p494

MR

2 Comparative Programming languages 2003

Give a brief description of the main syntactic constructs used in Smalltalk (or Squeak) programs, illustrating you answer by explaining the meaning of to following fragment of code:

```
[self isAwake]
whileTrue:
[| item |
 item := self askForCookie.
 (self isCookie: item)
  ifTrue: [self eat: item]
  ifFalse: [self complainAbout:item].
  (self isFull) ifTrue: [self sleep]]
```

[10 marks]

Suggest how you implement in Smalltalk (or Squeak) a binary tree in which each node contains an integer and pointers to two or fewer other nodes of the same kind. [4 marks]

Outline the code you would use (a) to construct this kind of tree, and (b) to sum all the integers in a given tree. [6 marks]

ANSWER NOTES:

Smalltalk/Squeak are explicitly covered in the couse.

This question is basically easy for people who have spent a little time learning the language and how to use it. It will be difficult for others.

Writing an answer in Java would attract little credit.

```
!Object subclass: #Tree
  instance variableNames: 'value left right'
  classVariableNames: ' '
  poolDictionaries: ' '
  category: 'Demo'!

!Tree class methodsFor: 'creation'!
new:
  ^ super new initialize
!
mk: aValue left: aTree1 right: aTree2
  | res |                    " a local variable "
  res := Tree new.           " create a Tree node "
  res value: aValue.         " initialise its three fields "
  res left:  aTree1.
  res right: aTree2.
  ^ result                   " return the node "
!
test
  | t l r |
  l := Tree mk: 12 left: nil right: nil.
  r := Tree mk: 20 left: nil right: nil.
```

```
  t := Tree mk: 15 left: l right r.
  Transcript show: (t sum asString)
!!

!Tree  methodsFor: 'access'!
value
  ^ value
!
value: aValue
  value := aValue
!
left
  ^ left
!
right
  ^ right
!
left: aTree
  left := aTree
!
right: aTree
  right := aTree
!!
!Tree methodsFor: 'initialisation'!
initialise
  value := 0.
  left := nil.
  right := nil
!!

!Tree  methodsFor: 'calculation'!
sum
  | res |
  res := 0 .
  (left = nil)
    ifTrue: [res := res + left.sum].
  (right = nil)
    ifTrue: [res := res + right.sum].
  ^ res
!
```