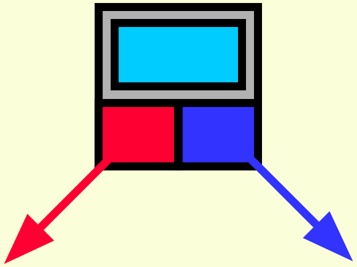
Traversing a Binary Tree
Binary Search Tree Insertion
Deleting from a Binary Search Tree

Traversing a Binary Tree

Inorder Traversal

The Scenario

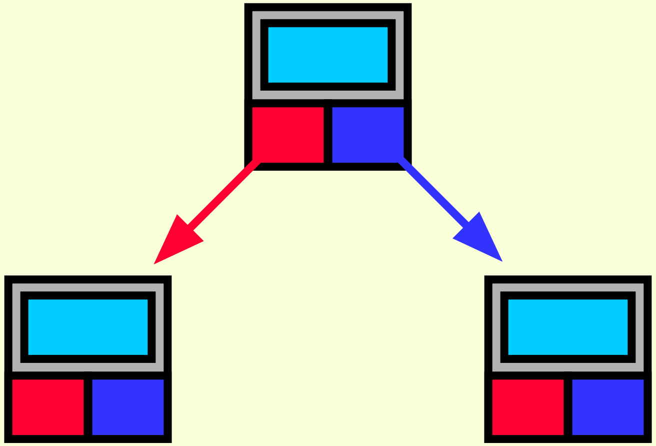


- Imagine we have a binary tree
- We want to traverse the tree
  - It’s not linear
  - We need a way to visit all nodes
- Three things must happen:
  - Deal with the entire left sub-tree
  - Deal with the current node
  - Deal with the entire right sub-tree

Outline of In-Order Traversal

- Three principle steps:
  - Traverse Left
  - Do work (Current)
  - Traverse Right

- Work can be anything
- Separate work from traversal



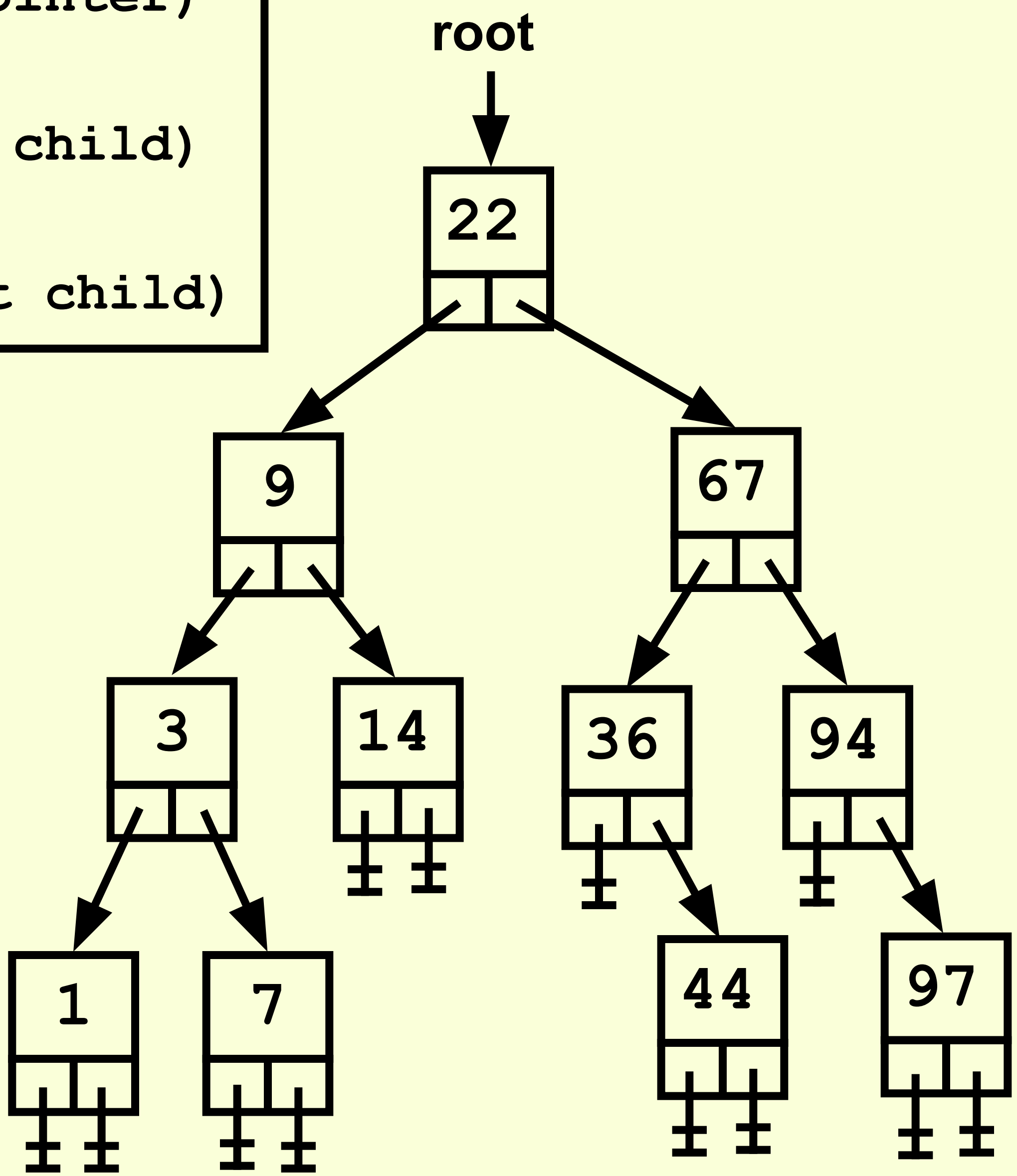
- Traverse the tree “In order”:
  - Visit the tree’s left sub-tree
  - Visit the current and do work
  - Visit right sub-tree

In-Order Traversal Function

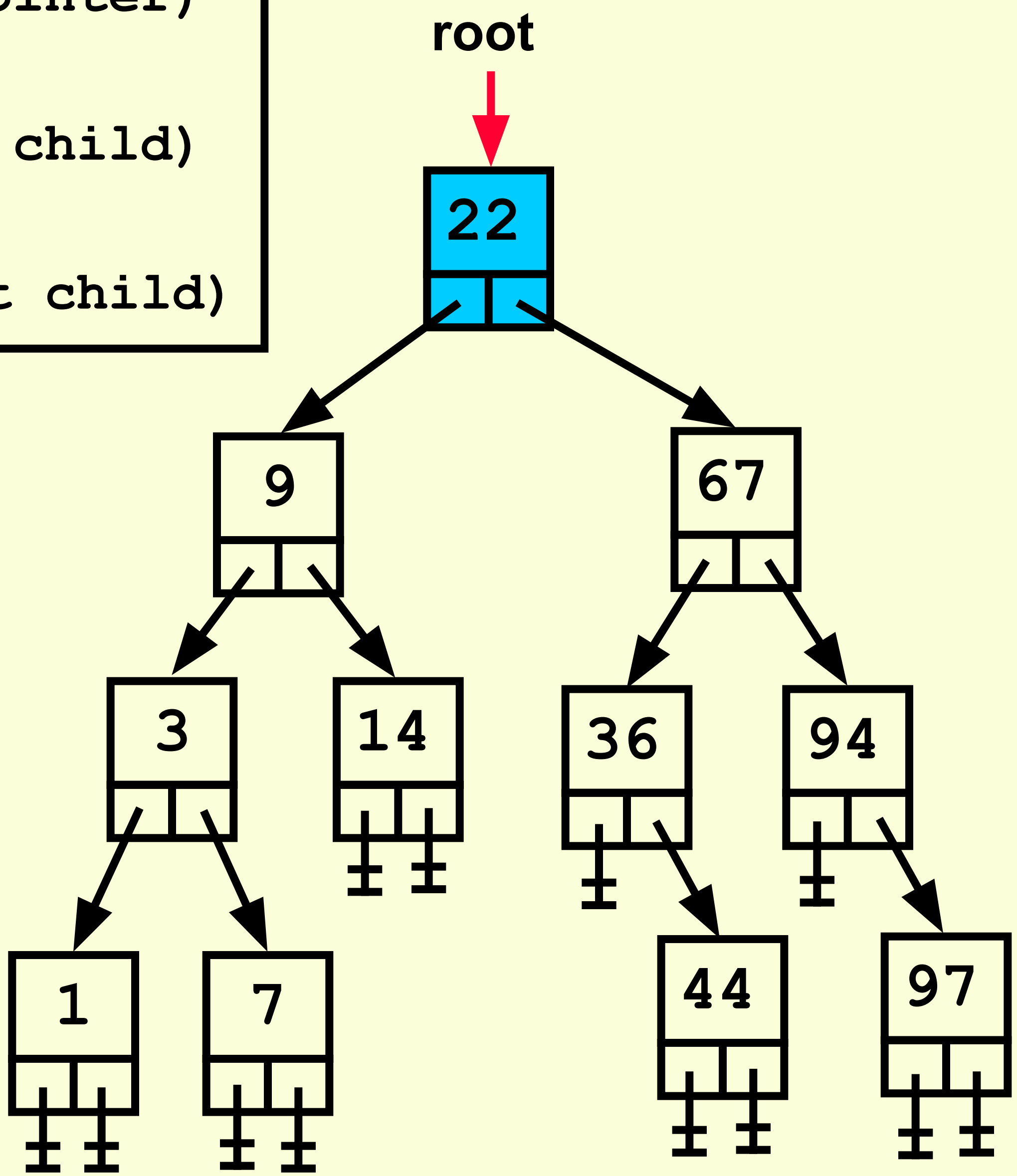
```
In_Order(cur Ptr to a Tree_Node)

if( cur != NIL ) {
    In_Order( cur->left_child )
    Do_Something( cur->data )
    In_Order( cur->right_child )
  }
```

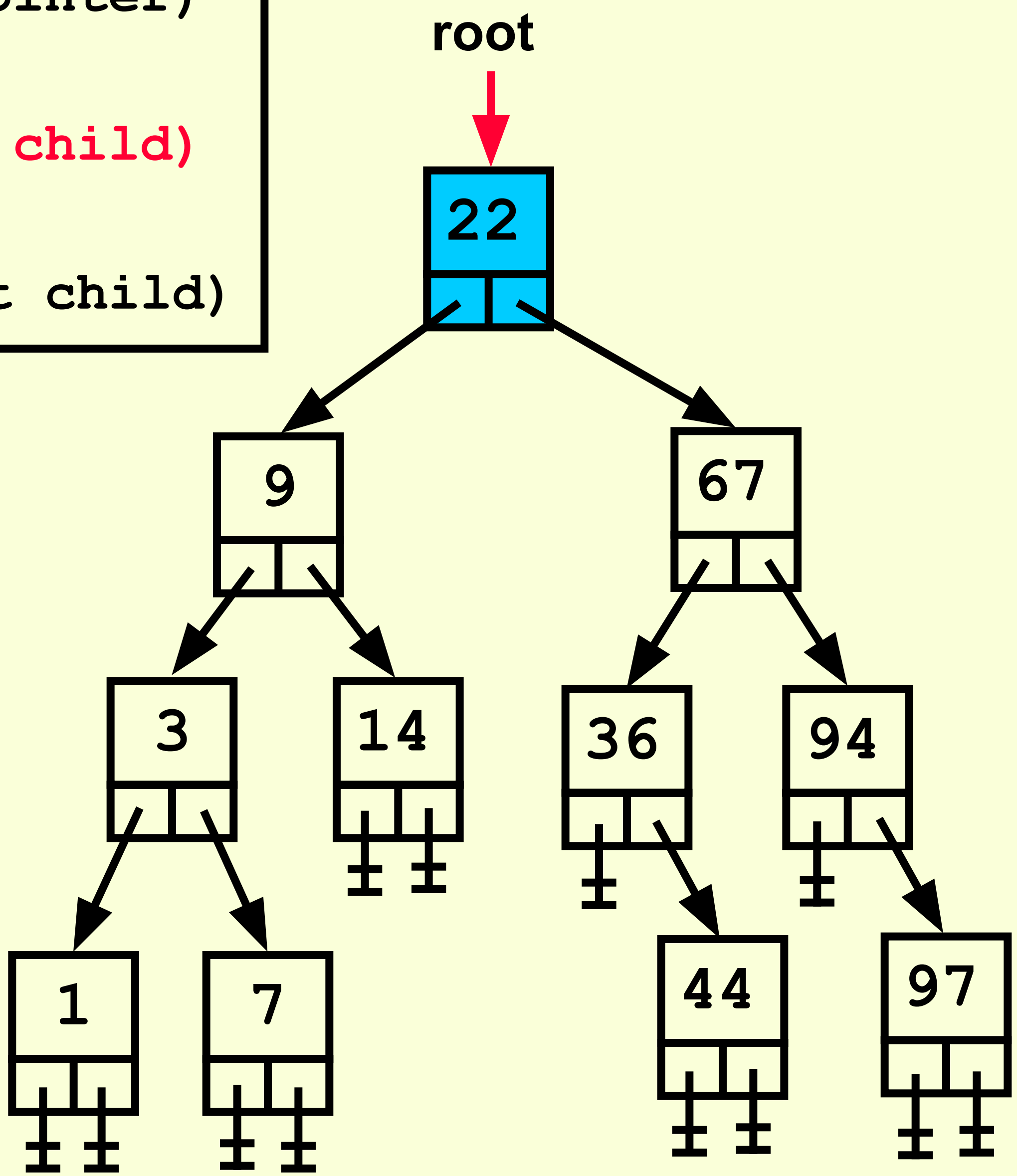
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



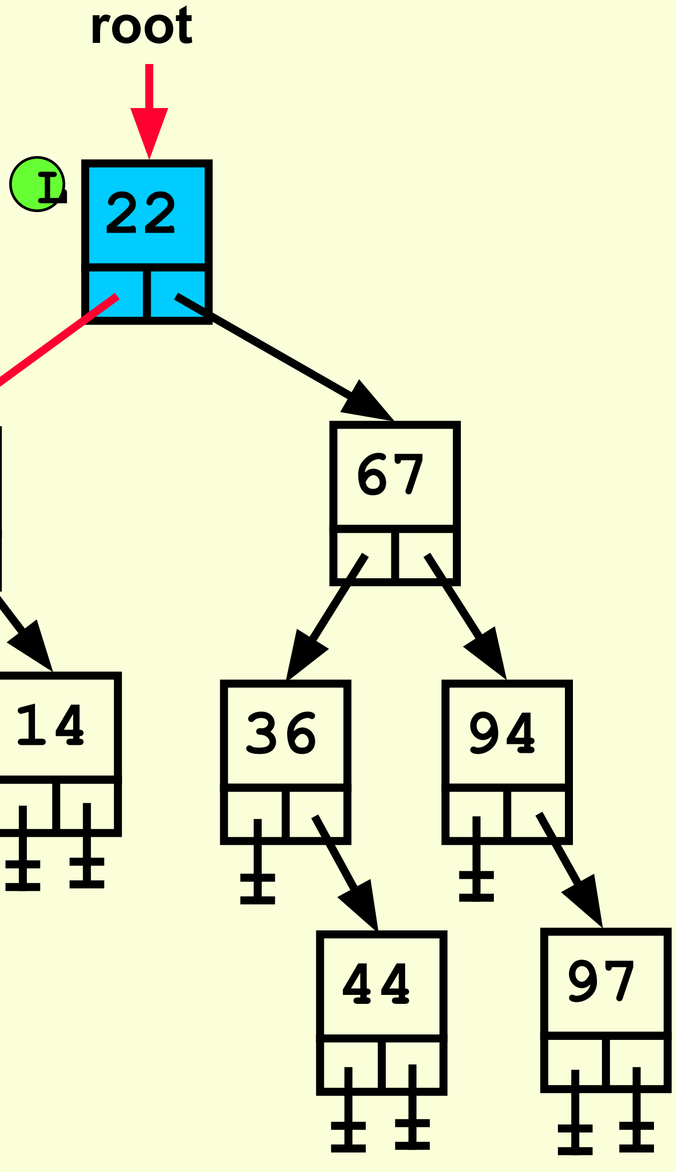
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



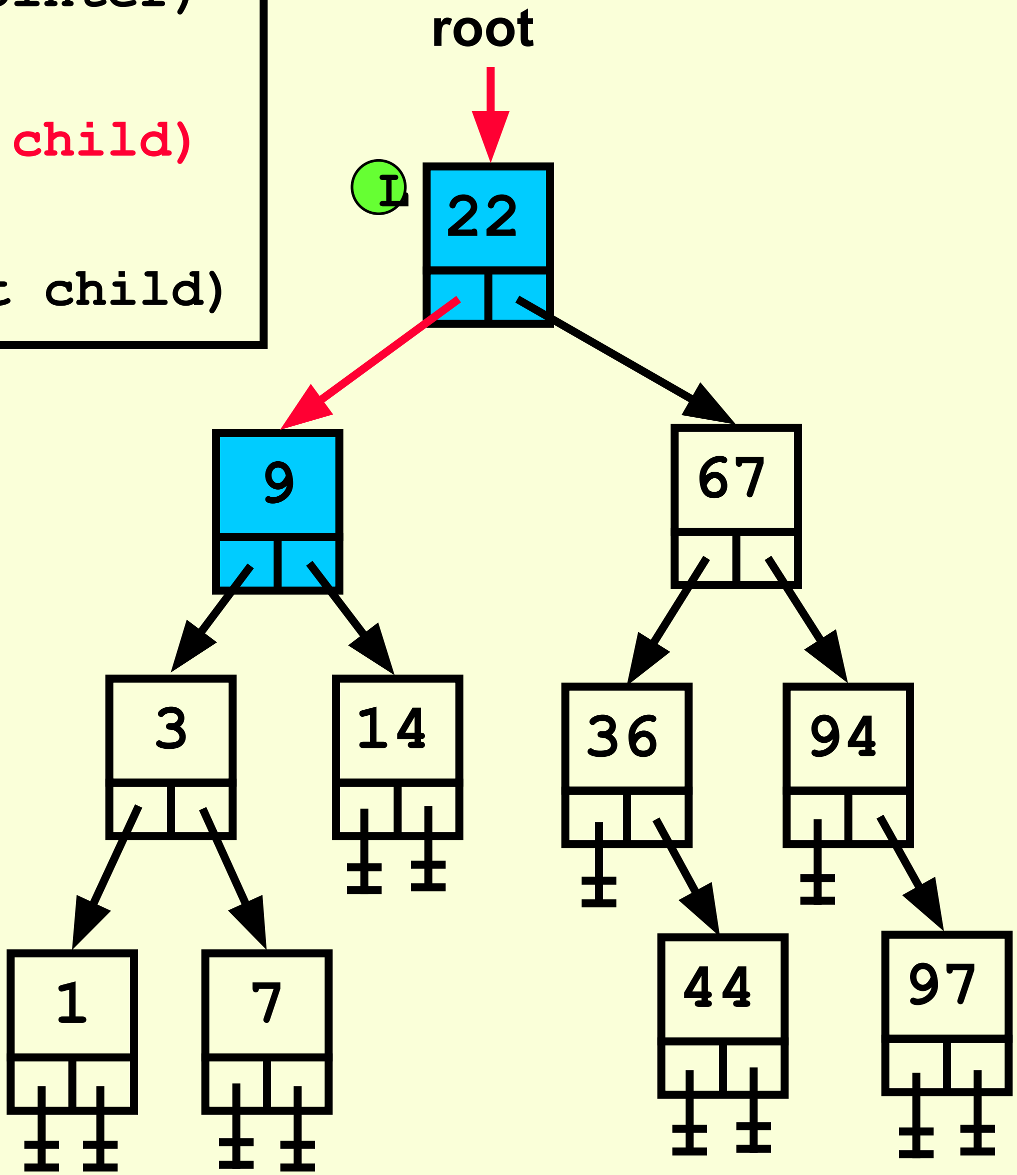
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



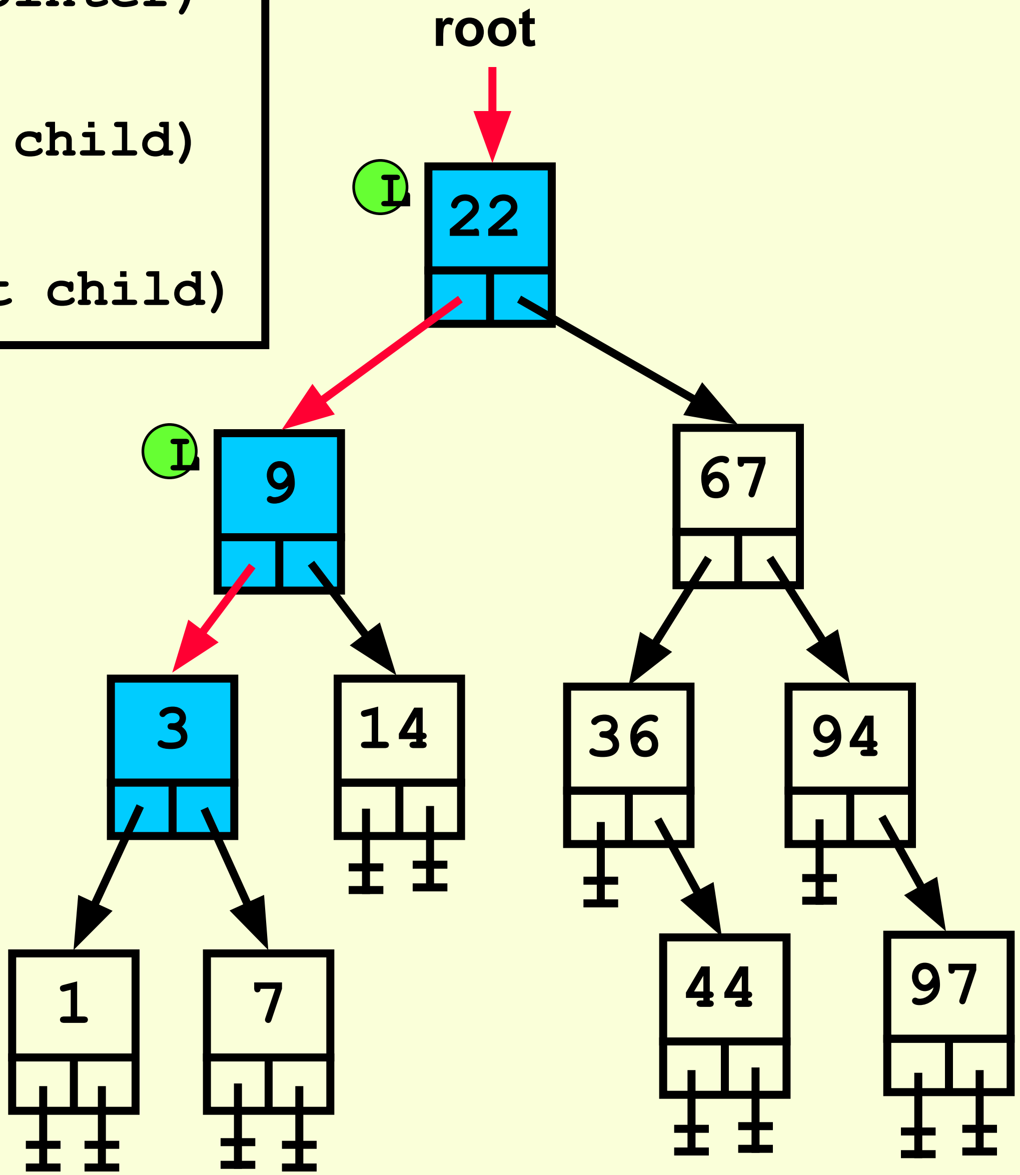
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



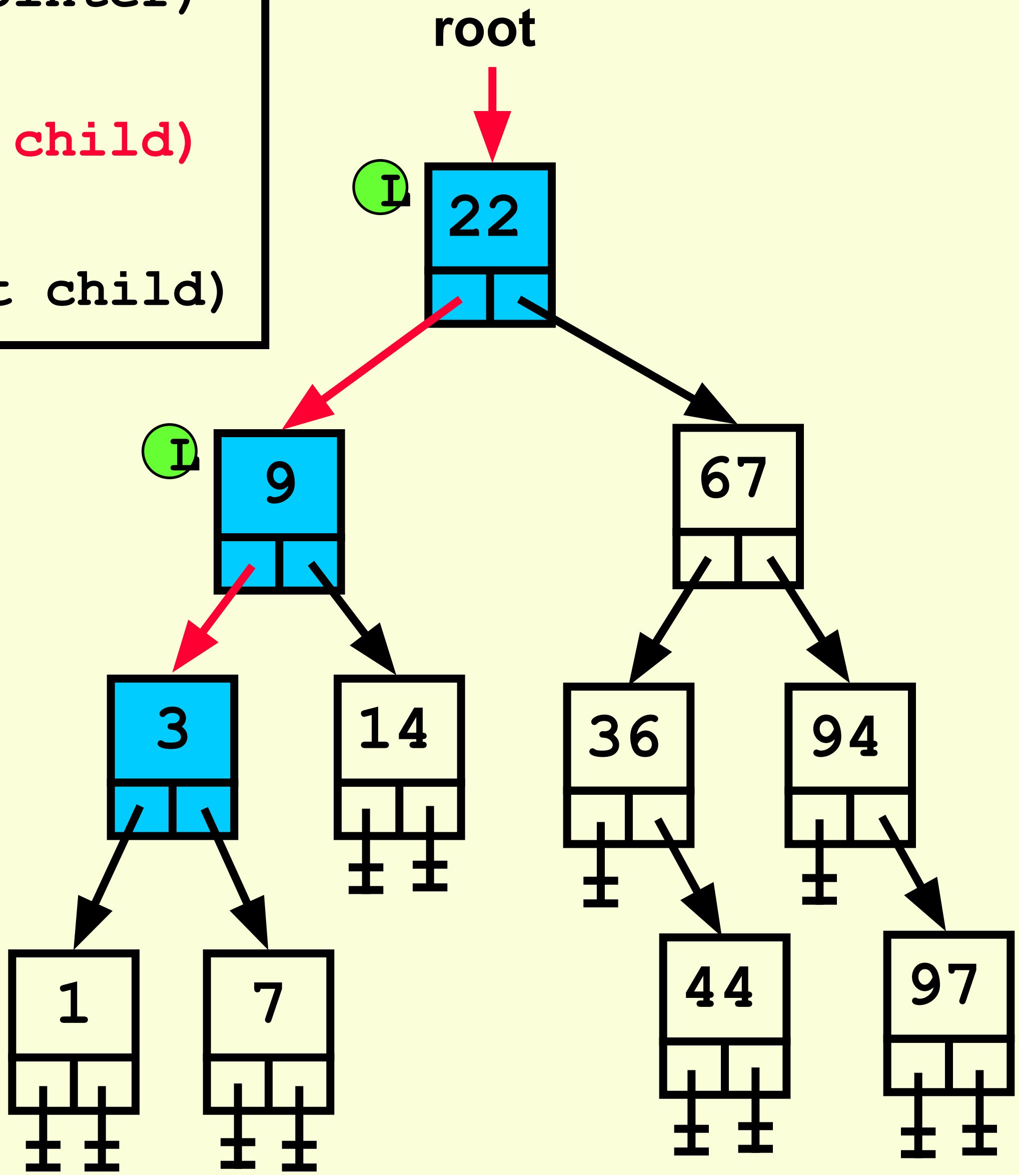
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



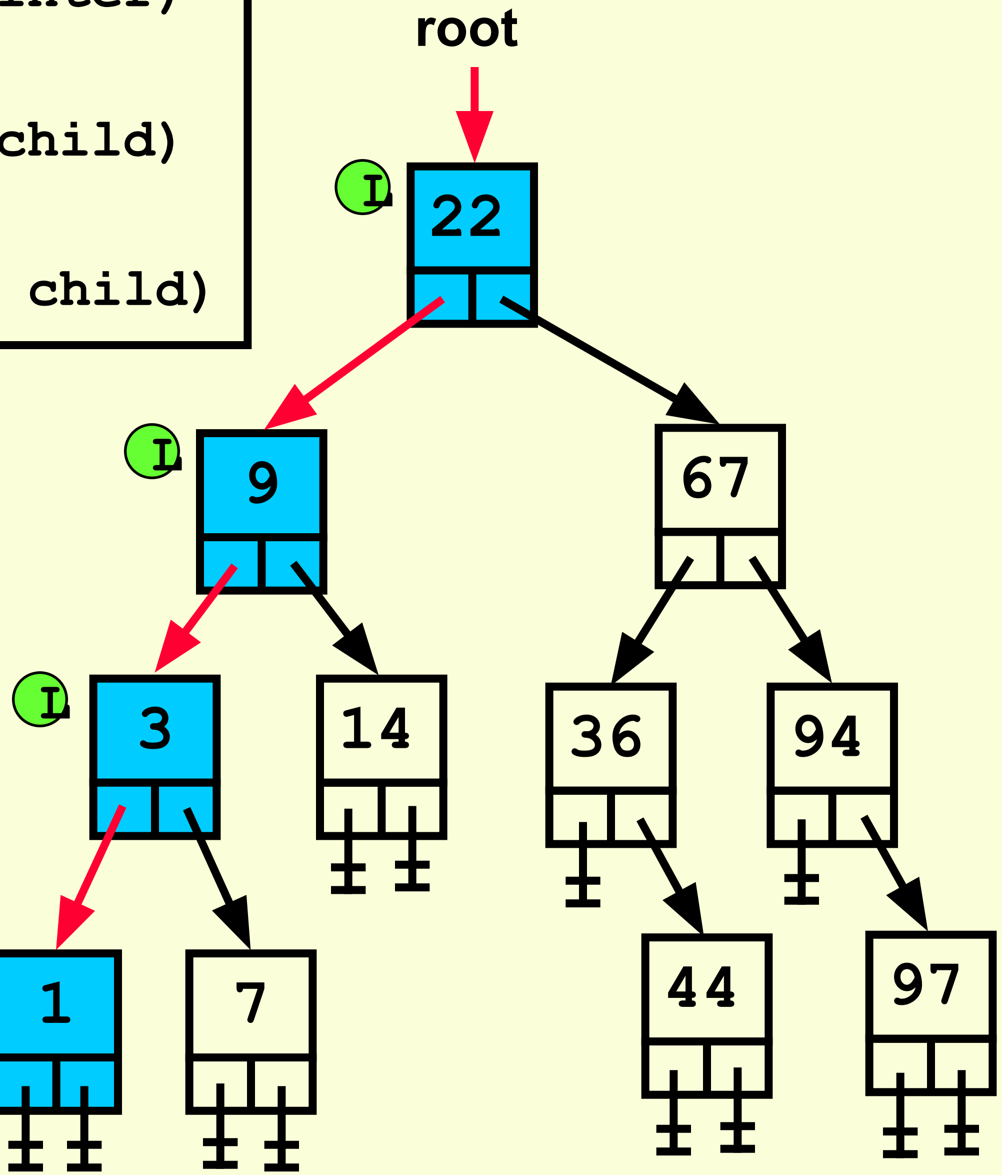
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



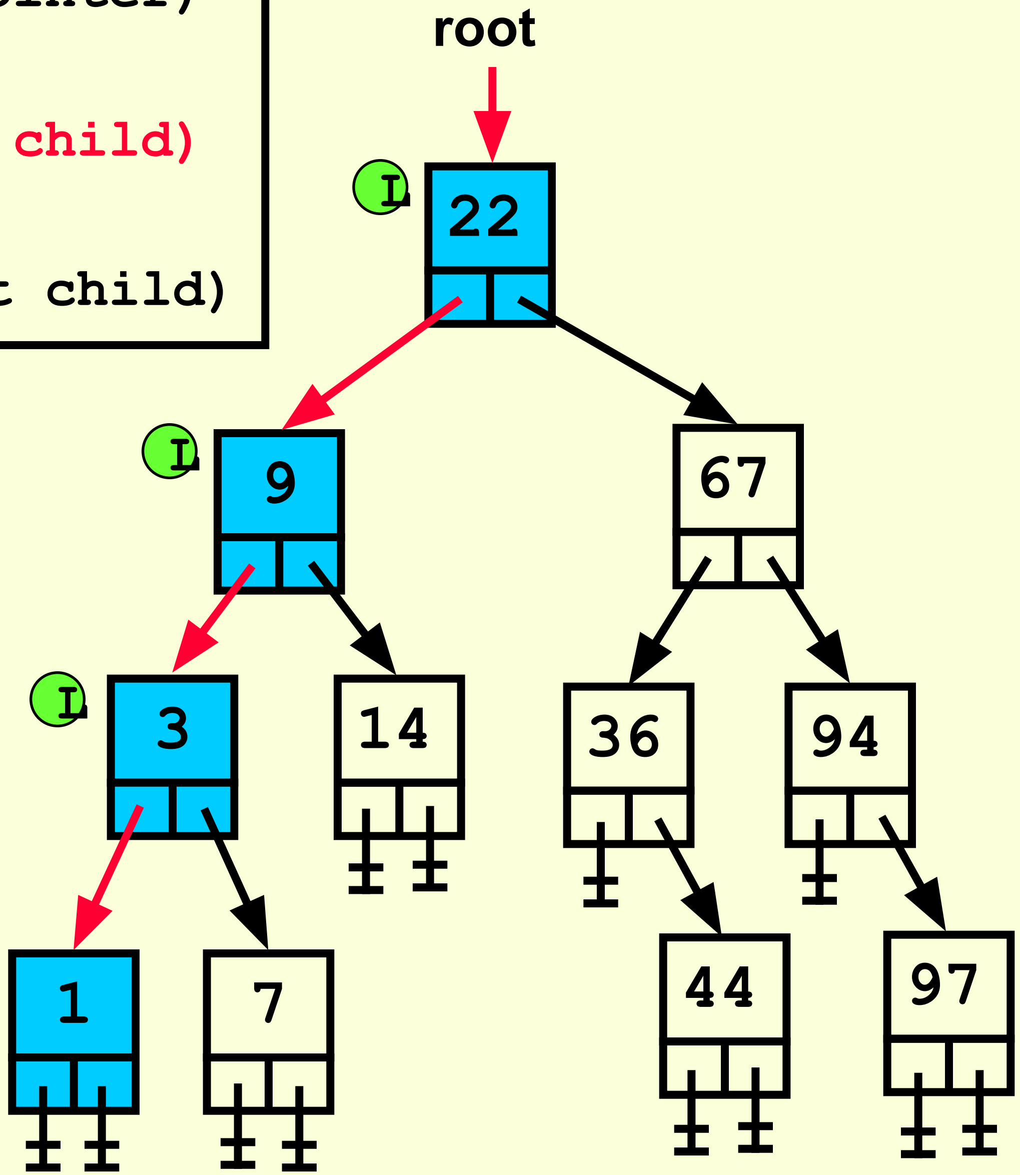
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



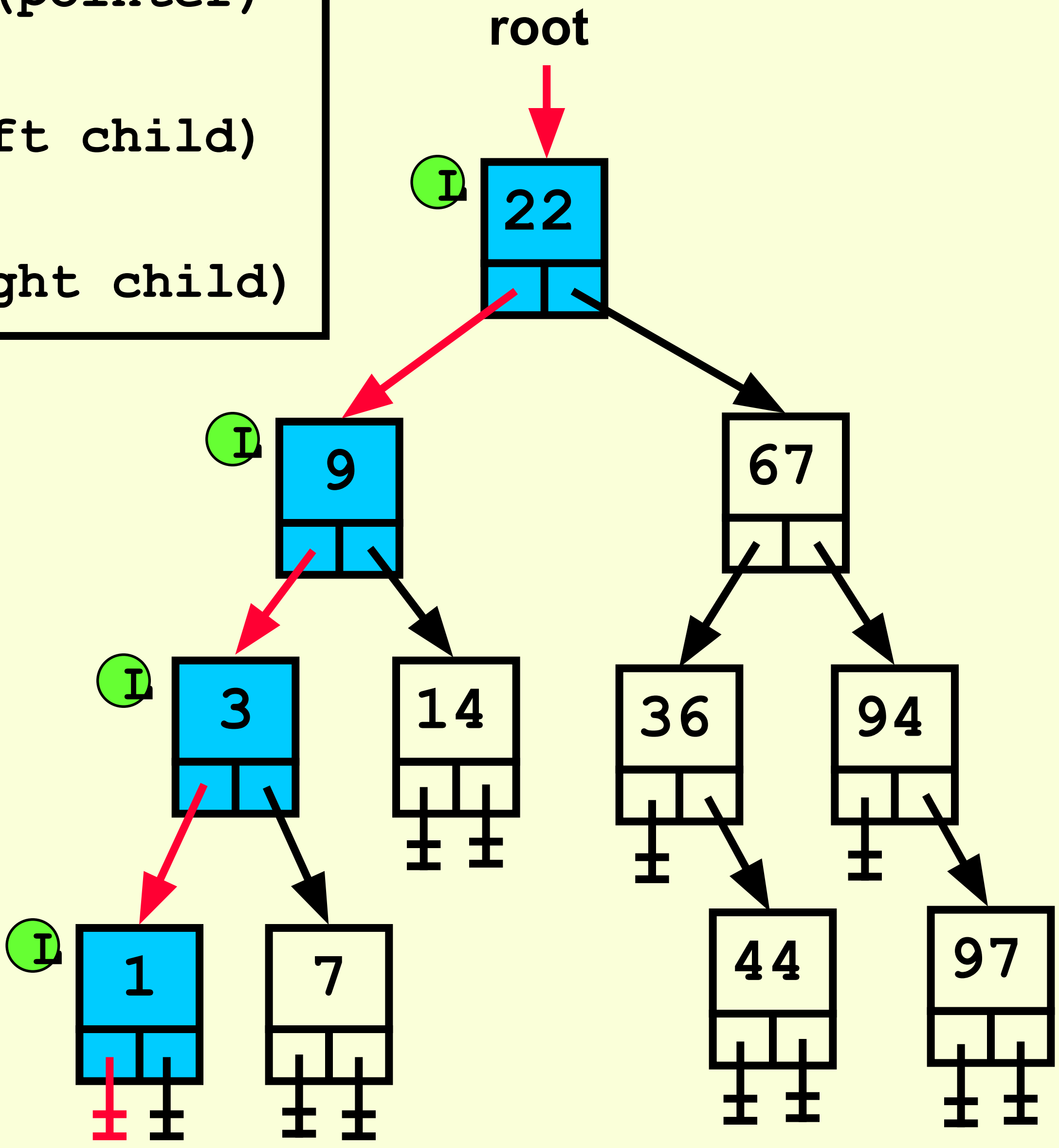
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



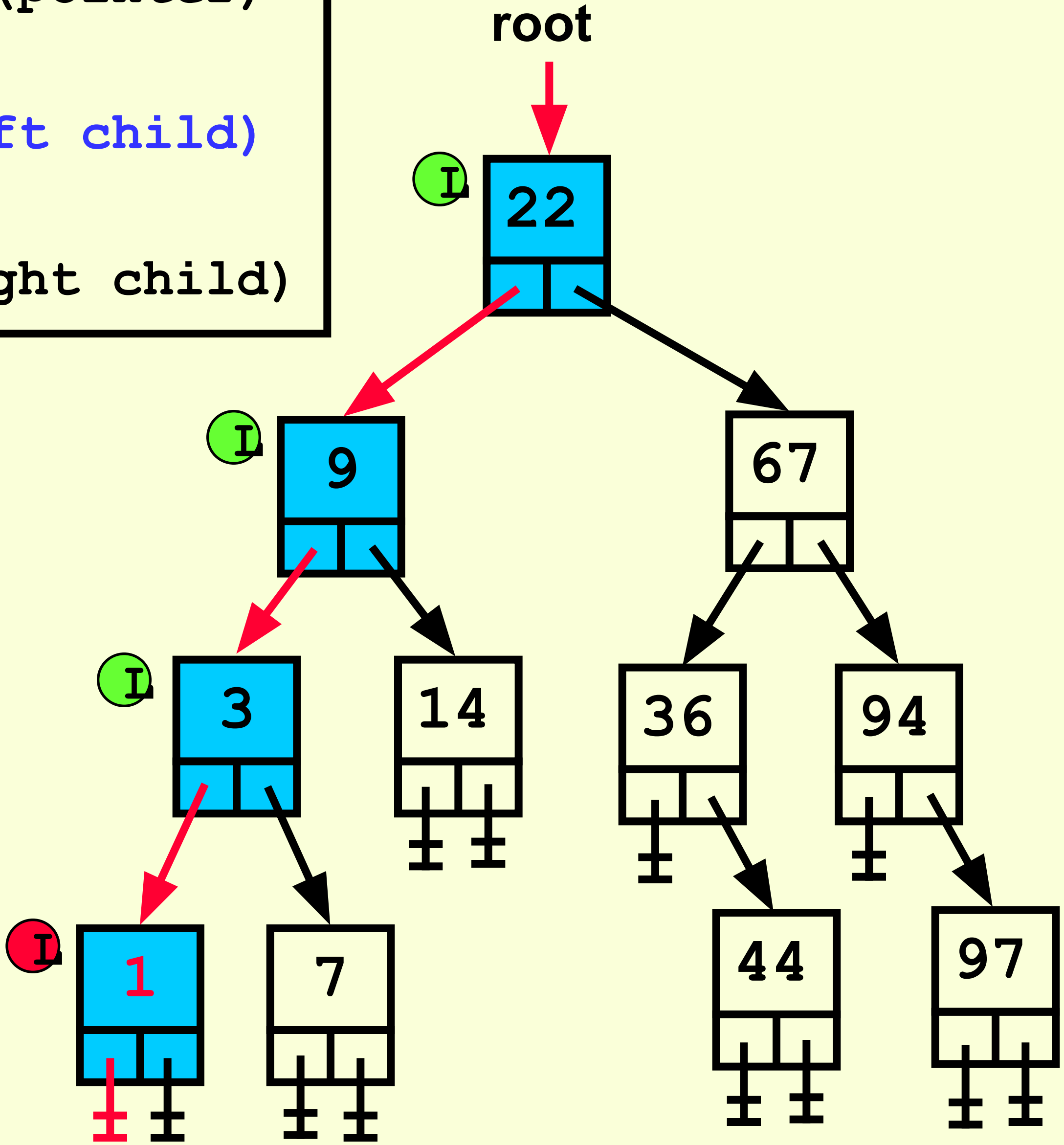
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

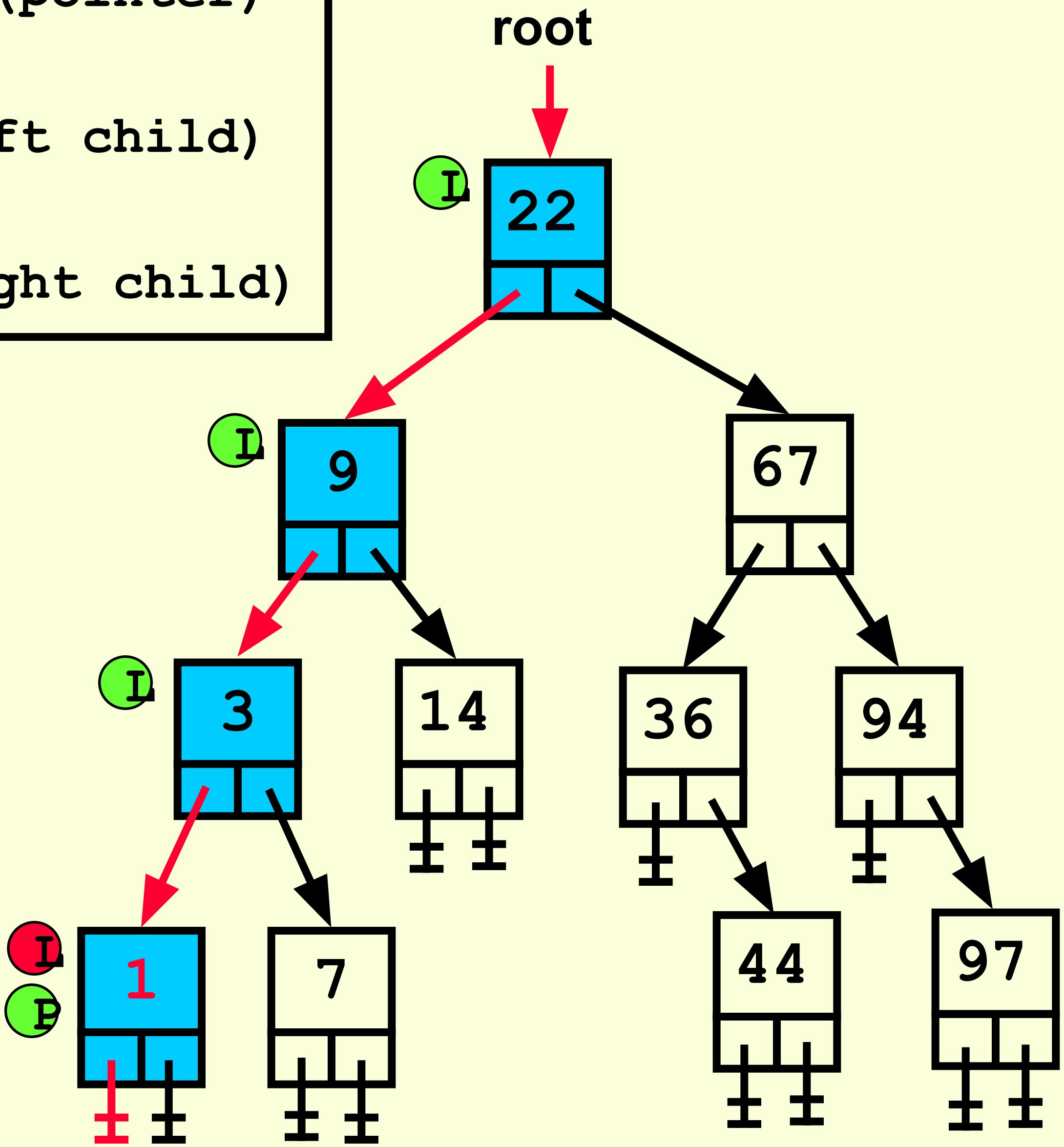


```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



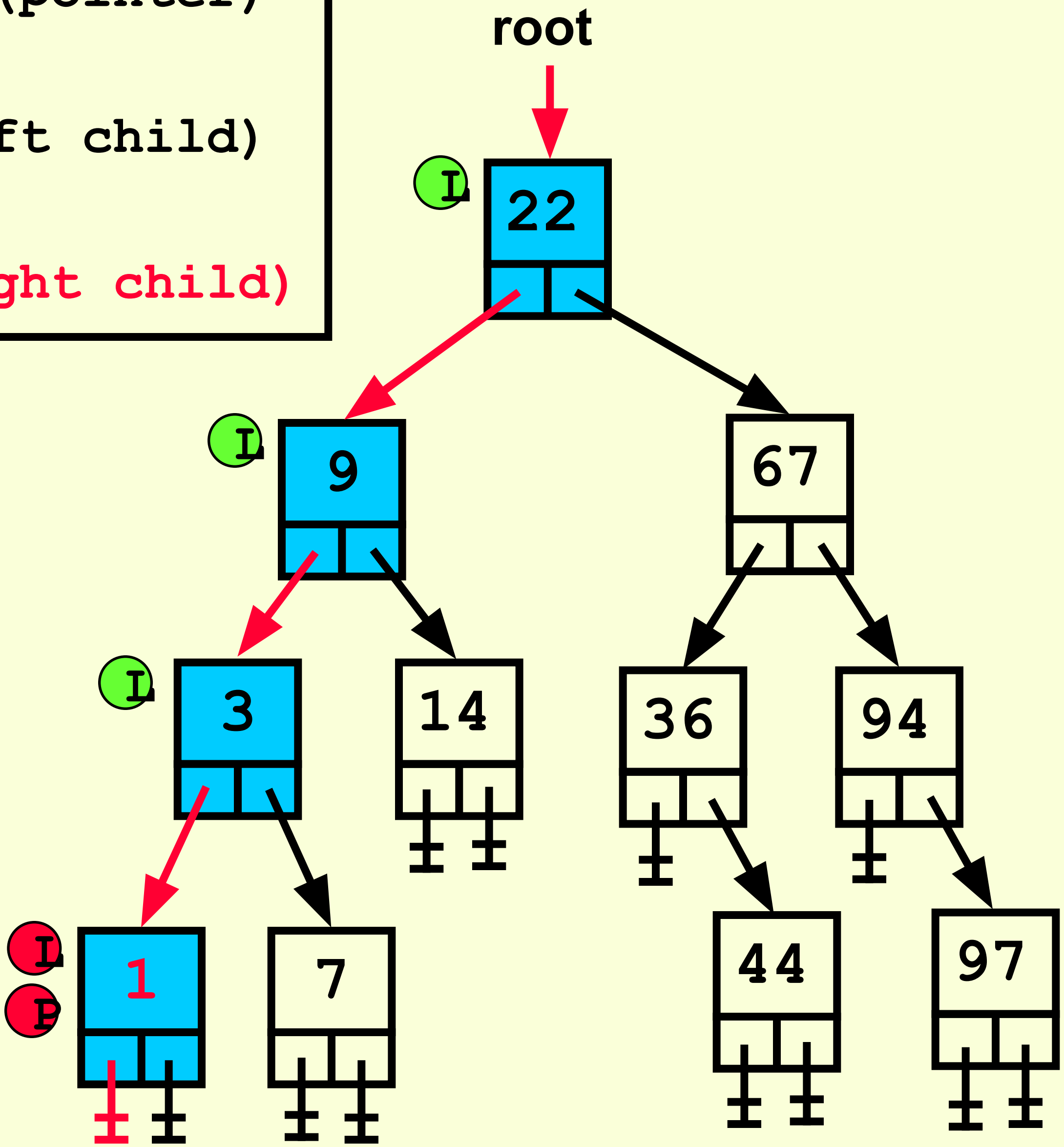
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1
```



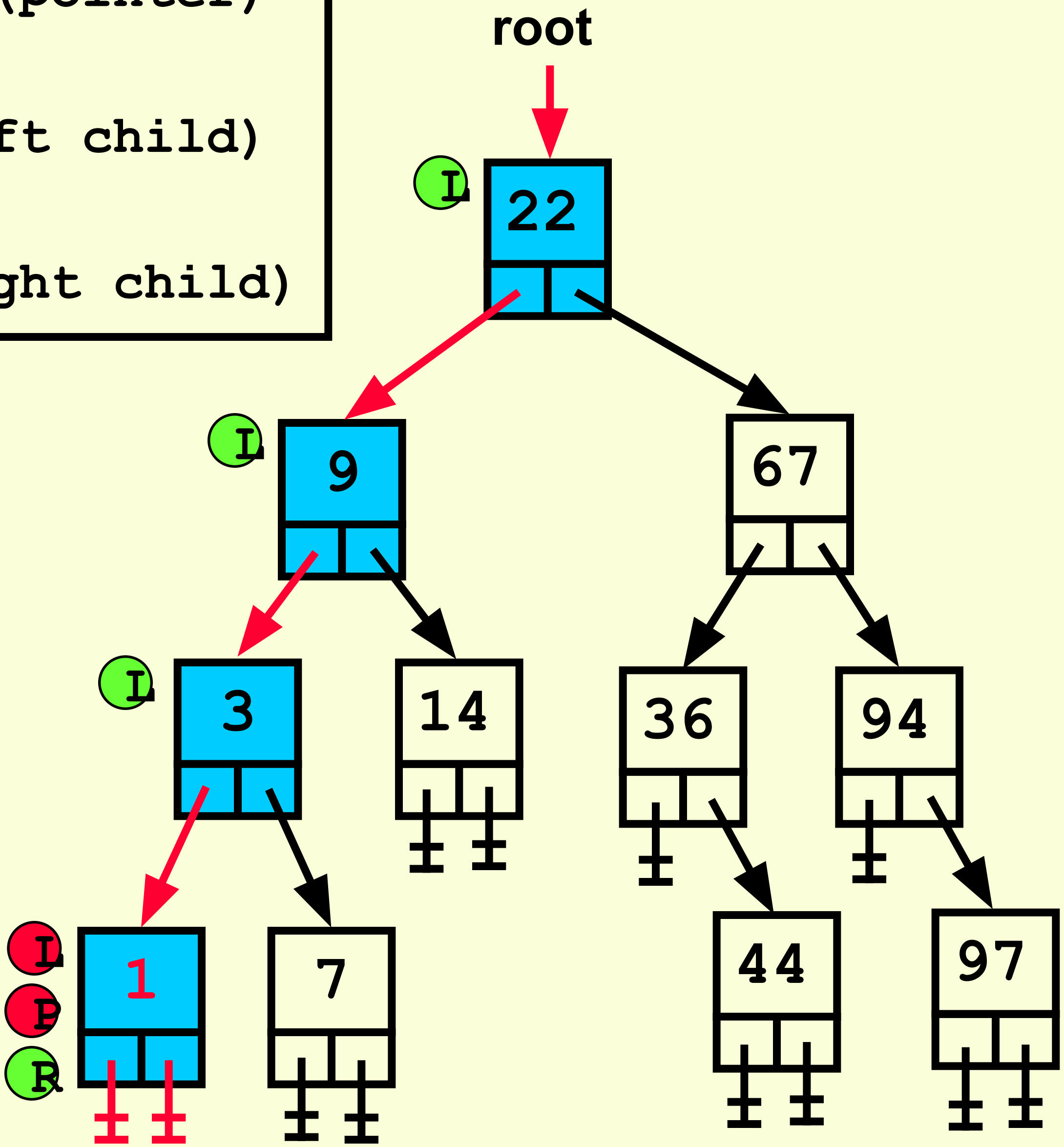
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1
```



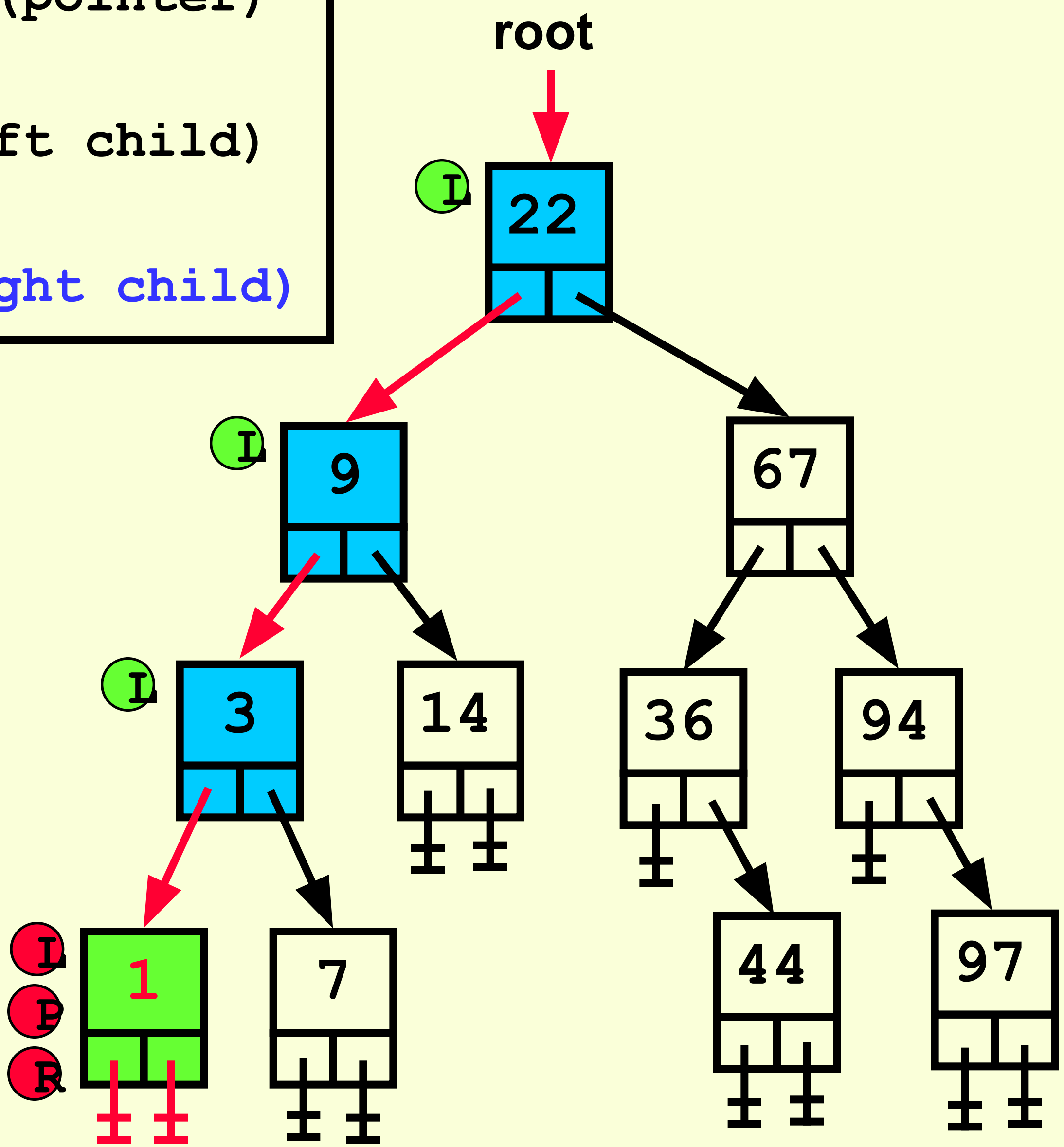
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

Output: 1



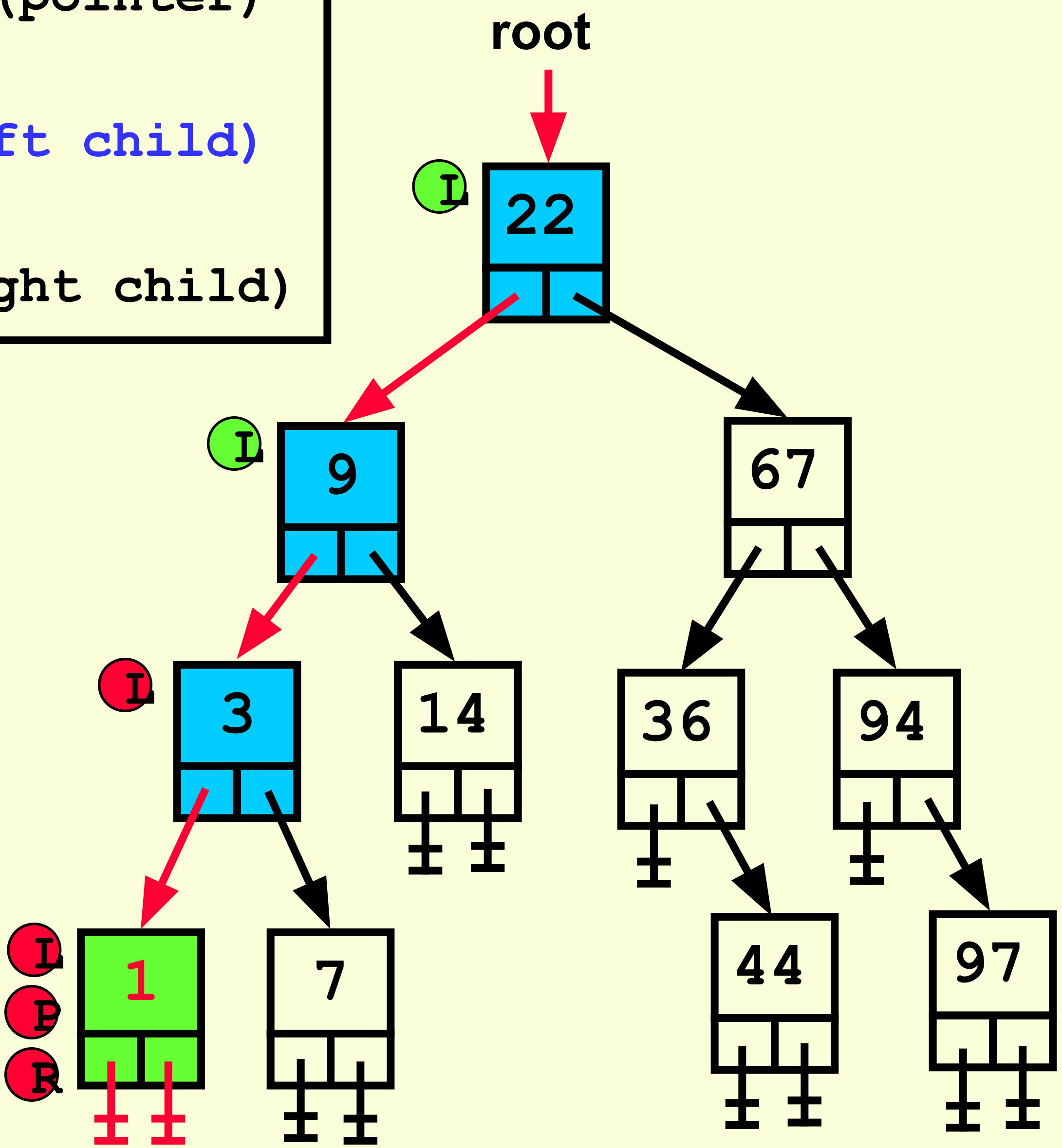
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1
```



