IFIPS – Cycle Ingénieur 2e année Informatique
Projet de compilation 2006–2007

Compilateur J- -

troisième partie : production de code

Le but de cette dernière partie consiste à produire du code exécutable correct à partir d'un fichier J- - (cf. section 2), pour la machine virtuelle utilisée en cours. Une version exécutable de la machine virutelle est disponible à l'adresse `www.lri.fr/~filliatr/ifips/`. Une fois votre compilateur réalisé, vous devrez écrire un programme en J- - pour découvrir le labyrinthe qui a été caché dans la machine virtuelle (cf. section 3).

1 Sémantique de J- -

Concernant le langage des expressions, la sémantique de J- - est évidente. Comme dans la plupart des langages les opérateurs booléens `and` et `or` sont évalués paresseusement : e_2 n'est pas évaluée dans e_1 `and` e_2 lorsque e_1 est évaluée en `false` ni dans e_1 `or` e_2 lorsque e_1 est évaluée en `true`.

Concernant les aspects objets (héritage, surcharge et redéfinition), la sémantique de J- - est identique à celle de Java.

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 Organisation des données

2.2.1 Représentation des valeurs

Types primitifs. Les valeurs de types primitifs sont représentées de la manière suivante :

- les entiers, par ceux de la machine virtuelle ;
- les chaînes de caractères, par celles de la machine virtuelle ;
- les booléens, par des entiers avec la même convention que la machine virtuelle : 0 représente `false` et 1 représente `true` ;
- les valeurs de type `unit`, par le même entier (0 par exemple).

Objets. Un objet est représenté par une adresse, pointant sur un bloc alloué sur le tas. Ce bloc contient

- un *descripteur de classe* (voir ci-dessous) ;

– les valeurs des attributs de l'objet (variables d'instance).

Les attributs sont organisés de telle manière qu'un même attribut d'une classe C se trouve dans le bloc à la même position pour tout objet de la classe C ou de toute sous-classe de C (organisation dite en *préfixe*). Ceci est possible car l'héritage est *simple*. Ainsi, les deux classes `A` et `B` suivantes :

```
class A begin int x; bool b; end
class B inherits A begin string s; end
```

donneront lieu à des objets ayant la forme suivante :

descr. A
x
b

descr. B
x
b
s

Le compilateur maintiendra donc une table donnant, pour chaque classe C et chaque attribut dynamique de C, la position où trouver cet attribut dans un objet de la classe C.

La valeur `null`. Elle est représentée par une adresse sur le tas, différente de toute autre adresse d'objet. Le plus simple est de l'allouer dès le démarrage du programme.

La classe prédéfinie `Objet`. On pourra allouer un descripteur pour cette classe (même si cela n'est pas nécessaire).

2.2.2 Descripteurs de classes

Chaque classe est représentée par un descripteur de classe. Il sera par exemple alloué dans la zone des variables globales (c'est-à-dire au fond de la pile et référencé par GP). Il contient :

– le descripteur de sa super-classe ;
– la liste des adresses pour les codes des *méthodes*.

Exactement comme pour la représentation des objets, la liste de ces adresses de codes obéit à une organisation préfixe : l'adresse pour le code pour la méthode `f` doit se trouver au même endroit dans le descripteur de la classe `A` qui la définit et dans celui de la classe `B` qui la redéfinit. Ainsi le code suivant :

```
class A begin
    int f() 0
    string g() "bonjour"
end
class B inherits A begin
    int f() 1
    bool h() true
end
```

donnera lieu à des descripteurs de la forme suivante :

descr. de `A` :

descr. de `Object`
A_f
A_g

descr. de `B` :

descr. de `A`
B_f
A_g
B_h

De même que pour les attributs, le compilateur maintiendra une table donnant pour chaque classe C et chaque méthode de C, la position où trouver le code de cette méthode dans le descripteur de la classe C.

2.3 Schéma de compilation

2.3.1 Méthodes et constructeurs

Les méthodes et les constructeurs sont représentés par des fonctions (i.e. du code appelé par un `call`). L'appelant empile successivement

1. une place pour la valeur de retour (le cas échéant) ;
2. la valeur de l'objet (`this`) ;
3. les arguments e_1, ..., e_n.

Au retour, l'appelant se charge de dépiler ces valeurs (sauf la valeur de retour le cas échéant).

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

FP →	
	e_n
	$\vdots$
	e_1
	`this`
	résultat

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

2.3.2 Ordre des opérations de compilation

On effectuera les différentes opérations nécessaires à la production de code dans l'ordre suivant :

1. Allocation de `null`.
2. Allocation des descripteurs de classe et construction de la table décrivant les objets (indiquant où se trouvent les attributs au sein des objets).

   Cette étape nécessite que les classes soient traitées dans un ordre compatible avec l'héritage (les classes les plus vieilles étant traitées les premières).
3. Compilation de l'expression principale du programme ;
4. Compilation de toutes les fonctions (code des constructeurs, des méthodes et fonctions primitives).

Au final, le code doit donc avoir la structure suivante :

allocation des globaux (`null` et descripteurs)
`start`
code de l'expression principale
`stop`
code des fonctions (constructeurs, méthodes, et fonctions primitives)

3 Le labyrinthe

Un labyrinthe de taille 100×50 a été caché dans le code de la machine virtuelle. Cette partie consiste à écrire un programme en `J--` utilisant les primitives de navigation de la machine virtuelle afin de découvrir le contenu de la case de coordonnées $(99, 49)$. Vous pouvez également utiliser les primitives graphiques de la machine pour afficher ce labyrinthe.

4 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.jmm`.

Modalités de remise de votre projet. Votre projet doit se présenter sous forme d'une archive contenant les *sources* de votre programme (ne donnez pas les fichiers compilés) ainsi qu'un court rapport expliquant les difficultés rencontrées et les solutions adoptées. Il doit être envoyé par e-mail à `Jean-Christophe.Filliatre@lri.fr`.