Rattrapage Compilation

EPITA – Promo 2006

Juillet 2004 (1h30)

Aucun document autorisé[1]

Une copie concise, bien orthographiée, dont les résultats sont clairement exposés, sera toujours mieux notée qu'une copie qui aura demandé une quelconque forme d'effort de la part du correcteur.

1 Polymorphismes

1. Qu'est-ce que le polymorphisme ? On s'attachera à donner une définition générale, et non pas liée à une forme particulière de polymorphisme, ou bien à un langage spécifique.
2. Soient `a` et `b` deux variables Tiger. Donner une expression Tiger dont le code effectif (c'est-à-dire celui qui sera exécuté) dépend du type de `a` et `b`.
3. Comment le compilateur détermine-t-il la version du code à générer ?
4. Dans l'expression C `sin (51)`, quel autre mécanisme de polymorphisme intervient ?
5. Voyez-vous en Tiger une valeur qui puisse avoir plusieurs types ?
6. Les langages orientés objets introduisent une forme de polymorphisme. Comment l'appelleriez-vous ?
7. À quel mot-clé est-il associé en C++ ?
8. Sur quelle forme de polymorphisme s'appuie largement STL ?

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 ?

1. « Aucun document autorisé » signifie que ni notes de cours, livres, annales, etc. ne sont consultables pendant l'épreuve.

3 Compilation : Boucles Expressions

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

1. Quelles sont les trois 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 ?

Annexe : Grammaire Tiger

We use Extended BNF, with '[' and ']' for zero or once, '(' and ')' for grouping, and '{' and '}' for any number of repetition including zero.

```
program ::= exp

exp ::=
  # Literals.
    'nil'
  | integer
  | string

  # Array and record creations.
  | type-id '[' exp ']' 'of' exp
  | type-id '{'[ id '=' exp { ',' id '=' exp } ] '}'

  # Variables, field, elements of an array.
  | lvalue

  # Function call.
  | id '(' [ exp { ',' exp }] ')'

  # Operations
  | '-' exp
  | exp op exp
  | '(' exps ')'

  # Assignment
  | lvalue ':=' exp

  # Control structures
  | 'if' exp 'then' exp ['else' exp]
  | 'while' exp 'do' exp
  | 'for' id ':=' exp 'to' exp 'do' exp
  | 'break'
  | 'let' decs 'in' exps 'end'

lvalue ::= id
  | lvalue '.' id
  | lvalue '[' exp ']'
exps ::= [ exp { ';' exp } ]
decs ::= { dec }
dec ::=
  # Type declaration
   'type' id '=' ty
  # Variable declaration
  | 'var' id [ ':' type-id ] ':=' exp
  # Function declaration
  | 'function' id '(' tyfields ')' [ ':' type-id ] '=' exp

# Types
ty ::= type-id
   | '{' tyfields  '}'
   | 'array' 'of' type-id
tyfields ::= [ id ':' type-id { ',' id ':' type-id } ]
type-id ::= id

op ::= '+' | '-' | '*' | '/' | '=' | '<>' | '>' | '<' | '>=' | '<=' | '&' | '|'
```