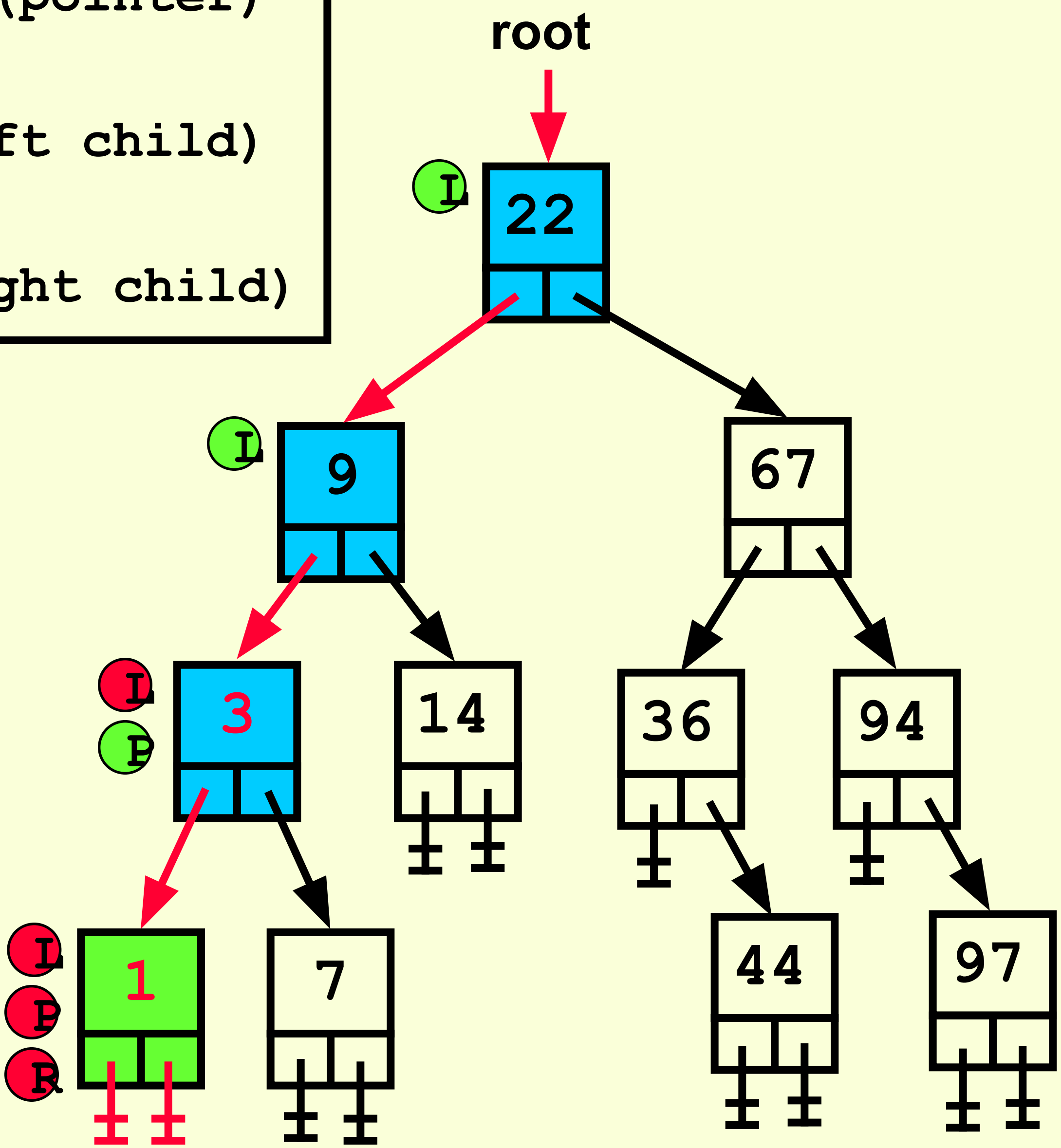
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

Output: 1



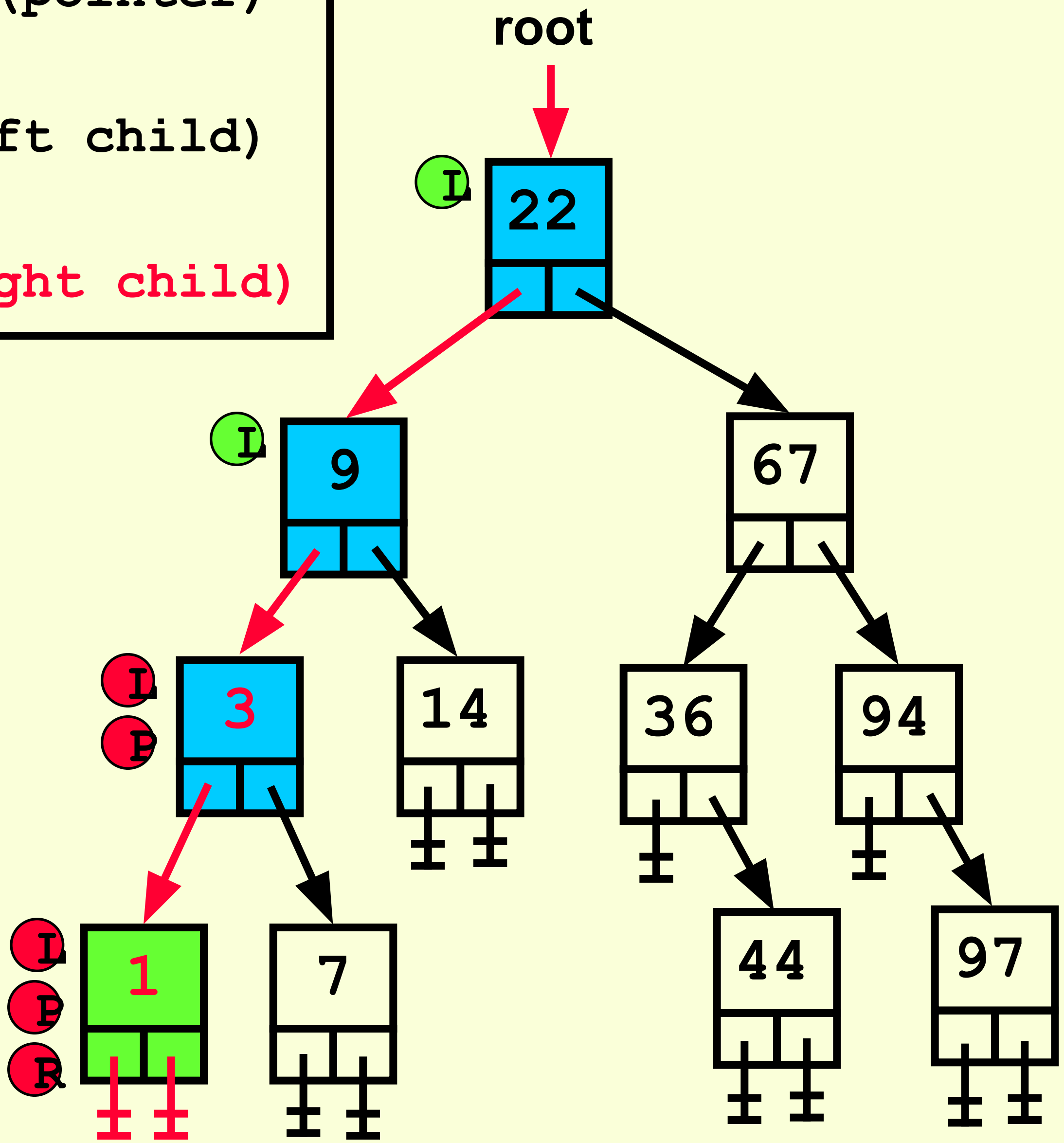
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3
```



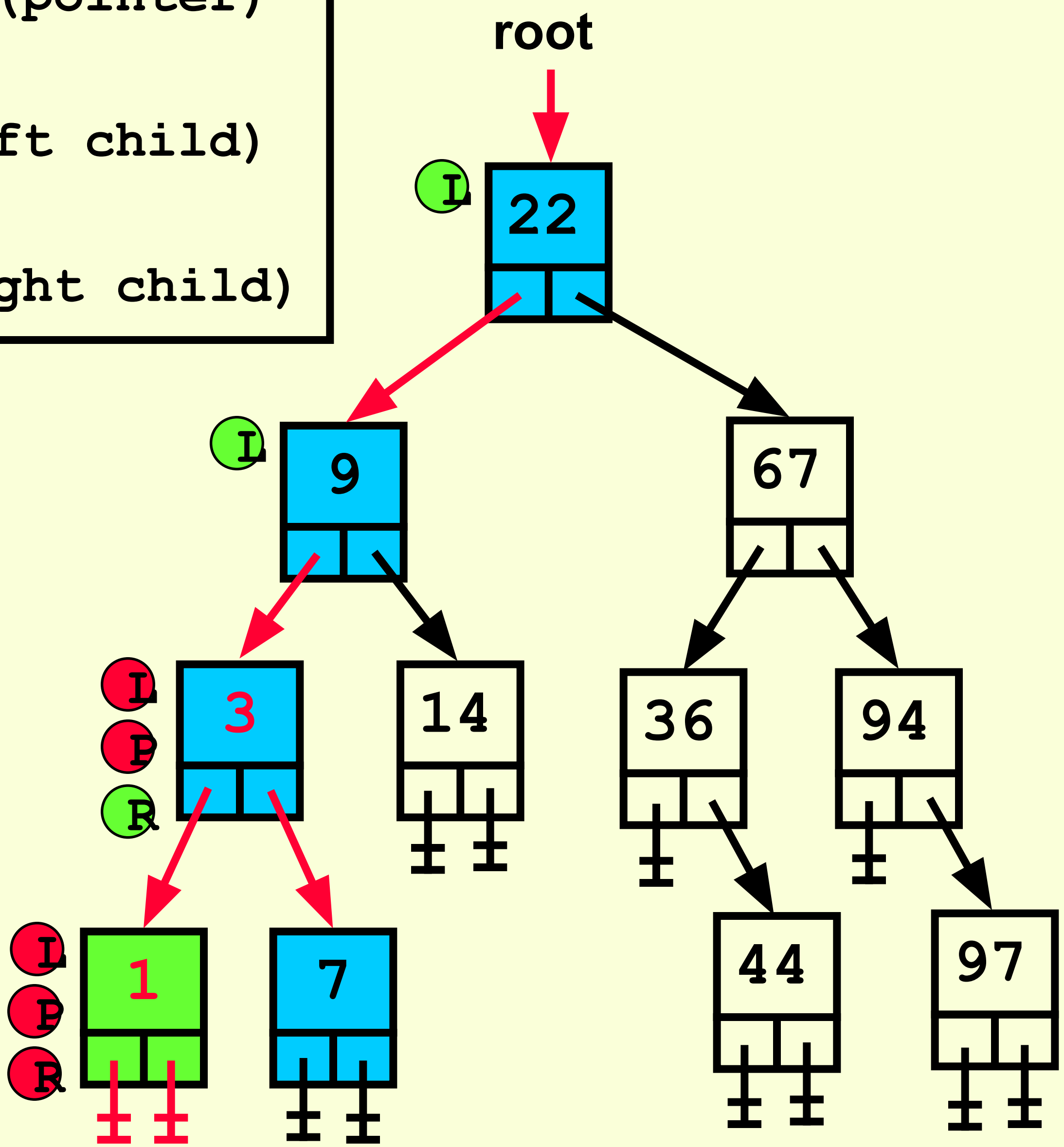
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3
```



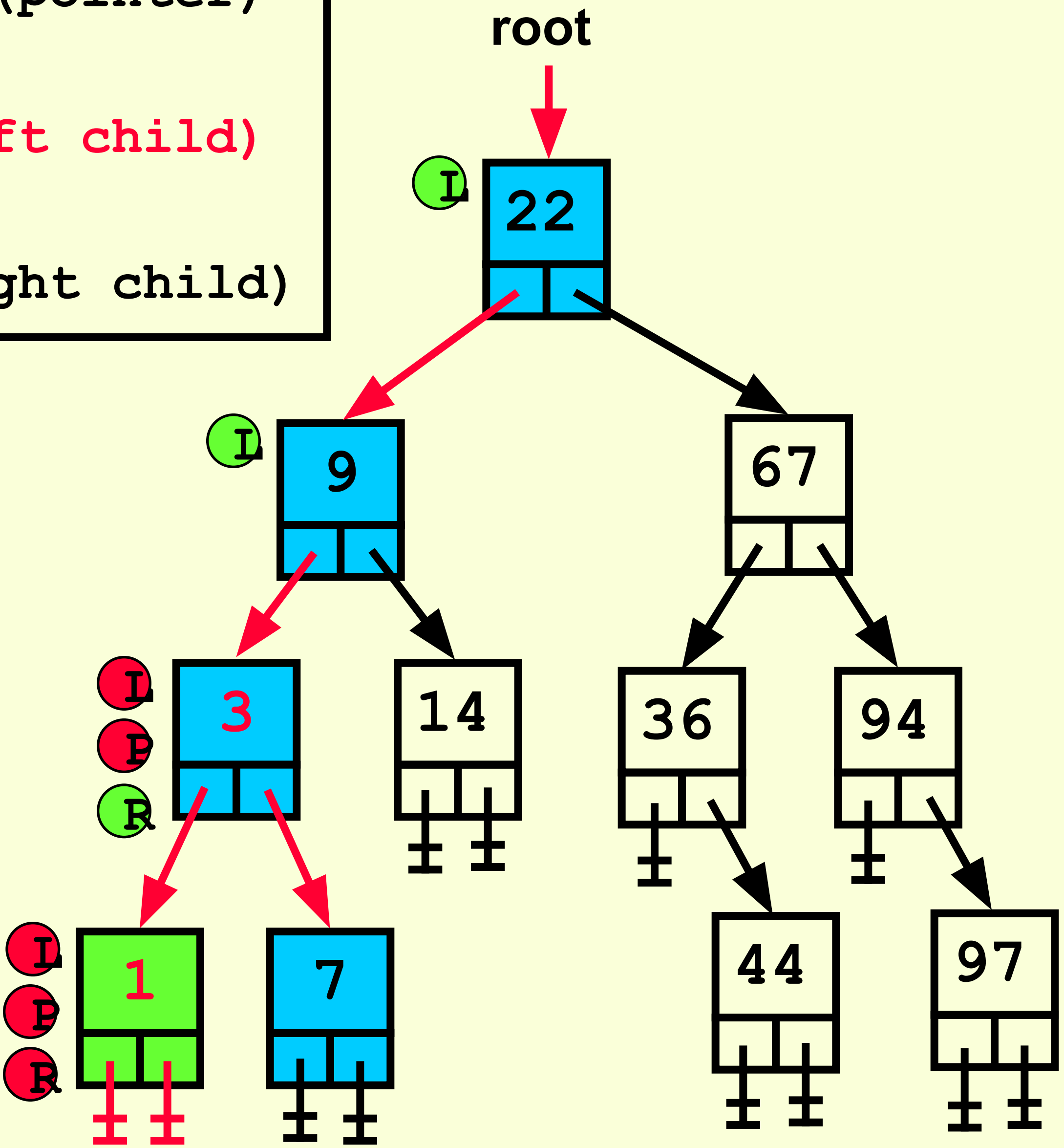
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3
```



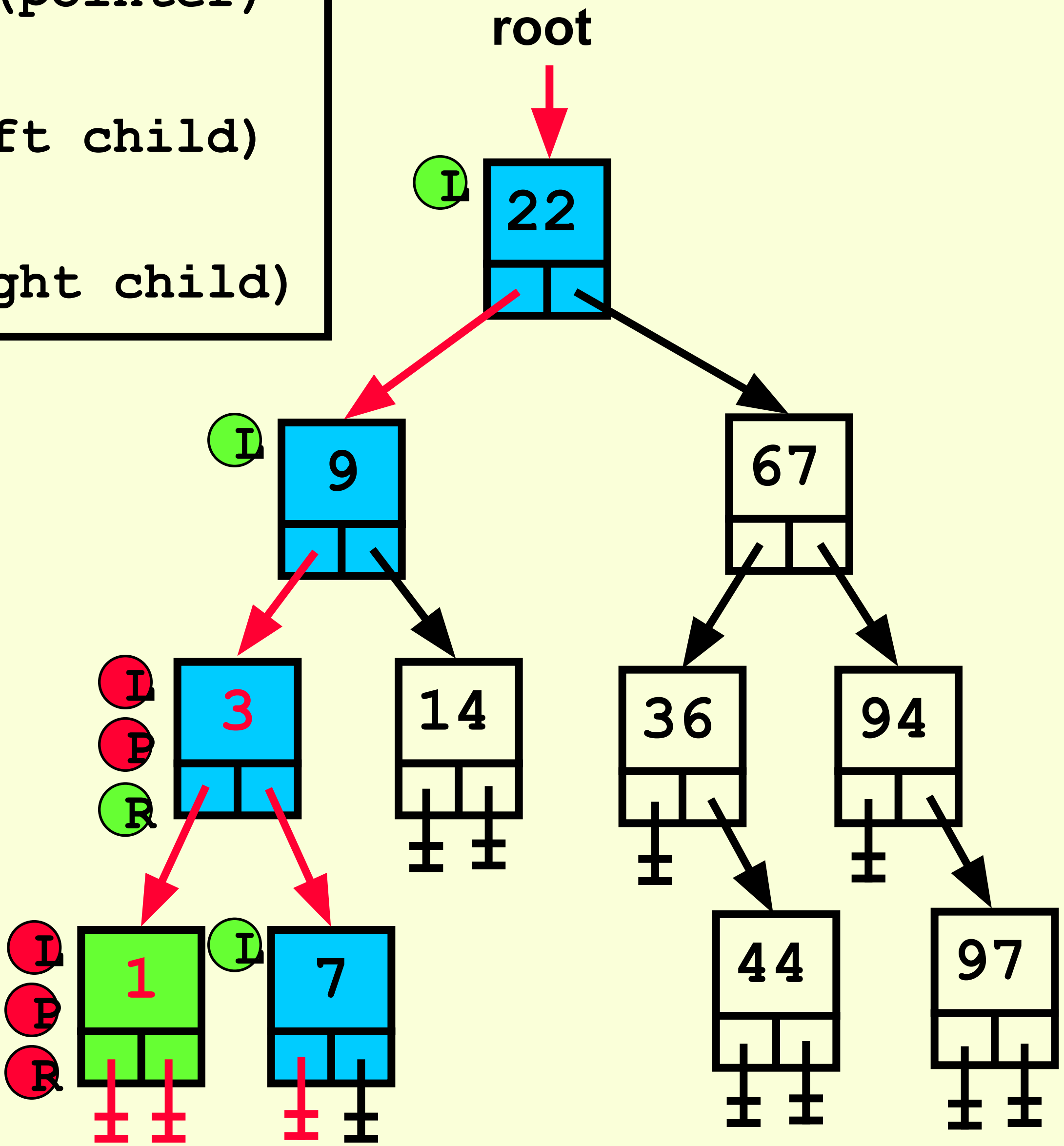
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3
```



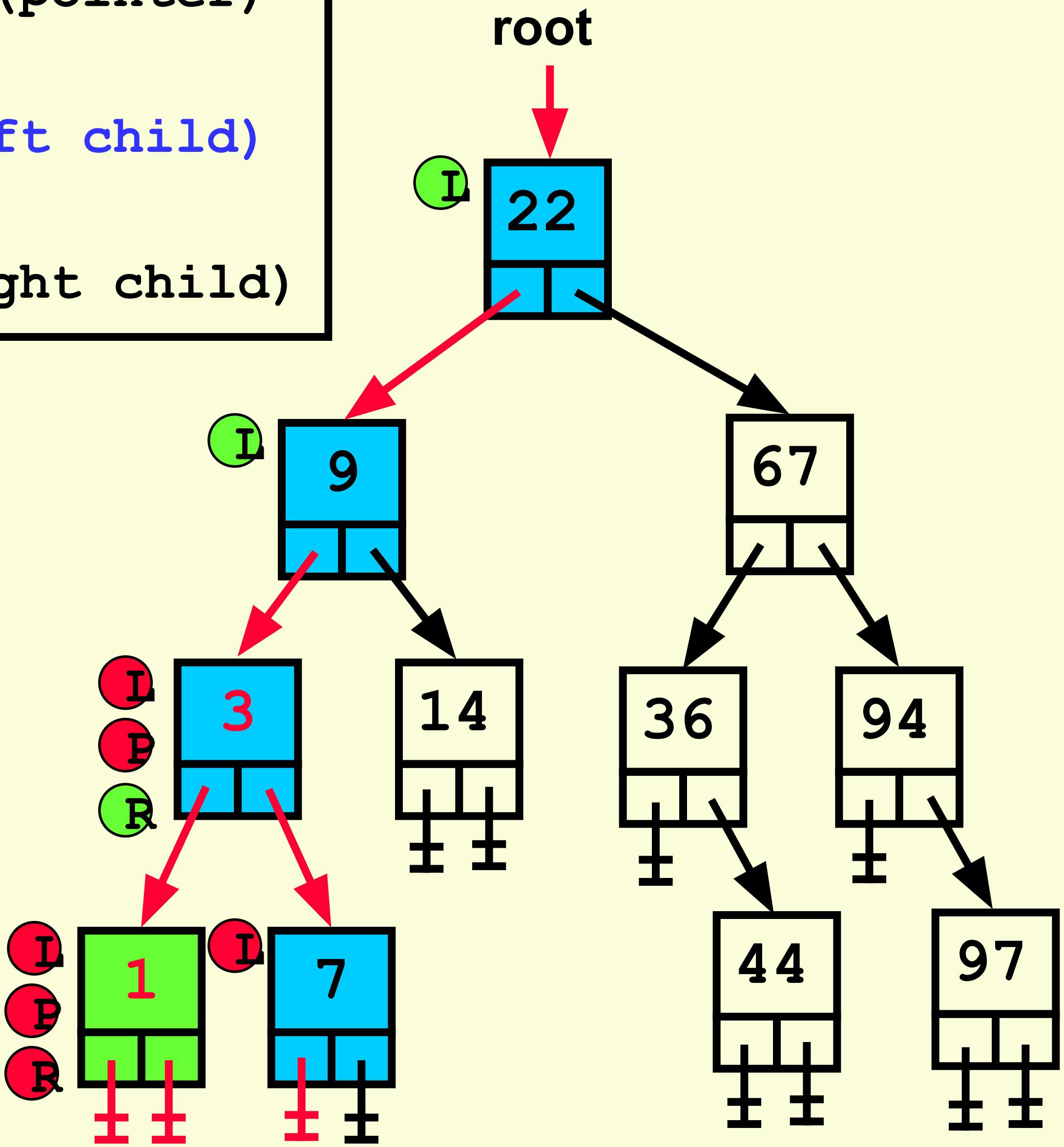
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

Output: 1 3



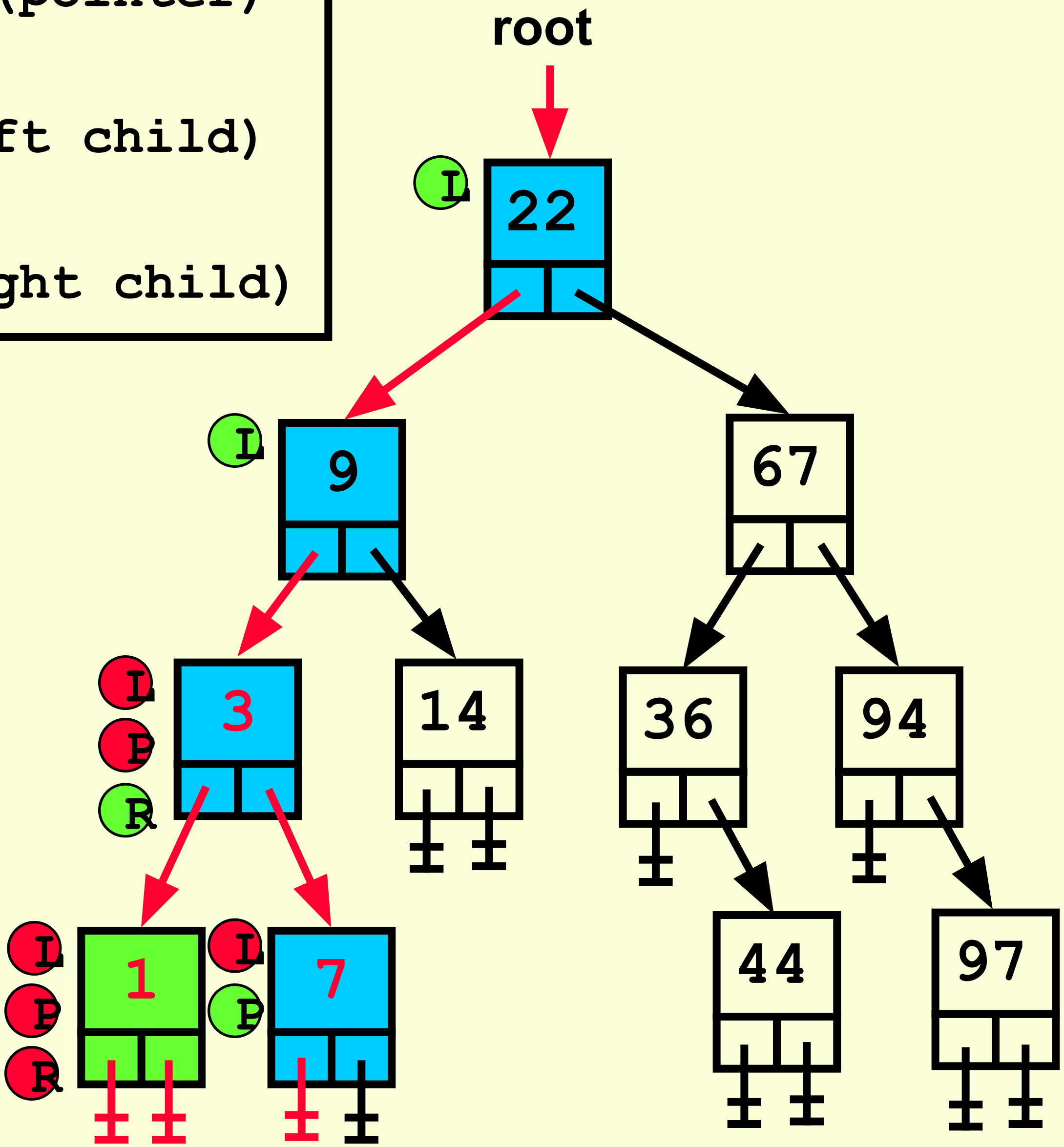
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3
```



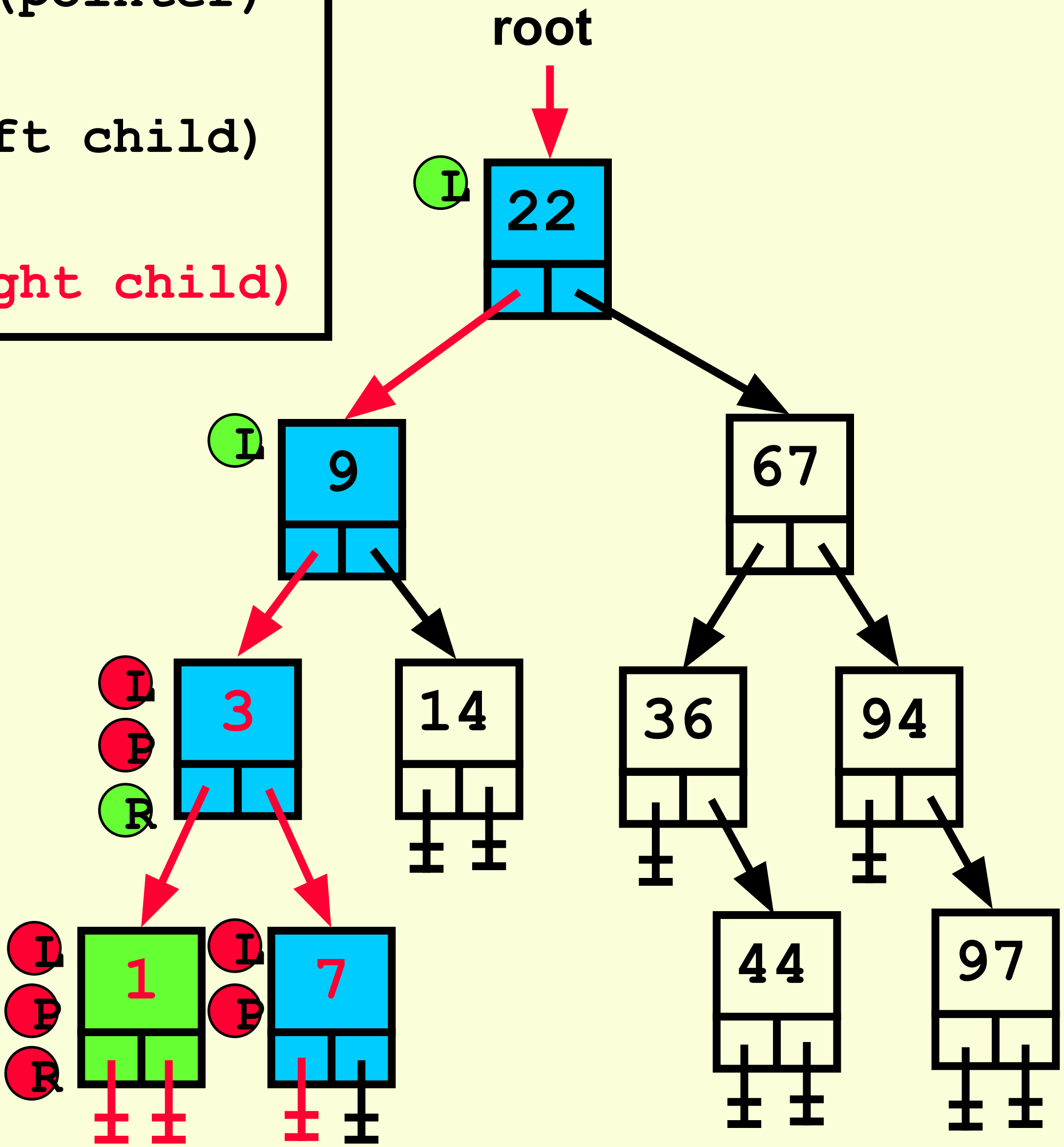
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
Ⓛ InOrderPrint(left child)
Ⓟ print(data)
Ⓡ InOrderPrint(right child)
```

Output: 1 3 7



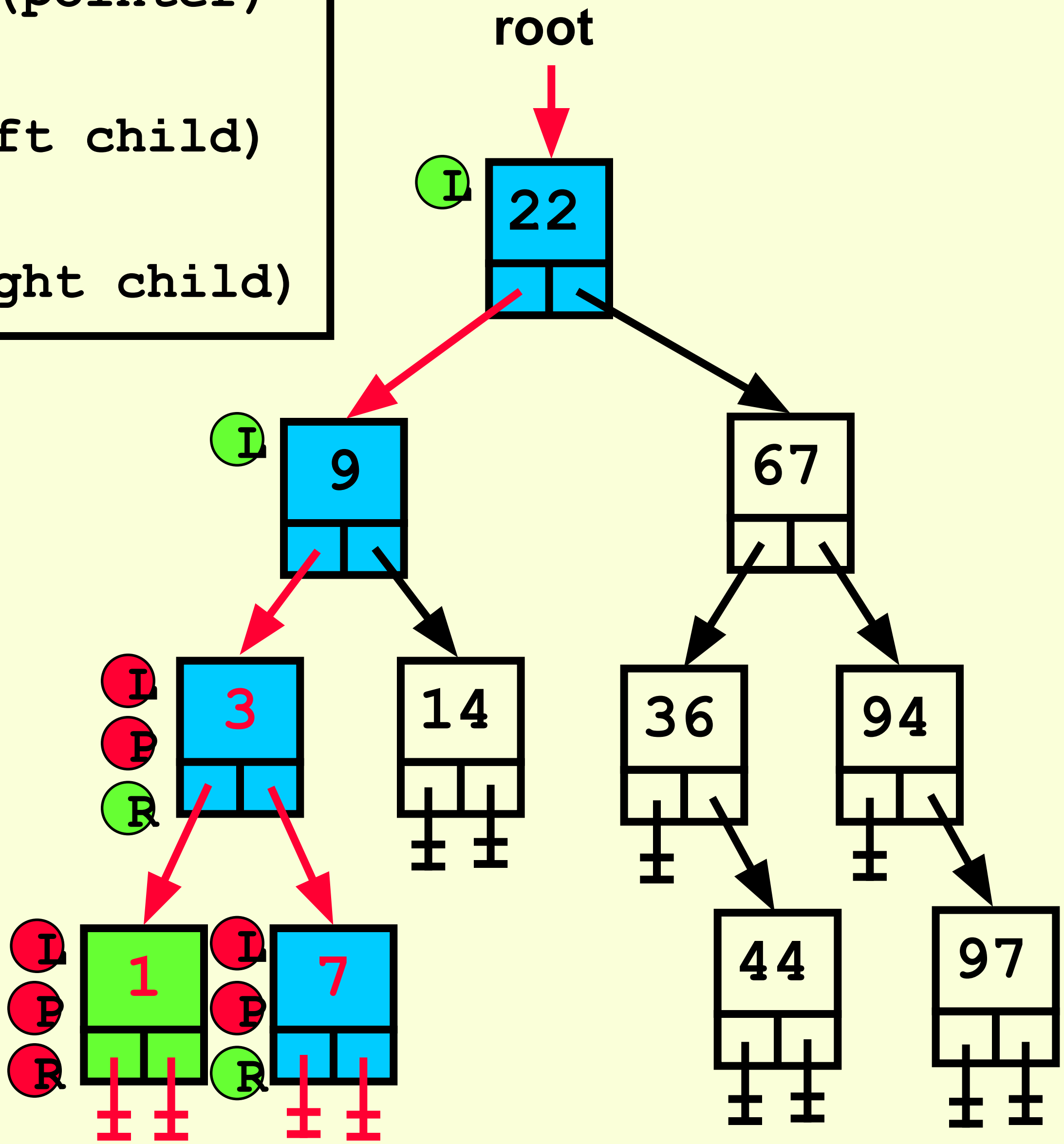
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

Output: 1 3 7



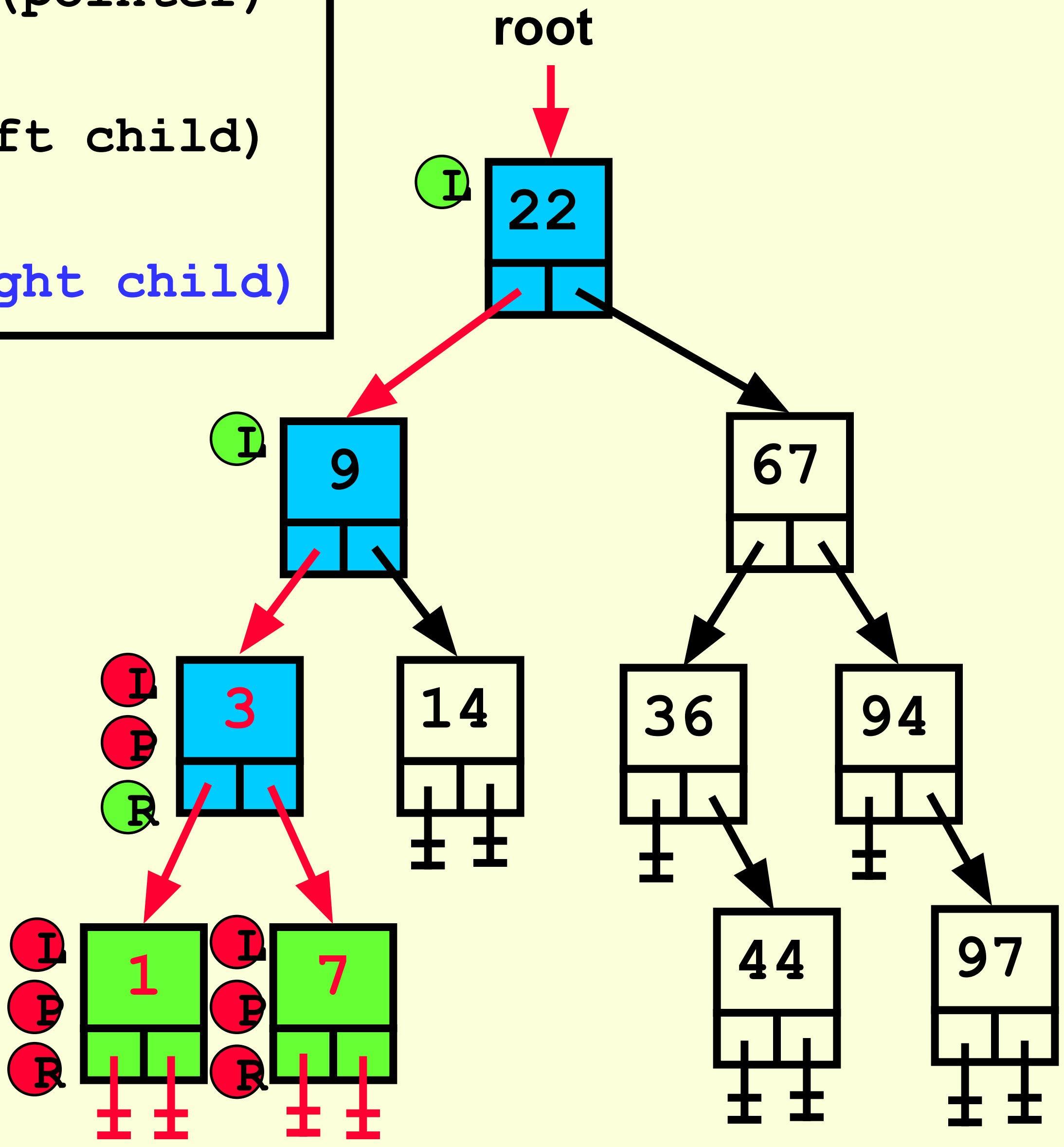
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7
```



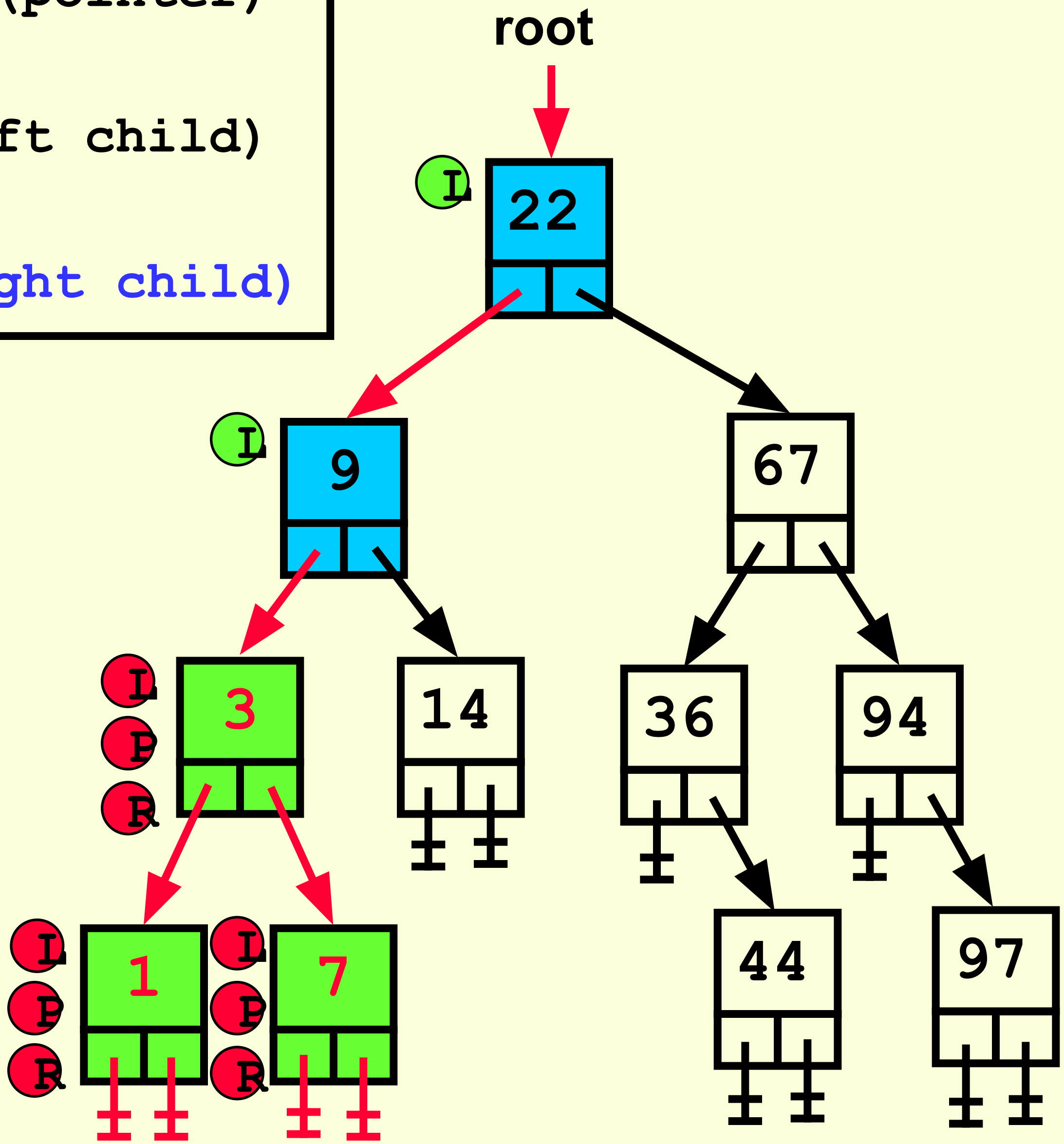
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7
```



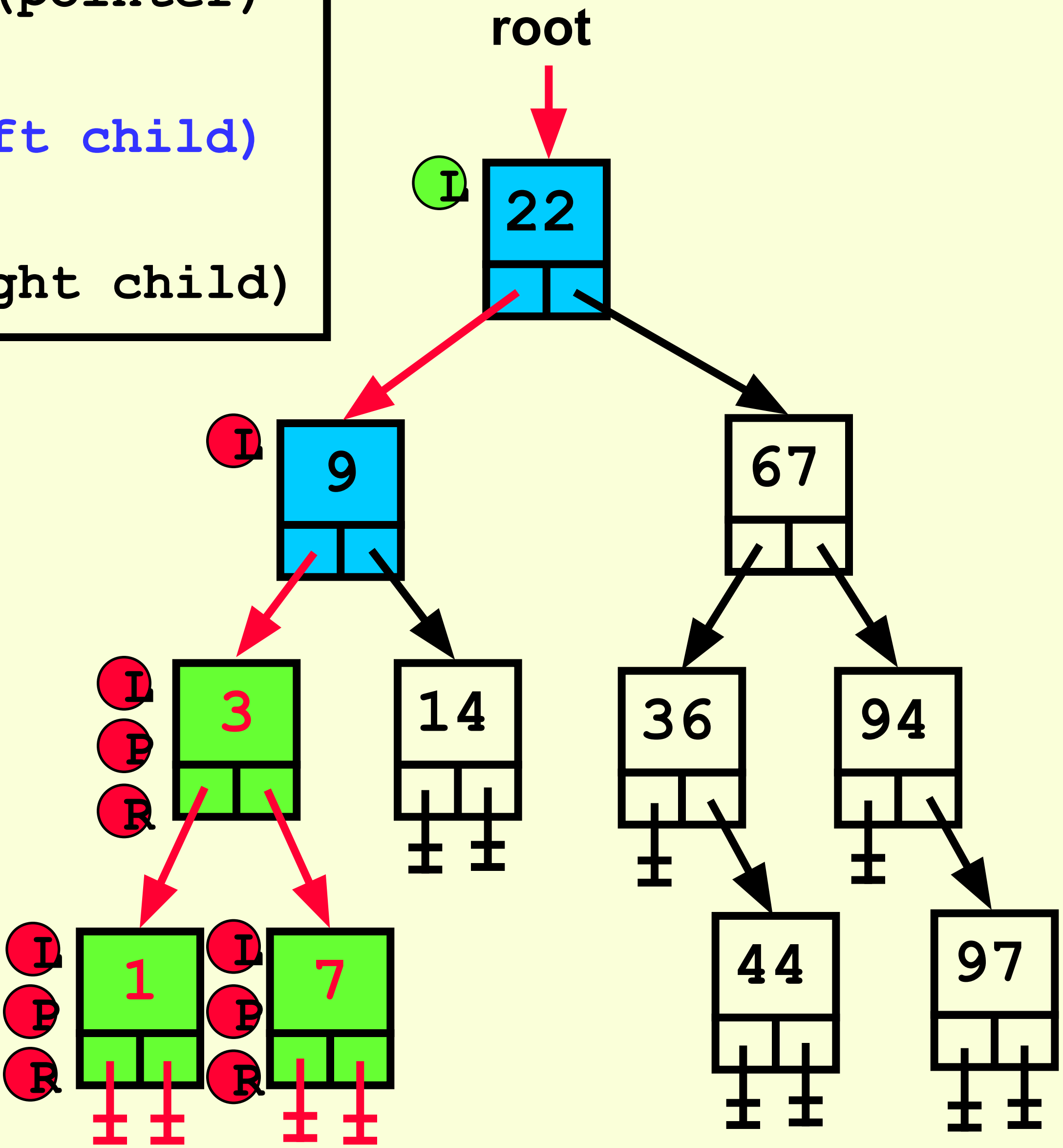
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7
```



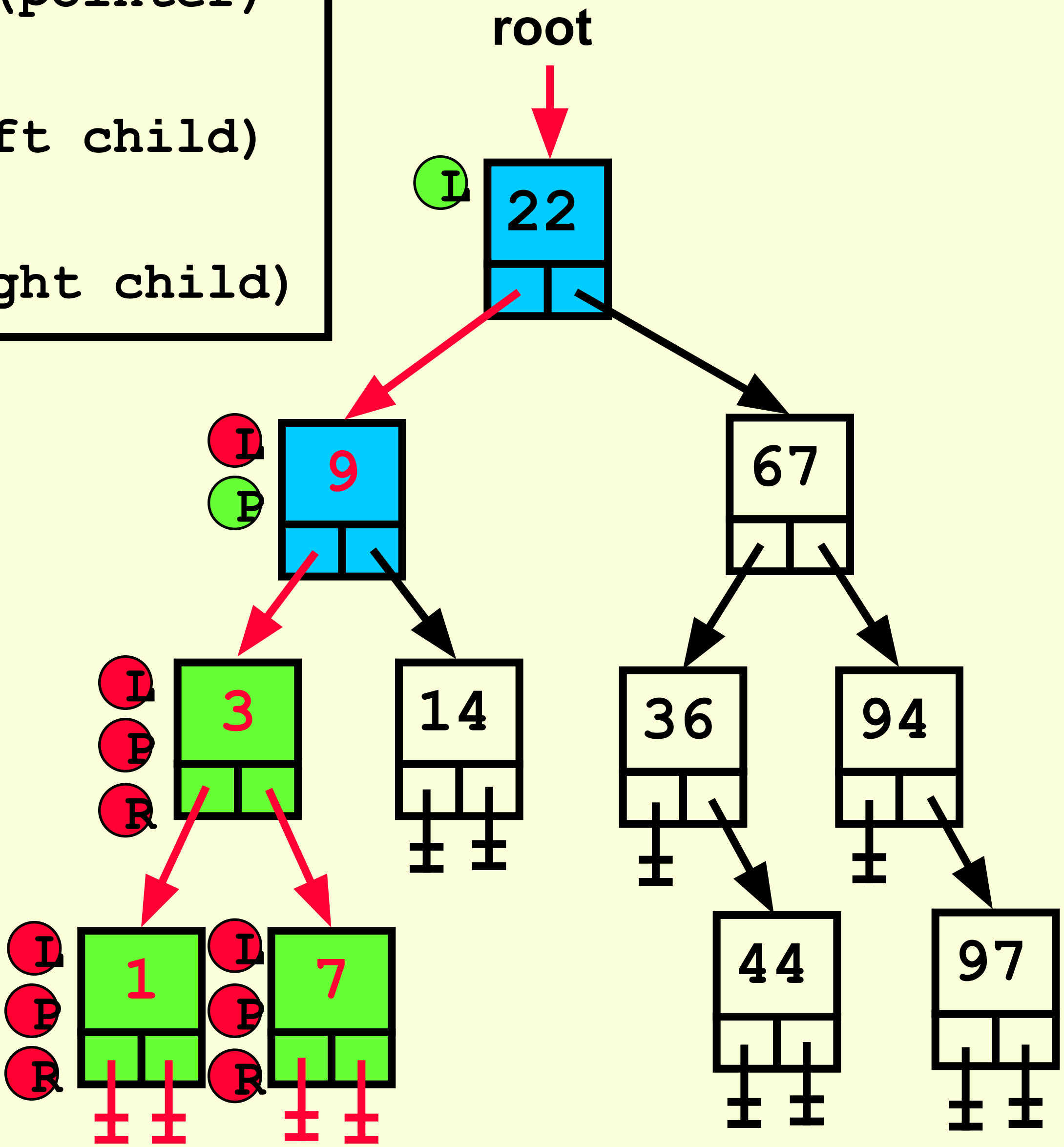
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7
```



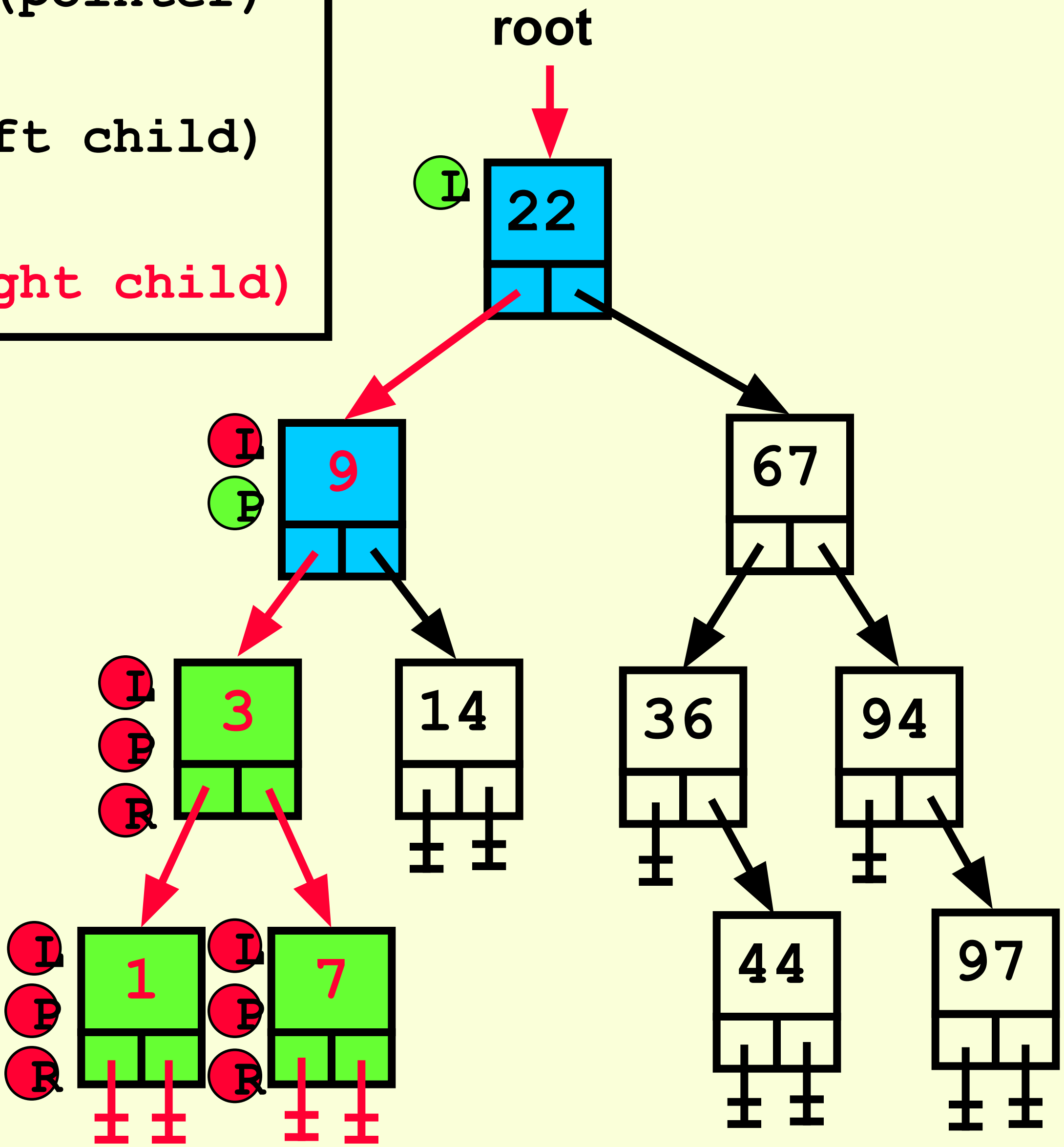
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

Output: 1 3 7 9



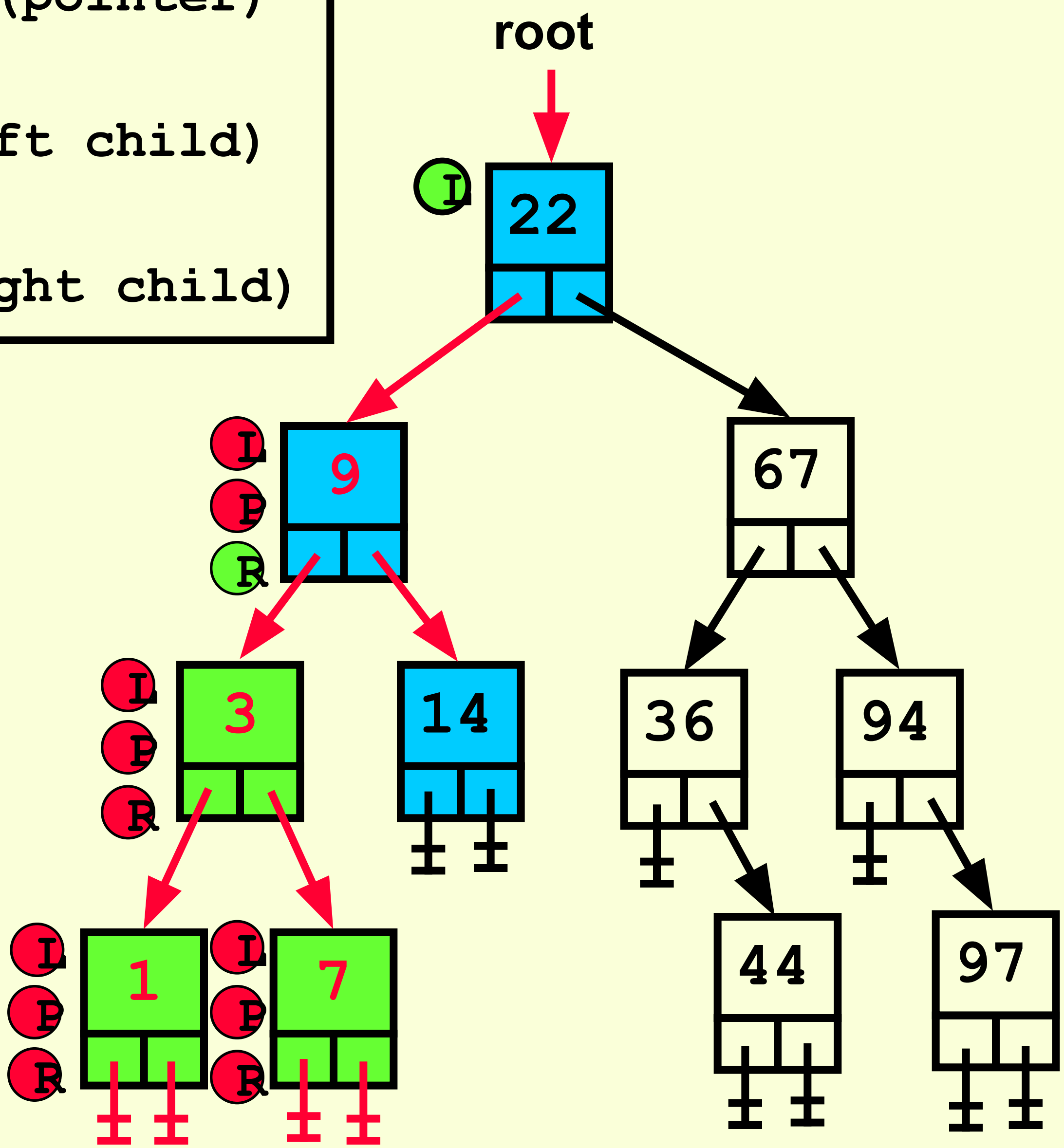
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

