DSA LAB PROGRAMMING

LAB 9-14

• Name :HITU RAJ

• Roll no. :2005025

• Branch :CSE

LAB 9

/\*•Q1.  Write a menu driven program to implement queue operations such as Enqueue,

 Dequeue, Peek, Display of elements, IsEmpty, IsFull using static array.\*/

#include <stdio.h>

#include <stdlib.h>

struct node

{

    int data;

    struct node \*next;

} \*front = NULL, \*rear = NULL;

//struct node\* front=NULL

//struct node\* rear=NULL

void enqueue(int x)

{

    struct node \*temp;

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

    if (temp == NULL)

    {

        printf("No memory\n");

    }

    else

    {

        temp->data = x;

        temp->next = NULL;

        if (front == NULL)

        {

            front = rear = temp;

        }

        else

        {

            rear->next = temp;

            rear = temp;

        }

    }

}

void dequeue()

{

   // int x = -1;

    struct node \*temp;

    if (front == NULL && rear == NULL)

    {

        printf("\n!!!!!!!!Cant delete  ueue is empty");

    }

    else

    {

        temp = front;

       printf("Elentent deleted is %d\n",front->data);

        front = front->next;

        free(temp);

    }

    //return x;

}

void display()

{

    struct node \*temp = front;

    if (front == NULL && rear == NULL)

    {

        printf("\n!!!!!!!!Cant delete  Queue is empty");

    }

    else

    {

        while (temp != NULL)

        {

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

            temp = temp->next;

        }

    }

}

void peek()

{

    printf("\nThe 1st element is %d", front->data);

}

int main()

{

    int n, c = 1;

    while (c !=2)

    {

        printf("\nPress 1 to enqueue");

        printf("\nPress 2 to dequeue");

        printf("\nPress 3 to peek");

        printf("\nPress 4 to display the elements");

        scanf("%d", &n);

        switch (n)

        {

        case 1:

        {

            int l;

            printf("\nenter the element to enqueue");

            scanf("%d", &l);

            enqueue(l);

            break;

        }

        case 2:

        {

            dequeue();

            break;

        }

        case 3:

        {

            peek();

            break;

        }

        case 4:

        {

            display();

            break;

        }

        default:

            break;

        }

        printf("Press 1 to continue\npress 2 to stop\n ");

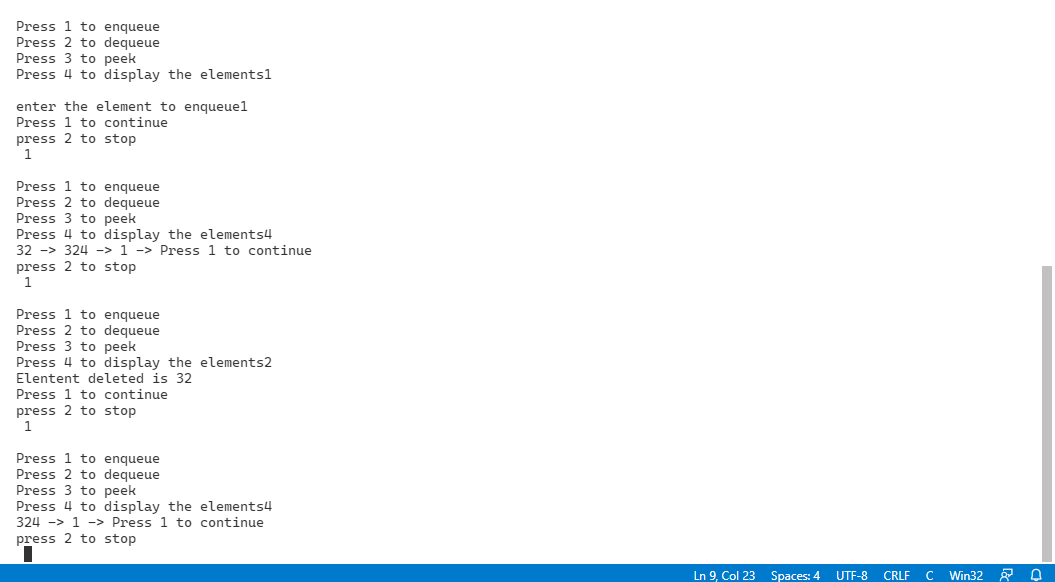
        scanf("%d", &c);

