1. Write a C program to implement the following operations on to a 1D Array: a. INSERT b. DELETE c. SEARCH d. TRAVERSE

#include <stdio.h>

#define MAX\_SIZE 100

int arr[MAX\_SIZE];

int size = 0;

void insert(int value) {

    if (size == MAX\_SIZE) {

        printf("Array is full. Cannot insert.\n");

        return;

    }

    arr[size] = value;

    size++;

}

int delete(int index) {

    if (index < 0 || index >= size) {

        printf("Invalid index.\n");

        return -1;

    }

    int value = arr[index];

    for (int i = index; i < size - 1; i++) {

        arr[i] = arr[i + 1];

    }

    size--;

    return value;

}

int search(int value) {

    for (int i = 0; i < size; i++) {

        if (arr[i] == value) {

            return i;

        }

    }

    return -1;

}

void traverse() {

    for (int i = 0; i < size; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

}

int main() {

    insert(10);

    insert(20);

    traverse();

    int del = delete(0);

    printf("Deleted: %d\n", del);

    int idx = search(20);

    if (idx!= -1)

    printf("Found at index: %d\n", idx);

    return 0;

2. Write a C program to implement Self-referential Structure.

#include <stdio.h>

#include <stdlib.h>

typedef struct sll {

int data;

struct sll \*ptr;

} node;

node \*start = NULL;

void main() {

char ch;

int value;

node \*newnode, \*temp;

do {

newnode = (node\*)malloc(sizeof(node));

if (newnode == NULL) {

printf("Memory is not created");

} else {

printf("Enter the value: ");

scanf("%d", &value);

newnode->data = value;

newnode->ptr = NULL;

if (start == NULL) {

start = newnode;

} else {

temp = start;

while (temp->ptr != NULL) {

temp = temp->ptr;

}

temp->ptr = newnode;

}

}

printf("Do you want to continue (Y/N): ");

scanf(" %c", &ch);

} while (ch == 'Y' || ch == 'y');

temp = start;

printf("\nNodes are: ");

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->ptr;

}

printf("\n");

}

//DMA

#include <stdio.h>

#include <stdlib.h>

int main() {

int\* arr = (int\*) malloc(5 \* sizeof(int));

if (arr == NULL) {

printf("Memory allocation failed!\n");

return 1;

}

for (int i = 0; i < 5; i++) {

arr[i] = i + 1;

}

printf("Array contents:\n");

for (int i = 0; i < 5; i++) {

printf("%d ", arr[i]);

}

printf("\n");

free(arr);

return 0;

}

4. Write a C program to implement Single linked list i) Insertion ii) Deletion iii) Display

#include <stdio.h>

#include <stdlib.h>

struct node {

    int data;

    struct node\* next;

};

struct node\* start = NULL;

void create();

void insert();

void delete\_node();

void display();

void traverse();

int main() {

    create();

}

void create() {

    int n, choice;

    char ch;

    printf("Enter the number of nodes: ");

    scanf("%d", &n);

    struct node\* temp;

    for (int i = 1; i <= n; i++) {

        struct node\* newnode = (struct node\*)malloc(sizeof(struct node));

        printf("Enter data for %d NODE: ", i);

        scanf("%d", &newnode->data);

        newnode->next = NULL;

        if (start == NULL) {

            start = newnode;

            temp = newnode;

        } else {

            temp->next = newnode;

            temp = newnode;

        }

    }

    do {

        printf("1. INSERT\n2. DELETE\n3. DISPLAY\n4. TRAVERSE\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                insert();

                break;

            case 2:

                delete\_node();

                break;

            case 3:

                display();

                break;

            case 4:

                traverse();

                break;

            default:

                printf("Invalid choice\n");

        }

        printf("Do you want to continue (Y/N)? ");

        fflush(stdin);

        scanf(" %c", &ch);

    } while (ch == 'y' || ch == 'Y');

}

void insert() {

    int pos, val, i = 1;

    printf("Enter position to insert: ");

    scanf("%d", &pos);

    printf("Enter data of node for insertion: ");

    scanf("%d", &val);

    struct node\* newnode = (struct node\*)malloc(sizeof(struct node));

    newnode->data = val;

    struct node\* temp = start;

    if (pos == 1) {

        newnode->next = start;

        start = newnode;

    } else {

        while (i++ < pos - 1 && temp != NULL) {

            temp = temp->next;

        }

        if (temp == NULL) {

            printf("Invalid position\n");

            return;

        }

        newnode->next = temp->next;

        temp->next = newnode;

    }

    display();

}

void delete\_node() {

    int pos, i = 1;

    printf("Enter position of the node to delete: ");

    scanf("%d", &pos);

    struct node\* temp = start;

    struct node\* prev = NULL;

    if (start == NULL) {

        printf("Linked list is empty\n");

    }

    if (pos == 1) {

        start = temp->next;

        free(temp);

    } else {

        while (i++ < pos && temp != NULL) {

            prev = temp;

            temp = temp->next;

        }

        if (temp == NULL) {

            printf("Invalid position\n");

        }

        prev->next = temp->next;

        free(temp);

    }

    display();

}

void display() {//

    int i = 1;

    struct node\* temp = start;

    printf("Elements of linked list are:\n");

    if (start == NULL) {

        printf("Linked list is empty\n");

    }

    while (temp != NULL) {

        printf("NODE %d data is %d\n", i++, temp->data);

        temp = temp->next;

    }

    printf("\n");

}

void traverse() {

    int i = 1;

    struct node\* temp = start;

    printf("Traversing forward:\n");

    while (temp != NULL) {

        printf("NODE %d data is %d\n", i++, temp->data);

        temp = temp->next;

    }

    printf("\n");

}

5. Write a function to reverse the nodes of a Single linked list

#include<stdio.h>

#include<stdlib.h>

typedef struct sll {

    int data;

    struct sll \*ptr;

} node;

node \*start = NULL;

node \*last = NULL;

int n = 0;

void create();

void reverse();

void transverse();

int main() {

    char c;

    do {

        int i;

        printf("Choose From the options\n1.Create\n2.Reverse\n");

        scanf("%d", &i);

        switch (i) {

            case 1: create(); break;

            case 2: reverse();  break;

            default: printf("Wrong Option");

        }

        printf("\nDo you want to continue (Y/N): ");

        scanf(" %c", &c);

    } while (c == 'Y' || c == 'y');

    return 0;

