DLP : Développement d'un langage de programmation

Master STL, Université Pierre et Marie Curie

Année 2020–2021

TME 5 – Boucles et échappements

1 Présentation

Les langages C et Java autorisent la rupture du flot de contrôle normal des boucles grâce aux instructions `break` et `continue` (en Perl, il s'agit de `last` et `next`). Dans le corps d'une boucle, ces instructions ont l'effet suivant :

- `break` sort immédiatement de la boucle en ignorant la suite du corps de la boucle ; la prochaine instruction exécutée est l'instruction suivant immédiatement la boucle ;
- `continue` ignore la suite du corps de la boucle et revient en tête de boucle ; la condition de boucle est à nouveau testée pour déterminer si un tour de boucle supplémentaire doit être effectué.

Ainsi, dans le programme suivant :

```
1  while i < 100 do (
2    if ... then
3        break;
4    if ... then
5        continue;
6    ...
7  )
8
```

l'instruction `break` de la ligne 3 saute directement à la ligne 8, tandis que l'instruction `continue` de la ligne 5 revient en tête de boucle en ligne 1.

Objectif : l'objectif du TME est d'ajouter à ILP2 des nouveaux traits `break` et `continue` permettant l'échappement d'une boucle.

Buts :
- implanter des opérateurs de contrôle avancés ;
- revoir l'extension d'ILP2 (grammaire, AST, interprète, compilateur).

2 Échappements simples

Nous allons ici introduire les deux nouvelles instructions `break` et `continue` qui réalisent les deux types d'échappement décrits ci-dessus sur la boucle courante (i.e., la boucle la plus interne contenant l'instruction d'échappement).

Travail à réaliser. Nous travaillerons dans des *sous-package* de `com.paracamplus.ilp2.ilp2tme5`.

- Écrivez la grammaire ANTLR ILP2 étendue aux deux mots-clefs demandés (vous la nommerez `ILPMLgrammar2tme5`) et générez l'analyseur syntaxique correspondant.
- Identifiez ce que la grammaire ne permet pas de contraindre, et réfléchissez à des exemples de programmes respectant la grammaire mais n'ayant pas de sens. Notez bien que cette question est importante car de tels programmes devront être rejetés par votre implantation, pour l'interprétation comme pour la compilation.
- Écrivez des programmes de test utilisant les nouvelles fonctionalités.

— Implantez la compilation de ces instructions en C. Notez que, lors de la compilation en C, une instruction `break` ou `continue` du langage se traduit par un `break` ou un `continue` en C ; le schéma de compilation est donc très simple ! Ce ne sera pas forcément aussi facile pour l'interprète.

— Implantez l'évaluation de ces instructions dans l'interprète.

— Vérifiez que votre implantation se comporte correctement sur vos programmes de test et que vos changements ne cassent pas l'exécution des programmes existants.

— Ajoutez vos tests à l'intégration continue (fichier `.gitlab-ci.yml`), faites un *push* et vérifiez que vos tests fonctionnent correctement sur le serveur GitLab.

3 Boucles nommées

En Java, les instructions `break` et `continue` ne sont pas limitées à sortir ou passer à l'itération suivante de la boucle la plus interne. Il est possible de s'échapper de plusieurs niveaux de boucles à la fois. Pour cela, nous plaçons une étiquette `x:` en tête de boucle, où `x` est un nom choisi par le programmeur. Les instructions d'échappement, `break x` et `continue x`, prennent alors en argument un nom d'étiquette pour indiquer de quelle boucle nous devons sortir. Cette possibilité n'existe pas dans le langage C. Voici un exemple simple, utilisant des étiquettes `b1` et `b2` :

```
1  b1: while i <= 100 do
2  b2:   while j <= i do
3           if i+j <= 50 then break b1;
4  x = i + j
5
```

Ici, `break b1` sort de la boucle externe, d'indice `i`, et non de la boucle la plus interne, d'indice `j`. L'exécution saute donc directement à la ligne 4. Notez que `break b1` n'a de sens que pour une instruction contenue dans la boucle étiquetée `b1:` Elle n'aurait pas de sens, par exemple, à la ligne 5, puisque la boucle `b1` n'englobe pas cette ligne.

Travail à réaliser.

— Définissez une nouvelle grammaire pour exprimer ces nouveaux traits.

— Écrivez un nouveau programme montrant l'utilisation de ces nouvelles fonctionnalités. Préciser les sorties désirées.

— Ici encore, réfléchissez à des exemples de programmes acceptés par la grammaire mais dont l'interprétation et la compilation doivent échouer.

— Implantez ces instructions dans l'interprète Java.

— Implantez la compilation de ces instructions en C.

— Testez vos implantation, en local d'abord puis sur le serveur GitLab en intégration continue.

4 Rendu

Comme pour les TME précédents, vous effectuerez un rendu en vous assurant que tout le code développé a été envoyé sur le serveur GitLab (*push*) dans votre *fork* d'ILP2. Vous vous assurerez que les tests d'intégration continue développés lors de ce TME ont été configurés (fichier `.gitlab-ci.yml`) et fonctionnent sur le serveur. Vous ajouterez un tag « rendu-initial-tme5 » en fin de séance, puis un tag « rendu-final-tme5 » quand le TME est finalisé.