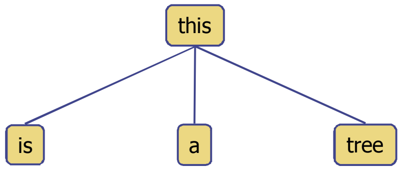
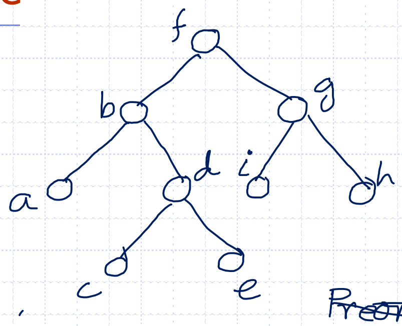
CSE 2215:
Data Structures and Algorithms-I

Trees and Tree Traversals



Preorder, Inorder, Postorder Traversals: Example

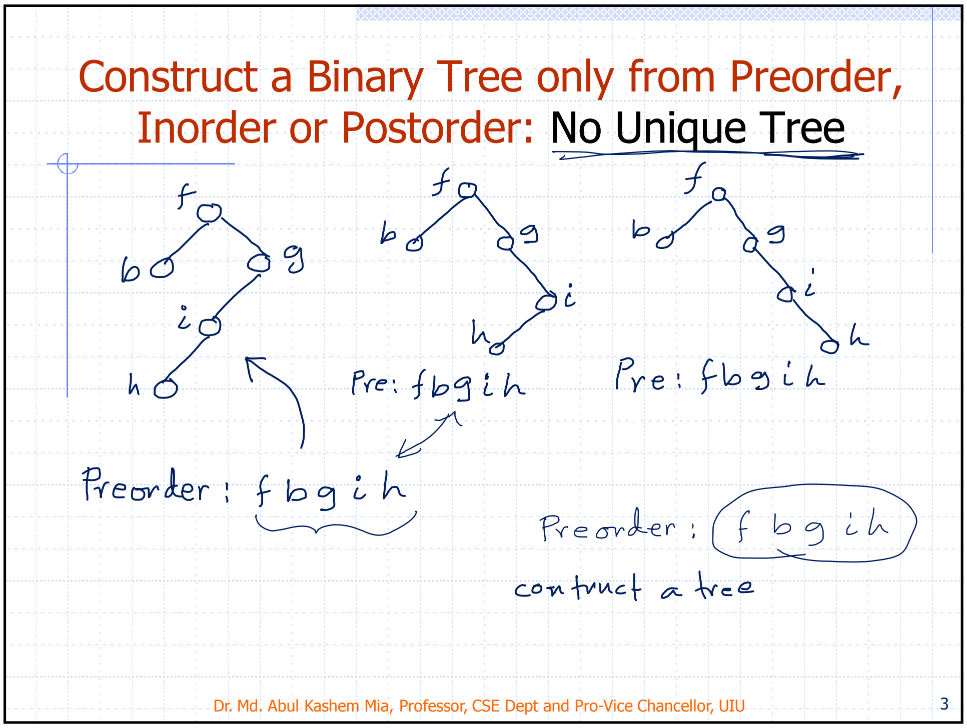


~~Preorder~~:

