Laboratory 8

1. Questions

    1. Implement a linked list and perform following operations.
        i. Insert a node before and after a given node
        ii. Delete a node before and after a given node
    2. Implement a linked list to create and print a binary tree.

2. Algorithm

2.1 Insert a node before and after a given node-Delete a node before and after a given node

```
Step 1: start

Step 2: to add node before the given node

        2.1 allocate node
        2.2 put in the data
        2.3 check if head == null
        2.4 if it is, head = new and head->next= NULL
        2.5 else, iterate through the nodes till found key
        2.6 point the previous node to the newnode
        2.7 point the newnode to the next node

Step 3: to add node after the given node

        3.1 allocate node
        3.2 put in the data
        3.3 check if head == null
        3.4 if it is, head = new and head->next= NULL
        3.5 else, iterate through the nodes till found key
        3.6 point the key node to the newnode
        3.7 point the newnode to the next node
```

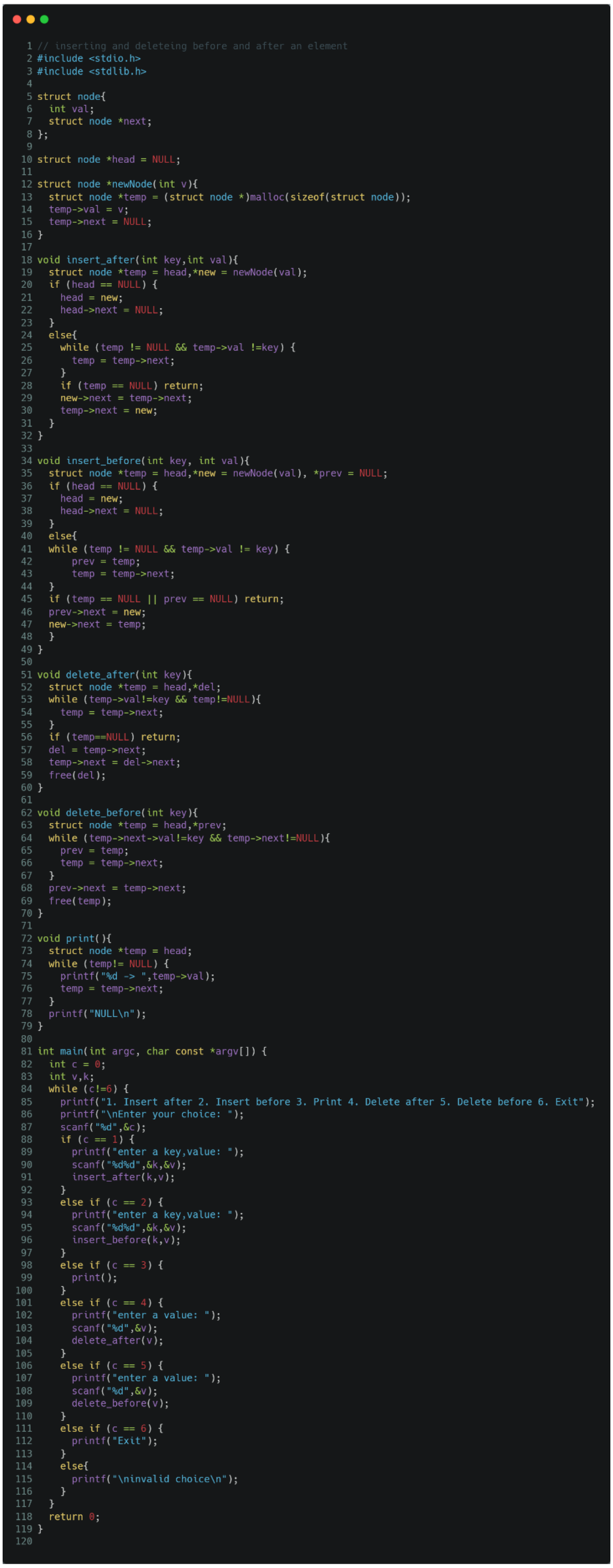
```
Step 4: to delete node before the given node
         4.1 iterate through the nodes till found key
         4.2 then, prev->next = temp->next
         4.3 free(temp)
Step 5: to delete node after the given node
         4.1 iterate through the nodes till found key
         4.2 del= temp->next
         4.2 then, temp->next = del->next
         4.3 free(del)
Step 6: stop
```

