FACULTÉ DES ARTS ET DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE ET DE RECHERCHE OPÉRATIONNELLE

TITRE DU COURS: **Langages de programmation et compilation**
SIGLE DU COURS : **IFT3065**

NOM DU PROFESSEUR: Marc Feeley

DATE DE L'EXAMEN INTRA A07: 2 novembre 2007 HEURE: 9h30-11h20 SALLE: PAA 1409

DIRECTIVES PEDAGOGIQUES:

- **Aucune documentation permise.**
- **Lorsque des explications ou définitions sont demandées, soyez bref mais précis (pas de blabla inutile).**
- **N'oubliez pas de mettre votre nom et code permanent sur le cahier réponse.**

Question 1 – (15 points). Décrivez l'approche d'amorçage incrémental de compilateur ("bootstrap incrémental"). Utilisez la notation "T" pour donner un exemple d'utilisation de cette approche. Dans quelle(s) situation(s) cette approche est-elle appropriée? Donnez deux avantages importants de cette approche et un problème important.

Question 2 – (15 points). Qu'est-ce qu'une "référence forte"? Qu'est-ce qu'une "référence faible"? Quel est l'attrait des références faibles? Expliquez une situation d'utilisation des références faibles (vous pouvez donner un exemple en Scheme).

Question 3 – (15 points). Écrivez un programme Scheme qui imprime en ordre croissant les 1000 premiers nombres qui sont le produit d'une puissance de 2 et une puissance de 3. C'est-à-dire tout nombre N tel qu'il existe $i, j \geq 0$ donnant

$$N = 2^i * 3^j$$

Note : 12 points sur 15 seront donnés pour une solution correcte, les 3 derniers points sont pour une solution rapide.

Question 4 – (15 points). Implantez la forme spéciale `define-type` dans sa forme la plus simple (aucun héritage, aucun attributs de champs et aucun prédicat pour tester le type). Utilisez les vecteurs pour représenter les enregistrements. Par exemple la définition de type

```
(define-type point x y)
```

devrait être expansée en

```
(begin
  (define make-point vector)
  (define point-x (lambda (obj) (vector-ref obj 0)))
  (define point-x-set! (lambda (obj val) (vector-set! obj 0 val)))
  (define point-y (lambda (obj) (vector-ref obj 1)))
  (define point-y-set! (lambda (obj val) (vector-set! obj 1 val))))
```

Vous pouvez supposer l'existence de ces fonction utilitaires :

```
(define (symbol-append . symbols)
  (string->symbol
   (apply string-append
          (map symbol->string symbols))))

(define (range i j)
  (if (<= i j)
      (cons i (range (+ i 1) j))
      '()))
```

Question 5 – (20 points). En Scheme les formes spéciales `or` et `letrec` sont dites *dérivées* car on peut les exprimer à l'aide de formes plus simples de Scheme. Donnez les règles de transformation pour chacune de ces formes. Vos règles peuvent être exprimées avec des définitions de macro, ou bien dans le style suivant :

```
(let ((V E) ...) E0) => ((lambda (V ...) E0) E ...)
```

Assurez-vous d'avoir un ensemble de règles qui couvre tous les cas.

Question 6 – (20 points). Soit la définition de fonction Scheme en style direct suivante :

```
(define (append-flatten t r)
  (cond ((null? t)
         r)
        ((pair? t)
         (append-flatten
          (car t)
          (append-flatten (cdr t) r)))
        (else
         (cons t r))))
```

En vous basant sur cette définition, dérivez une définition pour la fonction `append-flattenk` qui effectue le même calcul mais en style CPS.