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

    }

    return 0;

}



/\*•Q2   Write a menu driven program to implement queue operations such

 as Enqueue, Dequeue, Peek, Display of elements, IsEmpty using linked list.\*/

#include<stdio.h>

#include<math.h>

#include<stdlib.h>

#define MAX 5

int queue[MAX];

int front = -1;

int rear = -1;

void enqueue(int x)

{

    if(rear==MAX-1) //queue is full

    {

        printf("Overflow because queue is full");

    }

    else if(front==-1 && rear==-1) //queue is empty

    {

        front=rear=0;

        queue[rear]=x;

    }

    else

    {

        rear++;

        queue[rear]=x;

    }

}

void dequeue()

{

    if(front==-1 && rear==-1) //queue is empty

    {

        printf("Underflow");

    }

    else if(front==rear) //single element present

    {

        printf("\n %d is deleted \n ",queue[front]);

        front=rear=-1;

    }

    else

    {

        printf("\n%d is deleted \n",queue[front]);

        front++;

    }

}

void display()

{

    int i;

    for(i=front;i<=rear+1;++i)

    {

        printf("%d -> ",queue[i]);

    }

    printf("\n");

}

void peek()

{

    printf("\n%d ",queue[front]);

}

int main()

{

     int n, c = 1;

    while (c !=2)

    {

        printf("\nPress 1 to enqueue");

        printf("\nPress 2 to dequeue");

        printf("\nPress 3 to peek");

        printf("\nPress 4 to display the elements");

        scanf("%d", &n);

        switch (n)

        {

        case 1:

        {

            int l;

            printf("\nenter the element to enqueue");

            scanf("%d", &l);

            enqueue(l);

            break;

        }

        case 2:

        {

            dequeue();

            break;

        }

        case 3:

        {

            peek();

            break;

        }

        case 4:

        {

            display();

            break;

        }

        default:

            break;

        }

        printf("Press 1 to continue\npress 2 to stop\n ");

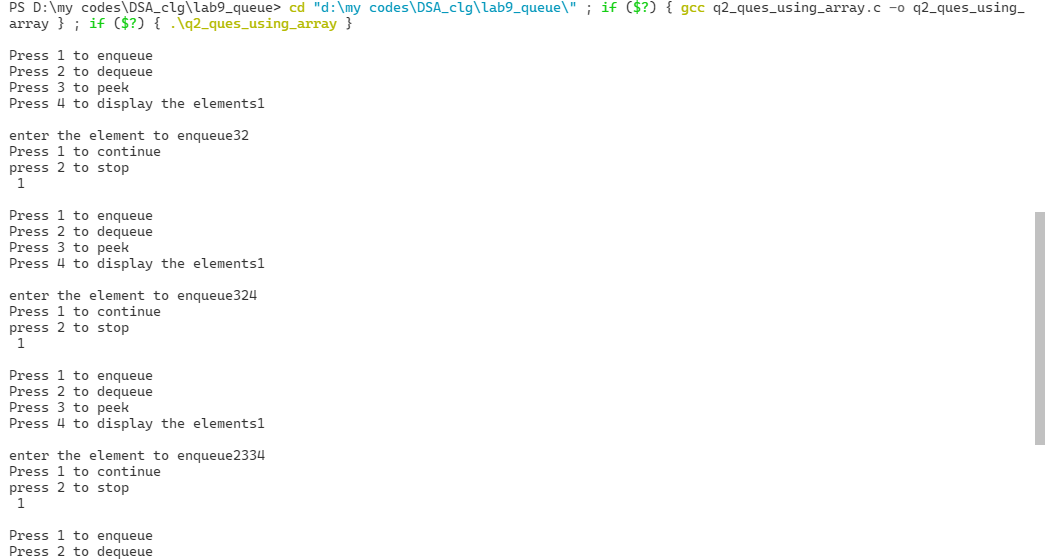
        scanf("%d", &c);

