Université de Montréal

IFT-2035

Concepts des Langages de Programmation

Stefan Monnier (AA-2341)

monnier@iro.umontreal.ca

http://www.iro.umontreal.ca/~monnier/2035/

- David A. Watt, *Programming Language Design Concepts*
- Ravi Sethi, *Programming Languages: Concepts & Constructs*
- Benjamin C. Pierce, *Types and Programming Languages*
- Simon Thompson, *Haskell: The Craft of Functional Programming*
- Paul Hudak, *The Haskell School of Expression*

Université de Montréal

Calendrier

Historique, syntaxe des langages

Compilation et interprétation

Sémantique des langages, λ-calcul, types

Programmation fonctionnelle

Portée, passage d'argument

Fonctions d'ordre supérieur, fermetures

Programmation logique

Programmation impérative

Représentation des données

Pointeurs et gestion mémoire

Macros, totalité, exceptions, modules, abstraction

Concurrence

Université de Montréal

Aperçu du cours

Il y a plusieurs milliers de langages de programmation

- Être capable d'utiliser efficacement la majorité de ces langages
- Savoir comparer et tirer parti de leurs similarités et différences
- Pouvoir comprendre comment ils interagissent
- Comprendre les concepts fondamentaux
  - Syntaxe et sémantique, types
  - Abstraction
  - Portée et passage d'arguments
  - Analyse, raisonnement, liberté d'implantation
- Mieux programmer

Université de Montréal

Comment?

- Peu de *syntaxes*
- Peu de différents *styles de programmation*
- Équivalence de Turing

Confiné par notre créativité, les contraintes des langages machines, et les besoins de performance

Université de Montréal

Styles de programmation

- Impératif
  - Procédural
  - Objet
- Déclaratif
  - Fonctionnel
  - Logique
- Concurrent
  - Mémoire partagée
  - Passage de messages

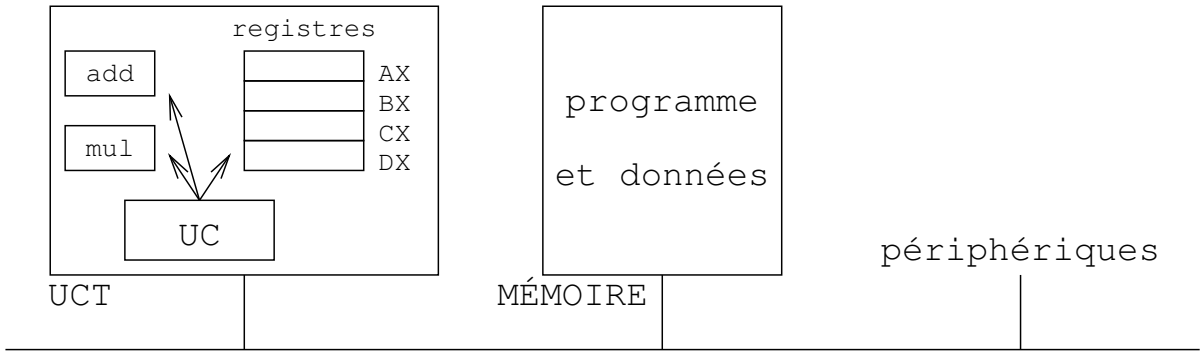
Université de Montréal

Historique: langage machine (1)

Seul langage compris directement par la machine

La mémoire contient le programme et les données, le processeur opère dessus:

- Transferts entre mémoire et registres
- Opérations internes (arithmétiques, tests, ...)
- Flot de contrôle



Université de Montréal

Historique: langage machine (1)

1. Lire le mot de l'instruction courante
2. Le décoder
3. Chercher les opérandes
4. Effectuer l'opération
5. Socker le résultat
6. Passer à l'instruction suivante.

```
10001011
01000101
00001010
00000011
01000101
00010100
```

Les instructions et les données sont encodées par des chaînes de bits

Différent pour chaque type de machine

Inintelligible pour les humains

Université de Montréal

Historique: assembleur

Représentation textuelle symbolique du langage machine

```
mov 10(%ebp),%ax
add 20(%ebp),%ax
```

