Compilateur Pascal avec modules
troisième partie (v1.0) : production de code

à rendre avant le lundi 7 décembre 10 heures

Le but de cette dernière partie est de produire un code exécutable, étant donné un fichier Pascal, une fois les analyses syntaxiques et sémantiques passées avec succès.

1 Sémantique

1.1 Pascal traditionnel

On donne ici les principaux aspects de la sémantique du Pascal :
- Pascal est un langage strict : toutes les opérandes ou arguments de fonctions sont évalués. Dans une expression, l'ordre d'évaluation des sous-expressions n'est pas spécifié.
- La seule exception au point précédent est l'évaluation *paresseuse* des expressions e_1 `and` e_2 et e_1 `or` e_2 : pour l'opérateur `and` (resp. `or`), e_2 n'est évaluée que si e_1 est vrai (resp. faux).
- Passage des arguments : quel que soit leur type, les arguments sont passés par référence lorsque le mot clé `var` est présent, par valeur sinon.
- Dans une boucle `for` x `:=` e_1 `to` e_2 `do` i, les expressions e_1 et e_2 ne sont évaluées qu'une seule fois. Aucune évaluation de i n'est faite si la valeur de e_2 est strictement inférieure à celle de e_1.
- La procédure `writeln` affiche chacun de ses arguments sur une ligne différente (c'est un point sur lequel on diffère légèrement du Pascal). Les booléens seront affichés `true` et `false` respectivement.

Pour tous les autres points de la sémantique du Pascal, on vous renvoie aux tests que vous pourrez effectuer avec un vrai compilateur Pascal ou à la littérature.

Les primitives graphiques sont compilées vers les primitives graphiques de la machine virtuelle de manière immédiate.

1.2 Compilation des modules

On se propose ici de compiler les modules par *défonctorisation*. Cette opération consiste à remplacer toute application de foncteur $F(M_1, \ldots, M_n)$ par le module obtenu en substituant dans le corps du foncteur F les arguments formels de F par les arguments effectifs $M_1, \ldots, M_n$. Au final, il ne reste que des modules, mais plus de foncteurs. Les modules peuvent alors être ignorés dans le processus de compilation, car ils ne participent qu'à la structuration de l'espace de noms. Au prix du renommage de certains identificateurs, on peut même supprimer les modules eux-mêmes, pour ne conserver qu'une suite linéaire de déclarations de variables et de fonctions.

Ainsi, on peut supprimer les foncteurs dans le programme suivant

```
program p;
module Double(X : sig procedure f (var x: integer); end) := struct
  procedure f (var y: integer);
  begin
    X.f (y);
    X.f (y)
  end;
end;
module X2 := struct
  procedure f (var x: integer);
  begin
    x := x*x
  end;
end;
module X4 := Double(X2);
module X16 := Double(X4);
begin end.
```

pour obtenir le programme suivant :

```
program p;
module X2 := struct
  procedure f (var x: integer);
  begin
    x := x*x
  end;
end;
module X4 := struct
  procedure f (var y: integer);
  begin
    X2.f (y);
    X2.f (y)
  end;
end;
module X16 := struct
  procedure f (var y: integer);
  begin
    X4.f (y);
    X4.f (y)
  end;
end;
begin end.
```

En renommant les trois procédures `f` on obtient enfin le programme suivant :

```
program p;
procedure f1 (var x: integer);
begin
  x := x*x
end;
```

```
procedure f2 (var y: integer);
begin
  f1 (y);
  f1 (y)
end;
procedure f3 (var y: integer);
begin
  f2 (y);
  f2 (y)
end;
begin end.
```

La phase de défonctorisation pourra être insérée entre l'analyse sémantique et la production de code.

2 Production de code

2.1 Langage cible

La machine abstraite à pile servant de langage cible est une légère extension de la machine décrite et utilisée dans le cours. Elle est détaillée, ainsi que les interprètes qui en sont fournis, dans le document annexe *Machine virtuelle pour le projet de compilation.*

2.2 Représentation des valeurs

Toutes les valeurs sont allouées sur la pile, à l'exception des constructeurs non constants qui sont alloués sur le tas.

2.2.1 Types primitifs

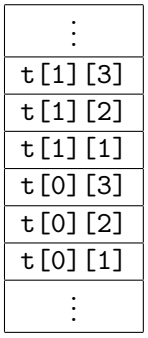
Les entiers sont représentés de manière immédiate par les entiers de la machine virtuelle. Les booléens sont représentés par des entiers, avec la même convention que la machine virtuelle : 0 représente `false` et 1 représente `true`.

2.2.2 Tableaux

Les tableaux sont représentés par l'empilement de leurs éléments dans l'ordre lexicographique croissant des indices. En particulier, un tableau multi-dimensionnel tel que

```
t : array [0..1] of array [1..3] of integer;
```

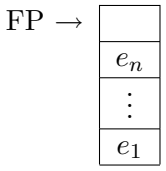
sera représenté de la manière suivante :



2.3 Procédures

Une procédure est compilée en un morceau de code appelé par un `call`. L'appelant empile successivement les arguments e_1, ..., e_n. Au retour, l'appelant se charge de dépiler ces valeurs.

L'appelé se retrouve donc dans la situation (générale) suivante :



et il alloue ensuite au dessus de FP la place nécessaire aux variables locales de son code.

3 Travail demandé

Adapter votre compilateur pour qu'il effectue la production de code. Votre compilateur devra reconnaître les deux options existantes `-parse-only` et `-type-only` (cette dernière indiquant désormais de s'arrêter juste après la phase d'analyse sémantique).

En l'absence d'erreur lexicale, syntaxique ou sémantique, la production de code ne doit pas échouer, et le code doit être produit dans le fichier `fichier.vm` si le fichier source s'appelle `fichier.pas`.

Votre code devra comprendre au moins un fichier `compile.ml` contenant les principales fonctions de production de code. L'introduction d'autres fichiers devra être justifiée dans le rapport.

Modalités de remise de votre projet. Votre projet doit se présenter sous forme d'une archive tar compressée (option "z" de tar), appelée *vos_noms*`.tgz` qui doit contenir un répertoire appelé *vos_noms* (exemple : `dupont-durand.tgz`). Dans ce répertoire doivent se trouver les *sources* de votre programme (ne donnez pas les fichiers compilés). Quand on se place dans ce répertoire, la commande `make` doit créer votre compilateur, qui sera appelé `compilo`. La commande `make clean` doit effacer tous les fichiers que `make` a engendré et ne laisser dans le répertoire que les fichiers sources.

Votre projet doit également s'accompagner d'un rapport (5 pages maximum) expliquant les difficultés rencontrées et les solutions techniques adoptées. Le rapport devra être rendu directement dans l'archive contenant les sources de votre projet (format pdf).

Votre projet est à rendre à votre chargé de TP par e-mail[1] au plus tard le **lundi 7 décembre à 10h00**.

[1] `francois.bobot@lri.fr`, `delphine.longuet@lri.fr`, `louis.mandel@lri.fr`