}

void create() {

    char ch;

    int value;

    node \*newnode, \*temp;

    do {

        newnode = (node\*)malloc(sizeof(node));

        if (newnode == NULL) {

            printf("Memory is not allocated");

        }

        printf("Enter the value: ");

        scanf("%d", &value);

        newnode->data = value;

        newnode->ptr = NULL;

        if (start == NULL) {

            start = newnode;

            last = newnode;

        }

        else {

            last->ptr = newnode;

            last = newnode;

        }

        n++;

        printf("Do you want to continue (Y/N): ");

        scanf(" %c", &ch);

    } while (ch == 'Y' || ch == 'y');

    printf("\nNo. of nodes is %d\n", n);

}

void reverse() {

    if (start == NULL )

        printf("LInked list is empty");

    node \*prev = NULL;

    node \*temp = start;

    node \*t;

    while (temp != NULL) {

        t = temp->ptr;

        temp->ptr = prev;

        prev = temp;

        temp= t;

    }

    start=prev;

transverse();

}

void transverse() {

    printf("\nReversed list: ");

    node \*temp = start;

    while (temp != NULL) {

        printf("%d ", temp->data);

        temp = temp->ptr;

    }

    printf("\n");

}

6. Write a program that implement Stack (its operations) using Arrays

#include <stdio.h>

#define MAX 20

int top = -1, st[MAX];

void push(int);

int pop();

void print();

int isEmpty();

int isFull();

int main() {

char ch;

int val, choice;

do {

printf("MENU\n");

printf("1. Push\n2. Pop\n3. Print\n4. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

if (isFull())

printf("Stack is full\n");

else {

printf("Enter the element: ");

scanf("%d", &val);

push(val);

}

break;

case 2:

if (isEmpty())

printf("Stack is empty\n");

else {

val = pop();

printf("Value is deleted: %d\n", val);

}

break;

case 3:

if (isEmpty())

printf("Stack is empty\n");

else {

print();

}

break;

case 4:

printf("Exiting...\n");

break;

default:

printf("Wrong option\n");

}

if (choice != 4) {

printf("Do you want to continue (Y/N): ");

scanf(" %c", &ch);

} else {

ch = 'n';

}

} while (ch == 'y' || ch == 'Y');

return 0;

}

void push(int a) {

st[++top] = a;

}

int pop() {

return st[top--];

}

void print() {

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

printf("%d ", st[i]);

}

printf("\n");

}

int isEmpty() {

return top == -1;

}

int isFull() {

return top == MAX - 1;

}

7. Write a program that implement Circular Queue (its operations) using Arrays

#include <stdio.h>

#include <stdlib.h>

#define MAX 3

int front = -1, rear = -1, queue[MAX];

void insert(int val);

int delete();

void print();

int isEmpty();

int isFull();

int main() {

int val, choice;

char cont;

do {

printf("\nMENU\n");

printf("1. Insert\n2. Delete\n3. Print\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

if (isFull())

printf("Queue is full.\n");

else {

printf("Enter the element: ");

scanf("%d", &val);

insert(val);

}

break;

case 2:

if (isEmpty())

printf("Queue is empty.\n");

else {

val = delete();

printf("Deleted value: %d\n", val);

}

break;

case 3:

if (isEmpty())

printf("Queue is empty.\n");

else

print();

break;

case 4:

exit(0);

default:

printf("Wrong option.\n");

}

printf("Do you want to continue? (Y/N): ");

scanf(" %c", &cont);

} while (cont == 'y' || cont == 'Y');

return 0;

}

void insert(int val) {

if (front == -1)

front = 0;

rear = (rear + 1) % MAX;

queue[rear] = val;

}

int delete() {

int val = queue[front];

if (front == rear)

front = rear = -1;

else

front = (front + 1) % MAX;

return val;

}

void print() {

int i = front;

printf("Queue contents: ");

do {

printf("%d ", queue[i]);

i = (i + 1) % MAX;

} while (i != (rear + 1) % MAX);

printf("\n");

}

int isEmpty() {

return (front == -1);

}

int isFull() {

return ((rear + 1) % MAX == front);

}

v8. Write C programs to implement Stack ADT using Linked List

#include <stdio.h>

#include <stdlib.h>

void push();

void pop();

void display();

typedef struct node

{

int val;

struct node \*next;

}node;

struct node \*start=NULL;

void main ()

{

    int choice;

    char ch;

    do

    {

        printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");

        printf("\n Enter your choice \n");

        scanf("%d",&choice);

        switch(choice)

        {

            case 1:

            {

                push();

                break;

            }

            case 2:

            {

                pop();

                break;

            }

            case 3:

            {

                display();

                break;

            }

            case 4:

            {

                printf("Exiting....");

                break;

            }

            default:

            {

                printf("Please Enter valid choice ");

            }

    };

        printf("\nDo you want to continue");

        scanf(" %c",&ch);

    }while(ch=='y'||ch=='Y');

}

void push ()

{

    int val;

    node \*newnode;

    newnode= (node\*)malloc(sizeof(node));

    if(newnode == NULL)

    {

        printf("not able to push the element");

    }

    else

    {

        printf("Enter the value");

        scanf("%d",&val);

        if(start==NULL)

        {

            newnode->val = val;

            newnode-> next = NULL;

            start=newnode;

        }

        else

        {

            newnode->val = val;

            newnode->next = start;

            start=newnode;

        }

        printf("Item pushed");

    }

}

void pop()

{

    int i;

    node \*temp;

    if (start== NULL)

    {

        printf("Stack is Empty");

    }

    else

    {

        i = start->val;

        temp= start;

        start = start->next;

        free(temp);

        printf("Item popped");

    }

}

void display()

{

    int i;

    node \*temp;

    temp=start;

    if(temp== NULL)

    {

        printf("Stack is empty\n");

    }

    else

    {

        printf("Printing Stack elements \n");

        while(temp!=NULL)

        {

            printf("%d\n",temp->val);

            temp= temp->next;

        }

    }

}

9. Write a program that implement Queue (its operations) using Arrays

#include<stdio.h>

#define MAX 20

int front = 0,rear=0,st[MAX] ;

void insert(int);

int lqdelete();

void print();

int empty();

int full ();

void main ()

{

    char ch;

    int val,choice;

    do

    {

        printf("MENU\n");

        printf("1.insert\n2.delete\n3.print\n4.Exit\n");

        scanf("%d",&choice);

        if(choice==4)

            printf("Enter correct option");

        else

        {

            switch(choice)