Preorder (root, L, R) : f b a d c e g i h

Inorder (L, root, R): a b c d e f i g h

Postorder (L, R, root): a c e d b i h g f



Construct a Binary Tree only from Preorder, Inorder or Postorder: No Unique Tree

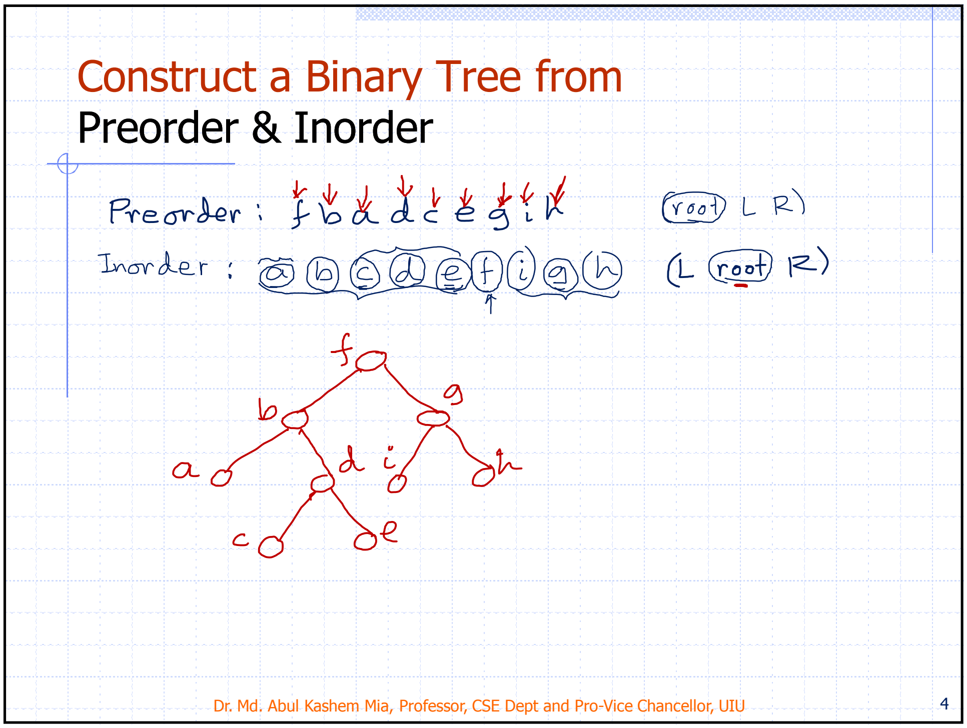
Pre: fbgih

Pre: fbgih

Preorder : f b g i h

Preorder : f b g i h

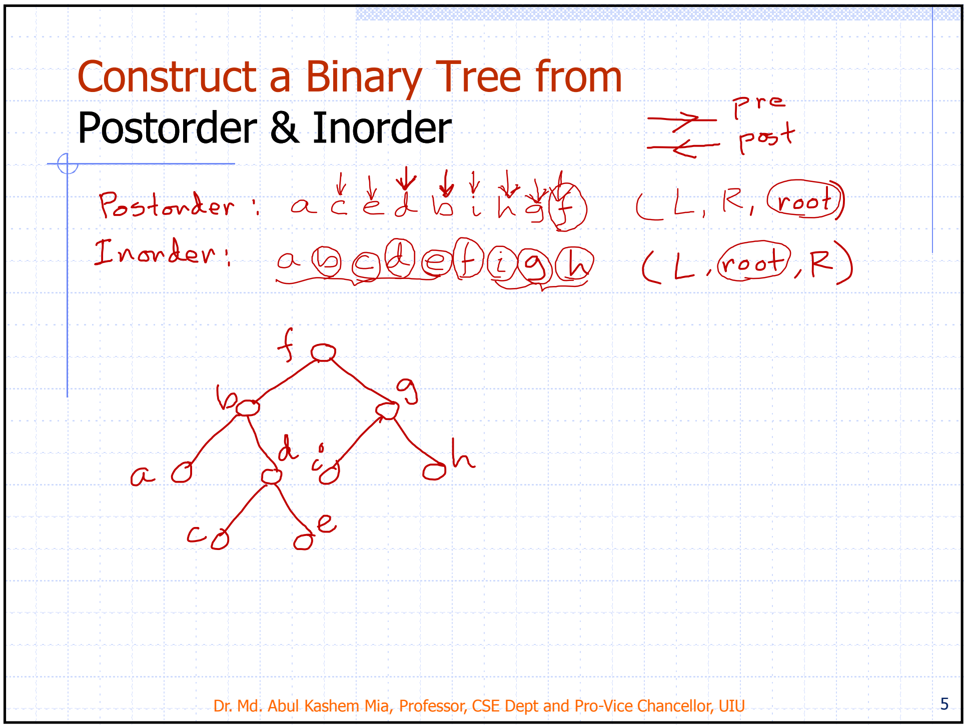
contruct a tree



Construct a Binary Tree from Preorder & Inorder

Preorder : f b a d c e g i h (root L R)

Inorder : a b c d e f i g h (L root R)



Construct a Binary Tree from Postorder & Inorder

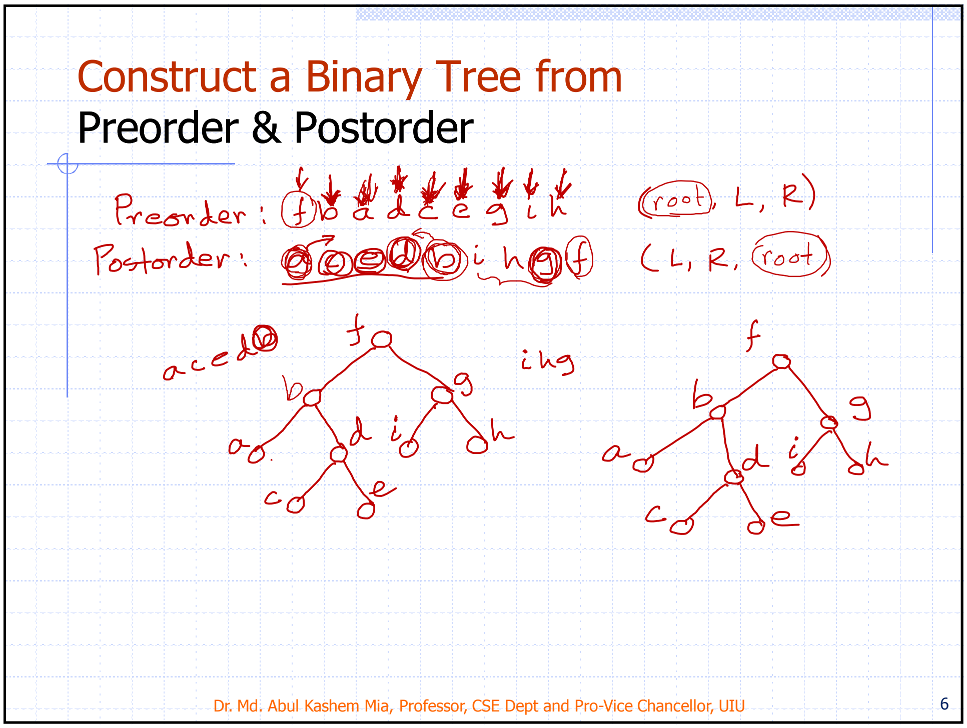
→ pre

← post

Postorder : a c e d b i h g f   (L, R, root)

Inorder : a b c d e f i g h   (L, root, R)

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU 5



Construct a Binary Tree from Preorder & Postorder

Preorder : f b a d c e g i h   (root, L, R)

Postorder : a c e d b i h g f   (L, R, root)

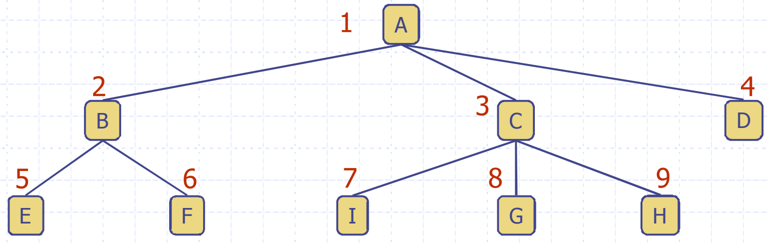
acedb

ihg

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU 6

Level Order Traversal

- In a level order traversal, every node on a level is visited before going to a lower level



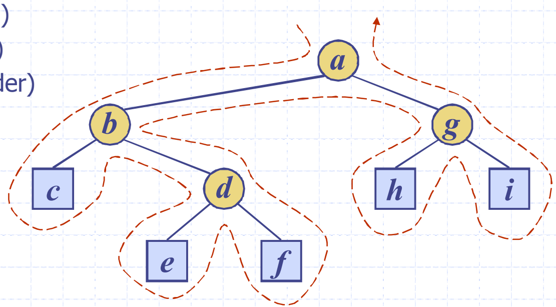
Level order: ***A B C D E F I G H***

Euler Tour Traversal

Graph

Oiler

- Generic traversal of a binary tree
- Includes a special cases the preorder, postorder and inorder traversals
- Walk around the tree and visit each node three times:
  - on the left (preorder)
  - from below (inorder)
  - on the right (postorder)