2 . 2   using linked list , create and print a binary tree.

```
Step 1: start
Step 2: allocate node
Step 3: put in the data
Step 4: push data into a linked list
      4.1 if (head == NULL):
      4.2 head = temp;
      4.3 head->next = NULL
      4.4 else: temp->next= head and head = temp
Step 5: print the linked list
Step 6: allocate a newtreenode
      6.1 temp ->info=value
      6.2 emp->count = 0;
      6.3 temp->left= temp->right = NULL;
```

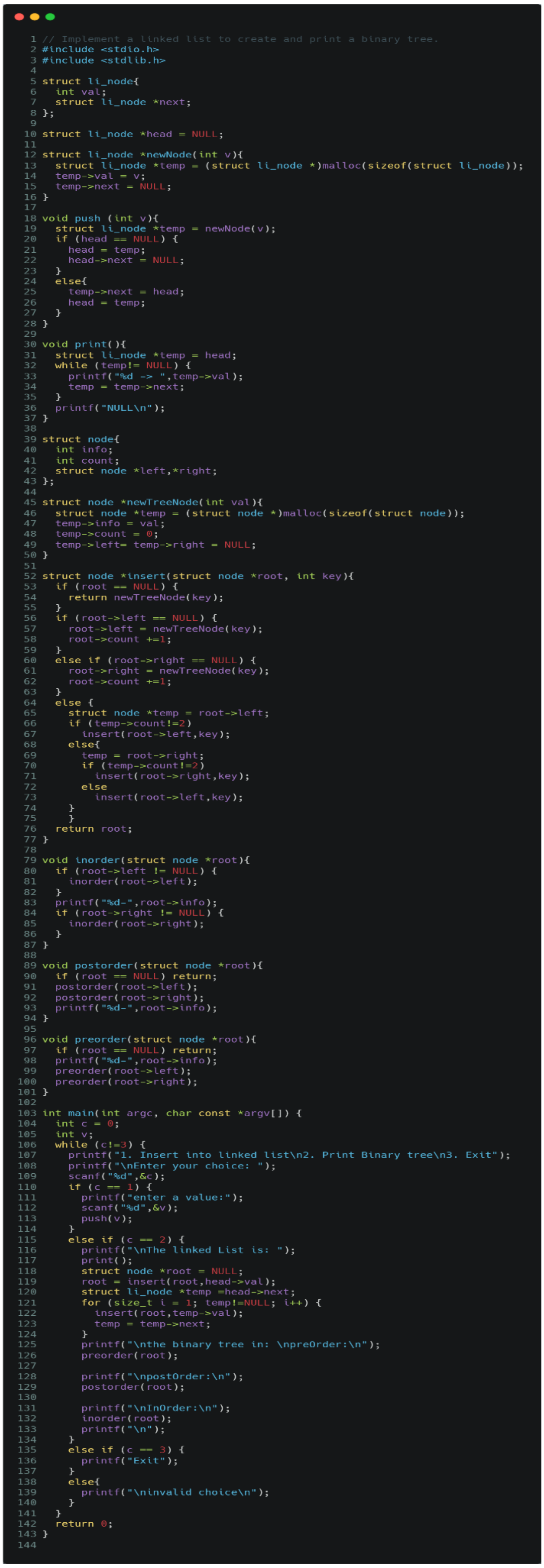
```
Step 7:  insert function

     7.1 if root=NULL: return newtreenode(key)

     7.2 if root->left = NULL: root->left = newTreeNode(key);

     7.3 if root->right = NULL: root->right = newTreeNode(key);

     7.4  if (temp->count! =2): insert(root->left,key);
          else {
                temp = root->right;
                7.4.1 if (temp->count! =2)
                    insert(root->right,key);
           7.4.2 else
               insert(root->left,key);
     7.5 return root

Step 8: stop
```

