DSC Lab Internals

1. Implementation of stack

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 3
int stack[5];
int top=-1;
void push(){
   if(top==MAX-1){
      printf("Stack overflow");
      return;
   }else{
      int x;
      printf("Enter element to be inserted: ");
      scanf("%d",&x);
      top++;
      stack[top]=x;
   }
}
void pop(){
   if(top==-1){
      printf("Stack underflow\n");
      return;
   }
   printf("Popped item : %d\n",stack[top]);
   top--;
}
void display(){
   if(top==-1){
      printf("Stack empty\n");
      return;
   }
   printf("Stack items are\n");
   for(int i=top;i>=0;i--){
      printf("%d\t",stack[i]);
   }
   printf("\n");
}
void main(){
```

```
    int option;
    printf("1.Push\n2.Pop\n3.Display\n4.Exit");
    while(1){
        printf("\nEnter your choice: ");
        scanf("%d",&option);
        switch(option){
            case 1:push();
            break;
            case 2:pop();
            break;
            case 3:display();
            break;
            case 4:return 0;
            default: printf("Invalid option");
            break;
        }
    }
}
```

2. **Implementation of Infix to postfix expression**

```
#define SIZE 50
#include <ctype.h>
#include <stdio.h>
char s[SIZE];
int top = -1;
void push(char elem)
{
s[++top] = elem;
}
char pop()
{
return (s[top--]);
}
int pr(char elem)
{
switch (elem)
{
case '#':       return 0;
case '(':       return 1;
case '+':
```

```
case '-':       return 2;
case '*':
case '/':
case '%':       return 3;
case '^':       return 4;
}
}
void main()
{
char infx[50], pofx[50], ch, elem;
int i = 0, k = 0;
printf("\n\nEnter Infix Expression: ");
scanf("%s", infx);
push('#');
while ((ch = infx[i++]) != '\0')
{
if (ch == '(')
push(ch);
else if (isalnum(ch))
pofx[k++] = ch;
else if (ch == ')')
{
while (s[top] != '(')
pofx[k++] = pop();
elem = pop();
} else {
while (pr(s[top]) >= pr(ch))
pofx[k++] = pop();
push(ch);
}
}
while (s[top] != '#')
pofx[k++] = pop();
pofx[k] = '\0';
printf("\n\nGiven Infix Expn: %s \nPostfix Expn: %s\n", infx, pofx);
}
```

3. **Implementation of evaluation of postfix**

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

```
#include<stdlib.h>
#include<math.h>
#define MAX 50
int stack[MAX];
char post[MAX];
int top=-1;
void pushstack(int tmp);
void calculator(char c);
void main()
{
int i;
printf("Insert a postfix notation: ");
scanf("%s",post);
for(i=0;i<strlen(post);i++)
{
if(post[i]>='0' && post[i]<='9')
{
pushstack(i);
}
if(post[i]=='+' || post[i]=='-' || post[i]=='*' || post[i]=='/' || post[i]=='^')
{
calculator(post[i]);
}
}
printf("\n\nResult : %d",stack[top]);
}
void pushstack(int tmp)
{
top++;
stack[top]=(int)(post[tmp]-48);
}
void calculator(char c)
{
int a,b,ans;
a=stack[top];
stack[top]='\0';
top--;
b=stack[top];
stack[top]='\0';
top--;
```

```
switch(c)
{
case '+': ans=b+a;
break;
case '-': ans=b-a;
break;
case '*': ans=b*a;
break;
case '/': ans=b/a;
break;
default: ans=0;
}
top++;
stack[top]=ans;
}
```

4. **Implementation of Queue**

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10
int queue[MAX];
int front=-1,rear=-1;
void enqueue(int x){
        if(rear==MAX-1)
                printf("Queue is overflow.\n");
        else{
                rear++;
                queue[rear]=x;
                printf("Successfully inserted.\n");
        }
        if(front==-1){
                front++;
        }
}
void dequeue() {
        if(front==-1){
                printf("Queue is underflow.\n");
                return;
        }
        else{
```

```
                printf("Deleted element = %d\n",queue[front]);
                if(rear==front){
                        front=-1;
                        rear=-1;
                }
                else{
                        front=front+1;
                }
        }
}
void display() {
        if(front==-1&&rear==-1){
                printf("Queue is empty.\n");
        }
        else{
                printf("Elements in the queue : ");
                for(int i = front;i<=rear;i++){
                        printf("%d ",queue[i]);
                }
                printf("\n");
        }
}
int main() {
        int op, x;
                printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
        while(1) {
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op) {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
```

```
                    case 4: return 0;
                    default: printf("Invalid Choice");
                }
        }
}
```