            {

                case 1 :

                    if(full())

                        printf("Queue is full ");

                    else{

                        printf("Enter the element ");

                        scanf("%d",&val);

                        insert(val);

                    }break;

                case 2:

                    if(empty())

                        printf("Queue is empty ");

                    else{

                        val=lqdelete();

                        printf("Val is deleted %d",val);

                    }break;

                case 3 :

                    if(empty())

                        printf("Queue is empty ");

                    else

                    {

                        print();

                    }break;

                default : printf("Wrong option");

            }

        }

        printf("\nDo you want to continue");

        scanf(" %c",&ch);

    }while(ch=='y'||ch=='Y');

}

void insert(int a)

{

    st[rear]=a;

    rear++;

}

int lqdelete()

{

   int a=st[front];

    front++;

    return a;

}

void print()

{

    int i =0;

    for(i=front;i<rear;i++)

    {

        printf("%d ",st[i]);

        i++;

    }

}

int empty()

{

    if(front==rear)

        return 1;

    else

        return 0 ;

}

int full()

{

    if(rear>=MAX)

        return 1;

    else

        return 0;

}

10. Write C programs to implement Queue ADT using Linked List

#include<stdio.h>

typedef struct node

{

    int data;

    struct node \*next;

}node;

node \*front;

 node \*rear;

void insert();

void lqdelete();

void display();

void main ()

{

    int choice;

    char ch;

    do

    {

        printf("1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");

        printf("Enter your choice ?\n");

        scanf("%d",& choice);

        switch(choice)

        {

            case 1:

            insert();

            break;

            case 2:

            lqdelete();

            break;

            case 3:

            display();

            break;

            case 4:

            exit(0);

            break;

            default:

            printf("Enter valid choice??\n");

        }

         printf("\nDo you want to continue");

        scanf(" %c",&ch);

    }while(ch=='y'||ch=='Y');

}

void insert()

{

     node \*newnode;

    int i;

    newnode= (node \*) malloc (sizeof(node));

    if(newnode == NULL)

    {

        printf("\nMem not allocated\n");

        return;

    }

    else

    {

        printf("\nEnter value?\n");

        scanf("%d",&i);

        newnode-> data = i;

        if(front == NULL)

        {

            front = newnode;

            rear = newnode;

            front -> next = NULL;

            rear -> next = NULL;

        }

        else

        {

            rear -> next = newnode;

            rear = newnode;

            rear->next = NULL;

        }

    }

}

void lqdelete ()

{

    node \*temp;

    if(front == NULL)

    {

        printf("\nUNDERFLOW\n");

        return;

    }

    else

    {

        temp= front;

        front = front -> next;

        free(temp);

    }

}

void display()

{

    node \*temp;

    temp = front;

    if(front == NULL)

    {

        printf("\nEmpty queue\n");

    }

    else

    {   printf("\nprinting values\n");

        while(temp!= NULL)

        {

            printf("%d  ",temp-> data);

            temp = temp -> next;

        }

    }

}

11. Write a C program to implement different hash methods

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define TS 1000

int hashMethod = 0;

typedef struct HE{

    int key;

    int value;

} HE;

HE \*ht[TS];

void HTE(){

    int i;

    for(i=0;i<TS;i++) {

        ht[i]=NULL;

    }

}

int divisionHash(int key) {

    return key%TS;

}

int multiplicativeHash(int key) {

    float a=0.6180339887;

    float temp=key\*a;

    temp-=(int)temp;

    return (int)(TS\*temp);

}

int midSquareHash(int key) {

    int s=key\*key;

    int n=0;

    int temp=key;

    while(temp!=0){

        temp/=10;

        n++;

    }

    int r=0,i;

    for(i=1; i<=n/2; i++)

    {

        s = s/10;

    }

    return s%TS;

}

int foldingHash(int key) {

    int temp = key;

    int n=0;

    while(temp!=0)

    {

        temp/=10;

        n++;

    }

    int arr[n];

    int i=0;

    while(key!=0)

    {

        arr[i] = key%10;

        key/=10;

        i++;

    }

    int sum=0;

    for(int i=0; i<n; i++)

    {

        sum += arr[i];

    }

    return sum%TS;

}

void insert(int key,int value,int (\*hf)(int)){

    int i=hf(key);

    while(ht[i]!=NULL && ht[i]->key!=key){

        i=(i+1)%TS;

    }

    if(ht[i]!=NULL){

        ht[i]->value=value;

    }else{

        HE \*e=(HE\*)malloc(sizeof(HE));

        e->key=key;

        e->value=value;

        ht[i]=e;

    }

}

void print()

{

    for(int i=0; i<TS; i++)

    {

        if(ht[i]!=NULL)

        {

            printf("Key: %d\t", ht[i]->key);

            printf("Value: %d\t", ht[i]->value);

            printf("Index: %d\n", i);

        }

    }

}

int main()

{

    int choice;

    char ch;

    int key, value;

    do

    {

        printf("Hashing Choices:\n1 - Division\n2 - Multiplication\n3 - Folding\n4 - Mid Square\n");

        printf("Enter your choice: ");

        scanf("%d", &hashMethod);

        printf("Operations Available:\n1 - Insert\n2 - Print\n3 - Exit\n");

        scanf("%d", &choice);

        switch(choice)

        {

            case 1:

                printf("Enter key and value to insert: ");

                scanf("%d %d",&key,&value);

                switch (hashMethod)

                {

                    case 1:

                        insert(key, value, divisionHash);

                        break;

                    case 2:

                        insert(key, value, multiplicativeHash);

                        break;

                    case 3:

                        insert(key, value, foldingHash);

                        break;

                    case 4:

                        insert(key, value, midSquareHash);

                        break;

                }

                break;

            case 2:

                print();

                break;

            case 3:

                return 0;

            default:

                printf("Invalid Choice!\n");

        }

        printf("Would you like to continue(Y/N): \n");

        scanf(" %c", &ch); *// Added space before %c to consume newline*

    } while(ch == 'y' || ch == 'Y');

    return 0;

}

12. Write a C program to implement Quadratic probing collision resolving technique

#include <stdio.h>

#include <stdlib.h>

#define TS 10000

typedef struct HE {

    int key;

} HE;

