https://www.laurentbloch.net/BlogLB/Compilation-separee-et-modules-de

Compilation séparée et modules de Bigloo

- Zinformatiques - Cours de bioinformatique au CNAM -

Date de mise en ligne : lundi 4 octobre 2004

Sommaire

Les modules de Bigloo : la compilation séparée

Les modules de Bigloo :
la compilation séparée

Laurent Bloch

Programmation modulaire

1 Premier exemple de compilation séparée

Supposons que nous voulions écrire un progamme compilable de calcul de factorielle, mais que nous hésitions entre l'algorithme itératif et l'algorithme récursif. Afin de ne pas avoir à tout recompiler à chaque fois, nous voulons donc séparer le programme en deux parties placées dans deux fichiers différents. Voici comment nous pouvons réaliser ceci avec les modules de Bigloo. Un premier fichier (`entre-fact.scm`) contiendra la procédure principale `entre` chargée d'appeler la procédure de calcul proprement dite :

Le fichier `entre-fact.scm` contient le programme principal (module entre-fact-module (main entre) (import (fact-module "factorielle.scm"))) (define (entre argv) (display "Entre un entier positif : ") (let boucle ((n (read))) (if (< n 1) (exit 0)) (print (fact n)) (display "Entre un entier positif : ") (boucle (read)))) On remarquera dans la déclaration du module `entre-fact-module` (contenu dans le fichier `entre-fact.scm`) la clause `import`, qui dit « importe tout ce que le module `fact-module` contenu dans le fichier `factorielle.scm` exporte ».
Un second fichier `factorielle.scm` contiendra la procédure de calcul en question :

Le fichier `factorielle.scm` contient module qui exporte la procédure de calcul (module fact-module (export (fact n))) (define (fact n) (do ((i n (- i 1)) (f 1 (* i f))) ((= i 0) f)))
D'autre part la clause de module `export` dans le fichier `factorielle.scm` qui contient la procédure de calcul dit que ce module `fact-module` exporte la procédure `fact`. On sait que c'est une procédure, et non un objet quelconque éventuellement mutable, par la syntaxe `(fact n)`. C'est important parce qu'un objet mutable quelconque serait affecté sur la pile et consommerait de la mémoire à chaque appel récursif éventuel.

Pour obtenir à partir de ces deux fichiers un programme exécutable il faut procéder en trois temps, compiler indépendamment chaque fichier, puis en effectuer l'*édition de liens* : bigloo -c entre-fact.scm bigloo -c factorielle.scm bigloo factorielle.o entre-fact.o -o fact Le drapeau `-c` indique au compilateur de s'arrêter à la production du code objet sans essayer de construire un exécutable. Le drapeau `-o` précède le nom que l'on souhaite donner à l'exécutable.

---

2 Construction d'un programme modulaire

Au chapitre sur la compilation nous avons vu comment construire un programme mono-module : # bigloo -o *<nom d'exécutable> <nom de fichier source>* [E] Ici c'est un peu plus compliqué, nous devons compiler chacun des deux modules, puis les relier ensemble pour construire l'exécutable. L'invocation de la commande `bigloo` montrée à la ligne précédente enchaînait automatiquement compilation et construction. Ici il va falloir décomposer la construction en deux temps, compilation et édition de liens.

Cette séparation en deux étapes peut sembler compliquée, mais elle est de toute façon inévitable si l'on veut relier ensemble des modules écrits dans des langages différents, donc forcément compilés par des compilateurs différents et réunis *in fine* par l'édition de liens.

2.1 Compilation de modules source en fichiers objets

Compiler sans édition de liens chaque module se commande avec l'option `-c` de la commande `bigloo` : # bigloo -c *<nom de fichier source>* qui produira un fichier dit fichier objet, de même nom que le fichier source, mais suffixé par `.o` au lieu de `.scm` : factorielle.scm -Â» factorielle.o Pour compiler un module `m` qui importe des choses d'un module `n` il faut indiquer au compilateur dans quel fichier se trouve `n`. Ceci peut se faire directement dans les clauses de module comme dans l'exemple ci-dessus mais il est préférable de rédiger un fichier de contrôle dit `access-file` qui contient les correspondances module-fichier sous la forme suivante :

`access-file.scm` donne les fichiers contenant les modules. ((fact-module "factorielle.scm") (entre-fact-module "entre-fact.scm"))

- fichier nommé, par exemple, `access-file.scm` ;

- utilisation de ce fichier par la commande `bigloo` : # bigloo -afile access-file.scm -c entre-fact.scm

2.2 Édition de liens pour construire l'exécutable

Une fois obtenus les fichiers objet pour chaque module (ici trois), leur liaison va se faire toujours par la commande `bigloo` : # bigloo factorielle.o entre-fact.o -o fact Le processus de construction est représenté par le schéma de la figure 1.

<IMG5|left>

Figure 1 : Construction d'un exécutable.

<IMG5|center> Ce processus est d'une complexité non négligeable, il serait plus confortable de pouvoir le programmer. Ce sera l'objet d'une leçon suivante.

Index

Ce document a été traduit de LATEX par HEVEA.