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

    }

return 0;

}





/\*•Q4   WAP using a function to reverse a queue by using stack.\*/

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#define MAX 5

int queue[MAX];

int stack[MAX];

int front = -1;

int rear = -1;

void enqueue(int x)

{

    if (rear == MAX - 1) //queue is full

    {

        printf("Overflow because queue is full");

    }

    else if (front == -1 && rear == -1) //queue is empty

    {

        front = rear = 0;

        queue[rear] = x;

    }

    else

    {

        rear++;

        queue[rear] = x;

    }

}

void dequeue()

{

    if (front == -1 && rear == -1) //queue is empty

    {

        printf("Underflow");

    }

    else if (front == rear) //single element present

    {

        printf("\n %d is deleted \n ", queue[front]);

        stack[front] = queue[front];

        front = rear = -1;

    }

    else

    {

        printf("\n%d is deleted \n", queue[front]);

        front++;

    }

}

void stackpop()

{

    int temp = front;

    while (temp != MAX - 1)

    {

        stack[temp] = queue[temp];

        temp++;

    }

    printf("reverse queue is is \n");

    int i;

    for (i = rear; i >=0; i--)

    {

        printf("%d -> ", stack[i]);

    }

    printf("\n");

}

void display()

{

    int i;

    for (i = front; i <= rear; ++i)

    {

        printf("%d -> ", queue[i]);

    }

    printf("\n");

}

void peek()

{

    printf("\n%d ", queue[front]);

}

int main()

{

    int n, c = 1;

    while (c != 2)

    {

        printf("\nPress 1 to enqueue");

        printf("\nPress 2 to dequeue");

        printf("\nPress 3 to peek");

        printf("\nPress 4 to display the elements");

        printf("\npress 5 to reverse thee queue");

        scanf("%d", &n);

        switch (n)

        {

        case 1:

        {

            int l;

            printf("\nenter the element to enqueue");

            scanf("%d", &l);

            enqueue(l);

            break;

        }

        case 2:

        {

            dequeue();

            break;

        }

        case 3:

        {

            peek();

            break;

        }

        case 4:

        {

            display();

            break;

        }

        case 5:

        {

            stackpop();

            break;

        }

        default:

            break;

        }

        printf("Press 1 to continue\npress 2 to stop\n ");

