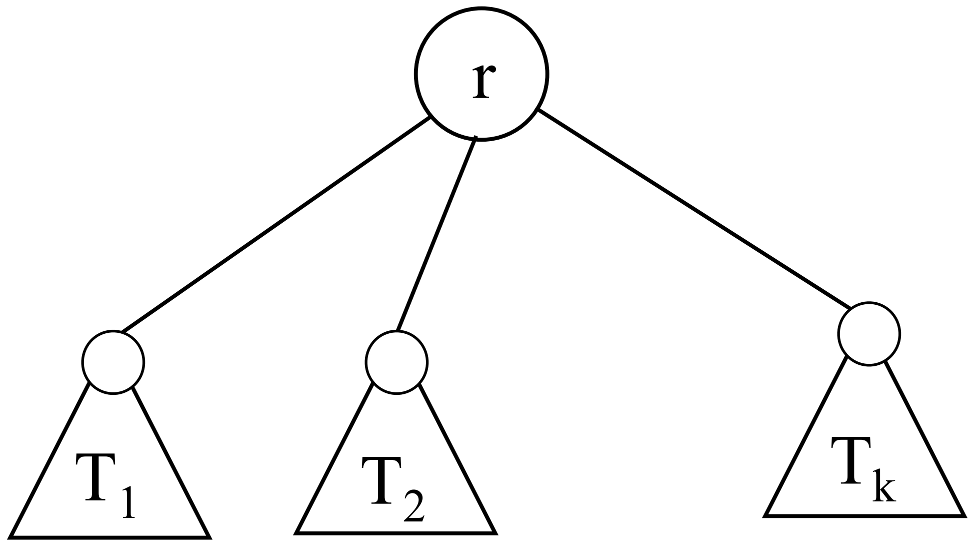
# Data Structure:

# Tree

# tree

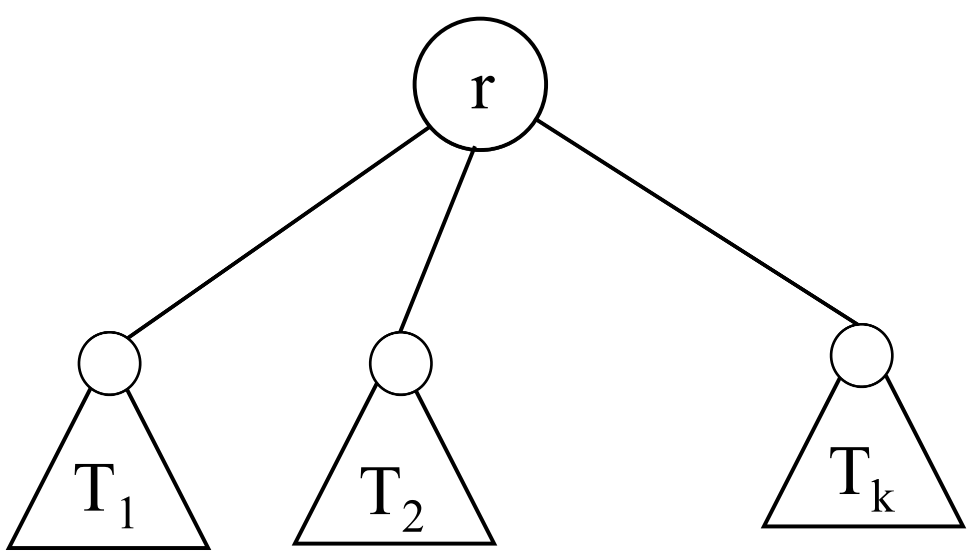
- a collection of nodes connected by edges without cycle
- by recursive definition:
  - an empty or
  - a root r and subtrees $T_1, T_2, \ldots, T_k$ (disjoint sets) each of whose roots are connected to r by an edge

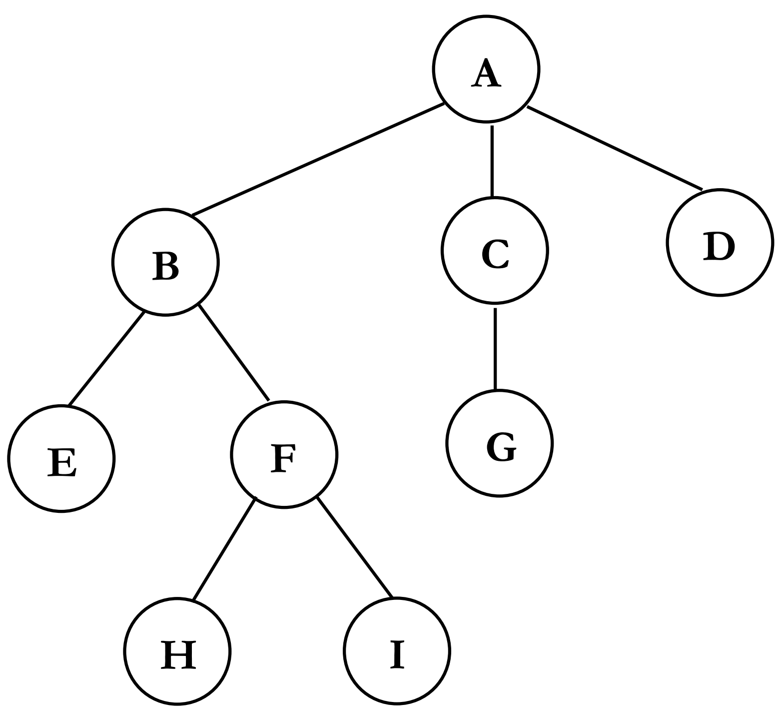



recursive definition of tree

# tree

- Each root of $T_1, T_2, \ldots, T_k$ is a *child* of *r*, and *r* is the *parent* of each root.
- The roots of the subtrees are *siblings* of one another
- If there is an order among the $T_i$'s, the tree is an *ordered tree*.
- The *degree* of a node is the number of children it has.
- The *degree* of a tree is the maximum degree of the nodes.
- A *leaf* is a node of degree 0.

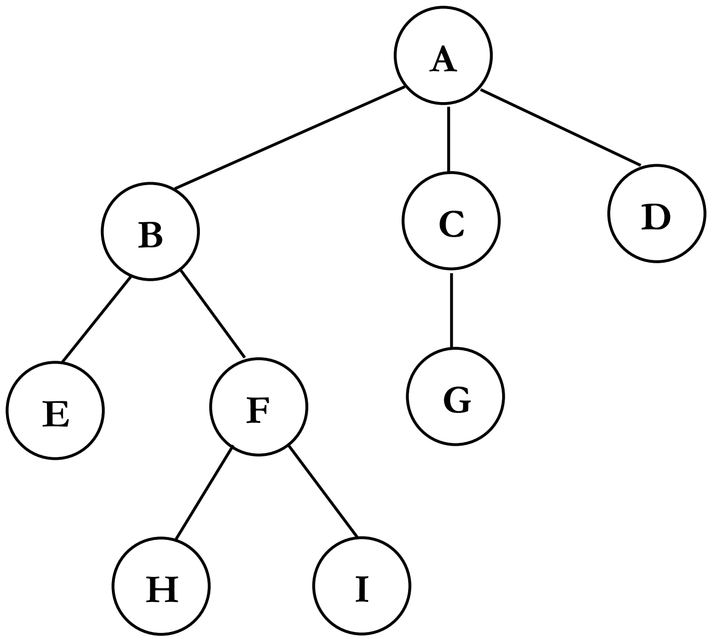



an example of tree

# tree

- *path* between two nodes is a sequence of nodes $n_1, n_2, \ldots n_k$, such that $n_i$ is a parent of $n_{i+1}$
- *length* of a path is the number of edges on the path (the path $n_1, n_2, \ldots n_k$: length k-1)
- *depth (level)* of a node is the length of the (unique) path from the root to that node (root: level 0)
- *height* of a node is the length of the longest path from that node to a leaf (leaf: height 0)
- the height of a tree is the height of the root



# representation of tree

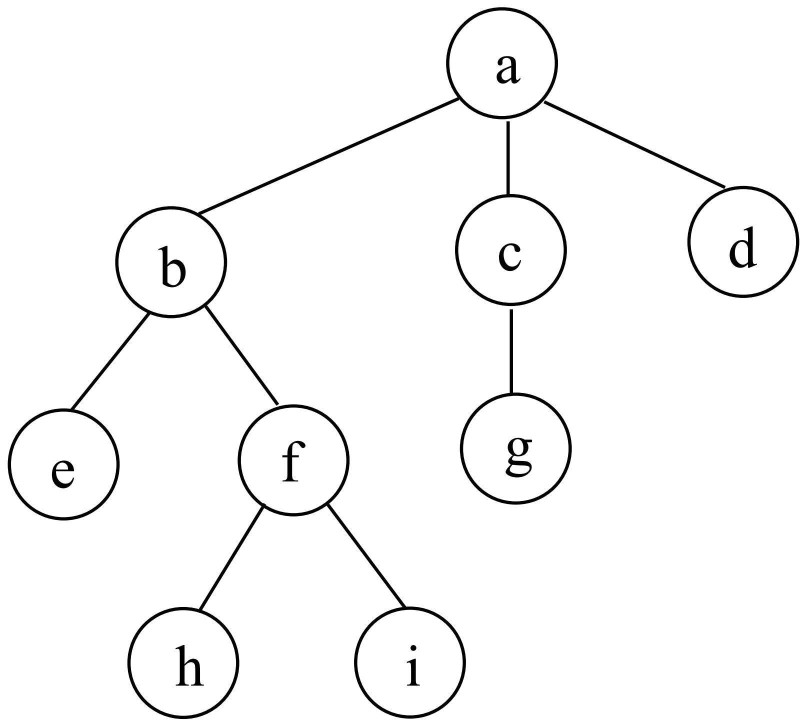
- for any node x, there exists exactly one path from the root to x?
- tree can be empty with no node?
- how many edges are in a tree with n nodes?

