Eliminating R

As an example of the problem for Inorder Traversal. A recursive version was given before as

```
Inorder(t : BIN_NODE[G]) Qs
do
    Qf t /= void
        Inorder(t.left)
"Process (t.value)"
        Inorder(t.right)
    end
end -- Inorder
```

The non-recursive version is less readable, more complQcated but may be more efficient in both time and space

Also some languages (e.g. Assembler, Fortran, Occam) doV't support recursion and

Version 1: Using an ExplQcit Stack

made explQcit by using a Stack. It is

takes

d trees, takes advantage of the empty links

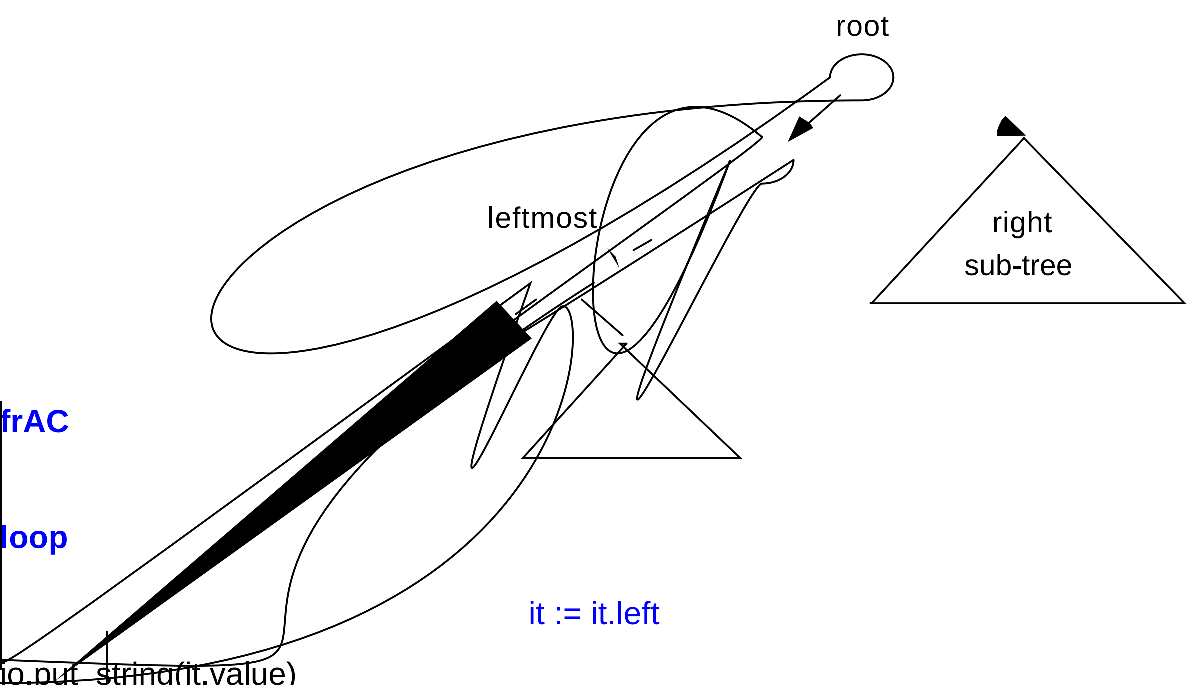
Version 1. Inorder Traversal Using an Explicit Stack

```
        Non_Rec_Inorder(t:BIN_NODE[STRING]) is
                local
                        stS : STACK[BIN_NODE[STRING]]
                        it : BIN_NODE[STRING]
                do
                        !!stk.Uake
                        from
it := t
                        until
it = void and stk.empty
                        loop

                                until
                                        it = void

                                        stk.add(it)

end
it := stk.item
stk.remove

io.put_string("  ")    -- process node
it := it.right
                        end
                end -- Non_Rec_Inorder
```

Strategy of this prograU:

In Inorder traversal, the 'first' node is the left-most node. The program finds the first node, while stacking all the items on the path to the leftmost node. The leftmost node is also stacked but then immediately removed (and processed). We then move to the right node (if any) of this leftmost node and this node is now the root of a (sub)tree.

W e r e p e
stack is empty.



```
frAC

loop

                                        it := it.left
io.put_string(it.value)
```