Output: 1 3 7 9



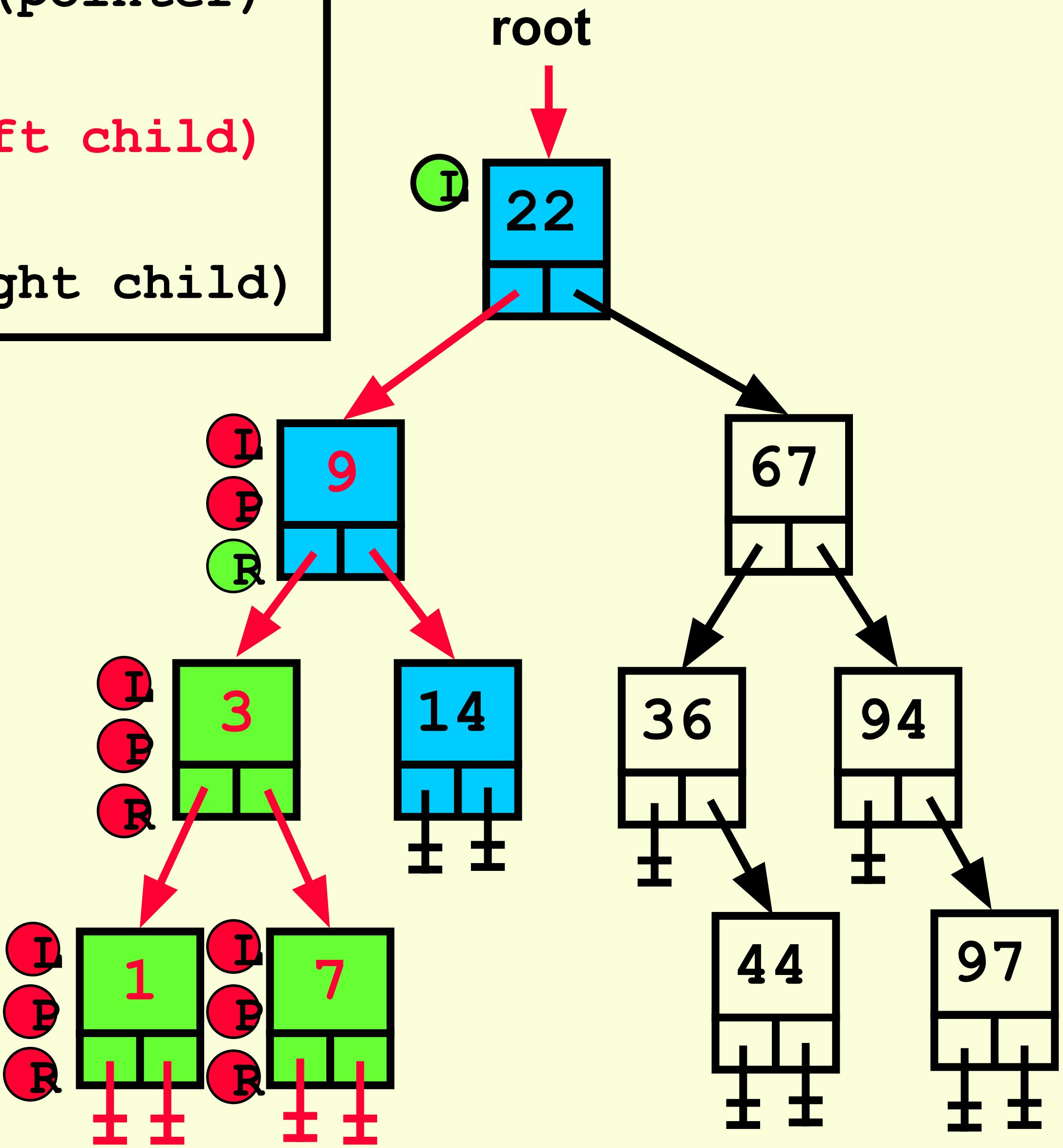
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9
```



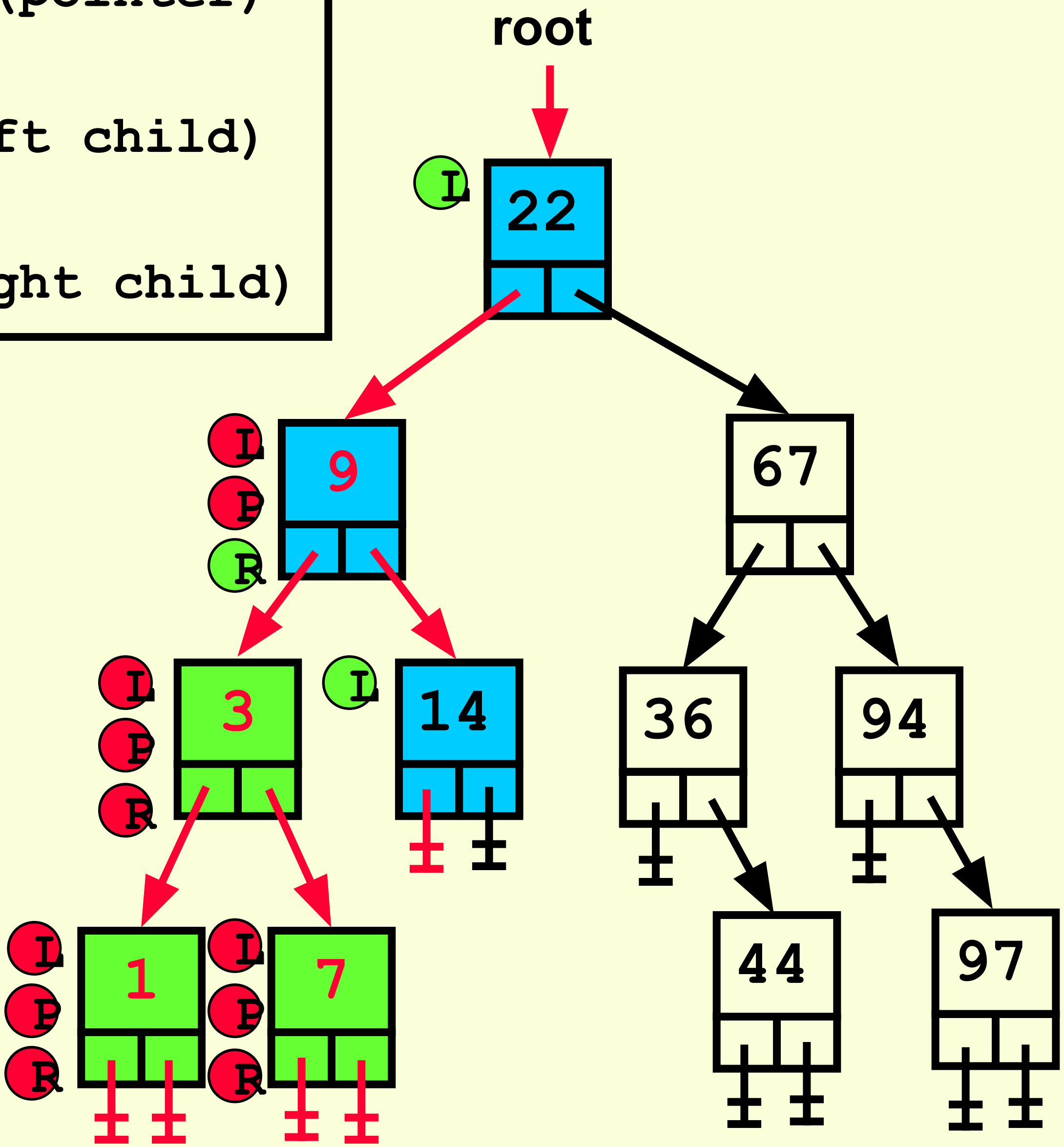
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9
```



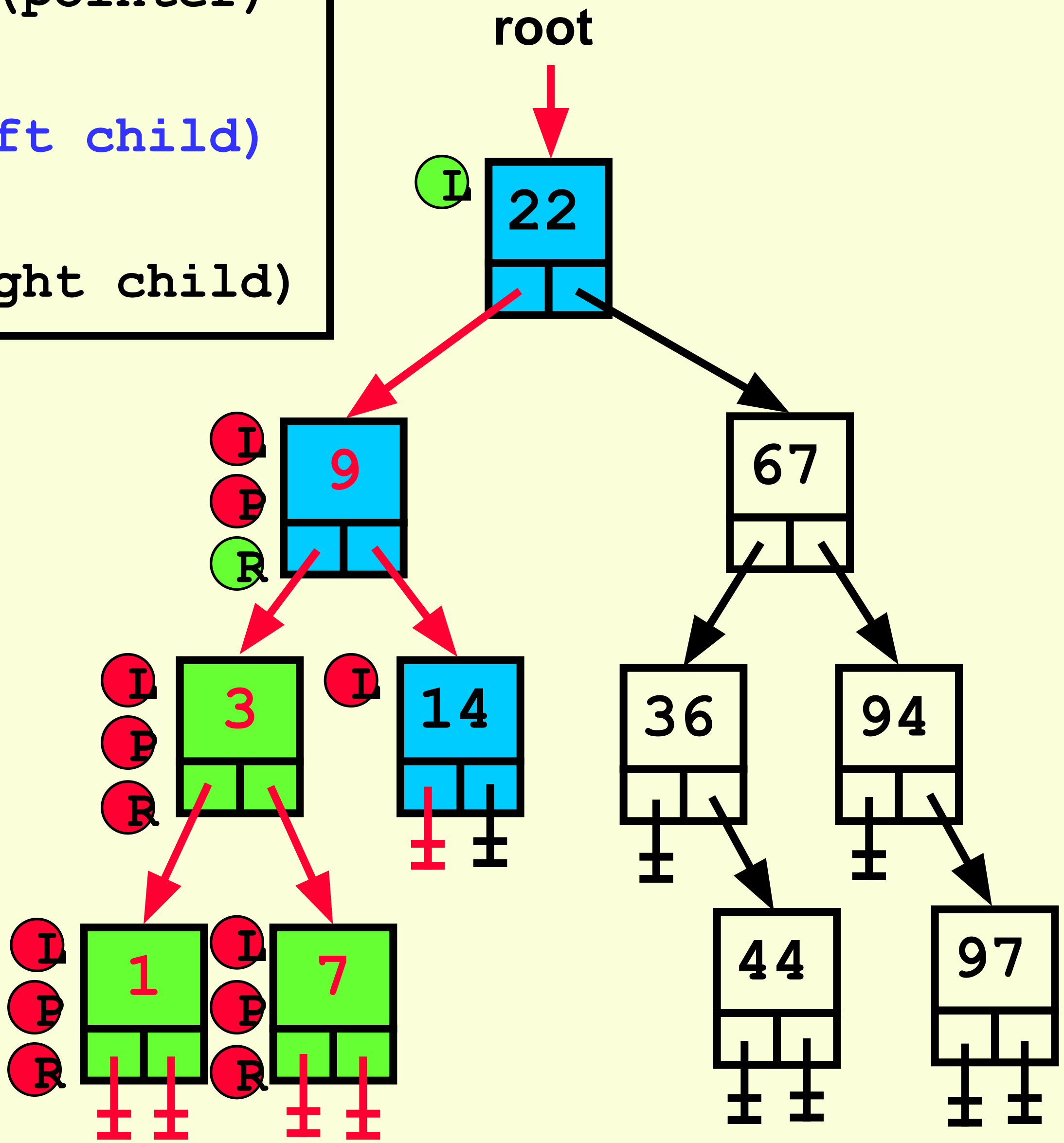
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9
```



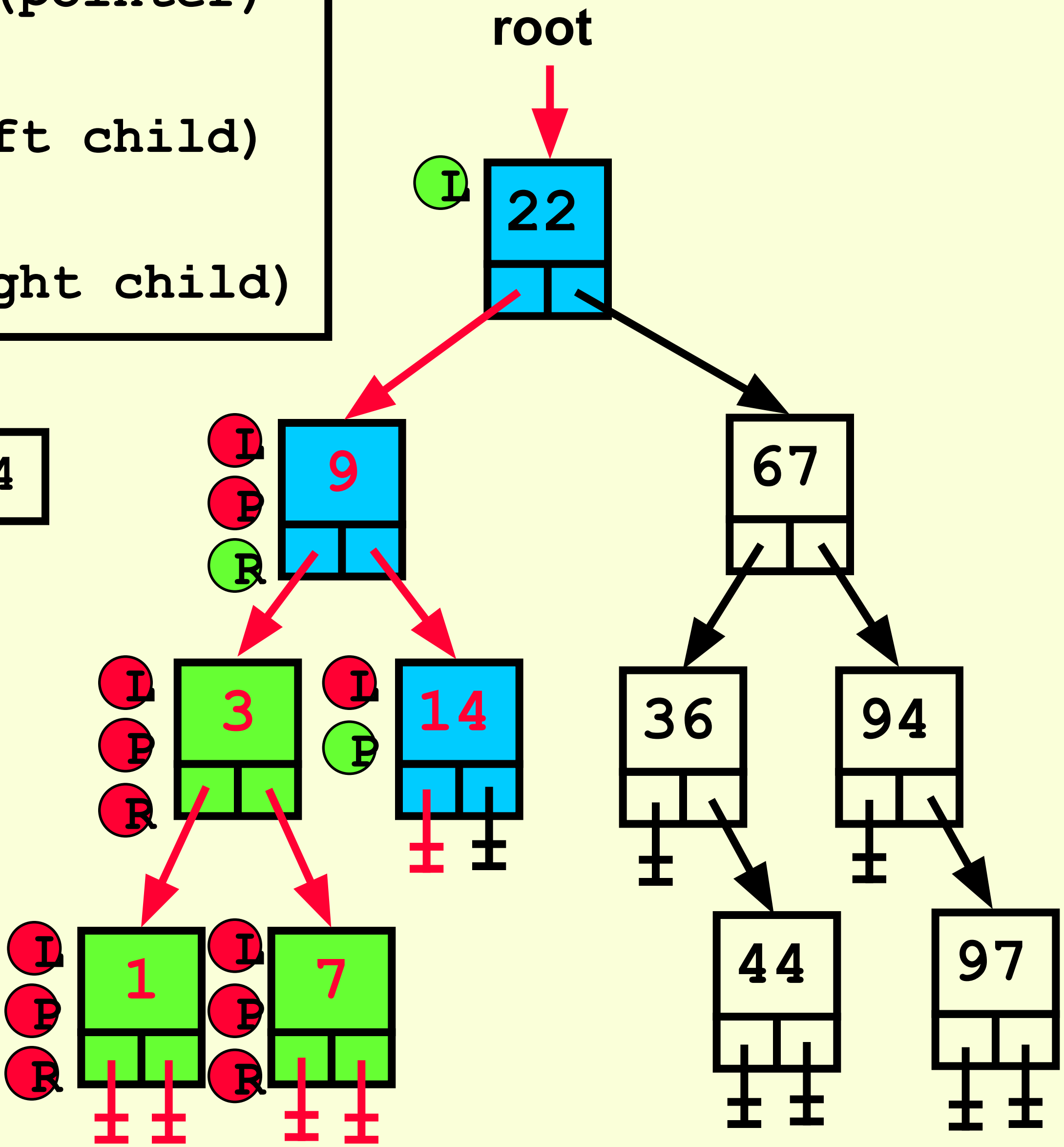
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9
```



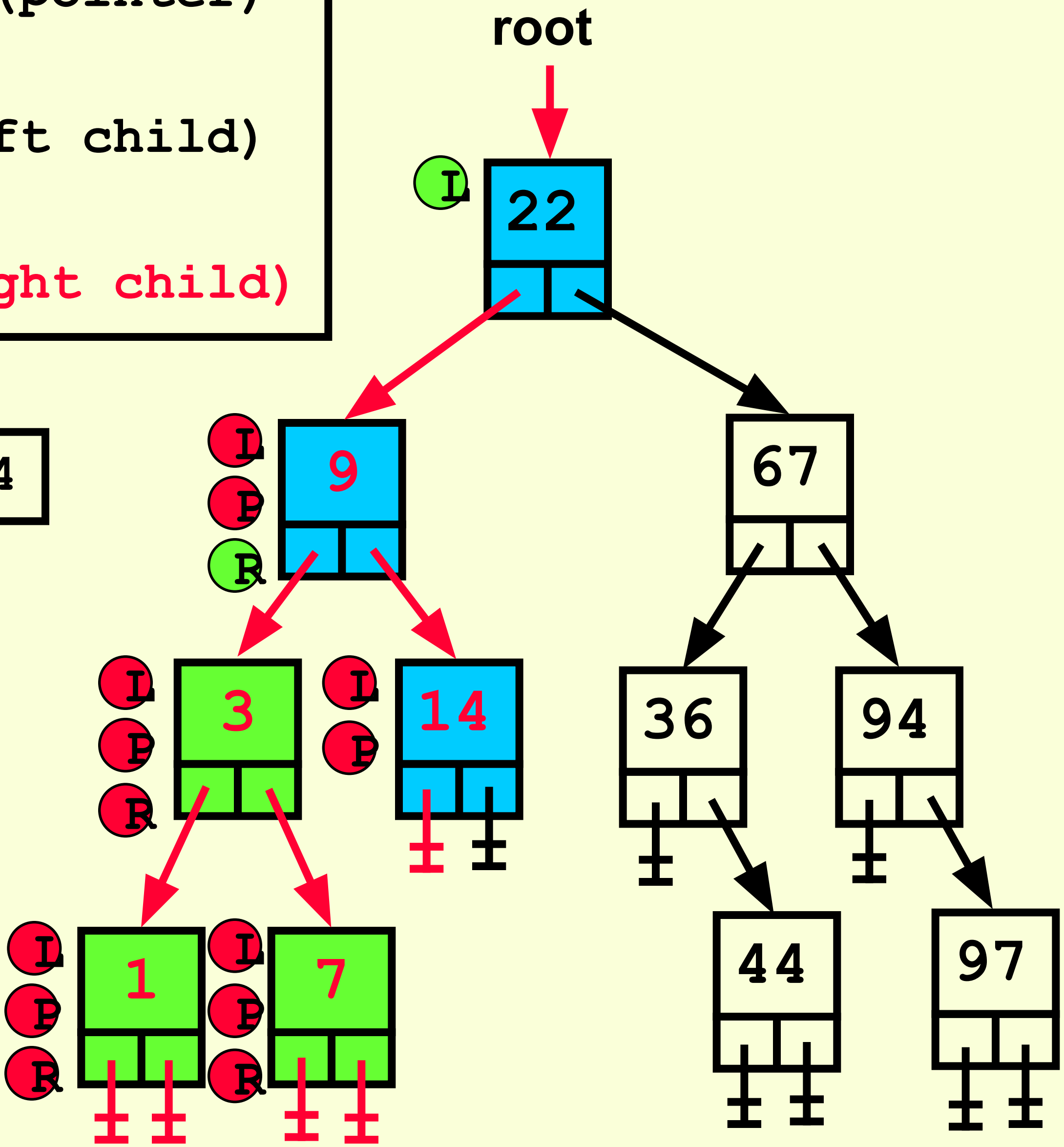
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14
```



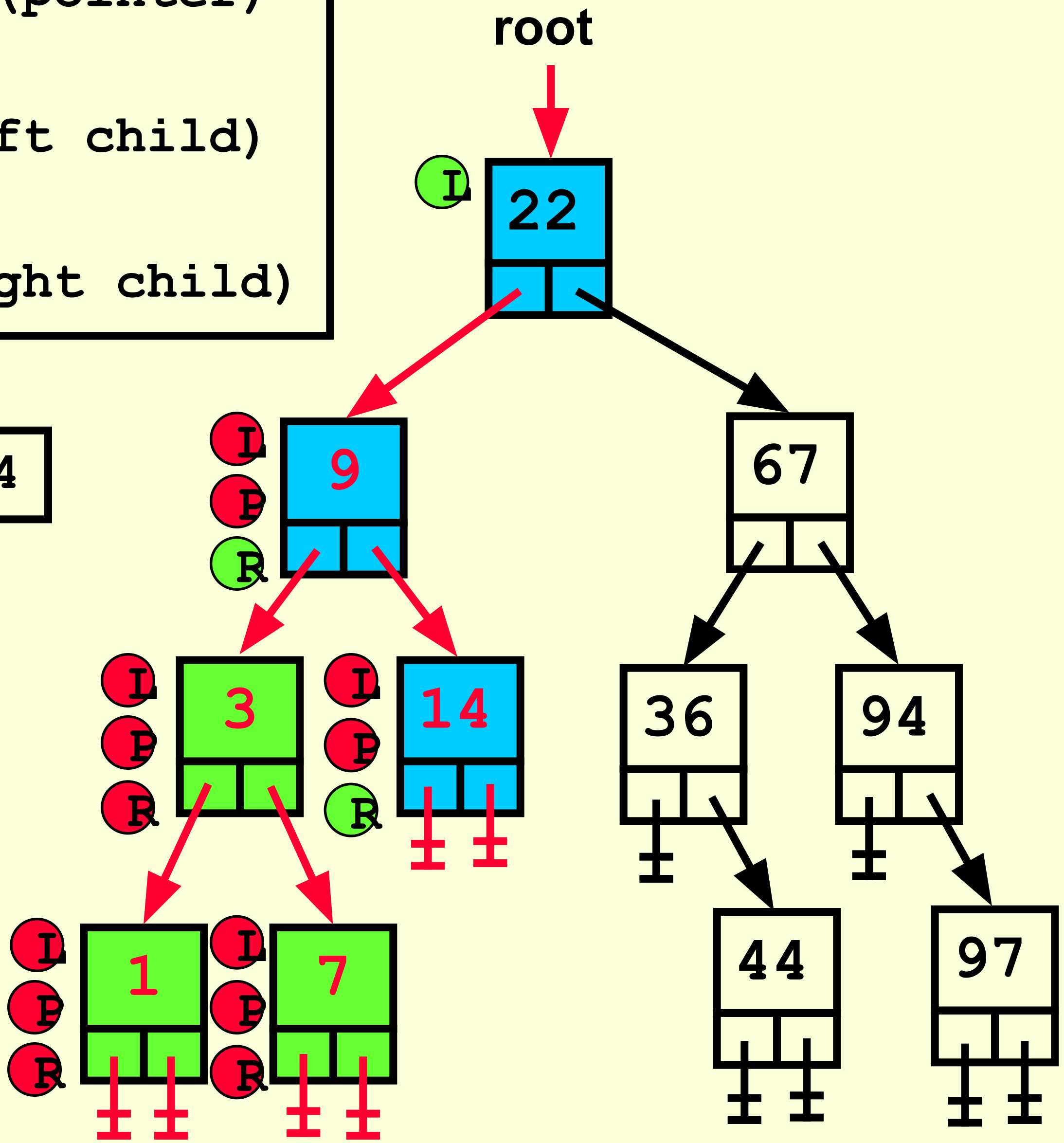
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14
```



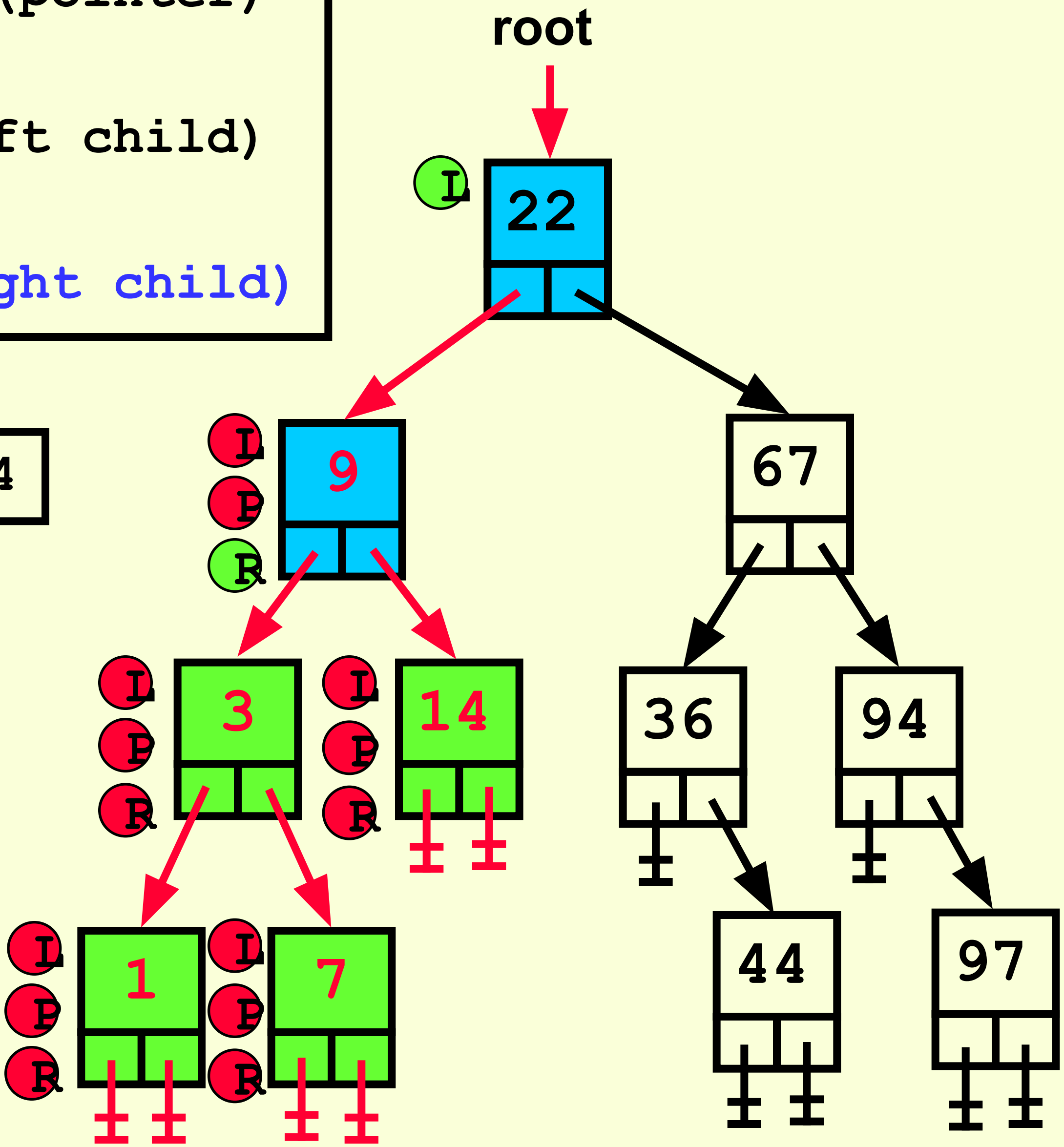
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14
```



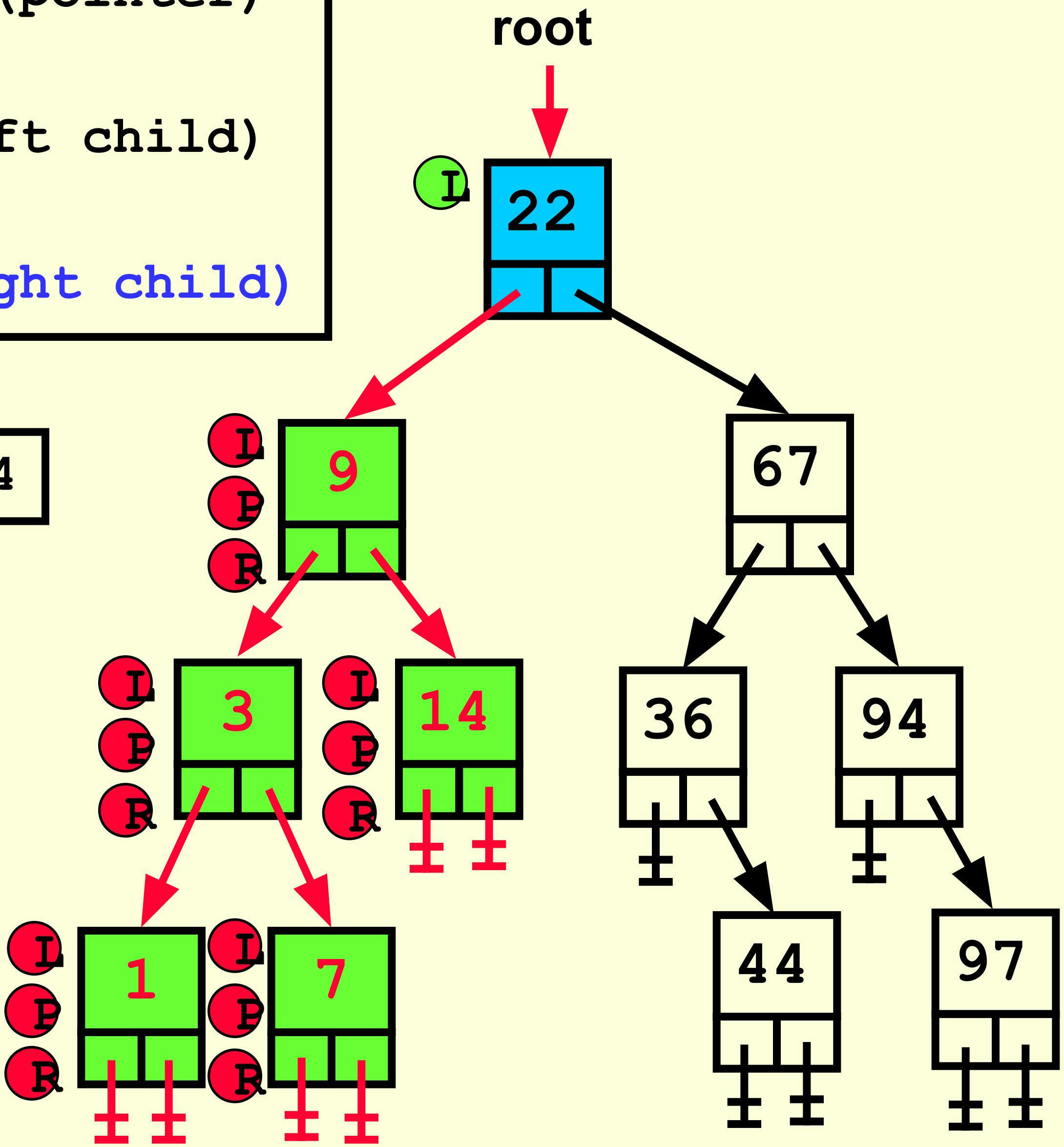
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14
```



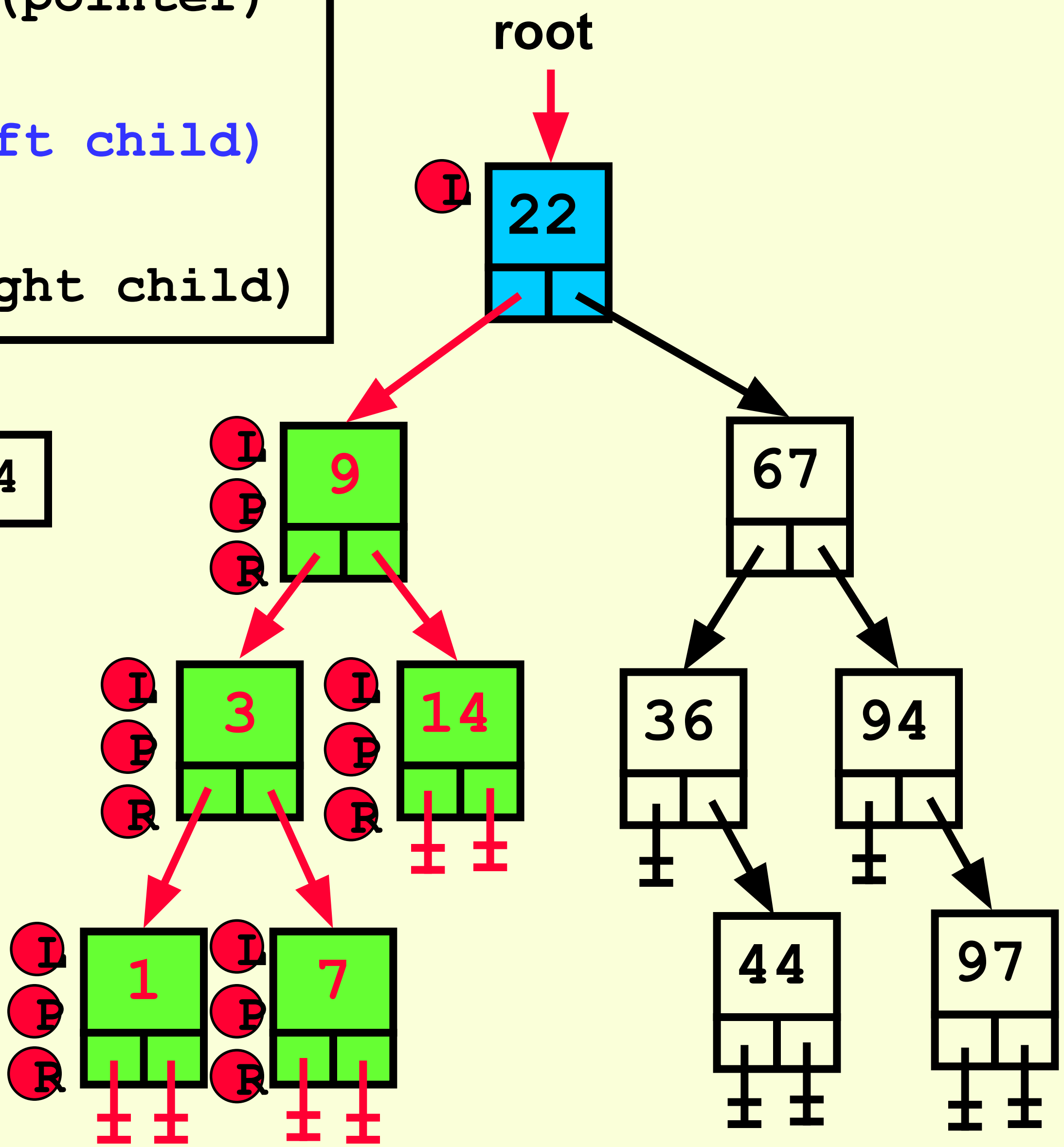
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14
```



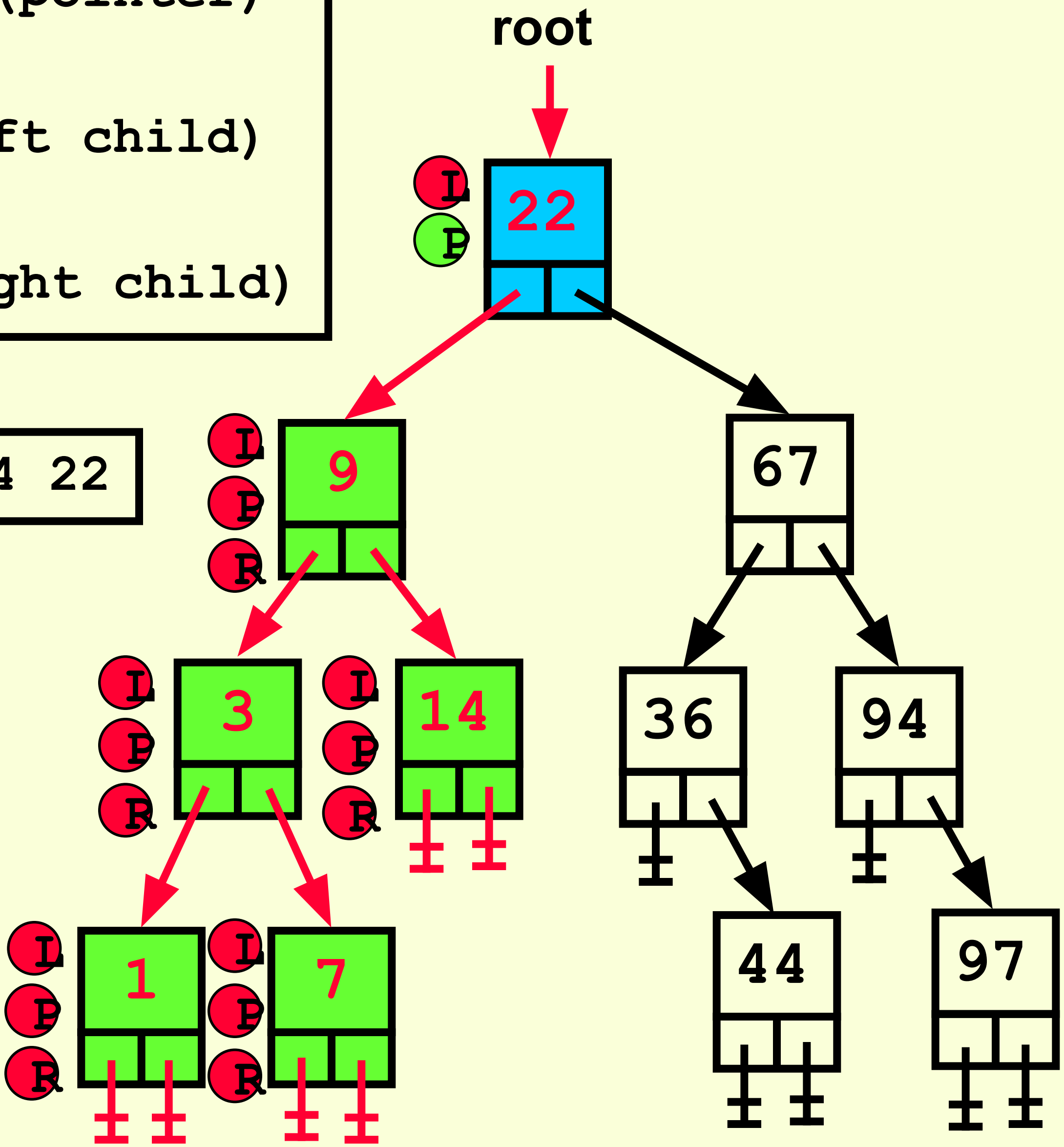
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14
```



```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
```



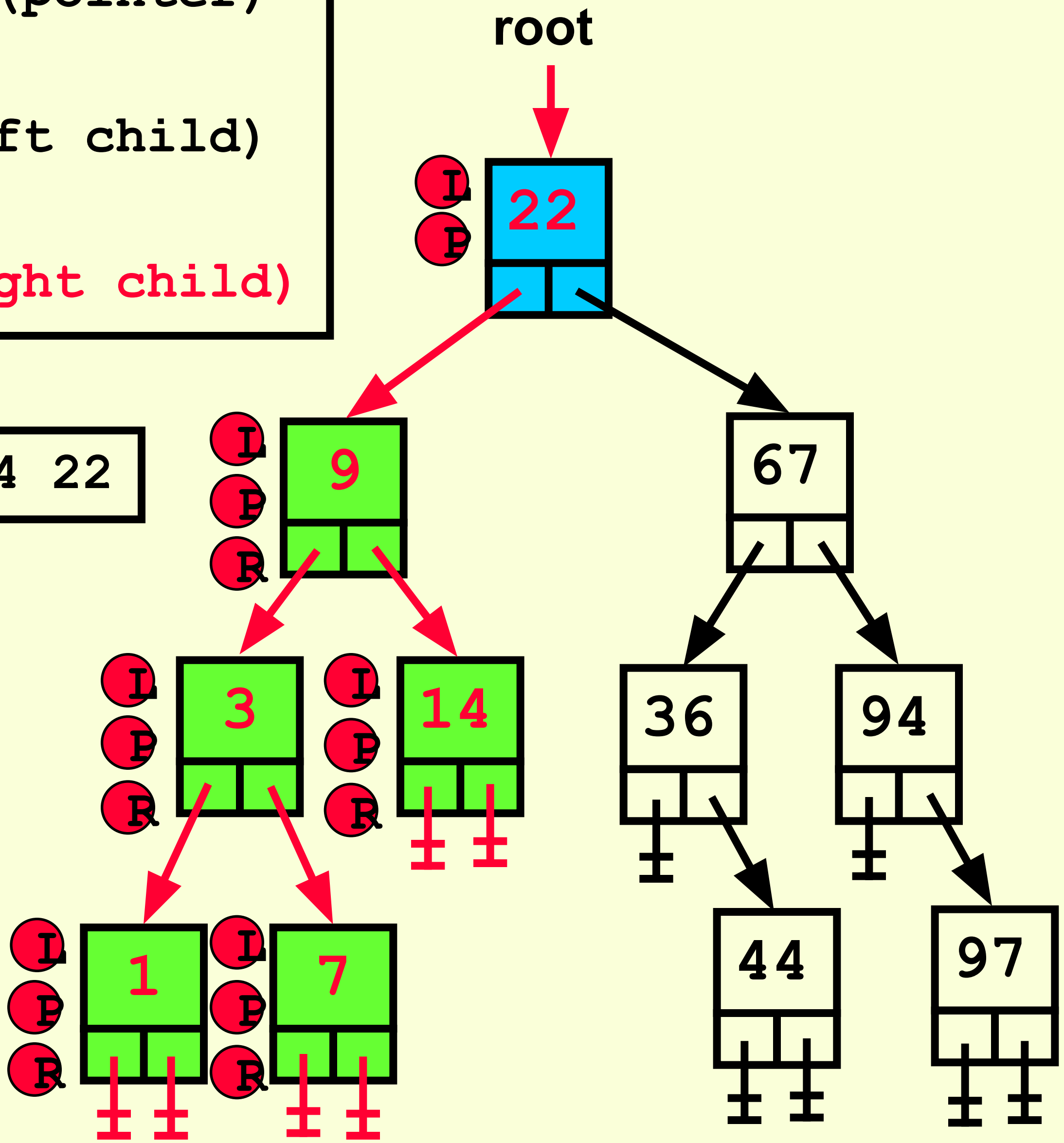
Continue?

Yes!

Enough Already!

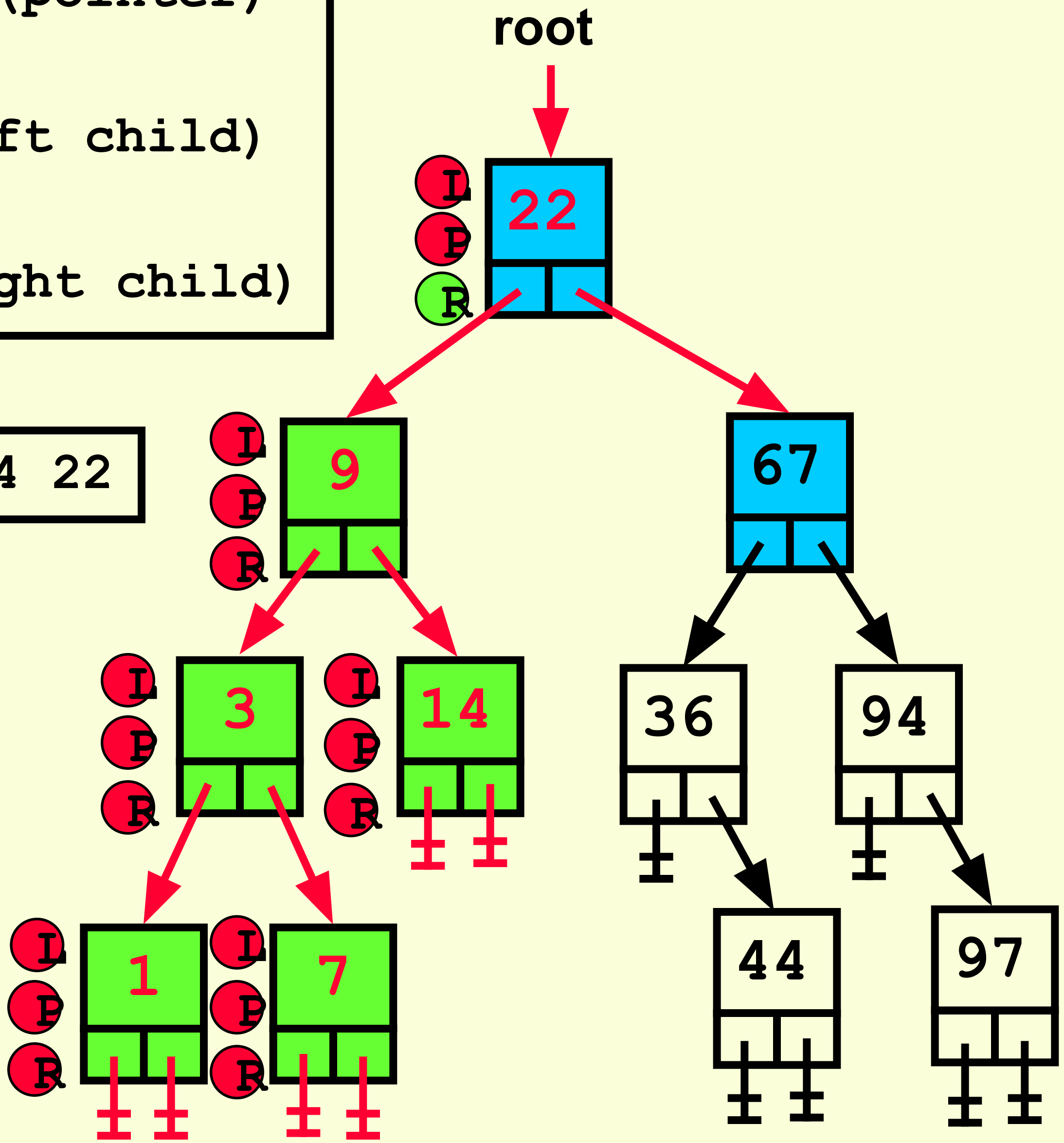
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

Output: 1 3 7 9 14 22



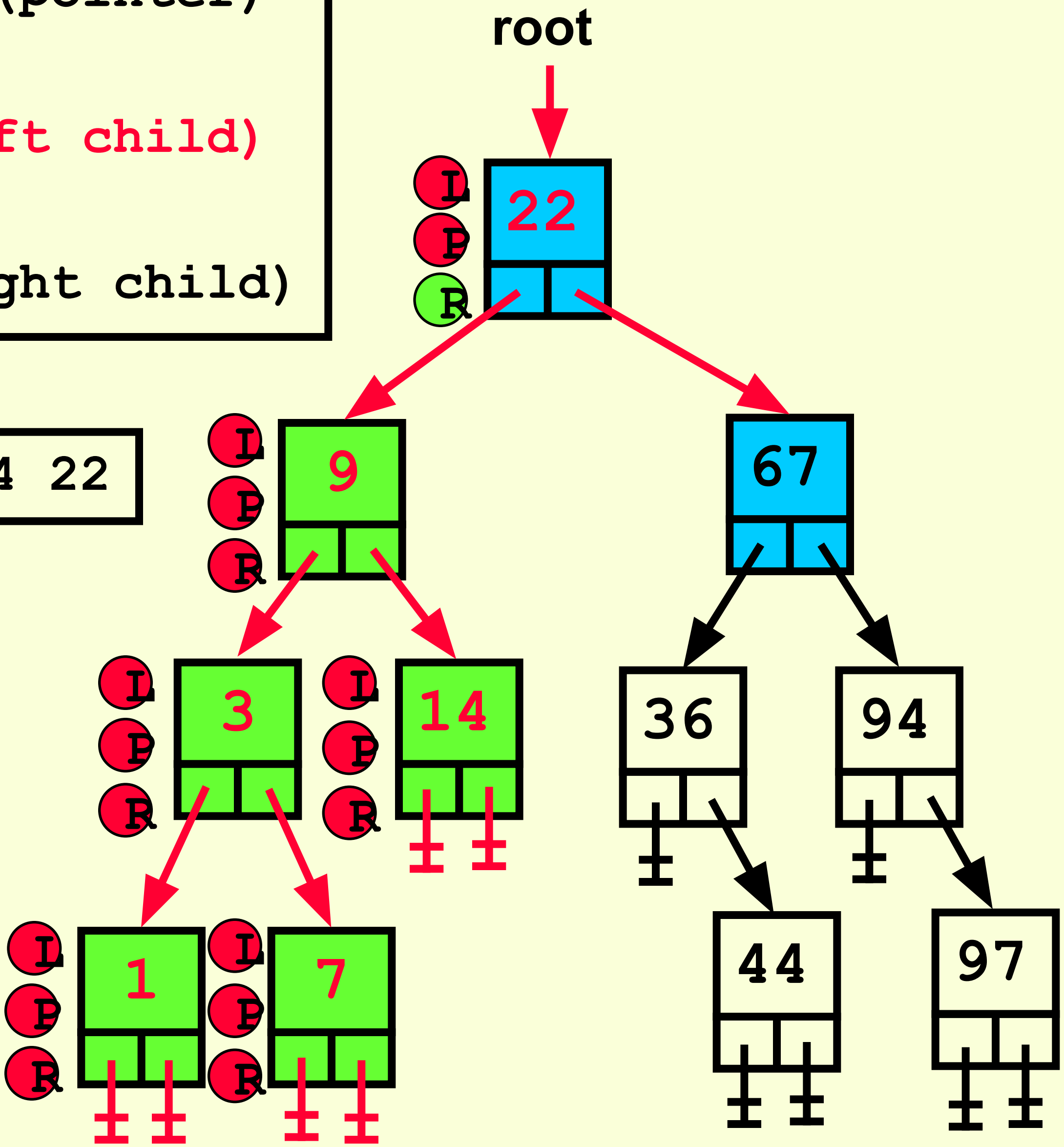
```
Proc InOrderPrint(pointer)
  pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
```



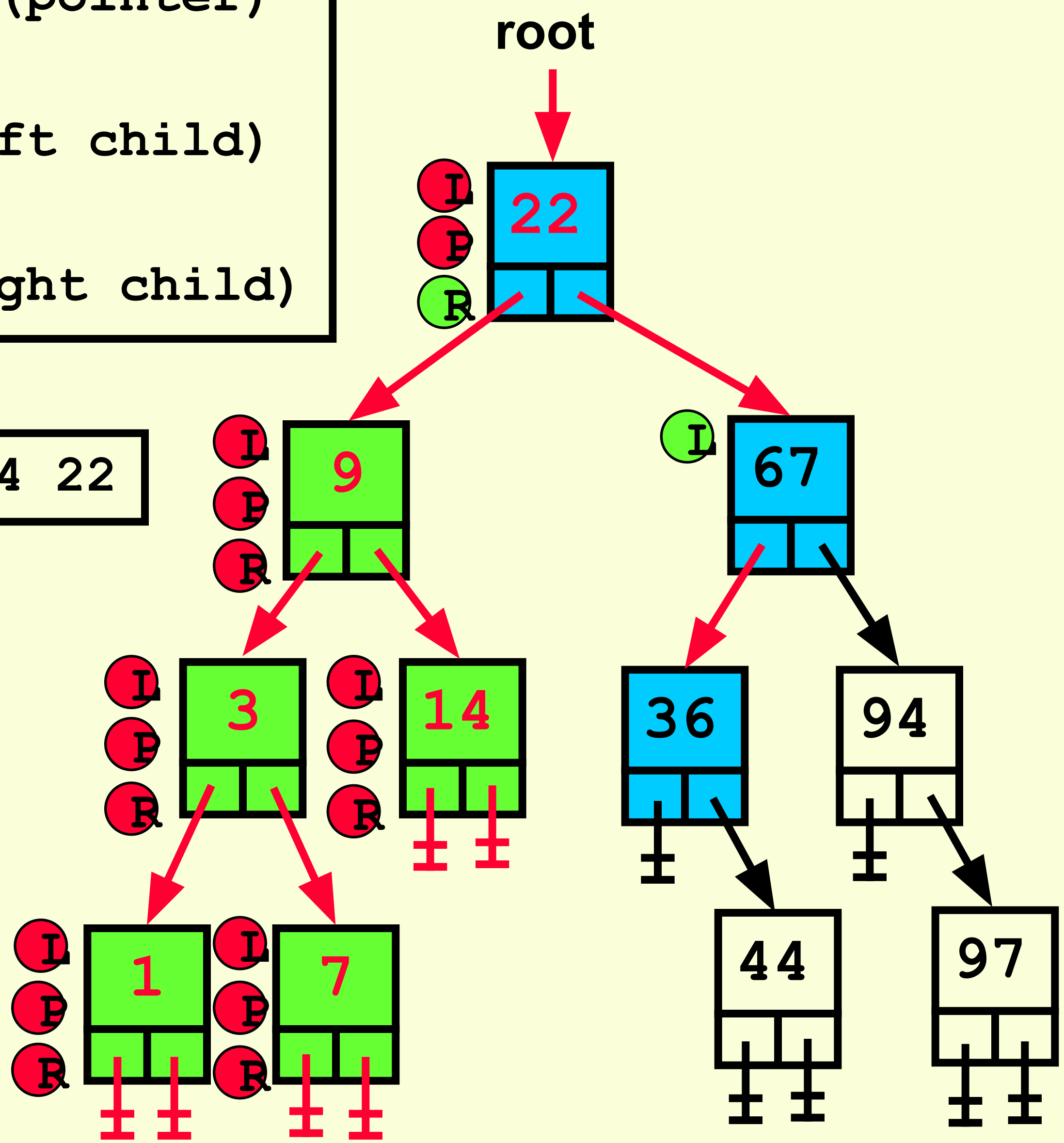
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
```



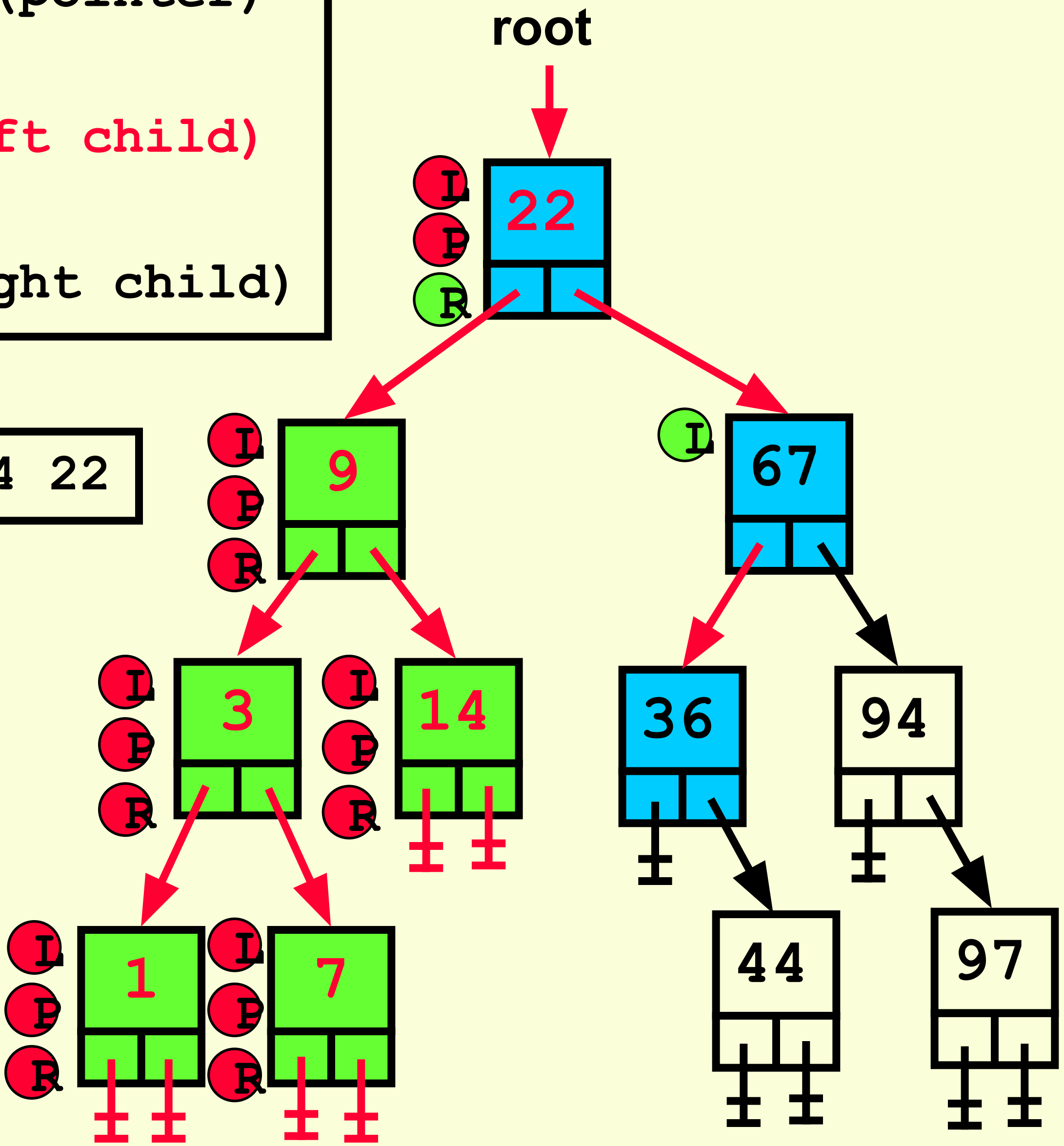
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

Output: 1 3 7 9 14 22



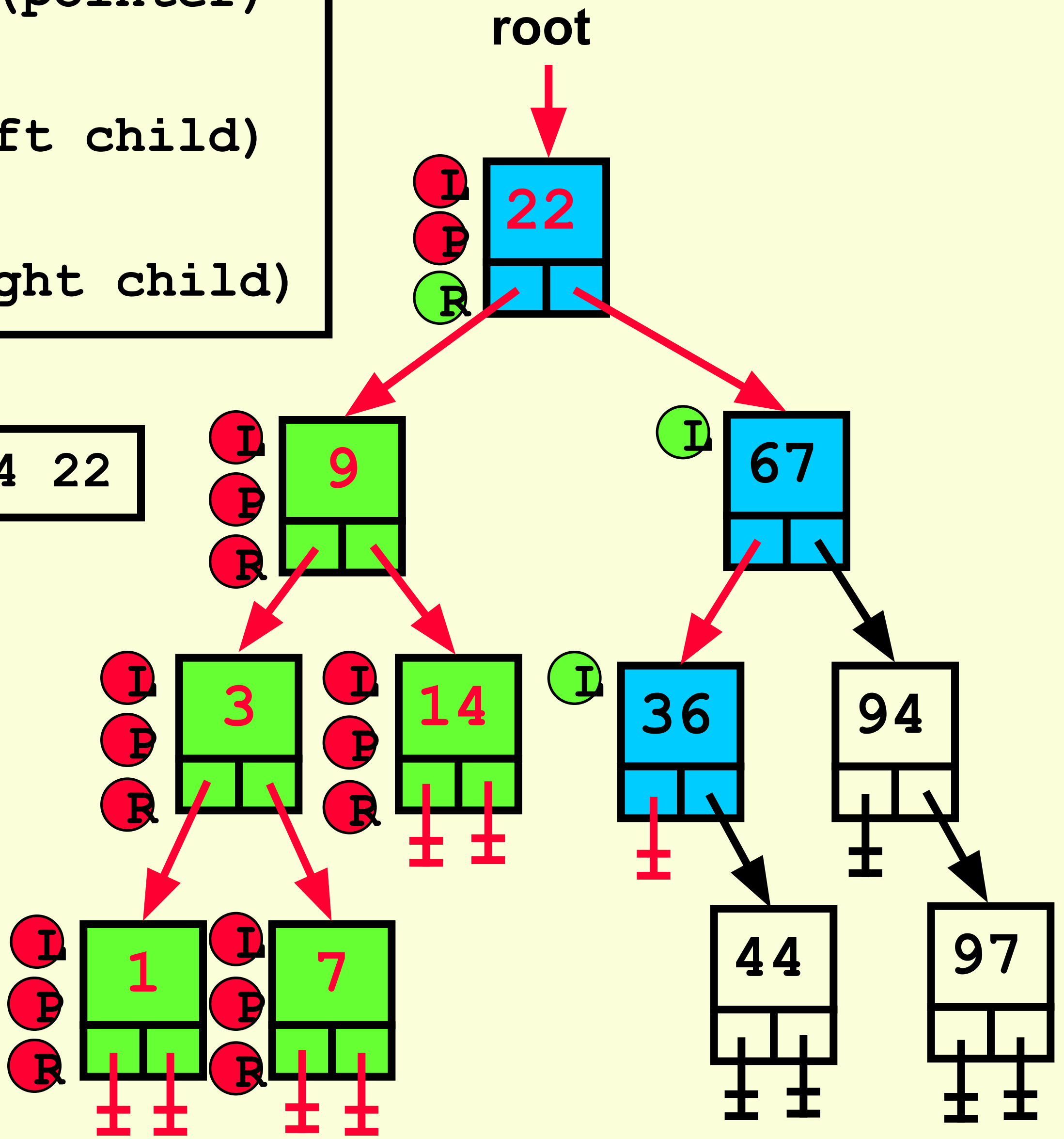
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
```