# representation of tree

- how can we implement a tree?

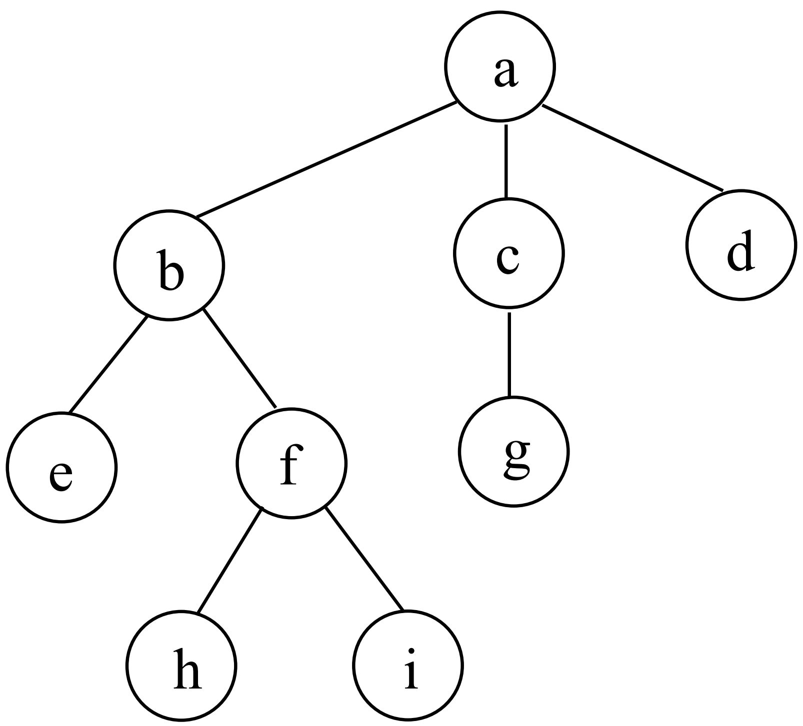
# representation of tree

- how can we implement a tree?
  - linked list?
  - can we have pointers for the children nodes?
  - can we have fixed number of pointers to represent a tree?
    - for a tree of fixed number of degree?
    - else?



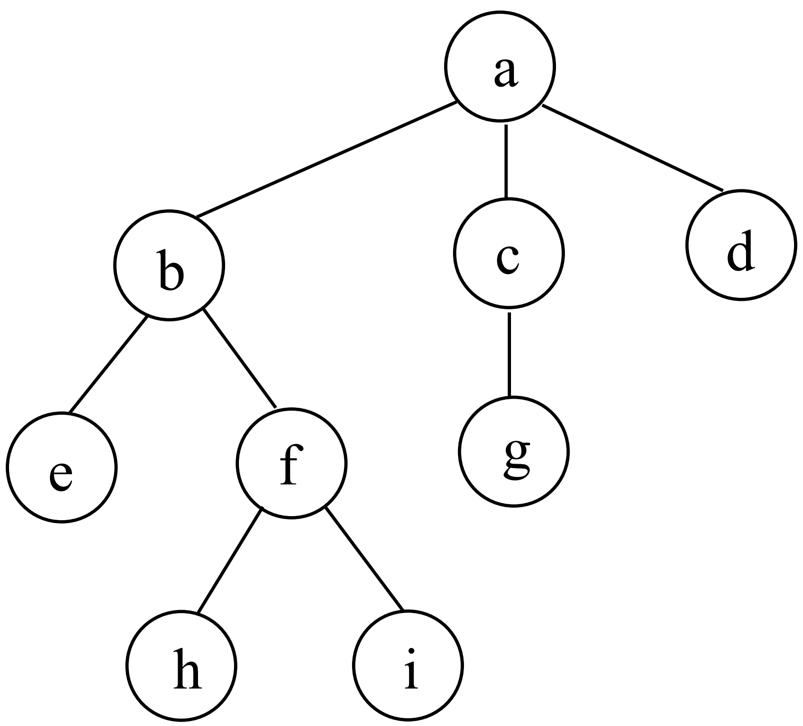

# left child-right sibling representation

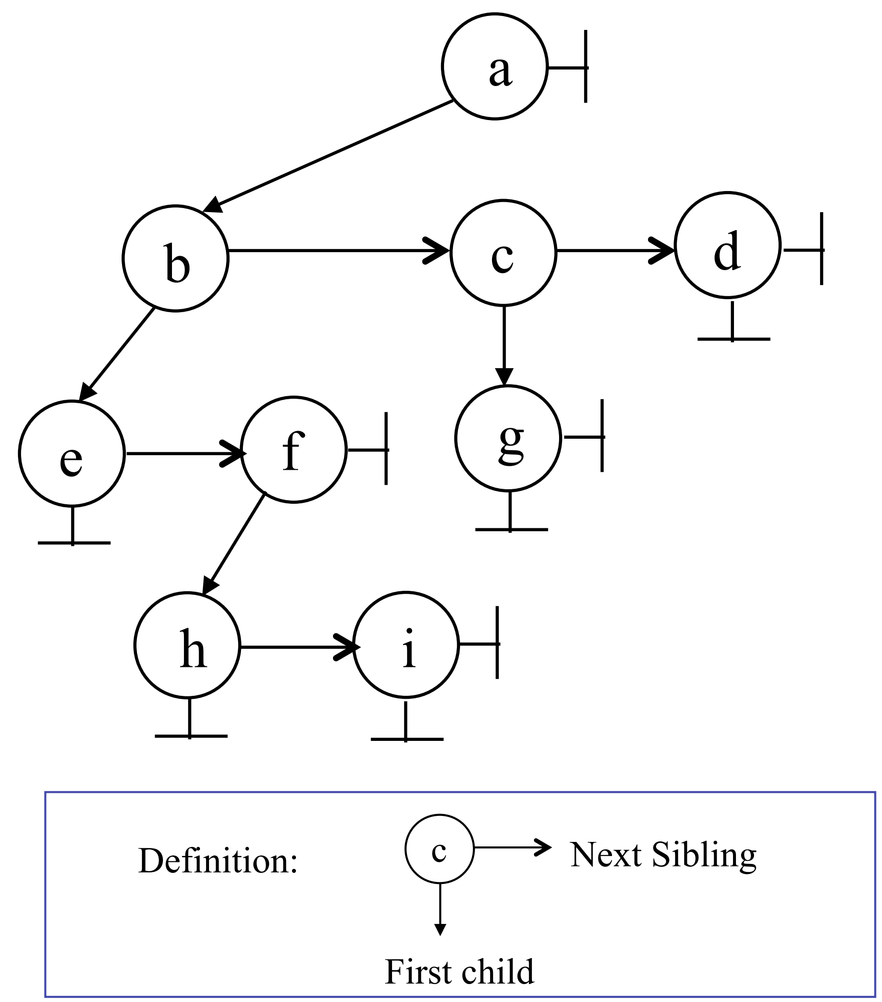
- every node has  at most one leftmost child and at most one closet right sibling



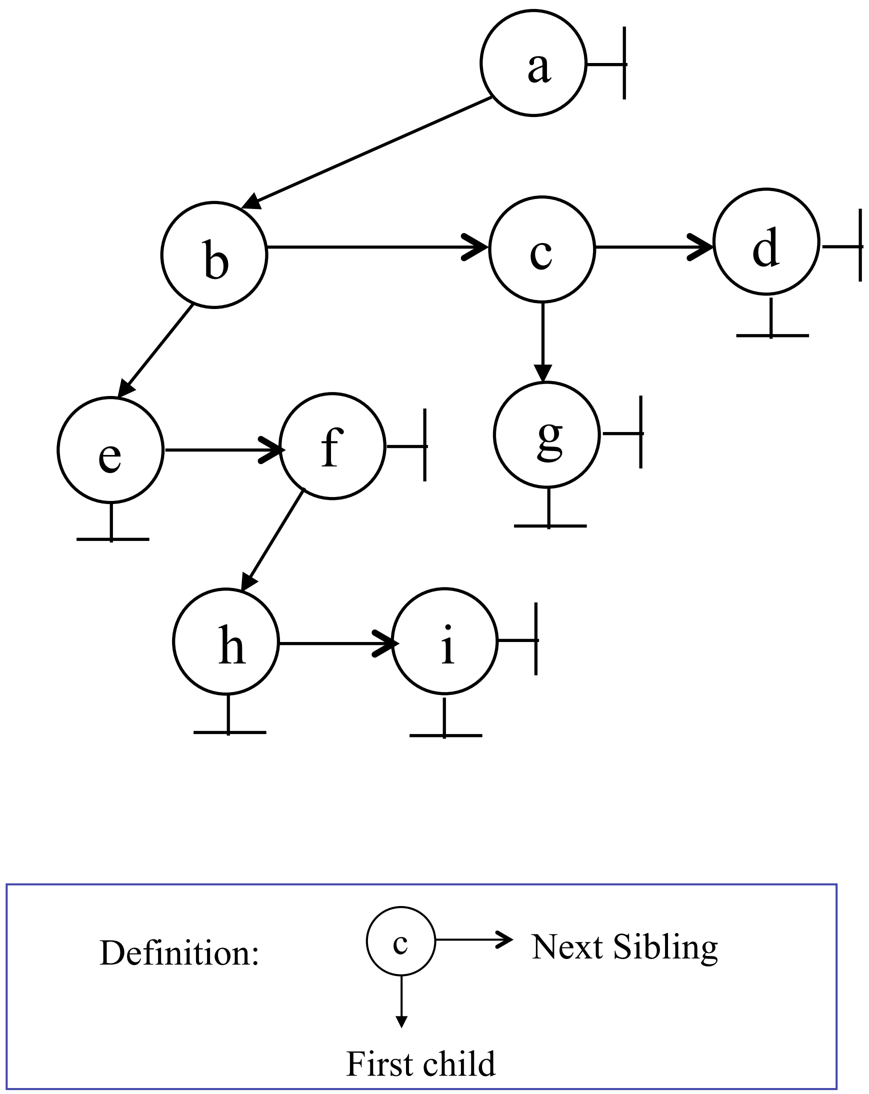
# left child-right sibling representation

