ECE 220: Computer Systems & Programming

Lecture 27: Review II

Thomas Moon

April 30, 2024

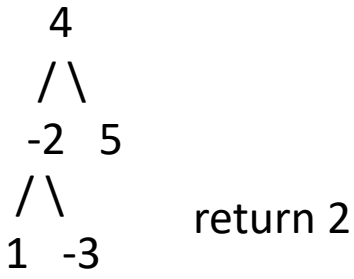
Get the example codes

- https://github.com/tmoon-illinois/ece220_sp24/tree/main/lec27

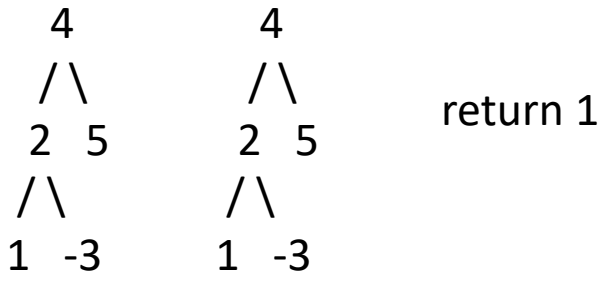
Practice Problem – Binary Tree

```
typedef struct TreeNode t_node;
struct TreeNode
{
    int data;
    t_node *left;
    t_node *right;
};
```

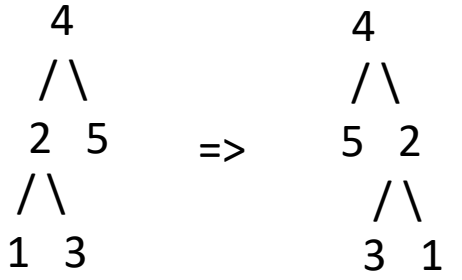
```
/* Count the number of negative values stored in a tree */
int countNeg(t_node *node);
```




```
/* Return 1, if the two trees are identical (otw, return 0)*/
int isSame(t_node *node1, t_node *node2);
```




```
/* Swap left and right subtree at every node */
void mirror(t_node *node);
```



1. Practice Problem – Count Negative Values

tree.c

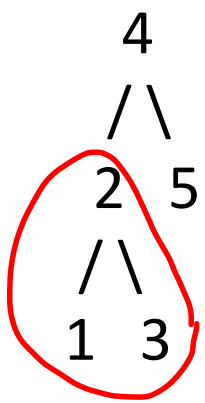
```
/* Count the number of negative values stored in a tree */
int countNeg(t_node *node){
    // Base case:
    if(node == NULL)
        return 0;
    // Recursive case:
    // add the num of negs from left subtree and right subtree
    int sum =   countNeg(node->left)  + countNeg(node->right);
    // if current node data is negative, return ???
    if(node->data < 0 )
        return sum+1;
    else
        return sum;
}
```

2. Practice Problem – Are they same tree?

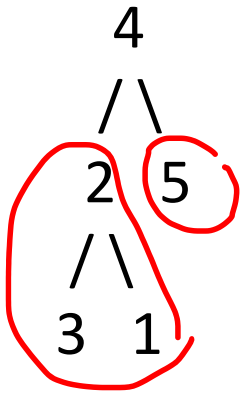
```
/* Return 1, if the two trees are identical (otw, return 0)*/
int isSame(t_node *node1, t_node *node2){
    // 1. both nodes are empty
    if(node1 == NULL && node2 == NULL)
        return 1;
    // 2. one node is empty, but the other one is not
    if( (node1 == NULL && node2 != NULL) || (node1 != NULL && node2 ==
NULL))
        return 0;
    // 3. both nodes are not empty (recursive)
    int isLeftSame = isSame(node1->left, node2->left);
    int isRightSame = isSame(node1->right, node2->right);
    if(isLeftSame && isRightSame && (node1->data == node2->data) )
        return 1;
    else
        return 0;
}
```

3. Practice Problem – Mirror the tree

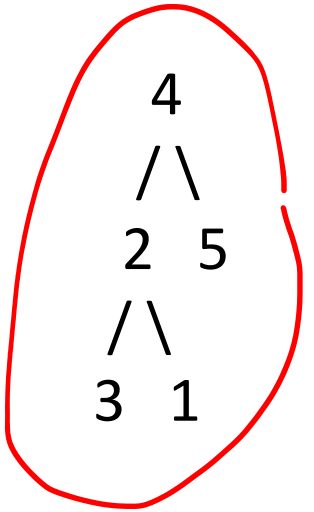
```
/* Swap left and right subtree at every node */
void mirror(t_node *node);
```



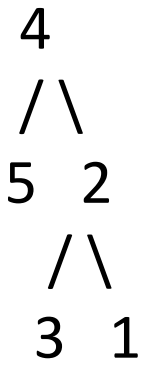

```
mirror(node->left);
        →
```




```
mirror(node->right);
        →
```




```
swapt the current
  left and right
        →
```

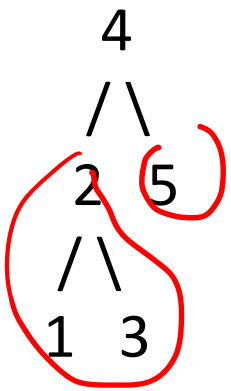


post-order

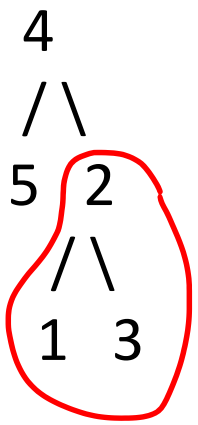
3. Practice Problem – Mirror the tree

```
/* Swap left and right subtree at every node */
void mirror(t_node *node);
```