3. Program



```c
// inserting and deleteing before and after an element
#include <stdio.h>
#include <stdlib.h>

struct node{
  int val;
  struct node *next;
};

struct node *head = NULL;

struct node *newNode(int v){
  struct node *temp = (struct node *)malloc(sizeof(struct node));
  temp->val = v;
  temp->next = NULL;
}

void insert_after(int key,int val){
  struct node *temp = head,*new = newNode(val);
  if (head == NULL) {
    head = new;
    head->next = NULL;
  }
  else{
    while (temp != NULL && temp->val !=key) {
      temp = temp->next;
    }
    if (temp == NULL) return;
    new->next = temp->next;
    temp->next = new;
  }
}

void insert_before(int key, int val){
  struct node *temp = head,*new = newNode(val), *prev = NULL;
  if (head == NULL) {
    head = new;
    head->next = NULL;
  }
  else{
  while (temp != NULL && temp->val != key) {
      prev = temp;
      temp = temp->next;
  }
  if (temp == NULL || prev == NULL) return;
  prev->next = new;
  new->next = temp;
  }
}

void delete_after(int key){
  struct node *temp = head,*del;
  while (temp->val!=key && temp!=NULL){
    temp = temp->next;
  }
  if (temp==NULL) return;
  del = temp->next;
  temp->next = del->next;
  free(del);
}

void delete_before(int key){
  struct node *temp = head,*prev;
  while (temp->next->val!=key && temp->next!=NULL){
    prev = temp;
    temp = temp->next;
  }
  prev->next = temp->next;
  free(temp);
}

void print(){
  struct node *temp = head;
  while (temp!= NULL) {
    printf("%d -> ",temp->val);
    temp = temp->next;
  }
  printf("NULL\n");
}

int main(int argc, char const *argv[]) {
  int c = 0;
  int v,k;
  while (c!=6) {
    printf("1. Insert after 2. Insert before 3. Print 4. Delete after 5. Delete before 6. Exit");
    printf("\nEnter your choice: ");
    scanf("%d",&c);
    if (c == 1) {
      printf("enter a key,value: ");
      scanf("%d%d",&k,&v);
      insert_after(k,v);
    }
    else if (c == 2) {
      printf("enter a key,value: ");
      scanf("%d%d",&k,&v);
      insert_before(k,v);
    }
    else if (c == 3) {
      print();
    }
    else if (c == 4) {
      printf("enter a value: ");
      scanf("%d",&v);
      delete_after(v);
    }
    else if (c == 5) {
      printf("enter a value: ");
      scanf("%d",&v);
      delete_before(v);
    }
    else if (c == 6) {
      printf("Exit");
    }
    else{
      printf("\ninvalid choice\n");
    }
  }
  return 0;
}