5. **Implementation of Circular Queue**

```
#include <stdio.h>
#include<stdlib.h>
#define SIZE 5
int items[SIZE];
int front = -1, rear = -1;
void enqueue(int element) {
if ((front==rear+1)||(front==0 && rear==SIZE-1))
printf("\n Queue is full!! \n");
else {
if (front == -1) front = 0;
rear = (rear + 1) % SIZE;
items[rear] = element;
printf("\n Inserted -> %d\n", element);
}
}
int dequeue() {
int element;
if (front==-1) {
printf("\n Queue is empty !! \n");
return;
}else{
element = items[front];
if (front == rear) {
front = -1;
rear = -1;
}else{
front = (front + 1) % SIZE;
}
printf("\n Deleted element -> %d \n", element);
return (element);
}
}
void display() {
```

```
int i;
if (front==-1)
printf(" \n Empty Queue\n");
else {
printf("\n Front -> %d ", front);
printf("\n Items -> ");
for (i = front; i != rear; i = (i + 1) % SIZE) {
printf("%d ", items[i]);
}
printf("%d ", items[i]);
printf("\n Rear -> %d \n", rear);
}
}
void main()
{
int ch;
int num1=0;
printf("1.Enqueue Operation\n2.Dequeue Operation\n3.Display the
Queue\n4.Exit\n");
while (1)
{
printf("Enter your choice of operations : ");
scanf("%d", &ch);
switch (ch)
{
case 1:
printf("\n\tEnter the element to be added to the queue: ");
scanf("%d",&num1);
enqueue(num1);
break;
case 2: dequeue();
break;
case 3: display();
break;
case 4: exit(0);
default: printf("Incorrect choice \n");
}
}
}
```

6. Implementation of Linked List

```
#include<stdio.h>
#include<stdlib.h>
struct node {
   int INFO;
   struct node *link;
};
struct node *FIRST = NULL;
struct node *LAST = NULL;
void insert(int);
int delete(int);
void print(void);
struct node *search(int);
void main() {
   int num1, num2, choice;
   struct node *location;
 while (1) {
       printf("\n\nSelect an option");
       printf("\n1 - Insert");
       printf("\n2 - Delete");
       printf("\n3 - Search");
       printf("\n4 - Print");
       printf("\n5 - Exit");
printf("\n\nEnter your choice: ");
       scanf("%d", &choice);
switch (choice) {
          case 1: {
             printf("\nEnter the element to be inserted into the linked list: ");
             scanf("%d", &num1);
             insert(num1);
             printf("\n%d successfully inserted into the linked list!", num1);
             break;
          }

   case 2:
      printf("\nEnter the element to be deleted from the linked list: ");
             scanf("%d", &num1);
             num2 = delete(num1);
             if (num2 == -9999)
      printf("\n\t%d is not present in the linked list\n\t");
```

```
            else
   printf("\n\tElement %d successfully deleted from the linked list\n\t",
num2);
            break;
case 3:
   printf("\nEnter the element to be searched: ");
   scanf("%d", &num1);
   location = search(num1);
   if (location == NULL)
   printf("\n\t%d is not present in the linked list\n\t");
            else {
if (location == LAST)
printf("\n\tElement %d is the last element in the list", num1);
else
printf("\n\tElement %d is present before element %d in the linked list",
num1, (location->link)->INFO);
            }
            break;
         case 4: print();
            break;
         case 5:exit(1);
            break;
default: printf("\nIncorrect choice. Please try again.");
            break;
      }
   }
}
void insert(int value) {
   struct node *PTR = (struct node *)malloc(sizeof(struct node));
   PTR->INFO = value;
   if (FIRST == NULL) {
      FIRST = LAST = PTR;
      PTR->link = NULL;
   } else {
      LAST->link = PTR;
      PTR->link = NULL;
      LAST = PTR;
   }
}
int delete(int value) {
```

```
    struct node *LOC, *TEMP;
    int i;
    i = value;
    LOC = search(i);
    if (LOC == NULL)
        return (-9999);
    if (LOC == FIRST) {
        if (FIRST == LAST)
            FIRST = LAST = NULL;
        else
            FIRST = FIRST->link;
        return (value);
    }
    for (TEMP = FIRST; TEMP->link != LOC; TEMP = TEMP->link);
    TEMP->link = LOC->link;
    if (LOC == LAST)
        LAST = TEMP;
    return (LOC->INFO);
}
struct node *search(int value) {
    struct node *PTR;
    if (FIRST == NULL)
        return (NULL);
    for (PTR = FIRST; PTR != LAST; PTR = PTR->link)
        if (PTR->INFO == value)
            return (PTR);
    if (LAST->INFO == value)
        return (LAST);
    else
        return (NULL);
}
void print() {
    struct node *PTR;
    if (FIRST == NULL) {
        printf("\n\tEmpty List!!");
        return;
    }
    printf("\nLinked list elements:\n");
    if (FIRST == LAST) {
        printf("\t%d", FIRST->INFO);
```

```
    return;
  }
  for (PTR = FIRST; PTR != LAST; PTR = PTR->link)
    printf("\t%d", PTR->INFO);
  printf("\t%d", LAST->INFO);
}
```