HE \*ht[TS];

void initializeHashTable() {

    int i;

    for(i = 0; i < TS; i++) {

        ht[i] = NULL;

    }

}

void display() {

    for(int i = 0; i < TS; i++) {

        if(ht[i] != NULL) {

            printf("Key: %d\tIndex: %d\n", ht[i]->key, i);

        }

    }

}

int divisionHash(int key) {

    int s = key % TS;

    int i = 0;

    int pos;

    while (i < TS) {

        pos = (s + i \* i) % TS;

        if (ht[pos] == NULL) {

            return pos;

        }

        i++;

    }

    return -1;

}

void insert(int key) {

    int pos = divisionHash(key);

    if (pos != -1 && ht[pos] == NULL) {

        ht[pos] = malloc(sizeof(HE));

        ht[pos]->key = key;

    } else {

        printf("Insertion failed. No empty slot available.\n");

    }

}

int main() {

    int choice, key;

    char ch;

    initializeHashTable();

    do {

        printf("Hashing Choices:\n1 - Insert\n2 - Print\n");

        printf("Enter your choice: ");

        scanf(" %d", &choice);

        switch (choice) {

            case 1:

                printf("Enter key to insert: ");

                scanf("%d", &key);

                insert(key);

                break;

            case 2:

                display();

                break;

            default:

                printf("Invalid choice.\n");

                break;

        }

        printf("Do you want to continue? (y/n): ");

        scanf(" %c", &ch);

    } while (ch == 'y' || ch == 'Y');

    return 0;

}

13. Write a C program to implement Linear Probing collision resolving technique.

#include <stdio.h>

#include <stdlib.h>

#define TS 10000

typedef struct HE {

    int key;

} HE;

HE \*ht[TS];

void initializeHashTable() {

    int i;

    for(i = 0; i < TS; i++) {

        ht[i] = NULL;

    }

}

void display() {

    for(int i = 0; i < TS; i++) {

        if(ht[i]!= NULL) {

            printf("Key: %d\tIndex: %d\n", ht[i]->key, i);

        }

    }

}

int divisionHash(int key) {

    int s = key % TS;

    int i = 0;

    int pos;

    while (1) {

        pos = (s + i) % TS;

        if (ht[pos] == NULL) {

            return pos;

        }

        i++;

    }

}

void insert(int key) {

    int pos = divisionHash(key);

    if (pos!= -1 && ht[pos] == NULL) {

        ht[pos] = malloc(sizeof(HE));

        ht[pos]->key = key;

    } else {

        printf("Insertion failed. No empty slot available.\n");

    }

}

int main() {

    int choice, key;

    char ch;

    initializeHashTable();

    do {

        printf("Hashing Choices:\n1 - Insert\n2 - Print\n");

        printf("Enter your choice: ");

        scanf(" %d", &choice);

        switch (choice) {

            case 1:

                printf("Enter key to insert: ");

                scanf("%d", &key);

                insert(key);

                break;

            case 2:

                display();

                break;

            default:

                printf("Invalid choice.\n");

                break;

        }

        printf("Do you want to continue? (y/n): ");

        scanf(" %c", &ch);

    } while (ch == 'y' || ch == 'Y');

    return 0;

}

14. Write a C program to find height of a Binary tree

#include <stdio.h>

#include <stdlib.h>

// Definition of the TreeNode structure

typedef struct TreeNode {

int value;

struct TreeNode\* left;

struct TreeNode\* right;

} TreeNode;

// Function to calculate the height of the tree

int height(TreeNode\* node) {

if (node == NULL) {

return 0; // Conventionally, height of an empty tree is 0

}

int leftHeight = height(node->left);

int rightHeight = height(node->right);

return (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;

}

// Function to create a new tree node

TreeNode\* newNode(int value) {

TreeNode\* node = (TreeNode\*) malloc(sizeof(TreeNode));

node->value = value;

node->left = NULL;

node->right = NULL;

return node;

}

// Function to insert a value into the BST

TreeNode\* insert(TreeNode\* root, int value) {

if (root == NULL) {

return newNode(value);

}

if (value < root->value) {

root->left = insert(root->left, value);

} else if (value > root->value) {

root->right = insert(root->right, value);

}

return root;

}

// Function to print the tree in pre-order traversal

void printPreOrder(TreeNode\* node) {

if (node == NULL) {

return;

}

printf("%d ", node->value);

printPreOrder(node->left);

printPreOrder(node->right);

}

// Main function

int main() {

TreeNode\* root = NULL;

// Insert values into the BST

root = insert(root, 50);

insert(root, 30);

insert(root, 20);

insert(root, 40);

insert(root, 70);

insert(root, 60);

insert(root, 80);

// Calculate the height of the tree

int heightOfTree = height(root);

printf("Height of the binary tree is: %d\n", heightOfTree);

// Print the tree in pre-order traversal

printf("Pre-order traversal of the binary tree is: ");

printPreOrder(root);

printf("\n");

return 0;

}

15. Write a program that uses functions to perform the following operations on doubly linked list: i) Creation ii) Insertion iii) Deletion iv) Traversal

#include <stdio.h>

#include<stdlib.h>

struct node

{

    int data;

    struct node\* prev;

    struct node\* next;

};

struct node\* start = NULL;

void create();

void insert();

void delete\_node();

void display();

void traverse();

int main()

{

    create();

}

void create()

{

    int n, choice;

    char ch;

    printf("Enter the number of nodes: ");

    scanf("%d", &n);

    struct node\* temp;

    for(int i = 1; i <= n; i++)

    {

        struct node\* newnode = (struct node\*)malloc(sizeof(struct node));

        printf("Enter data for %d NODE: ", i);

        scanf("%d", &newnode->data);

        newnode->prev = NULL;

        newnode->next = NULL;

        if (start == NULL)

        {

            start = newnode;

            temp = newnode;

        }

        else

        {

            temp->next = newnode;

            newnode->prev = temp;

            temp = newnode;

        }

    }

    do

    {

        printf("1. INSERT\n2. DELETE\n3. DISPLAY\n4. TRAVERSE\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch(choice)

        {

            case 1:

                insert();

                break;

            case 2:

                delete\_node();

                break;

            case 3:

                display();

                break;

            case 4:

                traverse();

                break;

            default:

                printf("Invalid choice\n");

        }

        printf("Do you want to continue (Y/N)? ");

        fflush(stdin);

        scanf(" %c", &ch);

