S.No: 9	Exp. Name: ***Implementation of binary tree using linked list.***	Date:2024-06-02

ID: 92132321021

Aim:

Write a C program to implement Binary tree using Linked list.

Driver code is already provided for you to run the test cases.

Source Code:

TreeMain.c

```
/*
#include <stdio.h>
#include <stdlib.h>
#include"TreeStructure.c"
// For Queue Size
#define SIZE 50
// A utility function to create a new Queue
struct Queue* createQueue(int size) {
   struct Queue* queue = (struct Queue*) malloc(sizeof( struct Queue ));
   queue->front = queue->rear = -1;
   queue->size = size;
   queue->array = (struct node**) malloc(queue->size * sizeof( struct node* ));
   int i;
   for (i = 0; i < size; ++i)
      queue->array[i] = NULL;
   return queue;
}
// Standard Queue Functions
int isEmpty(struct Queue* queue) {
   return queue->front == -1;
}
int isFull(struct Queue* queue) {
   return queue->rear == queue->size - 1;
}
int hasOnlyOneItem(struct Queue* queue) {
   return queue->front == queue->rear;
}
void Enqueue(struct node *root, struct Queue* queue) {
   if (isFull(queue))
      return;
   queue->array[++queue->rear] = root;
   if (isEmpty(queue))
      ++queue->front;
}
struct node* Dequeue(struct Queue* queue) {
   if (isEmpty(queue))
      return NULL;
   struct node* temp = queue->array[queue->front];
   if (hasOnlyOneItem(queue))
      queue->front = queue->rear = -1;
   else
      ++queue->front;
   return temp;
}
```

```
struct node* getFront(struct Queue* queue) {
   return queue->array[queue->front];
}
int hasBothChild(struct node* temp) {
   return temp && temp->left && temp->right;
}
void insert(struct node **root, int data, struct Queue* queue) {
*/

/*
}
void removethedeepestnode(struct node *root, struct node *d_node){
    struct Queue* queue = createQueue(SIZE);
*/
    // Write your code here to remove the deepest node in a binary tree


/*
}
void levelOrder(struct node* root) {
   struct Queue* queue = createQueue(SIZE);
   Enqueue(root, queue);
   while (!isEmpty(queue)) {
      struct node* temp = Dequeue(queue);
      printf("%d ", temp->data);
      if (temp->left)
         Enqueue(temp->left, queue);
      if (temp->right)
         Enqueue(temp->right, queue);
   }
}

void findDeepestnode(struct node *root, struct node *search){
    struct Queue* queue = createQueue(SIZE);
   Enqueue(root, queue);
   struct node *temp, *par;
   while (!isEmpty(queue))
   {   par = temp;
      temp = Dequeue(queue);
      if (temp->left)
         Enqueue(temp->left, queue);
      if (temp->right)
         Enqueue(temp->right, queue);
   }
   search->data = temp->data;
    removethedeepestnode(root,temp);
}
struct node* searchingNode( struct node *root, int data) {
     struct Queue* queue = createQueue(SIZE);
   Enqueue(root, queue);
   struct node *searchNode = NULL;
   while (!isEmpty(queue))
   {  struct node* temp = Dequeue(queue);
      if(temp->data == data){
          searchNode = temp;
          return searchNode;
```

```
      }
      if (temp->left)
         Enqueue(temp->left, queue);
      if (temp->right)
         Enqueue(temp->right, queue);
   }
   return searchNode;
}
// Driver program to test above functions
int main()
{  struct node* root = NULL;
   struct Queue* queue = createQueue(SIZE);
   int i;
   int limit;
   int data1;
   printf("Enter the limit to form binary tree: ");
   scanf("%d",&limit);
   for(i = 1; i <= limit; ++i) {
      printf("Enter node: ");
      scanf("%d",&data1);
      insert(&root, data1, queue);
   }
   printf("Level order traversal before deletion\n");
   levelOrder(root);
   int data;
   printf("\nEnter data to delete: ");
   scanf("%d",&data);
   struct node *search = searchingNode(root, data);
   if(search == NULL)
       printf("Node is not present\n");
   else{
       findDeepestnode(root, search);
       printf("Level order traversal after deletion\n");
       levelOrder(root);
   }
   return 0;
}
*/
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// Define the tree node structure
struct node {
    int data;
    struct node *right, *left;
};

// Define the queue structure
struct Queue {
    int front, rear;
    int size;
    struct node **array;
};

// Function prototypes
```