7. Implementation of stack using linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node {
   int INFO;
   struct node *link;
};

struct node *FIRST = NULL;
struct node *LAST = NULL;

void insert(int);
int delete(int);
void print(void);
struct node *search(int);

void main() {
   int num1, num2, choice;
   struct node *location;

   while (1) {
      printf("\nSelect an option");
      printf("\n1 - Insert");
      printf("\n2 - Delete");
      printf("\n3 - Search");
      printf("\n4 - Print");
      printf("\n5 - Exit");

      printf("\nEnter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1: {
            printf("\nEnter the element to be inserted into the linked list: ");
```

```
            scanf("%d", &num1);
            insert(num1);
            printf("\n%d successfully inserted into the linked list!", num1);
            break;
        }

        case 2: {
            printf("\nEnter the element to be deleted from the linked list: ");
            scanf("%d", &num1);
            num2 = delete(num1);
            if (num2 == -9999)
                printf("\n\t%d is not present in the linked list\n\t");
            else
                printf("\n\tElement %d successfully deleted from the linked
list\n\t", num2);
            break;
        }

        case 3: {
            printf("\nEnter the element to be searched: ");
            scanf("%d", &num1);
            location = search(num1);
            if (location == NULL)
                printf("\n\t%d is not present in the linked list\n\t");
            else {
                if (location == LAST)
                    printf("\n\tElement %d is the last element in the list",
num1);
                else
                    printf("\n\tElement %d is present before element %d in the
linked list", num1, (location->link)->INFO);
            }
            break;
        }

        case 4:
            print();
            break;

        case 5:
```

```
            exit(1);
            break;

        default:
            printf("\nIncorrect choice. Please try again.");
            break;
      }
   }
}

void insert(int value) {
   struct node *PTR = (struct node *)malloc(sizeof(struct node));
   PTR->INFO = value;
   if (FIRST == NULL) {
      FIRST = LAST = PTR;
      PTR->link = NULL;
   } else {
      LAST->link = PTR;
      PTR->link = NULL;
      LAST = PTR;
   }
}

int delete(int value) {
   struct node *LOC, *TEMP;
   int i;
   i = value;
   LOC = search(i);
   if (LOC == NULL)
      return (-9999);
   if (LOC == FIRST) {
      if (FIRST == LAST)
         FIRST = LAST = NULL;
      else
         FIRST = FIRST->link;
      return (value);
   }
   for (TEMP = FIRST; TEMP->link != LOC; TEMP = TEMP->link);
   TEMP->link = LOC->link;
   if (LOC == LAST)
```

```
        LAST = TEMP;
    return (LOC->INFO);
}

struct node *search(int value) {
    struct node *PTR;
    if (FIRST == NULL)
        return (NULL);
    for (PTR = FIRST; PTR != LAST; PTR = PTR->link)
        if (PTR->INFO == value)
            return (PTR);
    if (LAST->INFO == value)
        return (LAST);
    else
        return (NULL);
}