    } while(ch == 'y' || ch == 'Y');

}

void insert()

{

    int pos, val, i = 1;

    printf("Enter position to insert: ");

    scanf("%d", &pos);

    printf("Enter data of node for insertion: ");

    scanf("%d", &val);

    struct node\* newnode = (struct node\*)malloc(sizeof(struct node));

    newnode->data = val;

    struct node\* temp = start;

    if (pos == 1)

    {

        newnode->next = start;

        start->prev = newnode;

        start = newnode;

    }

    else

    {

        while (i++ < pos - 1 && temp != NULL)

        {

            temp = temp->next;

        }

        if (temp == NULL)

        {

            printf("Invalid position\n");

            return;

        }

        newnode->next = temp->next;

        if (temp->next != NULL)

        {

            temp->next->prev = newnode;

        }

        temp->next = newnode;

        newnode->prev = temp;

    }

    display();

}

void delete\_node()

{

    int pos, i = 1;

    printf("Enter position of the node to delete: ");

    scanf("%d", &pos);

    struct node\* temp = start;

    struct node\* prev = NULL;

    if (start == NULL)

    {

        printf("Linked list is empty\n");

        return;

    }

    if (pos == 1)

    {

        start = temp->next;

        if (start != NULL)

        {

            start->prev = NULL;

        }

        free(temp);

    }

    else

    {

        while (i++ < pos && temp != NULL)

        {

            prev = temp;

            temp = temp->next;

        }

        if (temp == NULL)

        {

            printf("Invalid position\n");

            return;

        }

        prev->next = temp->next;

        if (temp->next != NULL)

        {

            temp->next->prev = prev;

        }

        free(temp);

    }

    display();

}

void display()//

{

    int i = 1;

    struct node\* temp = start;

    printf("Elements of linked list are:\n");

    if (start == NULL)

    {

        printf("Linked list is empty\n");

    }

    while (temp != NULL)

    {

        printf("NODE %d data is %d\n", i++, temp->data);

        temp = temp->next;

    }

    printf("\n");

}

void traverse()

{

    int i = 1;

    struct node\* temp = start;

    printf("Traversing forward:\n");

    while (temp != NULL)

    {

        printf("NODE %d data is %d\n", i++, temp->data);

        temp = temp->next;

    }

    printf("\n");

    printf("Traversing backward:\n");

    temp = start;

    while (temp->next != NULL)

    {

        temp = temp->next;

    }

    i = 1;

    while (temp != NULL)

    {

        printf("NODE %d data is %d\n", i++, temp->data);

        temp = temp->prev;

    }

    printf("\n");

}

16. Write a program that uses functions to perform the following operations on circular linked list: i) Creation ii) Insertion iii) Deletion iv) Traversal

#include <stdio.h>

#include <stdlib.h>

struct node

{

    int data;

    struct node\* next;

};

struct node\* start = NULL;

void create();

void insert();

void delete\_node();

void display();

void traverse();

int main()

{

    create();

}

void create()

{

    int n, choice;

    char ch;

    printf("Enter the number of nodes: ");

    scanf("%d", &n);

    struct node\* temp;

    for(int i = 1; i <= n; i++)

    {

        struct node\* newnode = (struct node\*)malloc(sizeof(struct node));

        printf("Enter data for %d NODE: ", i);

        scanf("%d", &newnode->data);

        newnode->next = NULL;

        if (start == NULL)

        {

            start = newnode;

            temp = newnode;

        }

        else

        {

            temp->next = newnode;

            temp = newnode;

        }

    }

    temp->next = start;

    do

    {

        printf("1. INSERT\n2. DELETE\n3. DISPLAY\n4. TRAVERSE\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch(choice)

        {

            case 1:insert();break;

            case 2:delete\_node();break;

            case 3:display();break;

            case 4:traverse();break;

            default:printf("Invalid choice\n");

        }

        printf("Do you want to continue (Y/N)? ");

        fflush(stdin);

        scanf(" %c", &ch);

    } while(ch == 'y' || ch == 'Y');

}

void insert()

{

    int pos, val, i = 1;

    printf("Enter position to insert: ");

    scanf("%d", &pos);

    printf("Enter data of node for insertion: ");

    scanf("%d", &val);

    struct node\* newnode = (struct node\*)malloc(sizeof(struct node));

    newnode->data = val;

    struct node\* temp = start;

    if (pos == 1)

    {

        newnode->next = start;

        start = newnode;

    }

    else

    {

        while (i++ < pos - 1 && temp->next != start)

        {

            temp = temp->next;

        }

        if (temp->next == start && i != pos)

        {

            printf("Invalid position\n");

            return;

        }

        newnode->next = temp->next;

        temp->next = newnode;

    }

    display();

}

void delete\_node()

{

    int pos, i = 1;

    printf("Enter position of the node to delete: ");

    scanf("%d", &pos);

    struct node\* temp = start;

    struct node\* prev = NULL;

    if (start == NULL)

    {

        printf("Linked list is empty\n");

        return;

    }

    if (pos == 1)

    {

        start = temp->next;

        free(temp);

    }

    else

    {

        while (i++ < pos && temp->next != start)

        {

            prev = temp;

            temp = temp->next;

        }

        if (temp->next == start && i != pos)

        {

            printf("Invalid position\n");

            return;

        }

        prev->next = temp->next;

        free(temp);

    }

    display();

}

void display()

{

    int i = 1;

    struct node\* temp = start;

    printf("Elements of linked list are:\n");

    if (start == NULL)

    {

        printf("Linked list is empty\n");

        return;

    }

    do

    {

        printf("NODE %d data is %d\n", i++, temp->data);

        temp = temp->next;

    } while (temp != start);

    printf("\n");

}

void traverse()

{

    int i = 1;

    struct node\* temp = start;

    printf("Traversing:\n");

    do

    {

        printf("NODE %d data is %d\n", i++, temp->data);

        temp = temp->next;

    } while (temp != start);

    printf("\n");

}

17. Write a C program to Convert the given Infix Expression to Postfix Expression.

#include <stdio.h>

#include <ctype.h>

#define MAX 100

int stack[MAX];

int top = -1;

void push(char op) {

stack[++top] = op;

}

char pop() {

return stack[top--];

}

int precedence(char op) {

if (op == '+' || op == '-') {

return 1;

}

if (op == '\*' || op == '/') {

return 2;

}

return 0;