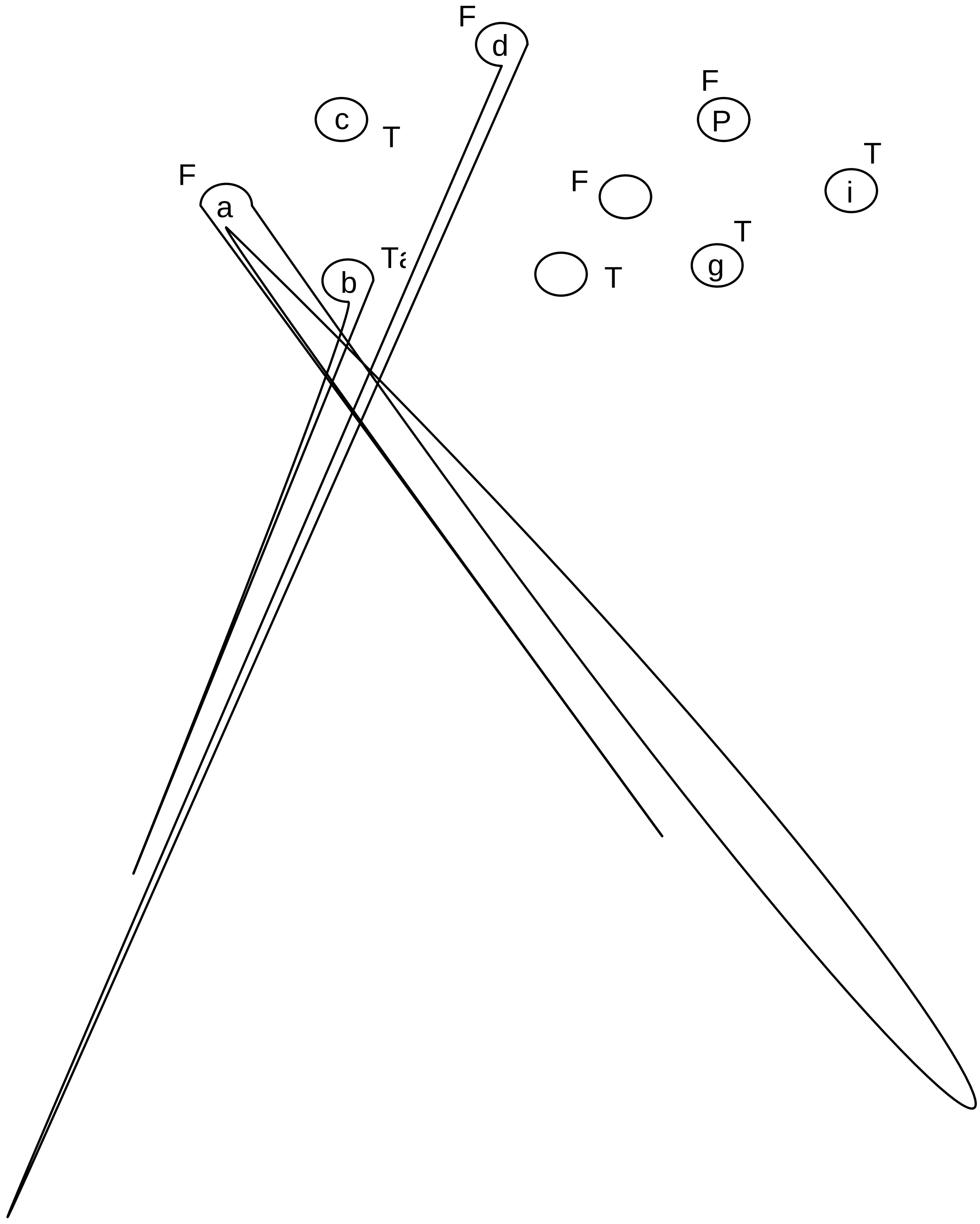
We can test this program in the context of Binary Search Trees by creating a BST and use Inorder to output the nodes. The nodes are printed in alphabetical order, i.e. the nodes will be sorted.

```
class       INORDER_TEST
creation
            make


    make is









                  equaT(io.last_striVg, "quit")
```

[illegible]BST[STRING]

Version 2.   Threaded TreesThreaded Trees

Since tPe context is InWrder Traversal, we consider RQght Threaded Trees.
Each nWde object in a Binary Tree has 2 link attributes, left and right, tPerefWre, in aN node tre
has in-degree 0). TPe otPer N+1 links are unused.
In aTjRQded tree we make tPe right link of each leaf nWde reference (point to) tPe



```
                                                                  next(tn : TH
    require

        s : THREAD_NODE[G]

        -- if right Tink is proper find leftUost of right 'subtree'

                s.left = void

                s := s.left
            end
        eVd
        result := s
    end -- next
```

The function 'next' finds the inorder successor of a node in a threaded tree. Consider the following cases:

- Node, tn, is an internal node with a proper right Tink, i.e. rthread is false. The reference, s, initially goes right aVd when the right Tink is not void it fiVds the leftUost of the right subtree of tn.
- Node, tn, is a leaf, i.e. its right Tink is a thread aVd so rthread is true. Since rthread is true, tn.right is the successor of tn.
-

true. In this case the successor is tn.right which is void. If the node has no successor the function next returns void.

Inorder Traversal

The function, next, has done Uost of the jWb. To implement Inorder we find the 'first' node, the leftUost of the whole tree. StartiVg with the leftUost node we traverse through the threaded tree usiVg the function 'next'.

```
            tn /= void
    local

    do
        s := tn.right
fooid tn.rthread then
```

```
        local
                p : THREAD_NODE[G]
        do      -- Find leftmost node
                frWm



                lWop
                        p  : =  p . l e f t
                end  -- p is at start

                frWm
                until
                        p = vWid
                loop
                        "prWcess nod/F3p"
                        p := next(p)
                end
        end -- Inorder
```

```
    build(v:G; L,R : THREAD_NODE[G]): THREAD_NODE[G] is
                  p.right_set(result) -- right thread to new rWWt nWde


      l o c a l
        p : THREAD_NODE[G]

        !!result

Qf/= vWid then

                frWm -- find rightmost of L
                        p := L
                until                                   result.right_set(R)
                        p.right = vWid
                lWop

                end

                 -- p.rthread_set(true)  -- already 13.6 to true
                result.left_set(L)
        end
        -- TinS in right subtree
        9 R@=4vWid then
```

```
                        p := t
result.value_set(v)
[illegible] the next function
```

<u>Comment</u>:

In buildQng a threaded tree, we don’t have the property that ift is a threaded tree then so is t.left and t.right.
This property is useful fWr designQng recursive programs.

In a threaded tree, t.right is a threaded tree, but d t.left is nWt. In buildQng the fulT threaded tree we change the right most lQnk of the origQnal d t.left.

ConvertQng a BQnary Tree to a Threaded Tree