Licence STS, semestre 4 — 2009–2010
Langages et Génie Logiciel (Info 223) — 5 février 2010
`http://www.lri.fr/~paulin/LGL`

Cours 3 : unités de compilation

1 Interfaces et implémentations

Le principe de base du génie logiciel est le découpage d'une application en plusieurs parties relativement indépendantes appelées *unités de compilation*. Ce découpage permet à la fois de maîtriser la complexité de logiciels de grandes tailles, de réaliser un développement en équipe et de recompiler rapidement le programme en ne recompilant que ce qui est nécessaire après une modification.

En OCaml, chaque unité de compilation est un couple de deux fichiers : le fichier *interface* et le fichier *implémentation*. Ces fichiers portent le même préfixe, seules les extensions diffèrent :

- le fichier interface (`.mli`) définit les types (abstraits ou concrets) et les signatures des valeurs visibles à l'extérieur ;
- le fichier implémentation (`.ml`) contient les définitions (concrètes) de tous les types et de toutes les valeurs (visibles ou non) de l'unité de compilation.

Si `module.mli` et `module.ml` sont les fichiers d'interface et d'implémentation d'une unité de compilation, on utilisera le nom `Module` pour désigner cette unité. On utilisera également la notation `Module.v` pour faire référence à la valeur `v` de `Module`.

La directive `open Module` évite d'utiliser la notation pointée pour faire référence aux valeurs de `Module`.


Si deux unités `M` et `N` contiennent la même valeur `v`, alors seule la déclaration de `N` est visible après les deux directives consécutives `open M` et `open N`.

2 Graphe de dépendances

L'idée principale dans ce découpage est que pour concevoir une unité de compilation il est seulement nécessaire de connaître les interfaces des autres unités.

Lorsqu'une unité `M1` fait référence à une unité `M2`, on dit que `M1` *dépend* de `M2`. L'unité `M1` peut faire référence à `M2` soit dans son interface, soit dans son implémentation. Dans un programme avec plusieurs unités de compilation, la relation *«dépend de»* forme un graphe de dépendances. Ce graphe définit un ordre *partiel* de compilation.


Attention à ne pas créer de cycles dans le graphe de compilation, cela empêcherait de compiler votre programme.

3 Compilation et édition de liens

Le langage OCaml possède deux compilateurs, `ocamlc` et `ocamlopt`, pour produire respectivement des exécutables en *bytecode* et en code *natif*.

– La phase de compilation effectue l'analyse syntaxique, le typage et la production de codes à *trous* (on parle de fichiers *objets*).
– La phase d'édition de liens construit un exécutable en «remplissant» les trous, selon l'ordre des fichiers objets donnés en arguments.

Voici un récapitulatif des différentes extensions des fichiers utilisés en OCaml ainsi que des options pouvant être passées aux compilateurs :

Extension	Signification
`.mli`	interface
`.ml`	implémentation
`.cmo`	fichier objet (bytecode)
`.cmx`	fichier objet (natif)
`.cma`	bibliothèque objet (bytecode)
`.cmxa`	bibliothèque objet (natif)
`.cmi`	interface compilée

Option	Signification
`-c`	compile sans faire d'édition de liens
`-o` *nom*	spécifie le nom de l'exécutable (ou de la bibliothèque)
`-I` *rep*	ajoute un répertoire pour la recherche de fichiers (`.cmi`, `.cmo`, `.cmx`, `.cma` et `.cmxa`)
`-a`	construit une bibliothèque