}

int main() {

char st[MAX] = "A + B \* C + D";

char \*e = st;

printf("Infix expression: %s\n", st);

printf("Postfix expression: ");

while (\*e != '\0') {

if (isalnum(\*e)) {

printf("%c", \*e);

} else if (\*e == ' ') {

// Ignore whitespace

} else if (\*e == '(') {

push(\*e);

} else if (\*e == ')') {

char ch1;

while (top != -1 && (ch1 = pop()) != '(') {

printf("%c", ch1);

}

} else {

while (top != -1 && precedence(stack[top]) >= precedence(\*e)) {

printf("%c", pop());

}

push(\*e);

}

e++;

}

while (top != -1) {

printf("%c", pop());

}

printf("\n");

return 0;

}

18. Write a C program to Evaluate the given Postfix Expression.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX 100

int st[MAX];

int top = -1;

void push(int val) {

st[++top] = val;

}

int pop() {

return st[top--];

}

int evaluatePostfix(char \*postfix) {

int i = 0;

while (postfix[i] != '\0') {

if (isdigit(postfix[i])) {

push(postfix[i] - '0'); // Convert char digit to integer

} else if (postfix[i] == '+' || postfix[i] == '-' || postfix[i] == '\*' || postfix[i] == '/') {

int operand2 = pop();

int operand1 = pop();

int result;

switch (postfix[i]) {

case '+':

result = operand1 + operand2;

break;

case '-':

result = operand1 - operand2;

break;

case '\*':

result = operand1 \* operand2;

break;

case '/':

if (operand2 != 0) {

result = operand1 / operand2;

} else {

printf("Error: Division by zero\n");

exit(EXIT\_FAILURE);

}

break;

}

push(result);

}

i++;

}

return pop(); // Final result after evaluating entire postfix expression

}

int main() {

char postfix[MAX];

printf("Enter the postfix expression: ");

fgets(postfix, MAX, stdin);

int result = evaluatePostfix(postfix);

printf("Result of postfix evaluation: %d\n", result);

return 0;

}

19. Write a C program to implement Binary search tree i) Insertion ii) deletion iii) Traversals

#include<stdio.h>

#include<stdlib.h>

struct Node {

    int key;

    struct Node \*left, \*right;

};

struct Node \*newNode(int key) {

    struct Node \*temp = (struct Node\*)malloc(sizeof(struct Node));

    temp->key = key;

    temp->left = temp->right = NULL;

    return temp;

}

struct Node \*insert(struct Node \*root, int key) {

    if (root == NULL) {

        return newNode(key);

    }

    if (key < root->key) {

        root->left = insert(root->left, key);

    } else if (key > root->key) {

        root->right = insert(root->right, key);

    }

    return root;

}

struct Node \*minValueNode(struct Node \*node) {

    struct Node \*current = node;

    while (current && current->left != NULL) {

        current = current->left;

    }

    return current;

}

struct Node \*deleteNode(struct Node \*root, int key) {

    if (root == NULL) {

        return root;

    }

    if (key < root->key) {

        root->left = deleteNode(root->left, key);

    }

    else if (key > root->key) {

        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->key = temp->key;

        root->right = deleteNode(root->right, temp->key);

    }

    return root;

}

void inorder(struct Node \*root) {

    if (root != NULL) {

        inorder(root->left);

        printf("%d ", root->key);

        inorder(root->right);

    }

}

void preorder(struct Node \*root)

{

    if(root==NULL)

    return;

    printf("%d",root->key);

    preorder(root->left);

    preorder(root->right);

}

int main() {

    struct Node \*root = NULL;

    int ch, key;

    char c;

    do {

        printf("\n1. Insert\n");

        printf("2. Delete\n");

        printf("3. Inorder Traversal\n");

        printf("4. preorder\n");

        printf("Enter your choice: ");

        scanf("%d", &ch);

        switch (ch) {

            case 1:

                printf("Enter key to insert: ");

                scanf("%d", &key);

                root = insert(root, key);

                break;

            case 2:

                printf("Enter key to delete: ");

                scanf("%d", &key);

                root = deleteNode(root, key);

                break;

            case 3:

                printf("Inorder traversal: ");

                inorder(root);

                printf("\n");

                break;

            case 4:

                printf("Inorder transversal: ");

                preorder(root);

                printf("\n");

            default:

                printf("Invalid choice\n");

        }

        printf("Do you want to continue (Y/N): ");

        scanf(" %c", &c);

    } while (c == 'Y' || c == 'y');

}

20. Write a C program to implement AVL tree i) Creation ii) Deletion iii) Traversal

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

    int key;

    struct Node\* left;

    struct Node\* right;

    int height;

} Node;

int height(Node\* N) {

    if (N == NULL)

        return 0;

    return N->height;

}

int max(int a, int b) {

    return (a > b) ? a : b;

}

Node\* newNode(int key) {

    Node\* node = (Node\*)malloc(sizeof(Node));

    node->key = key;

    node->left = NULL;

    node->right = NULL;

    node->height = 1;

    return(node);

}

Node\* rightRotate(Node\* y) {

    Node\* x = y->left;

    Node\* T2 = x->right;

    x->right = y;

    y->left = T2;

    y->height = max(height(y->left), height(y->right)) + 1;

    x->height = max(height(x->left), height(x->right)) + 1;

    return x;

}

Node\* leftRotate(Node\* x) {

    Node\* y = x->right;

    Node\* T2 = y->left;

    y->left = x;

    x->right = T2;

    x->height = max(height(x->left), height(x->right)) + 1;

    y->height = max(height(y->left), height(y->right)) + 1;

    return y;

}

int getBalance(Node\* N) {

    if (N == NULL)

        return 0;

    return height(N->left) - height(N->right);

}

Node\* insert(Node\* node, int key) {

    if (node == NULL)

        return(newNode(key));

    if (key < node->key)

        node->left = insert(node->left, key);

    else if (key > node->key)

        node->right = insert(node->right, key);

    else

        return node;

    node->height = 1 + max(height(node->left), height(node->right));

    int balance = getBalance(node);

    if (balance > 1 && key < node->left->key)

        return rightRotate(node);

    if (balance < -1 && key > node->right->key)

        return leftRotate(node);

    if (balance > 1 && key > node->left->key) {

        node->left = leftRotate(node->left);

        return rightRotate(node);

    }

    if (balance < -1 && key < node->right->key) {

        node->right = rightRotate(node->right);

        return leftRotate(node);

