



# Interprétation et compilation

## Chapitre 4 Représentation intermédiaire



Pablo Rauzy <pr@up8.edu>  
[pablo.rauzy.name/teaching/ic](http://pablo.rauzy.name/teaching/ic)

## Représentation intermédiaire

---

- ▶ Objectif : compilateur pour un langage impératif vers de l'assembleur MIPS.
- ▶ Jusque là nous savons :
  - écrire de l'assembleur MIPS,
  - représenter cet assembleur en OCaml,
  - générer du code MIPS depuis une représentation en OCaml.
- ▶ Pour poursuivre notre “remontée” des étapes de la compilation, nous devons maintenant voir comment représenter un langage impératif à haut niveau.
- ▶ Cette *représentation intermédiaire* sera la cible de notre *front-end*.

- ▶ Une fois établie notre représentation intermédiaire, il faudra apprendre à la *compiler* vers notre représentation de l'assembleur.
- ▶ Avant d'en arriver là, nous ferons un détour par l'*interprétation*, directement sur notre représentation intermédiaire.

- ▶ On qualifie souvent le langage C d'*assembleur portable*, à moitié par plaisanterie.
- ▶ C'est un peu exagéré, mais c'est vrai qu'on peut considérer C, parmi les langages de haut niveau, comme celui de plus bas niveau.
- ▶ Mais alors, quel est le saut qualitatif qui le distingue de l'assembleur ?

- ▶ Essentiellement, quatre choses permettent de qualifier le C de langage de plus haut niveau que l'assembleur :
  - ses types de données ;
  - les variables (dont on dispose d'autant qu'on veut contrairement aux registres) ;
  - sa syntaxe, plus proche des mathématiques et donc plus naturelle ;
  - la présence de structures de contrôle.

## Représenter un langage impératif “haut niveau”

- ▶ Oublions les spécificités de C et voyons comment représenter un langage impératif.

## Types de données

- ▶ Plusieurs types de base : rien, booléen, nombre, chaîne de caractères, etc.
  - On peut également créer de nouveaux types en composant ceux existants,
  - ou pourrait aussi avoir des types dérivés (liste de, pointeur sur, etc.),
  - mais mettons de côté ces possibilités pour l'instant.
- ▶ Dans notre représentation intermédiaire, nous avons besoin de représenter des *valeurs* de ces différents types de données :

---

```
1 type value =
2   | Nil
3   | Bool of bool
4   | Int of int
5   | Str of string
```

---

## Expressions

- ▶ Une expression dans un langage impératif peut être :

## Expressions

► Une expression dans un langage impératif peut être :

- une valeur constante,
- une variable,
- un appel de fonction (y compris les opérateurs natifs).

---

```
1 type ident = string
2
3 type expr =
4   | Value of value
5   | Var   of ident
6   | Call  of ident * expr list
```

---

## Instructions (et structures de contrôles)

- ▶ Une instruction dans un langage impératif peut être :

## Instructions (et structures de contrôles)

► Une instruction dans un langage impératif peut être :

- un **return**,
- une expression impérative (appel de procédure),
- une assignation d'une expression à une variable,
- une condition,
- une boucle.

---

```
1 type instr =
2   | Return of expr
3   | Expr   of expr
4   | Assign of ident * expr
5   | Cond   of expr * block * block
6   | Loop   of expr * block
7 and block = instr list
```

---

► On pourrait rajouter d'autres instructions impératives :

- de flots de contrôles (par exemple **break**),
- mais faisons au plus simple pour l'instant.

## Définition de fonctions

- ▶ Enfin, un programme est une suite de définitions.
  - Pour l'instant nous n'avons que des définitions de fonctions,
  - on pourrait avoir également des définitions de nouveaux types (structures) plus tard.
- ▶ Une fonction a un nom, des arguments, et un corps composé d'instructions :

---

```
1 type def =
2   | Func of ident * ident list * block
3
4 type prog = def list
```

---

## Des oubliés ?

- ▶ Est-ce qu'il n'y a pas quelque chose d'important qu'il vous semble qu'on aurait oublié ?

## Des oubliés ?

- ▶ Est-ce qu'il n'y a pas quelque chose d'important qu'il vous semble qu'on aurait oublié ?
- ▶ Les informations de typage !
  - types de variables,
  - types des arguments des fonctions,
  - type de retour des fonctions.

## Des oubliés ?

- ▶ Est-ce qu'il n'y a pas quelque chose d'important qu'il vous semble qu'on aurait oublié ?
- ▶ Les informations de typage !
  - types de variables,
  - types des arguments des fonctions,
  - type de retour des fonctions.
- ▶ En fait on en aura plus besoin à ce moment là car tout aura déjà été vérifié avant.
  - En cas d'erreur de typage, le compilateur ou interpréteur aura donc rejeté le programme avant d'en arriver à cette étape.
  - On verra tout cela lors de la phase d'*analyse sémantique*.