Preorder: *a b c d e f g h i*

Inorder: *c b e d f a h g i*

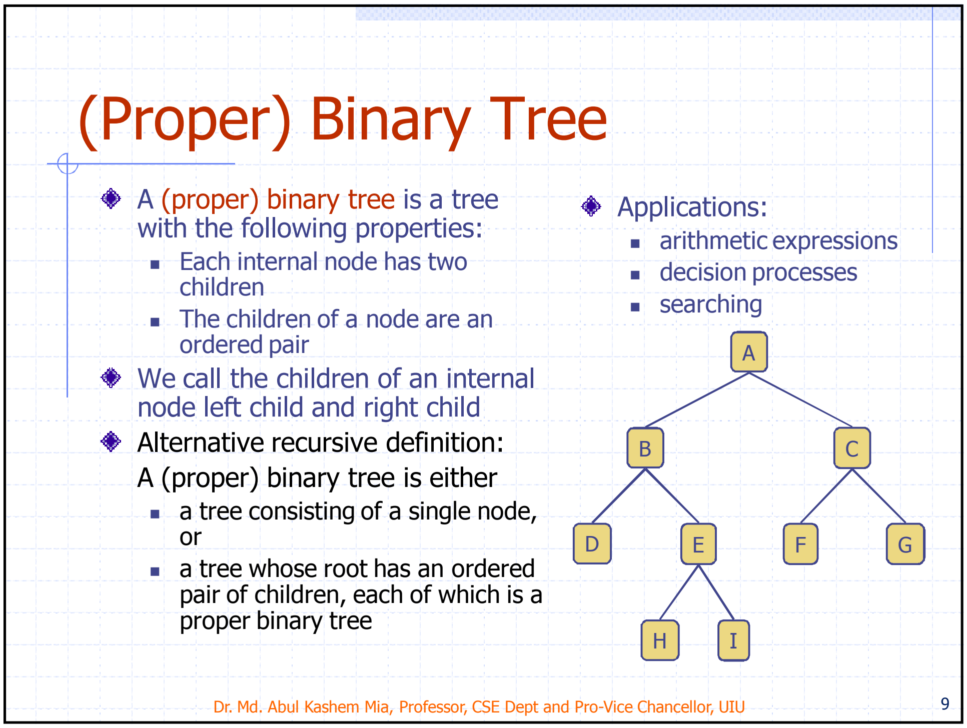
Postorder: *c e f d b h i g a*

Euler Tour: ***a b c b d e d f d b a g h g i g a***

Pre: a b c d e f g h i

In: c b e d f a h g i

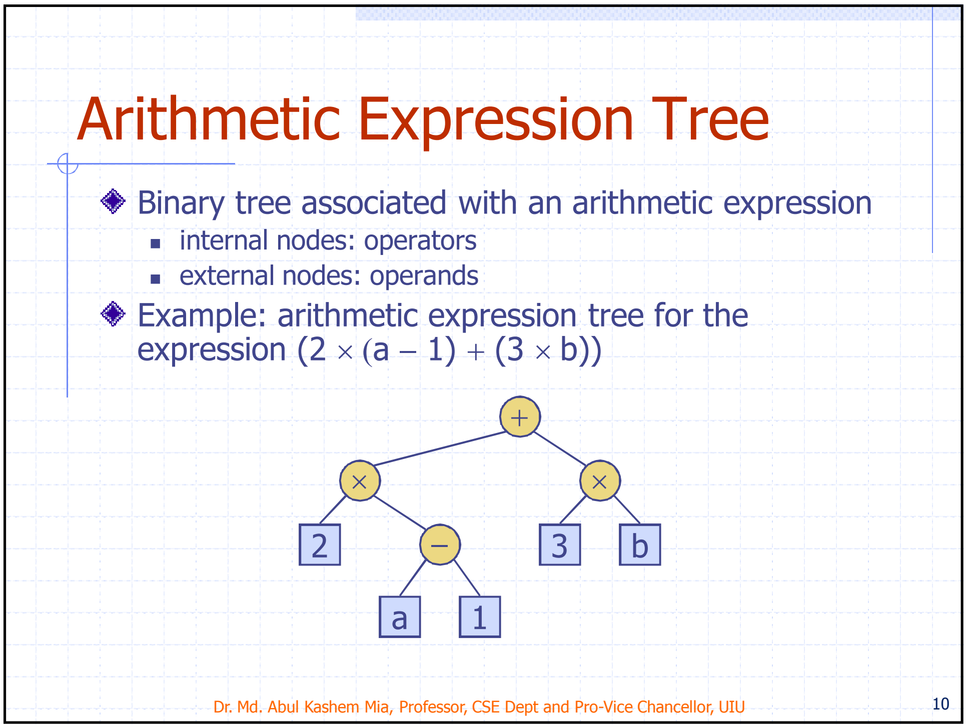
Post: c e f d b h i g a



(Proper) Binary Tree

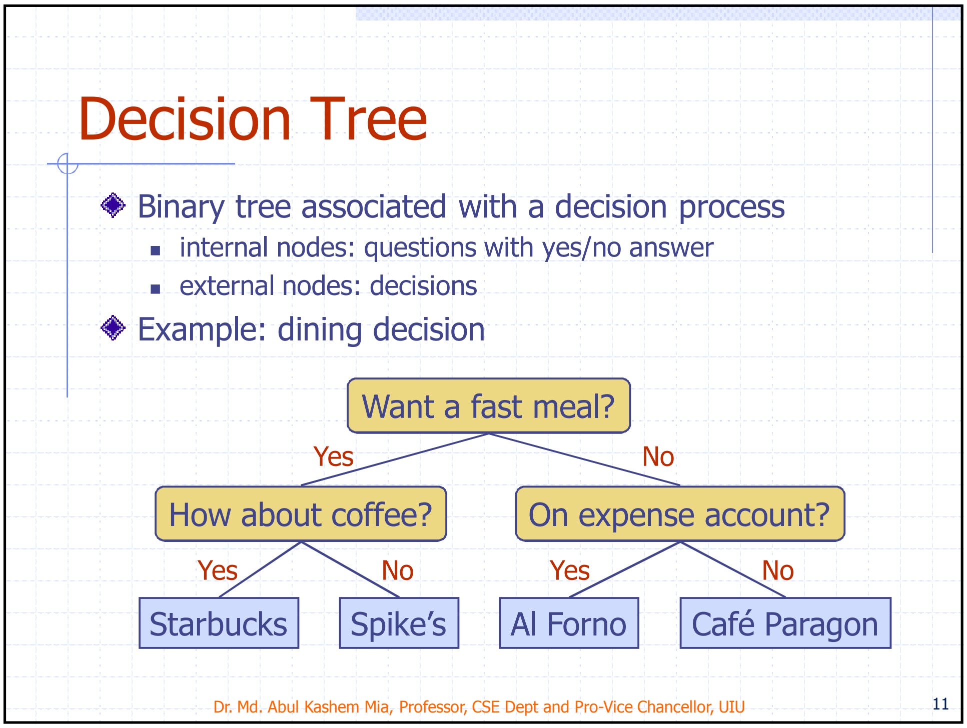
- A (proper) binary tree is a tree with the following properties:
  - Each internal node has two children
  - The children of a node are an ordered pair
- We call the children of an internal node left child and right child
- Alternative recursive definition:

  A (proper) binary tree is either
  - a tree consisting of a single node, or
  - a tree whose root has an ordered pair of children, each of which is a proper binary tree
- Applications:
  - arithmetic expressions
  - decision processes
  - searching



Arithmetic Expression Tree

- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$



Decision Tree

- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision

Want a fast meal?

Yes

No

How about coffee?

On expense account?

Yes

No

Yes

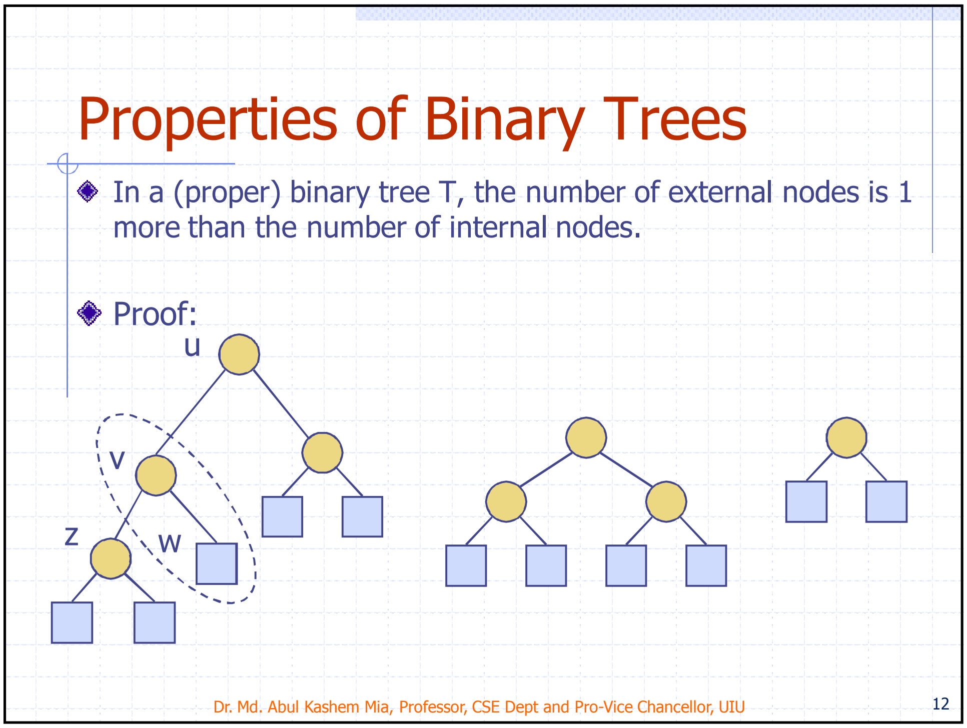
No

Starbucks

Spike's

Al Forno

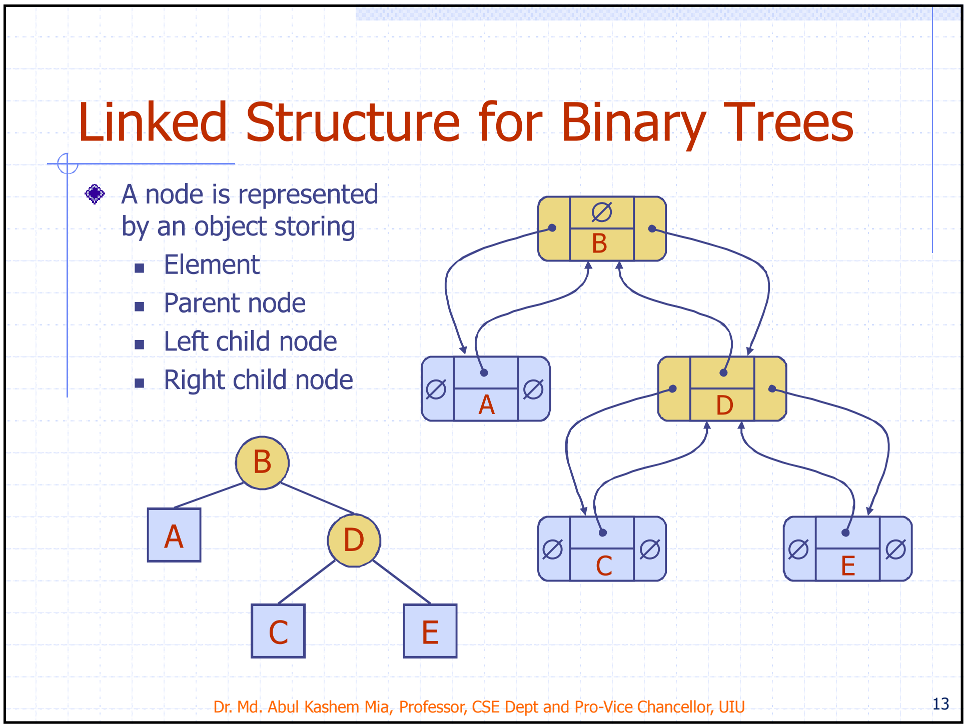
Café Paragon



Properties of Binary Trees

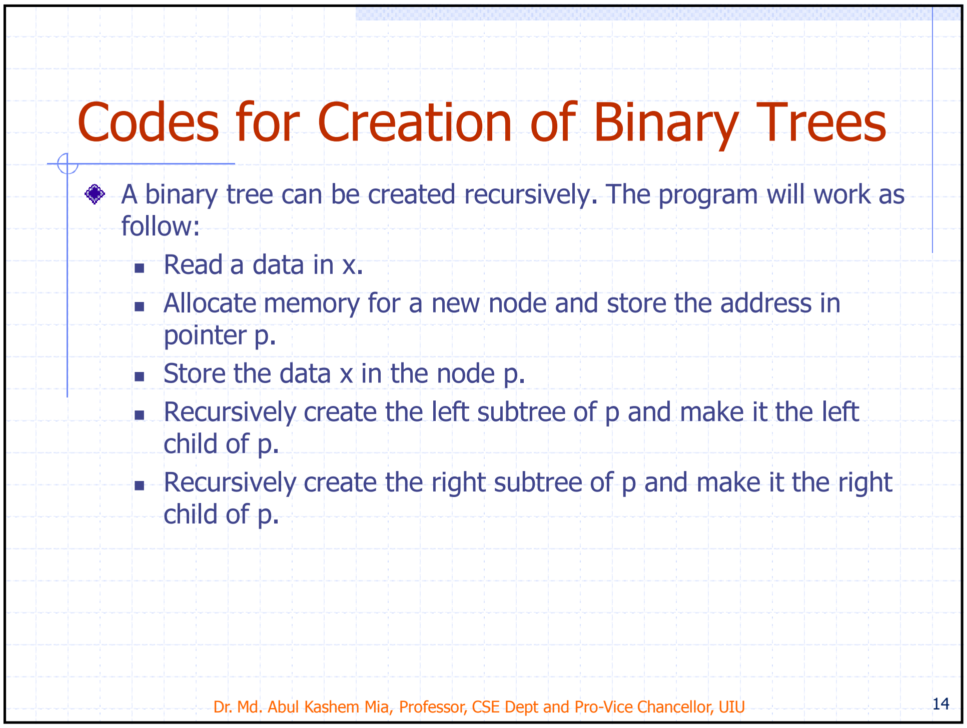
- In a (proper) binary tree T, the number of external nodes is 1 more than the number of internal nodes.

- Proof:



