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>

typedef struct sll

{

    int data;

    struct sll \*ptr;

}node;

node \*start=NULL;

int n;

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;

    }

#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>

typedef struct sll

{

    int data;

    struct sll \*ptr;

}node;

node \*start=NULL;

int n=0;

void create();

void Insert\_pos();

void del\_pos();

void transverse();

void main()

{

    int i;

    char c;

    do{

    printf("Choose From the options\n1.Create\n2.Insert\n3.del\n4.transverse\n");

    scanf("%d",&i);

    switch(i)

    {

        case 1: create();break;

        case 2: Insert\_pos();break;

        case 3: del\_pos();break;

        case 4: transverse();break;

        default: printf("Wrong Option");

    }

    printf("\nDo you want to insert or create\n(Y/N): ");

    scanf("%s",&c);

    }while(c=='Y'||c=='y');

}

void create()

{

    char ch;

    int value,t,i;

    node \*newnode,\*temp;

    do

    {

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

        n++;

        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');

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

}

void Insert\_pos()

{

    node \*newnode,\*temp;

    int pos,i=2,value;

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

    printf("\nEnter the value :");

    scanf("%d",&value);

    newnode->data=value;

    newnode->ptr=NULL;

    if(start==NULL)

        start=newnode;

    else{

        printf("Enter the position: ");

        scanf("%d",&pos);

        if(pos<1)

            printf("Incorrect position");

        else if (pos==1)

        {

            newnode->ptr=start;

            start=newnode;

        }

        else

        {

            temp=start;

            while(i<pos&&temp->ptr!=NULL)

        {

            temp=temp->ptr;

            i++;

        }

        newnode->ptr=temp->ptr;

        temp->ptr=newnode;

        }

    }

    transverse();

}

void del\_pos()

{

    node \*prev=start,\*temp=start->ptr;

    int pos,i=2;

    if(start==NULL)

        printf("Sll is empty ");

    else{

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

        scanf("%d",&pos);

        if(pos==1){

            start=start->ptr;

            free(prev);

        }

        else

        {

            while(i<pos&&temp->ptr!=NULL)

            {

                temp=temp->ptr;

                prev=prev->ptr;

                i++;

            }

            prev->ptr=temp->ptr;

            free(temp);

        }

    }

    transverse();

}

void transverse()

{

    node \*temp;

    temp=start;

    printf("\nvalues  are :");

    while(temp!=NULL){

         printf("%d ",temp->data);

         temp=temp->ptr;

    }

}

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 empty();

int full ();

void main ()

{

    char ch;

    int val,choice;

    do

    {

        printf("MENU\n");

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

        scanf("%d",&choice);

        if(choice==4)

            printf("Enter correct option");

        else

        {

            switch(choice)

            {

                case 1 :

                    if(full())

                        printf("Stack is full ");

                    else{

                        printf("Enter the element ");

                        scanf("%d",&val);

                        push(val);

                    }break;

                case 2:

                    if(empty())

                        printf("Stack is empty ");

                    else{

                        val=pop();

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

                    }break;

                case 3 :

                    if(empty())

                        printf("Stack is empty ");

                    else

                    {

                        print();

                    }break;

                default : printf("Wrong option");

            }

        }

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

        scanf(" %c",&ch);

    }while(ch=='y'||ch=='Y');

}

void push(int a)

{

    top++;

    st[top]=a;

}

int pop()

{

    top--;

    return(st[top+1]);

}

void print()

{

    int i =0;

    while(i<=top)

    {

        printf("%d ",st[i]);

        i++;

    }

}

int empty()

{

    if(top<=-1)

        return 1;

    else

        return 0 ;

}

int full()

{

    if(top>=MAX)

        return 1;

    else

        return 0;

}

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

#include <stdio.h>

#define MAX 3

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

void insert(int);

int lqdelete();

void print();

int empty();

int full();

int 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');

    return 0;

}

void insert(int a)

{

    if (front==-1)

        front= 0;

    rear=(rear + 1)%MAX;

    st[rear]= a;

}

int lqdelete()

{

    int val=st[front];

    if (front==rear)

        front=rear=-1;

    else

        front=(front+1)%MAX;

    return val;

}

void print()

{

    int i = front;

    do

    {

        printf("%d ", st[i]);

        i=(i+1)%MAX;

    } while (i!=(rear+1)%MAX);*//?*

}

int empty()

{

    if (rear==-1 && front==-1)

        return 1;

    else

        return 0;

}

int full()

{

    if ((front==0&&rear==MAX-1) || (rear==(front-1)%MAX))

        return 1;

    else

        return 0;

}

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>

typedef struct TreeNode {

    int value;

    struct TreeNode\* left;

    struct TreeNode\* right;