```

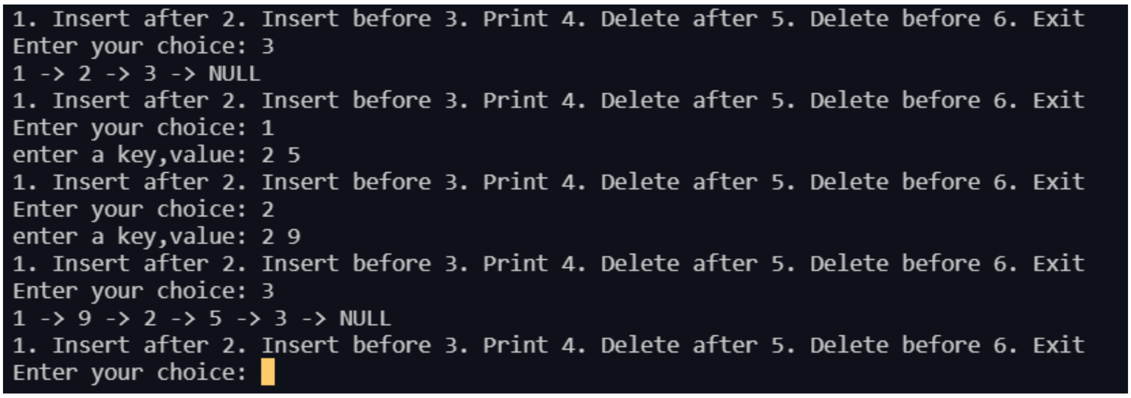
Figure 1 program to Insert a node before and after a given node-Delete a node before and after a given node



```c
// Implement a linked list to create and print a binary tree.
#include <stdio.h>
#include <stdlib.h>

struct li_node{
  int val;
  struct li_node *next;
};

struct li_node *head = NULL;

struct li_node *newNode(int v){
  struct li_node *temp = (struct li_node *)malloc(sizeof(struct li_node));
  temp->val = v;
  temp->next = NULL;
}

void push (int v){
  struct li_node *temp = newNode(v);
  if (head == NULL) {
    head = temp;
    head->next = NULL;
  }
  else{
    temp->next = head;
    head = temp;
  }
}

void print(){
  struct li_node *temp = head;
  while (temp!= NULL) {
    printf("%d -> ",temp->val);
    temp = temp->next;
  }
  printf("NULL\n");
}

struct node{
  int info;
  int count;
  struct node *left,*right;
};

struct node *newTreeNode(int val){
  struct node *temp = (struct node *)malloc(sizeof(struct node));
  temp->info = val;
  temp->count = 0;
  temp->left= temp->right = NULL;
}

struct node *insert(struct node *root, int key){
  if (root == NULL) {
    return newTreeNode(key);
  }
  if (root->left == NULL) {
    root->left = newTreeNode(key);
    root->count +=1;
  }
  else if (root->right == NULL) {
    root->right = newTreeNode(key);
    root->count +=1;
  }
  else {
    struct node *temp = root->left;
    if (temp->count!=2)
      insert(root->left,key);
    else{
      temp = root->right;
      if (temp->count!=2)
        insert(root->right,key);
      else
        insert(root->left,key);
    }
    }
  return root;
}

void inorder(struct node *root){
  if (root->left != NULL) {
    inorder(root->left);
  }
  printf("%d-",root->info);
  if (root->right != NULL) {
    inorder(root->right);
  }
}

void postorder(struct node *root){
  if (root == NULL) return;
  postorder(root->left);
  postorder(root->right);
  printf("%d-",root->info);
}

void preorder(struct node *root){
  if (root == NULL) return;
  printf("%d-",root->info);
  preorder(root->left);
  preorder(root->right);
}

int main(int argc, char const *argv[]) {
  int c = 0;
  int v;
  while (c!=3) {
    printf("1. Insert into linked list\n2. Print Binary tree\n3. Exit");
    printf("\nEnter your choice: ");
    scanf("%d",&c);
    if (c == 1) {
      printf("enter a value:");
      scanf("%d",&v);
      push(v);
    }
    else if (c == 2) {
      printf("\nThe linked List is: ");
      print();
      struct node *root = NULL;
      root = insert(root,head->val);
      struct li_node *temp =head->next;
      for (size_t i = 1; temp!=NULL; i++) {
        insert(root,temp->val);
        temp = temp->next;
      }
      printf("\nthe binary tree in: \npreOrder:\n");
      preorder(root);

      printf("\npostOrder:\n");
      postorder(root);

      printf("\nInOrder:\n");
      inorder(root);
      printf("\n");
    }
    else if (c == 3) {
      printf("Exit");
    }
    else{
      printf("\ninvalid choice\n");
    }
  }
  return 0;
}