- every node has  at most one leftmost child and at most one closet right sibling



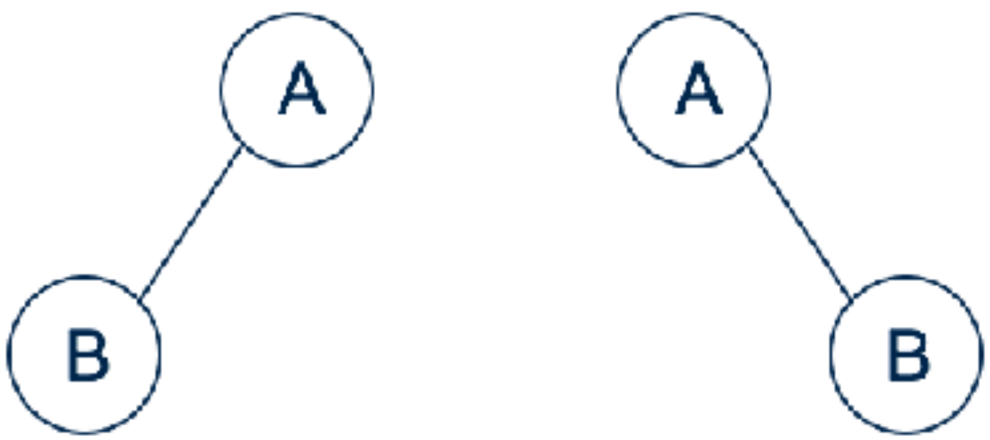
# left child- right sibling representation




```
struct TreeNode{
    ElementType Element;
    PtrToNode FirstChild;
    PtrToNode NextSibling;
    };
typedef struct TreeNode *PtrToNode;
```

# binary tree

- a finite set of nodes that is either
  - i) empty or
  - ii) a root node and two disjoint binary trees
- the tree on the left and the tree on the right are different



# binary tree

- the maximum number of nodes on level i of a binary tree is $2^i$, $i>=0$

the proof by induction

- base: for the root at level $i=0$, $2^0 = 1$
- induction hypothesis: assume that the maximum number of nodes on level $i-1 > 0$, $2^{i-1}$
- induction step: on level i,

  $2 * ($ the maximum number of nodes on level $i-1) = 2 * 2^{i-1} = 2^i$

- the maximum number of nodes in a binary tree of depth k is $2^{k+1}-1$, $k>=0$

# binary tree

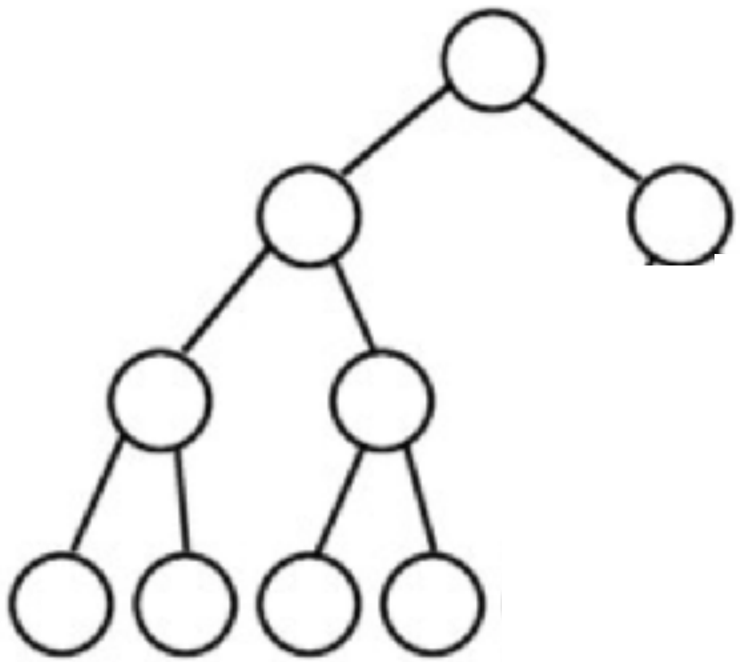
- For any nonempty binary tree T, if $n_0$ is the number of leaf nodes, and $n_2$ is the number of nodes of degree 2, then $n_0 = n_2 + 1$

$n = n_0 + n_1 + n_2$ , $n_i$ is the number of nodes with i degree
n is the number of nodes in the tree

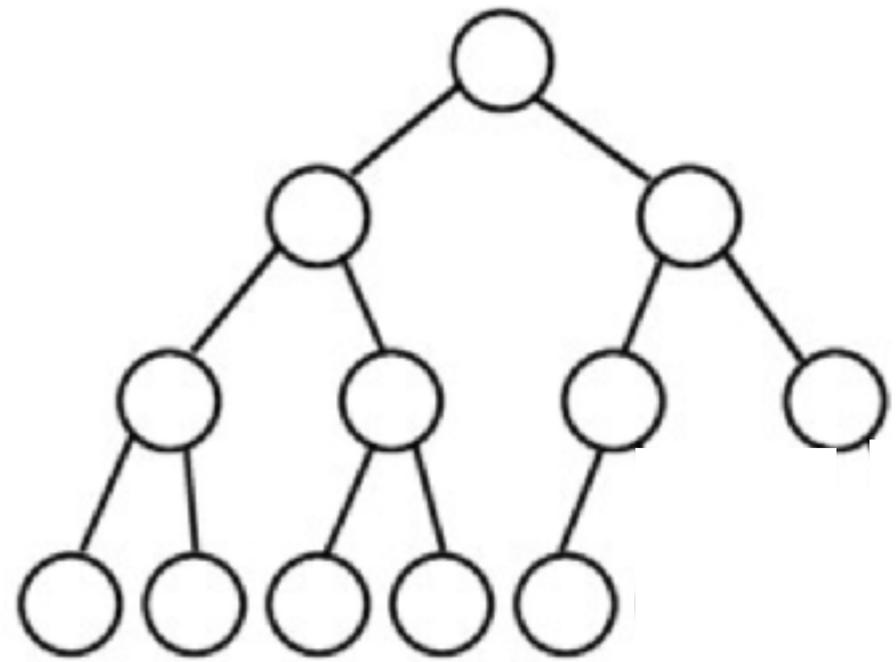
$n = B + 1 = n_1 + 2n_2 + 1$, B is the number of branches (edge)

# binary tree

- full binary tree is a binary tree in which every node has 0 or 2 children
- complete binary tree is a binary tree in which every level, except the last, is completely filled and the last level has all its nodes to the left side



