Un tout petit compilateur

1 Description

Vous pouvez récupérer à partir de
`http://perso.ens-lyon.fr/daniel.hirschkoff/PCo/`
un "compilateur minimal", qui prend en entrée des expressions arithmétiques construites à partir de constantes entières, opérations + et ∗ et parenthèses, et produit en sortie un fichier SPIM dont vous pouvez lancer l'exécution. Voir le fichier `README` contenu dans l'archive (que vous d'ecompresserez à l'aide de la commande `tar zxvf`) pour comprendre comment utiliser ce compilateur.

2 Description de ce qui est attendu

Il vous est demandé de partir de ce compilateur et de l'enrichir afin de prendre en compte les extensions suivantes:

- Traitement de la soustraction et de la division. Cette extension est élémentaire, il faudra notamment se débrouiller pour sortir élégamment si une division par zéro est déclenchée.
- Traitement des conditionnelles: ajouter les expressions de la forme

  `if` e1 `=` e2 `then` e3 `else` e4 `endif`

  au langage.

  Il faudra bien sûr que le compilateur soit en mesure de gérer des expressions comportant plusieurs conditionnelles.
- Traitement de la déclaration et l'utilisation de fonctions simples. Un programme pris en entrée par le compilateur sera de la forme:

  ```
  def f1(x1,y1) = e1;
  def f2(x2,y2) = e2;
  ⋮
  def fk(xk,yk) = ek;
  run e.
  ```

  Les noms des fonctions et de leurs paramètres ne sont bien sûr pas imposés. Les `e1,..., ek, e` sont des expressions (sans définition de fonction, mais potentiellement avec appels de fonction). Chaque fonction a deux arguments, et il peut y avoir une liste vide de déclarations de fonctions.

  Les extensions à faire au compilateur pour traiter les fonctions devront comporter:

  - une phase d'analyse statique, où le compilateur s'arrête en signalant une erreur s'il y a l'utilisation d'une variable non déclarée;
  - la gestion du passage des arguments aux fonctions; pour cela, on demande **deux** versions du compilateur:
    - la première où les valeurs des paramètres sont passées sur la pile;
    - la seconde où l'on utilise par convention les registres `$a1` et `$a2` pour passer les arguments aux fonctions.

Nul besoin de trop s'arracher les cheveux pour fournir élégamment les deux versions du compilateur: vous pourrez si vous voulez indiquer dans le fichier `README` quel point du fichier il faut modifier pour passer d'une version à l'autre. En revanche il sera bon de programmer cette alternative de manière élégante, afin que les parties traitant des deux modes de passage d'arguments soient localisées dans le source du compilateur.

Signalons pour résumer que chacune des extensions décrites ci-dessus fait intervenir des modifications dans les phases d'analyse lexicale et syntaxique, ainsi que dans la génération de code. Une nouvelle phase, d'analyse statique, est également nécessaire pour la gestion des fonctions.

3 Instructions

Le langage Caml est de fait imposé, et le travail sera à faire individuellement (un compilateur par étudiant). Vous devrez envoyer par mail à `daniel.hirschkoff@ens-lyon.fr` une archive contenant:

- le source de votre compilateur, *qui compile*;
- un fichier README expliquant ce que vous avez fait;
- un fichier avec des exemples que vous avez testés *(et qui tournent!)*.

La date limite de réception du tout est fixée au

16 février à 23h59