Université de Strasbourg

Département d'Informatique
Master 1 Informatique

Compilation — Travaux Pratiques n° 3
Génération de code intermédiaire

Pour ce TP, on utilise une grammaire permettant d'écrire des affectations, des instructions conditionnelles et des boucles, ainsi que des suites (non vides) d'instructions :

```
stmt      →  ID ASSIGN expr
          →  WHILE condition DO stmtlist DONE
          →  IF condition THEN stmtlist ENDIF
          →  IF condition THEN stmtlist ELSE stmtlist ENDIF
stmtlist  →  stmtlist stmt
          →  stmt
```

Les expressions sont dans un premier temps limitées aux identificateurs et aux constantes entières :

```
expr  →  ID
      →  NUM
```

Les conditions simples sont formées à partir de constantes, de comparaisons (égalité d'une variable à une valeur) et des opérateurs habituels :

```
condition  →  ID EQUAL NUM
           →  TRUE
           →  FALSE
           →  condition OR condition
           →  condition AND condition
           →  NOT condition
           →  OPAR condition CPAR
```

Le but du TP est de générer des quadruplets pour la totalité du langage. Dans un premier temps, générer le code pour :

1. conditions booléennes
2. instructions conditionnelles
3. programmes complets

Quand tout cela fonctionne (et seulement à ce moment-là), vous pouvez envisager de :

1. permettre qu'une comparaison utilise des expressions quelconques
2. ajouter un ensemble complet d'expressions arithmétiques
3. ajouter une instruction `FOR i ASSIGN expr TO expr DO stmtlist DONE`

Voici la démarche à suivre :

1. Écrivez la grammaire avec des actions vides. Donnez priorités et associativités à tous les opérateurs. Simplifiez les conditions pour qu'elles n'utilisent que `TRUE` et `FALSE`, et remplacez l'instruction d'affectation par un symbole terminal artificiel.
2. Définissez les types de symboles de la grammaire : il y a d'une part les types des symboles terminaux (chaîne de caractères et nombres ici), et d'autre part les types des symboles non-terminaux (`stmt`, `condition`, `expr`, et les différents marqueurs). Vous pouvez utiliser des structures à l'intérieur de l'union.
3. Définissez les structures de données pour les quadruplets, et une liste de quadruplets (vous pouvez utiliser un tableau global dans ce TP). Les seules opérations sont : 1) ajouter un quadruplet, et 2) obtenir la position du prochain quadruplet. Dans un premier temps, les quadruplets ne contiennent pas de nom, seulement les constantes `TRUE` et `FALSE`.
4. Définissez une structure de données pour les listes de positions (listes de quadruplets à compléter). Il est difficile d'utiliser un tableau (chaque symbole aura une ou plusieurs listes), utilisez plutôt une liste chaînée. Les seules opérations sont 1) la création (vide ou avec un élément) et 2) la concaténation de deux listes, et 3) le parcours d'une liste pour compléter ses quadruplets.
5. Écrivez les actions, et produisez du code à partir d'une ou plusieurs instructions.

Dans un second temps, on ajoute des noms dans les expressions et les comparaisons.

1. Définissez une structure de données pour la table des symboles. Ici vous pouvez utiliser une simple liste chaînée, et conserver des pointeurs sur les cellules dans les quadruplets.
2. Introduisez des identificateurs dans les comparaisons et dans l'affectation. Vérifiez à chaque utilisation d'un identificateur qu'il a bien reçu une valeur au préalable.
3. Modifiez votre programme principal pour qu'il affiche à la fin la table des symboles puis la liste de quadruplets.

Dans un troisième temps, vous pouvez ajouter des expressions quelconques formées à partir d'opérateurs arithmétiques, ajoutez d'autres instructions, etc.