- full binary tree



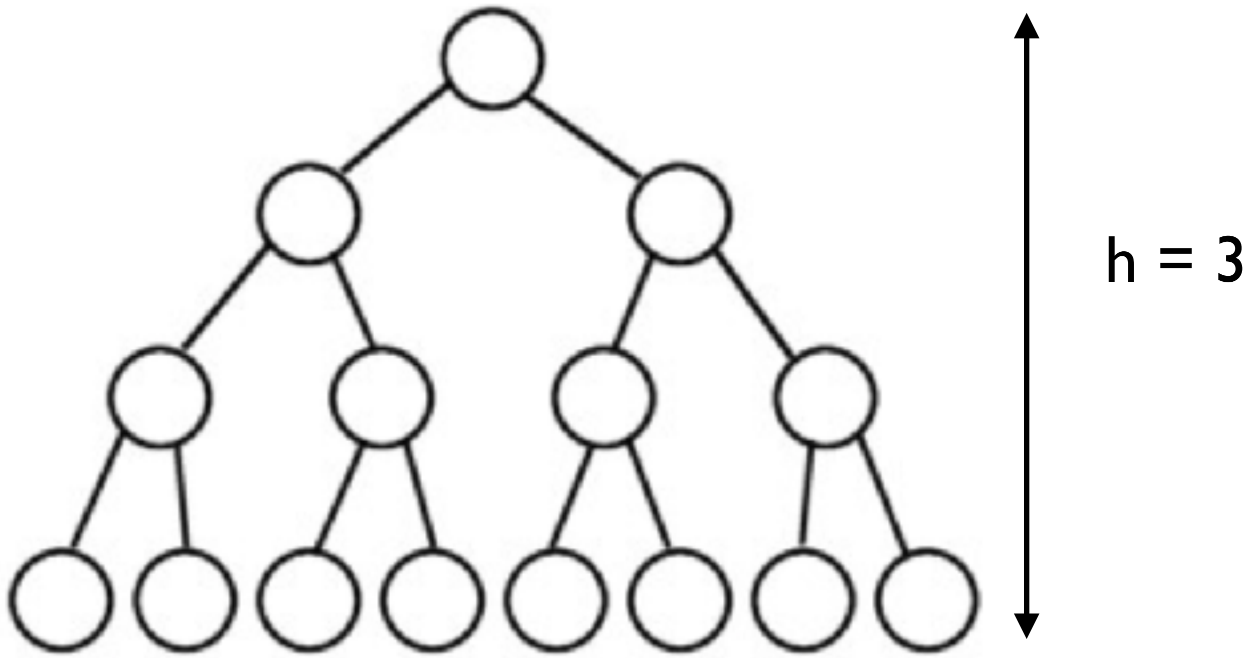
- complete binary tree

# binary tree

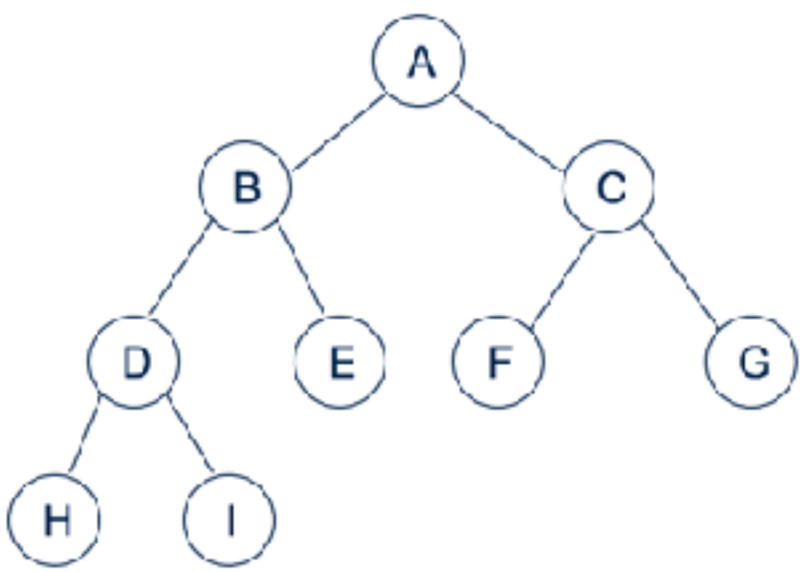
- perfect binary tree of height h is a binary tree of height h having $2^{h+1} - 1$ nodes, (h >=0)

- the max number of nodes in the complete binary tree (height h) is $2^{h+1} - 1$

$$2^0 + 2^1 + \ldots + 2^h = (2^{h+1} - 1)/(2 - 1) = 2^{h+1} - 1$$



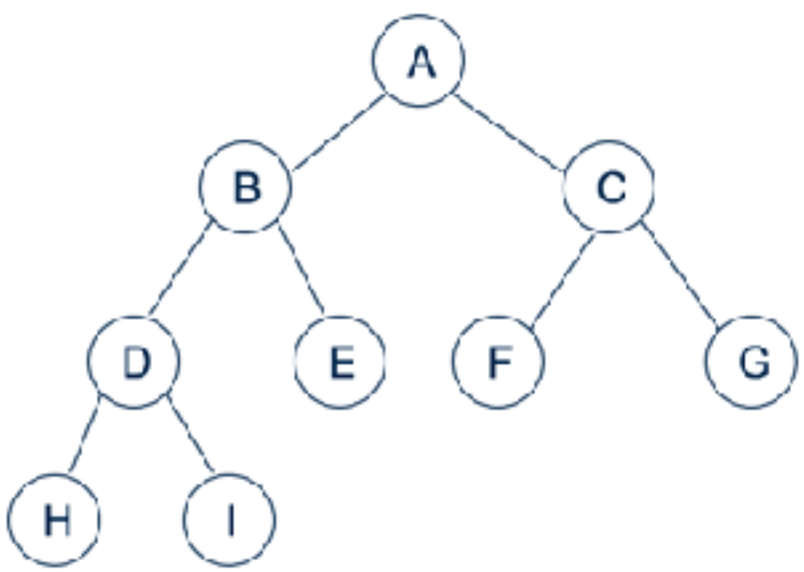
# binary tree: array representation




| | |
|---|---|
| [1] | A |
| [2] | B |
| [3] | C |
| [4] | D |
| [5] | E |
| [6] | F |
| [7] | G |
| [8] | H |
| [9] | I |

# binary tree: array representation

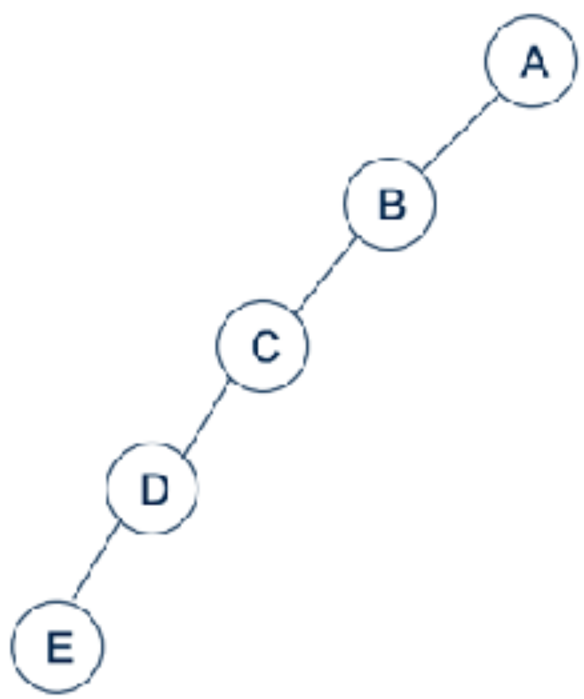
- if a complete binary tree with n nodes (i is the index) is represented sequentially,
  - leftChild(i) is at 2i   for 2i <=n
  - rightChild(i) is at 2i + 1   for 2i + 1 <= n
  - parent(i) is at $\lfloor i/2 \rfloor$   for i >1




| | |
|---|---|
| [1] | A |
| [2] | B |
| [3] | C |
| [4] | D |
| [5] | E |
| [6] | F |
| [7] | G |
| [8] | H |
| [9] | I |

# binary tree: array representation

- if a complete binary tree with n nodes (i is the index) is represented sequentially,
  - leftChild(i) is at 2i for 2i <=n
  - rightChild(i) is at 2i + 1 for 2i + 1 <= n
  - parent(i) is at $\lfloor i/2 \rfloor$ for i >1

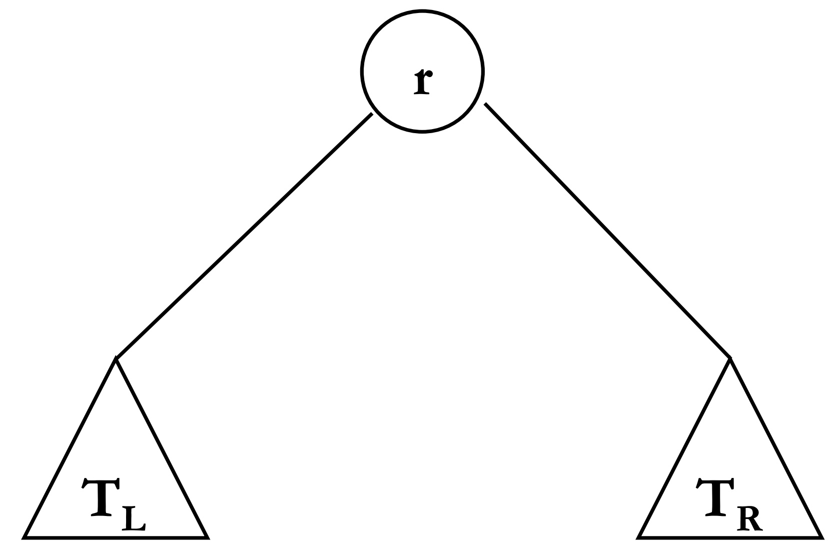



| | |
|---|---|
| [1] | A |
| [2] | B |
| [3] | - |
| [4] | C |
| [5] | - |
| [6] | - |
| [7] | - |
| [8] | D |
| [9] | - |
| . | . |
| . | . |
| . | . |
| [16] | E |

# binary tree: linked list representation

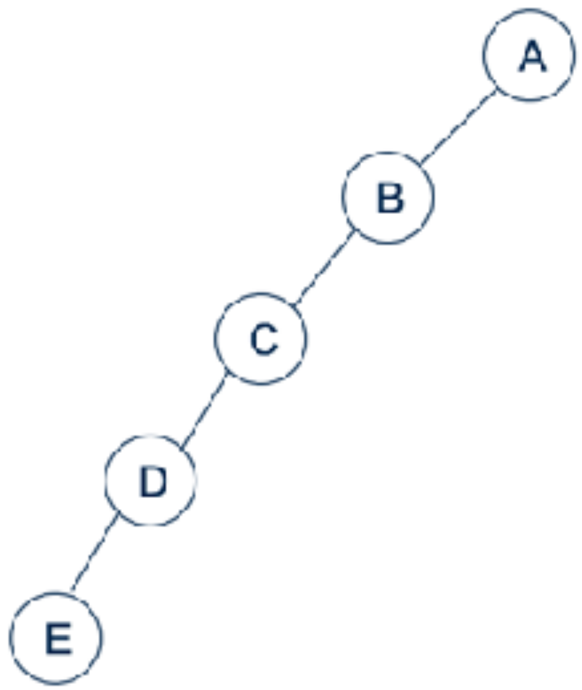
- a tree in which each node has no more than 2 children (left subtree and right subtree)

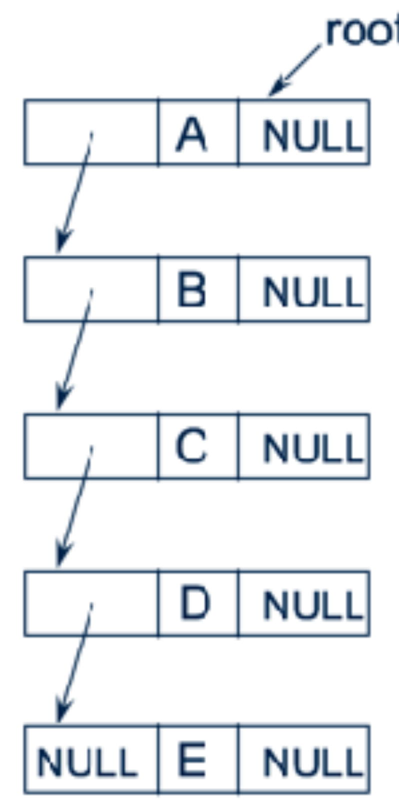
```
struct TreeNode
{
      ElementType  Element;
      Tree         Left;
      Tree         Right;
};

typedef struct TreeNode* PtrToNode;
typedef struct PtrToNode Tree;
```

