Tree traversal in C

```c
// Tree traversal in C

#include <stdio.h>
#include <stdlib.h>

struct node {
  int item;
  struct node* left;
  struct node* right;
};

// Inorder traversal
void inorderTraversal(struct node* root) {
  if (root == NULL) return;
  inorderTraversal(root->left);
  printf("%d ", root->item);
  inorderTraversal(root->right);
}

// Preorder traversal
void preorderTraversal(struct node* root) {
  if (root == NULL) return;
  printf("%d ", root->item);
  preorderTraversal(root->left);
  preorderTraversal(root->right);
}

// Postorder traversal
void postorderTraversal(struct node* root) {
  if (root == NULL) return;
  postorderTraversal(root->left);
  postorderTraversal(root->right);
  printf("%d ", root->item);
}

// Create a new Node
struct node* create(int value) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->item = value;
  newNode->left = NULL;
  newNode->right = NULL;

  return newNode;
}
```

```c
// Insert on the left of the node
struct node* insertLeft(struct node* root, int value) {
  root->left = create(value);
  return root->left;
}

// Insert on the right of the node
struct node* insertRight(struct node* root, int value) {
  root->right = create(value);
  return root->right;
}

int main() {
  struct node* root = create(1);
  insertLeft(root, 4);
  insertRight(root, 6);
  insertLeft(root->left, 42);
  insertRight(root->left, 3);
  insertLeft(root->right, 2);
  insertRight(root->right, 33);

  printf("Traversal of the inserted binary tree \n");
  printf("Inorder traversal \n");
  inorderTraversal(root);

  printf("\nPreorder traversal \n");
  preorderTraversal(root);


  printf("\nPostorder traversal \n");
  postorderTraversal(root);

}
```

Output:

```
PS C:\Users\vaishnavi> cd desktop
PS C:\Users\vaishnavi\desktop> gcc tree.c
PS C:\Users\vaishnavi\desktop> a
Traversal of the inserted binary tree
Inorder traversal
42 4 3 1 2 6 33
Preorder traversal
1 4 42 3 6 2 33
Postorder traversal
42 3 4 2 33 6 1
PS C:\Users\vaishnavi\desktop>
```

Graph

```c
// A C Program to demonstrate adjacency list
// representation of graphs
#include <stdio.h>
#include <stdlib.h>

// A structure to represent an adjacency list node
struct AdjListNode {
    int dest;
    struct AdjListNode* next;
};

// A structure to represent an adjacency list
struct AdjList {
    struct AdjListNode* head;
};

// A structure to represent a graph. A graph
// is an array of adjacency lists.
// Size of array will be V (number of vertices
// in graph)
struct Graph {
    int V;
    struct AdjList* array;
};

// A utility function to create a new adjacency list node
struct AdjListNode* newAdjListNode(int dest)
{
    struct AdjListNode* newNode
        = (struct AdjListNode*)malloc(
            sizeof(struct AdjListNode));
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
}

// A utility function that creates a graph of V vertices
struct Graph* createGraph(int V)
{
    struct Graph* graph
        = (struct Graph*)malloc(sizeof(struct Graph));
    graph->V = V;

    // Create an array of adjacency lists. Size of
    // array will be V
    graph->array = (struct AdjList*)malloc(
        V * sizeof(struct AdjList));
```

```
    // Initialize each adjacency list as empty by
    // making head as NULL
    int i;
    for (i = 0; i < V; ++i)
        graph->array[i].head = NULL;

    return graph;
}

// Adds an edge to an undirected graph
void addEdge(struct Graph* graph, int src, int dest)
{
    // Add an edge from src to dest. A new node is
    // added to the adjacency list of src. The node
    // is added at the beginning
    struct AdjListNode* check = NULL;
    struct AdjListNode* newNode = newAdjListNode(dest);

    if (graph->array[src].head == NULL) {
        newNode->next = graph->array[src].head;
        graph->array[src].head = newNode;
    }
    else {

        check = graph->array[src].head;
        while (check->next != NULL) {
            check = check->next;
        }
        // graph->array[src].head = newNode;
        check->next = newNode;
    }

    // Since graph is undirected, add an edge from
    // dest to src also
    newNode = newAdjListNode(src);
    if (graph->array[dest].head == NULL) {
        newNode->next = graph->array[dest].head;
        graph->array[dest].head = newNode;
    }
    else {
        check = graph->array[dest].head;
        while (check->next != NULL) {
            check = check->next;
        }
        check->next = newNode;
    }
```

```
    // newNode = newAdjListNode(src);
    // newNode->next = graph->array[dest].head;
    // graph->array[dest].head = newNode;
}
void printGraph(struct Graph* graph)
{
    int v;
    for (v = 0; v < graph->V; ++v) {
        struct AdjListNode* pCrawl = graph->array[v].head;
        printf("\n Adjacency list of vertex %d\n head ", v);
        while (pCrawl) {
            printf("-> %d", pCrawl->dest);
            pCrawl = pCrawl->next;
        }
        printf("\n");
    }
}

// Driver program to test above functions
int main()
{
    // create the graph given in above figure
    int V = 5;
    struct Graph* graph = createGraph(V);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 4);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 4);

    // print the adjacency list representation of the above
    // graph
    printGraph(graph);

    return 0;
}
```

Output:

```
Adjacency list of vertex 1
head -> 0-> 2-> 3-> 4

Adjacency list of vertex 2
head -> 1-> 3

Adjacency list of vertex 3
head -> 1-> 2-> 4

Adjacency list of vertex 4
head -> 0-> 1-> 3
PS C:\Users\vaishnavi\desktop>
```