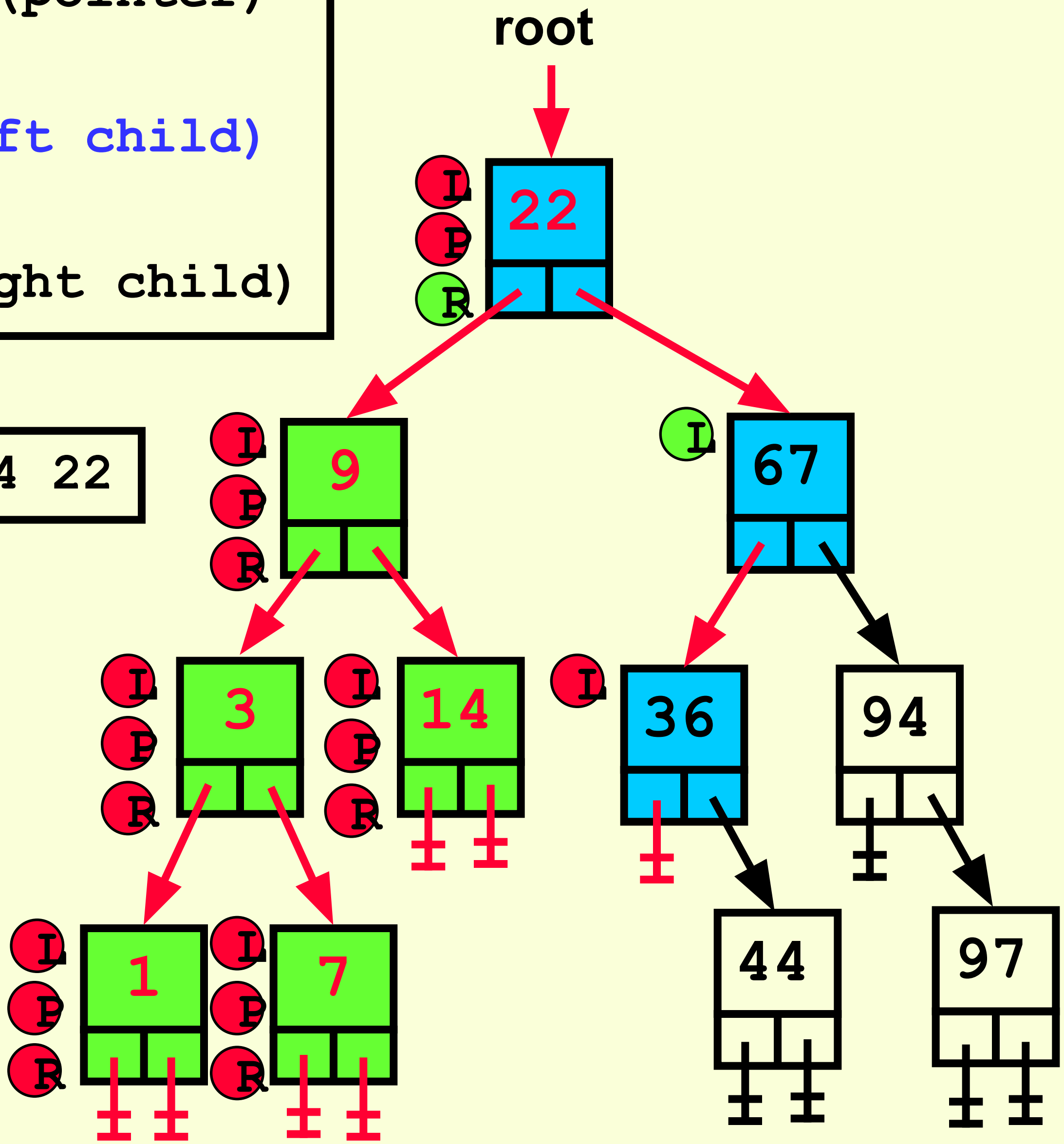
```
Proc InOrderPrint(pointer)
  pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
```



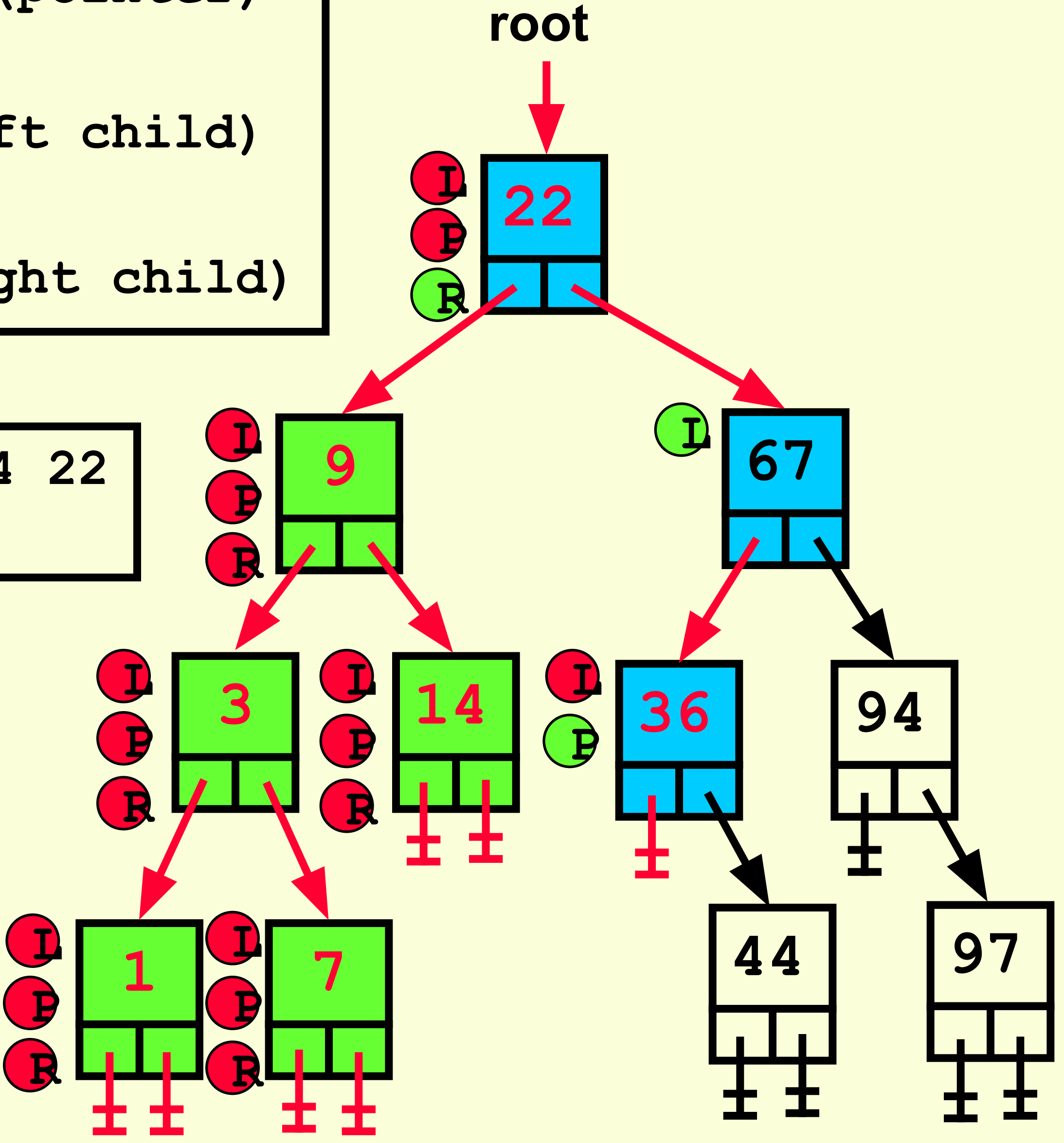
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
```



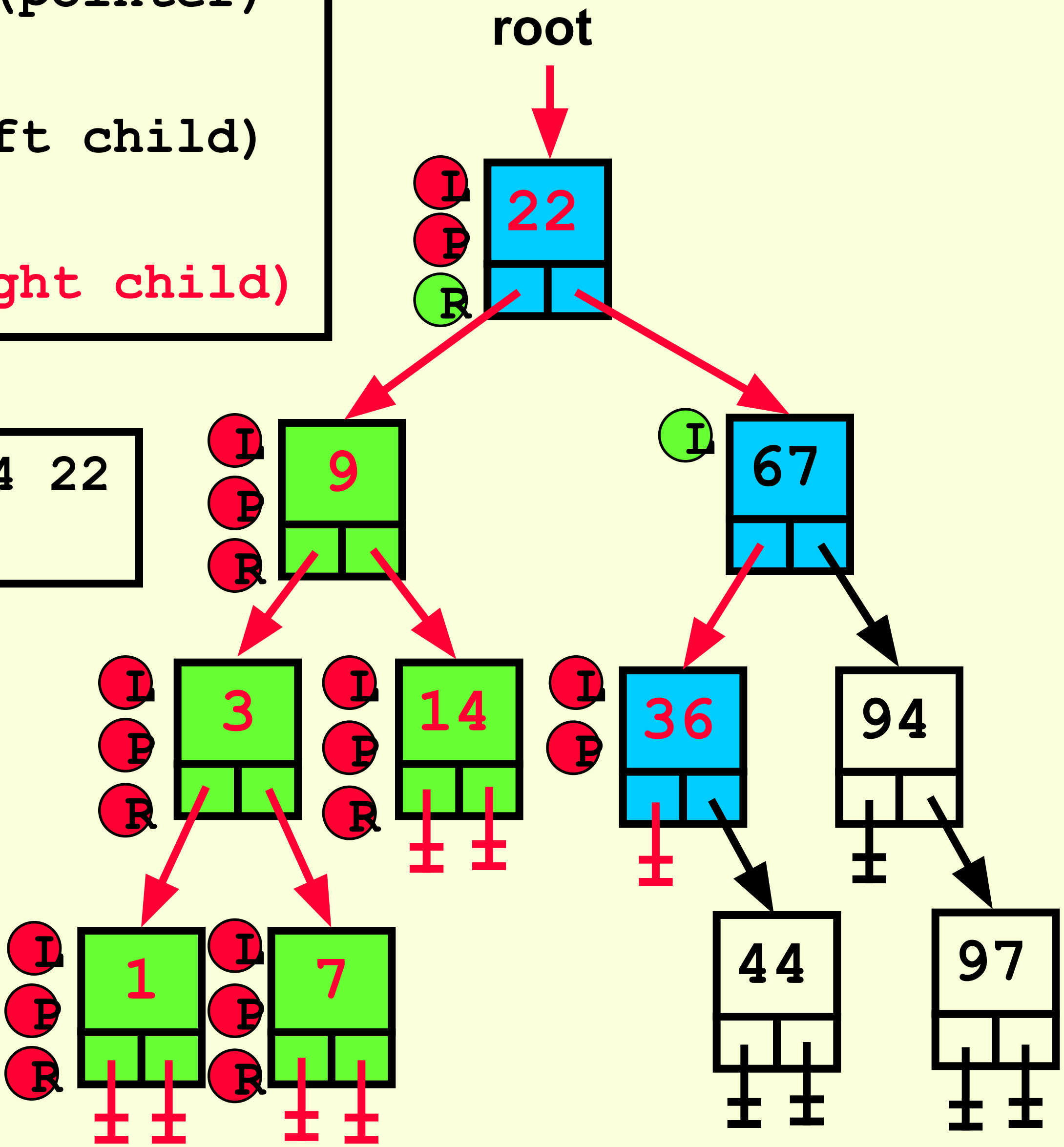
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36
```



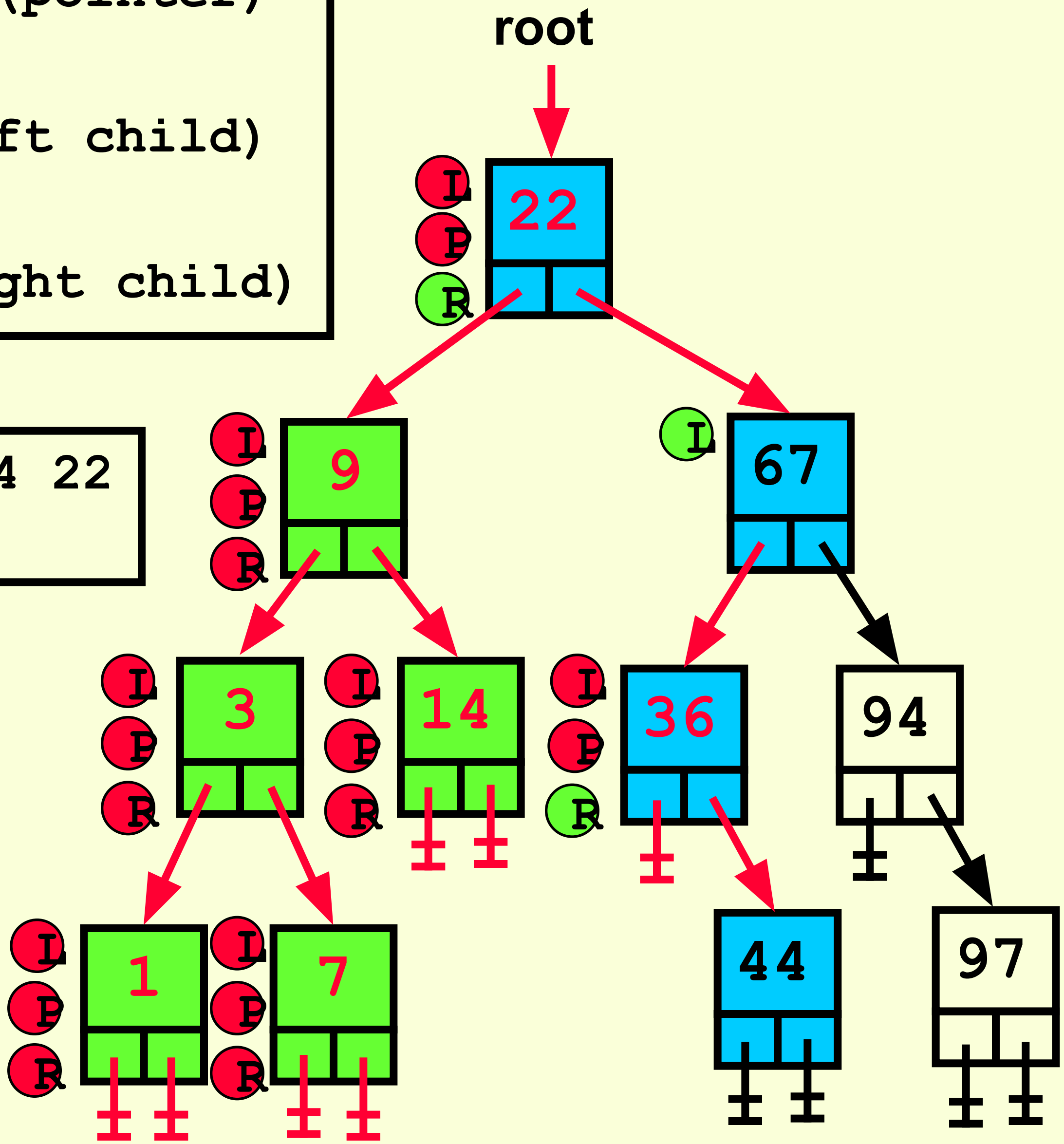
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36
```



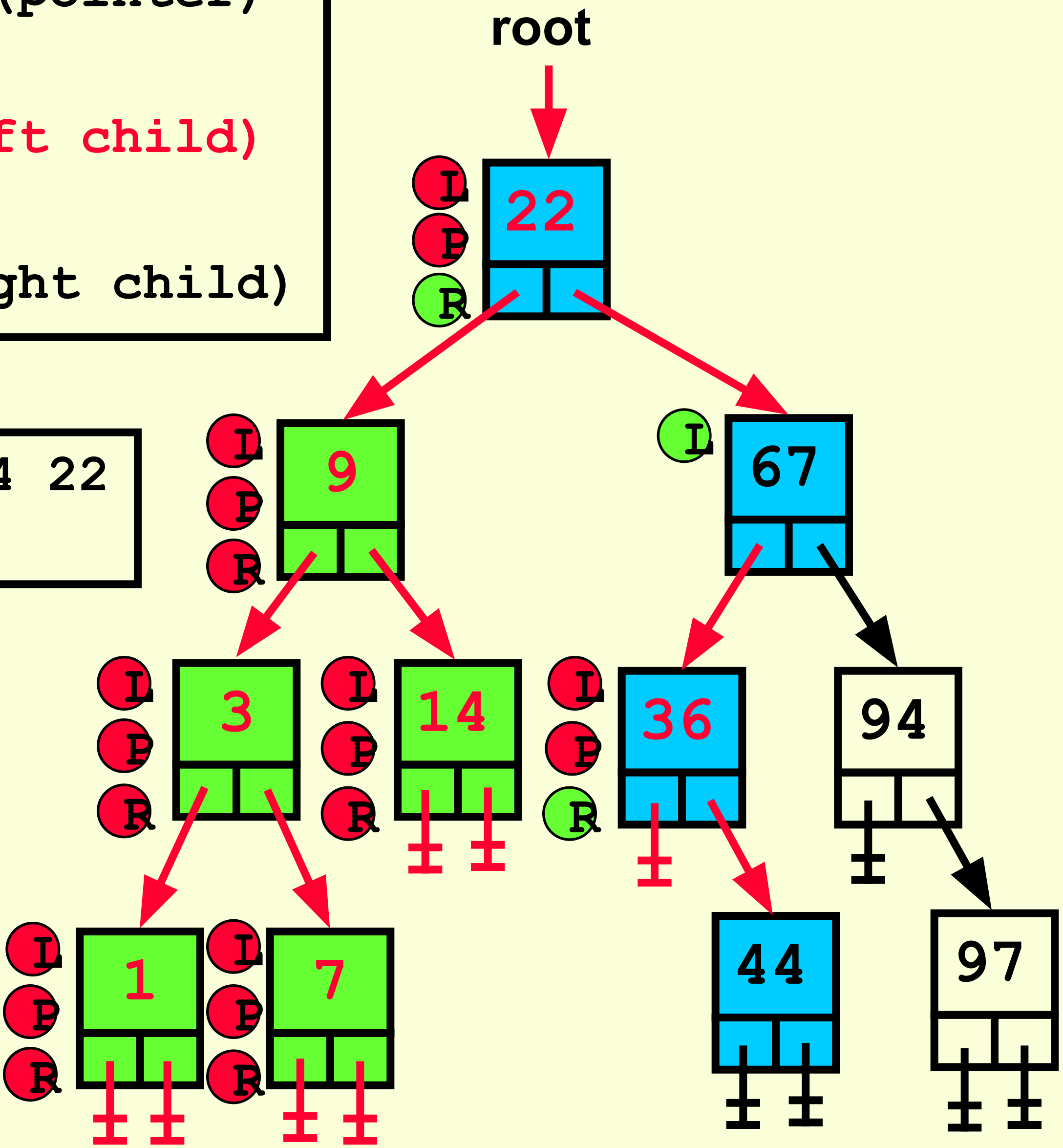
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36
```



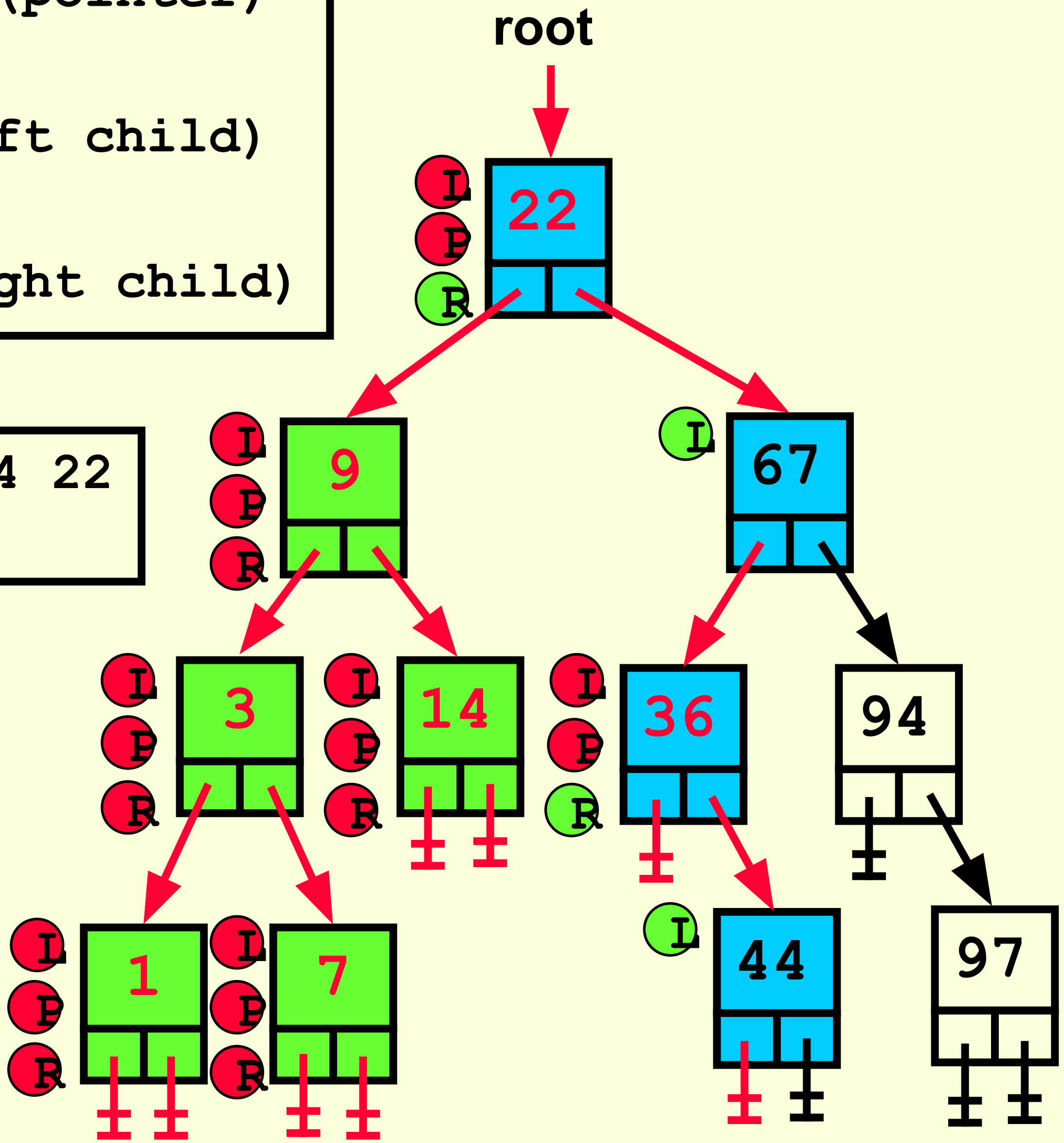
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36
```



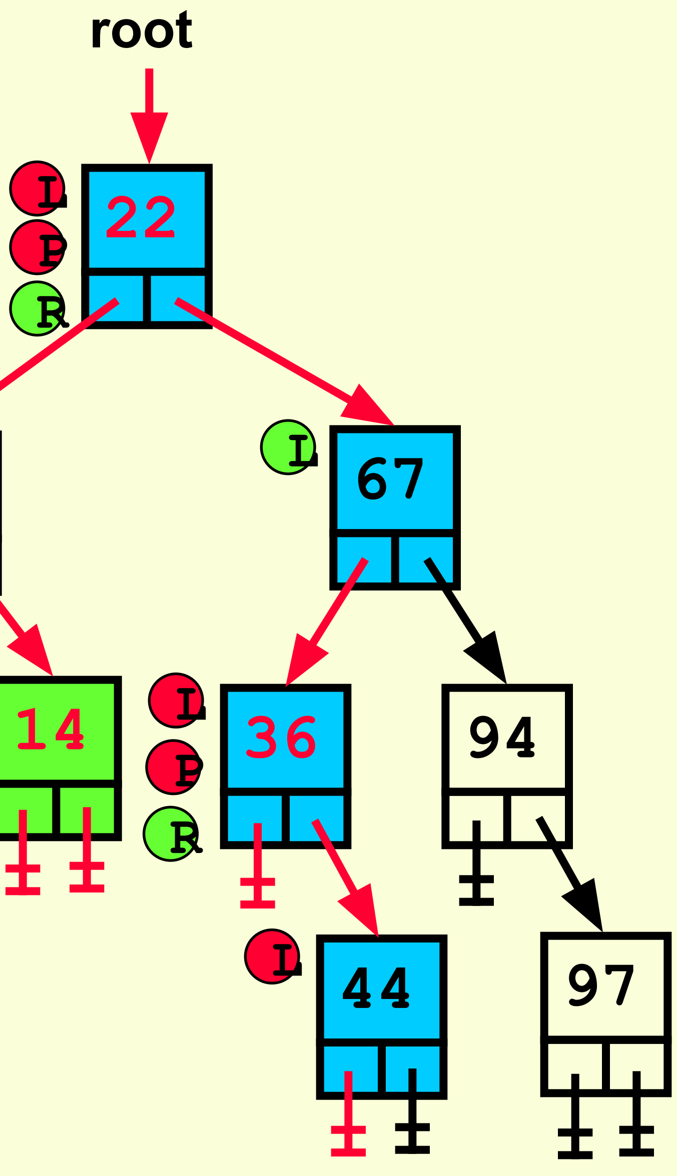
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36
```



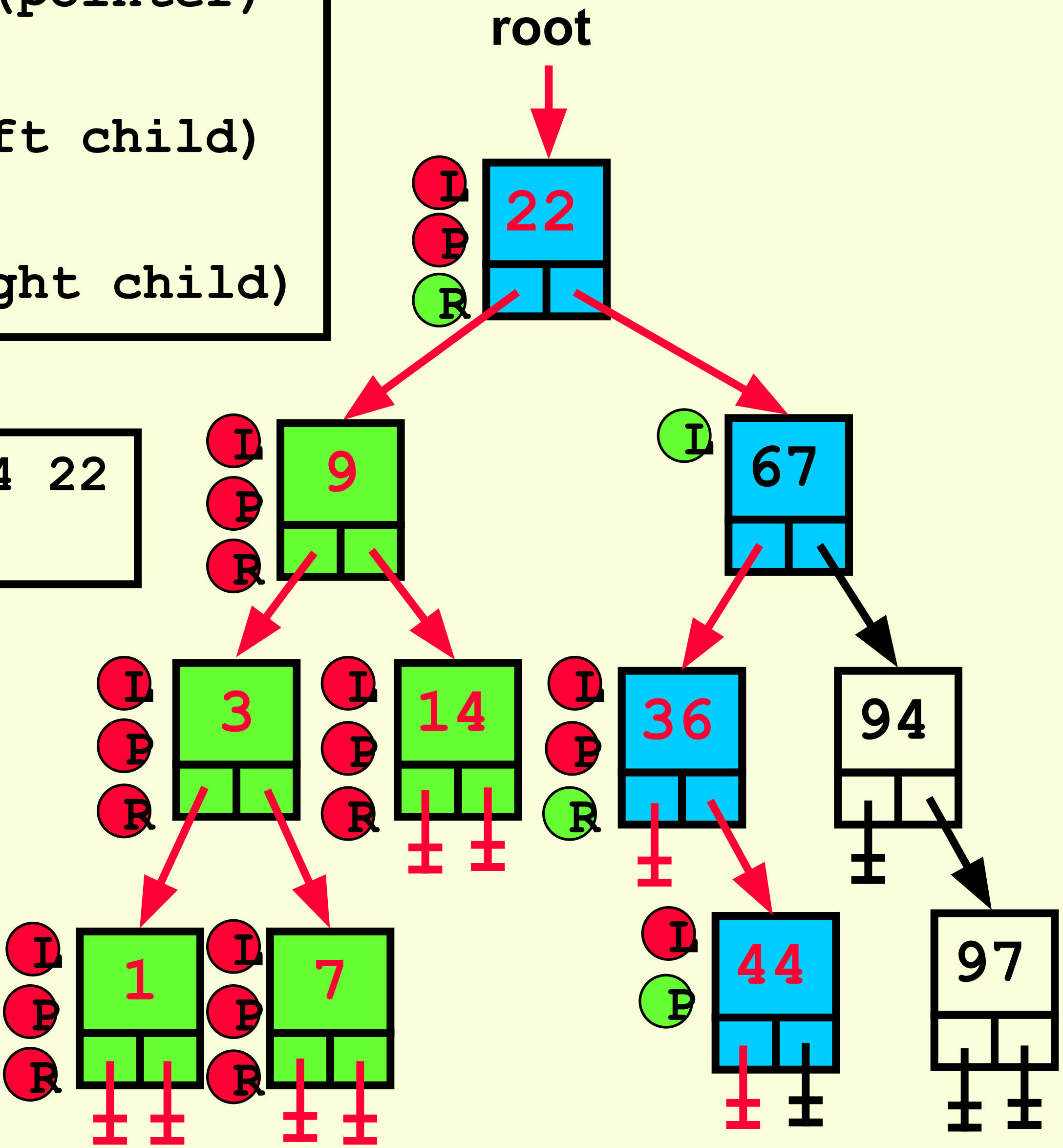
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36
```



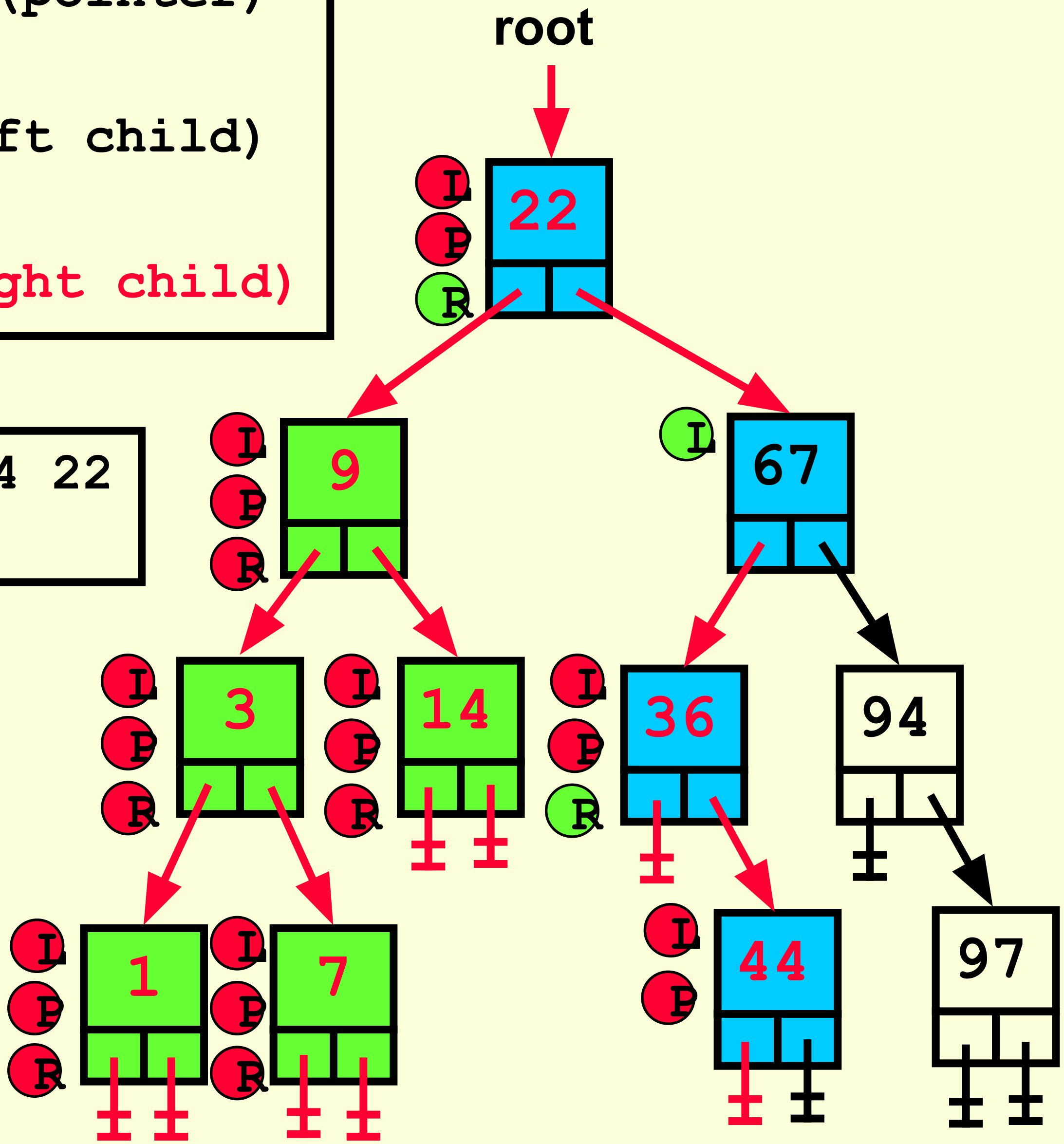
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44
```



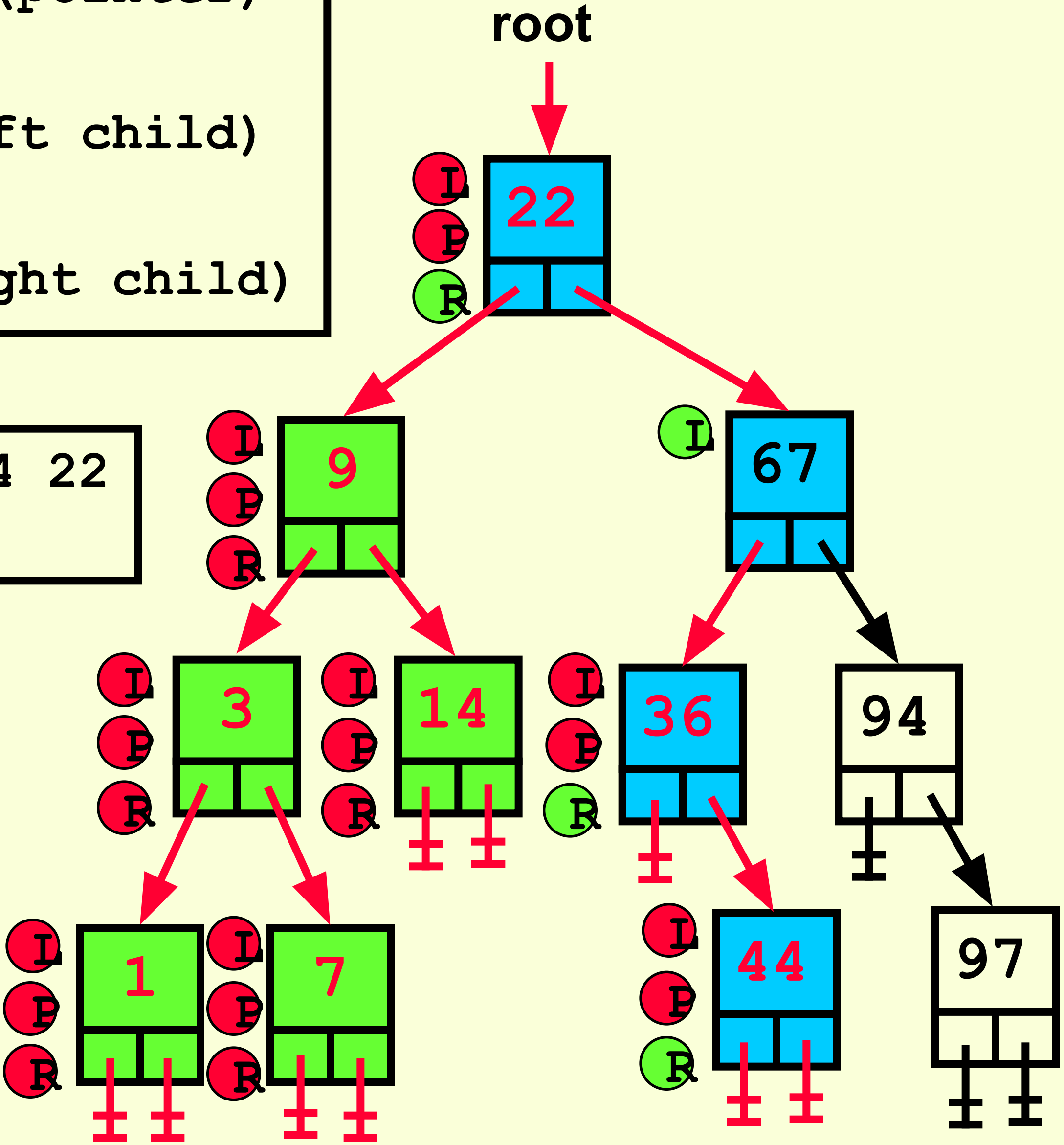
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44
```



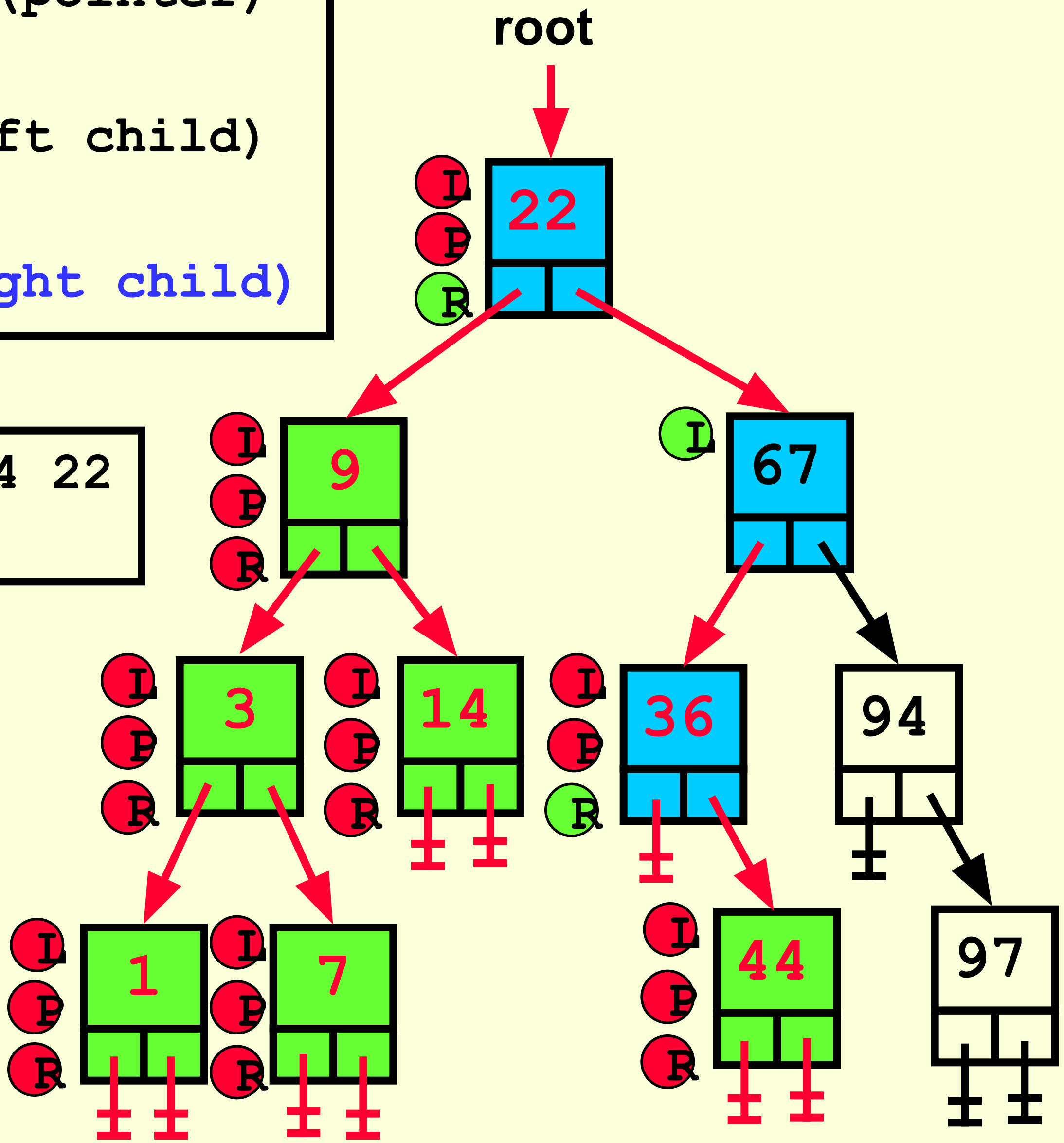
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44
```



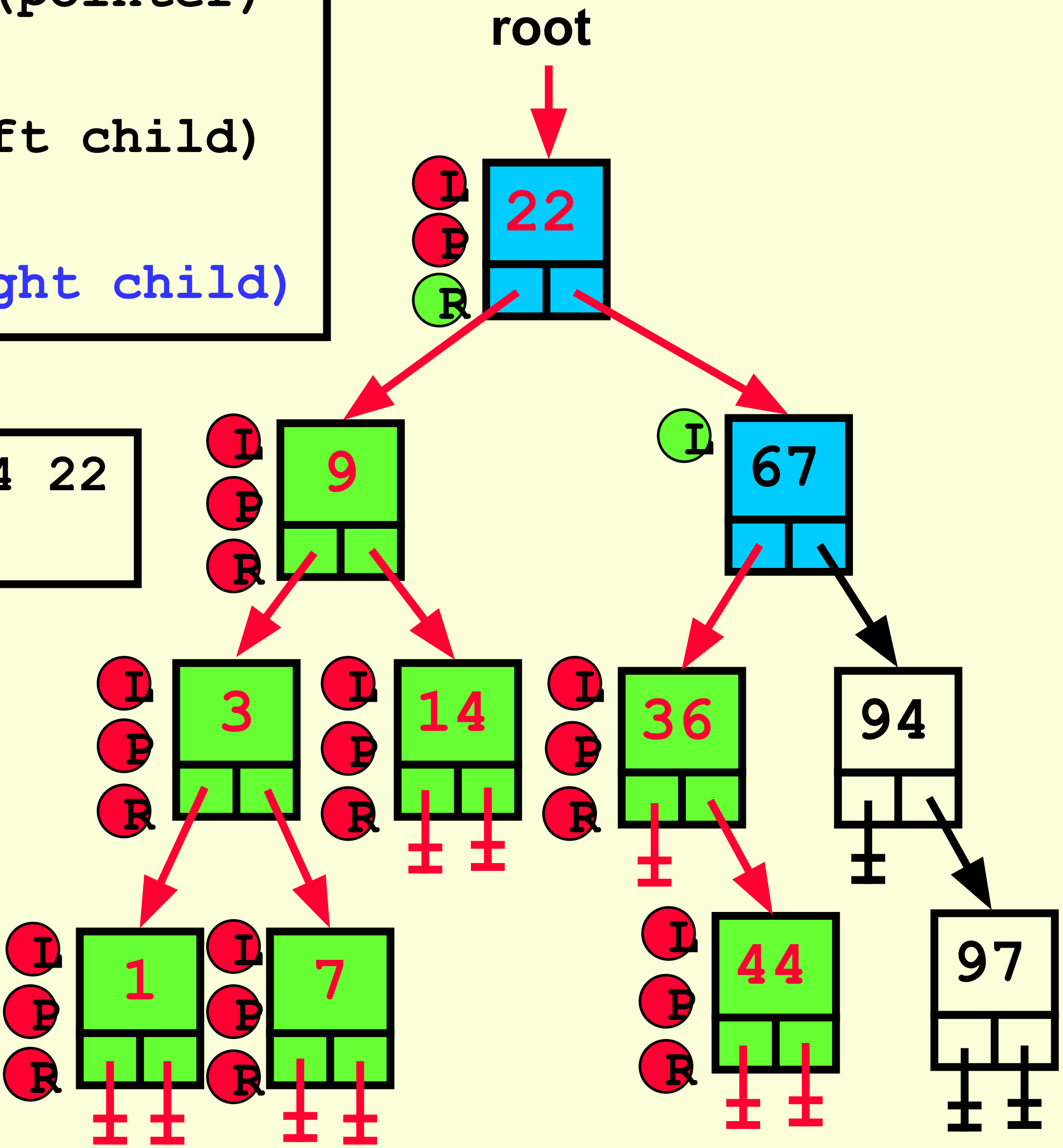
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44
```



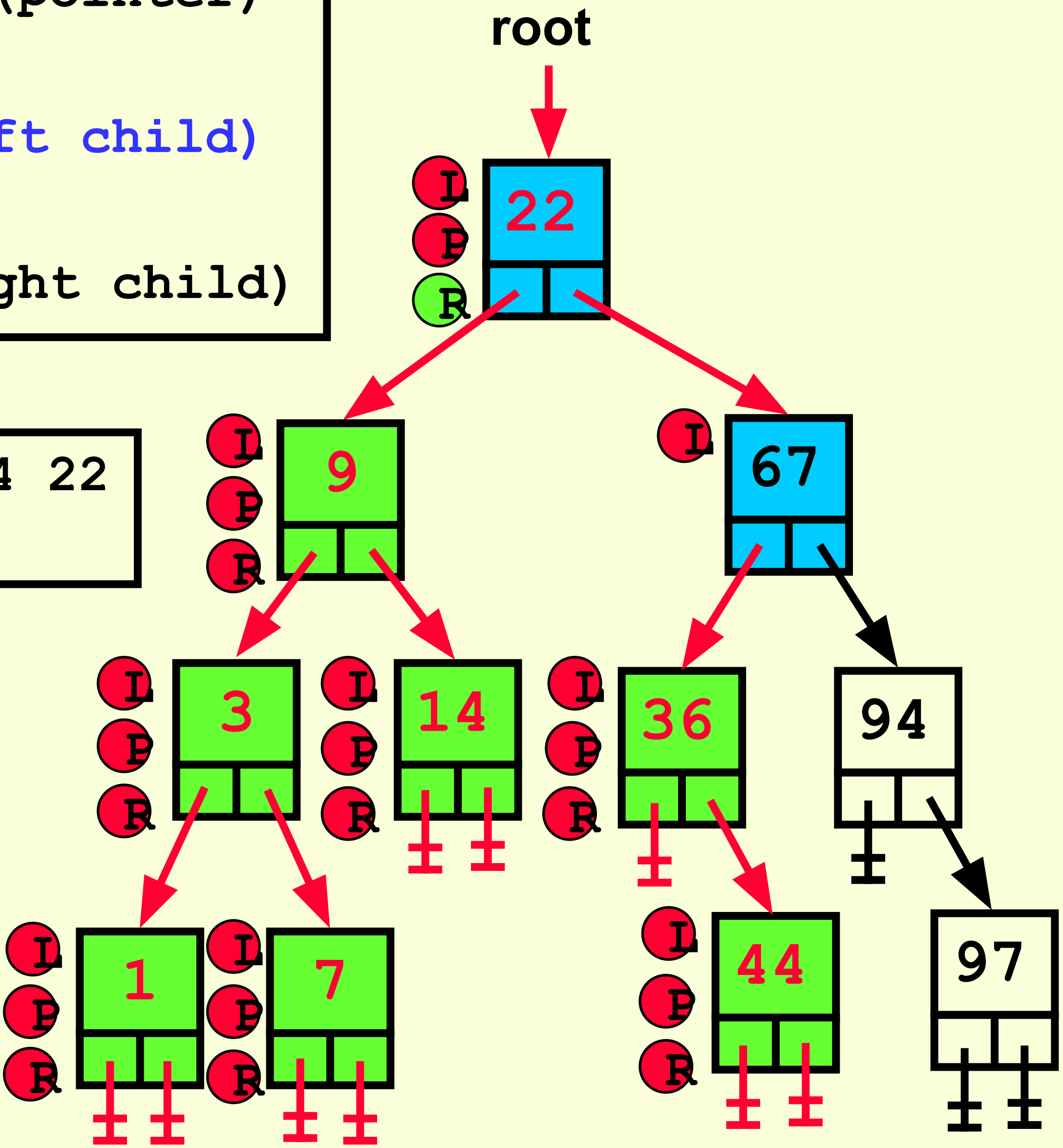
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44
```



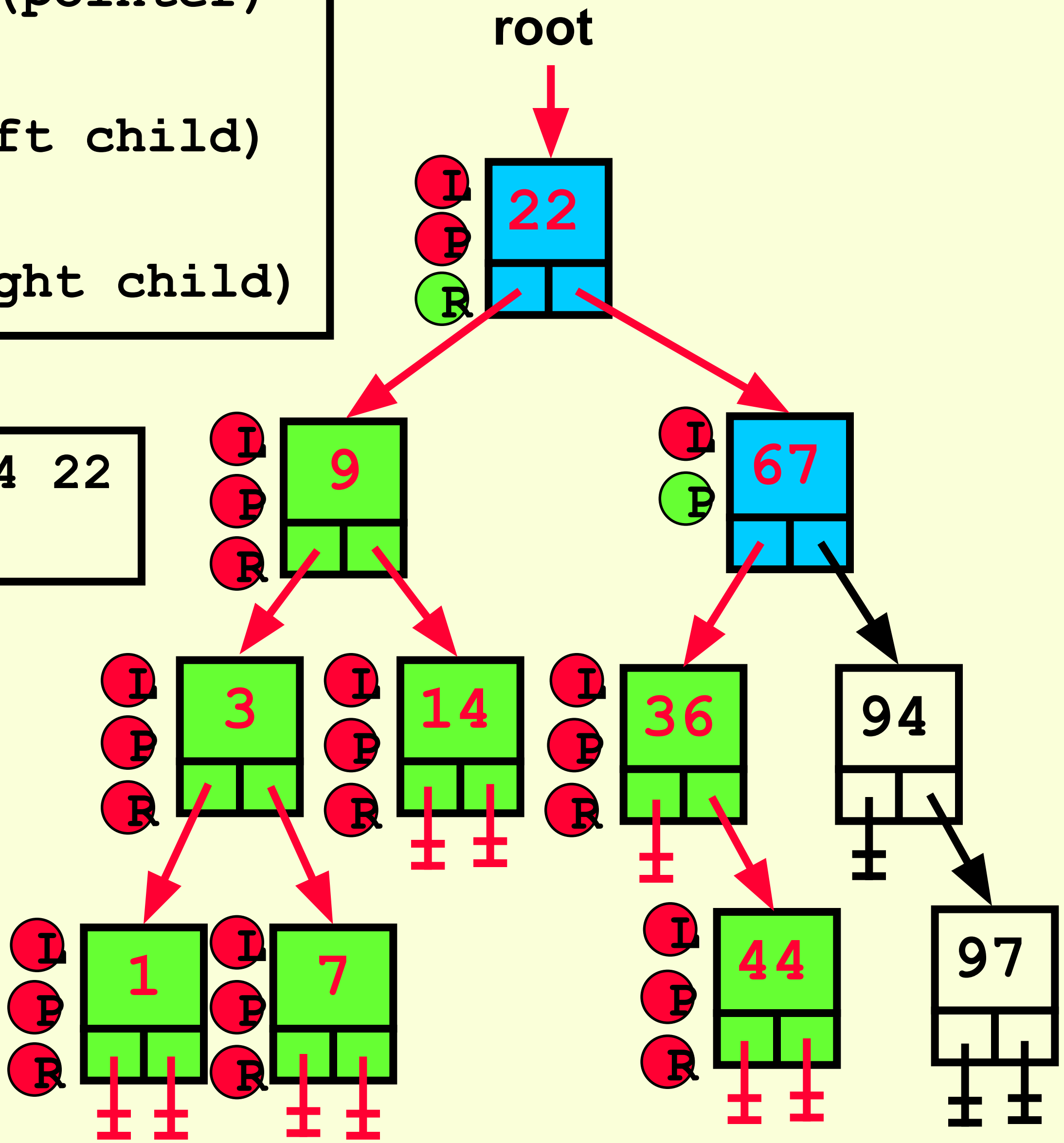
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44
```



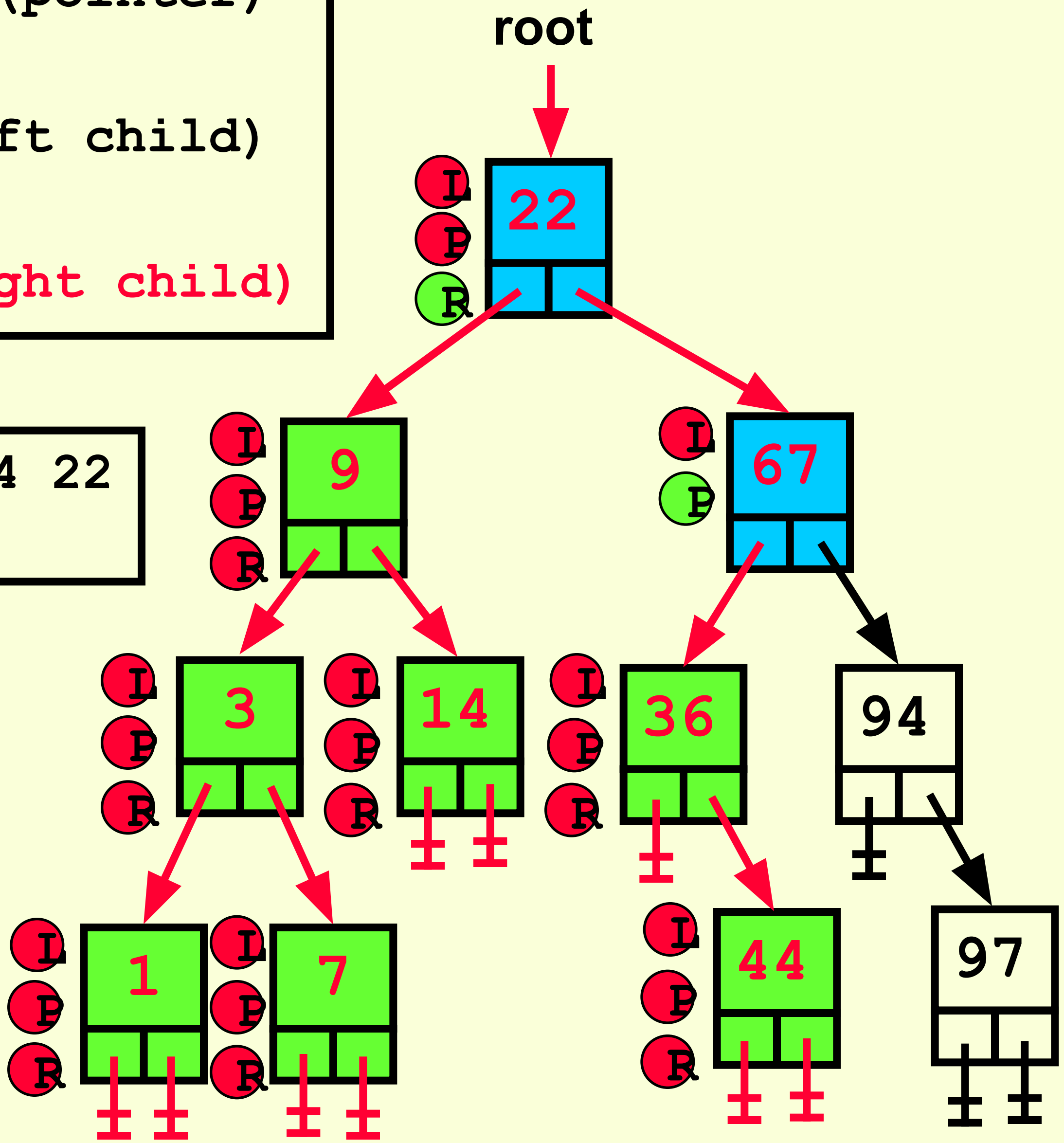
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67
```