} TreeNode;

int height(TreeNode\* node) {

    if (node == NULL) {

        return -1;

    }

    int leftHeight = height(node->left);

    int rightHeight = height(node->right);

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

}

TreeNode\* newNode(int value) {

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

    node->value = value;

    node->left = NULL;

    node->right = NULL;

    return node;

}

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;

}

void printPreOrder(TreeNode\* node) {

    if (node == NULL) {

        return;

    }

    printf("%d ", node->value);

    printPreOrder(node->left);

    printPreOrder(node->right);

}

int main() {

    TreeNode\* root = NULL;

    root = insert(root, 50);

    insert(root, 30);

    insert(root, 20);

    insert(root, 40);

    insert(root, 70);

    insert(root, 60);

    insert(root, 80);

    int heightOfTree = height(root);

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

    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>

typedef struct dll {

    int data;

    struct dll \*lptr;

    struct dll \*rptr;

} node;

node \*start = NULL;

int n = 0;

void create();

void Insert\_pos();

void del\_pos();

void transverse();

void main() {

    int i;

    char c;

    do {

        printf("Choose From the options\n1.Create\n2.Insert(pos)\n3.del(pos)\n4.transverse\n");

        scanf("%d", &i);

        switch(i) {

            case 1:

                create();

                break;

            case 2:

                Insert\_pos();

                break;

            case 3:

                del\_pos();

                break;

            case 4:

                transverse();

                break;

        }

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

        scanf(" %c", &c);

    } while(c == 'Y' || c == 'y');

}

void create() {

    char ch;

    int value;

    do {

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

        if(newnode == NULL) {

            printf("Memory is not created");

            return;

        }

        printf("Enter the value: ");

        scanf("%d", &value);

        newnode->data = value;

        newnode->rptr = NULL;

        newnode->lptr = NULL;

        if (start == NULL) {

            start = newnode;

        } else {

            node \*temp = start;

            while (temp->rptr!= NULL) {

                temp = temp->rptr;

            }

            temp->rptr = newnode;

            newnode->lptr = temp;

        }

        n++;

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

        scanf(" %c", &ch);

    } while (ch == 'Y' || ch == 'y');

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

    printf("\nNodes are: ");

    transverse();

}

void Insert\_pos() {

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

    if (newnode == NULL) {

        printf("Memory is not created");

        return;

    }

    int pos, i = 1, value;

    printf("\nEnter the value: ");

    scanf("%d", &value);

    newnode->data = value;

    newnode->rptr = NULL;

    newnode->lptr = NULL;

    if(start == NULL) {

        start = newnode;

        n++;

        return;

    }

    printf("Enter the position: ");

    scanf("%d", &pos);

    node \*temp = start;

    while (i < pos && temp->rptr!= NULL) {

        temp = temp->rptr;

        i++;

    }

    newnode->rptr = temp->rptr;

    if (temp->rptr!= NULL)

        temp->rptr->lptr = newnode;

    temp->rptr = newnode;

    newnode->lptr = temp;

    n++;

    printf("\nNodes are: ");

    transverse();

}

void del\_pos() {

    int pos, i = 1;

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

    scanf("%d", &pos);

    node \*temp = start;

    while (i < pos && temp!= NULL) {

        temp = temp->rptr;

        i++;

    }

    if (temp == NULL) {

        printf("Position not found.");

        return;

    }

    if (temp->lptr!= NULL)

        temp->lptr->rptr = temp->rptr;

    if (temp->rptr!= NULL)

        temp->rptr->lptr = temp->lptr;

    if (temp == start)

        start = start->rptr;

    free(temp);

    n--;

    printf("\nNode at position %d deleted.", pos);

    printf("\nNodes are: ");

    transverse();

}

void transverse() {

    node \*temp = start;

    printf("\nValues are: ");

    while (temp!= NULL) {

        printf("%d ", temp->data);

        temp = temp->rptr;

    }

}

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>

typedef struct csll {

    int data;

    struct csll \*ptr;

} node;

void create\_csll();

void insert\_csll\_by\_pos();

void del\_csll\_by\_pos();

void traverse\_csll();

node \*start = NULL;

int n = 0;

int main() {

    int choice;

    char ch;

    do {

        printf("\nMENU DRIVEN\n");

        printf("1: Create\n2: Insert by position\n3: Delete by position\n4: Traverse\n5: Exit\n");

        printf("Enter the choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                create\_csll();

                break;

            case 2:

                insert\_csll\_by\_pos();

                break;

            case 3:

                del\_csll\_by\_pos();

                break;

            case 4:

                traverse\_csll();

                break;

            case 5:

                return 0;

            default:

                printf("Pick a valid choice\n");