```

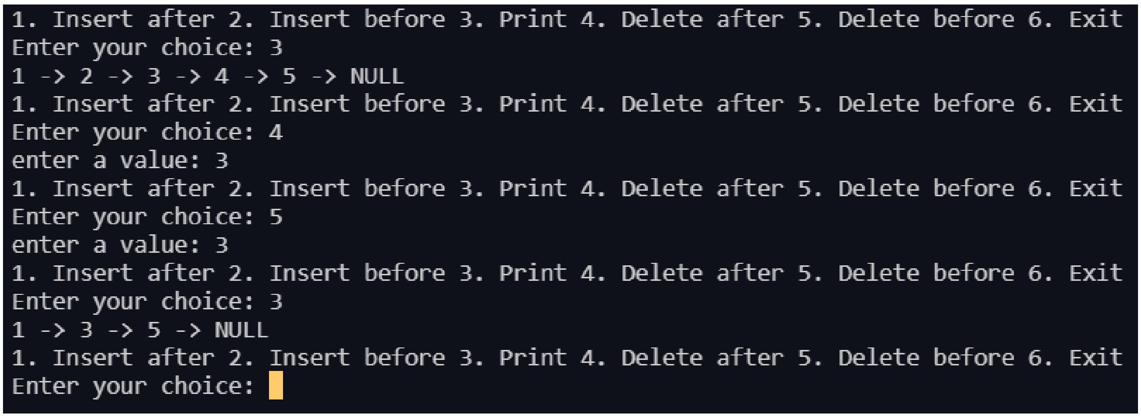
Figure 2 program to create and print a binary tree using linked list

4. Presentation of Results



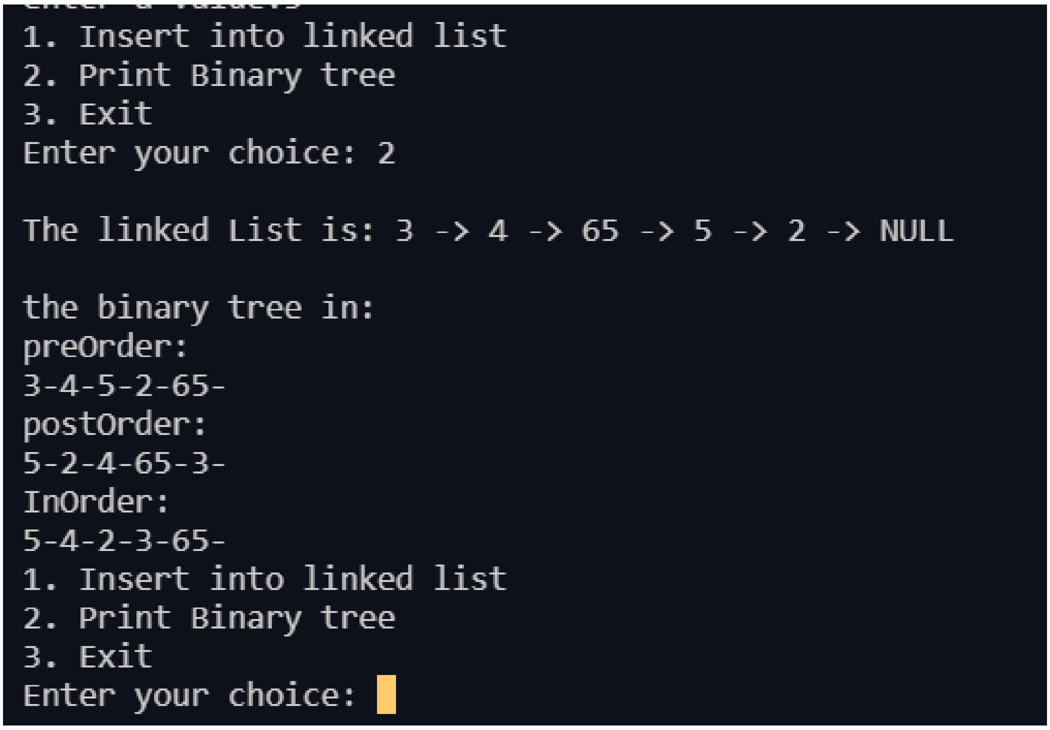
```
1. Insert after 2. Insert before 3. Print 4. Delete after 5. Delete before 6. Exit
Enter your choice: 3
1 -> 2 -> 3 -> NULL
1. Insert after 2. Insert before 3. Print 4. Delete after 5. Delete before 6. Exit
Enter your choice: 1
enter a key,value: 2 5
1. Insert after 2. Insert before 3. Print 4. Delete after 5. Delete before 6. Exit
Enter your choice: 2
enter a key,value: 2 9
1. Insert after 2. Insert before 3. Print 4. Delete after 5. Delete before 6. Exit
Enter your choice: 3
1 -> 9 -> 2 -> 5 -> 3 -> NULL
1. Insert after 2. Insert before 3. Print 4. Delete after 5. Delete before 6. Exit
Enter your choice:
```