    }

    return node;

}

Node\* deleteNode(Node\* root, int key) {

    if (root == NULL)

        return root;

    if (key < root->key)

        root->left = deleteNode(root->left, key);

    else if (key > root->key)

        root->right = deleteNode(root->right, key);

    else {

        if ((root->left == NULL) || (root->right == NULL)) {

            Node\* temp = root->left ? root->left : root->right;

            if (temp == NULL) {

                temp = root;

                root = NULL;

            }

            else

                \*root = \*temp;

            free(temp);

        }

        else {

            Node\* temp = root->right;

            while (temp->left != NULL)

                temp = temp->left;

            root->key = temp->key;

            root->right = deleteNode(root->right, temp->key);

        }

    }

    if (root == NULL)

        return root;

    root->height = 1 + max(height(root->left), height(root->right));

    int balance = getBalance(root);

    if (balance > 1 && getBalance(root->left) >= 0)

        return rightRotate(root);

    if (balance > 1 && getBalance(root->left) < 0) {

        root->left = leftRotate(root->left);

        return rightRotate(root);

    }

    if (balance < -1 && getBalance(root->right) <= 0)

        return leftRotate(root);

    if (balance < -1 && getBalance(root->right) > 0) {

        root->right = rightRotate(root->right);

        return leftRotate(root);

    }

    return root;

}

void preOrder(Node\* root) {

    if (root != NULL) {

        printf("%d ", root->key);

        preOrder(root->left);

        preOrder(root->right);

    }

}

void inOrder(Node\* root) {

    if (root != NULL) {

        inOrder(root->left);

        printf("%d ", root->key);

        inOrder(root->right);

    }

}

void postOrder(Node\* root) {

    if (root != NULL) {

        postOrder(root->left);

        postOrder(root->right);

        printf("%d ", root->key);

    }

}

int main() {

    Node\* root = NULL;

    int choice, key;

    while (1) {

        printf("\nAVL Tree Operations:\n");

        printf("1. Insert\n");

        printf("2. Delete\n");

        printf("3. Pre-order Traversal\n");

        printf("4. In-order Traversal\n");

        printf("5. Post-order Traversal\n");

        printf("6. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter key to insert: ");

                scanf("%d", &key);

                root = insert(root, key);

                break;

            case 2:

                printf("Enter key to delete: ");

                scanf("%d", &key);

                root = deleteNode(root, key);

                break;

            case 3:

                printf("Pre-order Traversal: ");

                preOrder(root);

                printf("\n");

                break;

            case 4:

                printf("In-order Traversal: ");

                inOrder(root);

                printf("\n");

                break;

            case 5:

                printf("Post-order Traversal: ");

                postOrder(root);

                printf("\n");

                break;

            case 6:

                exit(0);

            default:

                printf("Invalid choice.\n");

        }

    }

    return 0;

}

21. Write a C program to count the number of leaf nodes in a tree.

#include <stdio.h>

#include <stdlib.h>

// Definition of the Node structure

struct Node {

int key;

struct Node \*left, \*right;

};

// Function to create a new node

struct Node\* newNode(int key) {

struct Node \*temp = (struct Node\*) malloc(sizeof(struct Node));

temp->key = key;

temp->left = temp->right = NULL;

return temp;

}

// Function to insert a key into the BST

struct Node\* insert(struct Node \*root, int key) {

if (root == NULL) {

return newNode(key);

}

if (key < root->key) {

root->left = insert(root->left, key);

} else if (key > root->key) {

root->right = insert(root->right, key);

}

return root;

}

// Function to count the leaf nodes in the BST

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);

}

int main() {

struct Node \*root = NULL;

int key;

printf("Enter keys to insert into the BST (enter -1 to stop):\n");

while (1) {

scanf("%d", &key);

if (key == -1) {

break;

}

root = insert(root, key);

}

int leafCount = countLeafNodes(root);

printf("Number of leaf nodes in the tree: %d\n", leafCount);

return 0;

}

22. Write a C program for implement DFS Graph traversal

#include <stdio.h>

void dfs(int v);

int n, a[10][10];

int visited[10];

int main() {

int i, j, v;

printf("Enter the number of nodes in the Graph:\t");

scanf("%d", &n);

printf("Enter the adjacency matrix:\n");

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

for (j = 0; j < n; j++) {

scanf("%d", &a[i][j]);

}

}

printf("Enter the starting node for Depth First Search:\t");

scanf("%d", &v);

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

visited[i] = 0;

}

printf("DFS Traversal: ");

dfs(v);

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

if (visited[i] == 0) {

printf("->%d ", i);

}

}

printf("\n");

return 0;

}

void dfs(int v) {

int i, stack[10], top = -1, popped;

stack[++top] = v;

while (top >= 0) {

popped = stack[top--];

if (visited[popped] == 0) {

printf("->%d", popped);

visited[popped] = 1;

}

for (i = n - 1; i >= 0; i--) {

if (a[popped][i] == 1 && visited[i] == 0) {

stack[++top] = i;

}

}

}

}

23. Write a C program for implement BFS Graph traversal

#include <stdio.h>

int n;

int v[10] = {0}; // Visited array

int adj[10][10]; // Adjacency matrix

void bfs(int start) {

int q[10], f = -1, r = -1, i;

q[++r] = start;

v[start] = 1;

printf("BFS Traversal: ");

while (r != f) {

start = q[++f];

printf("%d\t", start);

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

if (adj[start][i] == 1 && v[i] == 0) {

q[++r] = i;

v[i] = 1;

}

}

}

printf("\n");

}

int main() {

int i, j, vt;

printf("Enter the number of vertices: ");

scanf("%d", &n);

printf("Enter the adjacency matrix:\n");

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

for (j = 0; j < n; j++) {

scanf("%d", &adj[i][j]);

}

}

printf("Enter the initial vertex number: ");

scanf("%d", &vt);

// Check if the input vertex is within the valid range

if (vt < 0 || vt >= n) {

printf("Invalid starting vertex\n");

return 1;

}

bfs(vt);

return 0;

}

24. Write C program to implement the Quick Sort technique.

#include <stdio.h>

#define size 50

int arr[size];

void quicksort(int,int);

void swap(int,int);

int main()