        }

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

        scanf(" %c", &ch);

    } while (ch == 'y' || ch == 'Y');

    return 0;

}

void create\_csll() {

    node \*temp = start, \*newnode;

    char ch;

    do {

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

        if (newnode == NULL) {

            printf("Memory is not allocated");

        } else {

            printf("Enter the value: ");

            scanf("%d", &newnode->data);

            newnode->ptr = NULL;

            if (start == NULL) {

                start = newnode;

                newnode->ptr = start;

            } else {

                temp = start;

                while (temp->ptr!= start) {

                    temp = temp->ptr;

                }

                temp->ptr = newnode;

                newnode->ptr = start;

            }

            n++;

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

            scanf(" %c", &ch);

        }

    } while (ch == 'y' || ch == 'Y');

    traverse\_csll();

}

void traverse\_csll() {

    node \*temp;

    printf("Linked List: ");

    if (start == NULL) {

        printf("Empty\n");

        return;

    }

    temp = start;

    int i = 1;

    if (n == 1) {

        printf("%d ", temp->data);

    } else {

        do {

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

            temp = temp->ptr;

            i++;

        } while (temp!= start);

    }

}

void insert\_csll\_by\_pos() {

    node \*temp = start, \*newnode;

    int pos, i;

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

    printf("Enter the value: ");

    scanf("%d", &(newnode->data));

    newnode->ptr = NULL;

    if (newnode == NULL) {

        printf("the memory is not allocated\n");

    } else {

        printf("Enter the position\n");

        scanf("%d", &pos);

        if (start == NULL) {

            start = newnode;

            newnode->ptr = start;

        } else {

            if (pos == 1) {

                newnode->ptr = start;

                while (temp->ptr!= start) {

                    temp = temp->ptr;

                }

                start = newnode;

                temp->ptr = start;

            } else {

                i = 2;

                while (i < pos && temp->ptr!= start) {

                    temp = temp->ptr;

                    i++;

                }

                newnode->ptr = temp->ptr;

                temp->ptr = newnode;

            }

            n++;

        }

    traverse\_csll();

}

void del\_csll\_by\_pos() {

    node \*temp = start->ptr, \*prev = start;

    int pos, i;

    printf("Enter the value of pos: ");

    scanf("%d", &pos);

    if (start == NULL) {

        printf("the cll is empty");

    } else {

        if (pos == 1) {

            if (start->ptr == start) {

                start = NULL;

                free(prev);

                traverse\_csll();

                return;

            }

            while (temp->ptr!= start) {

                temp = temp->ptr;

            }

            temp->ptr = start->ptr;

            start = start->ptr;

            free(prev);

        } else {

            i = 2;

            while (i < pos && temp->ptr!= start) {

                temp = temp->ptr;

                prev = prev->ptr;

                i++;

            }

            prev->ptr = temp->ptr;

            free(temp);

        }

    traverse\_csll();

}

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 check(int a)

{

    if(a=='+'||a=='-')

        return 1;

    if(a=='\*'||a=='/')

        return 2;

    return 0;

}

void main()

{

    char st[100]="A + B \* C + D";;

    char \*e=st;

    while(\*e!='\0')

    {

        if(isalnum(\*e))

            printf("%c",\*e);

            else if(\*e=='(')

            {

                push(\*e);

            }

            else if(\*e==')')

            {

                char ch1;

                while((ch1=pop())!='(')

                        printf("%c",ch1);

            }

      else

      {

          if(check(stack[top])>=check(\*e))

            printf("%c",pop());

      push(\*e);

      }

    e++;

    }

    while(top!=-1)

    {

        printf("%c",stack[top]);

        top=top-1;

    }

}

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

#include <stdio.h>

#include <ctype.h>

#include <math.h>

#define MAX 100

char st[MAX], top = -1;

void push(char);

char pop();

void print();

int isempty();

int isfull();

void empty();

void convertPf();

int evaluatePf(char \*);

int precedence(char ch);

void main()

{

    int val, choice;

    char ch;

    do

    {

        printf("Stack:\n1 - Push\n2 - Pop\n3 - Print\n4 - Evaluate Expression\n5 - Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch(choice)

        {

            case 1:

                if(isfull())

                    printf("Stack Is Full\n");

                else

                {

                    printf("Enter the element: \n");

                    scanf("%d", &val);

                    push(val);

                }

                break;

            case 2:

                if(isempty())

                    printf("Stack is Empty!\n");

                else

                {

                    val = pop();

                    printf("The value %d has been Popped off the stack\n", val);

                }

                break;

            case 3:

                print();

                break;

            case 4:

                convertPf();

                break;

            case 5:

                return;

            default:

                printf("Invalid Choice\n");