void print() {
    struct node *PTR;
    if (FIRST == NULL) {
        printf("\n\tEmpty List!!");
        return;
    }
    printf("\nLinked list elements:\n");
    if (FIRST == LAST) {
        printf("\t%d", FIRST->INFO);
        return;
    }
    for (PTR = FIRST; PTR != LAST; PTR = PTR->link)
        printf("\t%d", PTR->INFO);
    printf("\t%d", LAST->INFO);
}
```

8. **Implementation of doubly linked list**

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

struct dlnode
{
  int info;
```

```
  struct dlnode* next;
  struct dlnode* prev;
};

struct dlnode* first = NULL;
struct dlnode* last = NULL;

void insert(int value) {
  struct dlnode* ptr = (struct dlnode*)malloc(sizeof(struct dlnode));
  ptr->info = value;

  if (first == NULL) {
    first = last = ptr;
    ptr->next = NULL;
    ptr->prev = NULL;
  } else {
    last->next = ptr;
    ptr->next = NULL;
    ptr->prev = last;
    last = ptr;
  }
}

struct dlnode* search(int value) {
  struct dlnode* ptr;

  if (first == NULL)
    return(NULL);

  if (first == last && first->info == value)
    return(first);

  for (ptr = first; ptr != NULL; ptr = ptr->next)
    if (ptr->info == value)
      return(ptr);

  if (last->info == value)
    return(last);
  else
    return(NULL);
```

```
}

int delete(int value) {
  struct dlnode* loc, * temp;
  int i = value;
  loc = search(i);

  if (loc == NULL)
    return(-9999);

  if (loc == first) {
    if (first == last)
      first = last = NULL;
    else {
      first->next->prev = NULL;
      first = first->next;
    }
    return(value);
  }

  for (temp = first; temp->next != loc; temp = temp->next);

  if (loc == last) {
    last = temp;
    temp->next = NULL;
  } else {
    temp->next = loc->next;
    loc->next->prev = temp;
  }

  return(loc->info);
}

void print() {
  struct dlnode* ptr;

  if (first == NULL) {
    printf("empty list\n");
    return;
  }
```

```
  for (ptr = first; ptr != last; ptr = ptr->next) {
    printf(" %d ", ptr->info);
  }

  printf(" %d ", last->info);
}

void main() {
  int n1, n2, choice;
  struct dlnode* location;

  while (1) {
    printf("\n1.insert\n2.delete\n3.search\n4.print\n5.exit\nenter choice:");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        printf("\ninsert element:");
        scanf("%d", &n1);
        insert(n1);
        printf("\nelement %d inserted",n1);
        break;

      case 2:
        printf("\ndelete element:");
        scanf("%d",&n1);
        n2 = delete(n1);
        if (n2 == -9999)
          printf("\n%d not in linked list", n1);
        else
          printf("\n%d deleted", n2);
        break;

      case 3:
        printf("\nsearch element:");
        scanf("%d", &n1);
        location = search(n1);

        if (location == NULL)
```

```
        printf("\n%d not in linked list", n1);
      else {
       if (location == last)
        printf("\n%d = last element",n1);
       else
        printf("\n%d present before %d", n1, (location->next)->info);
      }
      break;

    case 4:
      print();
      break;

    case 5:
      exit(0);
      break;

    default:
      printf("\ninvalid input");
      break;
   }
  }
}
```

9. Implementation of Binary search tree

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* left;
   struct Node* right;
};
struct Node* createNode(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->left = NULL;
   newNode->right = NULL;
   return newNode;
}
```

```
struct Node* insert(struct Node* root, int key) {
   if (root == NULL) {
      return createNode(key);
   }
if (key < root->data) {
      root->left = insert(root->left, key);
   } else if (key > root->data) {
      root->right = insert(root->right, key);
   }
return root;
}
struct Node* minValueNode(struct Node* node) {
   struct Node* current = node;
while (current->left != NULL) {
      current = current->left;
   }
return current;
}
struct Node* deleteNode(struct Node* root, int key) {
   if (root == NULL) {
      printf("Element not found\n");
      return root;
   }
if (key < root->data) {
      root->left = deleteNode(root->left, key);
   } else if (key > root->data) {
      root->right = deleteNode(root->right, key);
   } else {
      if (root->left == NULL) {
         struct Node* temp = root->right;
         free(root);
         return temp;
      } else if (root->right == NULL) {
         struct Node* temp = root->left;
         free(root);
         return temp;
      }

struct Node* temp = minValueNode(root->right);
root->data = temp->data;
```

```
root->right = deleteNode(root->right, temp->data);
}
return root;
}
struct Node* search(struct Node* root, int key) {
if (root == NULL || root->data == key) {
        return root;
    }
if (key < root->data) {
    return search(root->left, key);
    }
return search(root->right, key);
}
void inOrderTraversal(struct Node* root) {
    if (root != NULL) {
        inOrderTraversal(root->left);
        printf("%d ", root->data);
        inOrderTraversal(root->right);
    }
}
void takeUserInput(struct Node** root, int n) {
    int value;
for (int i = 0; i < n; ++i) {
        printf("Enter a value to insert into the BST: ");
        scanf("%d", &value);
        *root = insert(*root, value);
    }
}
int main() {
    struct Node* root = NULL;
    int n;
printf("Enter the number of values to insert into the BST: ");
scanf("%d", &n);
takeUserInput(&root, n);
printf("In-order traversal of the BST: ");
inOrderTraversal(root);
printf("\n");
int keyToDelete;
printf("Enter a key to delete from the BST: ");
scanf("%d", &keyToDelete);
```