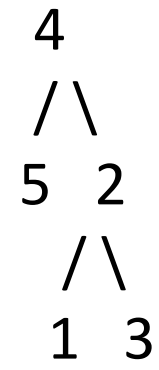
Alternative way



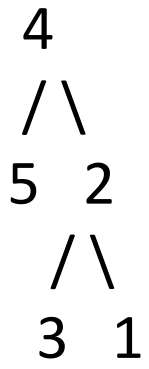
swapt the current left and right →



mirror(node->left); →



mirror(node->right); →

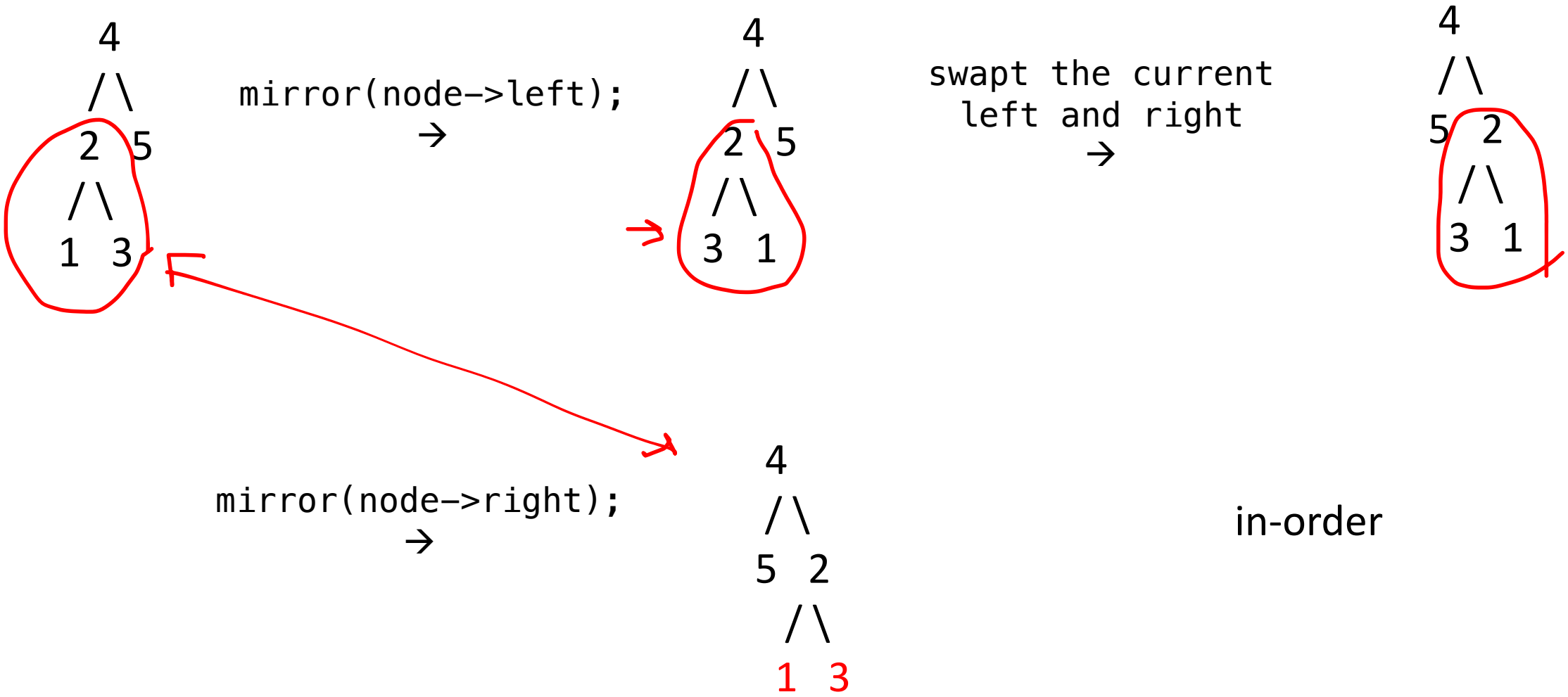


pre-order

3. Practice Problem – Mirror the tree

```
/* Swap left and right subtree at every node */
void mirror(t_node *node);
```

Wrong way



in-order

3. Practice Problem – Mirror the tree

```
/* Swap left and right subtree at every node */
void mirror(t_node *node)
{
    // base case
    if(node == NULL)
        return;
    // recursive for left subtree
    mirror(node->left);
    // recursive for right subtree
    mirror(node->right) ;
    // swap the left subtree and the right subtree
    t_node *temp = node->left;
    node->left = node->right;
    node->right = temp;
}
```

Final Exam

1. Linked-list
    1. single ptr vs double ptr vs dummy node vs return head pointer
    2. how to traverse
    3. do not lose the connection (sketch nodes and pointers)
2. Tree
    1. recursion (base+recursive)
    2. how to treverse to (or process) left subtree and right subtree
3. C++
    1. contructor, copy construct
    2. operator overloading
    3. iterator
    4. setter and getter
    5. string
4. C to LC3
    1. caller/callee buildup
    2. caller/callee teardown
    3. access struct member by pointer (see Lec24)
5. Concepts
    1. T/F
    2. common sense on LC3, C, C++