        }

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

        scanf("%s", &ch);

    } while(ch == 'y' || ch == 'Y');

}

void push(char a)

{

    top++;

    st[top] = a;

}

char pop()

{

    return st[top--];

}

void print()

{

    printf("Values in the stack are:\n");

    for(int i = 0; i<=top; i++)

    {

        printf("%d\t", st[i]);

    }

}

int isempty()

{

    if(top<=-1)

    {

        return 1;

    }

    return 0;

}

int isfull()

{

    if(top>=MAX)

    {

        return 1;

    }

    return 0;

}

int precedence(char ch)

{

    if (ch == '/' || ch == '\*')

        return 2;

    else if (ch == '+' || ch == '-')

        return 1;

    else

        return -1;

}

void convertPf()

{

    top = -1;

    char str[100];

    char output[100];

    printf("Enter the Expression: \n");

    gets(str);

    gets(str);

    int i = 0, flag = 0, j=0;

    while(str[i]!='\0')

    {

        if(isalnum(str[i]))

        {

            output[j] = str[i];

            j++;

            while(isdigit(str[i+1]))

            {

                i++;

                output[j] = str[i];

                j++;

            }

            output[j] = ' ';

            j++;

        }

        else if(str[i]=='(')

        {

            push(str[i]);

        }

        else if(str[i]==')')

        {

            char ret;

            while(!isempty() && (ret=pop())!='(')

            {

                output[j] = ret;

                j++;

            }

        }

        else if(str[i]=='+' || str[i]=='-' || str[i]=='\*' || str[i] == '/')

        {

            if(precedence(st[top])>=precedence(str[i]))

            {

                char ret;

                while(!isempty() && (ret=pop())!='(')

                {

                output[j] = ret;

                j++;

                }

            }

            push(str[i]);

        }

        i++;

    }

    while(!(isempty()))

    {

        output[j] = pop();

        j++;

    }

    output[j] = '\0';

    printf("%s\n", output);

    int result = evaluatePf(output);

    printf("%d", result);

}

int evaluatePf(char \*str)

{

    int i = 0;

    int num = 0;

    while (str[i] != '\0') {

        if (isdigit(str[i])) {

            num = 0;

            while (isdigit(str[i])) {

                num = num \* 10 + (str[i] - '0');

                i++;

            }

            push(num);

        }

        else if (str[i] == '+' || str[i] == '-' || str[i] == '\*' || str[i] == '/')

        {

            int num2 = pop();

            int num1 = pop();

            int result;

            switch (str[i]) {

                case '+':

                    result = num1 + num2;

                    break;

                case '-':

                    result = num1 - num2;

                    break;

                case '\*':

                    result = num1 \* num2;

                    break;

                case '/':

                    result = num1 / num2;

                    break;

            }

            push(result);

            i++;

        } else if (str[i] == ' ') {

            i++;

        }

    }

    return pop();

}

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>

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;

}

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 ch,key;

    printf("Enter keys (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);

}

22. Write a C program for implement DFS Graph traversal

#include<stdio.h>

#include<conio.h>

void dfs(int);

int n,a[10][10];

int visited[10];

void main()

{

 int i,j,v;

 printf("Enter the no. 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;

 dfs(v);

 for(i=0;i<n;i++)

 {

    if(visited[i]==0)

    printf("->%d ",i);

 }

}

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};

int adj[10][10];

void bfs(int start)

{

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

q[++r]=start;

v[start]=1;

while(r!=f)

{

start=q[++f];

if(start==n)

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

else

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

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

  {

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

         {

            q[++r]=i;

            v[i]=1;

         }

        }

    }

}

int main()

{

int i,j,vt;

printf("Enter No. of vertices");

scanf("%d",&n);

printf("enter the adjacency matrix");

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

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

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

printf("Enter the initial vertex no");

scanf("%d",&vt);

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>

typedef struct node

{

    int data;

    struct node \*ptr;

}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 ->ptr = front;

        }

        else

        {

            rear -> ptr = newnode;

            newnode->ptr=front;

            rear = newnode;

        }

    }

}

void lqdelete ()

{

    node \*temp,\*last=front;

    if(front == NULL)

    {

        printf("\nUNDERFLOW\n");

        return;

    }

    else

    {

        while(last->ptr!=front)

        {

            last=last->ptr;

        }

        temp= front;

        front = front ->ptr;

        last->ptr=front;

        free(temp);

    }

}

void display()

{

    node \*temp;

    temp = front;

    if(front == NULL)

    {

        printf("\nEmpty queue\n");

    }

    else

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

        do

        {

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

            temp = temp -> ptr;

        }while(temp!=front);

    }

}