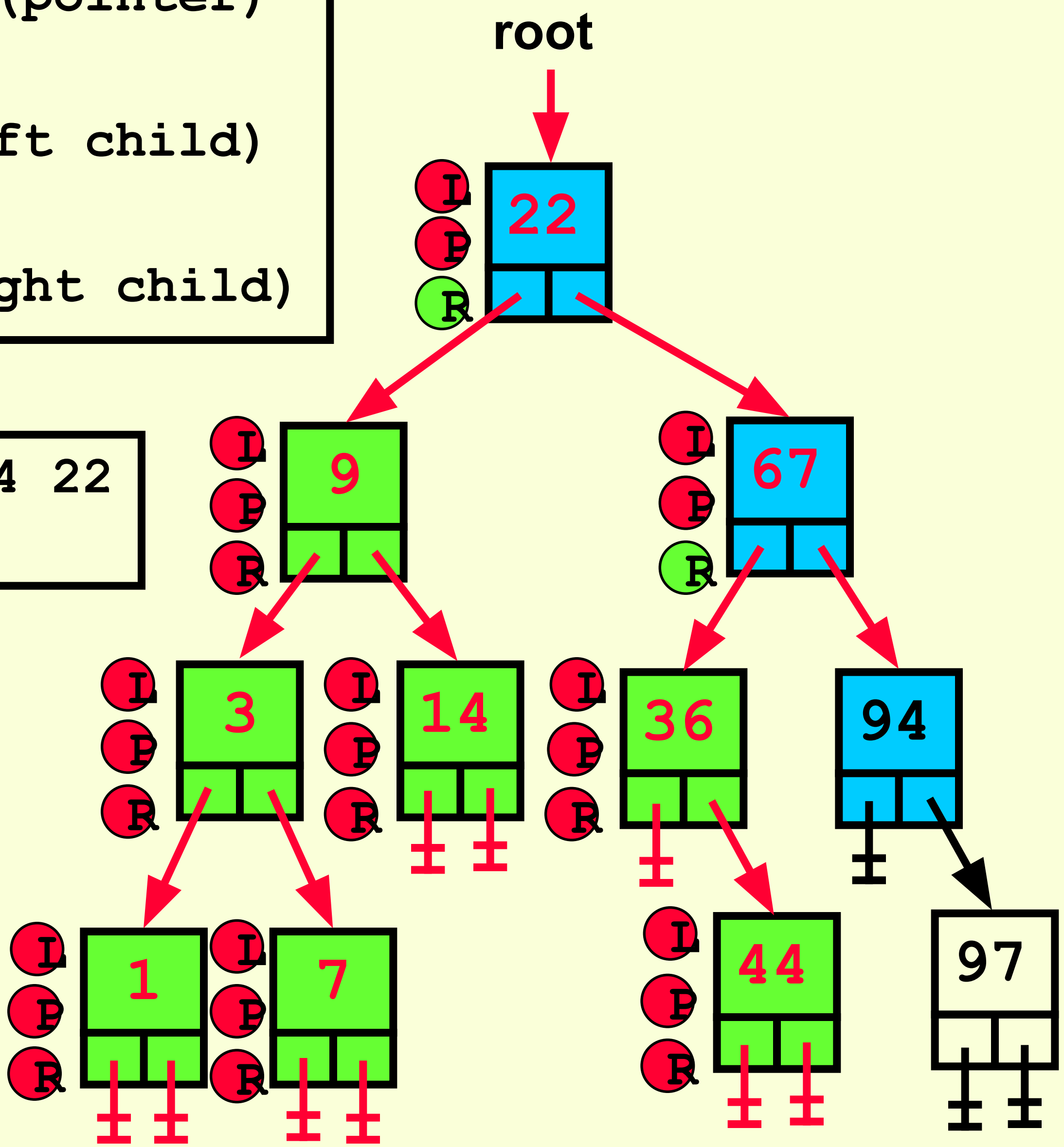
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67
```



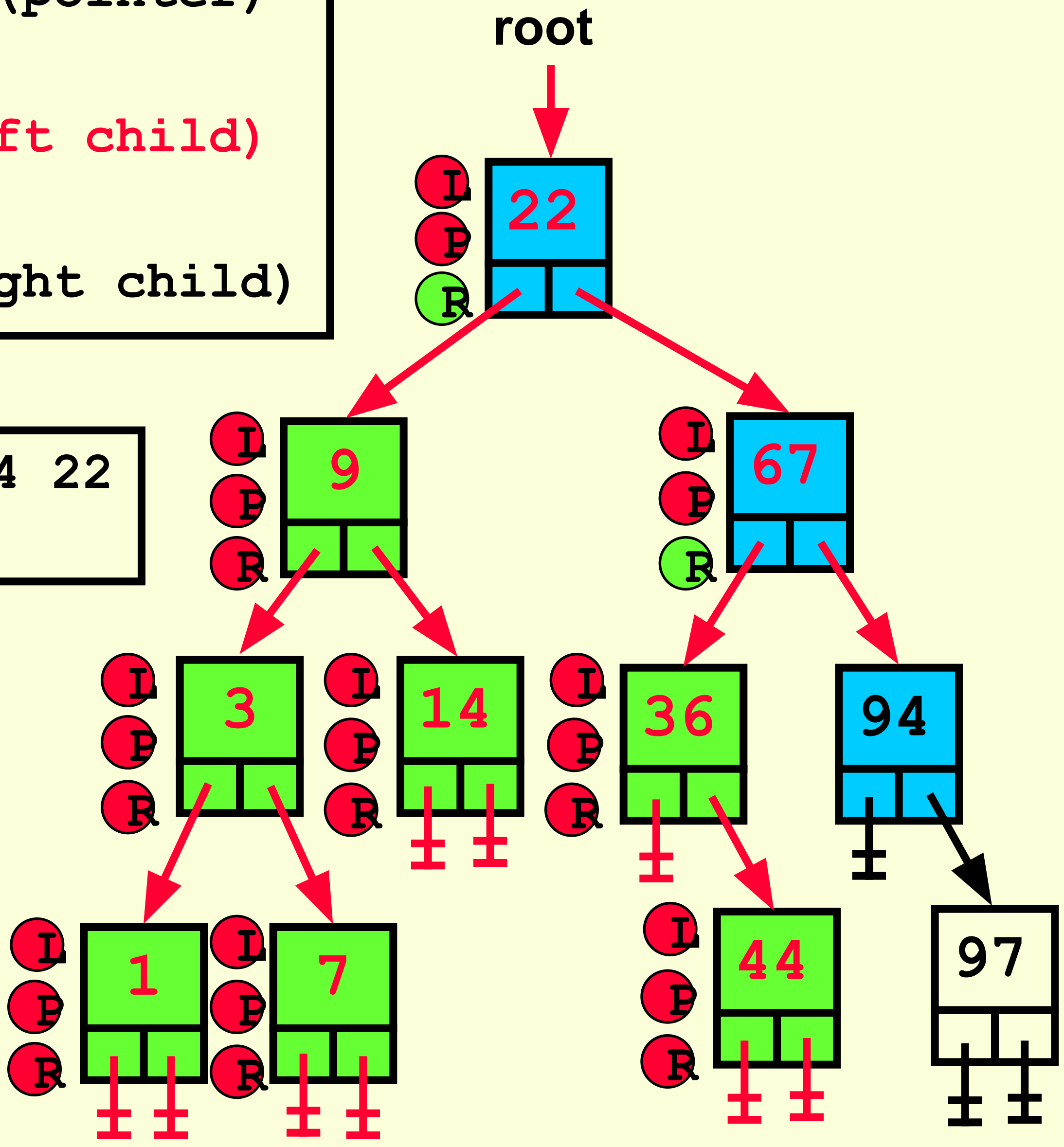
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67
```



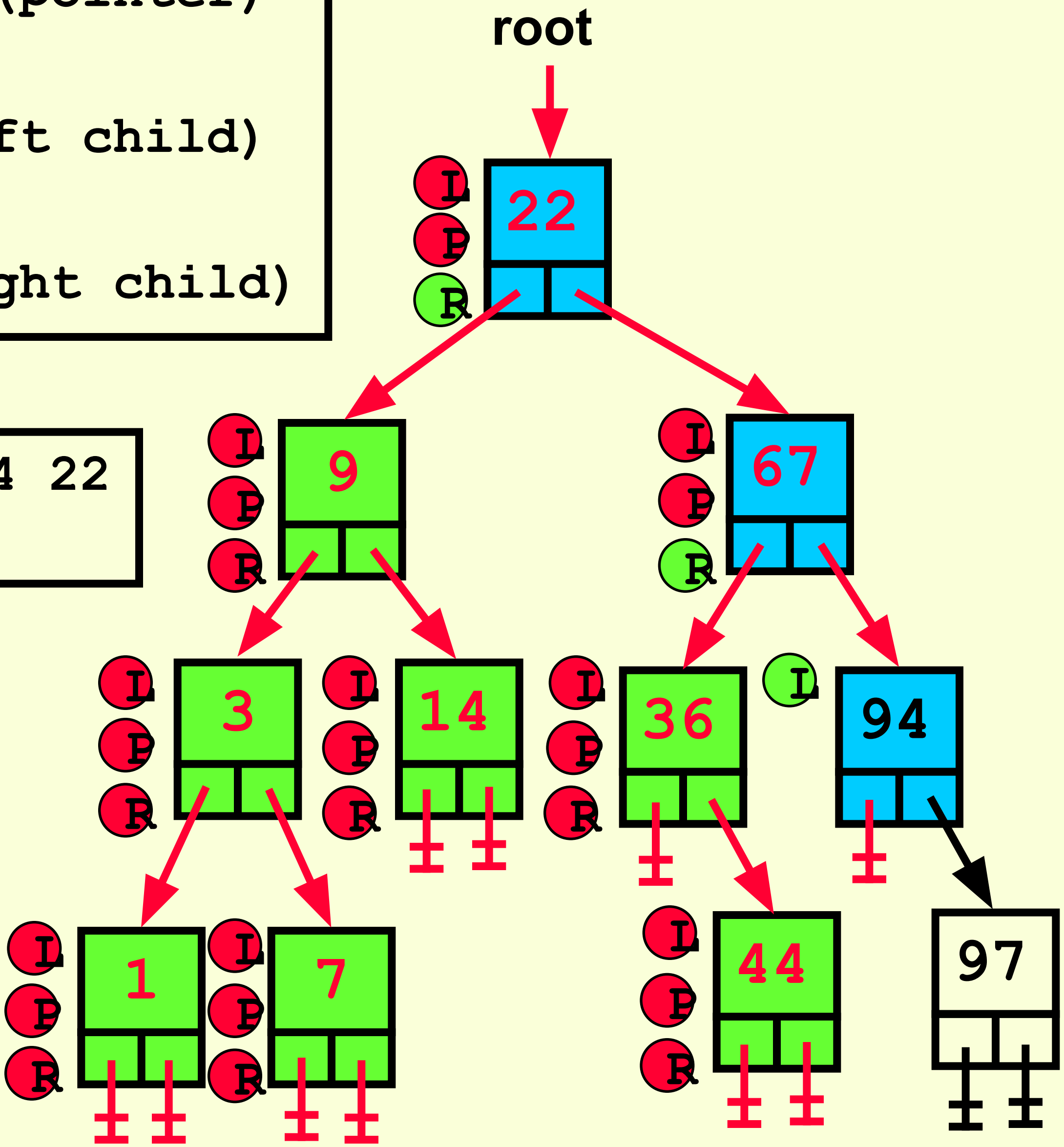
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67
```



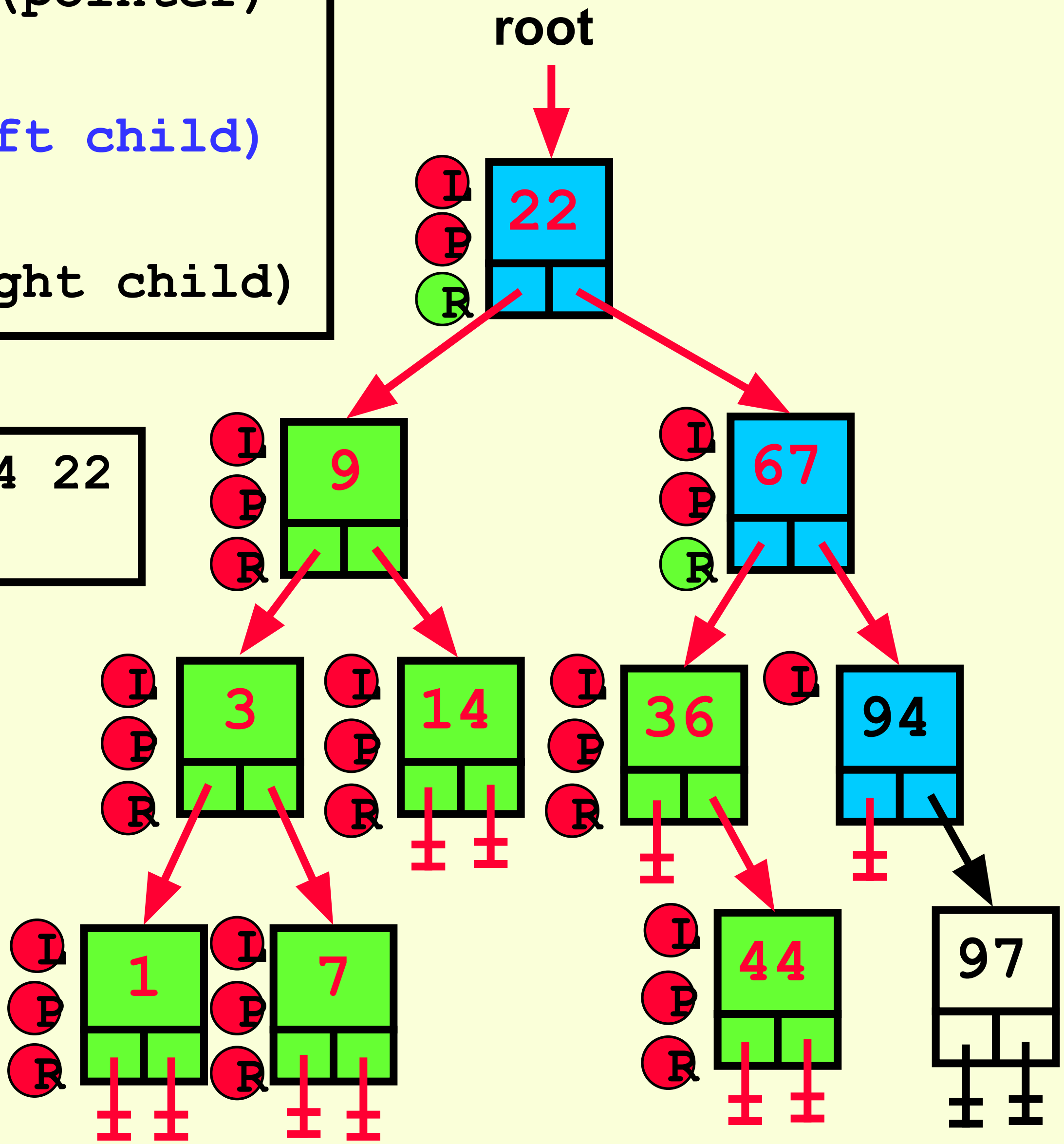
```
Proc InOrderPrint(pointer)
  pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67
```



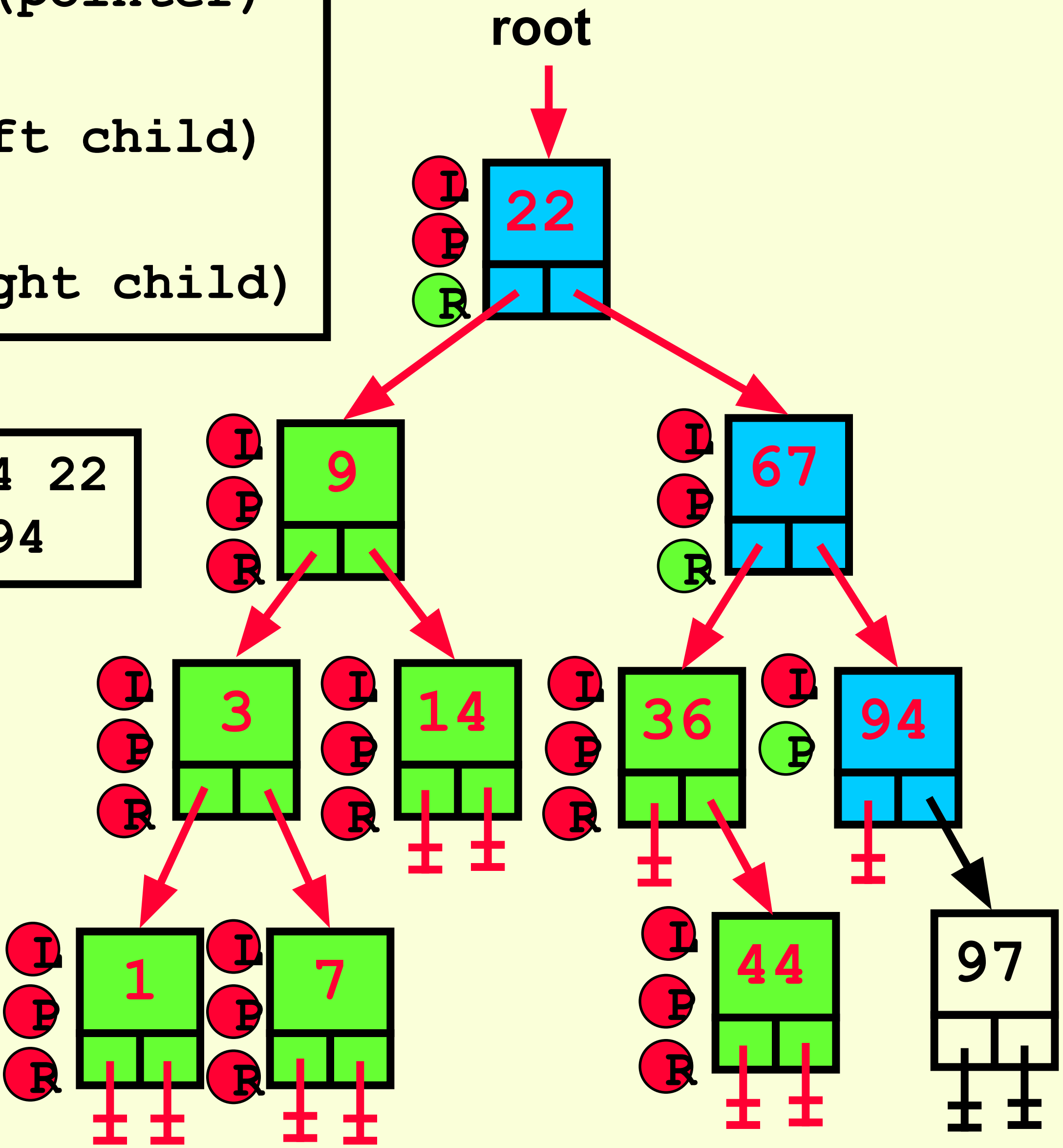
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67
```



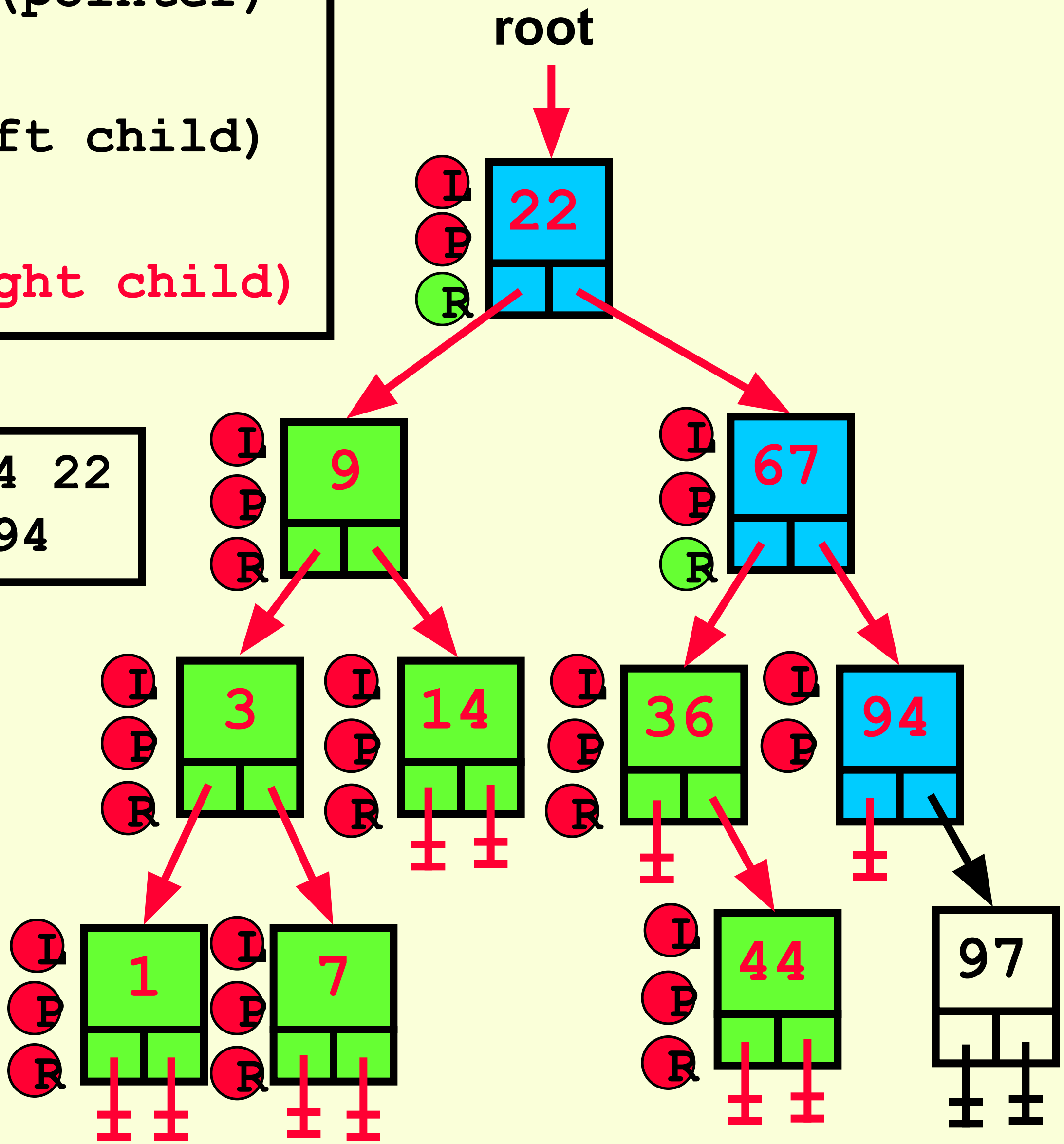
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94
```



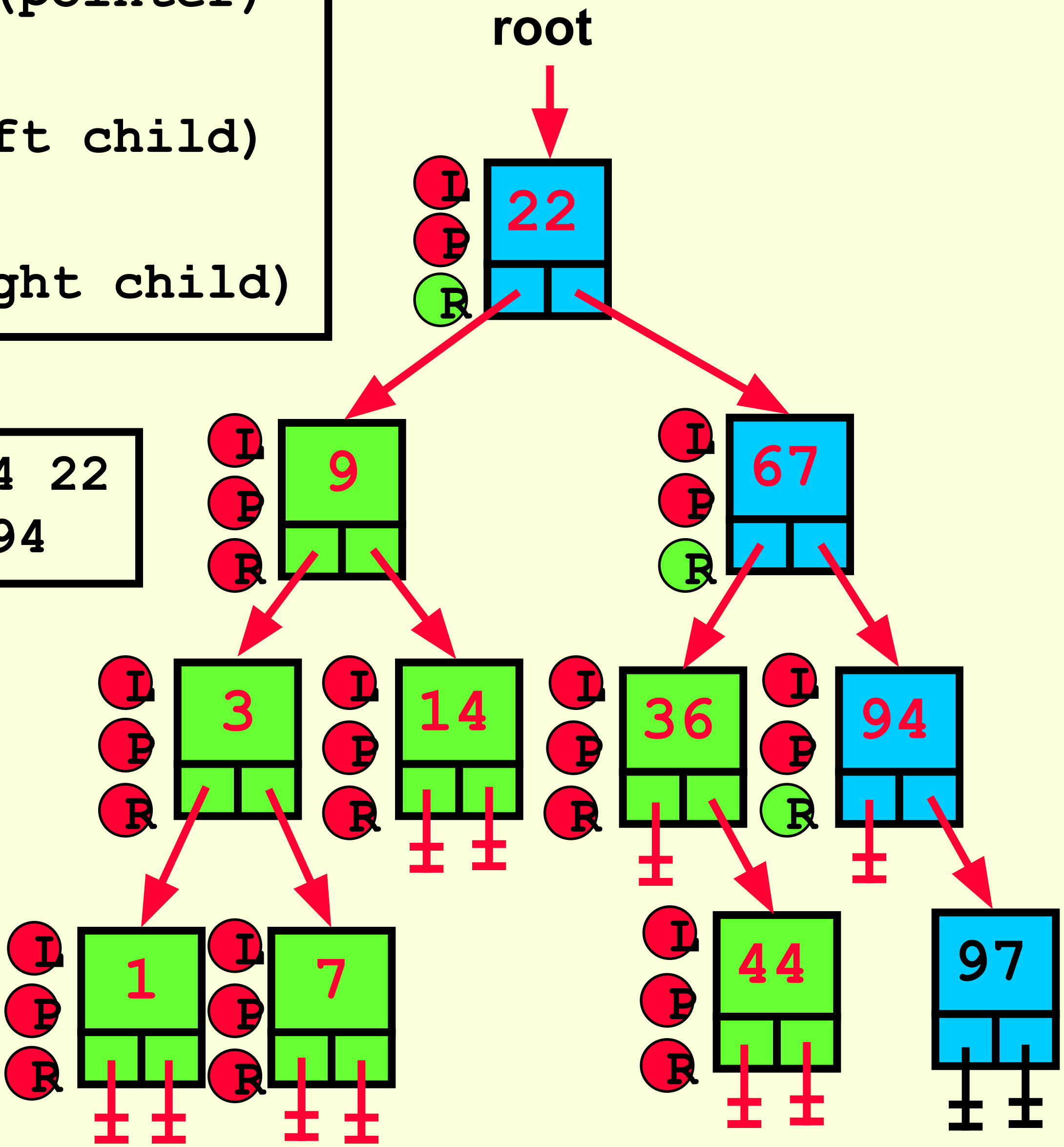
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94
```



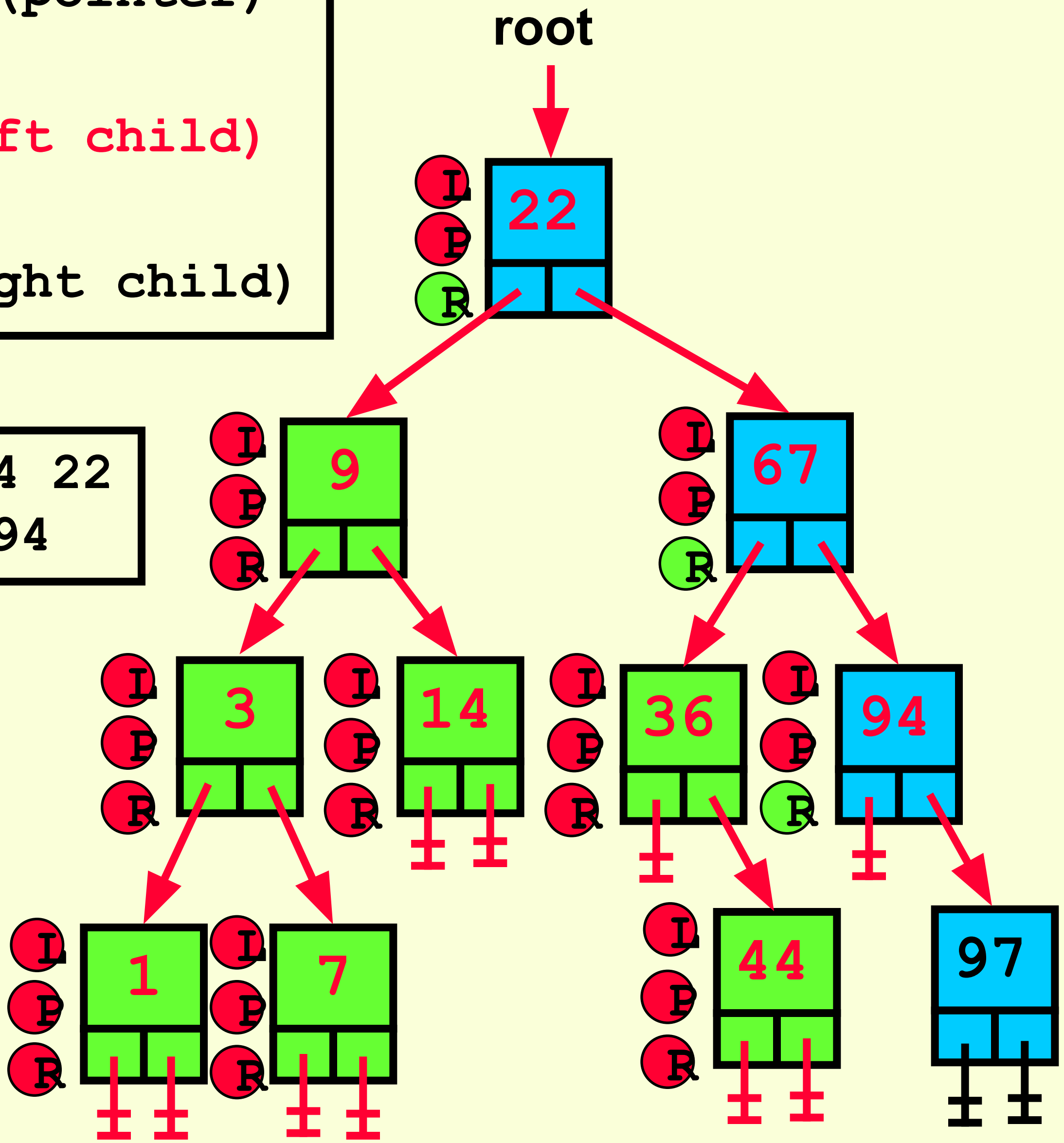
```
Proc InOrderPrint(pointer)
  pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94
```



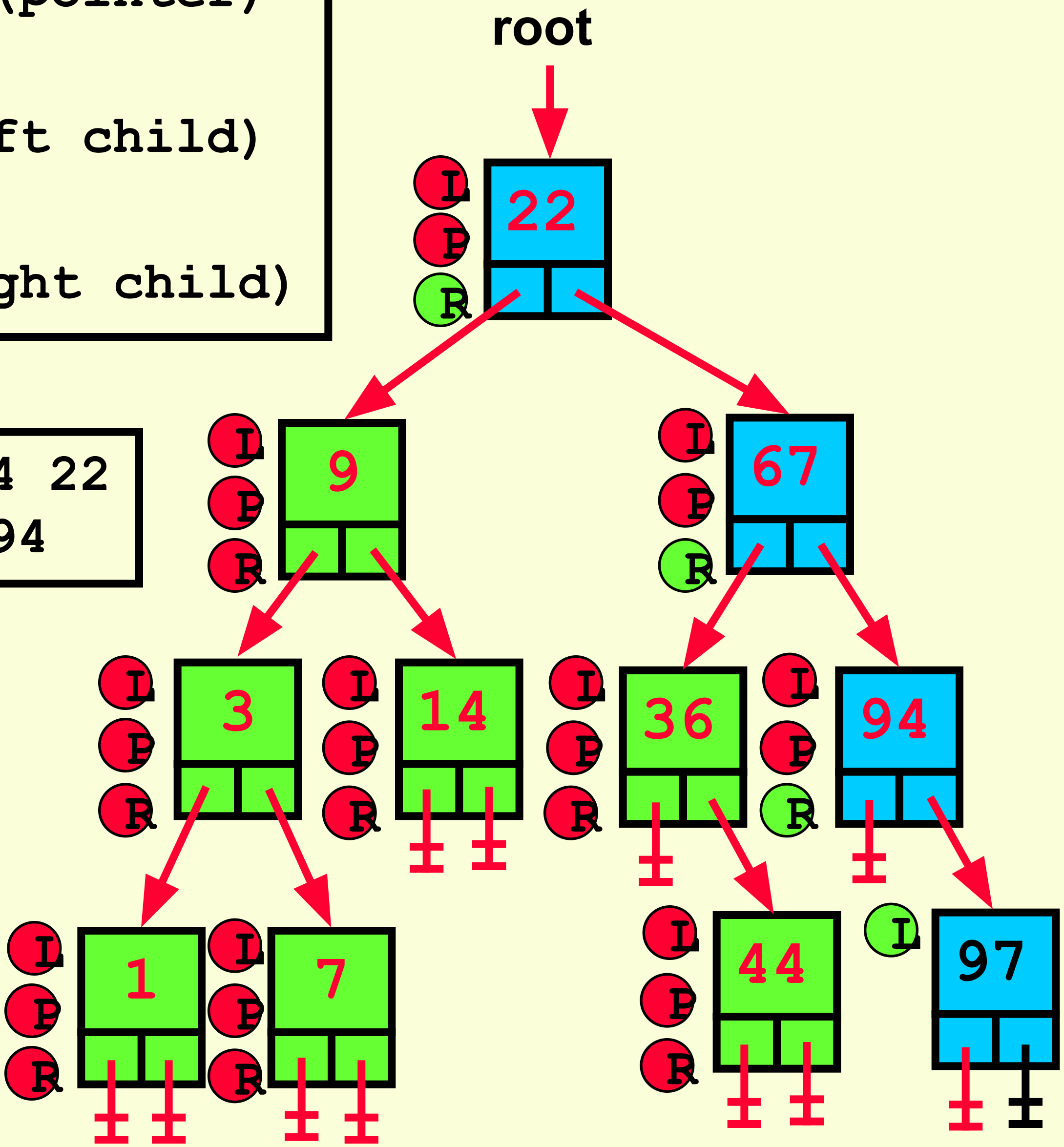
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94
```



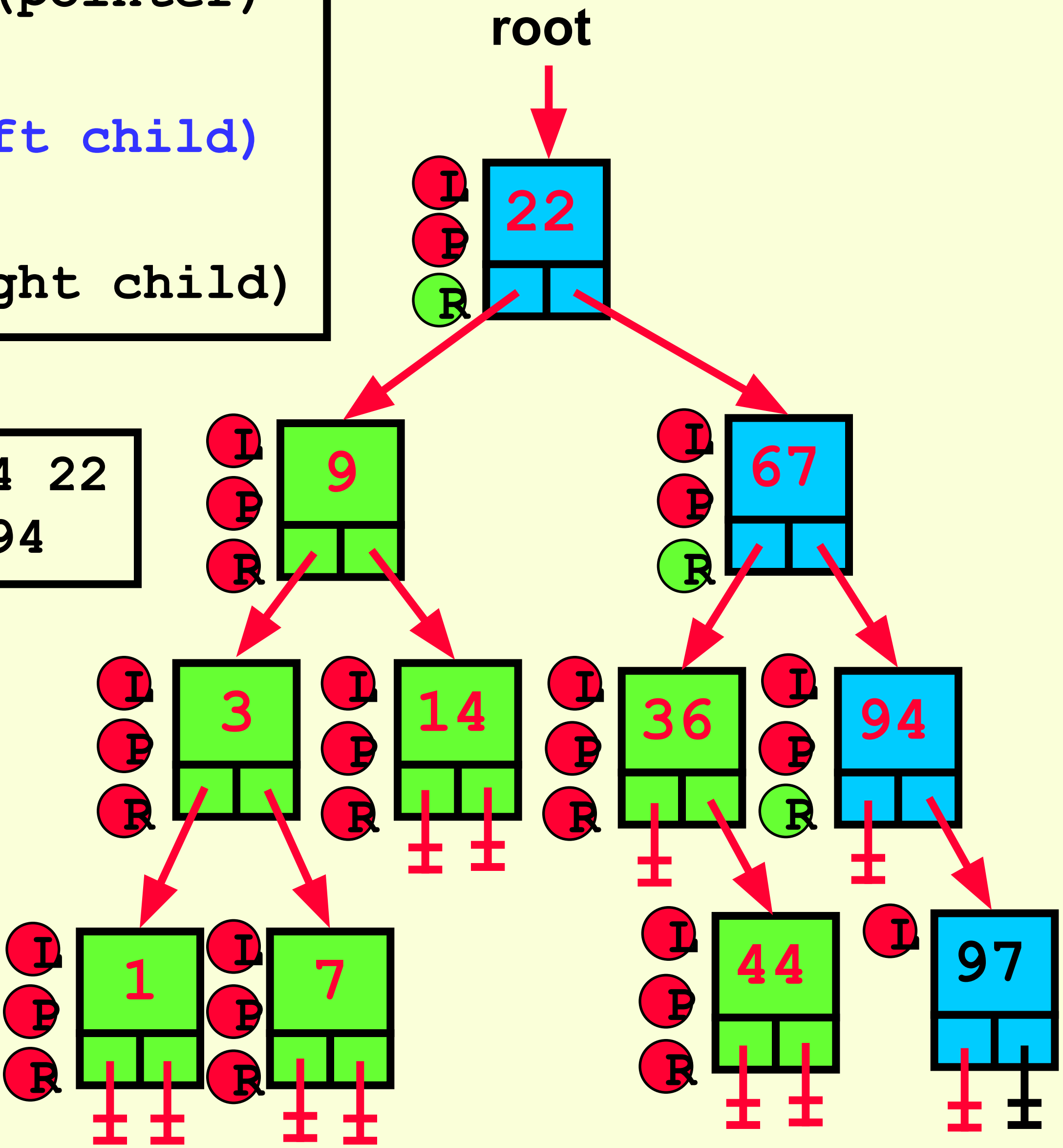
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94
```



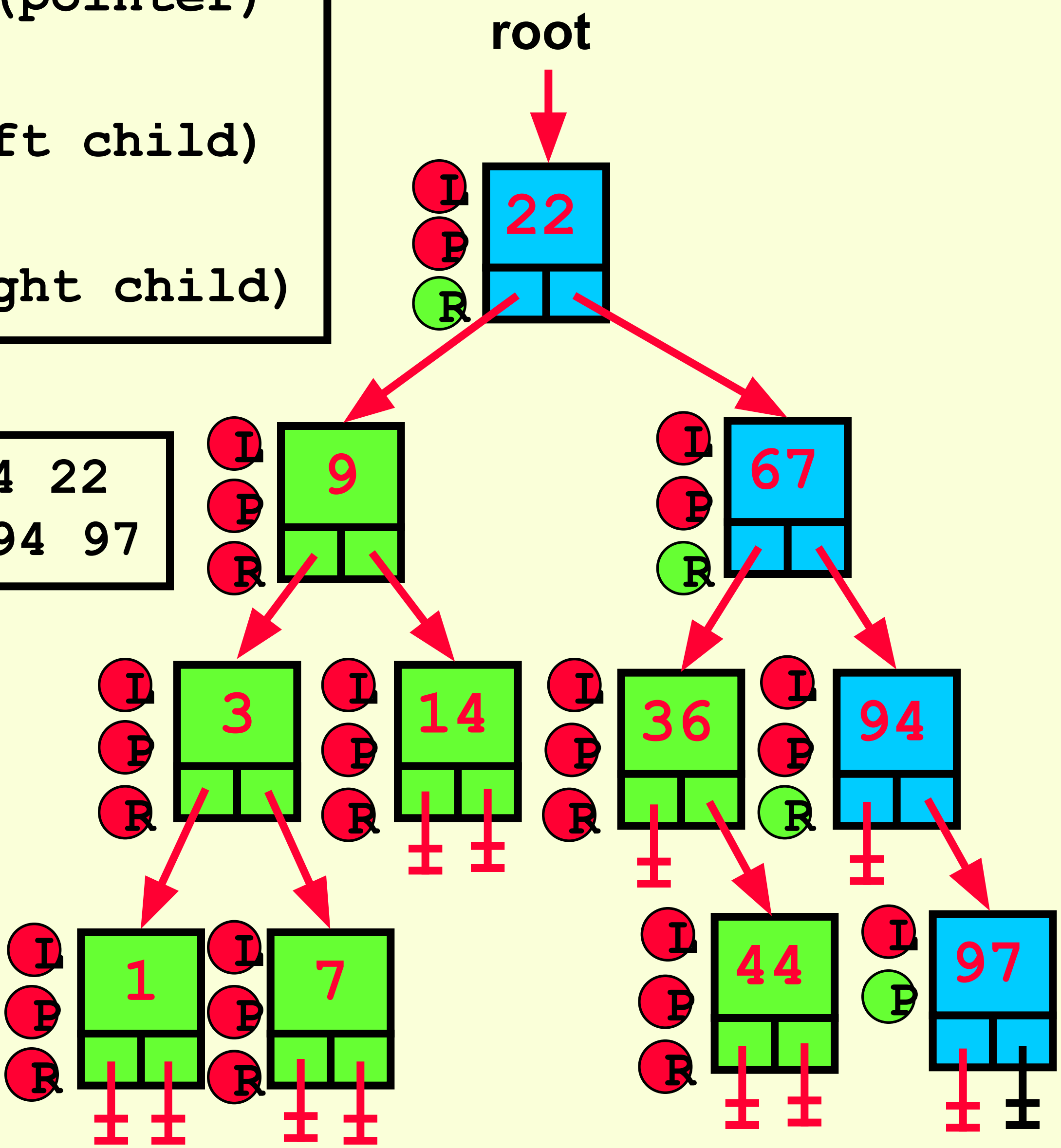
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94
```



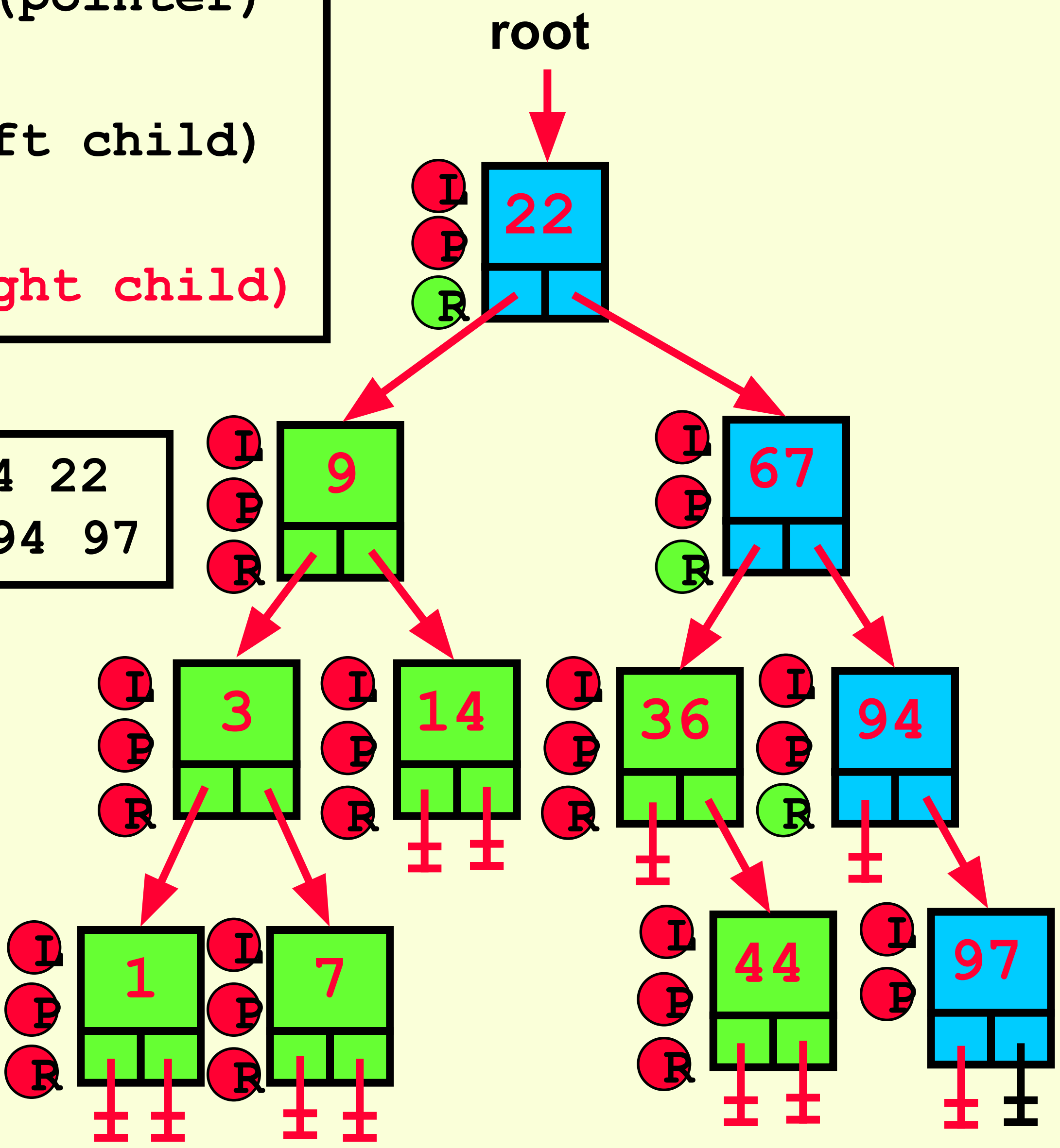
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



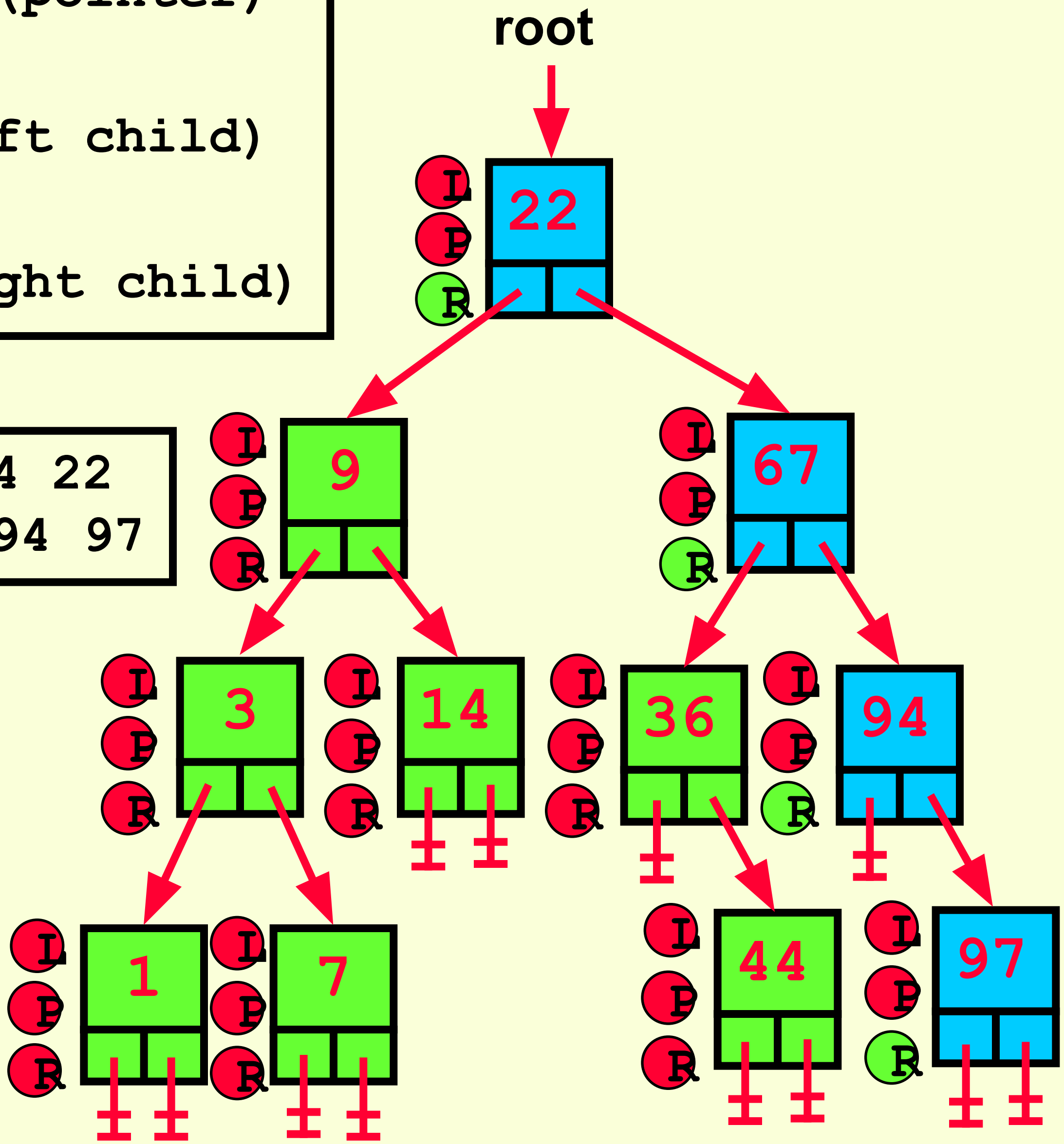
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



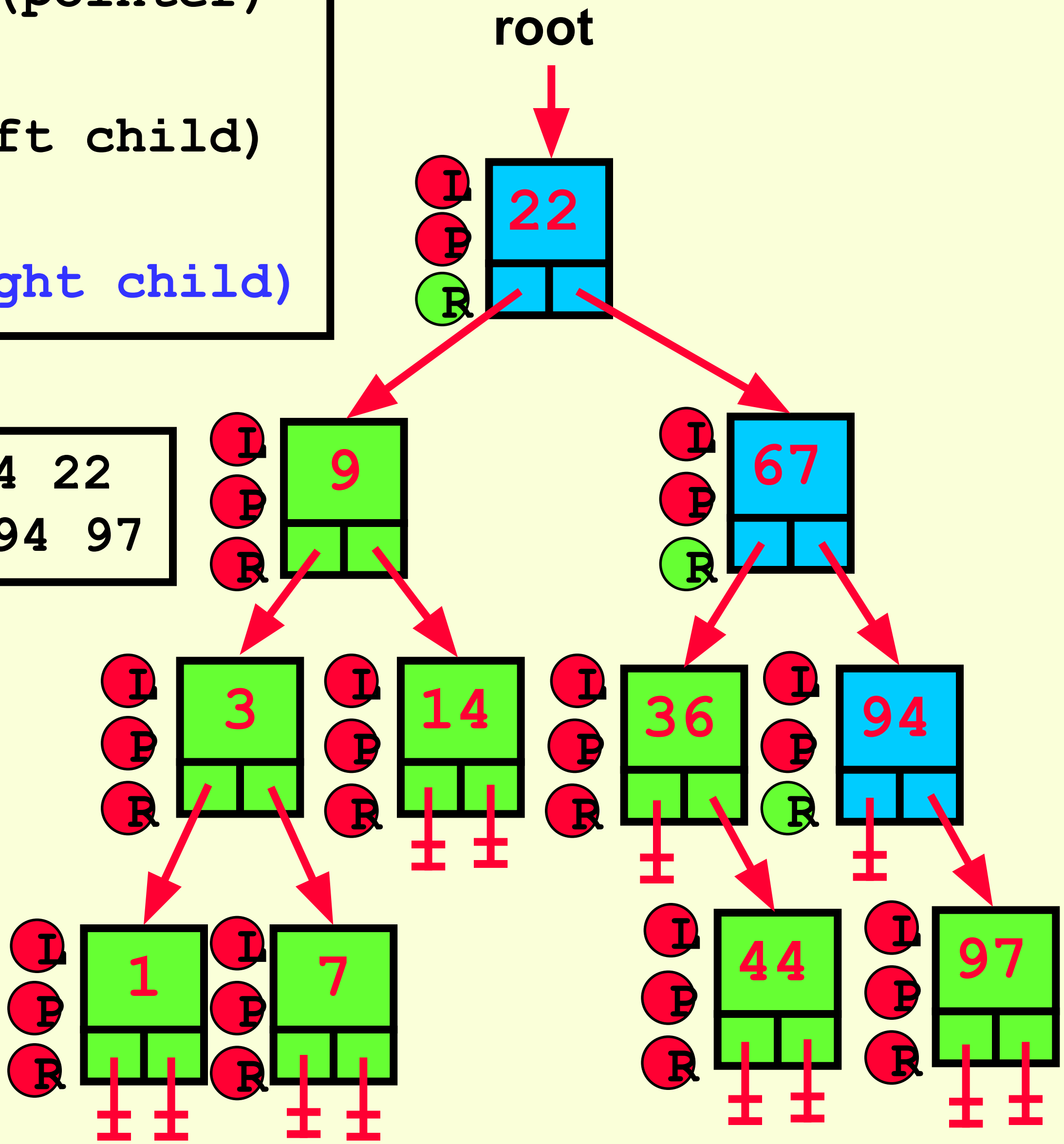
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



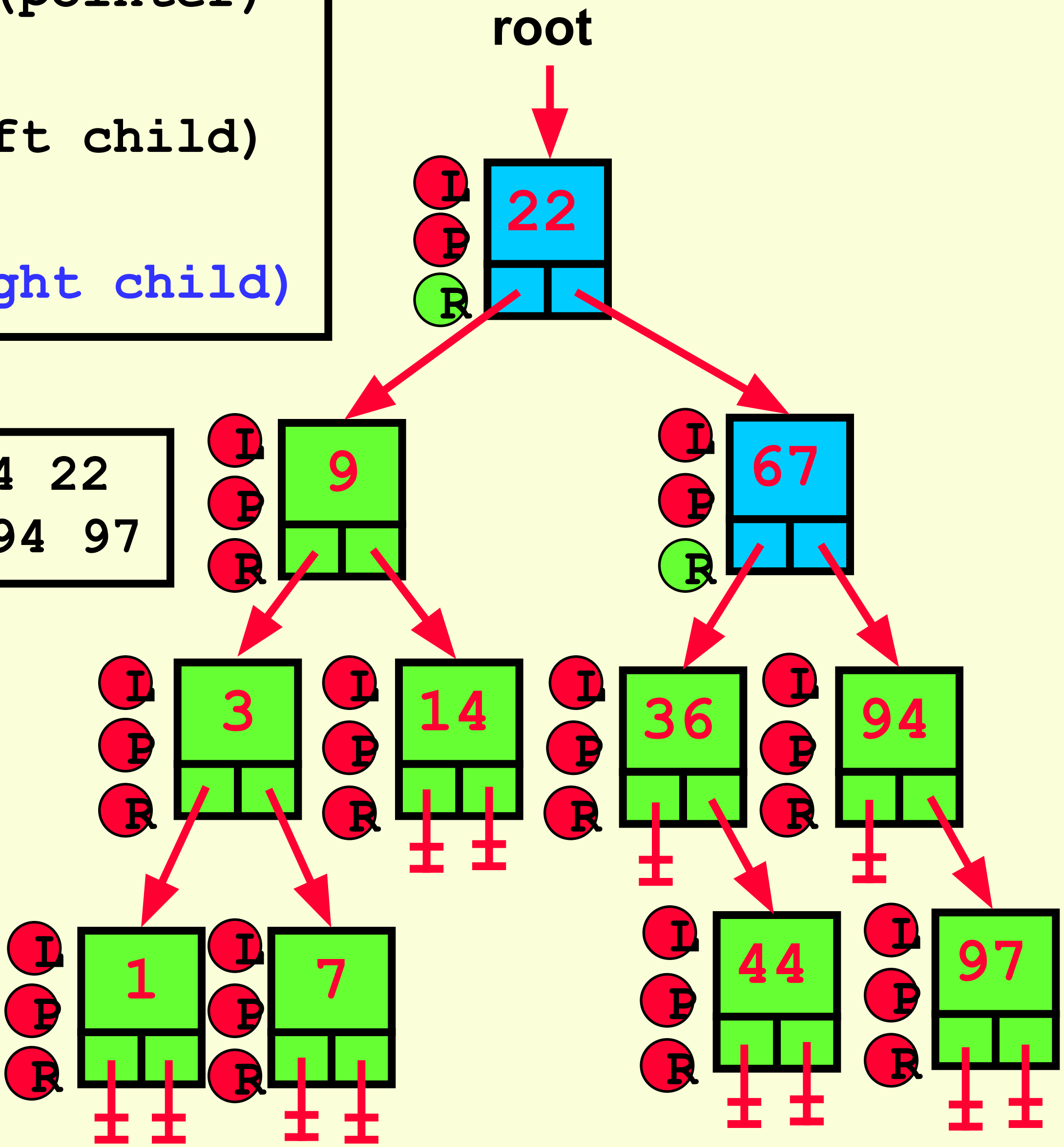
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



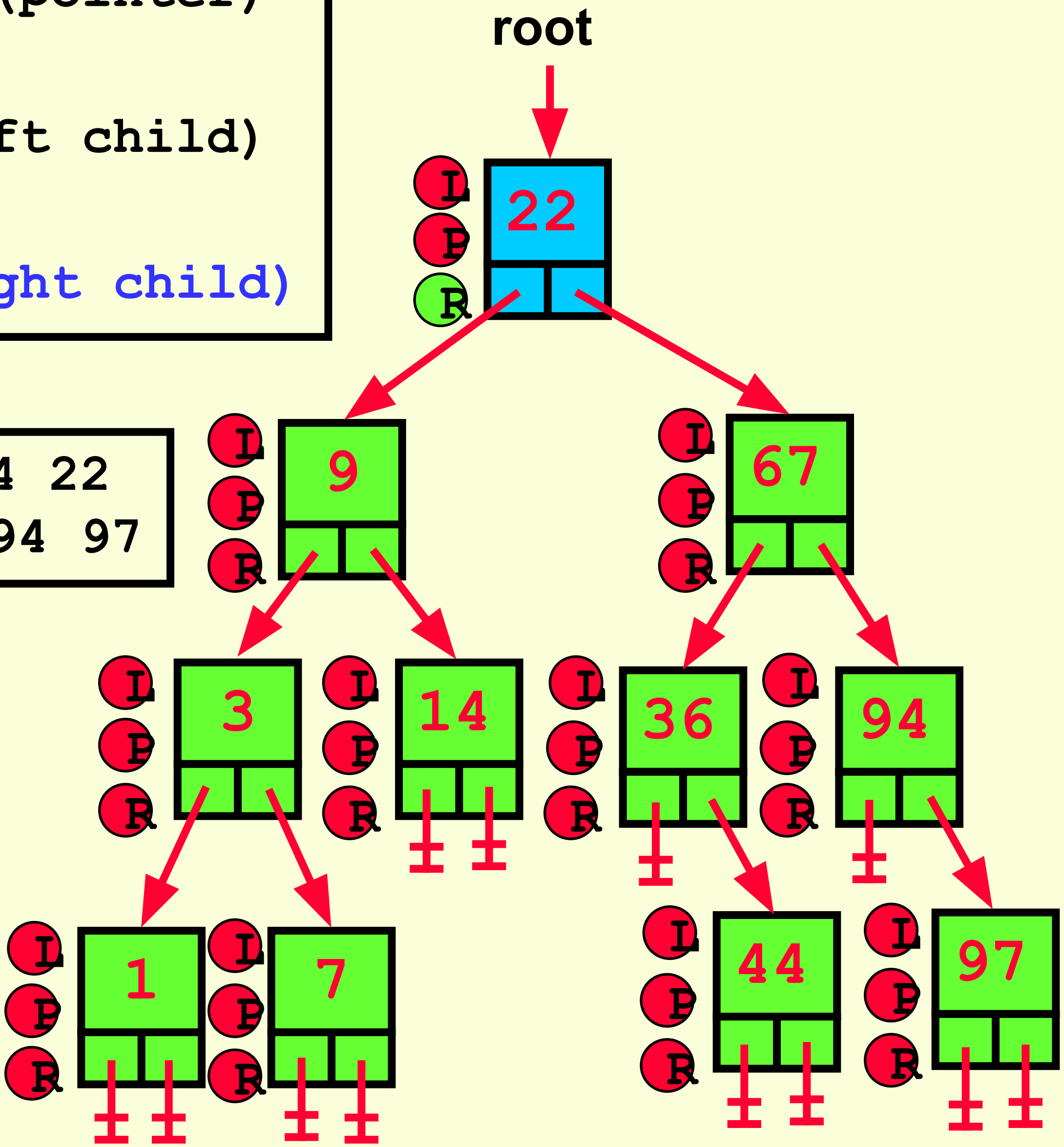
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



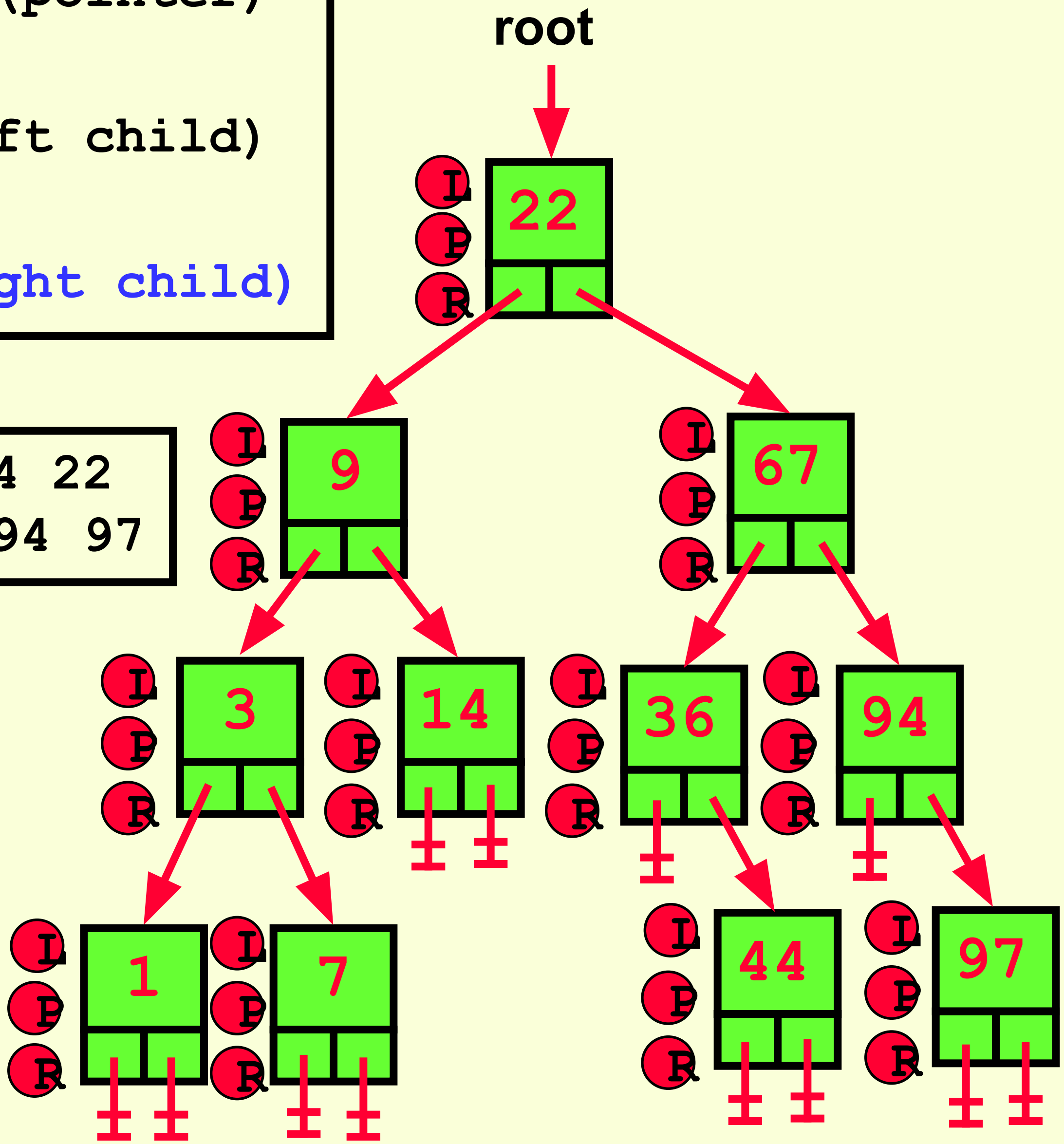
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



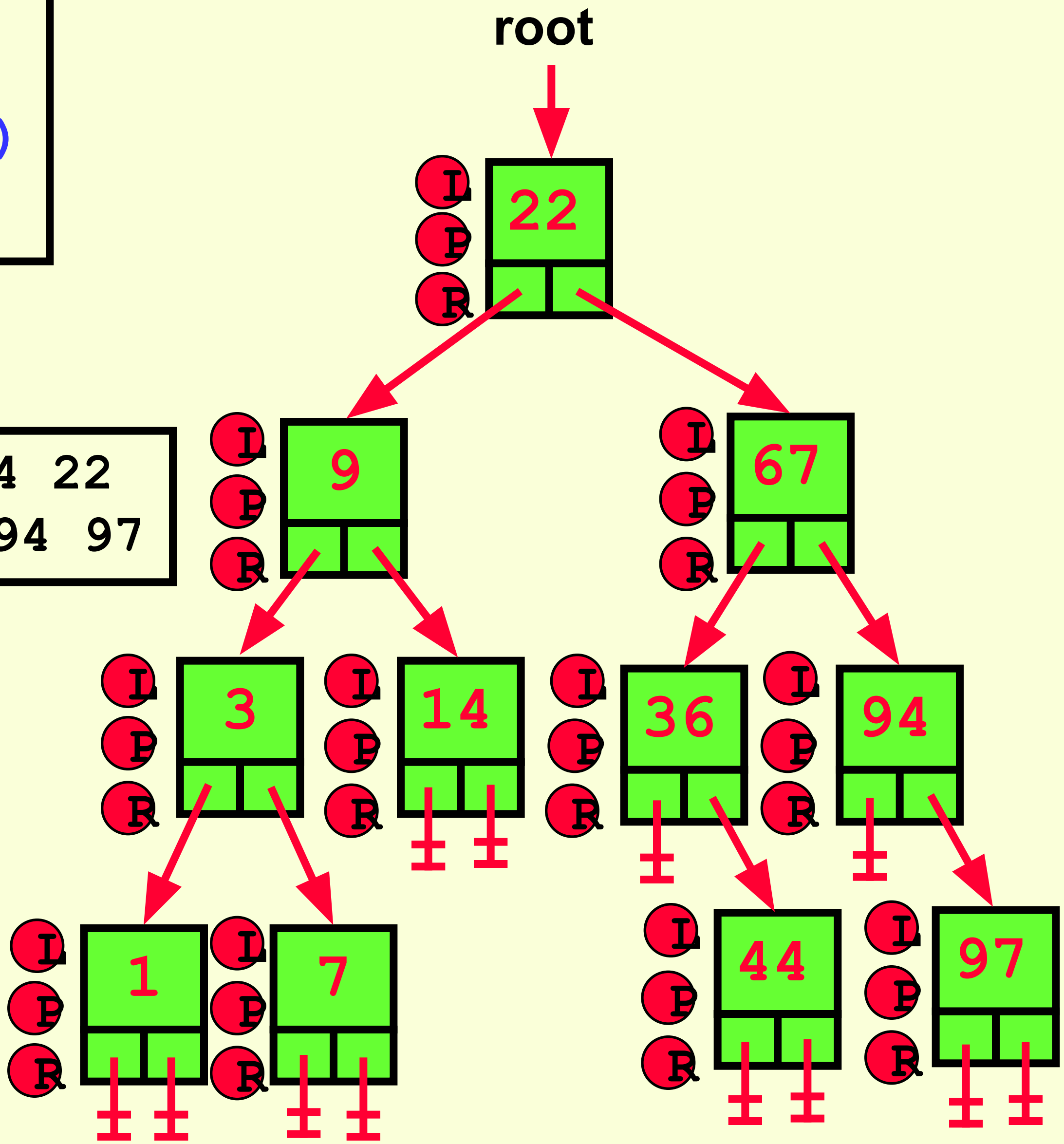
```
Proc InOrderPrint(pointer)
 pointer NOT NIL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



```
Algorithm Example
. . .
InOrderPrint(root)
. . .
```

```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



Summary

- An In-Order traversal visits every node
  - Recurse left first
  - Do something with current
  - Recurse right last

- The “left, current, right” logic is repeated recursively at every node.

- For a BST, an in-order traversal accesses the elements in ascending order.

Questions?

Binary Search Tree Insertion

Tree Node Defined

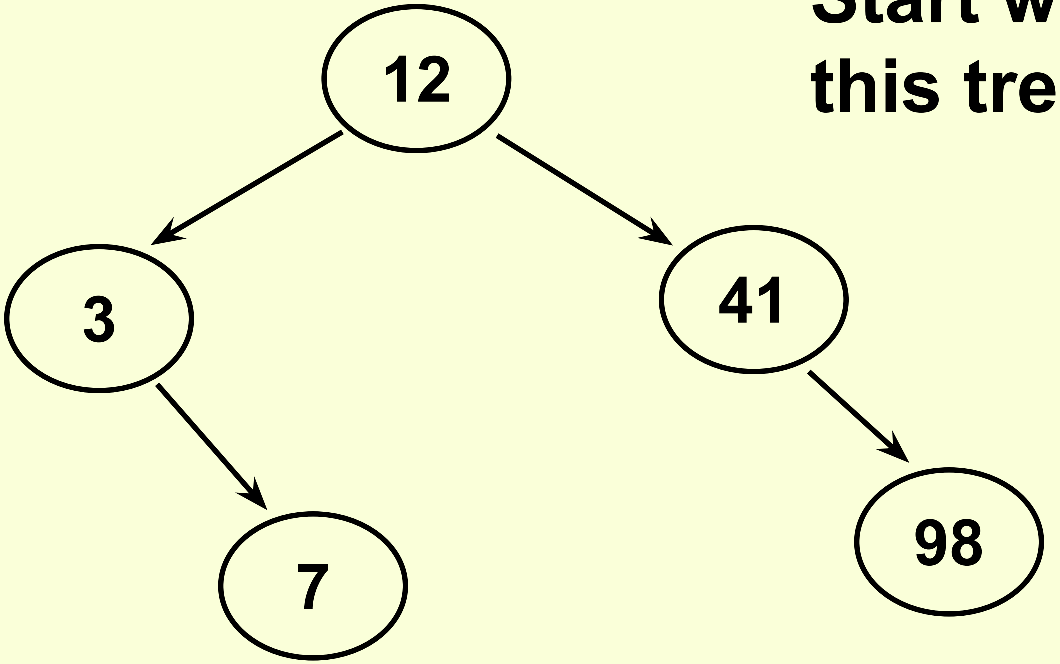
In general:

```
struct node{
Int data;
Struct node *left_child, *right_child;
}
```

Scenario

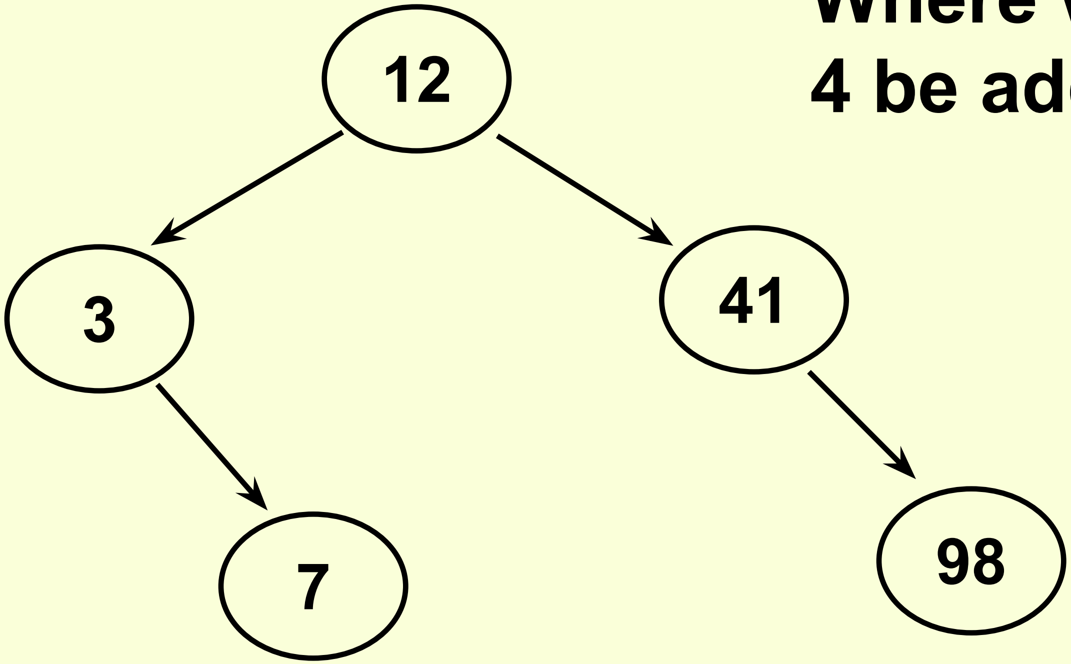
- We have a Binary Search Tree
  - It can be empty
  - Or have some elements in it already
- We want to add an element to it
  - Inserting/adding involves 2 steps:
    - Find the correct location
    - Do the steps to add a new node
- Must maintain “search” structure

Finding the Correct Location



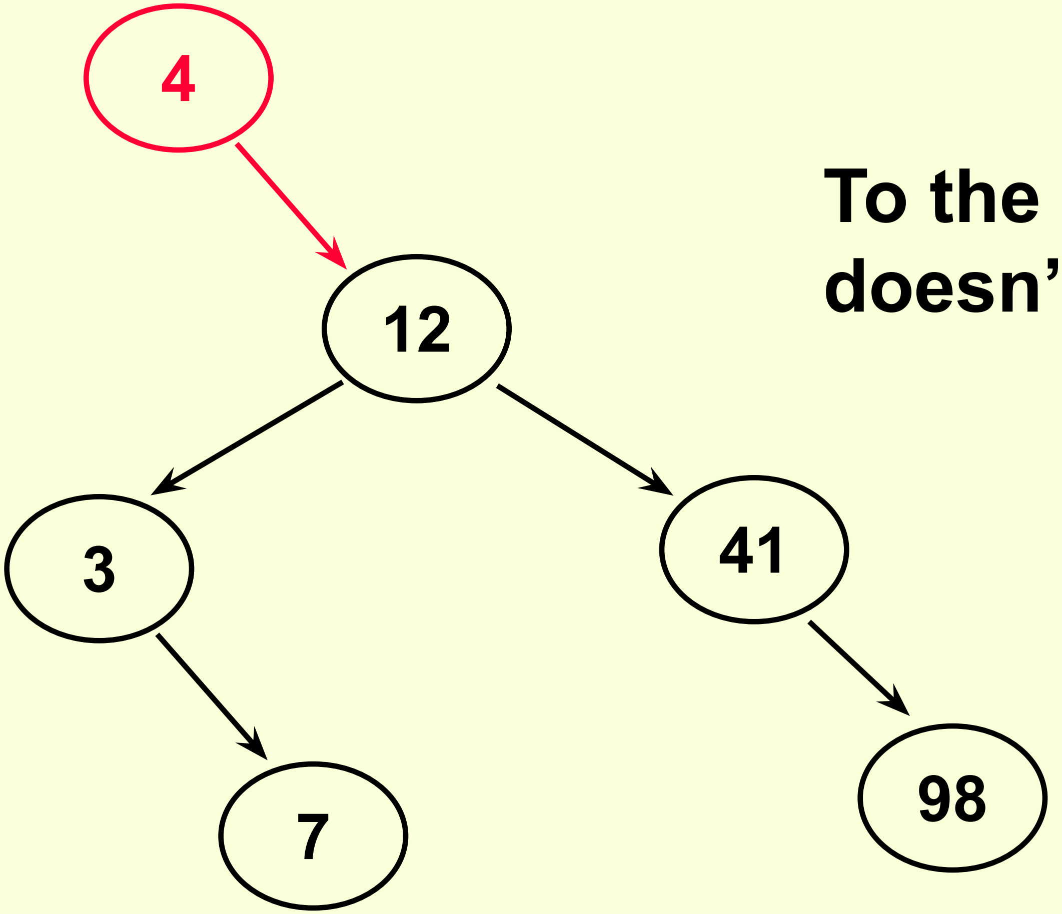
Start with this tree

Finding the Correct Location



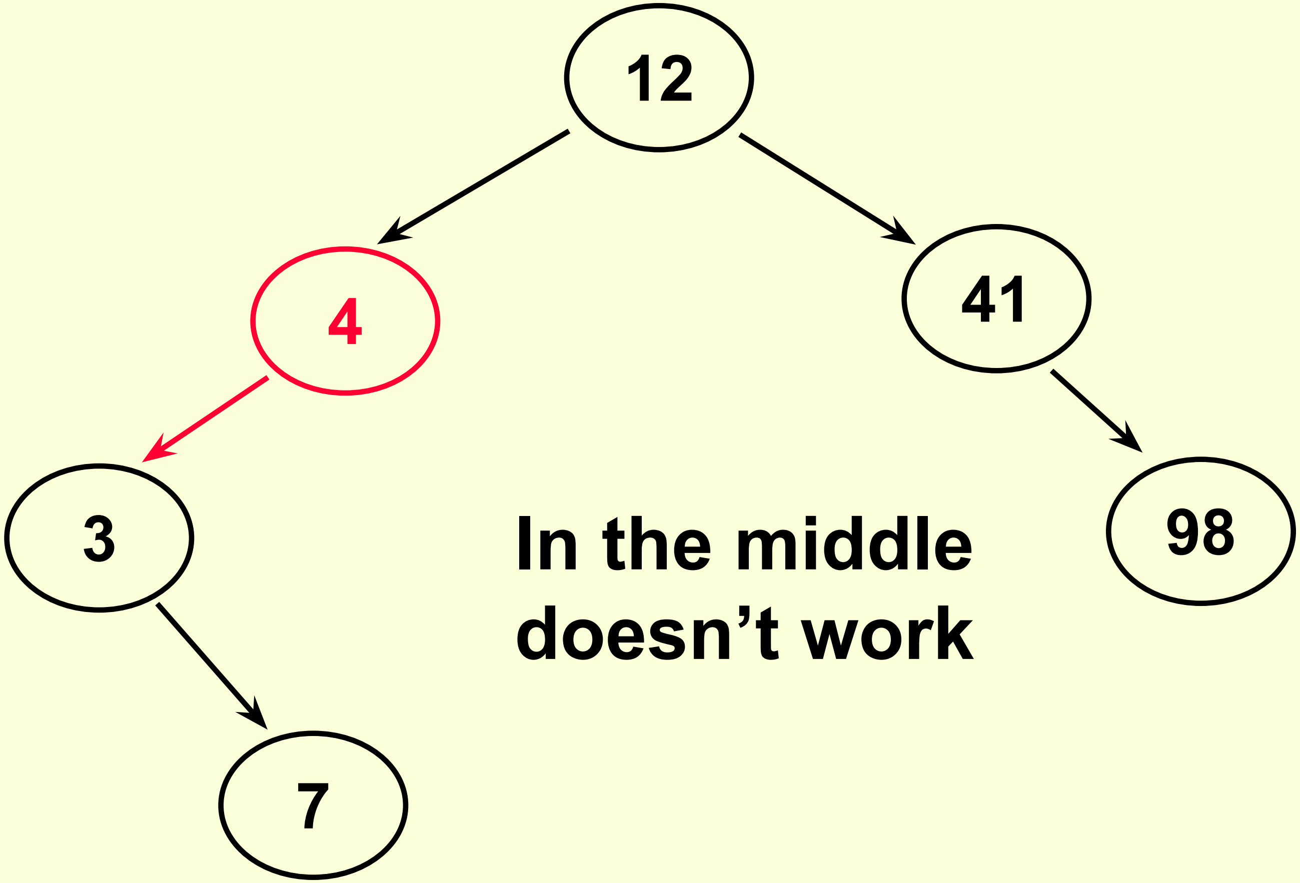
Where would
4 be added?

Finding the Correct Location



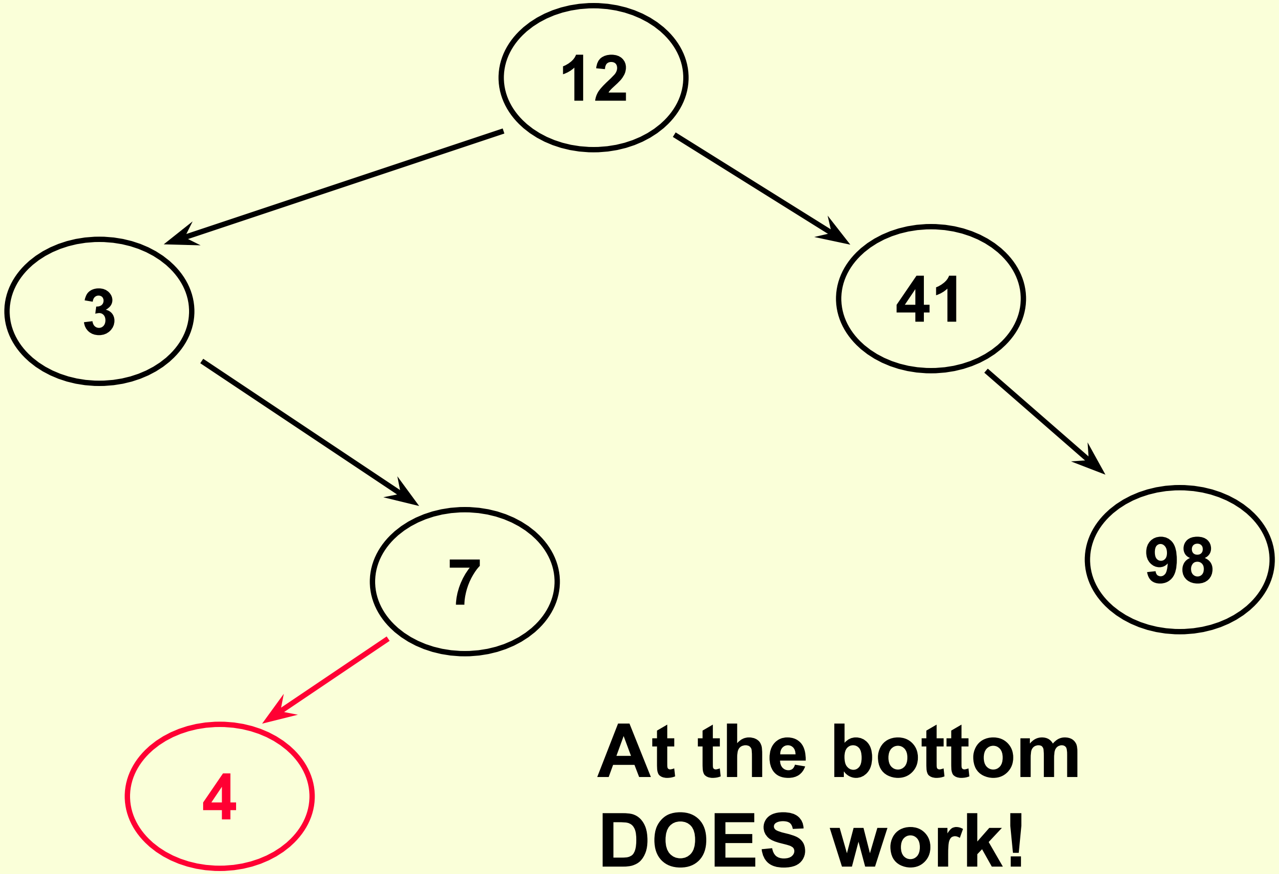
To the top
doesn't work

Finding the Correct Location



In the middle doesn't work

Finding the Correct Location



At the bottom DOES work!

Finding the Correct Location

- Must maintain “search” structure
  - Everything to left is less than current
  - Everything to right is greater than current
- Adding at the “bottom” guarantees we keep search structure.
- We’ll recurse to get to the “bottom” (i.e. when current = nil)

Finding the Correct Location

```
if (current == nil)
  DO "ADD NODE" WORK HERE
Else if (current->data > value_to_add)
  then
  // recurse left
  Insert(current->left, value_to_add)
else
  // recurse right
  Insert(current->right, value_to_add)
```

Adding the Node

- Current is an in/out pointer
  - We need information IN to evaluate current
  - We need to send information OUT because we're changing the tree (adding a node)

- Once we've found the correct location:
  - Create a new node
  - Fill in the data field
    (with the new value to add)
  - Make the left and right pointers point to nil
    (to cleanly terminate the tree)

Adding the Node

```
current = new(Node)
current->data = value_to_add
current->left = nil
current->right = nil
```

The Entire Module

```
Insert(cur iot in/out Ptr to a Node,
         data_in iot in integer)
  if(cur == NIL) then
    cur = new(Node)
    cur->data =data_in
    cur->left = NIL
    cur->right = NIL
  else if(cur->data > data_in)
    Insert(cur->left, data_in)
  else
    Insert(cur->right, data_in)
```

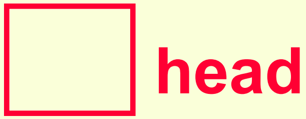
Tracing Example

The following example shows a trace of the BST insert.

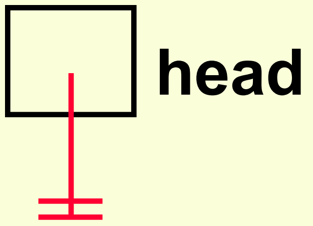
- Begin with an empty BST (a pointer)
- Add elements 42, 23, 35, 47 in the correct positions.

```
Head iot Ptr to a Node
head = NIL
Insert(head, 42)
```

```
Head iot Ptr toa Node
head = NIL
Insert(head, 42)
```

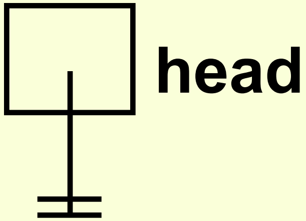


```
Head iot Ptr toa Node
head = NIL
Insert(head, 42)
```

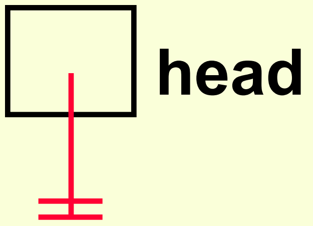


head

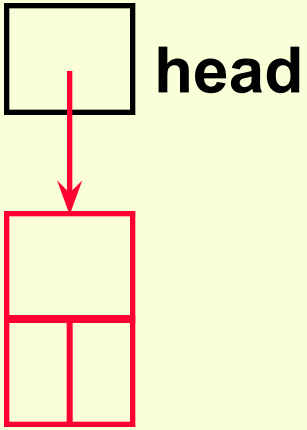
```
Head iot Ptr toa Node
head = NIL
Insert(head, 42)
```



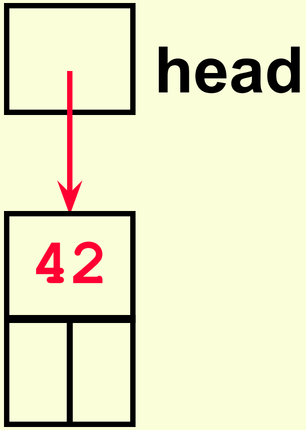
```
procedure Insert(
 cur iot in/out Ptr to a Node,
 data_in iot in num)
 if(cur == NIL) then
  cur = new(Node)
  cur->data = data_in
  cur->left = NIL
  cur->right = NIL
 else if(cur->data > data_in)
  Insert(cur->left, data_in)
 else
  Insert(cur->right, data_in)
```



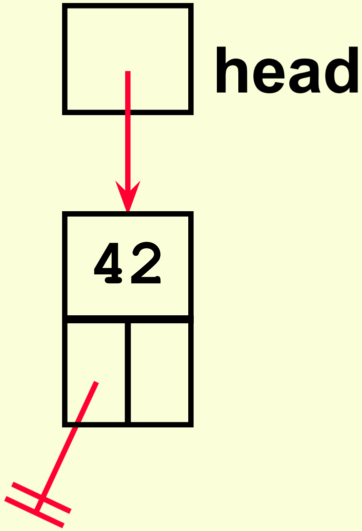
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur = new(Node)
  cur->data = data_in
  cur^.left = NIL
  cur->right = NIL
 elseif(cur->data > data_in)
  Insert(cur->left, data_in)
 else
  Insert(cur->right, data_in)
 endif
endprocedure // Insert
```



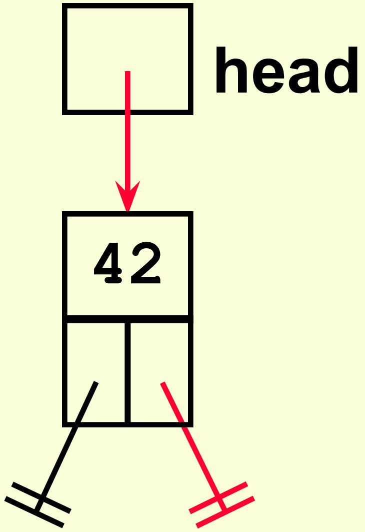
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur->data = data_in
  cur->left = NIL
  cur->right = NIL
 elseif(cur->data > data_in)
  Insert(cur->left, data_in)
 else
  Insert(cur->right, data_in)
 endif
endprocedure // Insert
```



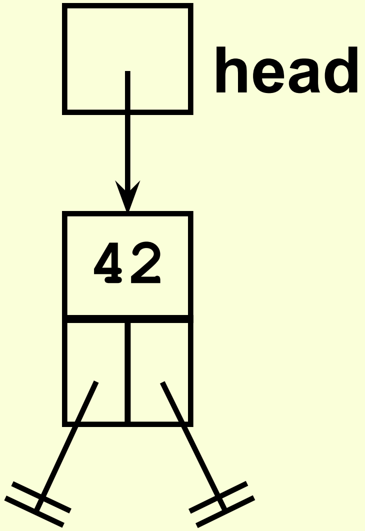
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



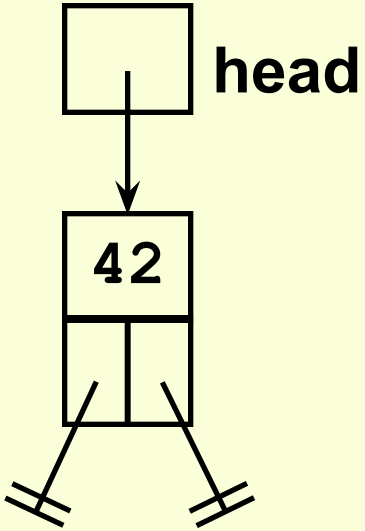
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



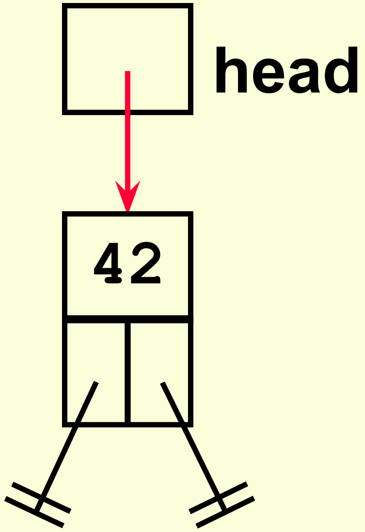
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



```
.
.
Insert(head, 23)
Insert(head, 35)
Insert(head, 47)
.
.
```

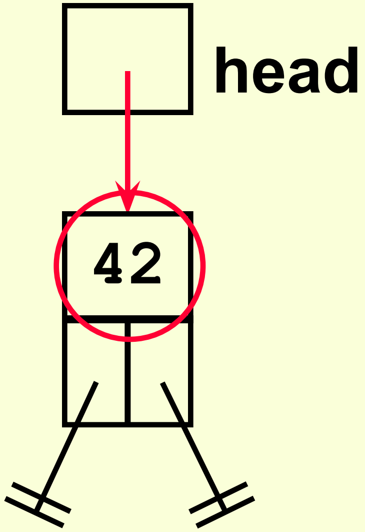


```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



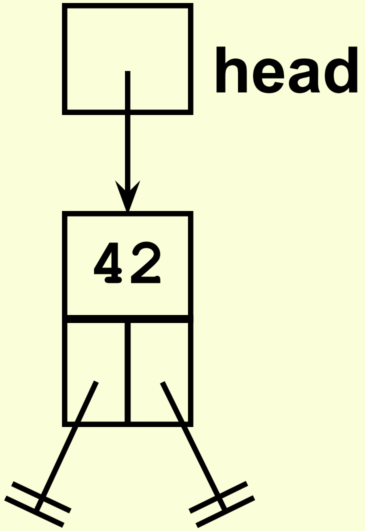
data_in = 23

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



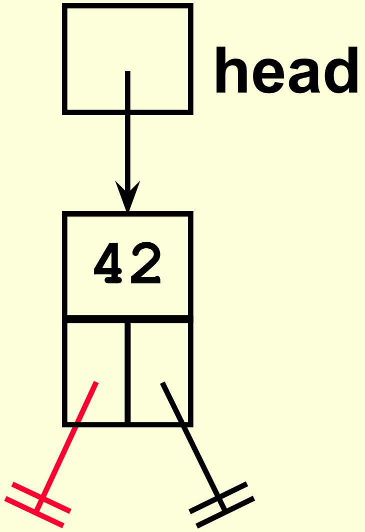
data_in = 23

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



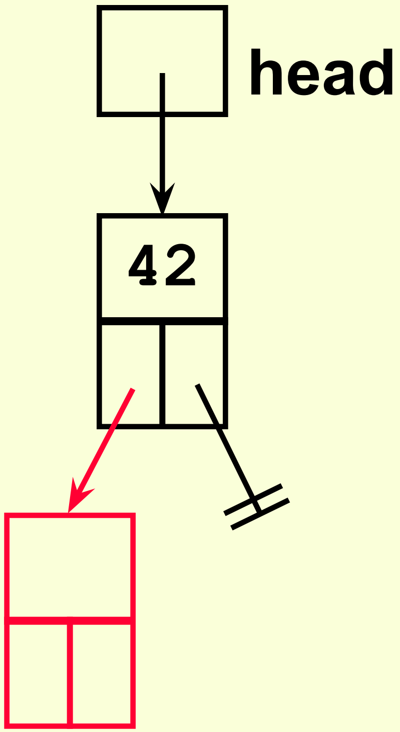
data_in = 23

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



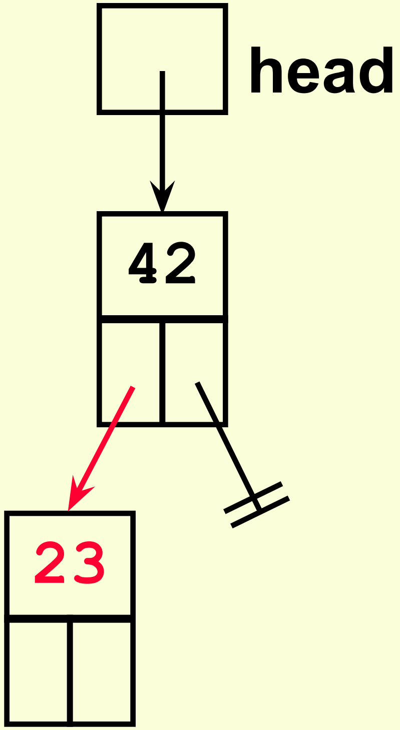
data_in = 23

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



```
data_in = 23
```

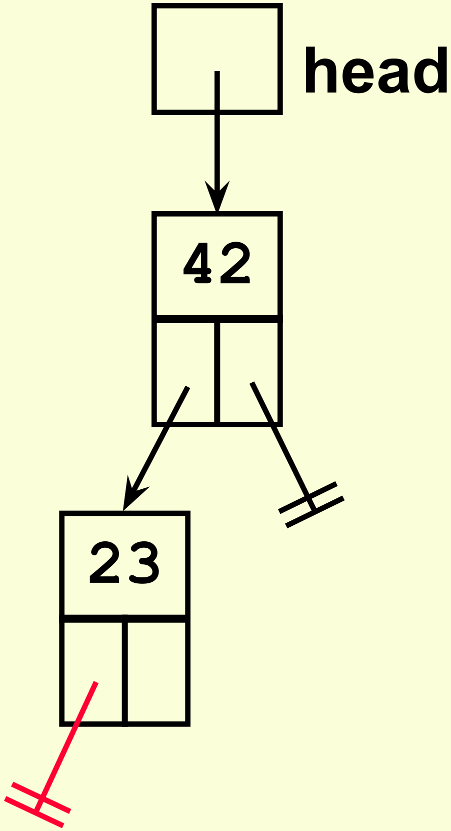
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



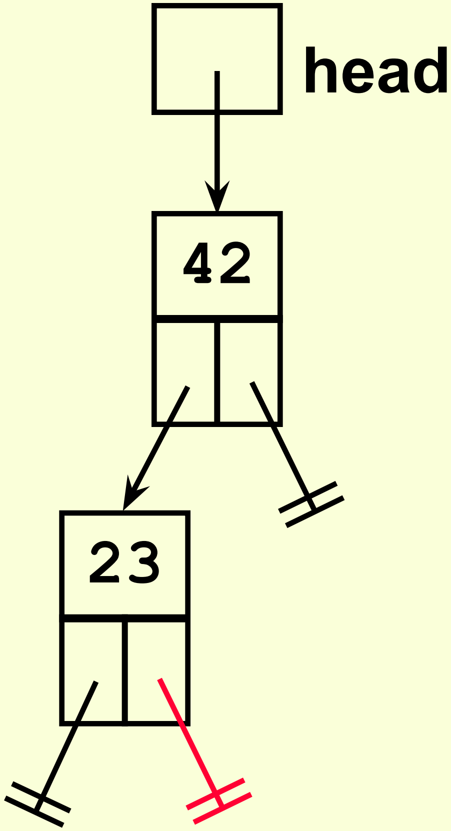
data_in = 23

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 23



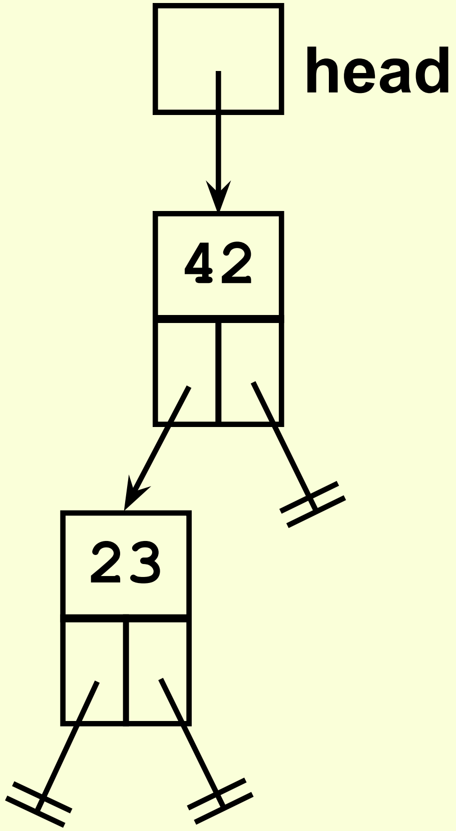
```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



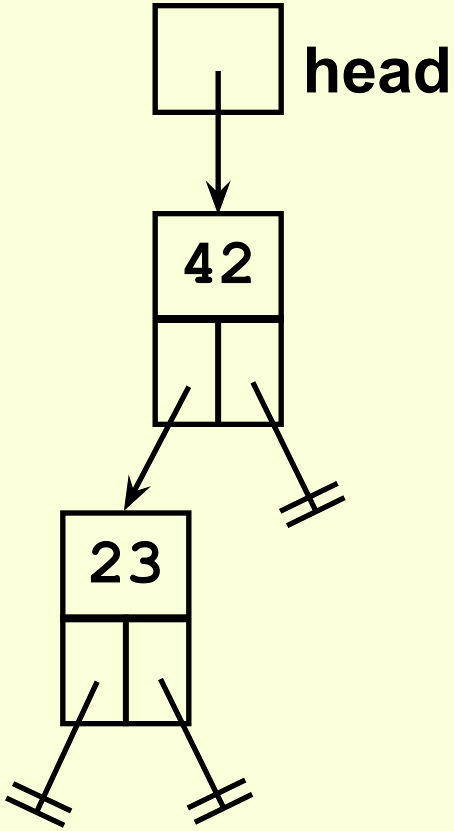
data_in = 23

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 23

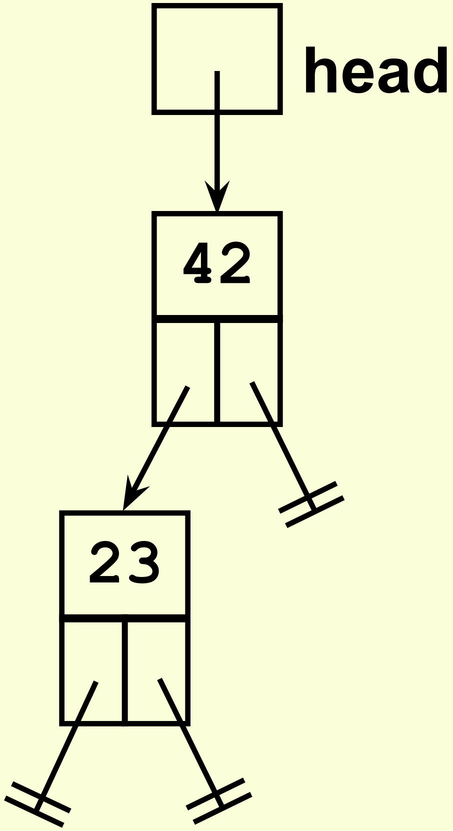


```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

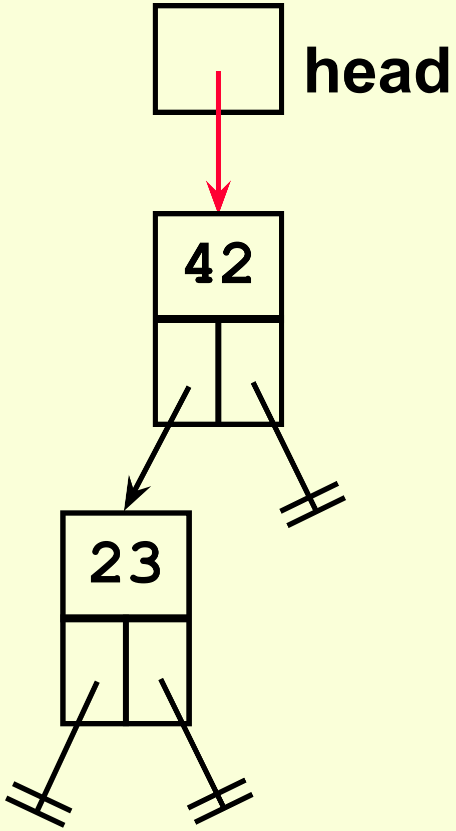


data_in = 23

```
.
.
Insert(head, 23)
Insert(head, 35)
Insert(head, 47)
.
.
```

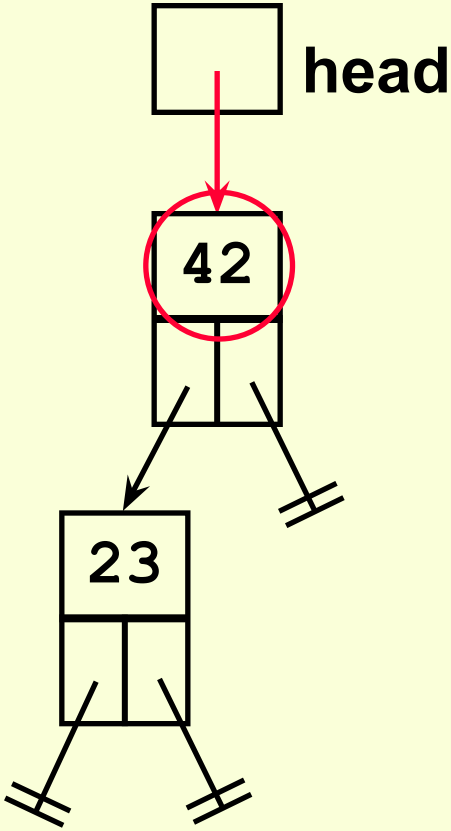


```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



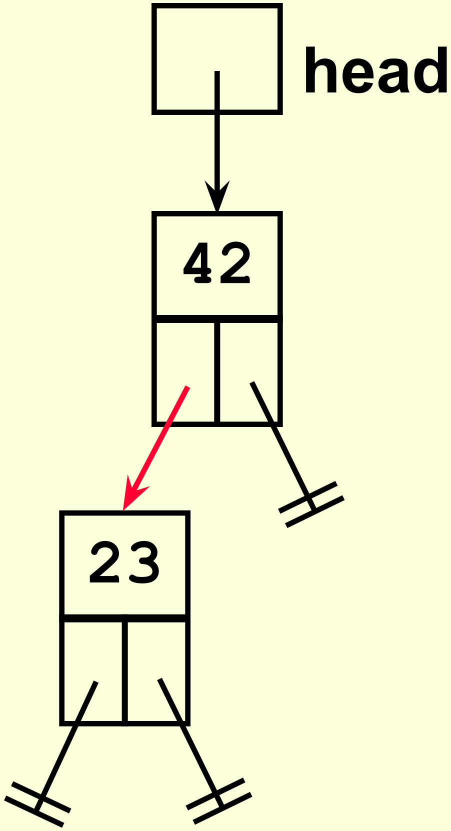
```
data_in = 35
```

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



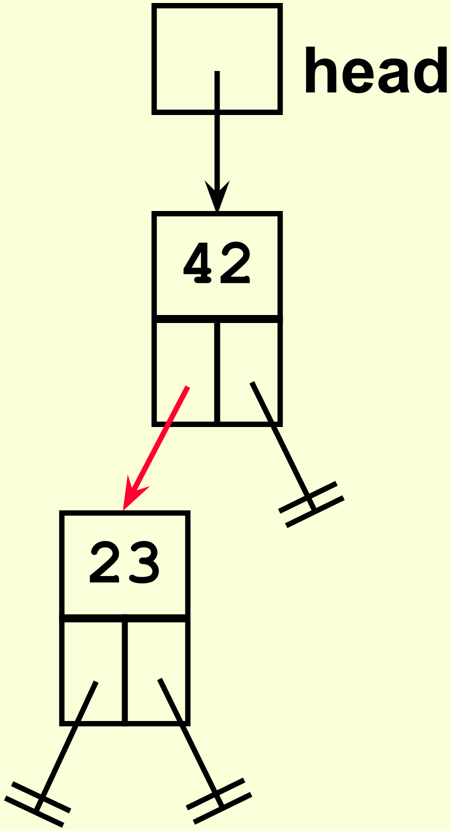
data_in = 35

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



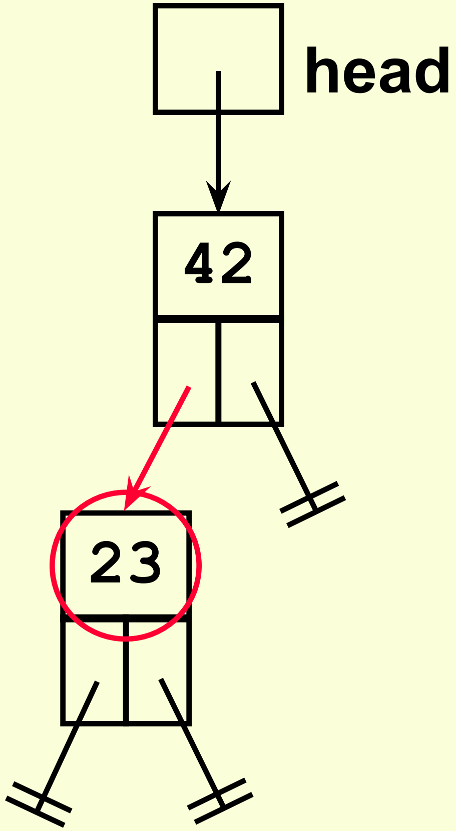
data_in = 35

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



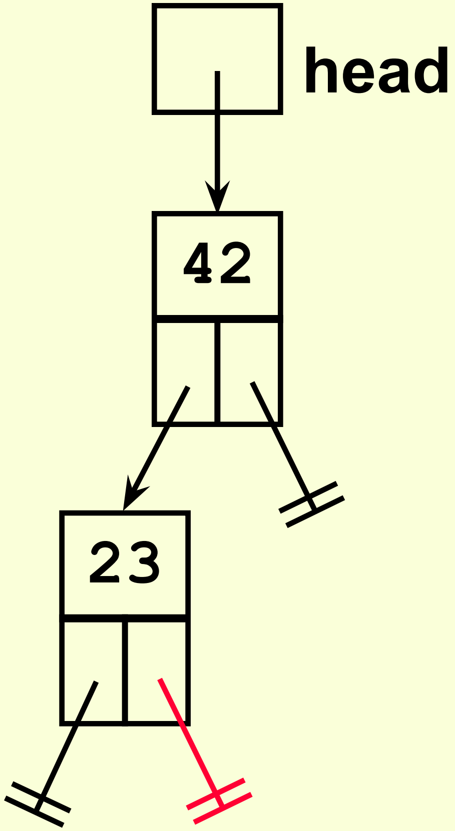
data_in = 35

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



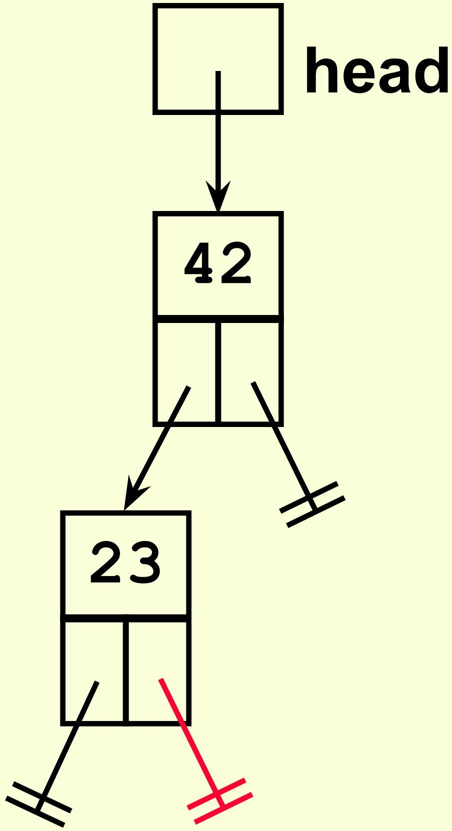
data_in = 35

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



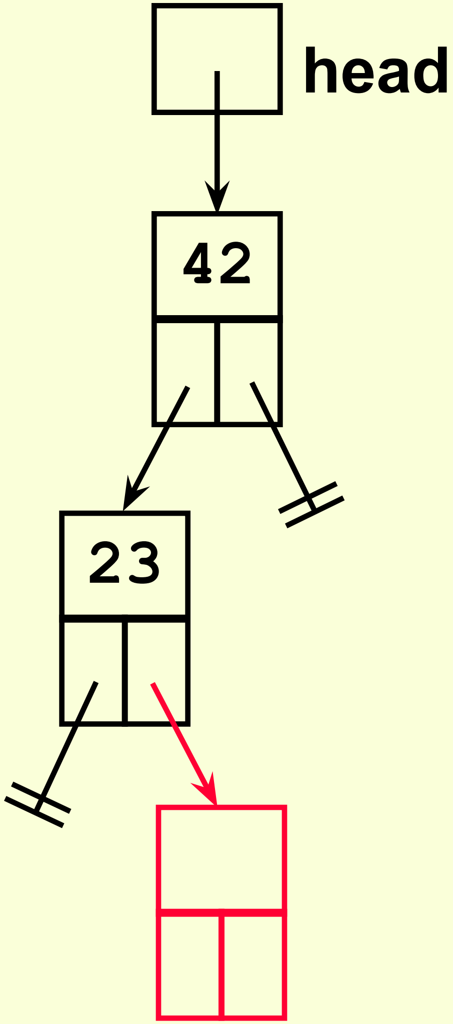
data_in = 35

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



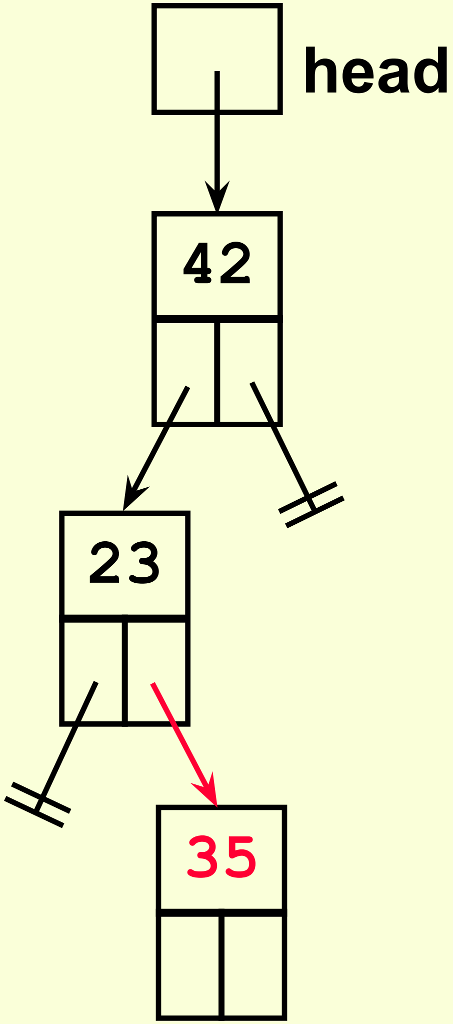
data_in = 35

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



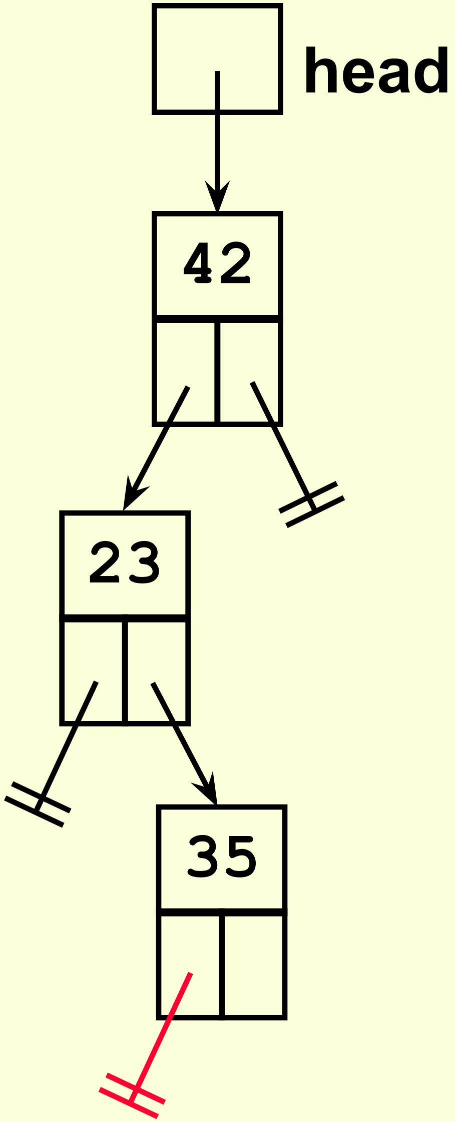
data_in = 35