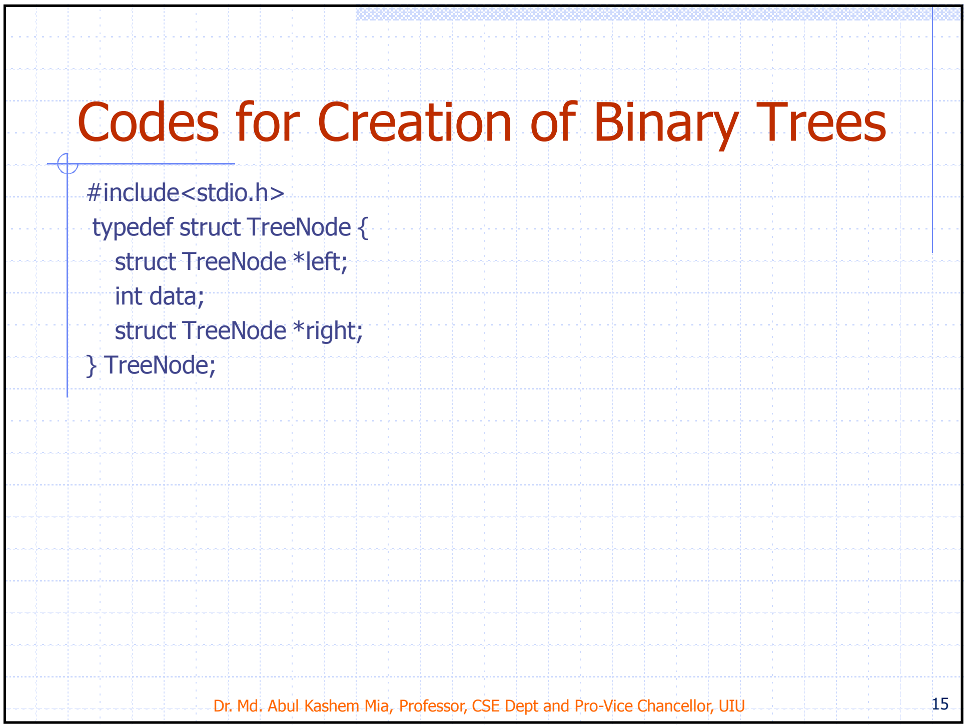
Linked Structure for Binary Trees

- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node



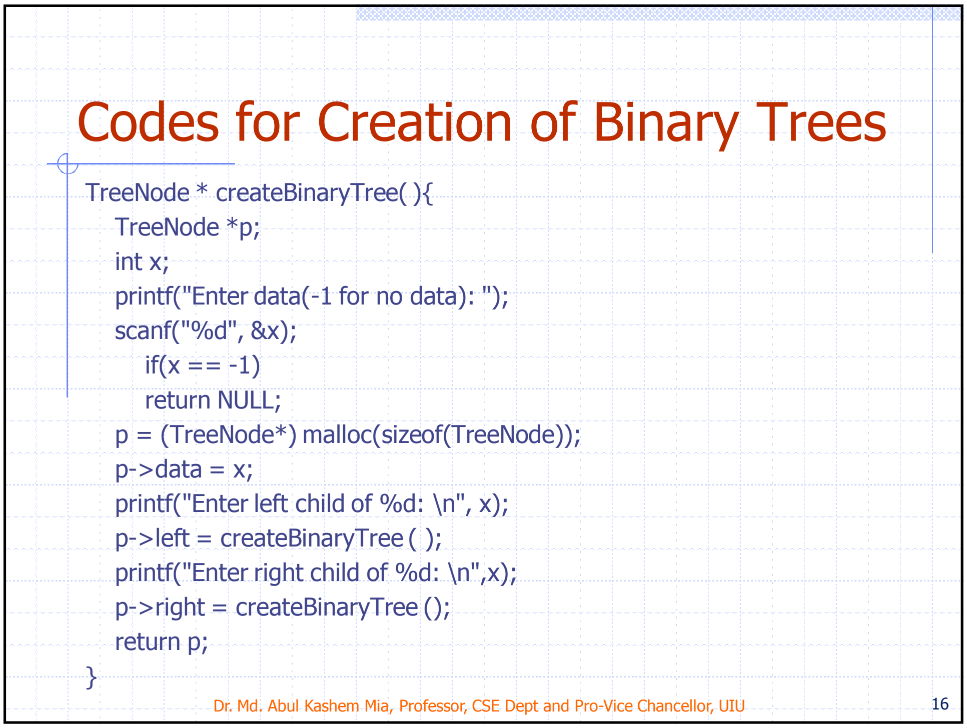
Codes for Creation of Binary Trees

- A binary tree can be created recursively. The program will work as follow:
  - Read a data in x.
  - Allocate memory for a new node and store the address in pointer p.
  - Store the data x in the node p.
  - Recursively create the left subtree of p and make it the left child of p.
  - Recursively create the right subtree of p and make it the right child of p.



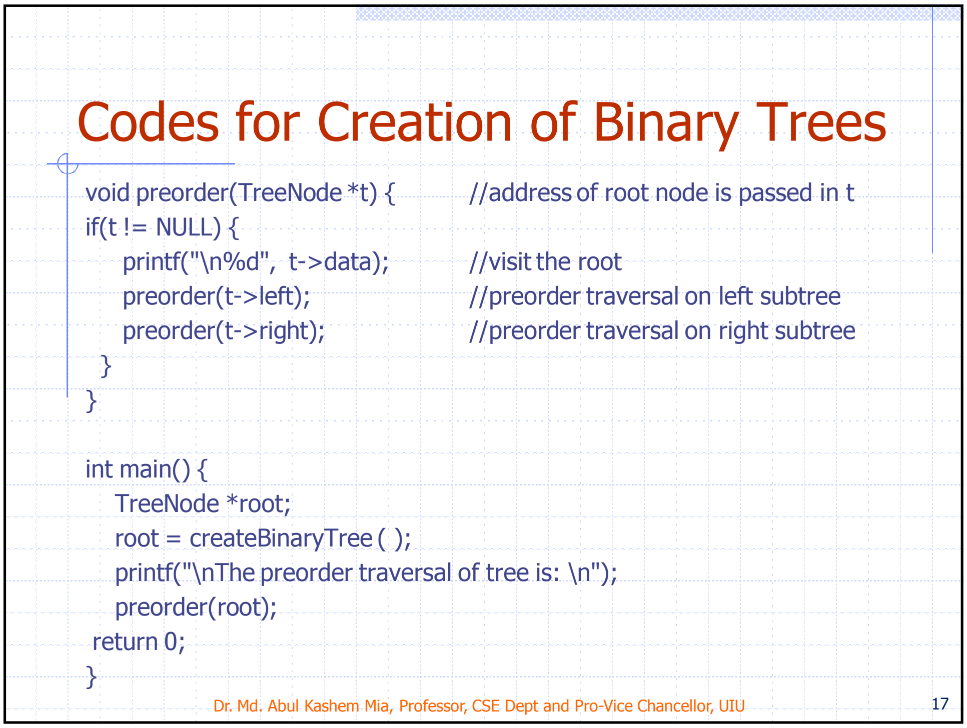
Codes for Creation of Binary Trees