Toujours différent pour chaque type de machine

La mémoire manque toujours de structure

Irrégularités du jeux d'instructions

Beaucoup de détails à écrire; les erreurs sont légion

Beaucoup de variantes; la meilleure est différente pour chaque machine

Université de Montréal

Historique: Langages de haut niveau

Langage indépendant de la machine; se rapproche de l'humain

Même exemple qu'avant: $X + Y$

- Notation plus familière
- Portabilité
- Fiabilité

Nombre de lignes/jour et bugs/ligne plus ou moins constant

- Plus facile *à lire*
- Plus restrictif

Université de Montréal

Une infinité de niveaux

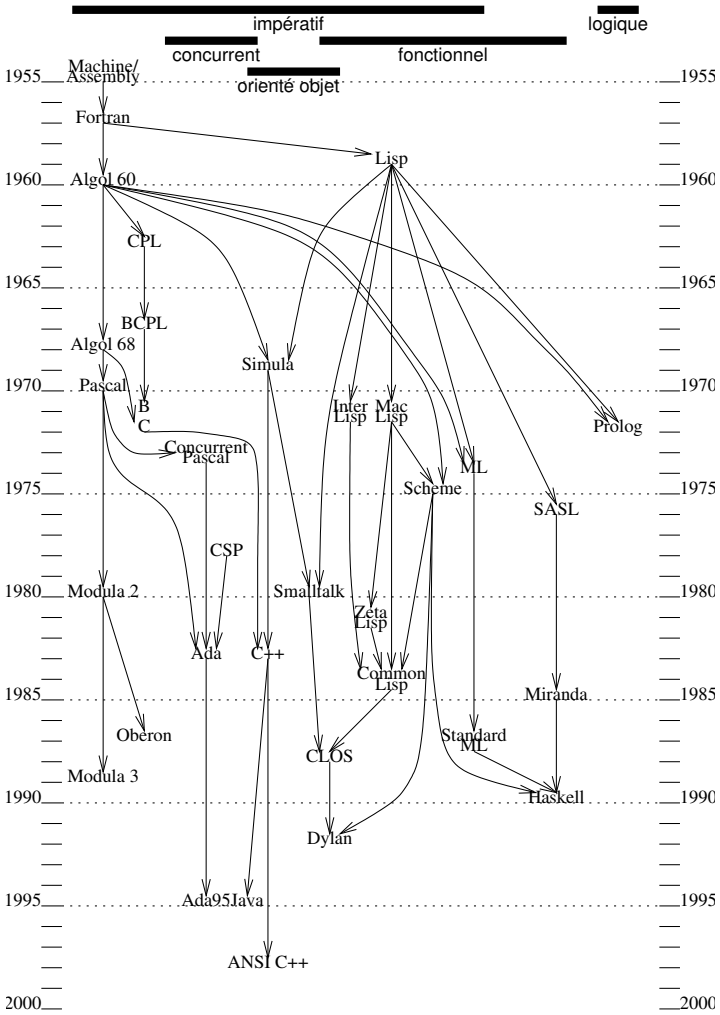
Tous les langages de haut niveau ne sont pas nés égaux

$$\textit{machine} < \textit{assembleur} < \textit{C} < \textit{Java} < \textit{anglais} < \textit{Stefan}$$

Le *niveau d'abstraction* est la distance conceptuelle par rapport au langage machine.

Les langages souvent combinent des aspects de niveaux différents, donc il n'y a pas toujours d'ordre clair entre langages.

Généalogie



Les pionniers:

Fortran, Lisp, Cobol

Les influents:

Algol, Simula, Prolog, ML

Université de Montréal

Programmation impérative: procédurale

Fortran, Algol 60, Pascal, C, Ada

Même modèle que la machine: *séquence* d'opérations sur la mémoire

La mémoire est composée d'un graphe d'objets

Les instructions sont regroupées en procédures

Notion implicite d'*état* omniprésente; *effets de bord*

Facile à traduire en langage machine (*compiler*)

De nos jours, considéré comme "bas niveau"

Université de Montréal

Programmation impérative: objets

Simula, Smalltalk, C++, Java

Chaque objet de la mémoire est accompagné de code qui lui permet d'interagir avec les autres objets

Les objets sont actifs

Les méthodes remplacent les procédures

Le flot de contrôle passe d'un objet à l'autre par appel de méthode

Plusieurs types d'objets peuvent être utilisés de manière uniforme

Université de Montréal

Programmation fonctionnelle

Lisp, ML, Haskell, APL

Calcul = fonction au sens mathématique

Éviter les effets de bord

Pas de destruction, pas de dépendance au temps

Facilite l'analyse, le raisonnement

Les fonctions peuvent être manipulées comme n'importe quel objet

Apprécié pour son élégance

Université de Montréal

Programmation logique

Prolog, Mercury, Oz

Calcul = recherche d'une preuve logique ou d'une solution

Très déclaratif: décrire ce que l'on veut, pas comment l'obtenir

Modèle d'exécution très différent: difficile à mélanger

On ne peut pas prouver n'importe quoi automatiquement

Université de Montréal

Programmation concurrente

CSP, Occam, Ada, Modula-2

Calcul divisé en plusieurs tâches, exécutées simultanément

Synchronisation et communication, implicites ou explicites

Apporte potentiellement des accélérations importantes

Non-déterminisme

Beaucoup d'occasions d'introduire des erreurs

Université de Montréal

Plagiat

Plagiat $\Longrightarrow$ Pas bien

Implantation

Interprétation (aka VM)

- Décodage progressif des instructions
- Simple, compact, portable, flexible

Compilation

- Traduction d'un langage dans un autre
- Vitesse, *standalone*, compilation croisée

Hybrides