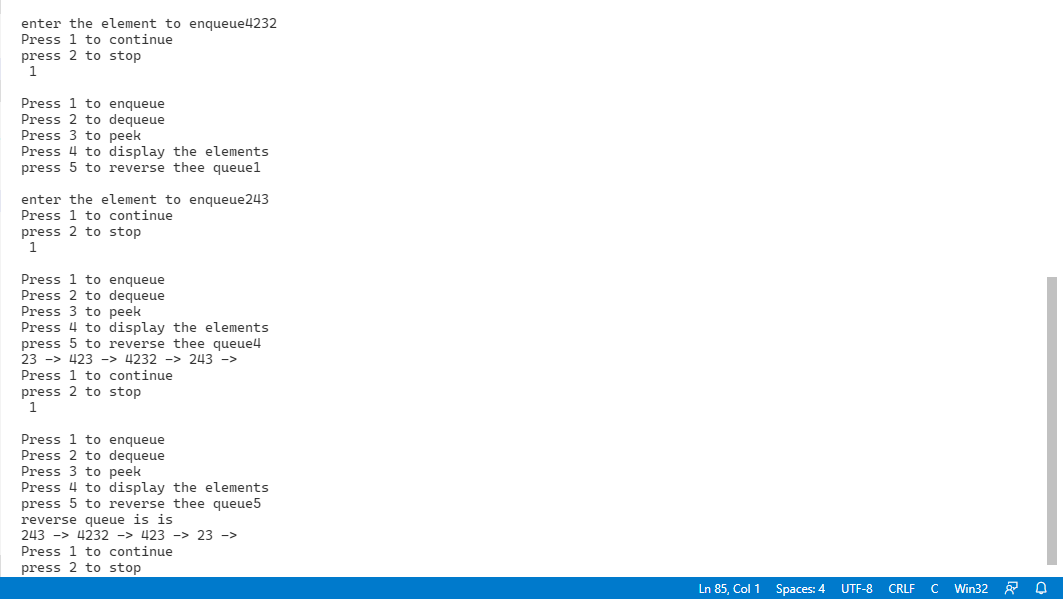
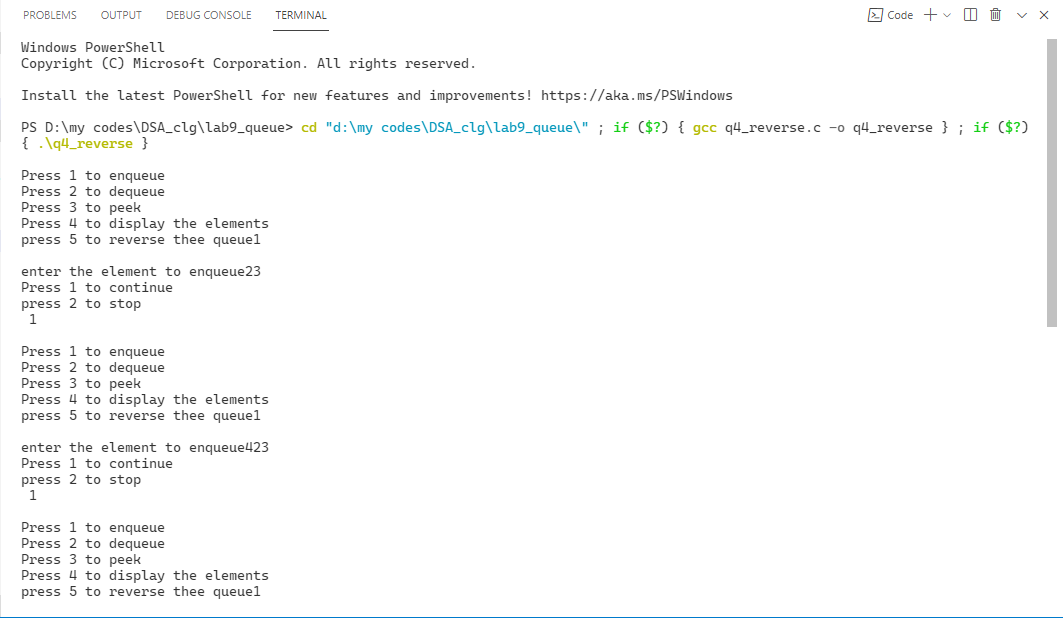
        scanf("%d", &c);

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

    }

    return 0;

}



LAB 10

/\*q1. Wap to create circular queue using link list with following operation.

•   1. insert at Begining.

•   2. insert at End.

•   3. insert at Position.

•   4. delete at Begining.

•   5. delete at End.

•   6. delete at Position.

•   7. traverse the List

\*/

#include <stdio.h>

#include <stdlib.h>

struct Cirqu

{

    int size;

    int front;

    int rear;

    int \*Arr;

};

void create(struct Cirqu \*q, int size)

{

    q->size = size;

    q->front = q->rear = 0;

    q->Arr = (int \*)malloc(q->size \* sizeof(int));

}

void enqueue(struct Cirqu \*q, int x)

{

    if ((q->rear + 1) % q->size == q->front)

        printf("Cirqu is Full");

    else

    {

        q->rear = (q->rear + 1) % q->size;

        q->Arr[q->rear] = x;

    }

}

int dequeue(struct Cirqu \*q)

{

    int x = -1;

    if (q->front == q->rear)

        printf("Cirqu is Empty\n");

    else

    {

        q->front = (q->front + 1) % q->size;

        x = q->Arr[q->front];

    }

    return x;

}

void Display(struct Cirqu q)

{

    int i = q.front + 1;

    while (i != (q.rear + 1) % q.size)

