MODULE -3

REGULAR PROGRAMS – INSERT A NODE AT THE HEAD OF A LINKED LIST, INSERT A NODE AT THE TAIL OF A LINKED LIST, INSERT A NODE AT A SPECIFIC POSITION IN A LINKED LIST

1. **INSERT A NODE AT THE HEAD OF A LINKED LIST**

```
// A complete working C program to demonstrate
// all insertion methods on Linked List
#include <stdio.h>
#include <stdlib.h>

// A linked list node
struct Node
{
int data;
struct Node *next;
};

// Given a reference (pointer to pointer) to
// the head of a list and an int, inserts a
// new node on the front of the list.
void push(struct Node** head_ref,
                int new_data)
{
        // 1. Allocate node
        struct Node* new_node =
                (struct Node*) malloc(sizeof(struct Node));

        // 2. Put in the data
        new_node->data = new_data;

        // 3. Make next of new node as head
        new_node->next = (*head_ref);

        // 4. move the head to point to
        // the new node
        (*head_ref) = new_node;
}

// Given a node prev_node, insert a
// new node after the given prev_node
void insertAfter(struct Node* prev_node,
                                int new_data)
{
        // 1. Check if the given prev_node
        // is NULL
        if (prev_node == NULL)
```

```
        {
        printf("the given previous node cannot be NULL");
        return;
        }

        // 2. Allocate new node
        struct Node* new_node =
                (struct Node*) malloc(sizeof(struct Node));

        // 3. Put in the data
        new_node->data = new_data;

        // 4. Make next of new node as next
        // of prev_node
        new_node->next = prev_node->next;

        // 5. Move the next of prev_node
        // as new_node
        prev_node->next = new_node;
}

// Given a reference (pointer to pointer) to
// the head of a list and an int, appends a
// new node at the end
void append(struct Node** head_ref,  int new_data)
{

}

// This function prints contents of the
// linked list starting from head
void printList(struct Node *node)
{
while (node != NULL)
{
        printf(" %d ", node->data);
        node = node->next;
}
}

// Driver code
int main()
{
// Start with the empty list
struct Node* head = NULL;

// Insert 6. So linked list
```

```
    // becomes 6->NULL
    append(&head, 6);

    // Insert 7 at the beginning.
    // So linked list becomes 7->6->NULL
    push(&head, 7);

    // Insert 1 at the beginning. So
    // linked list becomes 1->7->6->NULL
    push(&head, 1);

    // Insert 4 at the end. So linked list
    // becomes 1->7->6->4->NULL
    append(&head, 4);

    // Insert 8, after 7. So linked list
    // becomes 1->7->8->6->4->NULL
    insertAfter(head->next, 8);

    printf("Created Linked list is: ");
    printList(head);

    return 0;
    }
```

2. **INSERT A NODE AT THE TAIL OF A LINKED LIST**

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

// Structure for a node in the linked list
struct Node {
   int data;
   struct Node *next;
};

// Function to insert a node at the tail
void insertAtTail(struct Node **head, int data)
{
}

// Function to print the linked list
void printList(struct Node *head) {
   struct Node *current = head;
   while (current != NULL) {
      printf("%d -> ", current->data);
      current = current->next;
```

```
    }
    printf("NULL\n");
}

int main() {
    struct Node *head = NULL; // Initialize the head pointer to NULL

    // Insert some nodes at the tail
    insertAtTail(&head, 10);
    insertAtTail(&head, 20);
    insertAtTail(&head, 30);

    // Print the linked list
    printf("Linked List: ");
    printList(head);

    // Free the allocated memory (important to avoid memory leaks)
    struct Node *current = head;
    while (current != NULL) {
        struct Node *temp = current;
        current = current->next;
        free(temp);
    }

    return 0;
}
```

3. INSERT A NODE AT A SPECIFIC POSITION IN A LINKED LIST

Program:

```
#include <stdio.h>
#include <stdlib.h>
struct slinklist {
    int data;
    struct slinklist *next;
};
typedef struct slinklist node;
node *start = NULL;
int menu() {
    int ch;
    printf("\n 1.Create a list ");
    printf("\n-------------------------");
    printf("\n 2.Insert a node at specified position");
    printf("\n-------------------------");
    printf("\n 3.Display");
    printf("\n-------------------------");
```

```
    printf("\n 4. Exit ");
    printf("\n\n Enter your choice: ");
    scanf("%d", &ch);
    return ch;
}
node* getnode() {
    node *newnode;
    newnode = (node *)malloc(sizeof(node));
    printf("\n Enter data: ");
    scanf("%d", &newnode->data);
    newnode->next = NULL;
    return newnode;
}
void createlist(int n) {
    int i;
    node *newnode;
    node *temp;
    for (i = 0; i < n; i++) {
        newnode = getnode();
        if (start == NULL) {
            start = newnode;
        } else {
            temp = start;
            while (temp->next != NULL)
                temp = temp->next;
            temp->next = newnode;
        }
    }
}
int countnode(node *ptr) {
    int count = 0;
    while (ptr != NULL) {
        count++;
        ptr = ptr->next;
    }
    return count;
}
void display() {
    node *temp;
    temp = start;
    printf("\n The contents of List (Left to Right): \n");
    if (start == NULL) {
        printf("\n Empty List");
        return;
    } else {
        while (temp != NULL) {
            printf("%d-->", temp->data);
```

