Week 1 DAA Lab Submission

Name: Davasam Karthikeya

Reg No: 230962326

Class and Lab Batch: AIMLB and B2

Date: 04/01/2025

Question 1)

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

typedef struct Node{
   int data;
   struct Node *next;
   struct Node *prev;
}*node_ptr;
node_ptr create_node(int data){
   node_ptr ptr = (node_ptr) malloc(sizeof(node_ptr));
   ptr->data = data;
   ptr->next = NULL;
   ptr->prev = NULL;
}
void display(node_ptr head){
   node_ptr ptr;
   ptr = head;
   while(ptr->next != NULL){
      printf("%d -> ",ptr->data);
      ptr = ptr->next;
   }
   printf("%d \n",ptr->data);
}
node_ptr insert_end(node_ptr head,int item){
```

```
    node_ptr new_node = create_node(item);
    if(head == NULL)return new_node;
    node_ptr ptr;
    ptr = head;
    while(ptr->next != NULL)ptr = ptr->next;
    new_node->prev = ptr;
    ptr->next = new_node;
    return head;
}
node_ptr insert_beg(node_ptr head,int item){
    node_ptr new_node = create_node(item);
    if(head == NULL)return new_node;
    new_node->next = head;
    head->prev = new_node;
    return new_node;
}
node_ptr insert_left(node_ptr head, int key, int item){
    node_ptr new_node = create_node(item);
    if(head == NULL)return NULL;
    node_ptr ptr,ptr1;
    ptr = NULL;ptr1 = head;
    while(ptr1 != NULL){
        if(ptr1->data == key){
            if(ptr == NULL){
                ptr1->prev = new_node;
                new_node->next = ptr1;
                return new_node;
            }
            ptr->next = new_node;
            new_node->prev = ptr;
            new_node->next = ptr1;
```

```
        ptr1->prev = new_node;
        return head;
      }
      ptr = ptr1;
      ptr1 = ptr1->next;
    }
}
void search_item(node_ptr head, int item){
    node_ptr ptr;
    int count = 0;
    ptr = head;
    while(ptr != NULL){
      if(ptr->data == item)count++;
      ptr = ptr->next;
    }
    printf("Found Element %d at %d Instances!\n",item,count);
}
void delete_all(node_ptr head, int item){
    node_ptr ptr,ptr_prev;
    ptr_prev = NULL;ptr = head;
    while(ptr != NULL){
      if(ptr->data == item){
        if(ptr != NULL){
          ptr_prev->next = ptr->next;
          if(ptr->next != NULL)ptr->next->prev = ptr_prev;
        }
        free(ptr);
        ptr = ptr_prev;
      }
      ptr_prev = ptr;
      ptr = ptr->next;
```

```
    }
}
int main(){
    node_ptr head;
    head = NULL;
    head = insert_beg(head,2);
    head = insert_beg(head,3);
    head = insert_beg(head,4);
    head = insert_beg(head,1);
    display(head);
    head = insert_left(head,3,8);
    display(head);
    search_item(head,8);
    delete_all(head, 8);
    display(head);
    return 0;
}
```

Output:

```
4 3 2 student@lpcp-23:~/Desktop/230962326/Week1$ ./q1
1 -> 4 -> 3 -> 2
1 -> 4 -> 8 -> 3 -> 2
Found Element 8 at 1 Instances!
1 -> 4 -> 3 -> 2
student@lpcp-23:~/Desktop/230962326/Week1$ ./q2
```

Question 2)

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{
    int data;
    struct Node *left;
    struct Node *right;
```

```
}*node_ptr;

node_ptr create_node(int data){
   node_ptr ptr = (node_ptr) malloc(sizeof(node_ptr));
   ptr->data = data;
   ptr->left = NULL;
   ptr->right = NULL;
}

node_ptr insert_BT(node_ptr head, int item){
   node_ptr new_node = create_node(item);
   if(head == NULL)return new_node;
   node_ptr ptr,ptr_prev;
   ptr = head; ptr_prev = NULL;
   while(ptr != NULL){
      ptr_prev = ptr;
      if(ptr->data > item)ptr = ptr->left;
      else if(ptr->data < item) ptr = ptr->right;
      else{
         printf("Key %d already Exists! \n",item);
         return head;
      }
   }
   if(ptr_prev->data > item)ptr_prev->left = new_node;
   else ptr_prev->right = new_node;
   return head;
}
void recInOrder(node_ptr ptr){
   if(ptr != NULL){
      recInOrder(ptr->left);
      printf("%d ",ptr->data);
```

```
        recInOrder(ptr->right);
    }
}
void recPreOrder(node_ptr ptr){
    if(ptr != NULL){
        printf("%d ",ptr->data);
        recPreOrder(ptr->left);
        recPreOrder(ptr->right);
    }
}
void recPostOrder(node_ptr ptr){
    if(ptr != NULL){
        recPostOrder(ptr->left);
        recPostOrder(ptr->right);
        printf("%d ",ptr->data);
    }
}
int main(){
    node_ptr root;
    root = NULL;
    root = insert_BT(root,2);
    root = insert_BT(root,3);
    root = insert_BT(root,4);
    root = insert_BT(root,4);
    printf("RecInOrder: ");recInOrder(root);
    printf("\nRecPreOrder: ");recPreOrder(root);
    printf("\nRecPostOrder: ");recPostOrder(root);printf("\n");
    return 0;
}
```