    {

        printf("%d ", q.Arr[i]);

        i = (i + 1) % q.size;

    }

    printf("\n");

}

int main()

{

    int n, t;

    struct Cirqu q;

    printf("\n Enter thesize of queue");

    int no;

    scanf("%d", &no);

    create(&q, no);

    while (n)

    {

        printf("Press 1 to enqueue\n");

        printf("Press 2 to dequeue\n");

        printf("Press 3 to Traverse the Circularqueue\n");

        scanf("%d", &t);

        switch (t)

        {

        case 1:

            printf("\n Enter the no. to enque");

            int no;

            scanf("%d", &no);

            enqueue(&q, no);

            break;

        case 2:

            printf("\n  the no.which is removed is\n");

            printf("%d ", dequeue(&q));

            break;

        case 3:

            printf("\nQueue displayed is  \n");

            Display(q);

            break;

        default:

            break;

        }

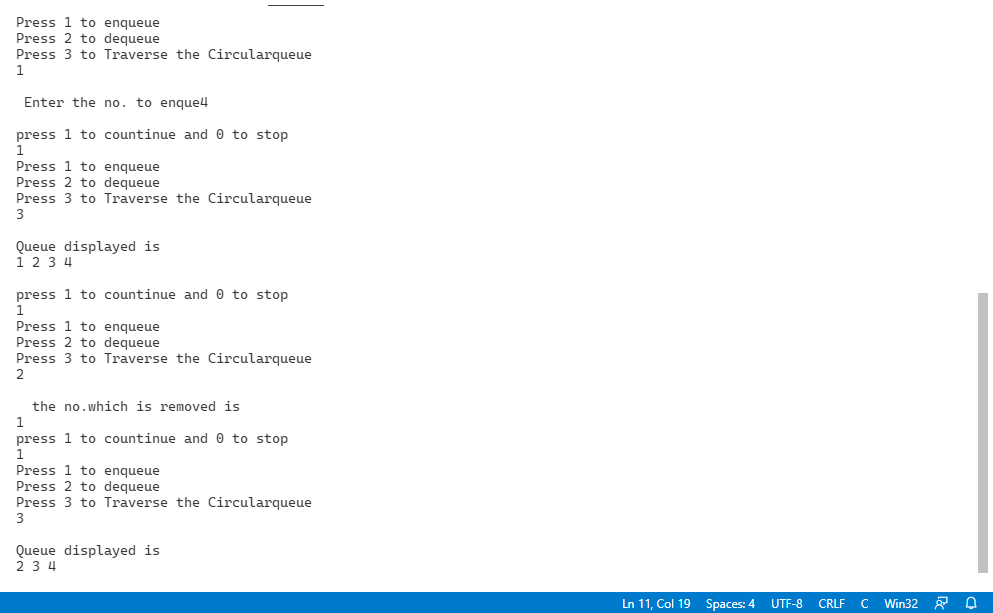
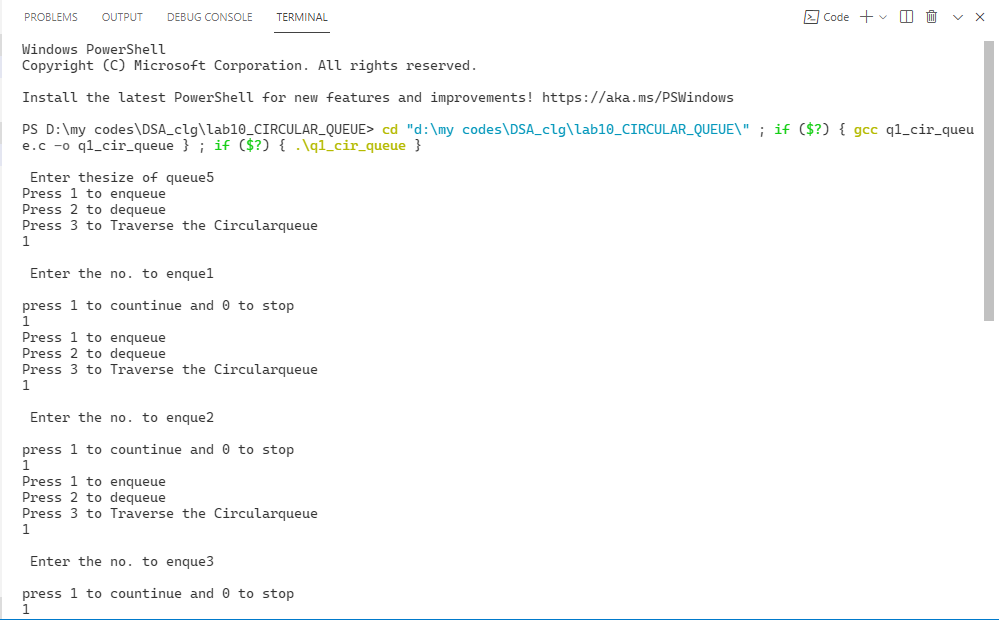
        printf("\npress 1 to countinue and 0 to stop\n");