```
root = deleteNode(root, keyToDelete);
printf("In-order traversal after deleting %d: ", keyToDelete);
inOrderTraversal(root);
printf("\n");
return 0;
}
```

10. Implementation of circular linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node {
   int INFO;
   struct node *link;
};
struct node *FIRST = NULL;
struct node *LAST = NULL;
void insert(int);
int delete(int);
void print(void);
struct node *search(int);
void main() {
   int num1, num2, choice;
   struct node *location;
 while (1) {
       printf("\n\nSelect an option");
       printf("\n1 - Insert");
       printf("\n2 - Delete");
       printf("\n3 - Search");
       printf("\n4 - Print");
       printf("\n5 - Exit");
printf("\n\nEnter your choice: ");
       scanf("%d", &choice);
switch (choice) {
          case 1: {
             printf("\nEnter the element to be inserted into the linked list: ");
             scanf("%d", &num1);
             insert(num1);
             printf("\n%d successfully inserted into the linked list!", num1);
             break;
```

```
        }

    case 2:
        printf("\nEnter the element to be deleted from the linked list: ");
            scanf("%d", &num1);
            num2 = delete(num1);
            if (num2 == -9999)
        printf("\n\t%d is not present in the linked list\n\t");
            else
    printf("\n\tElement %d successfully deleted from the linked list\n\t",
num2);
            break;
case 3:
    printf("\nEnter the element to be searched: ");
    scanf("%d", &num1);
    location = search(num1);
    if (location == NULL)
    printf("\n\t%d is not present in the linked list\n\t");
            else {
if (location == LAST)
printf("\n\tElement %d is the last element in the list", num1);
else
printf("\n\tElement %d is present before element %d in the linked list",
num1, (location->link)->INFO);
            }
            break;
        case 4: print();
            break;
        case 5:exit(1);
            break;
default: printf("\nIncorrect choice. Please try again.");
            break;
        }
    }
}
void insert(int value) {
    struct node *PTR = (struct node *)malloc(sizeof(struct node));
    PTR->INFO = value;
    if (FIRST == NULL) {
        FIRST = LAST = PTR;
```

```
        PTR->link = NULL;
    } else {
        LAST->link = PTR;
        PTR->link = NULL;
        LAST = PTR;
    }
}
int delete(int value) {
    struct node *LOC, *TEMP;
    int i;
    i = value;
    LOC = search(i);
    if (LOC == NULL)
        return (-9999);
    if (LOC == FIRST) {
        if (FIRST == LAST)
            FIRST = LAST = NULL;
        else
            FIRST = FIRST->link;
        return (value);
    }
    for (TEMP = FIRST; TEMP->link != LOC; TEMP = TEMP->link);
    TEMP->link = LOC->link;
    if (LOC == LAST)
        LAST = TEMP;
    return (LOC->INFO);
}
struct node *search(int value) {
    struct node *PTR;
    if (FIRST == NULL)
        return (NULL);
    for (PTR = FIRST; PTR != LAST; PTR = PTR->link)
        if (PTR->INFO == value)
            return (PTR);
    if (LAST->INFO == value)
        return (LAST);
    else
        return (NULL);
}
void print() {
```

```
    struct node *PTR;
    if (FIRST == NULL) {
        printf("\n\tEmpty List!!");
        return;
    }
    printf("\nLinked list elements:\n");
    if (FIRST == LAST) {
        printf("\t%d", FIRST->INFO);
        return;
    }
    for (PTR = FIRST; PTR != LAST; PTR = PTR->link)
        printf("\t%d", PTR->INFO);
    printf("\t%d", LAST->INFO);
}
```

11. Implementation of Binary tree

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
struct Node {
    int data;
    struct Node* left,* right;
};
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
int countNodes(struct Node* root) {
    if (root == NULL) {
        return 0;
    }
    return 1 + countNodes(root->left) + countNodes(root->right);
}
int countLeafNodes(struct Node* root) {
    if (root == NULL) {
        return 0;
```