```
#include<stdio.h>
 typedef struct TreeNode {
    struct TreeNode *left;
    int data;
    struct TreeNode *right;
} TreeNode;
```



Codes for Creation of Binary Trees

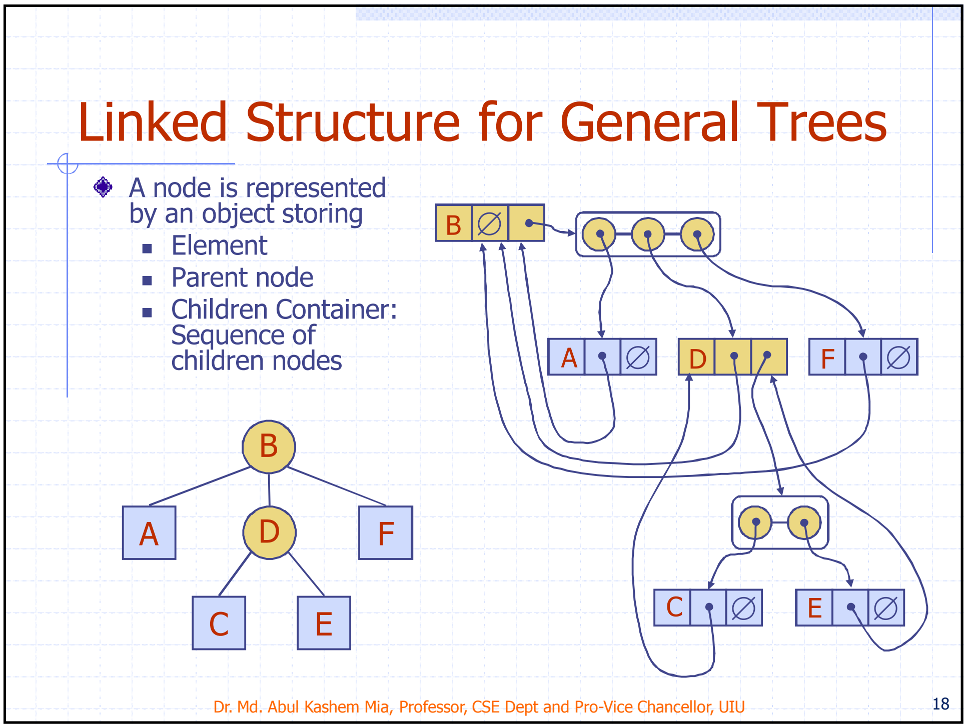
```
TreeNode * createBinaryTree( ){
    TreeNode *p;
    int x;
    printf("Enter data(-1 for no data): ");
    scanf("%d", &x);
        if(x == -1)
        return NULL;
    p = (TreeNode*) malloc(sizeof(TreeNode));
    p->data = x;
    printf("Enter left child of %d: \n", x);
    p->left = createBinaryTree( );
    printf("Enter right child of %d: \n",x);
    p->right = createBinaryTree ();
    return p;
}
```