        scanf("%d", &n);

    }

    return 0;

}



//Q2A

//Write a menu driven program to implement DEquee ( Input-restricted) operations such as

//Enqueue, Dequeue, Peek, Display of elements, IsEmpty using Array.

// o/p restricted queue

#include <stdio.h>

#include <stdlib.h>

struct Cirqu

{

    int size;

    int front;

    int rear;

    int \*Arr;

};

void create(struct Cirqu \*q, int size)

{

    q->size = size;

    q->front = q->rear = 0;

    q->Arr = (int \*)malloc(q->size \* sizeof(int));

}

void enqueue(struct Cirqu \*q, int x)

{

    if ((q->rear + 1) % q->size == q->front)

        printf("Cirqu is Full");

    else

    {

        q->rear = (q->rear + 1) % q->size;

        q->Arr[q->rear] = x;

    }

}

int dequeue\_fro(struct Cirqu \*q)

{

    int x = -1;

    if (q->front == q->rear)

        printf("Cirqu is Empty\n");

    else

    {

        q->front = (q->front + 1) % q->size;

        x = q->Arr[q->front];

    }

    return x;

}

int dequeue\_rear(struct Cirqu \*q)

{

    int x = -1;

    if (q->front == q->rear)

        printf("Cirqu is Empty\n");

    else

    {

        printf("%d is deleted \n", q->Arr[q->rear]);

        q->rear = (q->rear - 1) % q->size;

    }

    return x;

}

void Display(struct Cirqu q)

{

    int i = q.front + 1;

    while (i != (q.rear + 1) % q.size)

    {

        printf("%d ", q.Arr[i]);

        i = (i + 1) % q.size;

    }

    printf("\n");

}

int main()

{

    int n, t;

    struct Cirqu q;

    printf("\n Enter thesize of queue");

    int no;

    scanf("%d", &no);

    create(&q, no);

    while (n)

    {

        printf("Press 1 to enqueue using rear\n");

        printf("Press 2 to dequeue from front\n");

        printf("Press 3 to dequeue from rear\n");

        printf("Press 4 to display the Circularqueue\n");

        scanf("%d", &t);

        switch (t)

        {

        case 1:

            printf("\n Enter the no. to enque");

            int no;

            scanf("%d", &no);

            enqueue(&q, no);

            break;

        case 2:

            printf("\n  the no.which is removed from front is\n");

            printf("%d ", dequeue\_fro(&q));

            break;

        case 3:

            printf("\n  the no.which is removed from rear is\n");

            printf("%d ", dequeue\_rear(&q));

            break;

        case 4:

            printf("\nQueue displayed is  \n");

            Display(q);

            break;

        default:

            break;

