordinateurs personnels principalement

Langages de programmation

The three chief virtues of a programmer are: Laziness, Impatience and Hubris.
Larry Wall.

Pourquoi *des* langages de programmation?

- Il faut que l'homme communique avec la machine
  - texte source qui décrit l'algorithme
  - ensemble d'expressions et instructions organisées selon une grammaire
- Il en existe une grande variété pour plusieurs raisons
  - pratiques: adaptation à un problème considéré (traitement de texte)
  - culturelles: développement par des communautés selon des goûts ou une « philosophie » de programmation
  - conceptuelles: il existe des classes de langages qui ont des propriétés génériques, suivent des règles de conception (impératif, *orienté objet*, fonctionnel,...)
- Les logiciels et les systèmes d'exploitation actuels sont programmés à l'aide de plusieurs langages.
- Il est aujourd'hui plus *utile* de connaître un peu de *plusieurs langages* plutôt qu'un seul à fond.

Concevoir un langage de programmation

- Il faut une grammaire logique et stricte pour éviter toute ambiguïté dans les instructions:
  - Syntaxe: le texte écrit doit respecter des règles d'écriture
  - Sémantique: le texte écrit doit correspondre au comportement attendu (problème de conception ou de logique)
- Le langage doit être compris par la machine et par l'homme: il faut trouver le bon point d'équilibre entre le détail dont la machine a besoin et le souhait de concision et de symbolisation du langage humain.
- Indépendant de la plateforme, de la machine sur lequel il va être compilé (sauf le langage machine)
- Les langages sont normalisés ou standardisés. La plupart dispose de nombreuses bibliothèques.

Langage machine et assembleur

- Langage machine: instructions sur des lignes écrites en binaire et directement interprétées par le processeur. Humainement inutilisable.
  Ex: ajouter les registres 1 et 2 et placer le résultat dans le registre 6
  ```
  [  op  |  rs |  rt |  rd |shamt| fonct ]
   000000 00001 00010 00110 00000 100000  (mot de 32bits)
  ```
- Langage assembleur: langage de bas niveau écrivable par un humain ayant des instructions proches du langage machine (il est *assemblé* par un programme appelé *assembleur*)

```
        .data       #(1) zone de données

nb1:    .byte 1     #(2) premier octet :
                    # valeur 1 en base décimale
res:    .byte 0     #(3) troisième octet :
                    #valeur 0 en base décimale

        .text       #(4) zone d'instructions

Addition :
            #(5) copier la valeur contenue dans l'octet
            # d'adresse nb1 dans le registre al
        movb    nb1,    %al
            #(6) ajouter la constante entière de
            # valeur 5 en base décimale au registre al
        addb    $5,     %al
            #(7) copier la valeur contenue dans le
            # registre al (1 octet) à l'adresse res
        movb    %al,    res
```

En langage C:

```
short int nb1=1;
short int res = 0;
res = nb1+5;
```

Vers un langage de haut niveau

- du langage machine au langage C en passant par l'assembleur

55 48 89 E5	`push rbp` `mov rbp, rsp`	`int main() {`
C7 45 FC 02	`mov DWORD PTR [rbp-4], 2`	`int a = 2;`
C7 45 F8 03	`mov DWORD PTR [rbp-8], 3`	`int b = 3;`

8B 45 F8
8B 55 FC
01 D0
89 45 F4
	`mov eax, DWORD PTR [rbp-8]`
	`mov edx, DWORD PTR [rbp-4]`
	`add eax, edx`
	`mov DWORD PTR [rbp-12], eax`
		`int c = a + b;`

8B 45 F4	`mov eax, DWORD PTR [rbp-12]`	`return c;`
5D C3	`pop rbp` `ret`	`}`

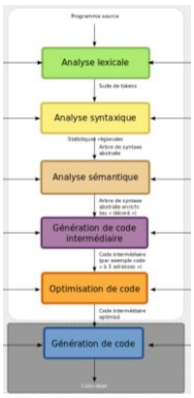
Compilation et exécution

- Code source: fichier texte content les instructions écrites dans le langage de programmation
- Compilateur: ~*traducteur*, un programme/logiciel qui transforme le code source pour créer:
  - soit en langage machine (une application)
    - un fichier objet => pour un éditeur de liens ou une bibliothèque
    - un exécutable, c'est-à-dire une application ou un programme
  - soit des instructions en langage intermédiaire pour réutilisation
    - bytecode => pour un interpréteur (fichiers .pyc en Python)
- Les compilateurs actuels sont des logiciels extrêmement importants et sophistiqués, *capable de comprendre la structure d'un programme et de l'optimiser.* Proches de l'intelligence artificielle.

Les étapes de la compilation

- Analyse lexicale
  on vérifie que les mots sont corrects
- Analyse syntaxique
  que les phrases sont correctes
- Analyse sémantique
  qu'une partie du sens est correct
  (argument d'une fonction, définition d'une variable...)
- Génération d'un code intermédiaire
  contenant la structure

- Optimisation: réorganisation, minimisation des calculs, gestion de la mémoire, étapes en parallèle.
- Génération de l'exécutable
  produit en langage machine



Interpréteur et scripts

- Un langage interprété est un langage dont les instructions sont lues « à la volée » par un logiciel (l'interpréteur) et interagit dynamiquement avec l'utilisateur.
- Pas une spécificité de la grammaire du langage mais plutôt de son implémentation.
- On appelle scripts les petits programmes écrits en langage interprété ou langage script.
- Très pratique et rapide pour de petits programmes, pour apprendre et faire des tests, car évite la compilation.
- Exemples: JavaScript, Python,..., Mathematica, Matlab

- Shell: langage script utilisé pour les interfaces en ligne de commande avec les utilisateurs:
  bash (Unix, Mac OS), MS-DOS (Windows)

Les logiciels et bibliothèques

- On ne peut/veut pas réinventer la roue pour de nombreuses fonctions et méthodes. Pour cela, il existe des bibliothèques écrites par des programmeurs dont on peut utiliser les fonctions déjà définies et précompilées en code objet.

- Il faut savoir utiliser ces bibliothèques. Il faut savoir passer du temps à lire la documentation.
  Exemples: fonctions mathématiques de base, interface graphique...

- On peut écrire des bibliothèques soi-même pour gagner du temps et structurer son programme.

Introduction au langage Python

Pourquoi Python?

- Caractéristiques principales
  - langage interprété
  - typage dynamique, autogestion de la mémoire
  - orienté objet
  - nombreuses bibliothèques (scientifiques et graphiques)

- Avantages
  - libre, gratuit, présent sur toutes les plate-formes, en expansion
  - lisibilité, concision, facile à apprendre, idéal pour la pédagogie
  - Très bien documenté
  - IPython: feuille de travail très pratique
  - choisi par l'éducation nationale (lycée et classes prépa)
- Inconvénients
  - bugs parfois difficile à trouver à cause du typage dynamique
  - lenteur pour les calculs exigeants

Python: versions et bibliothèques

- Attention il y a deux versions pas complètement compatibles:
  - Python 2.x (nous utiliserons Python 2.7)
  - Python 3.x

- Bibliothèques scientifiques utiles et reliées entre elles
  - Scipy: librairie scientifique générale
  - Numpy: calcul numérique, en particulier algèbre linéaire
  - Matplotlib: pour les graphes