Codes for Creation of Binary Trees

```
void preorder(TreeNode *t) {          //address of root node is passed in t
if(t != NULL) {
     printf("\n%d",  t->data);         //visit the root
     preorder(t->left);                //preorder traversal on left subtree
     preorder(t->right);               //preorder traversal on right subtree
  }
}

int main() {
    TreeNode *root;
    root = createBinaryTree ( );
    printf("\nThe preorder traversal of tree is: \n");
    preorder(root);
 return 0;
}
```

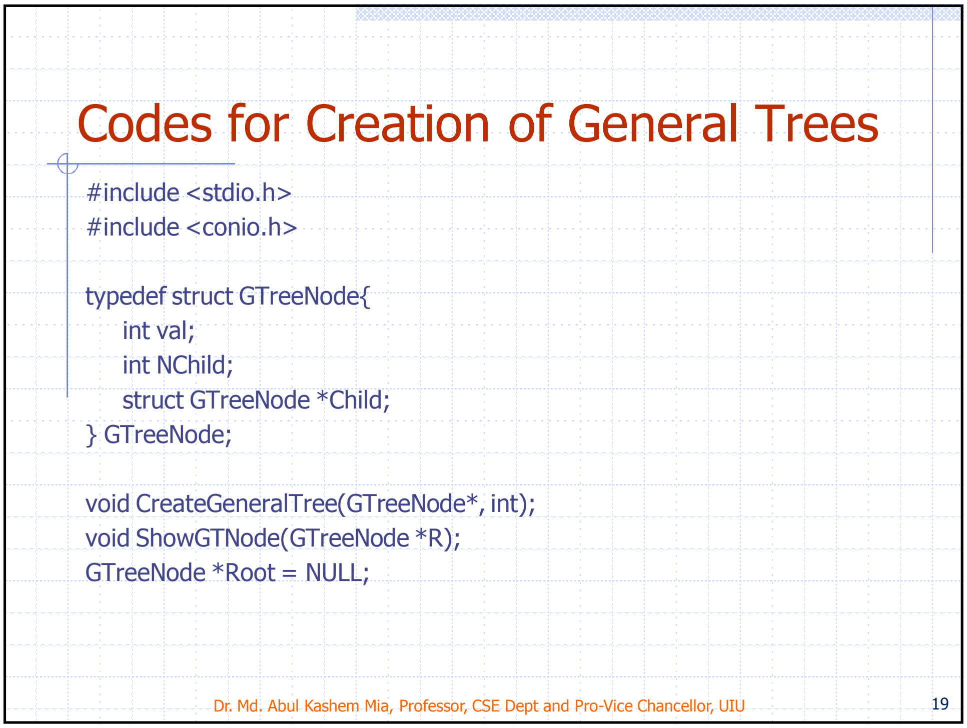
Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU 17



Linked Structure for General Trees

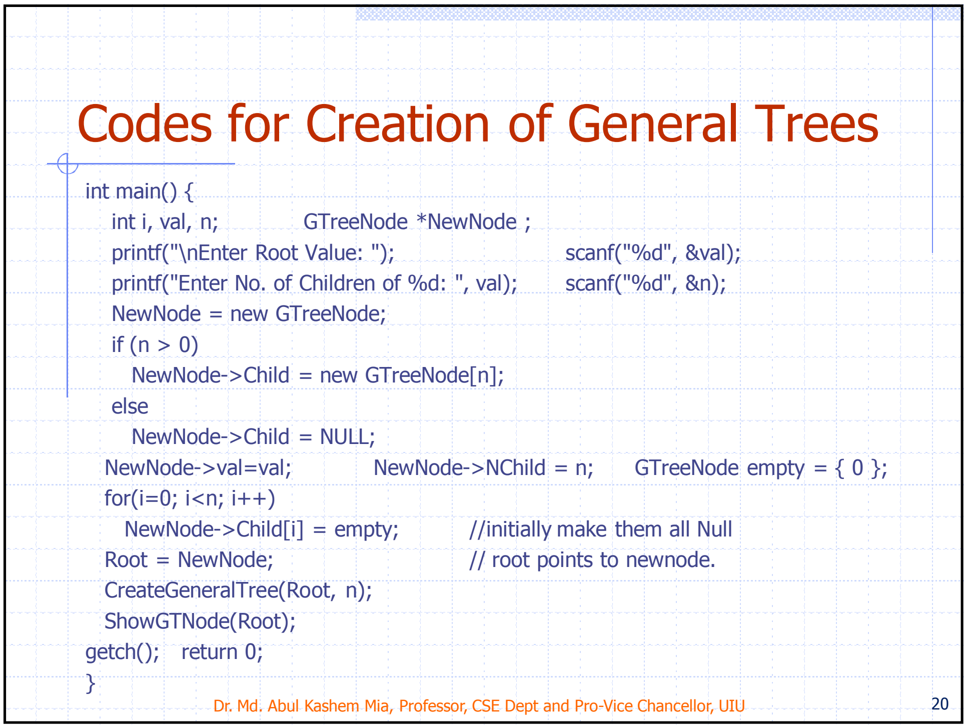
- A node is represented by an object storing
  - Element
  - Parent node
  - Children Container: Sequence of children nodes