        }

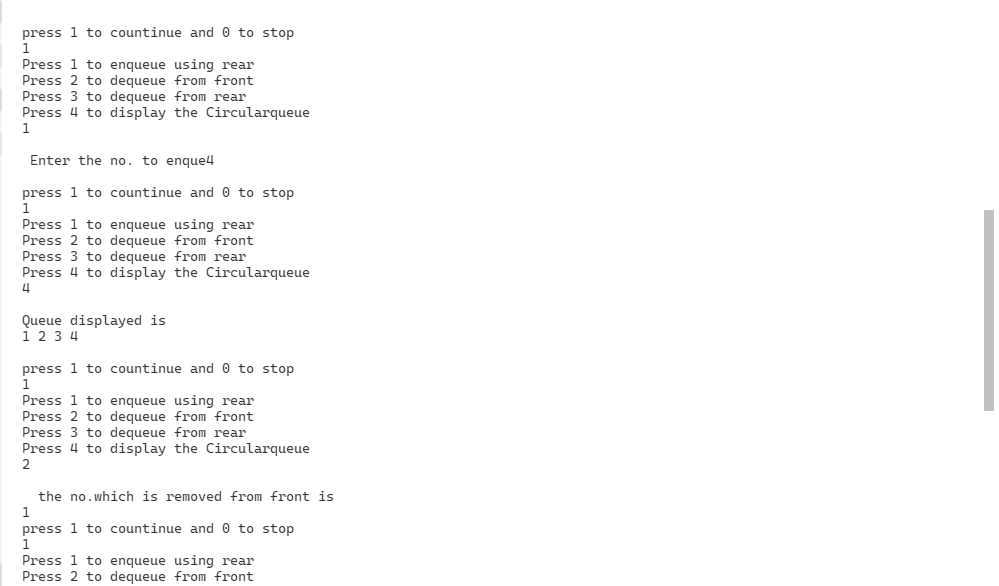
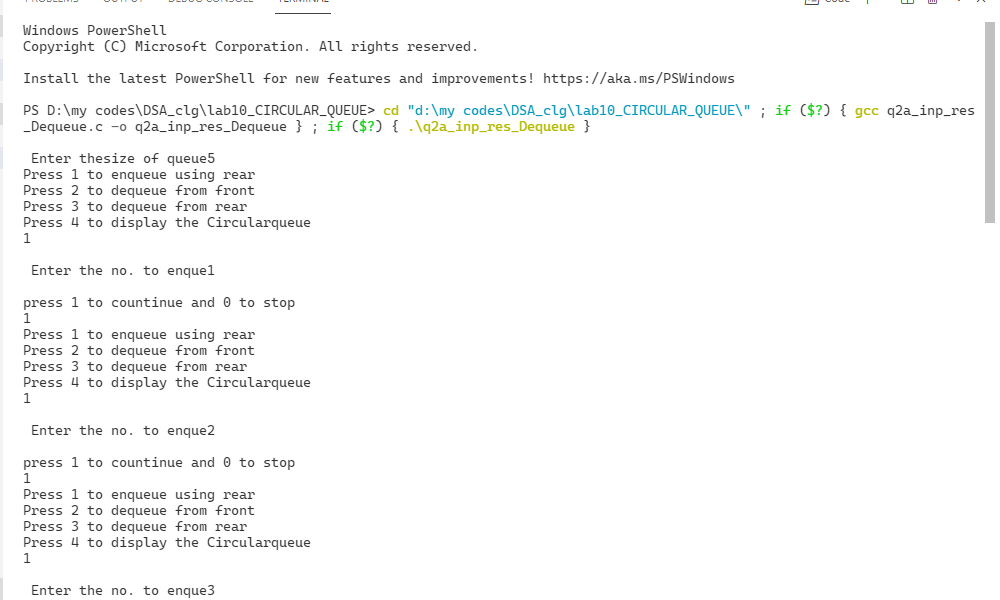
        printf("\npress 1 to countinue and 0 to stop\n");

        scanf("%d", &n);

    }

    return 0;

}



//Q2B

//Write a menu driven program to implement Deques ( Output-restricted) operations such as

//Enqueue, Dequeue, Peek, Display of elements, IsEmpty using Array.