```
procedure Insert(
  cur iot in/out Ptr toa Node,
  data_in iot in num)
  if(cur = NIL) then
    cur <- new(Node)
    cur^.data <- data_in
    cur^.left <- NIL
    cur^.right <- NIL
  elseif(cur^.data > data_in)
    Insert(cur^.left, data_in)
  else
    Insert(cur^.right, data_in)
  endif
endprocedure // Insert
```



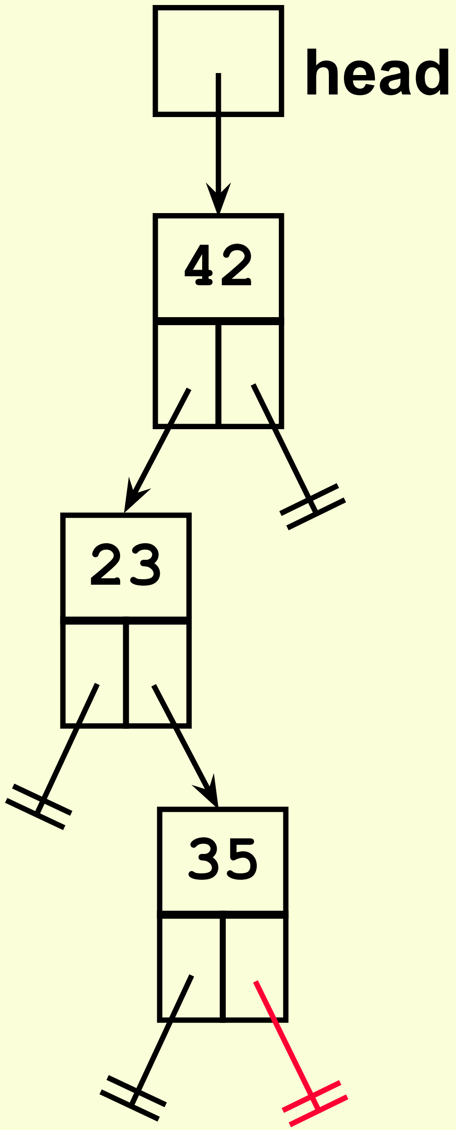
data_in = 35

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



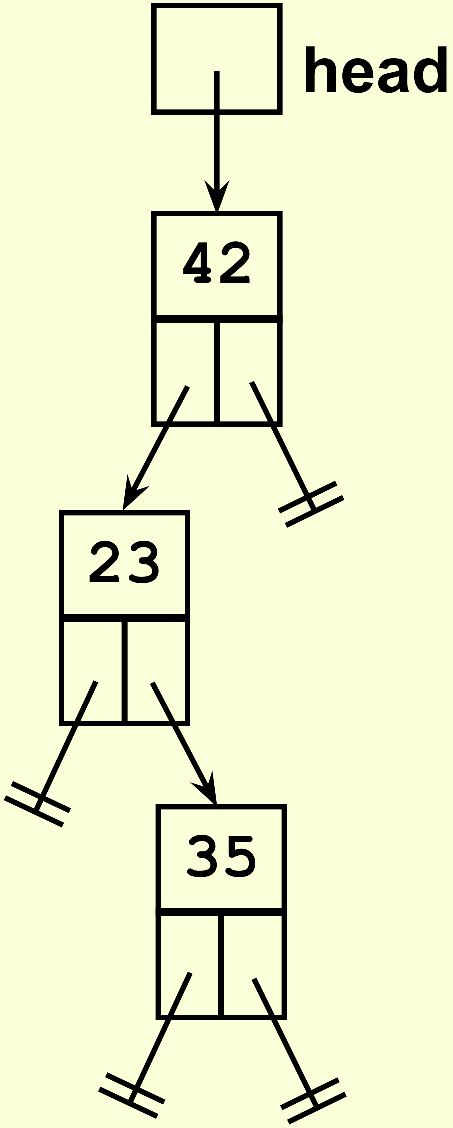
data_in = 35

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



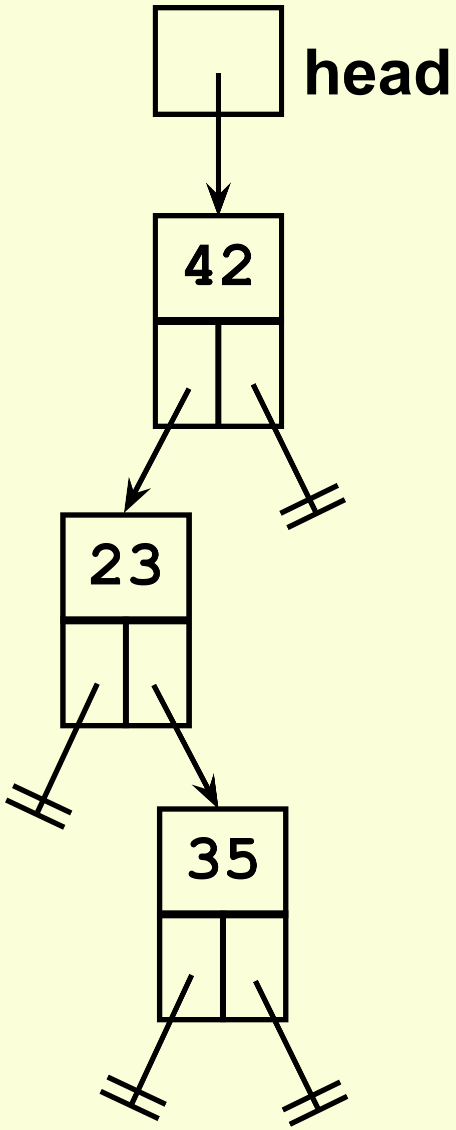
```
data_in = 35
```

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



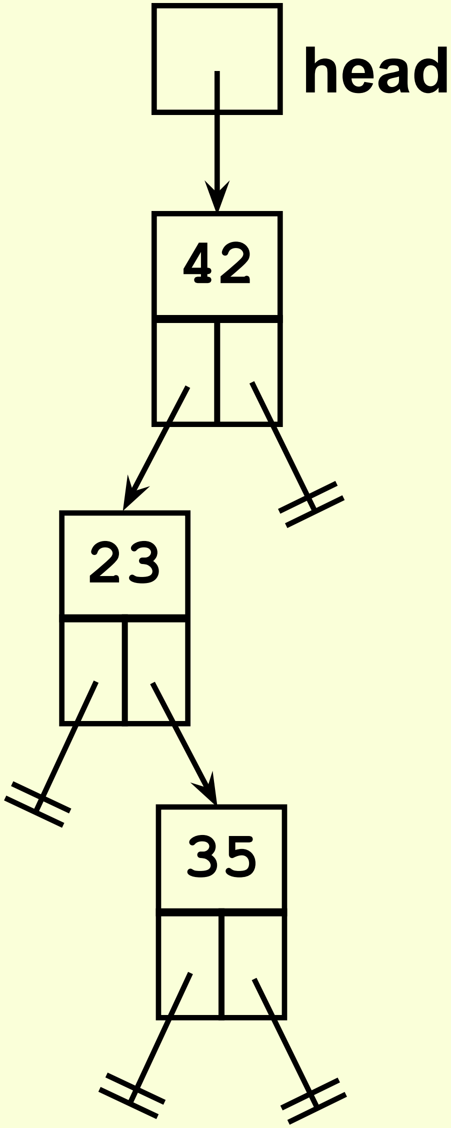
data_in = 35

```
procedure Insert(
  cur iot in/out Ptr toa Node,
  data_in iot in num)
  if(cur = NIL) then
    cur <- new(Node)
    cur^.data <- data_in
    cur^.left <- NIL
    cur^.right <- NIL
  elseif(cur^.data > data_in)
    Insert(cur^.left, data_in)
  else
    Insert(cur^.right, data_in)
  endif
endprocedure // Insert
```



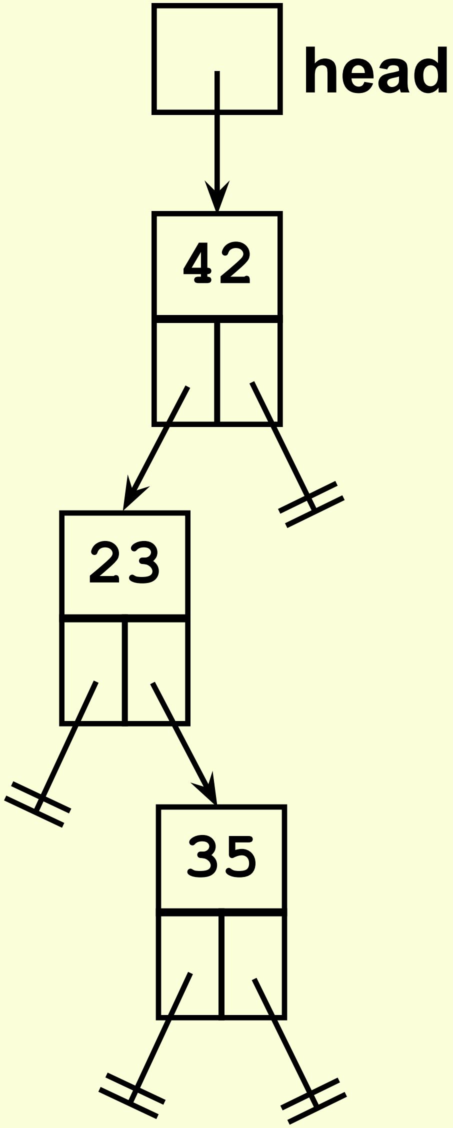
```
data_in = 35
```

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



```
data_in = 35
```

```
.
.
Insert(head, 23)
Insert(head, 35)
Insert(head, 47)
.
.
```



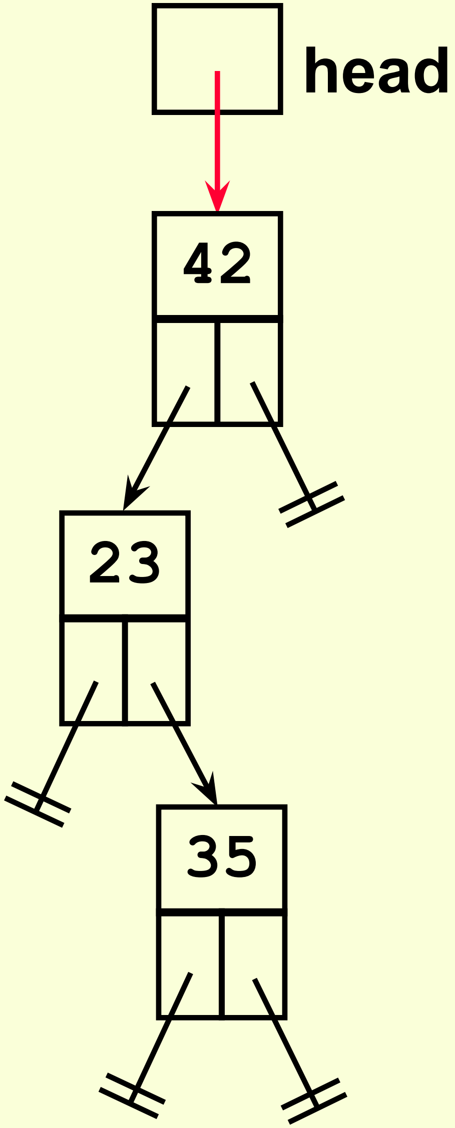
Continue?

yes...

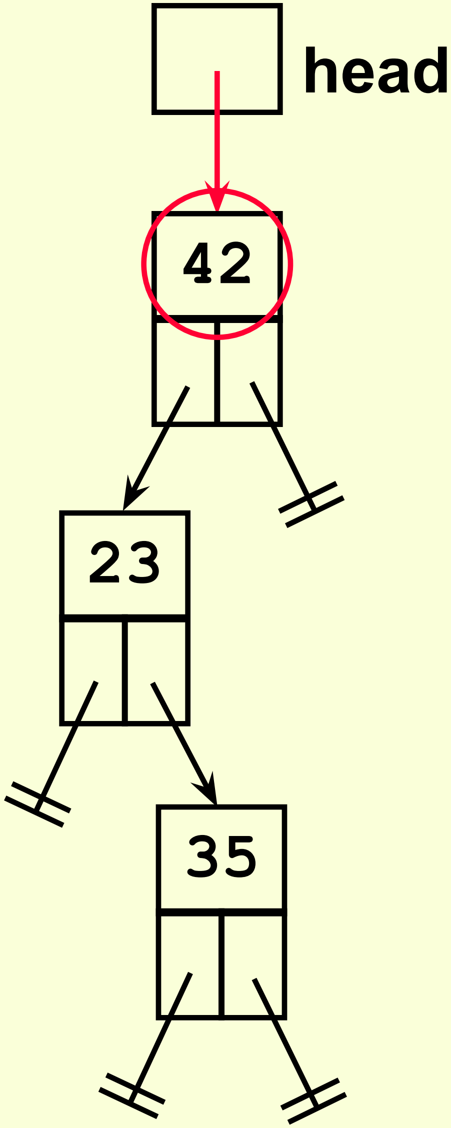
I’ve had enough!

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```

data_in = 47

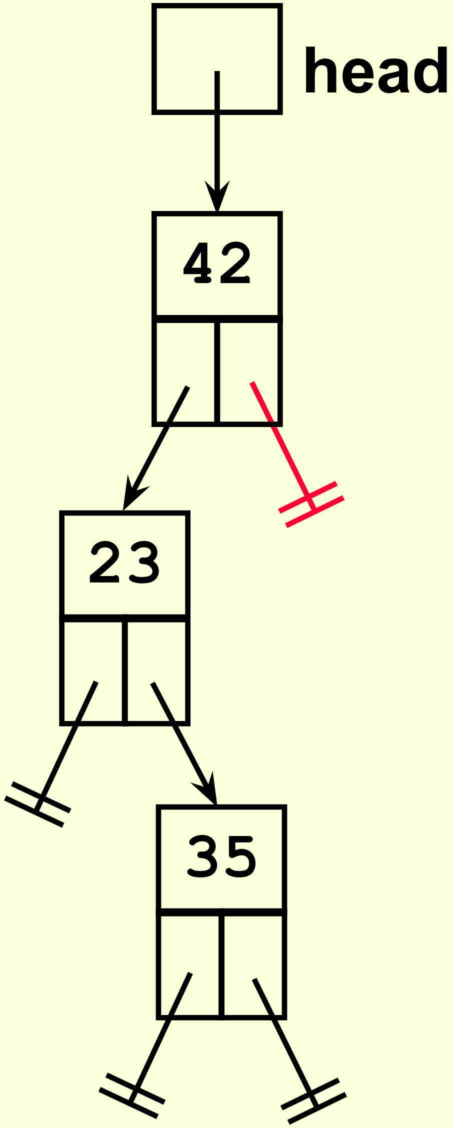


```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



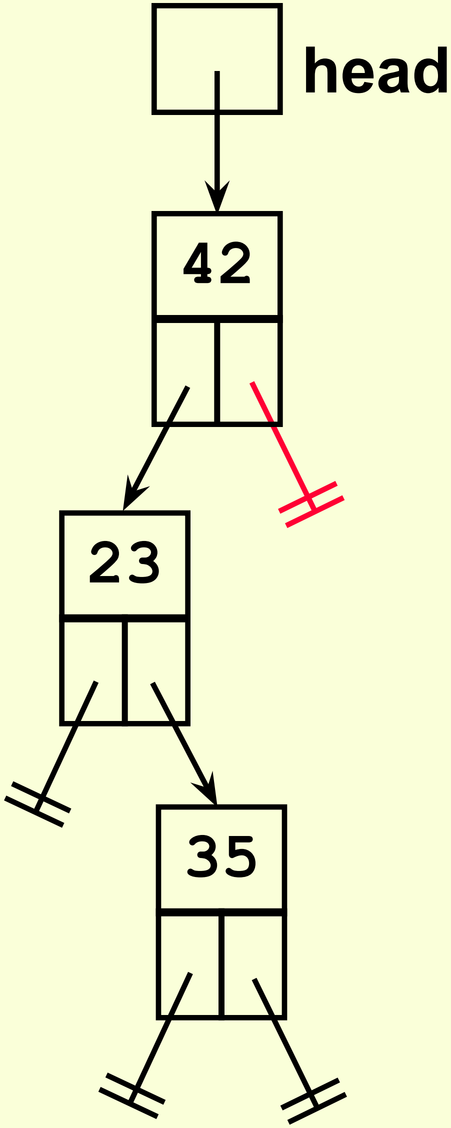
data_in = 47

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



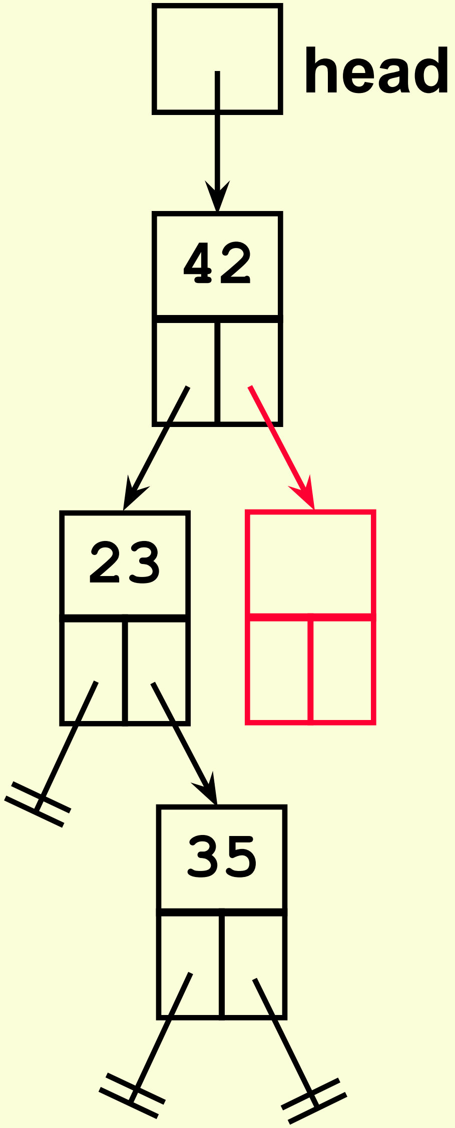
data_in = 47

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



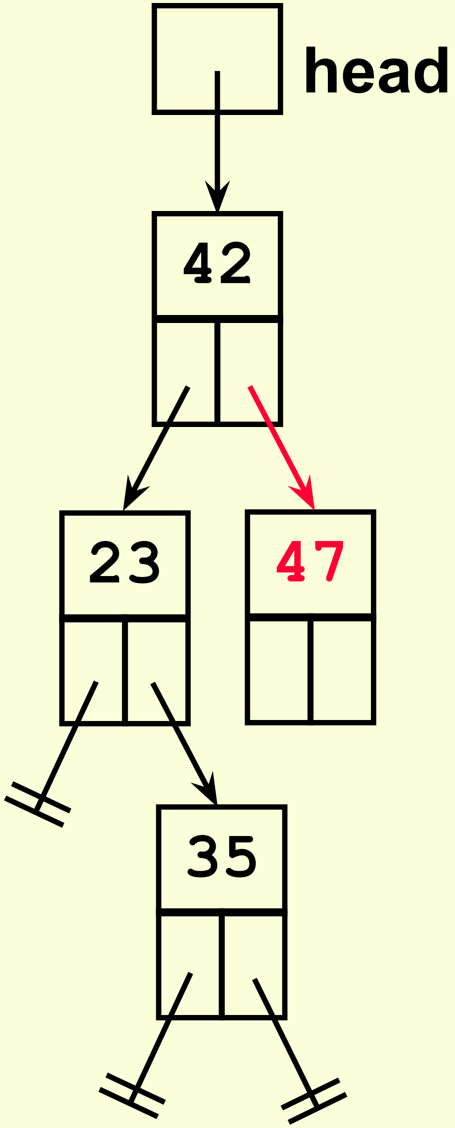
data_in = 47

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



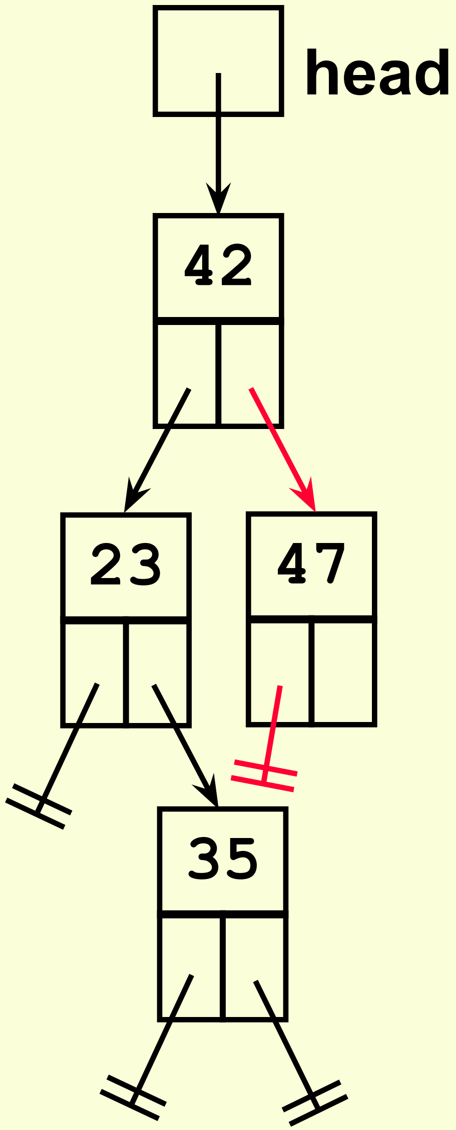
data_in = 47

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



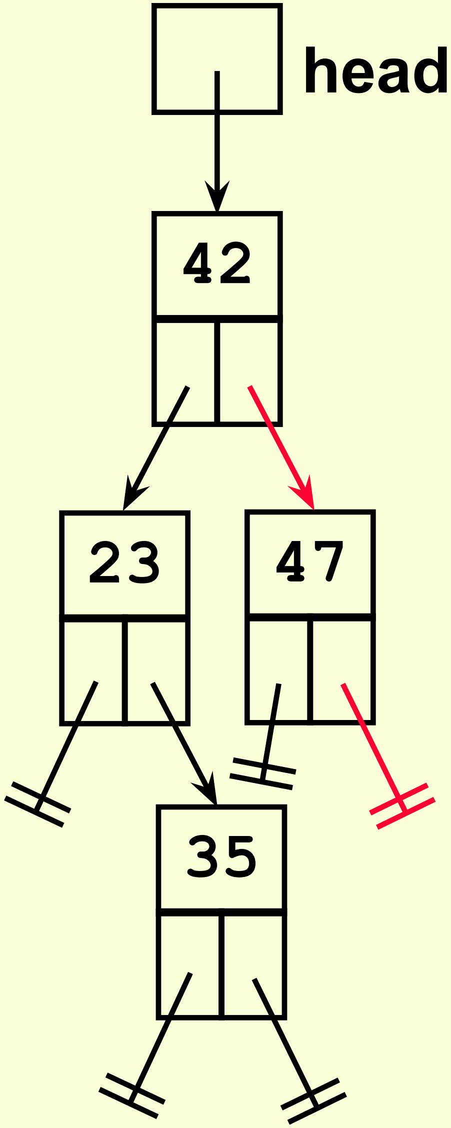
data_in = 47

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



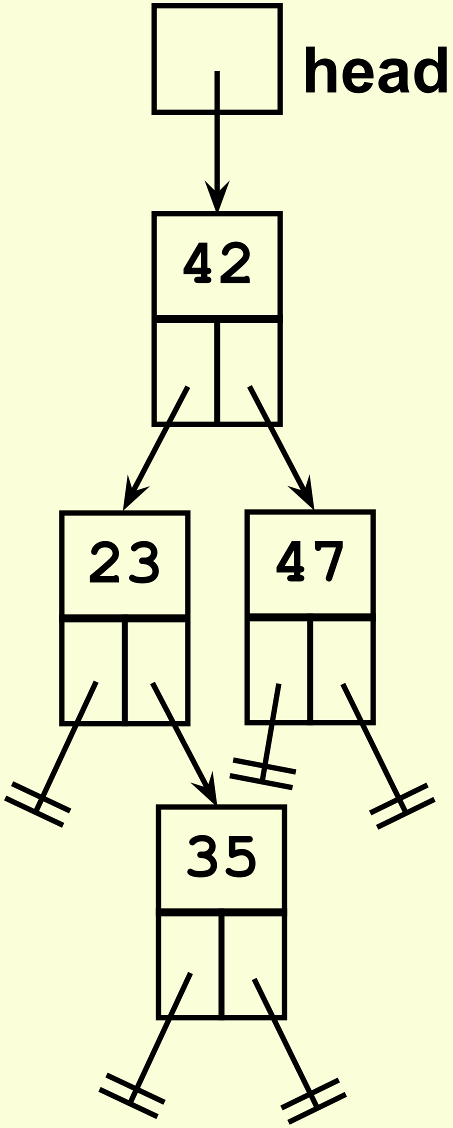
data_in = 47

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



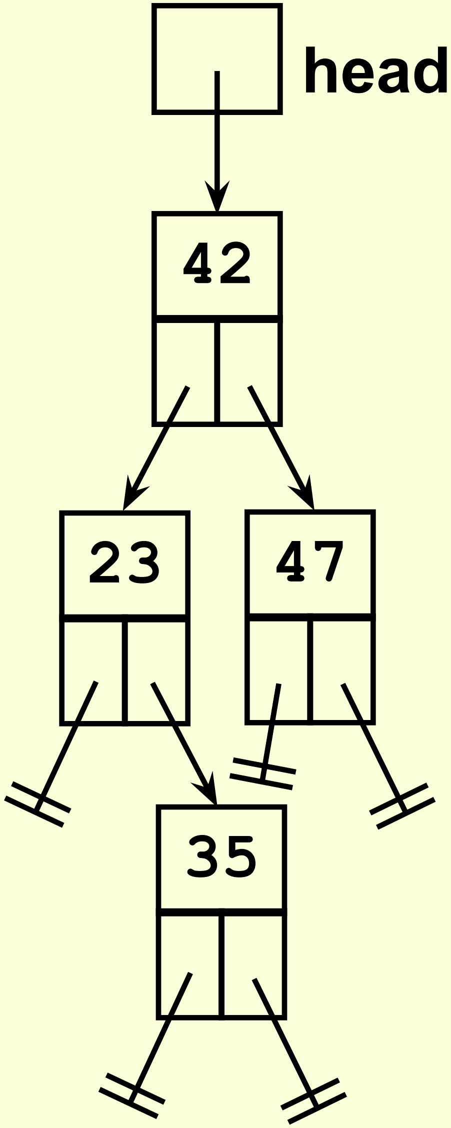
```
data_in = 47
```

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



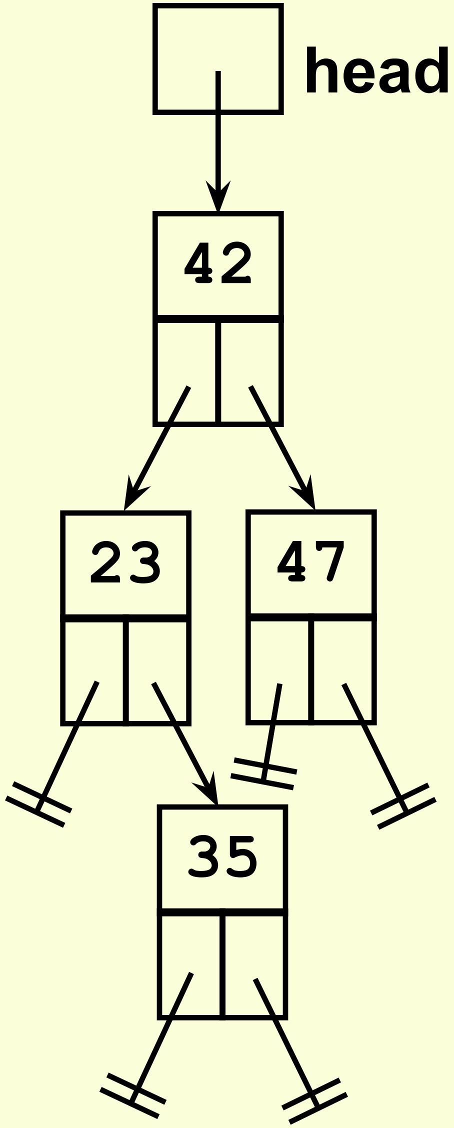
data_in = 47

```
procedure Insert(
 cur iot in/out Ptr toa Node,
 data_in iot in num)
 if(cur = NIL) then
  cur <- new(Node)
  cur^.data <- data_in
  cur^.left <- NIL
  cur^.right <- NIL
 elseif(cur^.data > data_in)
  Insert(cur^.left, data_in)
 else
  Insert(cur^.right, data_in)
 endif
endprocedure // Insert
```



data_in = 47

```
.
.
Insert(head, 23)
Insert(head, 35)
Insert(head, 47)
.
.
```



Summary

- Preserve "search" structure!
- Inserting involves 2 steps:
  - Find the correct location
    - For a BST insert, always insert at the "bottom" of the tree
  - Do commands to add node
    - Create node
    - Add data
    - Make left and right pointers point to nil

Questions?

Deleting from a Binary Search Tree

(BST)

The Scenario

- We have a Binary Search Tree and want to remove some element based upon a match.

- Must preserve “search” property
- Must not lose any elements (i.e. only remove the one element)

BST Deletion

- Search for desired item.
- If not found, then return NIL or print error.
- If found, perform steps necessary to accomplish removal from the tree.

Four Cases for Deletion

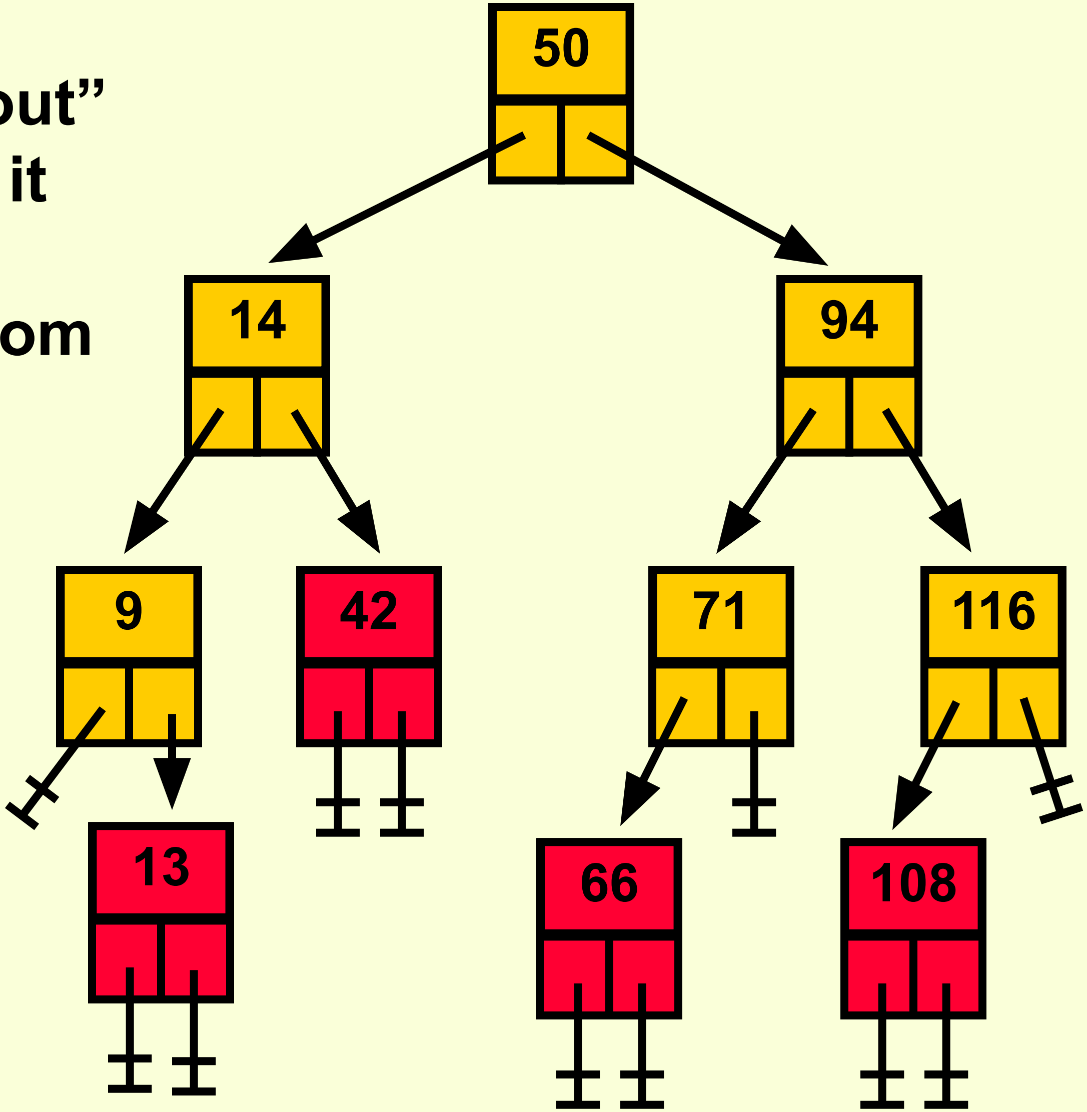
- Delete a leaf node
- Delete a node with only one child (left)
- Delete a node with only one child (right)
- Delete a node with two children

Cases 2 and 3 are comparable and only need slight changes in the conditional statement used

Delete a Leaf Node

Simply use an “in/out” pointer and assign it to “nil”. This will remove the node from the tree.

```
cur <- nil
```

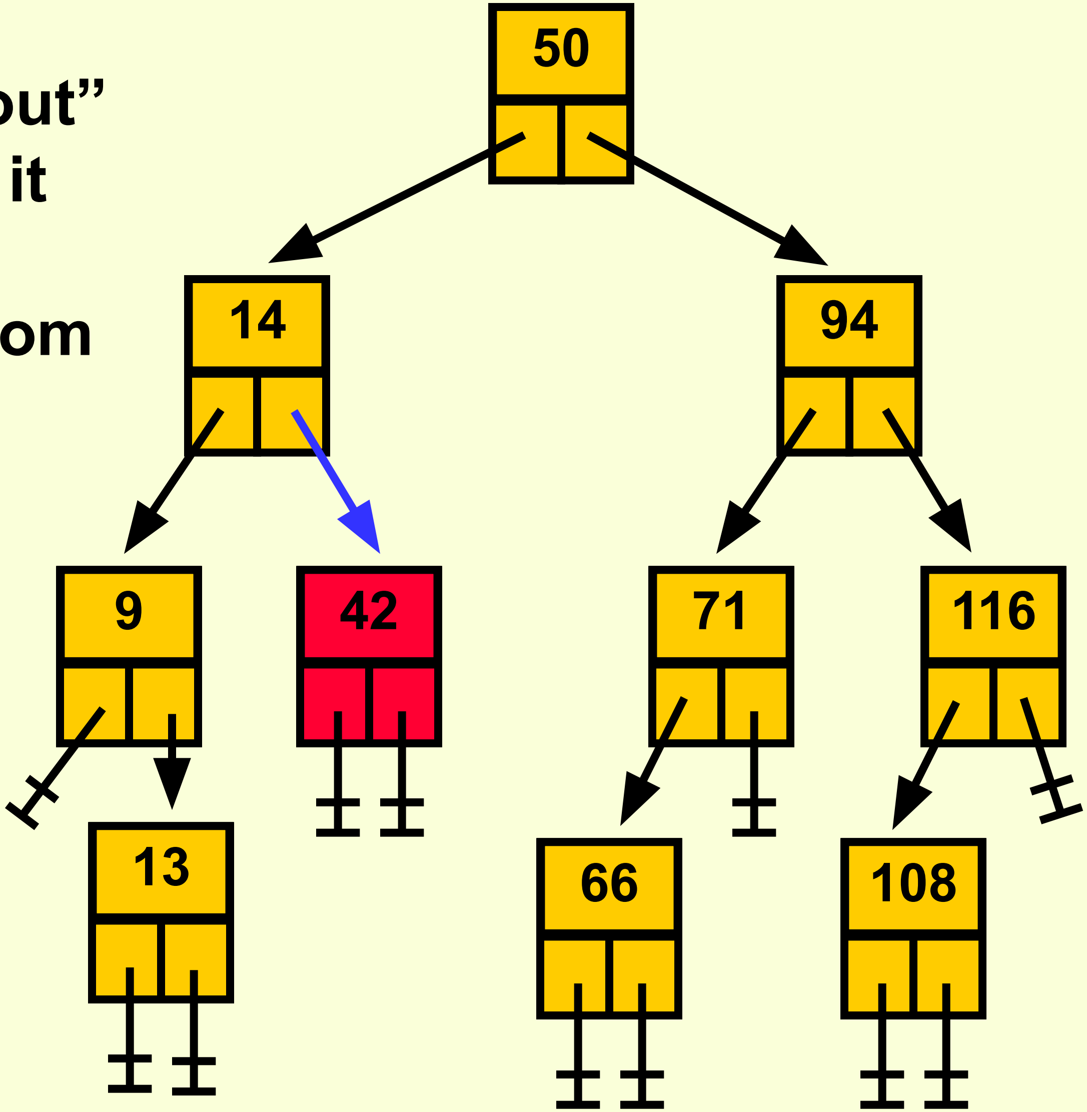


Delete a Leaf Node

Simply use an "in/out" pointer and assign it to "nil". This will remove the node from the tree.

```
cur <- nil
```

Let's delete 42.

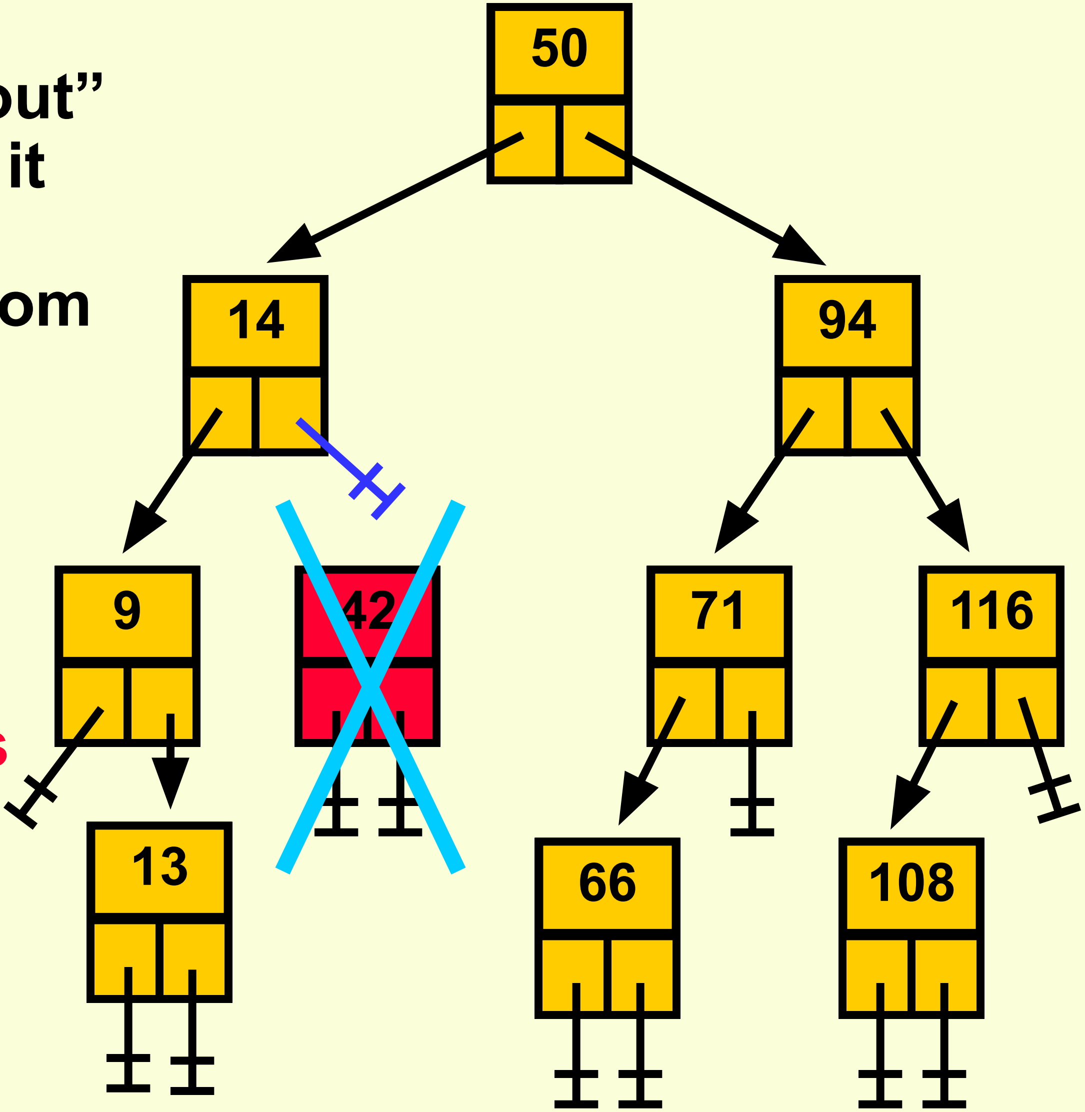


Delete a Leaf Node

Simply use an “in/out” pointer and assign it to “nil”. This will remove the node from the tree.

```
cur <- nil
```

Move the pointer; now nothing points to the node.

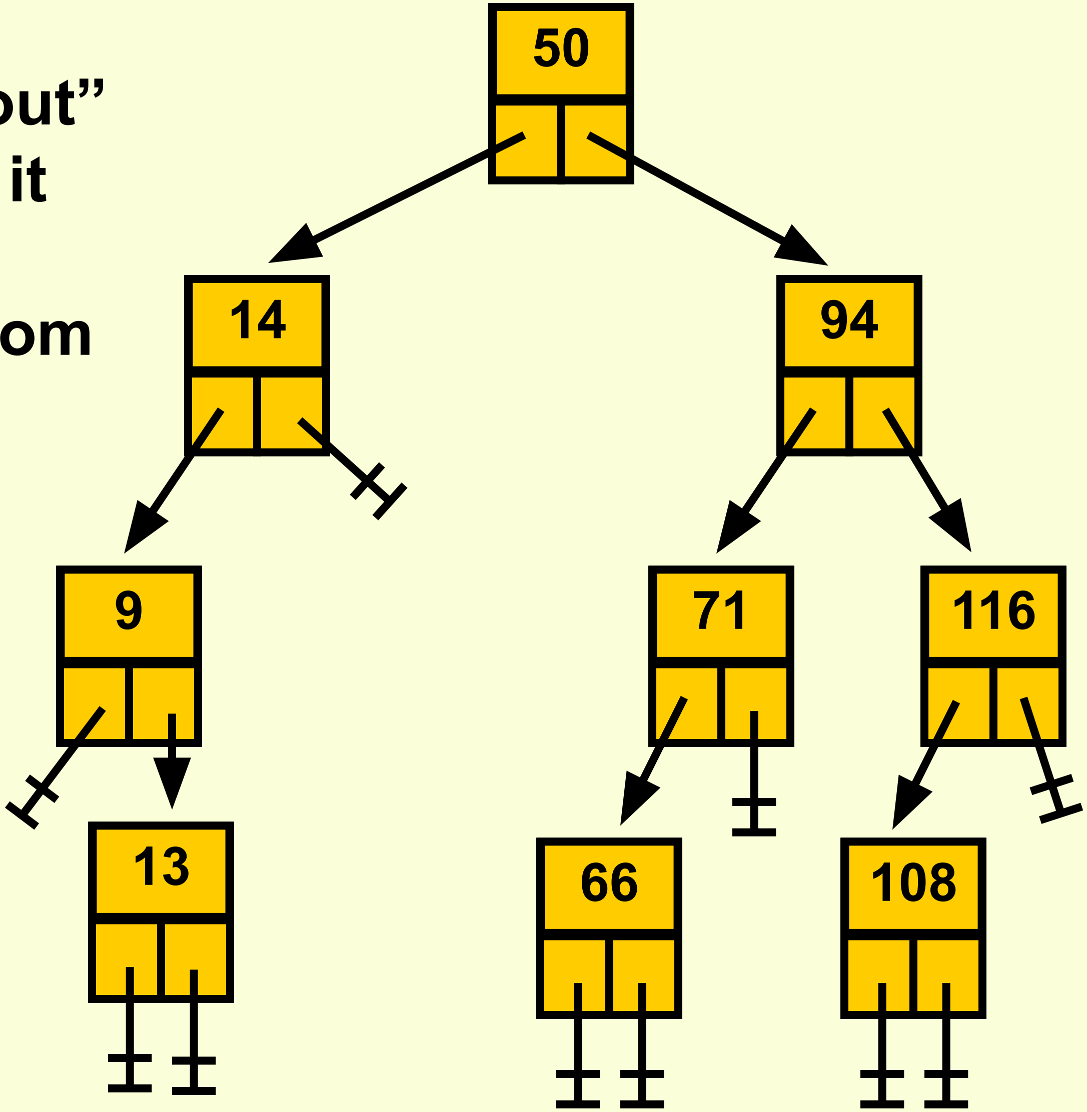


Delete a Leaf Node

Simply use an “in/out” pointer and assign it to “nil”. This will remove the node from the tree.

```
cur <- nil
```

The resulting tree.



Delete a Node with One Child

Use an “in/out” pointer.

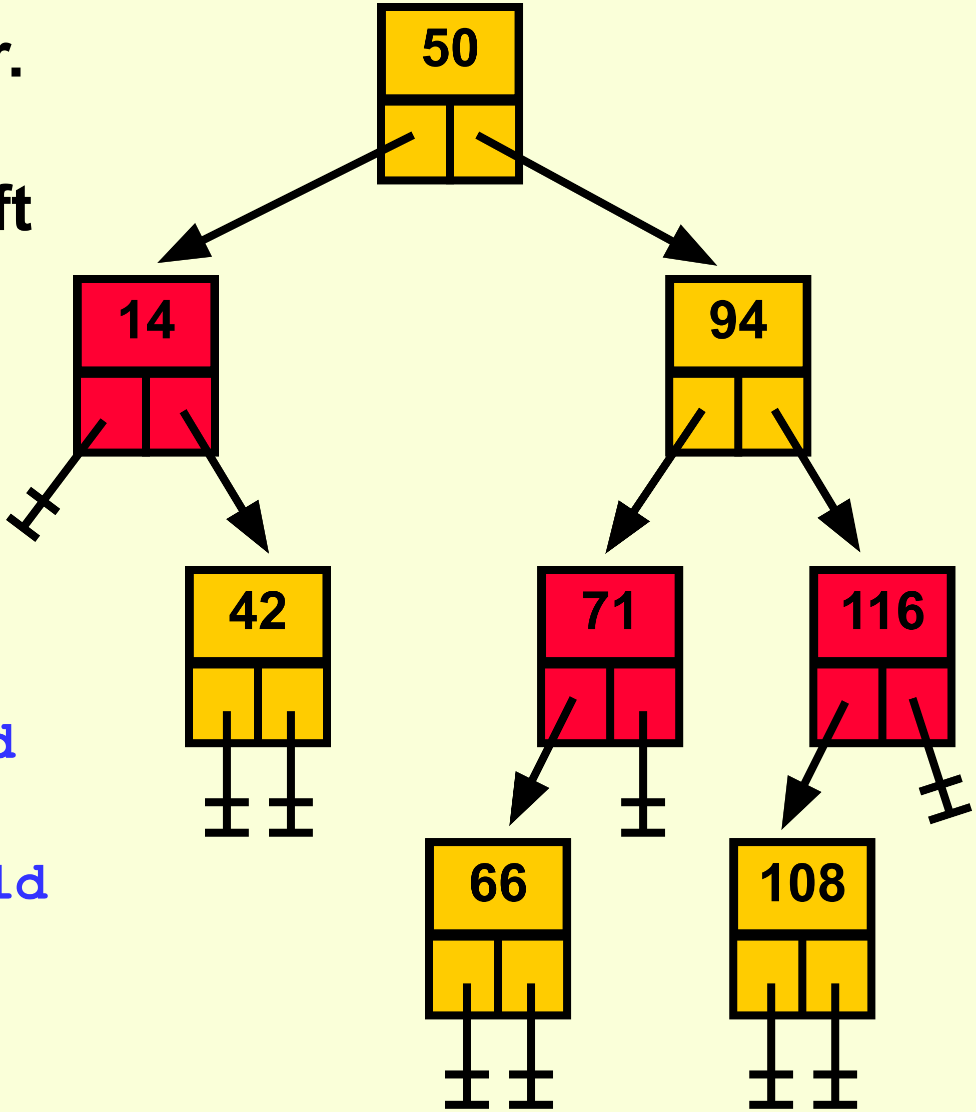
Determine if it has a left or a right child.

Point the current pointer to the appropriate child:

```
cur <- cur^.left_child
```

or

```
cur <- cur^.right_child
```



Delete a Node with One Child

Use an “in/out” pointer.

Determine if it has a left or a right child.

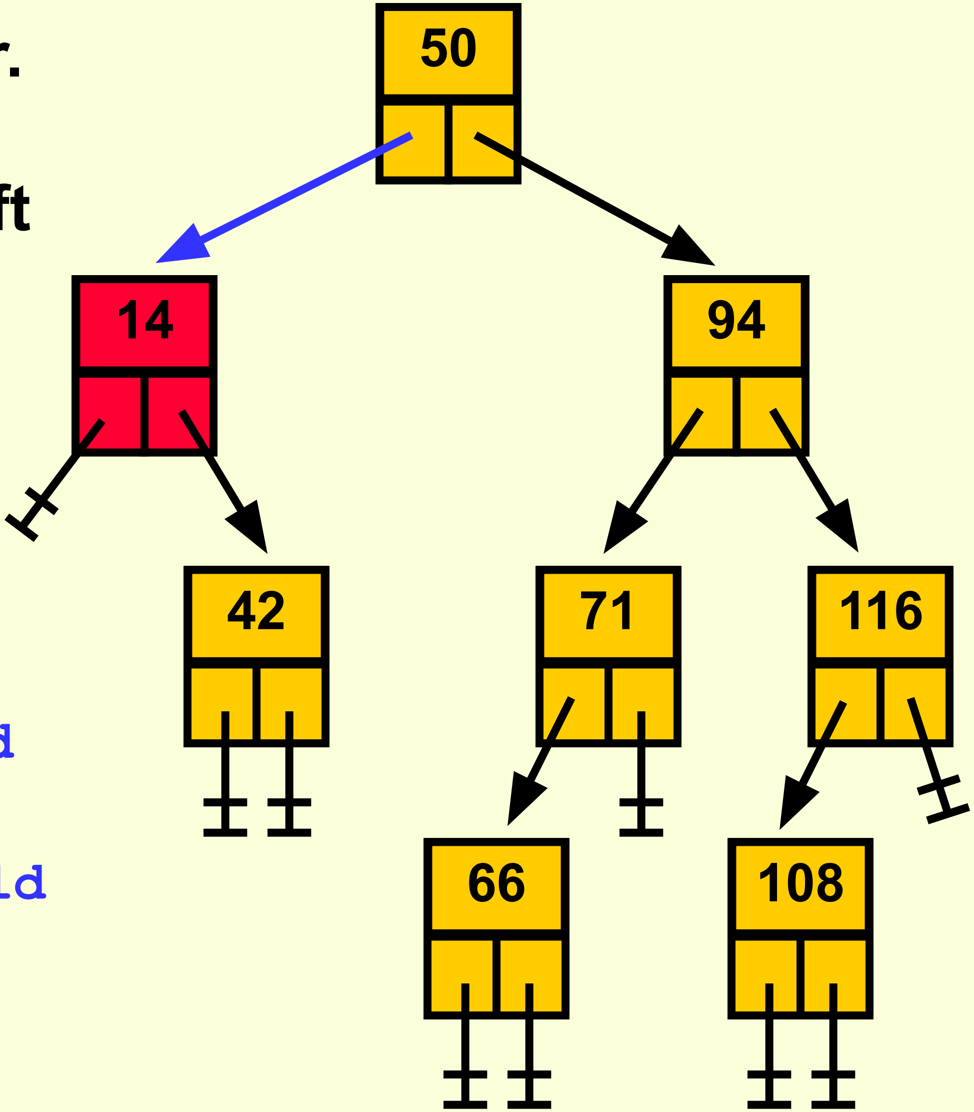
Point the current pointer to the appropriate child:

```
cur <- cur^.left_child
```

or

```
cur <- cur^.right_child
```

Let’s delete 14.



Delete a Node with One Child

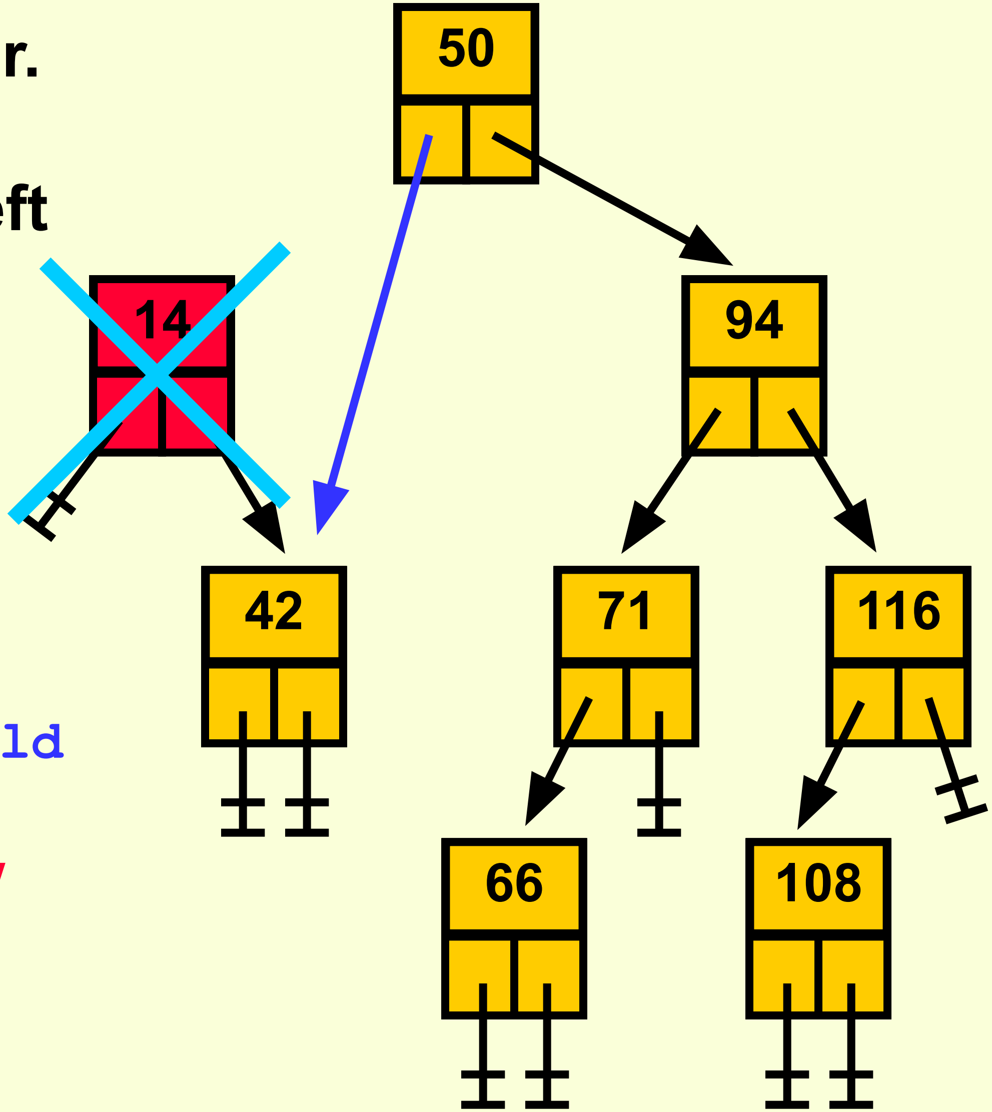
Use an “in/out” pointer.

Determine if it has a left or a right child.

Point the current pointer to the appropriate child:

```
cur <- cur^.right_child
```

Move the pointer; now nothing points to the node.



Delete a Node with One Child

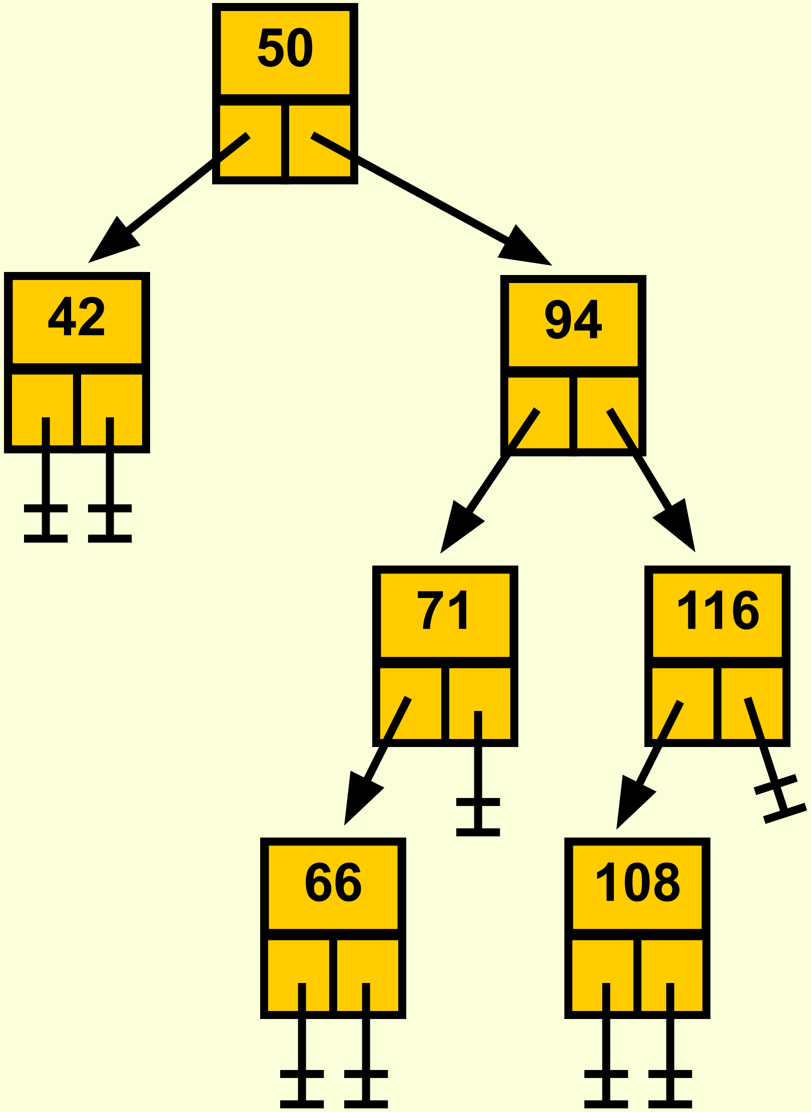
Use an "in/out" pointer.

