Polytech Nice Sophia
SI4 / 2018-2019
Erick Gallesio

Compilation

Feuille 6

Le langage Toy-base

Corrections et extensions

Introduction

Pour réaliser ce TD, vous devez charger l'archive qui contient le compilateur du langage **Toy-base**.

Généralités

- Pour construire le compilateur, il suffit de faire `make` dans le répertoire principal.
- Pour tester le compilateur, lancer la commande `make tests` dans le répertoire principal ou dans le répertoire `tests` .
- Une documentation (très incomplète) se trouve dans le répertoire `doc`

Tests

La distribution qui vous est fourni contient un certain nombre de programmes de tests. Les tests sont de deux catégories:

- les programmes de la forme `ok-*.toy` sont des programmes qui doivent réussir (ne pas afficher `FAILURE` et se terminer avec un code de retour nul)
- les programmes de la forme `fail-*.toy` ne compilent pas et doivent fournir un nombre donné d'erreurs.

Parmi les tests, certains sont actifs (ceux qui testent les fonctionnalités de base), d'autres sont présents mais inactifs et vous serviront au fur et à mesure que vos implémenterez vos extensions. Les tests inactifs sont déclarés dans la variable `IGNORE` du fichier `tests/Makefile` . Pour rendre un test actif, il suffit de le supprimer de la liste `IGNORE` .

La commande `make tests` lance les tests actifs. Pour voir la liste des tests désactivés, vous pouvez faire `make untested` .

Travail demandé

Le travail à réaliser se décompose en deux catégories:

- les **corrections** correspondent à des erreurs dans le compilateur que vous avez récupéré. La correction de ces erreurs vous permettra de vous familiariser avec le code qui vous a été fourni.
- les **extensions** consistent à étendre/améliorer le langage *Toy-base*.

1 Corrections du compilateur existant

1.1 Conditions booléennes

Dans la version distribuée, les conditions que l'on peut mettre dans un `if` ou dans un `while` peuvent être de n'importe quel type. On peut donc écrire (comme en C):

```
if (1)
  print("YES");
else
  print("NO");
```

Or, *Toy* dispose d'un vrai type booléen. On veut donc éviter ce type de construction afin d'obtenir un code un peu plus «propre».

1. Modifiez le compilateur pour qu'il signale une erreur si la condition d'un `if` ou d'un `while` n'est pas de type booléen;
2. Terminer en activant le fichier `fail-bool` dans le répertoire de tests. Vérifier que les anciens tests passent toujours après l'inclusion de ce nouveau test.

1.2 Comparaisons de chaînes de caractères

La version qui vous est distribuée ne gère pas correctement les comparaisons de chaînes de caractères. En effet, le code *toy* suivant

```
if (ch < "OK")
  print("inférieur\n");
```

est traduit plus ou moins à l'identique en C, ce qui permet de comparer des adresses mémoire entre elles (et ce qui n'est probablement pas ce que l'on veut). Le code produit en C devrait plutôt être:

```
if (strcmp(ch, "OK") < 0)
  print("inférieur\n");
```

1. Modifier le compilateur *toy* pour produire du code correct pour les comparaisons de chaînes de caractères;
2. Terminer en activant le fichier `ok-string` dans le répertoire de tests. Vérifier que tests passent toujours après l'inclusion de ce nouveau test.

2 Extensions du compilateur existant

2.1 Ajout des opérateurs `'++'` et `'--'`

La première extension consiste à rajouter les opérateurs `'++'` et `'--'` du langage C. Ces opérateurs auront (à peu près) la même priorité qu'en C: ils seront plus prioritaires que le moins unaire.

1. Modifier le lexical pour accepter les nouveaux opérateurs d'incrémentation/décrémentation;
2. Commencer par introduire les nouvelles règles syntaxiques pour ces opérateurs;
3. Dans la phase d'analyse, vérifier que ces opérateurs sont bien appliqués à des entiers;
4. Produire enfin le code pour les `'++'` et `'--'` **préfixes** et **postfixes**.
5. Terminer en activant les fichiers `ok-plusplus` et `fail-plusplus` dans le répertoire de tests. Vérifier que tous les tests passent toujours.

2.2 Ajout de l'énoncé break

Ajouter l'énoncé `break` au langage *toy*. Vous vérifierez que cet énoncé est bien utilisé à l'intérieur d'une boucle `while` . Dans le cas contraire, vous déclenchez une erreur à la compilation.

Les étapes sont sensiblement identiques à celles de la première extension.

Les tests pour cette extension s'appellent `ok-break1` et `fail-break1` . Vérifiez bien que vous n'avez pas de régression et que tous les tests passent toujours.

2.3 Ajout de l'opérateur puissance

On veut ajouter l'opérateur puissance sur les nombres entiers. Cet opérateur sera dénoté `**` . Ainsi l'expression `2**4` a pour valeur `16` . Cet opérateur est plus prioritaire que tous les autres et il est associatif à droite. Pour réaliser cette extension, vous ajouterez la primitive `_toy_powint` au *runtime* de *Toy*.

Pour tester la nouvelle version du compilateur, vous pourrez utiliser les programmes `ok-power` et `fail-power` qui sont dans `tests` .

2.4 Ajout de l'énoncé for

Cette extension consiste à ajouter l'énoncé `for` de C à *Toy-base*.

1. Commencer par définir la syntaxe du `for` avec:
    - une ou plusieurs des composantes de la "condition" éventuellement absentes;
    - possibilité de déclarer une variable dans la première composante de la "condition";
    - pas d'opérateur "," (les boucles ne pourront pas faire varier simultanément deux indices dans la "condition").
2. Vérifier que le composant du milieu est bien de type booléen.
3. Produire du code qui n'utilise pas `for` mais qui passe par un `while` en C. Cela permettra de déclarer des variables dans un `for` *Toy*, même si le compilateur C utilisé pour compiler le programme produit n'est pas conforme à *C99*.
4. Tester votre extension avec les programmes `ok-for` , `fail-for` et `ok-break2` .

2.5 Ajout du type float

Cette extension consiste à ajouter le type `float` comme un type primitif. Vous ferez particulièrement attention au fait que l'on pourra mixer des entiers et des flottants dans une même expression (celle-ci étant du coup promue au type flottant).

Tester avec le programme `ok-float` .

2.6 Pour finir

Vérifiez (avec `make untested`), qu'il ne reste plus de programme de test inactif.

That's all folks