{

    int i,n;

    printf("\n Enter no.of elements u want?");

    scanf("%d",&n);

    printf("\n Enter any %d elements:",n);

    for(i=0;i<n;i++)

      scanf("%d",&arr[i]);

    quicksort(0,n-1);

     printf("\n After sorting the elements are:");

    for(i=0;i<n;i++)

      printf("%d\t",arr[i]);

    return 0;

}

void quicksort(int l, int h)

{

    int pivot= l,i=l,j=h;

    if(l<h)

    {

        while(i<j)

        {

            while(arr[i]<=arr[pivot]&&i<h)

             i++;

            while(arr[j]>arr[pivot] && j>0)

              j--;

            if(i<j)

              swap(i,j);

        }

        swap(j,pivot);

        quicksort(l,j-1);

        quicksort(j+1,h);

    }

}

void swap(int l, int r)

{

    int temp;

    temp=arr[l];

    arr[l]=arr[r];

    arr[r]=temp;

}

25. Write C program to implement the Merge sort technique

25 - Merge Sort

#include <stdio.h>

void mergeSort(int \*arr, int left, int right);

void merge(int \*arr, int left, int right, int mid);

int main() {

    int size;

    printf("Enter size of array: ");

    scanf("%d", &size);

    int arr[size];

    printf("\nEnter array elements: \n");

    for (int i = 0; i < size; i++) {

        scanf("%d", &arr[i]);

    }

    printf("\nArray before sorting: \n");

    for (int i = 0; i < size; i++) {

        printf("%d ", arr[i]);

    }

    mergeSort(arr, 0, size-1);

    printf("\nArray after sorting: \n");

    for (int i = 0; i < size; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

}

void mergeSort(int \*arr, int left, int right){

    if(left < right){

        int mid = (left+right)/2;

        mergeSort(arr, left, mid);

        mergeSort(arr, mid+1, right);

        merge(arr, left, right, mid);

    }

}

void merge(int \*arr, int left, int right, int mid){

    int n1 = mid - left + 1;

    int n2 = right - mid;

    int i, j, k;

    int leftArr[n1];

    int rightArr[n2];

    for(i = 0; i<n1; i++){

        leftArr[i] = arr[left+i];

    }

    for(j = 0; j<n2; j++){

        rightArr[j] = arr[mid+j+1];

    }

    i = 0;

    j = 0;

    k = left;

    while(i<n1 && j<n2){

        if(leftArr[i]<=rightArr[j]){

            arr[k] = leftArr[i];

            i++;

        } else {

            arr[k] = rightArr[j];

            j++;

        }

        k++;

    }

    while(i<n1){

        arr[k] = leftArr[i];

        i++;

        k++;

    }

    while(j<n2){

        arr[k] = rightArr[j];

        j++;

        k++;

    }

}

g

26. Write a C program for implement Boyer –Moore pattern matching algorithm

# include <string.h>

# include <stdio.h>

# define MAX 256

int max(int a, int b) {

    return (a > b) ? a : b;

}

void badCharHeuristic(char \*str, int size, int badchar[MAX]) {

    int i;

    for (i = 0; i < MAX; i++)

        badchar[i] = -1;

    for (i = 0; i < size; i++)

        badchar[(int) str[i]] = i;

}

void search(char \*txt, char \*pat) {

    int m = strlen(pat);

    int n = strlen(txt);

    int badchar[MAX];

    badCharHeuristic(pat, m, badchar);

    int i = 0;

    while (i <= (n - m))

    {

        int j = m - 1;

        while (j >= 0 && pat[j] == txt[i + j])

            j--;

        if (j < 0)

        {

            printf("Pattern occurs at Index = %d\n", i);

            i += (i + m < n) ? m - badchar[txt[i + m]] : 1;

        }

        else

            i += max(1, j - badchar[txt[i + j]]);

    }

}

void main()

{

    char txt[100];

    char pat[100];

    printf("Enter Text String: \n");

    gets(txt);

    printf("Enter Pattern String: \n");

    gets(pat);

    search(txt, pat);

}

27. Write a C program to implement Brute Force pattern matching algorithm

# include <string.h>

# include <stdio.h>

void compare(char \*txt, char \*pat)

{

    int m = strlen(txt);

    int n = strlen(pat);

    int i =0, j=0;

    for(i=0; i<=m-n; i++)

    {

        while(j<n && txt[i+j]==pat[j])

            j++;

        if(j==n)

        {

            printf("Pattern found at index: %d\n", i);

        }

        j=0;

    }

}

int main() {

    char txt[100];

    char pat[100];

    printf("Enter Text String: \n");

    gets(txt);

    printf("Enter Pattern String: \n");

    gets(pat);

    compare(txt, pat);

    return 0;

}

28. Write C programs to implement Circular Queue ADT using Linked List

#include <stdio.h>

#include <stdlib.h>

typedef struct node {

int data;

struct node \*ptr;

} node;

node \*front = NULL;

node \*rear = NULL;

void insert();

void lqdelete();

void display();

int main() {

int choice;

char ch;

do {

printf("1. Insert an element\n2. Delete an element\n3. Display the queue\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch(choice) {

case 1:

insert();

break;

case 2:

lqdelete();

break;

case 3:

display();

break;

case 4:

exit(0);

default:

printf("Enter valid choice.\n");

}

printf("\nDo you want to continue (Y/N)? ");

scanf(" %c", &ch);

} while(ch == 'y' || ch == 'Y');

return 0;

}

void insert() {

node \*newnode = (node \*) malloc(sizeof(node));

if (newnode == NULL) {

printf("\nMemory allocation failed.\n");

return;

}

int value;

printf("\nEnter value: ");

scanf("%d", &value);

newnode->data = value;

newnode->ptr = NULL;

if (front == NULL) {

front = newnode;

rear = newnode;

front->ptr = front; // Circular link to itself

} else {

rear->ptr = newnode;

newnode->ptr = front;

rear = newnode;

}

}

void lqdelete() {

if (front == NULL) {

printf("\nUNDERFLOW: Queue is empty.\n");

return;

}

node \*temp = front;

if (front == rear) { // Only one element in queue

front = NULL;

rear = NULL;

} else {

rear->ptr = front->ptr;

front = front->ptr;

}

free(temp);

}

void display() {

if (front == NULL) {

printf("\nEmpty queue.\n");

return;

}

node \*temp = front;

printf("\nQueue elements: ");

do {

printf("%d ", temp->data);

temp = temp->ptr;

} while (temp != front);

printf("\n");

}