

Program 6

CS 141
Winter 1991
Craig A. Rich

Consider a binary search tree whose elements are of type ELEMENT:

```
NAME_CAPACITY: constant := 40;
type ELEMENT is record
    PIN: NATURAL range 0..999_999;
    NAME: STRING(1..NAME_CAPACITY);
    NAME_LENGTH: NATURAL range 0..NAME_CAPACITY;
end record;
```

The elements in the search tree are ordered by increasing PIN number. Create a procedure `program_6.ada` which starts with an empty search tree CLASS and processes a sequence of insertions (I) and locates (L) from an input file of the following form:

```
I 6748Alfred Aho
I 4963Michael A Arbib
I 4576Howard Anton
L 4963
L 8293
I 8293John Bell
L 8293
```

In response to each locate request, output a message. For example, the output produced from the input file above might be

```
Student 4963 is Michael A Arbib
Student 8293 not located
Student 8293 is John Bell
```

The following is an Ada package specification for a binary tree ADT.

binary_tree_.ada

```
generic
  type ELEMENT is private;
package BINARY_TREE is
  type NODE is private;
  type TREE is private;
  EMPTY_TREE: constant TREE;

  function ROOT          (T: TREE)          return NODE;
  function PARENT        (T: TREE; n: NODE) return NODE;
  function RETRIEVE      (T: TREE; n: NODE) return ELEMENT;
  function LEFT_SUBTREE  (T: TREE; n: NODE) return TREE;
  function RIGHT_SUBTREE (T: TREE; n: NODE) return TREE;
  function MAKE_TREE      (X: ELEMENT; LEFT_SUBTREE, RIGHT_SUBTREE: TREE)
                          return TREE;
  procedure REPLACE_LEFT (T: in out TREE; n: NODE; LEFT_SUBTREE: TREE);
  procedure REPLACE_RIGHT (T: in out TREE; n: NODE; RIGHT_SUBTREE: TREE);

  NODE_ERROR: EXCEPTION;
private
  type ELEMENT_NODE;
  type NODE is access ELEMENT_NODE;
  type ELEMENT_NODE is record
    VALUE: ELEMENT;
    LEFT_SUBTREE: TREE;
    RIGHT_SUBTREE: TREE;
  end record;
  type TREE is new NODE;
  EMPTY_TREE: constant TREE := null;
end BINARY_TREE;
```

1. Create the package specification `binary_tree_.ada` exactly as shown above.
2. Create the package body `binary_tree.ada`:
 - `ROOT` returns the root node of the tree `T`. A `NODE_ERROR` exception is raised if `T` is an empty tree.
 - `PARENT` returns the parent node of the node `n` in the tree `T`. A `NODE_ERROR` exception is raised if `n` is the root node of `T`.
 - `RETRIEVE` returns the element attached to node `n` in the tree `T`.
 - `LEFT_SUBTREE` and `RIGHT_SUBTREE` return a subtree of the node `n` in the tree `T`.
 - `MAKE_TREE` returns a tree having a root node with attached element `X` and subtrees `LEFT_SUBTREE` and `RIGHT_SUBTREE`.
 - `REPLACE_LEFT` and `REPLACE_RIGHT` replace a subtree of the node `n` in the tree `T` with a new subtree.
 - A `NODE_ERROR` exception should also be raised in any operation into which a `null` node `n` is passed.
3. Create the procedure `program_6.ada` which processes the insertion and locate requests. Include operations `INSERT` and `LOCATE` to process individual requests.
 - `LOCATE` returns the subtree of `T` whose root node has an attached element with the same `PIN` component as `X`, or an empty tree if no such node exists.


```
function LOCATE (T: TREE; X: ELEMENT) return TREE is ...
```
 - `INSERT` “adds” a new node with attached element `X` as a subtree of the appropriate leaf node of `T`. Any request to insert an element whose `PIN` already exists in the tree should have no effect.


```
procedure INSERT (T: in out TREE; X: ELEMENT) is ...
```
4. Hand in one contiguous printout containing compiler listings of the files `binary_tree_.ada`, `binary_tree.ada`, and `program_6.ada`, and a listing of the output produced by `PROGRAM_6` from input file `[cs.carich]program_6.dat`—in that order.