ESIAL 1ère année

Examen - Samedi 8 février 2003
Durée : 2h00 (10h00 - 12h00)

Examen d'Informatique de Base - module IB2 -

Le barême est donné à titre indicatif.
Aucun document autorisé.

Exercice 1 : Héritage et liaison dynamique (8 points)

On considère la classe abstraite A et les classes B, C qui en héritent.

```
deferred class A
feature
   f(x:INTEGER):INTEGER is
deferred
end
end

class B
inherit A
feature
   f(x:INTEGER):INTEGER is
      do
        Result:=2*x
      end
end

class C
inherit A
feature
   f(x:INTEGER):INTEGER is
      do
        Result:=x+5
      end
end
```

Question 1
Qu'affiche l'exécution de la classe TEST1 donnée ci-dessous ?

```
class TEST1
creation
   make
feature
   make is
     local
        b1 : B
        c1 : C
     do
        !!b1
        !!c1
        io.put_integer(b1.f(3))
        io.put_new_line
        io.put_integer(c1.f(4))
     end
end
```

Question 2
On ajoute dans la classe A les primitives suivantes

```
est_un_b : BOOLEAN is deferred end

g(autre:like Current):A is
   require
     autre/=Void
   deferred
   ensure
     Result=Current or Result=autre
   end

h(autre:A):A is
   require
     autre/=Void
   deferred
   ensure
     Result=Current or Result=autre
   end
```

Parmi les types A, B et C, quels sont les types statiques autorisés pour x, y et z dans l'instruction x:=y.h(z) ? Quels peuvent être les types dynamiques de y et z dans cette instruction ? Justifier les réponses.

Question 3
Répondre aux mêmes questions qu'en 2) pour l'instruction x:=y.g(z).

Question 4
Sans modifier la classe A, écrire les primitives `est_un_b`, g et `h` des classes B et C de telle sorte que l'exécution de la classe TEST2 se fasse sans violer aucune assertion (on rappelle que les assertions se trouvant entre check et end doivent rendre true)

```
class TEST2
creation
    make
feature
    make is
      local
        b1, b2 : B
        c1, c2 : C
    do
        !!b1
        !!c1
        check -- est_un_b
          b1.est_un_b
          not c1.est_un_b
        end
        !!b2
        !!c2
        check -- g
          b2=b1.g(b2)
          c1=c1.g(c2)
        end
        check -- h
          b1=b1.h(b2)
          b1=b1.h(c1)
```

```
            c1=c1.h(c2)
            b1=c1.h(b1)
          end
      end
end
```

Question 5

On ajoute maintenant dans la classe A la primitive suivante :

```
k(autre:C):A is
  deferred
  ensure
    Result=Current or Result=autre
  end
```

Sans autre modification dans la classe A, donner la définition de k dans les classes B et C et une nouvelle définition de h dans C utilisant k, de telle sorte qu'on n'utilise aucun test, aucune tentative d'affectation inverse (?=), et que le programme ne viole pas le principe de cohérence globale.

Exercice 2 : Backtracking (8 points)

On considère les chaînes de caractères construites sur l'alphabet $\{a, b, c\}$. On dit qu'une chaîne est acceptable, s'il n'y a pas dans cette chaîne deux sous-chaînes adjacentes égales. Par exemple *b*, *ac*, *cbabc*, *abcbabc* sont acceptables, *cc* n'est pas acceptable à cause des sous-chaînes *c* égales et adjacentes, *cbabacbc* n'est pas acceptable (présence des sous-chaînes égales et adjacentes *ba*). Etant donné un entier n, le but de cet exercice est de trouver la première chaîne (par ordre alphabétique) acceptable de longueur n.

L'algorithme que l'on propose consiste en partant de la chaîne vide à faire croître systématiquement la chaîne courante en lui ajoutant un caractère $\{a, b, c\}$ à chaque étape. Comme il est inutile de faire croître des chaînes non acceptables, il est nécessaire à chaque étape de vérifier que la chaîne obtenue est acceptable.

Dans la suite de l'exercice on impose les choix d'implantation suivants :

```
class ABC

feature
  chstk : STRING
  n : INTEGER
```

L'attribut `chstk` de type *STRING* permettra de stocker les caractères des chaînes que l'on veut générer. L'attribut `n` correspond à la longueur de la chaîne cherchée.

Question 1

On note `chstk`$[u..v]$ la sous-chaîne de `chstk` composée des caractères compris entre les indices u et v. Ecrire une fonction *Eiffel* **récursive** `egal (i, j, k, l: INTEGER): BOOLEAN`, qui teste si les deux sous-chaînes, `chstk`$[i..j]$ et `chstk`$[k..l]$ sont égales.

Question 2

Lors de la génération des chaînes acceptables, on tente d'ajouter à chaque étape un des trois caractères $\{a, b, c\}$ à la fin de la chaîne courante, (qui est une chaîne acceptable), de façon à obtenir une chaîne acceptable. Pour vérifier qu'une telle chaîne est acceptable il suffit de comparer les sous-chaînes `chstsk`$[p - 2 * u + 1..p - u]$ et `chstk`$[p - u + 1..p]$ pour toutes les valeurs de u comprises entre 1 et $l//2$, où p est l'indice maximum de `chstk`, l la longueur de `chstk` et // est la division entière.

Ecrire une fonction *Eiffel* `acceptable : BOOLEAN` permettant de tester que la chaîne `chstk` est acceptable.

Question 3

Ecrire une fonction *Eiffel* `valide(c:CHARACTER): BOOLEAN` qui teste si la chaîne obtenue en ajoutant le caractère `c` à la fin de `chstk` est acceptable, la chaîne `chstk` étant inchangée après l'exécution de valide.

Question 4

Ecrire une fonction *Eiffel* `premiere : BOOLEAN` qui retourne *true* lorsque `chstk` contient la première (par ordre alphabétique) chaîne acceptable de longueur n.

Indications : On rappelle quelques primitives de la classe *STRING* :

```
item (i: INTEGER): CHARACTER
     -- Character at position i
upper: INTEGER
     -- Maximum index;
count: INTEGER
     -- String length
add_last (c: CHARACTER)
      -- Append c to string.
remove_tail (n: INTEGER)
      -- Remove n last characters.
```

et quelques primitives de la classe *CHARACTER*.

```
infix ">" (other: CHARACTER): BOOLEAN
      -- Comparison using code.
next: CHARACTER
      -- Give the next character (the following code)
```

Exercice 3 : Récursivité (4 points)

La numérotation en chiffres romains utilise les caractères M,D,C,L,X,V et I qui représentent respectivement les valeurs 1000, 500, 100, 50, 10, 5 et 1. Pour calculer la valeur d'un nombre romain, on fait la somme algébrique des valeurs de tous les caractères, en comptant positivement la valeur d'un caractère si elle est supérieure ou égale à la valeur du caractère suivant (en allant de gauche à droite), en la comptant négativement sinon.

Exemple : MCDLXIV vaut 1000-100+500+50+10-1+5 = 1464.

On suppose donnée la fonction `val : CARACTERE -> ENTIER`, qui donne la valeur des caractères utilisés dans la numérotation romaine, ainsi que les fonctions sur les chaînes de caractères déjà vues en cours et TD :

`premier : CHAINE -> CARACTERE`

`reste : CHAINE -> CHAINE`

`longueur : CHAINE -> ENTIER.`

Question

Donner une définition récursive de la fonction `valeur : CHAINE -> ENTIER`, qui calcule la valeur d'un nombre romain, supposé bien écrit.