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU 18



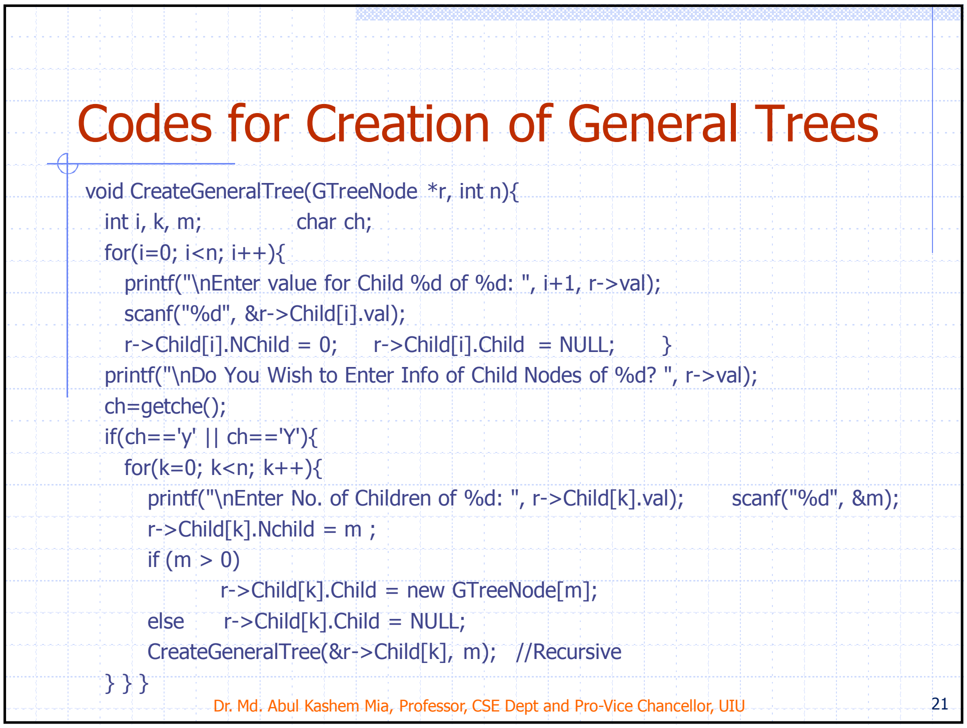
Codes for Creation of General Trees

```
#include <stdio.h>
#include <conio.h>

typedef struct GTreeNode{
     int val;
     int NChild;
     struct GTreeNode *Child;
} GTreeNode;

void CreateGeneralTree(GTreeNode*, int);
void ShowGTNode(GTreeNode *R);
GTreeNode *Root = NULL;
```



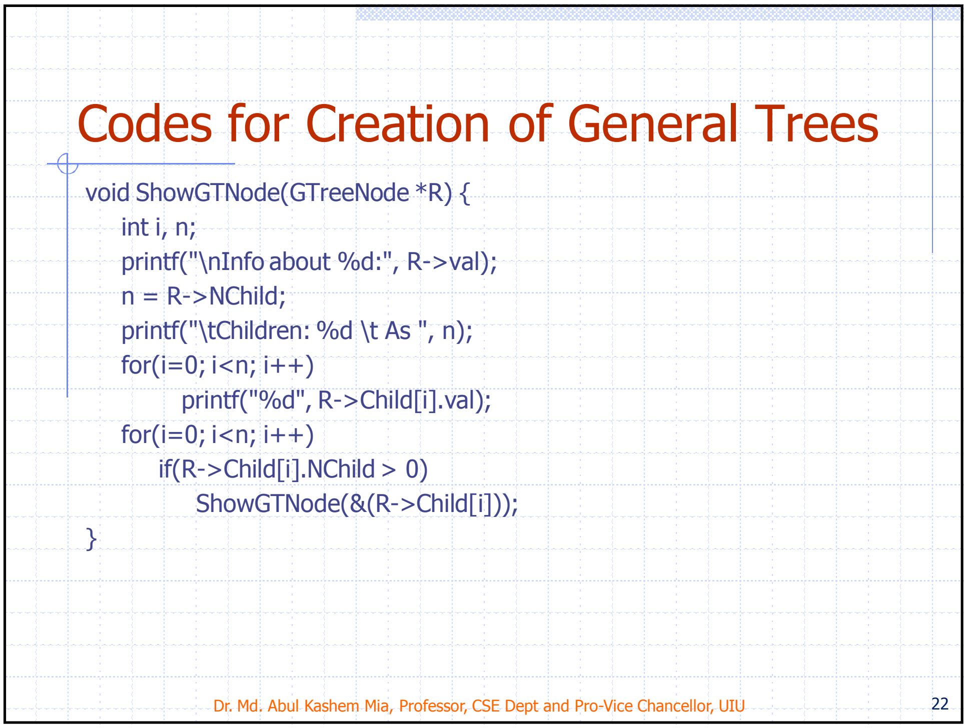
Codes for Creation of General Trees

```
int main() {
    int i, val, n;            GTreeNode *NewNode ;
    printf("\nEnter Root Value: ");                scanf("%d", &val);
    printf("Enter No. of Children of %d: ", val);       scanf("%d", &n);
    NewNode = new GTreeNode;
    if (n > 0)
       NewNode->Child = new GTreeNode[n];
    else
       NewNode->Child = NULL;
   NewNode->val=val;            NewNode->NChild = n;      GTreeNode empty = { 0 };
   for(i=0; i<n; i++)
      NewNode->Child[i] = empty;          //initially make them all Null
   Root = NewNode;                        // root points to newnode.
   CreateGeneralTree(Root, n);
   ShowGTNode(Root);
getch();   return 0;
}
```



Codes for Creation of General Trees

```
void CreateGeneralTree(GTreeNode  *r, int n){
  int i, k, m;              char ch;
  for(i=0; i<n; i++){
    printf("\nEnter value for Child %d of %d: ", i+1, r->val);
    scanf("%d", &r->Child[i].val);
    r->Child[i].NChild = 0;     r->Child[i].Child  = NULL;       }
  printf("\nDo You Wish to Enter Info of Child Nodes of %d? ", r->val);
  ch=getche();
  if(ch=='y' || ch=='Y'){
    for(k=0; k<n; k++){
       printf("\nEnter No. of Children of %d: ", r->Child[k].val);       scanf("%d", &m);
       r->Child[k].Nchild = m ;
       if (m > 0)
                r->Child[k].Child = new GTreeNode[m];
       else      r->Child[k].Child = NULL;
       CreateGeneralTree(&r->Child[k],  m);   //Recursive
  } } }
```



Codes for Creation of General Trees

```
void ShowGTNode(GTreeNode *R) {
    int i, n;
    printf("\nInfo about %d:", R->val);
    n = R->NChild;
    printf("\tChildren: %d \t As ", n);
    for(i=0; i<n; i++)
            printf("%d", R->Child[i].val);
    for(i=0; i<n; i++)
        if(R->Child[i].NChild > 0)
            ShowGTNode(&(R->Child[i]));
}
```