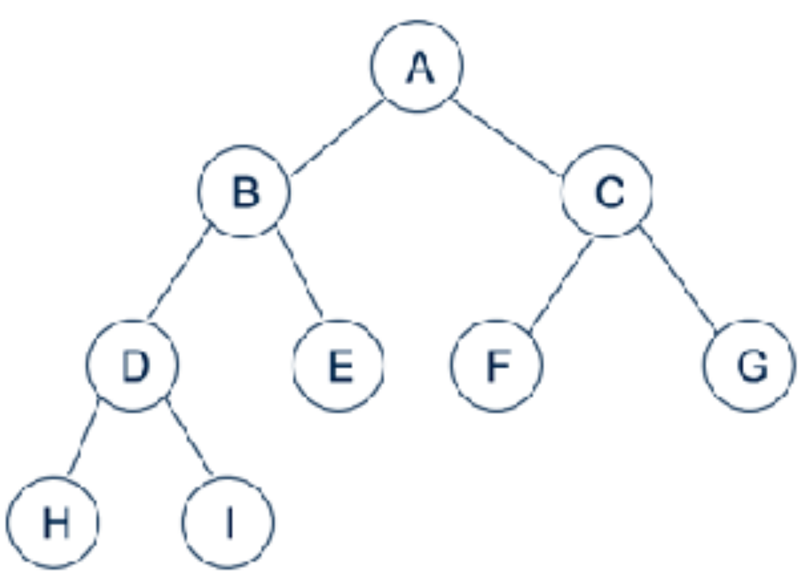


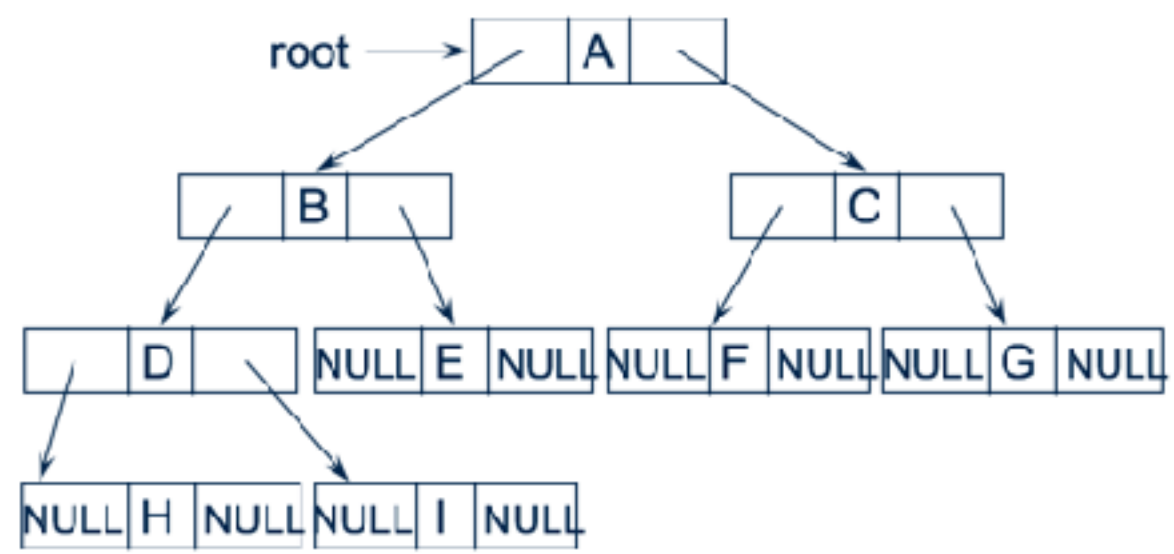

| | | |
|---|---|---|
| Left | Element | Right |

# binary tree: linked list representation



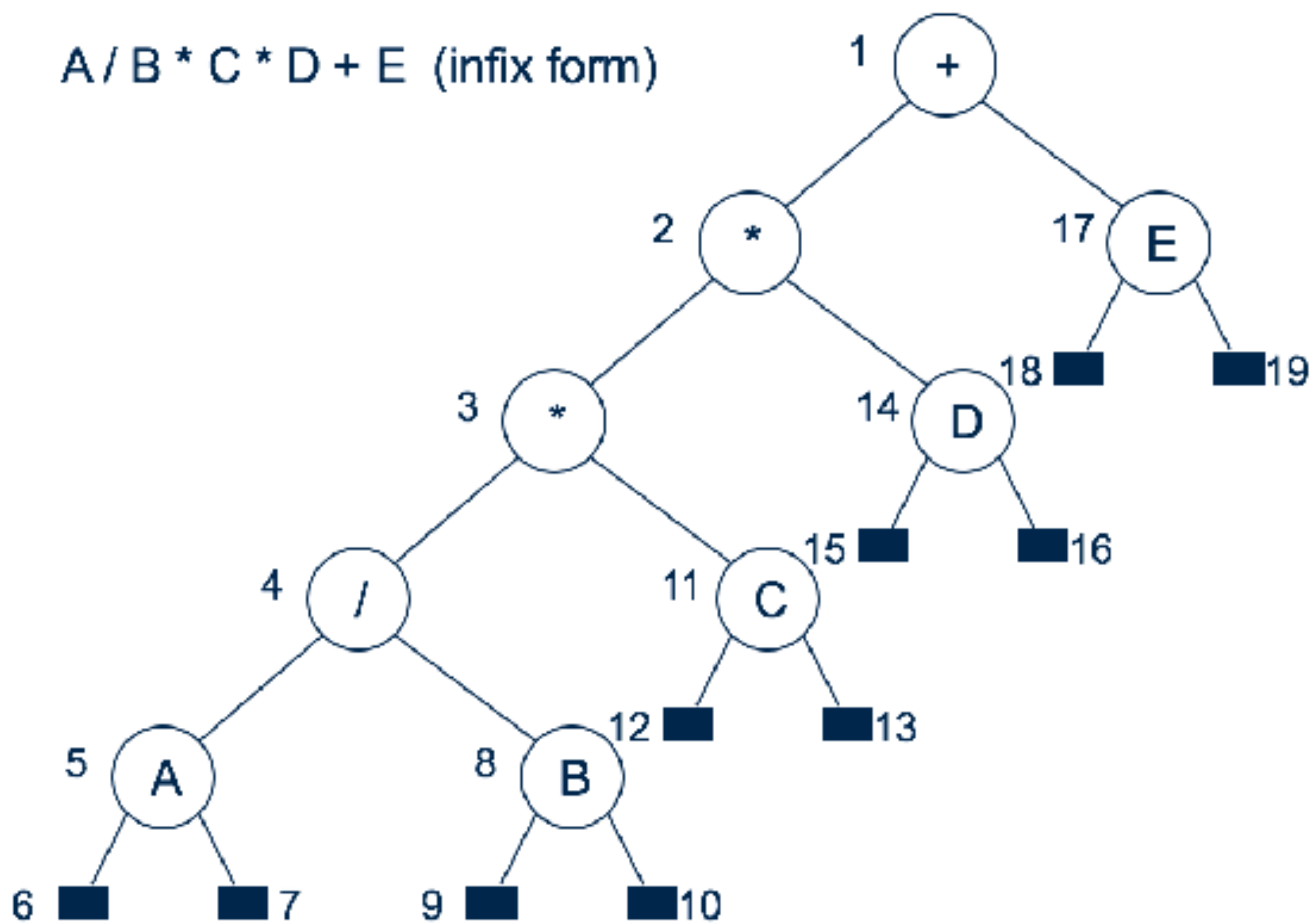



# application of binary tree

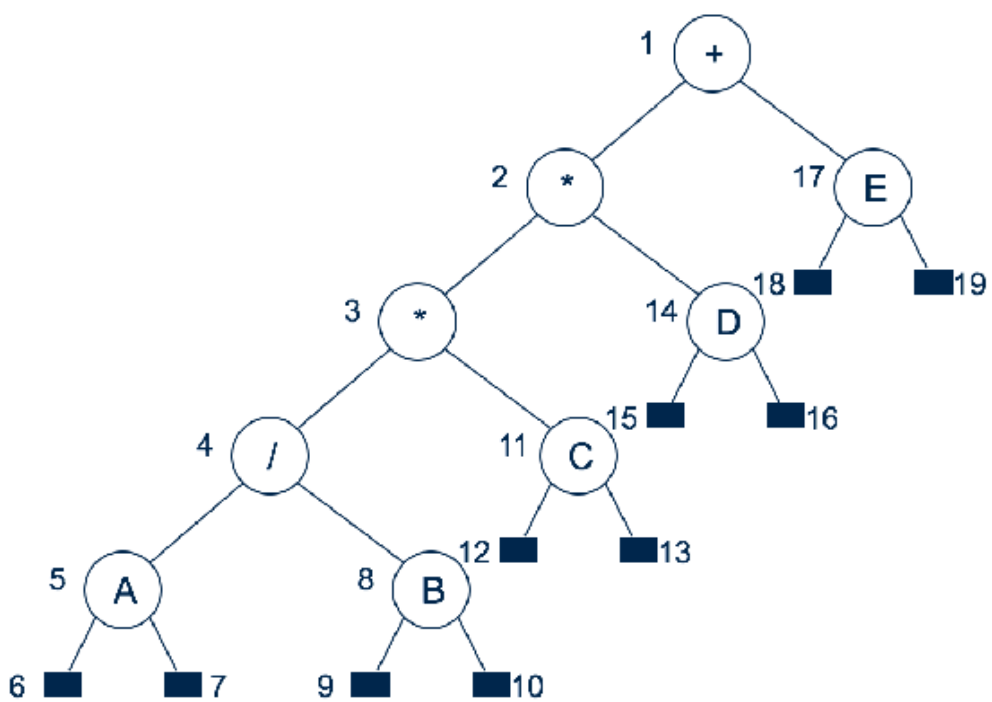
- *Expression Tree*: intermediate representation for expressions used by the compiler