Figure 3 inserting a node before and after a given node



```
1. Insert after 2. Insert before 3. Print 4. Delete after 5. Delete before 6. Exit
Enter your choice: 3
1 -> 2 -> 3 -> 4 -> 5 -> NULL
1. Insert after 2. Insert before 3. Print 4. Delete after 5. Delete before 6. Exit
Enter your choice: 4
enter a value: 3
1. Insert after 2. Insert before 3. Print 4. Delete after 5. Delete before 6. Exit
Enter your choice: 5
enter a value: 3
1. Insert after 2. Insert before 3. Print 4. Delete after 5. Delete before 6. Exit
Enter your choice: 3
1 -> 3 -> 5 -> NULL
1. Insert after 2. Insert before 3. Print 4. Delete after 5. Delete before 6. Exit
Enter your choice:
```

Figure 4 deleting a node before and after a given node



```
1. Insert into linked list
2. Print Binary tree
3. Exit
Enter your choice: 2

The linked List is: 3 -> 4 -> 65 -> 5 -> 2 -> NULL

the binary tree in:
preOrder:
3-4-5-2-65-
postOrder:
5-2-4-65-3-
InOrder:
5-4-2-3-65-
1. Insert into linked list
2. Print Binary tree
3. Exit
Enter your choice:
```

Figure 5 output of program to create and print a binary tree using linked list

5. Conclusions

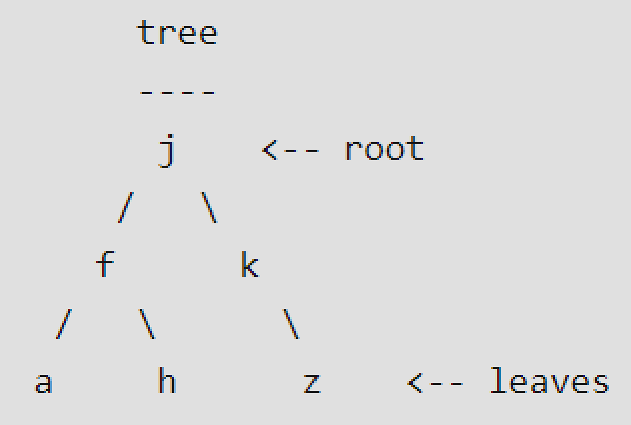
Learning happened

Trees are hierarchical data structures.

The topmost node is called **Root** of the tree. The elements that are directly under an element are called **its Children**. The element directly above something is called its **Parent**.

Binary Tree: A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.

Example:



```
     tree
     ----
      j    <-- root
    /   \
   f      k
 /   \      \
a     h      z    <-- leaves
```

Here,

- **'j'** is the root.
- children of **'f'** are **'a'** and **'h'**
- Parent of **'a'** and **'h'** is **'f'**
- **'a'**, **'h'** and **'z'** are leaves of the tree.