UNIVERSITY OF DUBLIN

TRINITY COLLEGE

Faculty of Engineering and Systems Sciences
Department of Computer Science

B.A.(Mod.) Computer Science
Senior Freshman Examination

Trinity Term 1999

2BA2 – Programming Techniques

Monday 31st May **Exam Hall** **14.00 - 17.00**

Dr. Hugh Gibbons

Attempt FOUR questions

(In presenting programs explain clearly the design of the Eiffel code)

Qs 1

i) Present an Eiffel function

```
is_ordered(a : ARRAY[G]; L, H : INTEGER) : BOOLEAN
        require
                a /= void and L <= H;
                a.lower <= L and  H <= a.upper
        ensure
                -- ( All i | L ≤ i < H : a.item(i) ≤ a.item(i+1))
```

that will check whether an array segment, a[L..H], is ordered.

ii) Present an Eiffel routine

```
Search(a:ARRAY[G]; L,H:INTEGER; x:G) is
        require
                Ordered:  Is_Ordered(a,L,H)
        ensure
                --(found → x = a.item(index)) & (¬found → x ∉ a[L..H]
```

that binary searches an array section, a[L..H], for an item, x.

iii) Assume we have an attribute, a:ARRAY[G], in a class.
Present a routine

```
reverse(L,H:INTEGER)
        require
                Ordered:  Is_Ordered(a,L,H)
        ensure
                -- ( All i | L ≤ i < H : a.item(i) ≥ a.item(i+1))
```

that reverses the items in array segment, a[L..H]. Ensure that none of the items in the array are overwritten.

Qs 2

A Matrix M has a Saddle Point iff for some position (i,j), M(i,j) is the minimum of row i and maximum of column j.

a) Show that all saddle points have the same value,
   i.e. if M(i,j) and M(s,t) are Saddle Points then M(i,j) = M(s,t).

b) Assume we have the following classes for VECTOR and MATRIX with short forms,

```
class interface
      VECTOR [G -> COMPARABLE]
creation
      make
feature
      arr_copy (a: ARRAY [G])
      -- copy an array, a, into current vector

      item (i: INTEGER): G

      make (n: INTEGER)

      max_index: INTEGER
      -- an index for maximum item

      min_index: INTEGER
      -- an index for minimum item

      put (x: G; i: INTEGER)

      size: INTEGER

end -- class VECTOR
```

```
class interface
      MATRIX [G -> COMPARABLE]
creation
      make
feature

      cols: INTEGER -- # columns

      item (i, j: INTEGER): G

      make (r, c: INTEGER)

      max_row: VECTOR [G]
      -- vector of maximums for each row

      min_row: VECTOR [G]
      -- vector of minimums for each row

      put (x: G; i, j: INTEGER)

      put_row (a: ARRAY [G]; i: INTEGER)

      rows: INTEGER -- # rows

      transpose: like Current

end -- class MATRIX
```

i) Present an Eiffel routine
   one_saddle (m: MATRIX [INTEGER])
   that will find the location of a saddle point in matrix, m.

ii) Present an Eiffel routine
   all_saddle (m: MATRIX [INTEGER])
   that will find the location of all saddle points in matrix, m.

Qs 3

Assume we have classes, LIST_BAG and PAIR with short forms

```
class interface LIST_BAG [G]

    add(x : G)
    -- Add x, maybe again

    join(other : LIST_BAG[G])
    -- join to the end of current

    count : INTEGER

    empty : BOOLEAN

    has (x : G) : BOOLEAN

    remove (x : G)

    copy(s: LIST_BAG[G])

    -- traversal routines

    item : G        -- item at cursor

    start   -- set cursor back to start

    first -- first item in list

    forth   -- move cursor forward

    off : Boolean -- Cursor beyond end?

end -- class LIST_BAG
```

```
class interface PAIR[G]
    first, second : G

    set_first(x :G) is

    set_second(x :G)

end -- PAIR
```

Present an Eiffel class that will provide a routine for sorting a LIST_BAG object via the algorithm for quicksort, i.e. provide routines that will quicksort a list.
Use linked list diagrams to explain the routines.
<u>Suggestion</u>:

Provide a routine/function

partition(s : LIST_BAG[G]; pivot : G) : PAIR[LIST_BAG[G]]

that will partition a list, s, into a pair of lists and a routine/function

quicksort(s : LIST_BAG[G]) : LIST_BAG[G]

that will do the sorting.

Qs 4

A derangement of 1..n is a permutation, p, of 1..n such that $p(i) \neq i$.

a) Write out all the derangements of 1..4.

b) The number of derangements of 1..n can be given by the recursive function

$$Der(1) = 0$$
$$Der(2) = 1$$
$$Der(n) = (n-1)(Der(n-1) + Der(n-2)), \text{ for } n > 2.$$

Present an iterative/non-recursive function that will calculate Der(n).

c) Present an Eiffel routine that will generate all the derangements of 1..n

Qs 5

Assume class interfaces for a Binary Search Tree class, BST, and a BIN_NODE class as follows: (Note: No repeated items in an object of type BST.)

```
class interface
    BST [G -> COMPARABLE]
    -- Binary Search Tree
feature

    empty -- make current empty

    is_empty: BOOLEAN

    inorder: ARRAY [G]
        require
            not is_empty

    add (x: G) -- add x, if not in current.

    array2bst(a:ARRAY[G])
    -- add all items of array, a, to current.
    -- repeat items are not added.

    inorder : ARRAY[G]
    --puts items in tree into an array

    remove (x: G)
        require
            not is_empty

    root: BIN_NODE [G]

    count: INTEGER -- # items in tree

end -- class BST
```

```
class interface
    BIN_NODE [G]
feature

    value: G;

    left: BIN_NODE [G];

    right: BIN_NODE [G];

    value_set (v: G)

    left_set (n: BIN_NODE [G])

    right_set (n: BIN_NODE [G])

    build (v: G; l, r: BIN_NODE [G])

end -- class BIN_NODE
```

a) The routine, add(x : G), which uses an auxiliary routine, insert, is as follows:

```
Add(x:G) is
      do
            if root /= void then
                  insert(x,root)
            else
                  !!root
                  root.build(x,void,void)
                  count := 1
            end
end -- Add
```

Implement the routine,

insert(x : G; t : BIN_NODE[G])

that inserts an item, x, into a tree with root, t, if it is not already there.

b) Implement the routine,

array2tree(a:ARRAY[G])

that adds all the items in the array, a, to the binary search tree, with no repetitions.

c) Implement the auxiliary routine,

inord(t:BIN_NODE[G])

which is used in the routine, inorder, as follows:

```
inorder : ARRAY[G] is
      require
            not is_empty
      do
            !!a_ord.make(1,size)
            count_ord := 1
            inord(root)
            result := clone(a_ord)
      end -- inorder
```

The array, a_ord, and integer, count_ord, are hidden attributes of the class. The routine, inord, inorder traverses the tree, starting at the root, and adds the items of the tree to the array, a_ord.

Qs. 6.

Assume that a Dirrected Acyclic Graph (DAG), D, is stored as an adjacency list.

a) Present an Eiffel routine that will Breadth First Traverse the DAG, D.

b) Present an Eiffel routine that will Toplogical Sort the DAG, D.