# tree traversal

- inorder traversal

```
void inorder(Tree ptr) {
    if(ptr) {
        inorder(ptr->left_child);
        printf("%d",   ptr->data);
        inorder(ptr->right_child);
    }
}
```




| call of inorder | value in root | action | call of inorder | value in root | action |
|---|---|---|---|---|---|
| 1 | + | | 11 | C | |
| 2 | * | | 12 | NULL | |
| 3 | * | | 11 | C | printf |
| 4 | / | | 13 | NULL | |
| 5 | A | | 2 | * | printf |
| 6 | NULL | | 14 | D | |
| 5 | A | printf | 15 | NULL | |
| 7 | NULL | | 14 | D | printf |
| 4 | / | printf | 16 | NULL | |
| 8 | B | | 1 | + | printf |
| 9 | NULL | | 17 | E | |
| 8 | B | printf | 18 | NULL | |
| 10 | NULL | | 17 | E | printf |
| 3 | * | printf | 19 | NULL | |

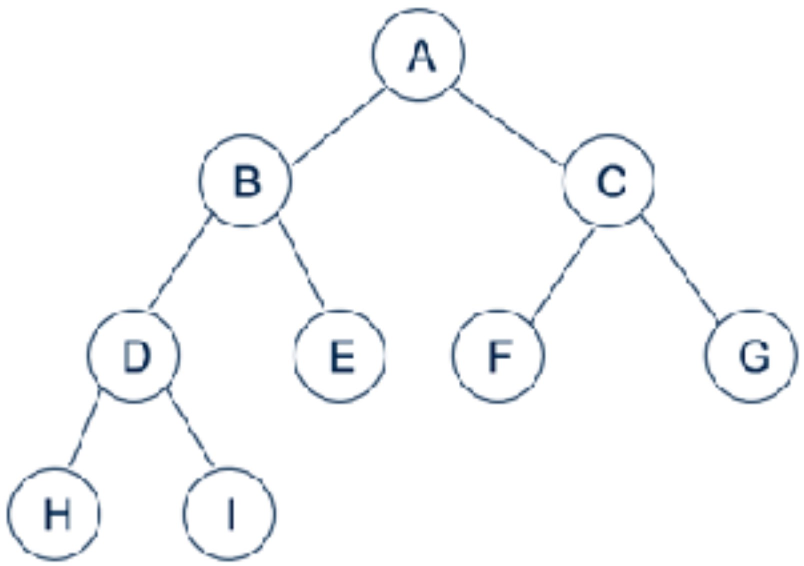
# tree traversal

```
void preorder(Tree ptr) {
    if(ptr) {
        printf(“%d”,   ptr->data);
        preorder(ptr->left_child);
        preorder(ptr->right_child);
    }
}
```

```
void postorder(Tree ptr) {
    if(ptr) {
        postorder(ptr->left_child);
        postorder(ptr->right_child);
        printf(“%d”,   ptr->data);
    }
}
```

# tree traversal

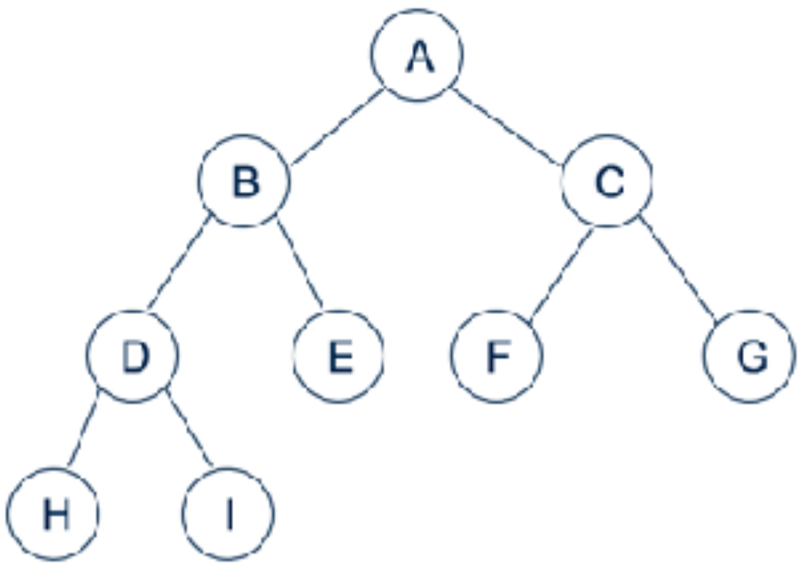
- iterative in-order traversal using stack




```
void iterInorder (Tree node)   {

    int top = -1
    Tree stack[MAX_SIZE];
    for (; ;)   {
        for (; node; node = node -> leftChild)
            push(node);

        node = pop();              // pop parent
        if (!node)    break;
        printf("%d", node -> data);
        node = node -> rightChild;
    }
}
```

# tree traversal

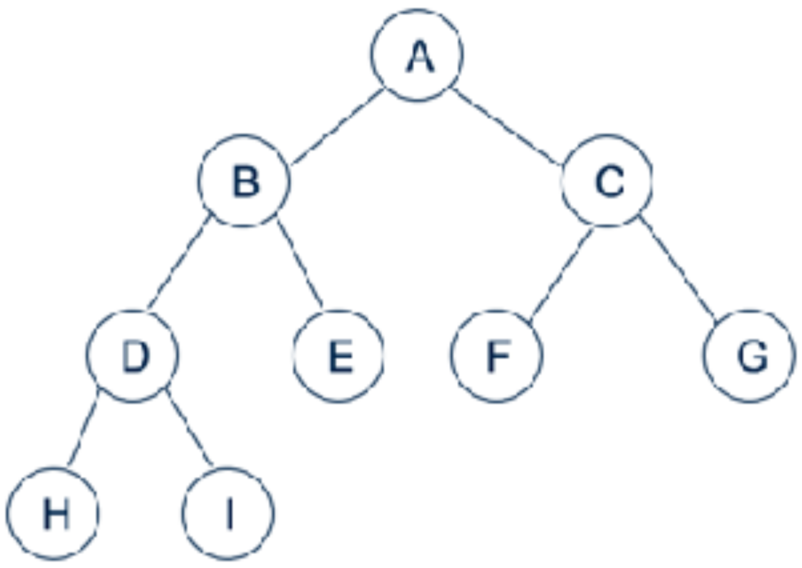
- level-order traversal



# tree traversal

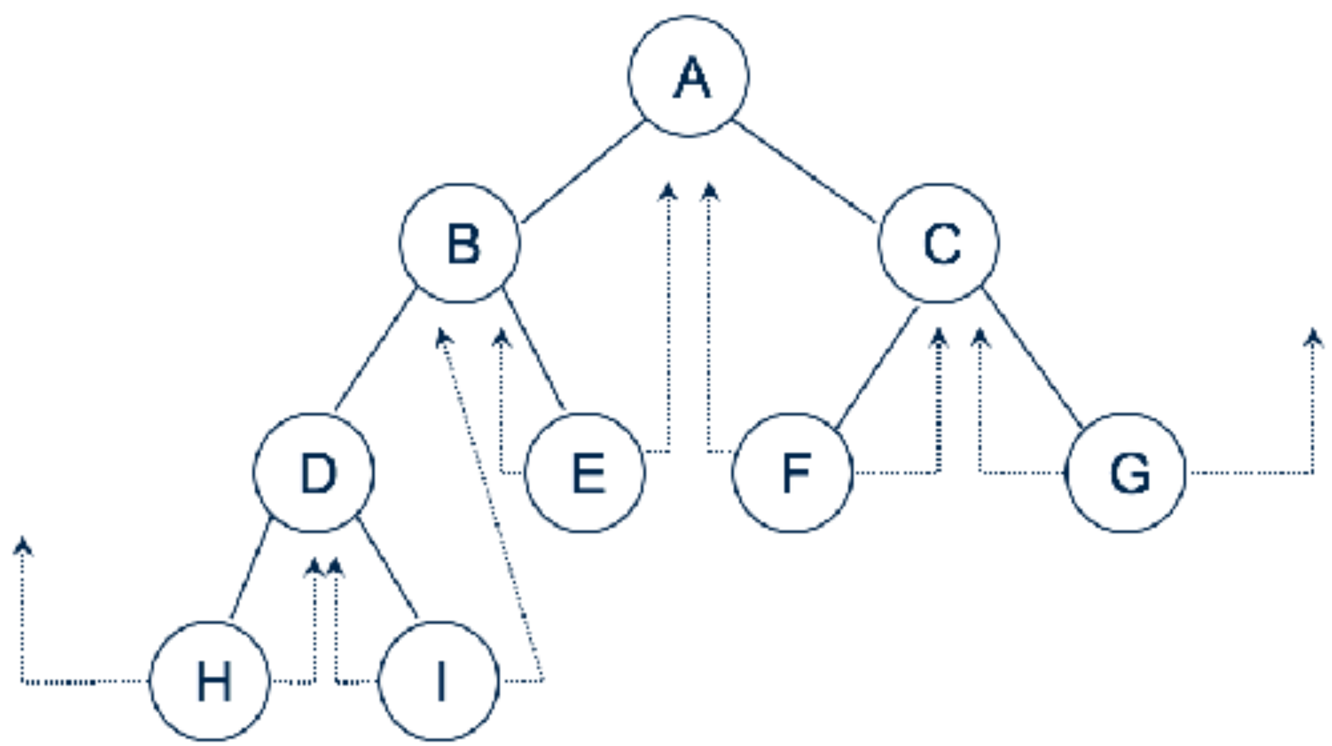
- level-order traversal

```
void levelOrder (Tree ptr)   {
        int front = rear = 0;
        Tree queue[MAX];
        if (! node)     return;
        addq(ptr);
        for (; ; )  {
                ptr = deleteq();
                if (ptr)  {
                        printf("%d", ptr->data);
                        if (ptr -> leftChild)
                                addq(ptr -> leftChild);
                        if (ptr -> rightChild)
                                addq(ptr -> rightChild);
                }
                else break;
        }
}
```