Output:

```
student@lpcp-23:~/Desktop/230962326/Week1$ ./q2
Key 4 already Exists!
RecInOrder: 2 3 4
RecPreOrder: 2 3 4
RecPostOrder:
4 3 2 student@lpcp-23:~/Desktop/230962326/Week1$
```

Question 3)

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{
   int data;
   struct Node *next;
}*node_ptr;
typedef struct NodeArray{
   int data;
   struct NodeArray *next;
   struct Node *right;
}*nodearr_ptr;

void GenerateGraph(){
   int nov;
   printf("Enter total no of vertices: ");scanf("%d",&nov);
   nodearr_ptr AdjacencyList;
   AdjacencyList = NULL;
   for(int i = 0;i<nov;i++){
      nodearr_ptr new_node = (nodearr_ptr) malloc(sizeof(nodearr_ptr));
      new_node->data = i;
      new_node->next = NULL;
      if(AdjacencyList == NULL)AdjacencyList = new_node;
      else{
         nodearr_ptr ptr = AdjacencyList;
```

```
        while(ptr->next != NULL)ptr = ptr->next;
        ptr->next = new_node;
    }
}
int AdjacencyMatrix[nov][nov];
for(int i=0;i<nov;i++)for(int j=0;j<nov;j++)AdjacencyMatrix[i][j] = 0;
// Adding Nodes
nodearr_ptr ptr_trav;ptr_trav = AdjacencyList;
for(int i=0; i<nov;i++){
    int deg;
    node_ptr ptr1_trav;
    printf("    Enter %d Node degree: ",i);scanf("%d",&deg);
    for(int j=0;j<deg;j++){
        int value;
        printf("       Enter %d Node %d Connection: ",i,j);scanf("%d",&value);
        node_ptr new_node = (node_ptr) malloc(sizeof(node_ptr));
        new_node->data = value;
        new_node->next = NULL;
        AdjacencyMatrix[i][value] = 1;
        if(j == 0){ptr1_trav = new_node;ptr_trav->right = new_node;}
        else{ptr1_trav->next = new_node;ptr1_trav=new_node;}
    }
    ptr_trav = ptr_trav->next;
}
//Printing Adjacency MAtrix
printf("Adjacency Matrix: \n");
for(int i=0;i<nov;i++){for(int j=0;j<nov;j++)printf("%d ",AdjacencyMatrix[i][j]);printf("\n");}

//Printing Adjacency List
printf("\n AdjacencyList: \n");
nodearr_ptr ptr;
```

```
  ptr = AdjacencyList;
  while(ptr != NULL){
    printf("%d",ptr->data);
    node_ptr ptr1 = ptr->right;
    while(ptr1->next != NULL){
      printf(" -> %d",ptr1->data);
      ptr1 = ptr1->next;
    }
    printf(" -> %d \n",ptr1->data);
    ptr = ptr->next;
  }
}

int main(){
  GenerateGraph();
  return 0;
}
```

Output:

```
E:\MIT_SEM4_DAA_LAB\Week1>q3.exe
Enter total no of vertices: 3
    Enter 0 Node degree: 1
        Enter 0 Node 0 Connection: 1
    Enter 1 Node degree: 1
        Enter 1 Node 0 Connection: 2
    Enter 2 Node degree: 0
Adjacency Matrix:
0 1 0
0 0 1
0 0 0

 AdjacencyList:
0 -> 1
1 -> 2
2
E:\MIT_SEM4_DAA_LAB\Week1>
```