

École Polytechnique

# INF564 – Compilation

Jean-Christophe Filliâtre

introduction

**9 blocs**, du 4 janvier au 8 mars

- **cours** 14h00–16h00
  - sur [bbb.lri.fr/b/jea-7km-uh9](http://bbb.lri.fr/b/jea-7km-uh9)
  - transparents / photocopié sur moodle
- **TD** dans la foulée, 16h15–18h15
  - sur [inf564.slack.com](https://inf564.slack.com)
  - avec Georges-Axel Jaloyan et moi-même

- un **examen** écrit (lundi 15 mars, 14h–17h)
- un **projet** = un mini compilateur C vers x86-64
  - réalisé en TD (à partir du TD 4) et un peu en dehors
  - seul ou en binôme
  - en Java ou en OCaml

$$note\ finale = \frac{examen + projet}{2}$$

on privilégie **slack** pour la communication

- `inf564.slack.com` (vous avez reçu une invitation)
- pour les TD, sur le canal #td
- canal #cours pour les questions en dehors du cours
- message direct à un enseignant si besoin

n'hésitez pas à l'utiliser entre vous (messages directs),  
par exemple avec votre binôme de projet

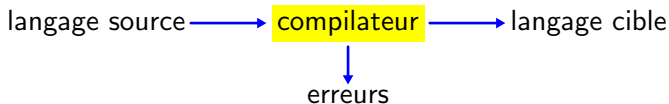
<https://www.enseignement.polytechnique.fr/informatique/INF564/>

- archives examen
- ressources diverses : outils, lectures, sites web, etc.

maîtriser les mécanismes de la **compilation**,  
c'est-à-dire de la transformation d'un langage dans un autre

comprendre les différents aspects des **langages de programmation**  
par le biais de la compilation

schématiquement, un compilateur est un programme qui traduit un « programme » d'un langage **source** vers un langage **cible**, en signalant d'éventuelles erreurs



# compilation vers le langage machine

quand on parle de compilation, on pense typiquement à la traduction d'un langage de haut niveau (C, Java, OCaml, ...) vers le langage machine d'un processeur

```
% gcc -o sum sum.c
```

source `sum.c` → **compilateur C (gcc)** → exécutable `sum`

```
int main(int argc, char **argv) {  
    int i, s = 0;  
    for (i = 0; i <= 100; i++) s += i*i;  
    printf("0*0+...+100*100 = %d\n", s);  
}
```

```
0010011111011111011111111111111100000  
10101111110111111100000000000010100  
1010111111010010000000000000100000  
1010111111010010100000000000100100  
101011111101000000000000000011000  
101011111101000000000000000011100  
100011111101011100000000000011100  
...
```



dans ce cours, nous allons effectivement nous intéresser à la compilation vers de **l'assembleur**, mais ce n'est qu'un aspect de la compilation

un certain nombre de techniques mises en œuvre dans la compilation ne sont pas liées à la production de code assembleur

certains langages sont d'ailleurs

- interprétés (Basic, COBOL, Ruby, Python, etc.)
- compilés dans un langage intermédiaire qui est ensuite interprété (Java, OCaml, Scala, etc.)
- compilés à la volée (Julia, etc.)
- compilés vers un autre langage de haut niveau

# différence entre compilateur et interprète

un **compilateur** traduit un programme  $P$  en un programme  $Q$  tel que pour toute entrée  $x$ , la sortie de  $Q(x)$  soit la même que celle de  $P(x)$

$$\forall P \exists Q \forall x \dots$$

un **interprète** est un programme qui, étant donné un programme  $P$  et une entrée  $x$ , calcule la sortie  $s$  de  $P(x)$

$$\forall P \forall x \exists s \dots$$

dit autrement,

le compilateur fait un travail complexe **une seule fois**, pour produire un code fonctionnant pour n'importe quelle entrée

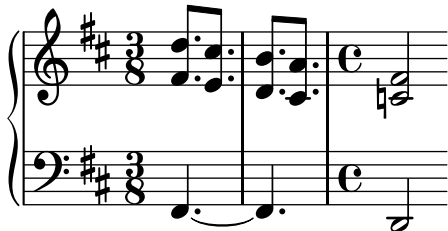
l'interprète effectue un travail plus simple, mais le refait sur chaque entrée

autre différence : le code compilé est généralement bien plus efficace que le code interprété

# exemple de compilation et d'interprétation

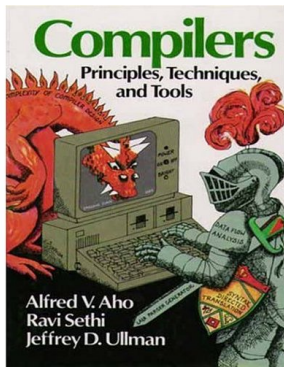
source → lilypond → fichier PDF → evince → image

```
\new PianoStaff <<  
  \new Staff { \clef "treble" \key d \major \time 3/8  
    <<d8. fis,8.>> <<cis'8. e,8.>> | ... }  
  \new Staff { \clef "bass" \key d \major  
    fis,,4. ~ | fis4. | \time 4/4 d2 }  
>>
```



à quoi juge-t-on la qualité d'un compilateur ?

- à sa correction
- à l'efficacité du code qu'il produit
- à sa propre efficacité



"Optimizing compilers are so difficult to get right that we dare say that no optimizing compiler is completely error-free! Thus, the most important objective in writing a compiler is that it is correct."

(Dragon Book, 2006)

typiquement, le travail d'un compilateur se compose

- d'une phase d'**analyse**
  - reconnaît le programme à traduire et sa signification
  - signale les erreurs et peut donc échouer (erreurs de syntaxe, de portée, de typage, etc.)
- puis d'une phase de **synthèse**
  - production du langage cible
  - utilise de nombreux langages intermédiaires
  - n'échoue pas