# threaded binary trees

- there are n+1 null links out of 2n total links
- replace the null links by pointers, called threads to other nodes in the tree
  - if ptr -> leftChild is null, replace the null with a pointer to the node that would be visited before ptr in an in-order traversal
  - if ptr -> rightChild is null, replace the null with a pointer to the node that would be visited after ptr in an in-order traversal



# threaded binary trees

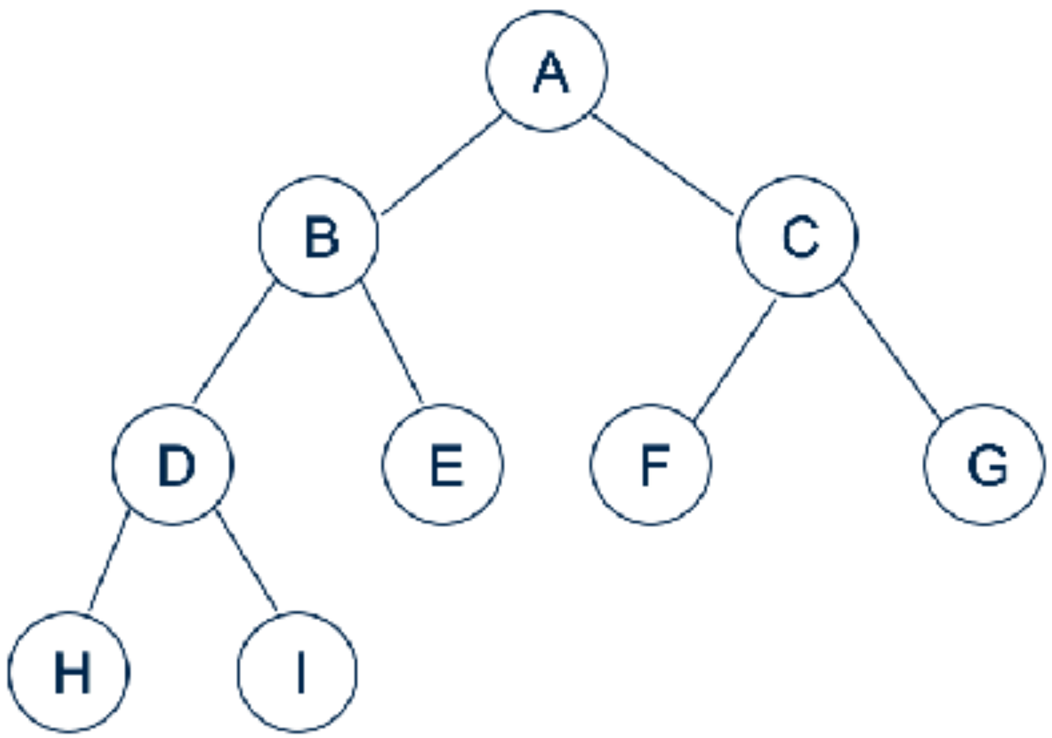
- How to distinguish actual pointers and threads?

  →add two additional fields to the node structure

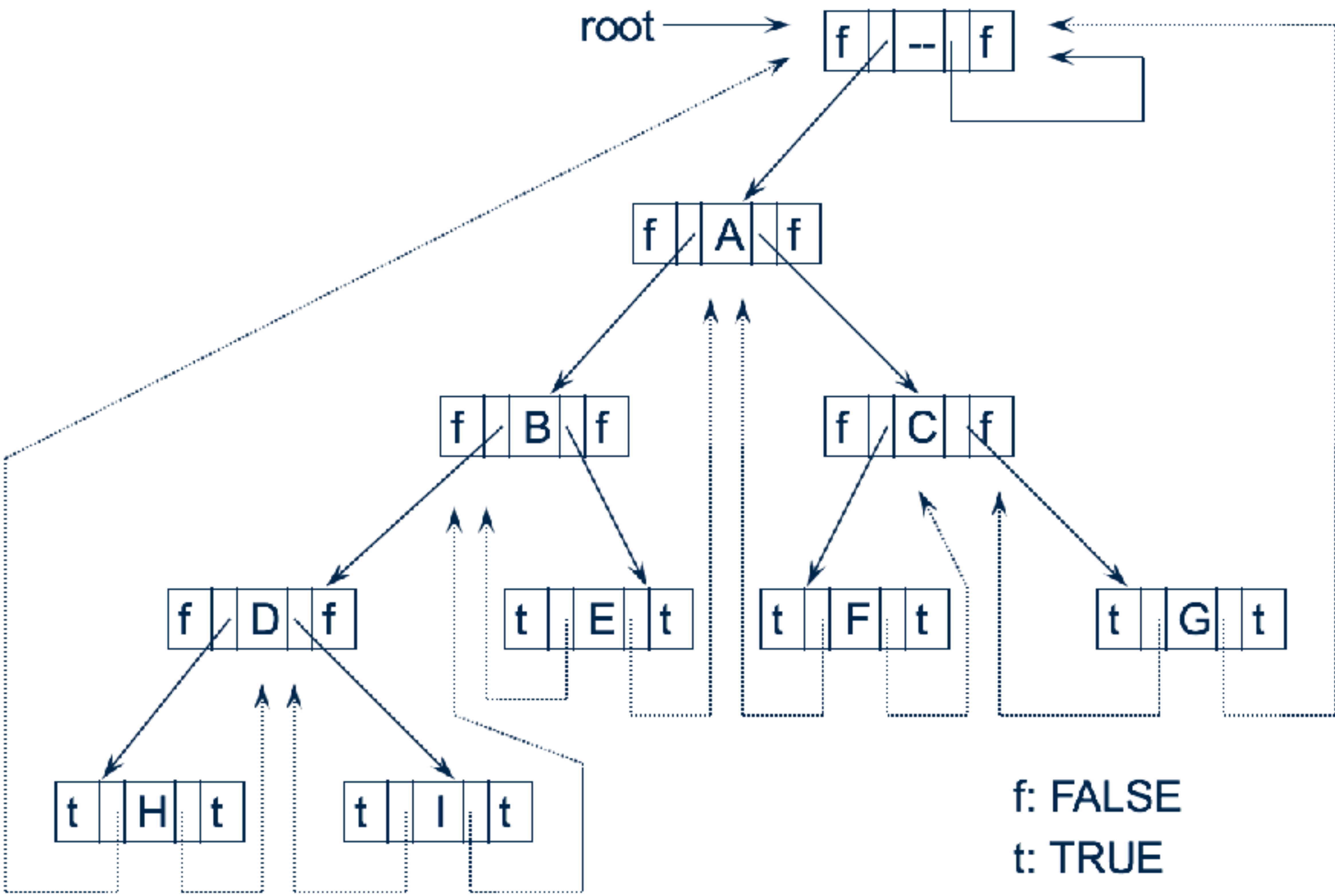
  - if ptr->left_thread = true, ptr->left_child contains thread
  - if ptr->left_thread = false, ptr->left_child contains a pointer to the left child

```
typedef struct threaded_tree *threaded_ptr;
typedef struct threaded_tree {
        short int left_thread;
        threaded_ptr left_child;
        char data;
        threaded_ptr right_child;
        short int right_thread;
};
```