ID: 92132321021

```
struct node *newNode(int data);
struct Queue *createQueue(int size);
void insert(struct node **root, int data, struct Queue *queue);
bool deleteNode(struct node **root, int key);
void levelOrder(struct node *root);
void enQueue(struct Queue *queue, struct node *item);
struct node *deQueue(struct Queue *queue);
bool isEmpty(struct Queue *queue);

// Create a new tree node
struct node *newNode(int data) {
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Create a queue
struct Queue *createQueue(int size) {
    struct Queue *queue = (struct Queue *)malloc(sizeof(struct Queue));
    queue->front = queue->rear = -1;
    queue->size = size;
    queue->array = (struct node **)malloc(queue->size * sizeof(struct node *));
    return queue;
}

// Enqueue a tree node
void enQueue(struct Queue *queue, struct node *item) {
    if (queue->rear == queue->size - 1)
        return;
    if (queue->front == -1)
        queue->front = 0;
    queue->array[++queue->rear] = item;
}

// Dequeue a tree node
struct node *deQueue(struct Queue *queue) {
    if (queue->front == -1)
        return NULL;
    struct node *temp = queue->array[queue->front];
    if (queue->front == queue->rear)
        queue->front = queue->rear = -1;
    else
        queue->front++;
    return temp;
}

// Check if the queue is empty
bool isEmpty(struct Queue *queue) {
    return queue->front == -1;
}

// Insert a new node into the binary tree
void insert(struct node **root, int data, struct Queue *queue) {
    struct node *temp = newNode(data);
    if (!*root) {
```

```
        *root = temp;
    } else {
        struct node *frontNode = queue->array[queue->front];
        if (!frontNode->left) {
            frontNode->left = temp;
        } else if (!frontNode->right) {
            frontNode->right = temp;
            deQueue(queue);
        }
    }
    enQueue(queue, temp);
}

// Delete the deepest node in the binary tree
void deleteDeepest(struct node *root, struct node *deepest) {
    struct Queue *queue = createQueue(100);
    enQueue(queue, root);

    struct node *temp;
    while (!isEmpty(queue)) {
        temp = deQueue(queue);

        if (temp == deepest) {
            temp = NULL;
            free(deepest);
            return;
        }
        if (temp->right) {
            if (temp->right == deepest) {
                temp->right = NULL;
                free(deepest);
                return;
            } else {
                enQueue(queue, temp->right);
            }
        }

        if (temp->left) {
            if (temp->left == deepest) {
                temp->left = NULL;
                free(deepest);
                return;
            } else {
                enQueue(queue, temp->left);
            }
        }
    }
}

// Delete a node from the binary tree
bool deleteNode(struct node **root, int key) {
    if (*root == NULL)
        return false;

    if ((*root)->left == NULL && (*root)->right == NULL) {
        if ((*root)->data == key) {
```

```
            free(*root);
            *root = NULL;
            return true;
        }
        return false;
    }

    struct Queue *queue = createQueue(100);
    enQueue(queue, *root);

    struct node *temp;
    struct node *keyNode = NULL;
    while (!isEmpty(queue)) {
        temp = deQueue(queue);

        if (temp->data == key) {
            keyNode = temp;
        }

        if (temp->left)
            enQueue(queue, temp->left);

        if (temp->right)
            enQueue(queue, temp->right);
    }

    if (keyNode != NULL) {
        keyNode->data = temp->data;
        deleteDeepest(*root, temp);
        return true;
    } else {
        printf("Node is not present\n");
        return false;
    }
}

// Perform level order traversal of the binary tree
void levelOrder(struct node *root) {
    if (root == NULL)
        return;

    struct Queue *queue = createQueue(100);
    enQueue(queue, root);

    while (!isEmpty(queue)) {
        struct node *temp = deQueue(queue);
        printf("%d ", temp->data);

        if (temp->left)
            enQueue(queue, temp->left);

        if (temp->right)
            enQueue(queue, temp->right);
    }
    printf("\n");
}
```

```
int main() {
    int limit, data1;
    struct node *root = NULL;
    struct Queue *queue = createQueue(100); // Assuming a maximum of 100 nodes

    printf("Enter the limit to form binary tree: ");
    scanf("%d", &limit);

    for (int i = 0; i < limit; i++) {
        printf("Enter node: ");
        scanf("%d", &data1);
        insert(&root, data1, queue);
    }

    printf("Level order traversal before deletion\n");
    levelOrder(root);

    int data;
    printf("Enter data to delete: ");
    scanf("%d", &data);
    if (deleteNode(&root, data)) {
        printf("Level order traversal after deletion\n");
        levelOrder(root);
    }

    return 0;
}
```

TreeStructure.c

```
// A tree node
struct node
{  int data;
   struct node *right,*left;
};

// A queue node
struct Queue
{  int front, rear;
   int size;
   struct node* *array;
};

// A utility function to create a new tree node
struct node* newNode(int data)
{  struct node* temp = (struct node*) malloc(sizeof( struct node ));
   temp->data = data;
   temp->left = temp->right = NULL;
   return temp;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the limit to form binary tree: 5
Enter node: 45
Enter node: 16
Enter node: 26
Enter node: 36
Enter node: 78
Level order traversal before deletion 26
45 16 26 36 78 26
Enter data to delete: 26
Level order traversal after deletion
45 16 78 36

Test Case - 2
User Output
Enter the limit to form binary tree: 6
Enter node: 45
Enter node: 15
Enter node: 26
Enter node: 48
Enter node: 26
Enter node: 36
Level order traversal before deletion 86
45 15 26 48 26 36 86
Enter data to delete: 86
Node is not present

ID: 92132321021