```
            temp = temp->next;
        }
    }
    printf(" X ");
}

void insert_at_pos()
{

}
void main(void) {
    int ch, n;
    while (1) {
        ch = menu();
        switch (ch) {
            case 1:
                if (start == NULL) {
                    printf("\n Number of nodes you want to create: ");
                    scanf("%d", &n);
                    createlist(n);
                    printf("\n List created..");
                } else
                    printf("\n List is already created..");
                break;
            case 2:
                insert_at_pos();
                break;
            case 3:
                display();
                break;
            default:
                exit(0);
        }
    }
}
```

MODULE- 4

REGULAR PROGRAMS – POISONOUS PLANT, TRUCK TOUR, QUEUE USING TWO STACKS

4. POISONOUS PLANT

```
#include<stdio.h>
#include<stdlib.h>
// Define the structure for a stack element
typedef struct {
   int pesticide;
   int days;
} Plant;
// Function to find the number of days until no plants die
int poisonousPlants(int n, int* p)
{
}
int main() {
   int n;
   scanf("%d", &n);
   int* p = (int*)malloc(n * sizeof(int));
   for (int i = 0; i < n; i++) {
      scanf("%d", &p[i]);
   }
   int result = poisonousPlants(n, p);
   printf("%d\n", result);
   free(p);
   return 0;
}
```

5. Truck Tour

```
#include <stdio.h>

int main() {

   int n;

   scanf("%d", &n);

   int petrol[n], distance[n];

      for (int i = 0; i < n; i++) {

      scanf("%d %d", &petrol[i], &distance[i]);

   }

      return 0;

}
```

MODULE - 5

REGULAR PROGRAMS – LOWEST COMMON ANCESTOR, HEIGHT OF BINARY TREE, BINARY SEARCH TREE INSERTION

6. Lowest Common Ancestor in Binary Tree

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
 int data;
 struct node *left, *right;
};
struct node *lca (struct node *root, int n1, int n2)
{

}
struct node *newNode (int data)
{
 struct node *node = (struct node *) malloc (sizeof (struct node));
 node->data = data;
 node->left = node->right = NULL;
 return (node);
}
int main ()
{
 struct node *root = newNode (20);
 root->left = newNode (8);
 root->right = newNode (22);
 root->left->left = newNode (4);
 root->left->right = newNode (12);
 root->left->right->left = newNode (10);
 root->left->right->right = newNode (14);
 int n1 = 10, n2 = 14;
 struct node *t = lca (root, n1, n2);
 printf ("LCA of %d and %d is %d \n", n1, n2, t->data);
 n1 = 14, n2 = 8;
 t = lca (root, n1, n2);
 printf ("LCA of %d and %d is %d \n", n1, n2, t->data);
 n1 = 10, n2 = 22;
 t = lca (root, n1, n2);
 printf ("LCA of %d and %d is %d \n", n1, n2, t->data);
 getchar ();
 return 0;
}
```

7. Height of a Binary Tree

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
 int data;
 struct node *left;
 struct node *right;
};
int height (struct node *node)
{
}
struct node *newNode (int data)
{
 struct node *node = (struct node *) malloc (sizeof (struct node));
 node->data = data;
 node->left = NULL;
 node->right = NULL;
 return (node);
}
int main ()
{
 struct node *root = newNode (10);
 root->left = newNode (20);
 root->right = newNode (30);
 root->left->left = newNode (40);
 root->left->right = newNode (50);
 printf ("Height of tree is %d", height (root));
 return 0;
}
```

8. BINARY SEARCH TREE INSERTION

```
#include<stdio.h>
#include<stdlib.h>
// Basic struct of Tree
struct node
{
   int val;
   struct node *left, *right;
};
  // Function to create a new Node
struct node* newNode(int item)
{
   struct node* temp =  (struct node *)malloc(sizeof(struct node));
   temp->val = item;
   temp->left = temp->right = NULL;
   return temp;
}
  // Function print the node in inorder format, when insertion is complete
void inorder(struct node* root)
{
   if (root != NULL)
   {
      inorder(root->left);
      printf("%d \n", root->val);
      inorder(root->right);
   }
}
  // Here we are finding where to insert the new node so BST is followed
struct node* insert(struct node* node, int val)
{


}
 int main()
{
```

```
  /* Our BST will look like this
        100
       /   \
      40    140
     / \   / \
    40  80 120 160 */
  struct node* root = NULL;
  root = insert(root, 100);
  insert(root, 60);
  insert(root, 40);
  insert(root, 80);
  insert(root, 140);
  insert(root, 120);
  insert(root, 160);
      // Finally printing the tree using inorder
  inorder(root);
    return 0;
}
```