```
    }
    if (root->left == NULL && root->right == NULL) {
        return 1;
    }
    return countLeafNodes(root->left) + countLeafNodes(root->right);
}
bool areEqual(struct Node* root1, struct Node* root2) {
    if (root1 == NULL && root2 == NULL) {
        return true;
    }
    if (root1 != NULL && root2 != NULL) {
        return (
            root1->data == root2->data &&
            areEqual(root1->left, root2->left) &&
            areEqual(root1->right, root2->right)
            );
    }
    return false;
}
void inOrderTraversal(struct Node* root) {
    if (root != NULL) {
        inOrderTraversal(root->left);
        printf("%d ", root->data);
        inOrderTraversal(root->right);
    }
}
void preOrderTraversal(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preOrderTraversal(root->left);
        preOrderTraversal(root->right);
    }
}
void postOrderTraversal(struct Node* root) {
    if (root != NULL) {
        postOrderTraversal(root->left);
        postOrderTraversal(root->right);
        printf("%d ", root->data);
    }
}
```

```
int main() {
    struct Node* root1 = createNode(1);
    root1->left = createNode(2);
    root1->right = createNode(3);
    root1->left->left = createNode(4);
    root1->left->right = createNode(5);

    struct Node* root2 = createNode(1);
    root2->left = createNode(2);
    root2->right = createNode(3);
    root2->left->left = createNode(4);
    root2->left->right = createNode(5);

    printf("Number of nodes in tree 1: %d\n", countNodes(root1));

    printf("Number of leaf nodes in tree 1: %d\n",
countLeafNodes(root1));

    printf("Number of nodes in tree 2: %d\n", countNodes(root2));

    printf("Number of leaf nodes in tree 2: %d\n",
countLeafNodes(root2));

    if (areEqual(root1, root2)) {
        printf("Tree 1 and Tree 2 are equal.\n");
    }
    else {
        printf("Tree 1 and Tree 2 are not equal.\n");
    }
    printf("In-order traversal of tree 1: ");
    inOrderTraversal(root1);
    printf("\n");

    printf("Pre-order traversal of tree 1: ");
    preOrderTraversal(root1);
    printf("\n");

    printf("Post-order traversal of tree 1: ");
    postOrderTraversal(root1);
    printf("\n");
```

```
    printf("In-order traversal of tree 2: ");
    inOrderTraversal(root2);
    printf("\n");

    printf("Pre-order traversal of tree 2: ");
    preOrderTraversal(root2);
    printf("\n");

    printf("Post-order traversal of tree 2: ");
    postOrderTraversal(root2);
    printf("\n");

    return 0;
}
```

12. Implementation of queue using linked list

```
#include<stdio.h>
#include<stdlib.h>
struct queue
{
        int element;
        struct queue* next;
};
struct queue* front = NULL;
struct queue* rear = NULL;
void insert(int value) {
        struct queue* ptr;
        ptr = (struct queue*)malloc(sizeof(struct queue));
        ptr->element = value;
        if (front == NULL) {
        front = rear = ptr;
        ptr->next = NULL;
        }
        else {
                rear->next = ptr;
                ptr->next = NULL;
                rear=ptr;
        }
```

```
}
int del(int i) {
        if (front == NULL)
                return(-9999);
        else {
                i = front->element;
                front = front->next;
                return(i);
        }
}
void display() {
        struct queue* ptr = front;
        if (front == NULL){
                printf("\nEmpty queue\n");
                return;}
        else {
                printf("\nelements:");
                while (ptr != NULL) {
                        printf(" %d ", ptr->element);
                        ptr=ptr->next;
                }printf("\n");
        }
}
void main() {
        int n1, n2, choice;
        printf("\n1.insert\n2.delete\n3.display\n4.exit\n");
        while (1) {
                printf("enter choice:");
                scanf("%d", &choice);
                switch (choice) {
                case 1: printf("\ninsert element:");
                        scanf("%d", &n1);
                        insert(n1);
                        break;
                case 2: n2 = del(n1);
                        if (n2 == -9999)
                                printf("\nempty queue\n");
                        else
                                printf("\nelement deleted:%d\n",n2);
                        break;
```

```
                case 3: display();
                        break;
                case 4: exit(1);
                        break;
                default: printf("\ninvalid input\n");
                        break;
                }
        }
}
```