Determine if it has a left or a right child.

Point the current pointer to the appropriate child:

```
cur <- cur^.right_child
```

The resulting tree.



Delete a Node with One Child

Use an "in/out" pointer.

Determine if it has a left or a right child.

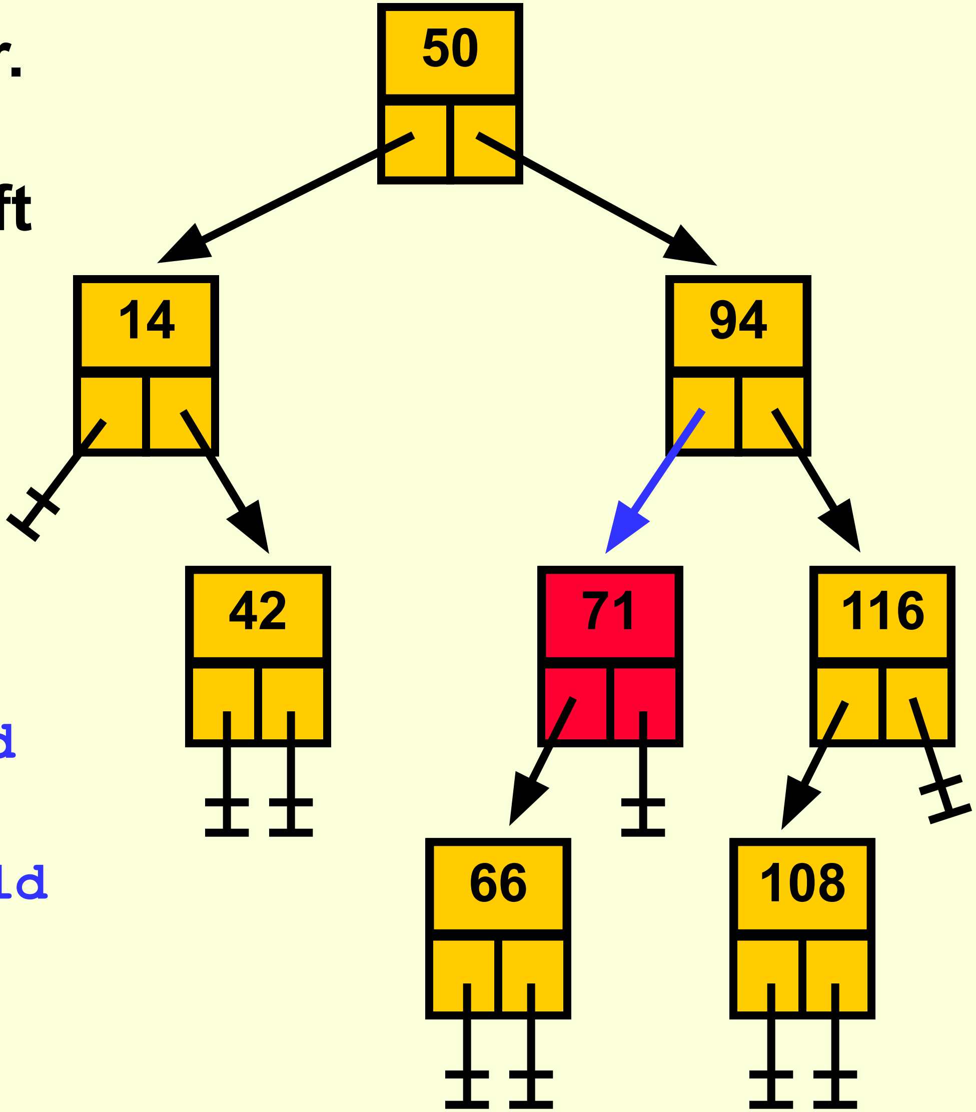
Point the current pointer to the appropriate child:

```
cur <- cur^.left_child
```

or

```
cur <- cur^.right_child
```

Let's delete 71.



Delete a Node with One Child

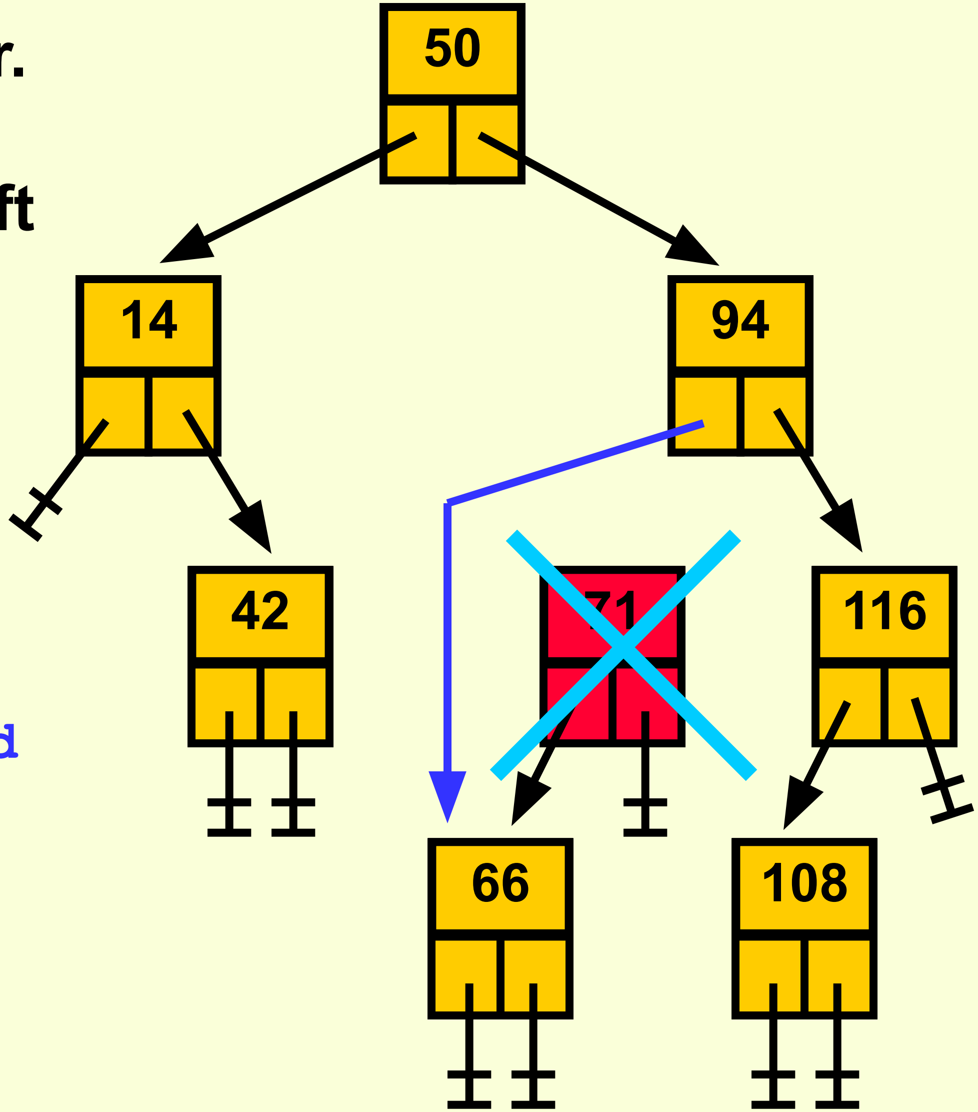
Use an "in/out" pointer.

Determine if it has a left or a right child.

Point the current pointer to the appropriate child:

```
cur <- cur^.left_child
```

Move the pointer; now nothing points to the node.



Delete a Node with One Child

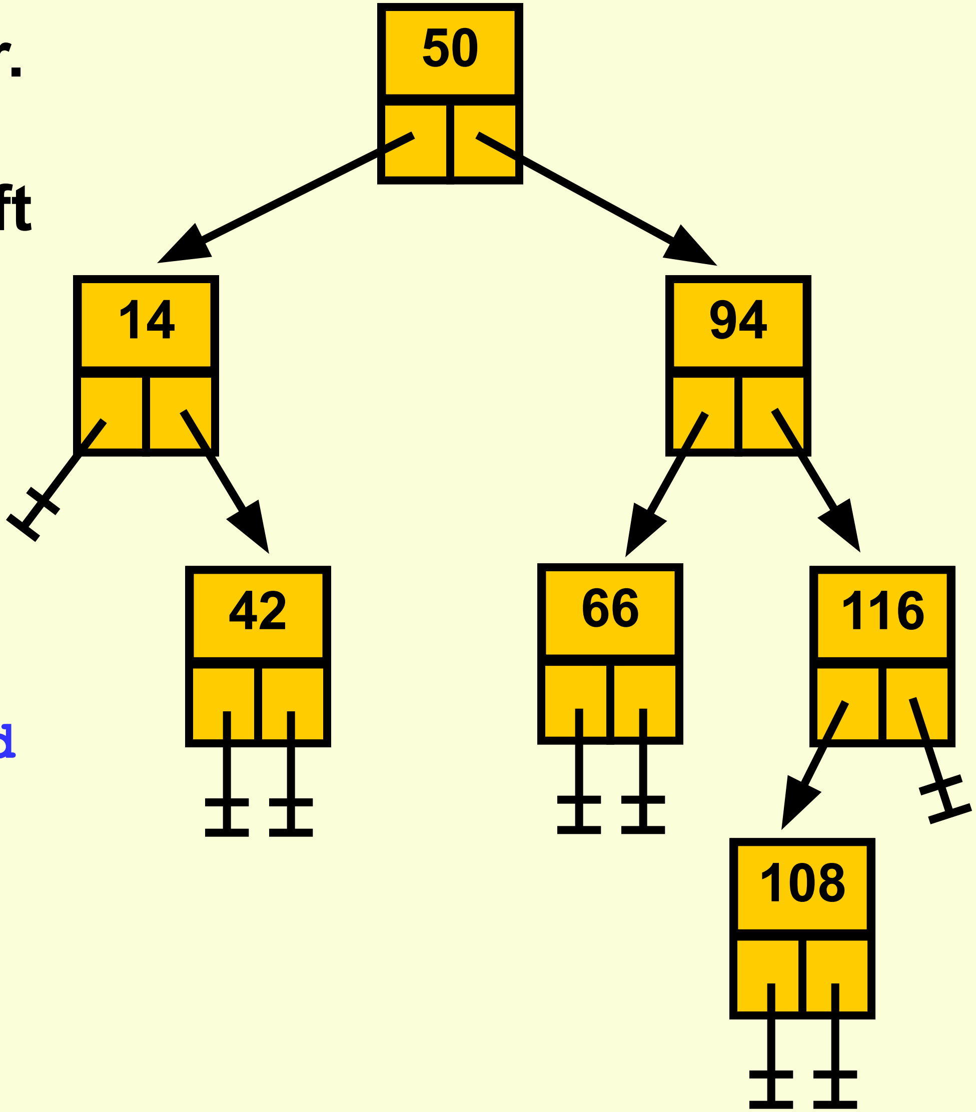
Use an "in/out" pointer.

Determine if it has a left or a right child.

Point the current pointer to the appropriate child:

```
cur <- cur^.left_child
```

The resulting tree.



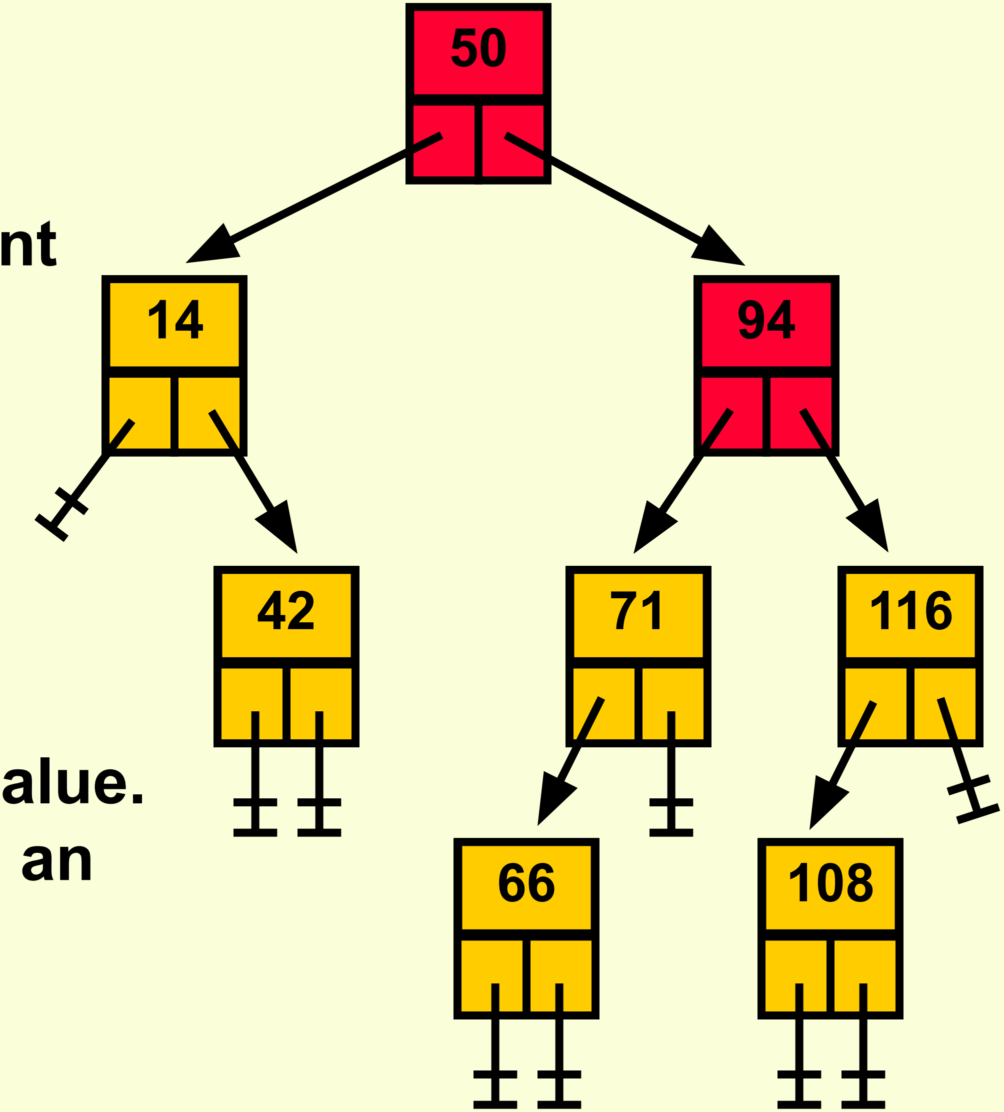
Delete a Node with Two Children

Copy a replacement value from a descendant node.

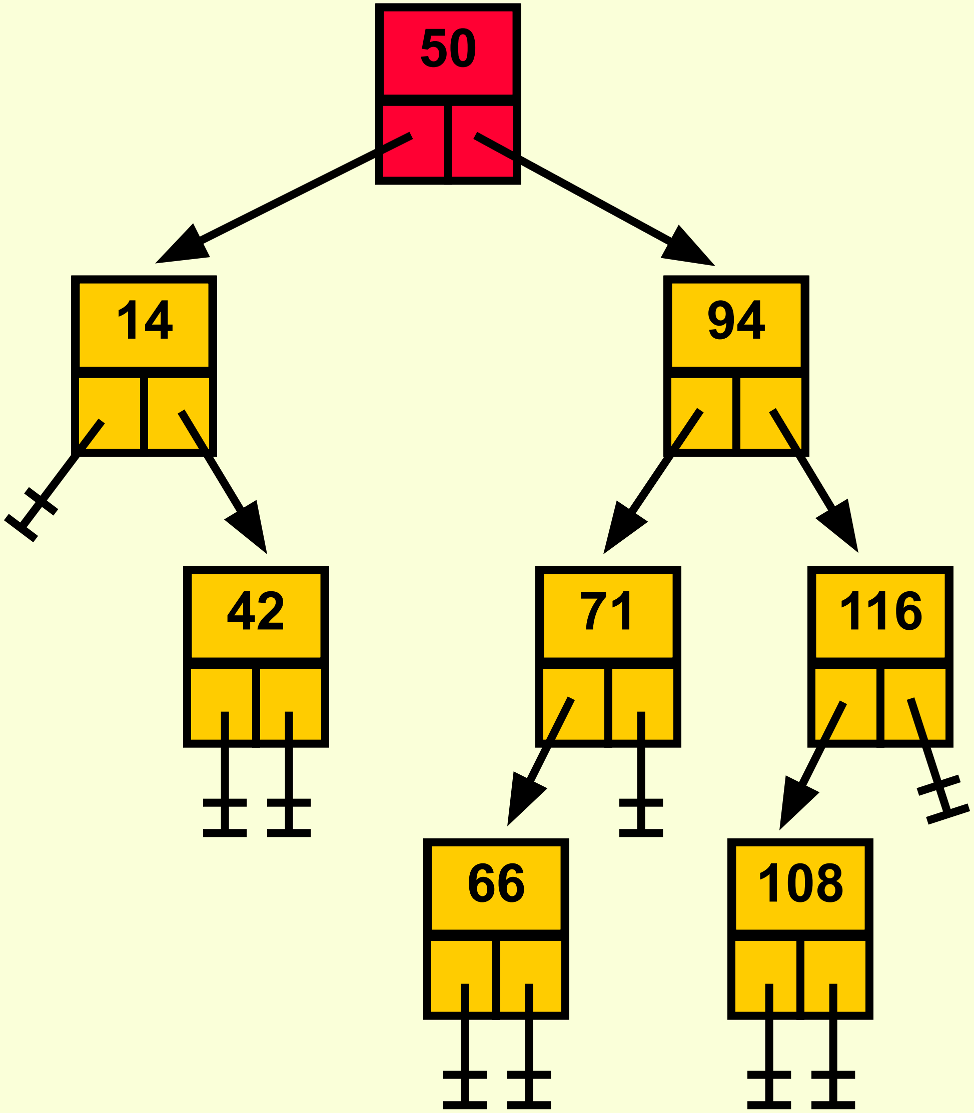
- Largest from left
- Smallest from right

Then delete that descendant node to remove the duplicate value.

- We know this will be an easier case.

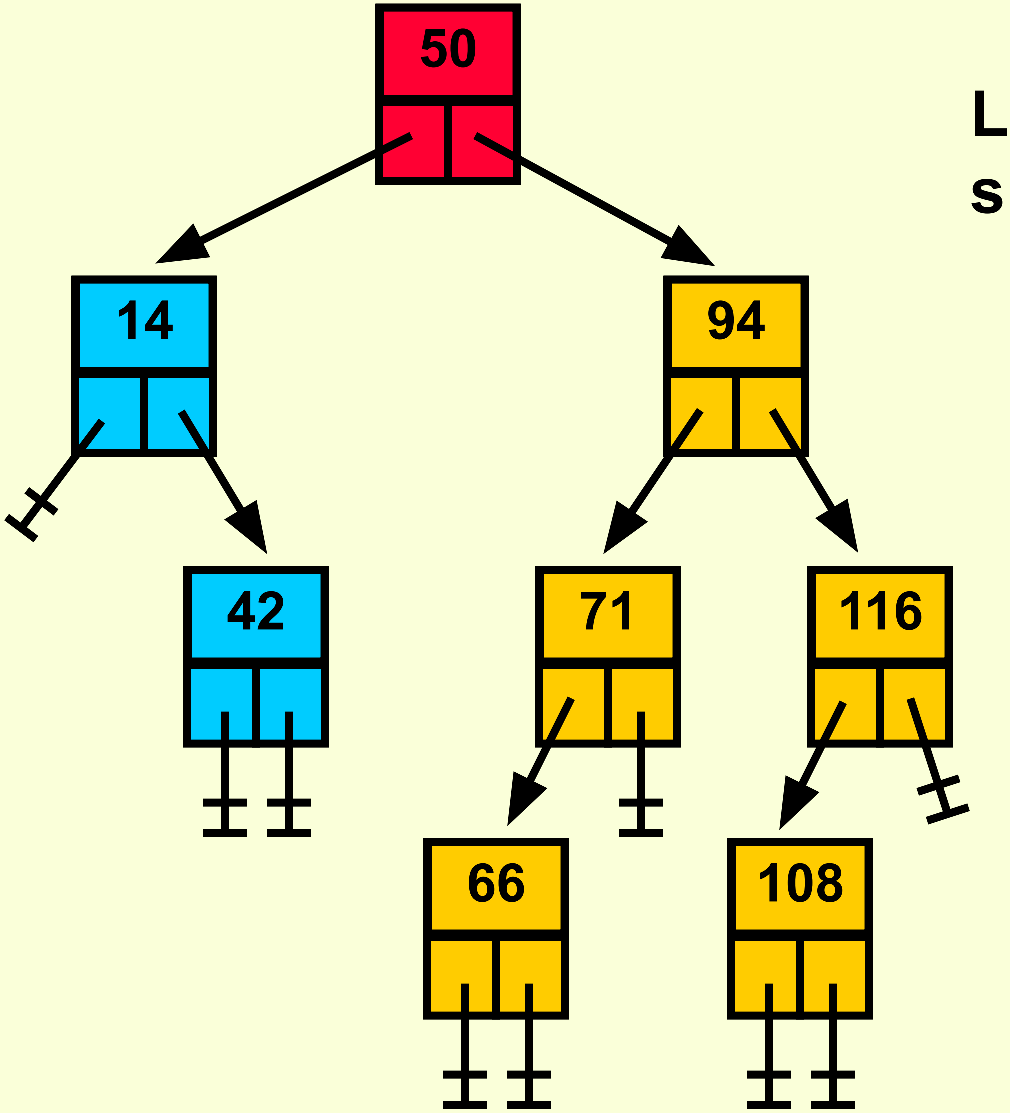


Delete a Node with Two Children



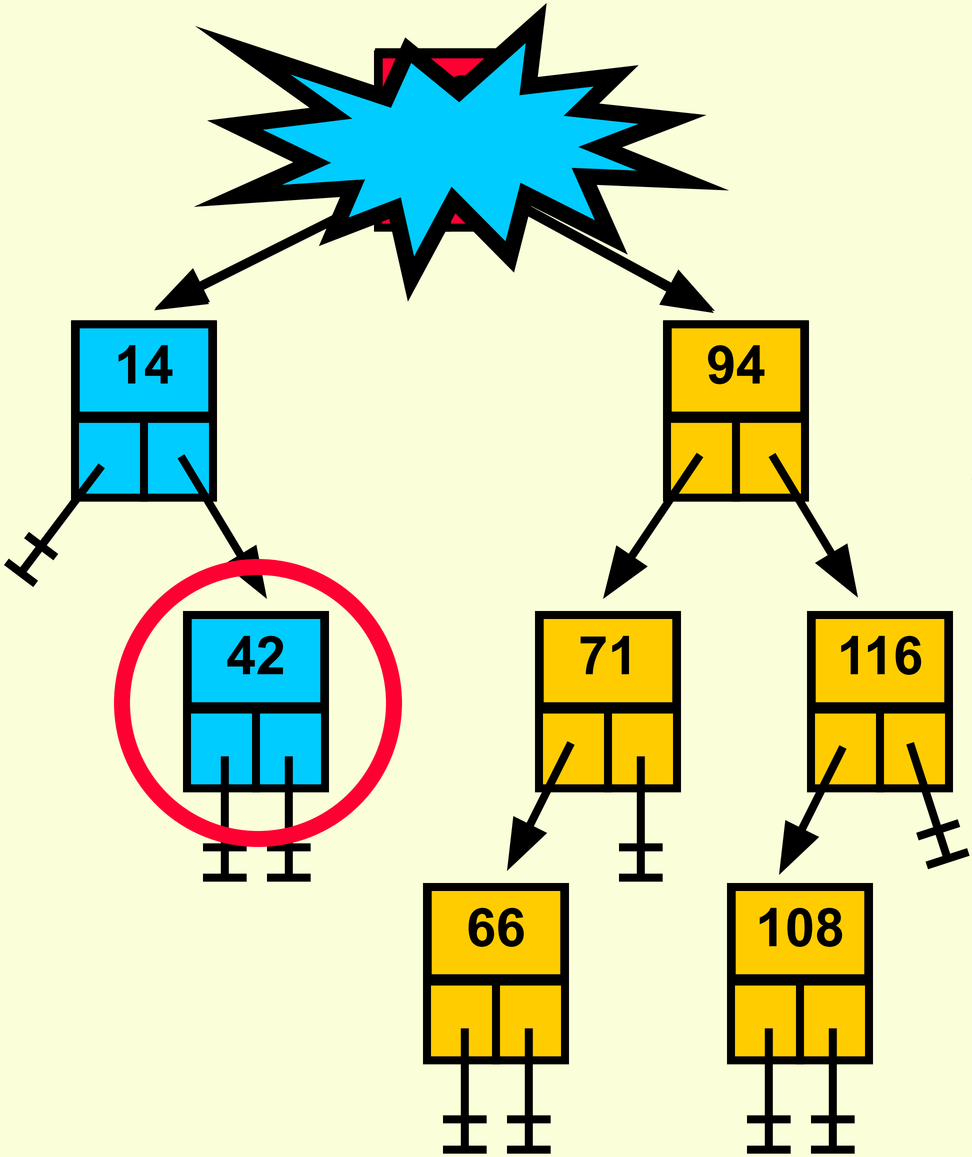
Let’s delete 50.

Delete a Node with Two Children



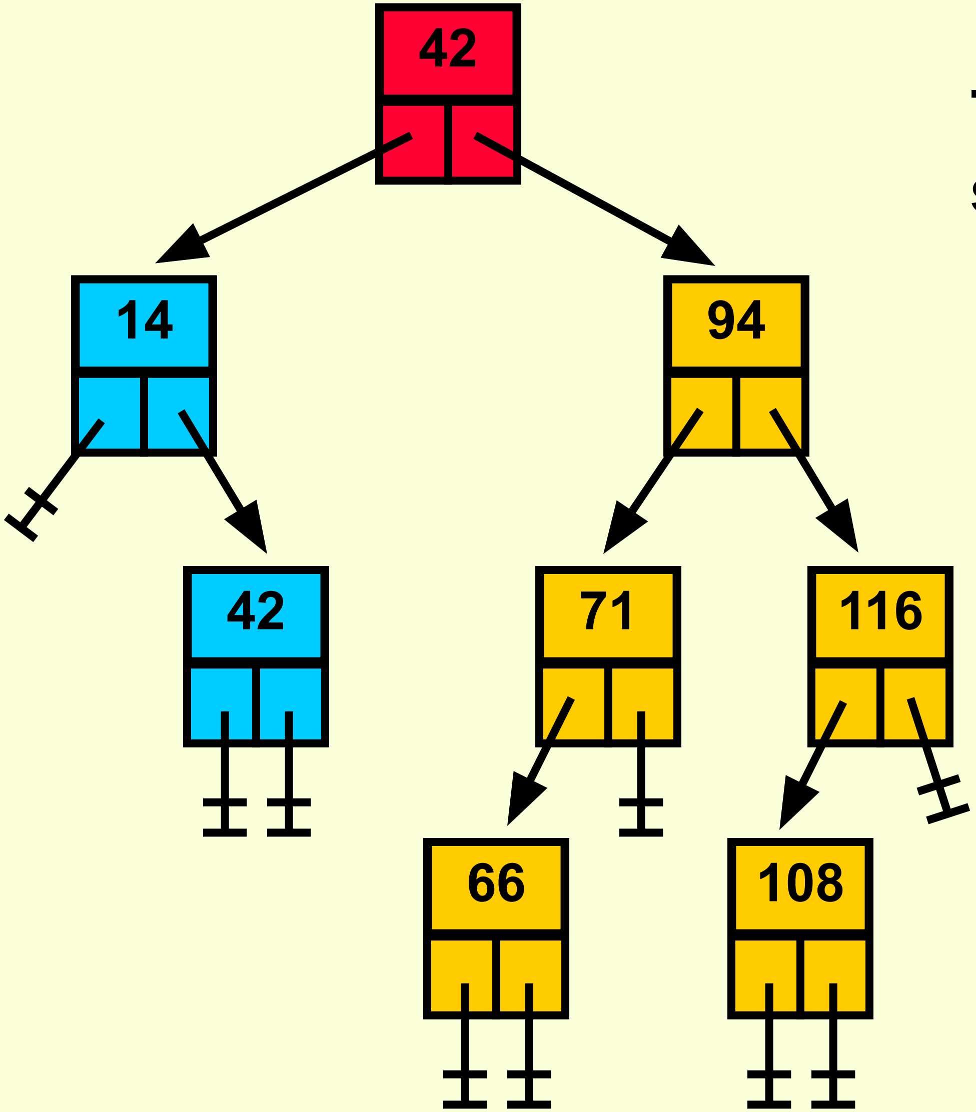
Look to the left sub-tree.

Delete a Node with Two Children



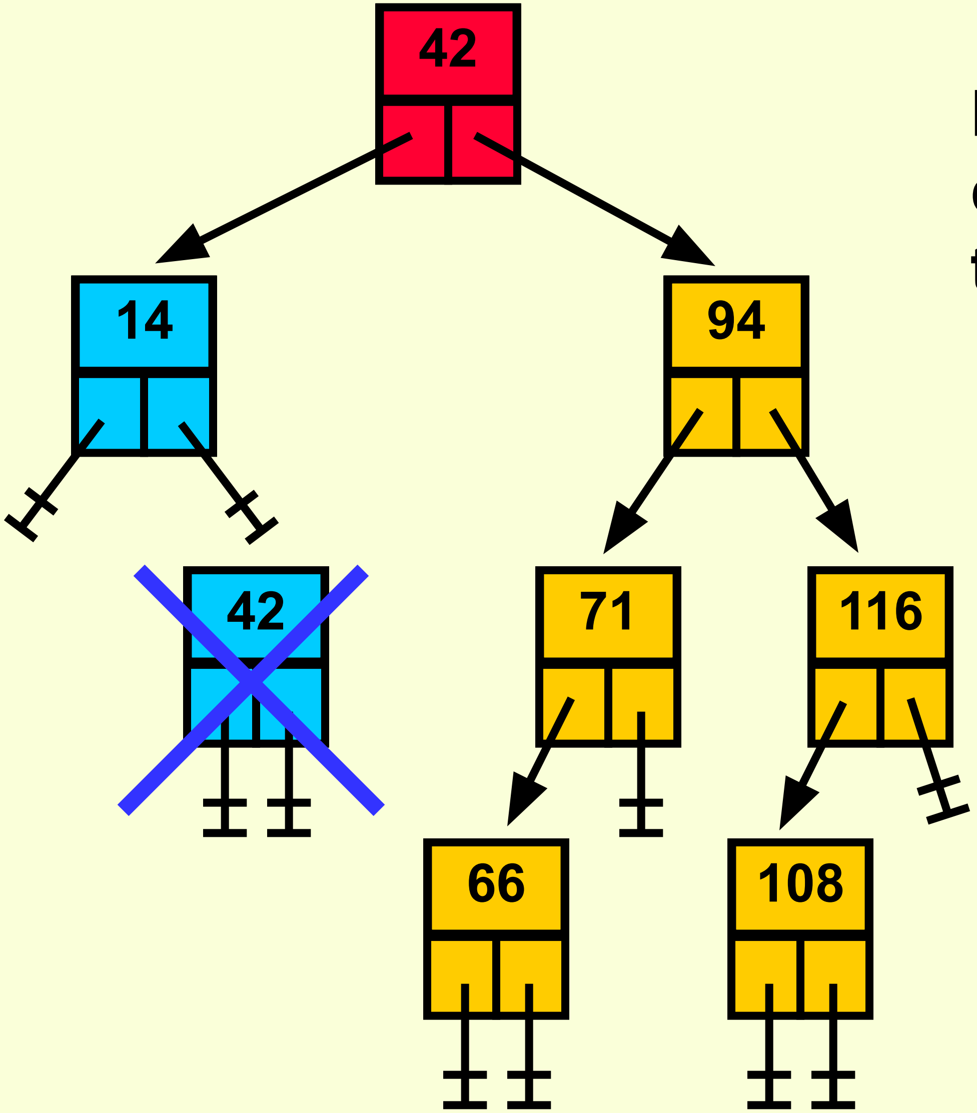
Find and copy the largest value
(this will erase the old value but creates a duplicate).

Delete a Node with Two Children



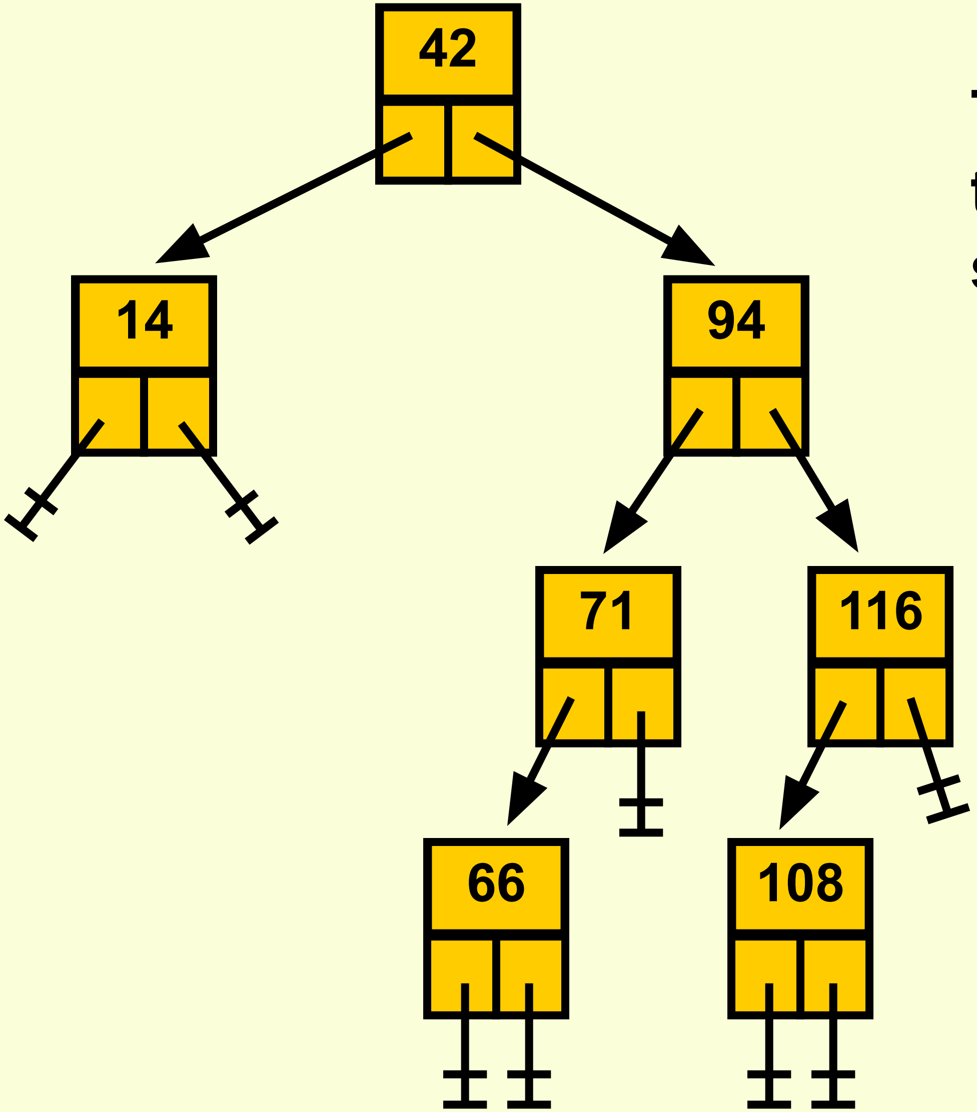
The resulting tree so far.

Delete a Node with Two Children



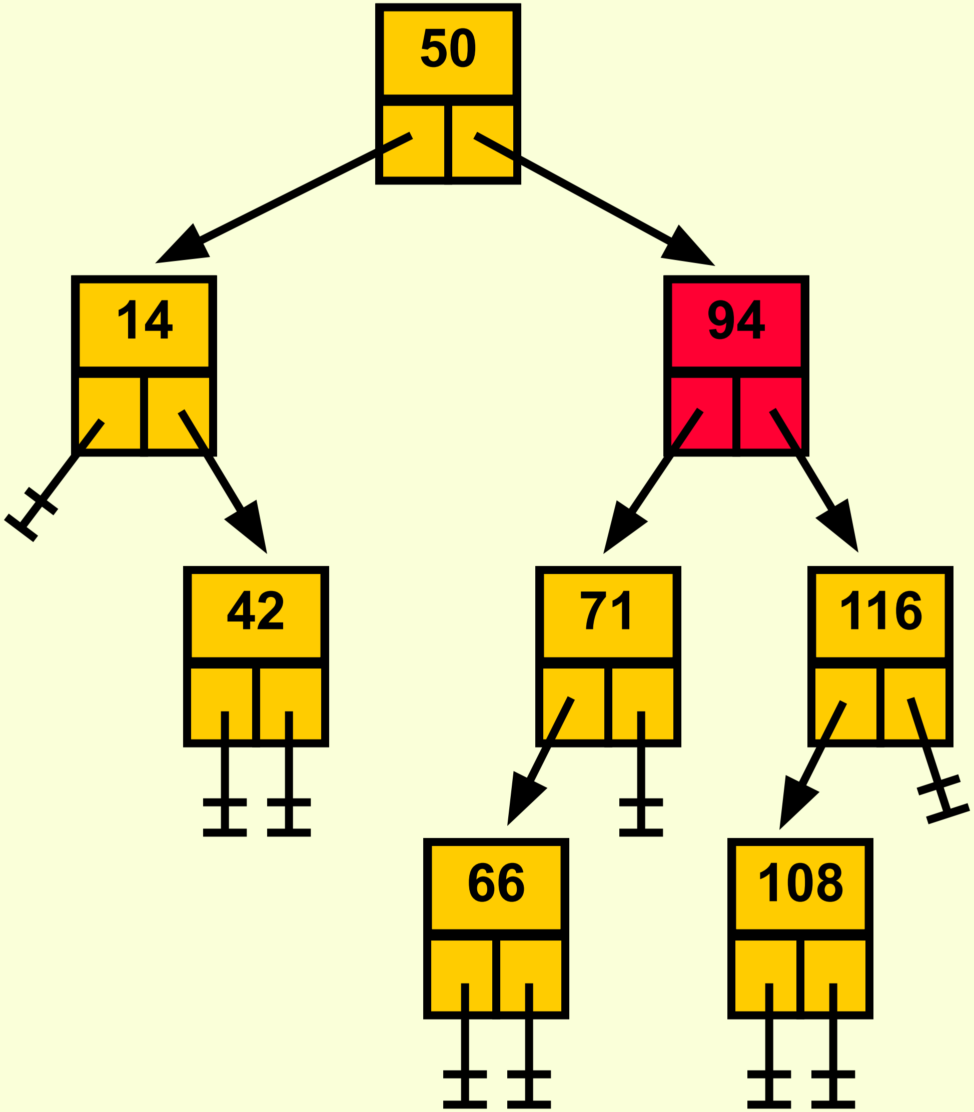
Now delete the duplicate from the left sub-tree.

Delete a Node with Two Children



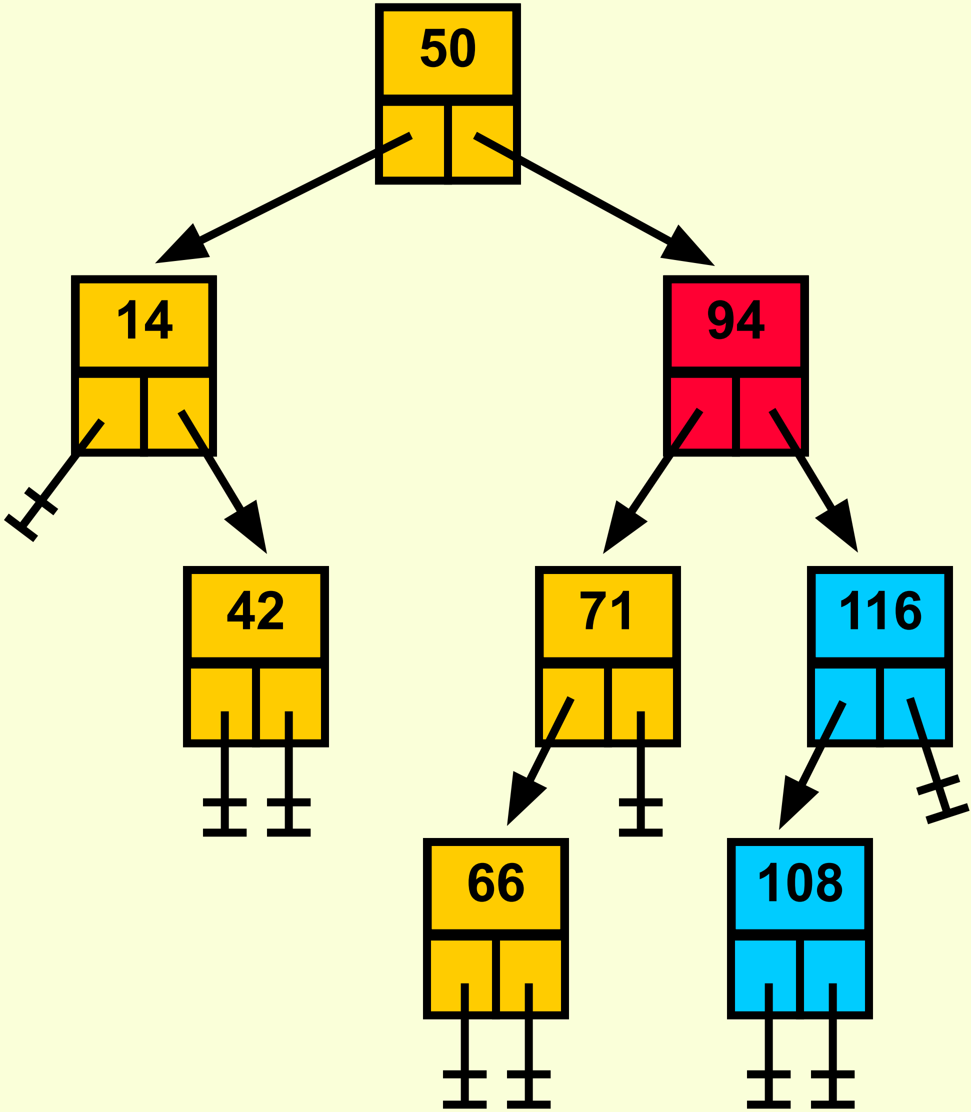
The final resulting tree – still has search structure.

Delete a Node with Two Children



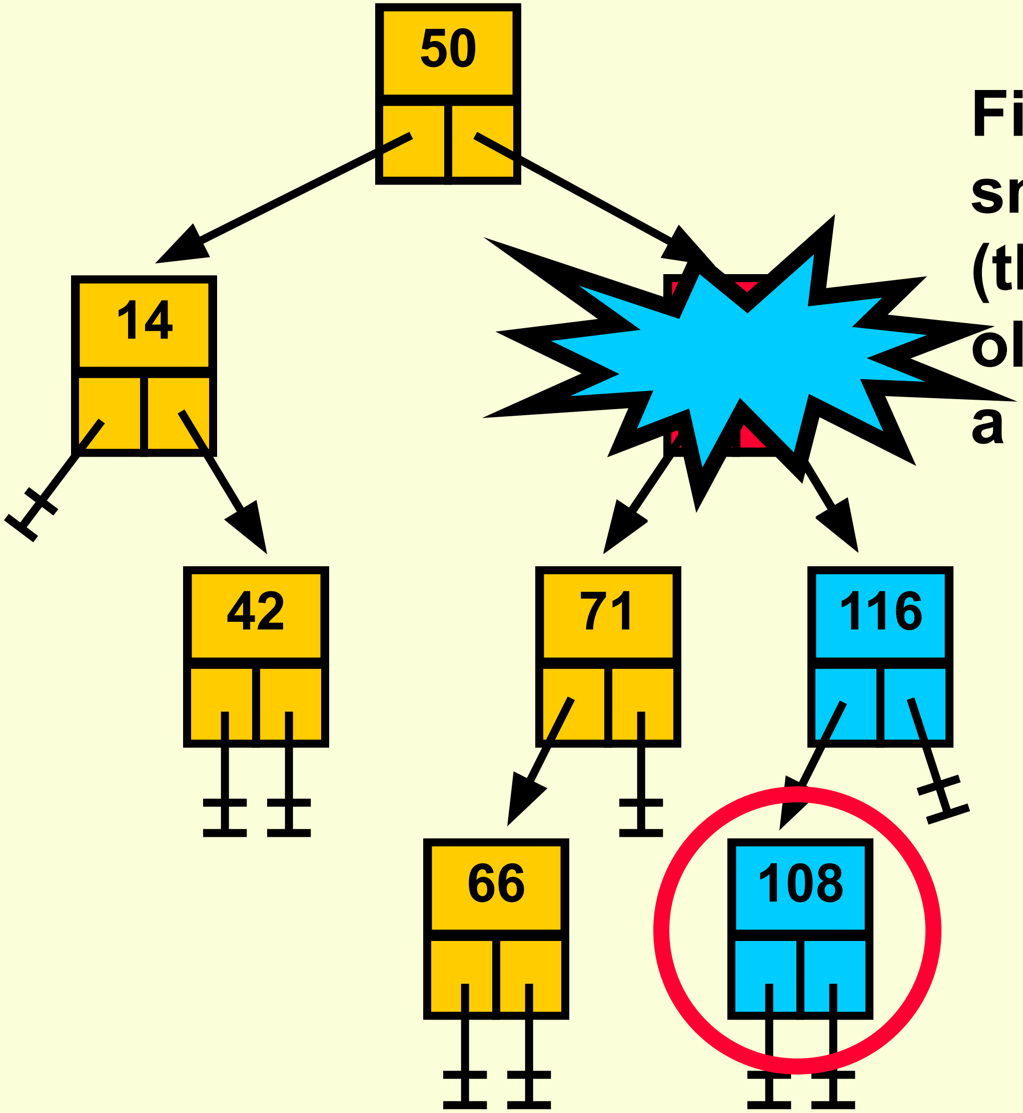
Let’s delete 94.

Delete a Node with Two Children



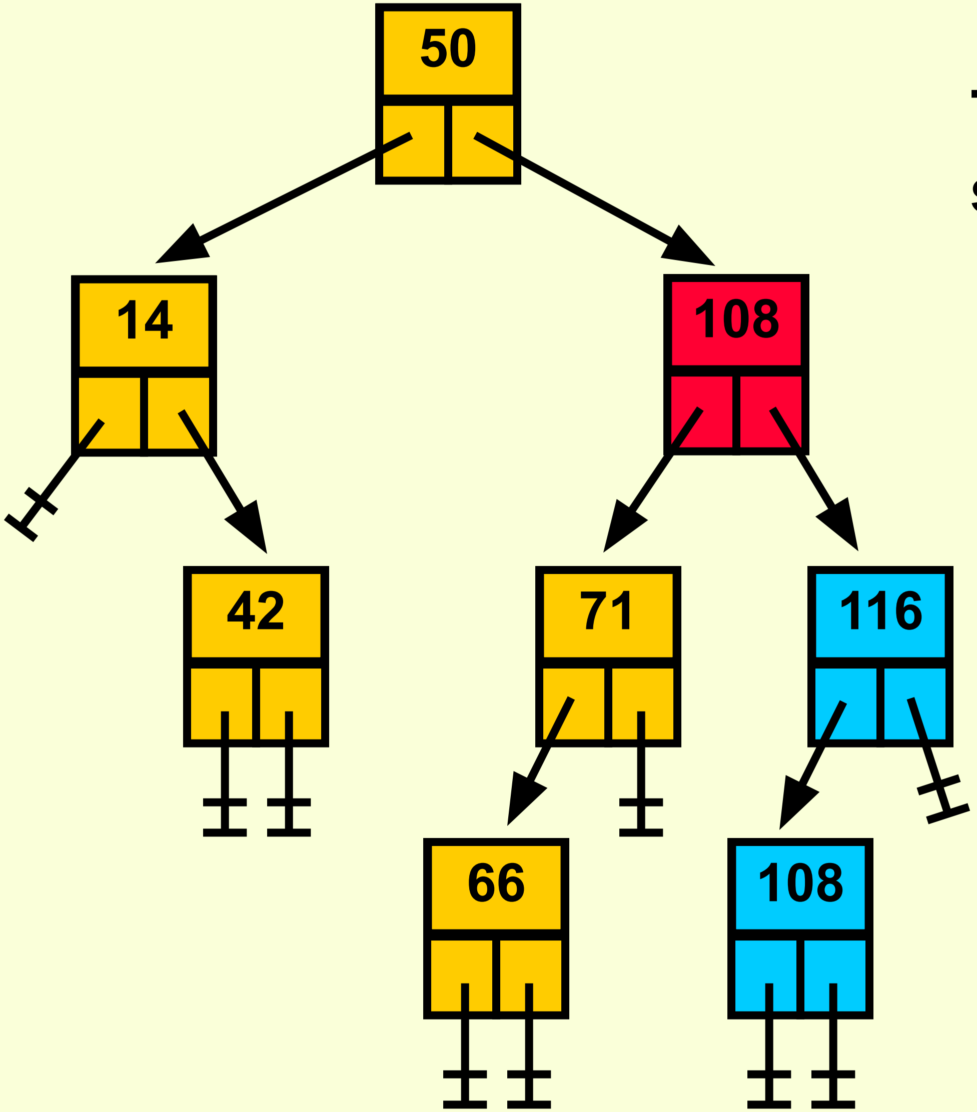
Look to the right sub-tree.

Delete a Node with Two Children



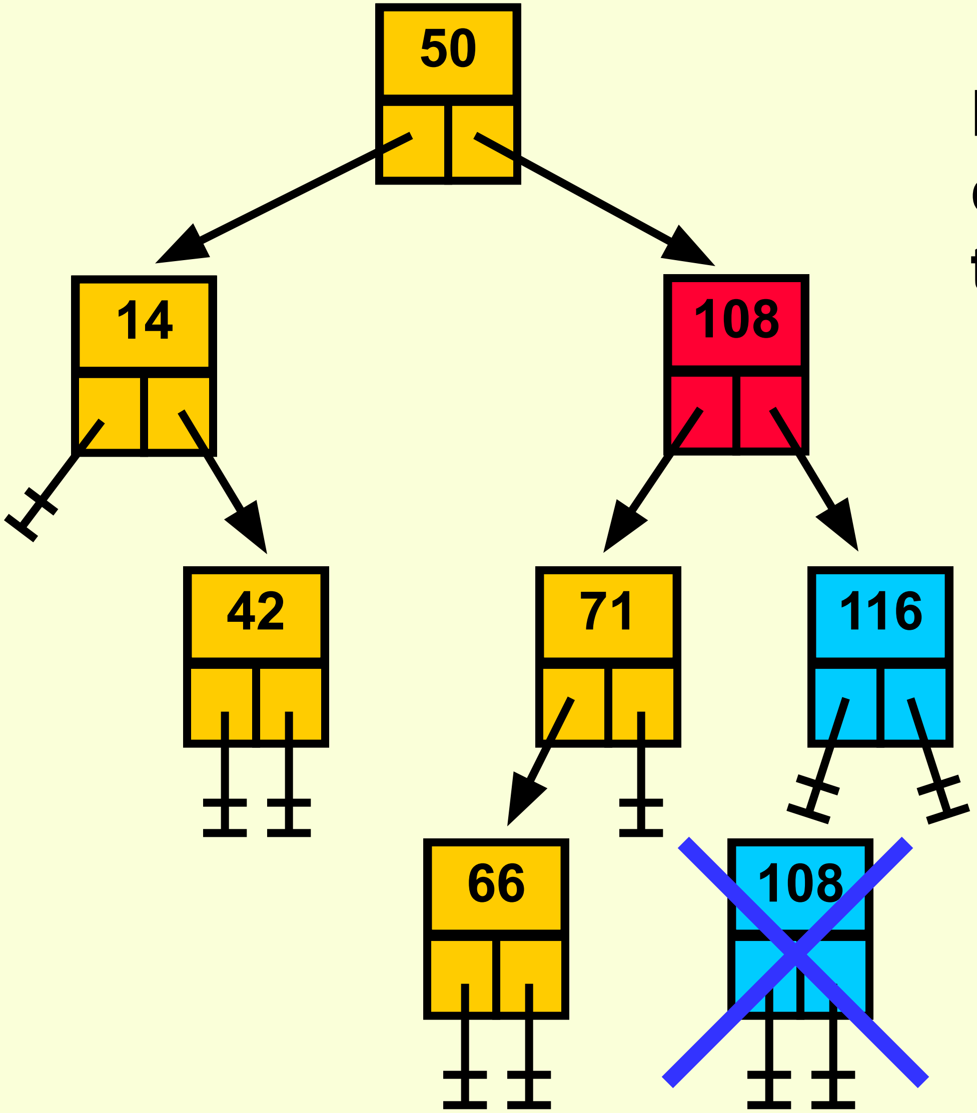
Find and copy the
smallest value
(this will erase the
old value but creates
a duplicate).

Delete a Node with Two Children



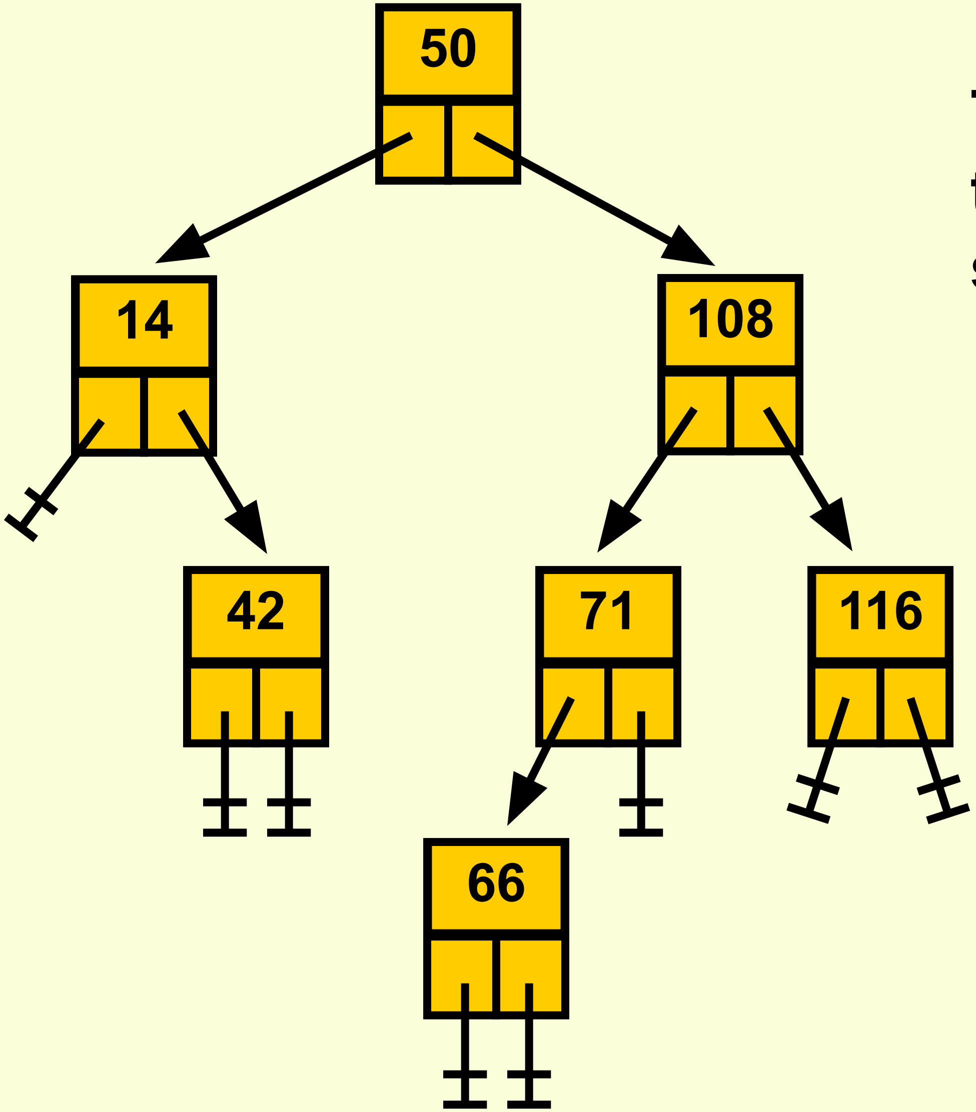
The resulting tree so far.

Delete a Node with Two Children



Now delete the duplicate from the left sub-tree.

Delete a Node with Two Children



The final resulting tree – still has search structure.

Summary

- Deleting a node from a binary search tree involves two steps:
  - Search for the element
  - Then perform the deletion
- We must preserve the search structure and only delete the element which matches.
- Four cases:
  - Deleting a leaf node
  - Deleting a node with only the left child
  - Deleting a node with only the right child
  - Deleting a node with both children

Questions?