Compilation, Langages de Programmation

EPITA – Promo 2006 – **Tous documents autorisés** *

Avril 2004 (1h30)

Une copie synthétique, bien orthographiée, avec un affichage clair des résultats, sera toujours mieux notée qu'une autre demandant une quelconque forme d'effort de la part du correcteur.

Les résultats ne doivent pas être balancés comme « évidents », sous peine de disqualification pour tentative de bluffage. Néanmoins, une argumentation informelle mais convaincante, sera souvent suffisante.

1 Typologie des Langages de Programmation

1. Apparier chaque auteur avec son langage :

   a. Alan Kay
   b. Andrew Appel
   c. Bjarne Stroustrup
   d. Denis Ritchie
   e. John Backus
   f. Kristen Nygaard
   g. Niklaus Wirth
   h. Ole-Johan Dahl

   1. Ada 83
   2. C
   3. C++
   4. Pascal
   5. Simula
   6. Smalltalk
   7. Tiger

2. Que sont les multiméthodes ?
3. Citer quelques cas où les multiméthodes offrent une facilité d'implémentation.
4. Citer un langage supportant les multiméthodes.
5. En C++, à quelle technique fait-on appel pour simuler les multiméthodes ?
6. En Eiffel, lorsqu'une méthode est redéfinie, qu'advient-il de ses préconditions ?

   a. elles ne sont pas modifiables
   b. elles peuvent être affaiblies
   c. elles peuvent être renforcées
   d. elles sont librement modifiables

7. En Eiffel, lorsqu'une méthode est redéfinie, qu'advient-il de ses postconditions ?

   a. elles ne sont pas modifiables
   b. elles peuvent être affaiblies
   c. elles peuvent être renforcées
   d. elles sont librement modifiables

*"Tout document autorisé" signifie que notes de cours, livres, annales, etc. sont explicitement consultables pendant l'épreuve. Le zèle de la part des surveillants n'a pas lieu d'être, mais dans un tel cas contacter le LRDE au 01 53 14 59 22.

2 Techniques C++ : “Traits”

1. À quoi sert le mot-clé `typename` ?
2. Que sont les “traits” ?
3. À quoi peuvent servir les “traits” ?
4. Écrire un “traits” qui pour tout type “retourne” son type souche non pointeur, i.e.

   `int` → `int` `int*` → `int` `int****` → `int`

5. Qu’est-ce qu’un objet fonction ?

3 Souvenirs, souvenirs... Analyse Syntaxique

Écrire un fichier `bison/yacc` *complet* qui analyse une liste de zéro ou plusieurs entiers (token `NUM` de type `int`) séparés par des virgules, et à la fin la stocke dans la variable globale `the_list` de type liste d’entiers.

Ce fichier `parselist.yy` doit pouvoir être utilisable *directement* ; prendre garde :
– aux `#include`s nécessaires
– déclarer les variables et fonctions (`yylex`) externes dont on dépend
– définir `void yyerror (const char *)`
– définir les types de valeur dont on aura besoin (`%union`)
– déclarer les types des tokens (`%token`) et des non terminaux (`%type`) nécessaires
– faire une grammaire qui compile sans conflits
– faire une grammaire efficace
– expliquer pourquoi votre réponse est “efficace”.

Ni l’analyseur lexical, ni `main` ne sont demandés.

4 Compilation : Boucles Expressions

On étudie la possibilité de considérer les boucles comme des expressions, et non plus simplement des instructions. On prendra Tiger comme exemple concret de langage souche.

0. Préliminaire : Les `SeqExp` peuvent poser des problèmes de compilation, ce pourquoi on cherchera à les éliminer. Expliquer cependant pourquoi la démarche naïve consistant à remplacer `v := (s; e)` par `s; v:= e` n’est pas toujours correcte (`v` désigne une variable, `s` une instruction, et `e` une expression).
1. Quelles sont les constructions dans l’AST qui sont **en rapport** avec les boucles ?
2. Dans quel cas est-il manifestement difficile pour une boucle non interrompue de retourner une valeur ?
3. Proposer une extension de syntaxe des boucles qui permettrait de spécifier le comportement à tenir dans ces cas-là. On suggère fortement de s’inspirer d’une autre partie du langage qui pose un problème similaire. En particulier, éviter l’introduction de nouveau mot clé.
4. Votre extension est-elle susceptible de provoquer de nouvelles ambiguïtés dans la grammaire ? Si oui, définir la bonne lecture d’une telle phrase ambiguë.
5. Quelle autre fonctionnalité citée en question 1 compromet le concept de “valeur” d’une boucle ?
6. Proposer, si nécessaire, une modification de la syntaxe Tiger de cette fonctionnalité pour supporter les boucles expressions.
7. Proposer des règles de typage pour toutes ces constructions.
8. Proposer une implémentation très économe (mais naïve) du support des boucles expressions dans un compilateur comme `tc`. En quoi est-elle naïve ?