# threaded binary trees



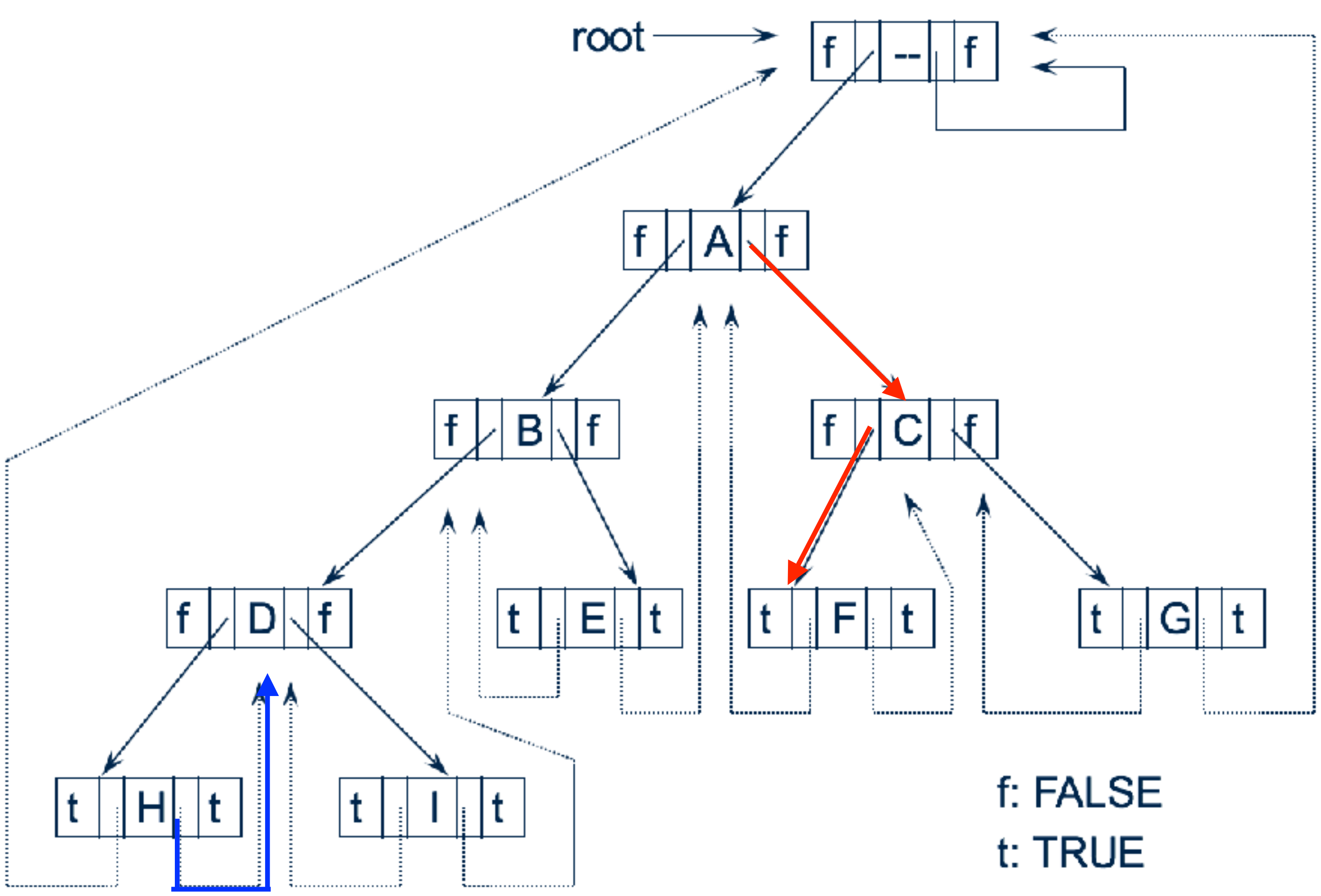
# threaded binary trees



# in-order traversal of threaded binary trees

- find the in-order successor of ptr without using stack
  - if ptr -> right_thread = TRUE, ptr -> right_child
  - otherwise follow a path of left_child links from the right_child of ptr until we reach a node with left_thread = TRUE

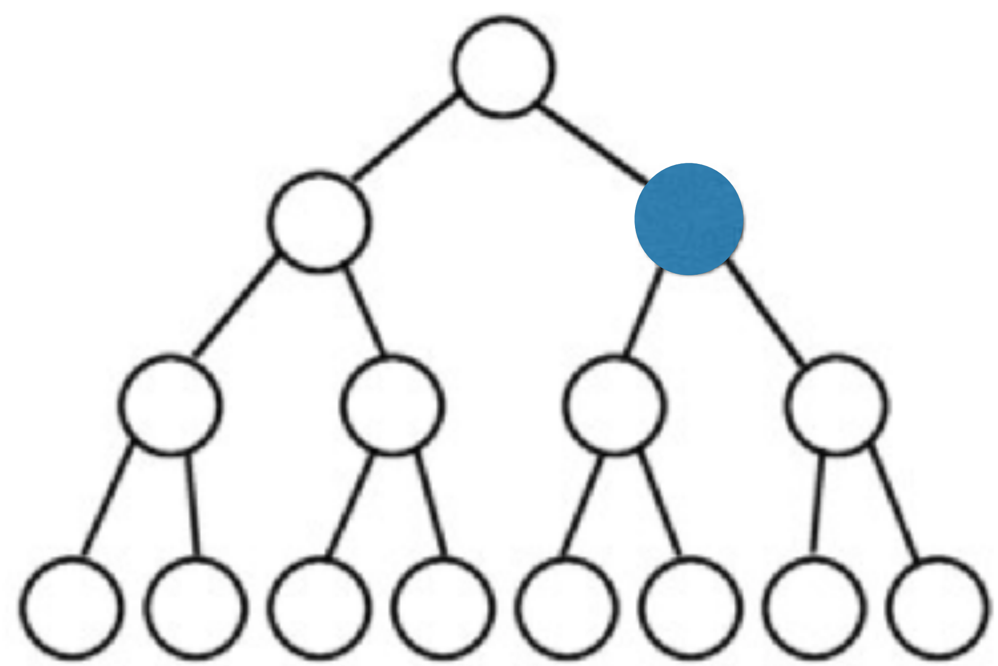
```
threaded_ptr insucc(threaded_ptr tree) {
    threaded_ptr temp;
    temp = tree->right_child;
    if (!tree->right_thread)
        while (!temp->left_thread)
            temp = temp->left_child;
    return temp;
}
```



# in-order traversal of threaded binary trees

- find the in-order successor of ptr without using stack
  - if ptr -> right_thread = TRUE, ptr -> right_child
  - otherwise follow a path of left_child links from the right_child of ptr until we reach a node with left_thread = TRUE

```
threaded_ptr insucc(threaded_ptr tree) {
    threaded_ptr temp;
    temp = tree->right_child;
    if (!tree->right_thread)
        while (!temp->left_thread)
            temp = temp->left_child;
    return temp;
}
```

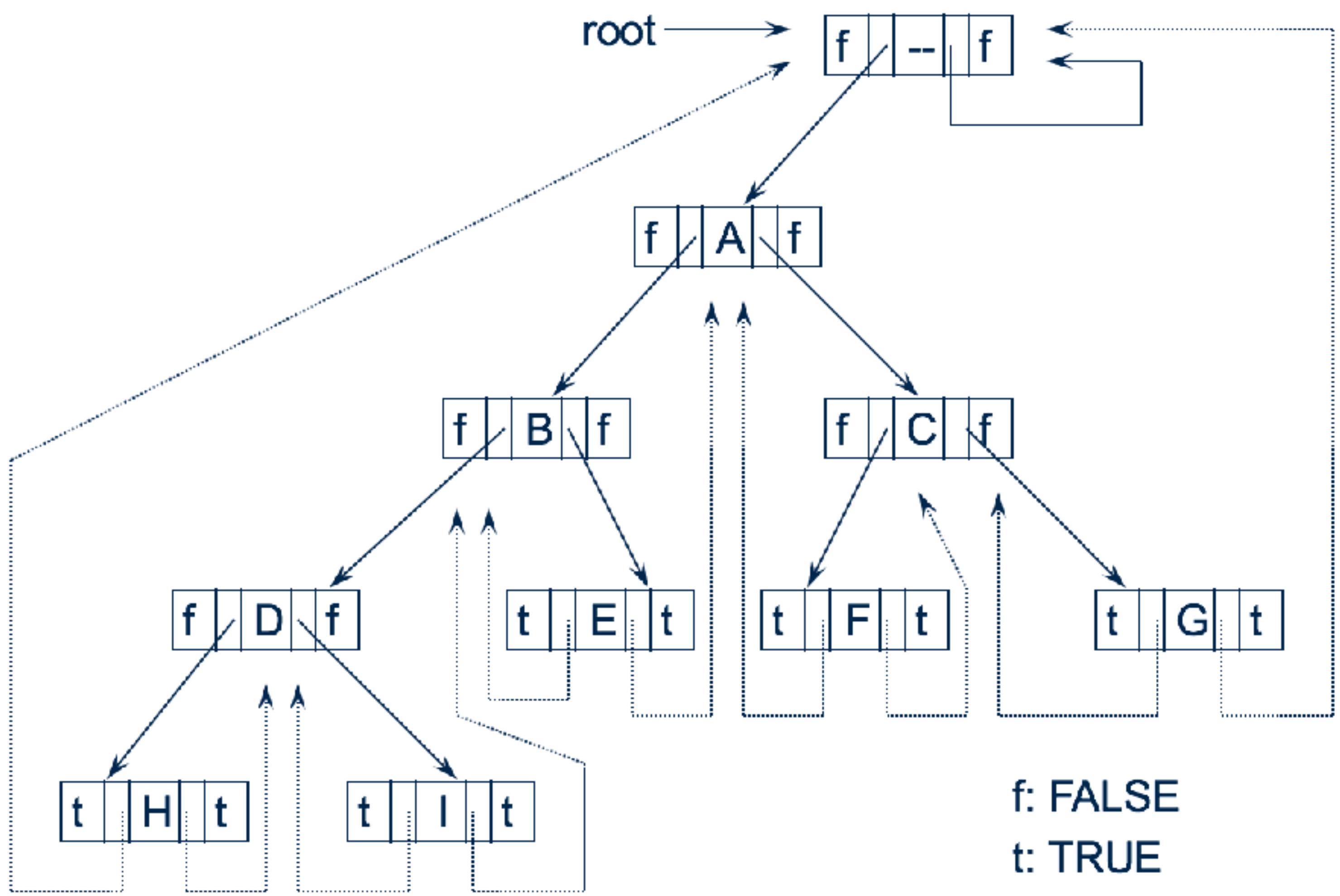


Which node will be returned if blue node is passed into the function insucc?

# in-order traversal of threaded binary trees

- find the in-order successor of ptr without using stack
  - if ptr -> right_thread = TRUE, ptr -> right_child
  - otherwise follow a path of left_child links from the right_child of ptr until we reach a node with left_thread = TRUE

```
threaded_ptr insucc(threaded_ptr tree) {
      threaded_ptr temp;
      temp = tree->right_child;
      if (!tree->right_thread)
            while (!temp->left_thread)
                  temp = temp->left_child;
      return temp;
}
```




Which node will be returned if root node is passed into the function insucc?

# in-order traversal of threaded binary trees

```
void tinorder(threaded_ptr tree) {
        threaded_ptr temp = tree;
        for (;;) {
                temp = insucc(temp);
                if (temp == tree) break;
                printf("%3c", temp->data);
        }
}
```