// o/p restricted queue

#include <stdio.h>

#include <stdlib.h>

struct Cirqu

{

    int size;

    int front;

    int rear;

    int \*Arr;

};

void create(struct Cirqu \*q, int size)

{

    q->size = size;

    q->front = q->rear = 0;

    q->Arr = (int \*)malloc(q->size \* sizeof(int));

}

void enqueue\_rear(struct Cirqu \*q, int x)

{

    if ((q->rear + 1) % q->size == q->front)

        printf("Cirqu is Full");

    else

    {

        q->rear = (q->rear + 1) % q->size;

        q->Arr[q->rear] = x;

    }

}

void enqueue\_front(struct Cirqu \*q, int x)

{

    if ((q->rear + 1) % q->size == q->front)

        printf("Cirqu is Full");

    else

    {

        q->Arr[q->front] = x;

        q->front = (q->front - 1) % q->size;

    }

}

int dequeue\_fro(struct Cirqu \*q)

{

    int x = -1;

    if (q->front == q->rear)

        printf("Cirqu is Empty\n");

    else

    {

        q->front = (q->front + 1) % q->size;

        x = q->Arr[q->front];

    }

    return x;

}

void Display(struct Cirqu q)

{

    int i = q.front + 1;

    while (i != (q.rear + 1) % q.size)

    {

        printf("%d ", q.Arr[i]);

        i = (i + 1) % q.size;

    }

    printf("\n");

}

int main()

{

    int n, t;

    struct Cirqu q;

    printf("\n Enter thesize of queue");

    int no;

    scanf("%d", &no);

    create(&q, no);

    while (n)

    {

        printf("Press 1 to enqueue using rear\n");

        printf("Press 2 to enqueue from front\n");

        printf("Press 3 to dequeue from front\n");

        printf("Press 4 to display the Circularqueue\n");

        scanf("%d", &t);

        switch (t)

        {

        case 1:

        {

            printf("\n Enter the no. to enque");

            int no;

            scanf("%d", &no);

            enqueue\_rear(&q, no);

            break;

        }

        case 2:

        {

            printf("\n Enter the no. to enque");

            int no;

            scanf("%d", &no);

            enqueue\_front(&q, no);

            break;

        }

        case 3:

            printf("\n  the no.which is removed from front is\n");

            printf("%d ", dequeue\_fro(&q));

            break;

        case 4:

            printf("\nQueue displayed is  \n");

            Display(q);

            break;

        default:

            break;

        }

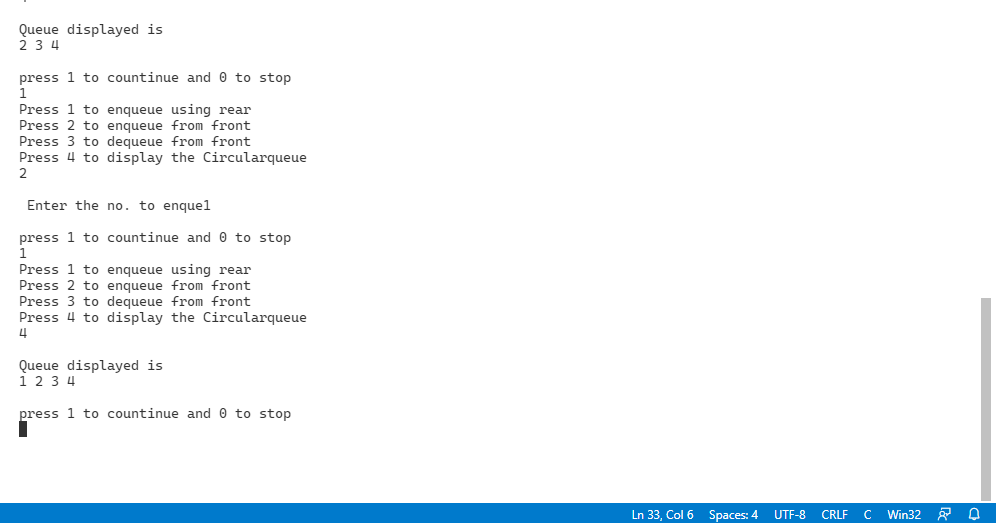
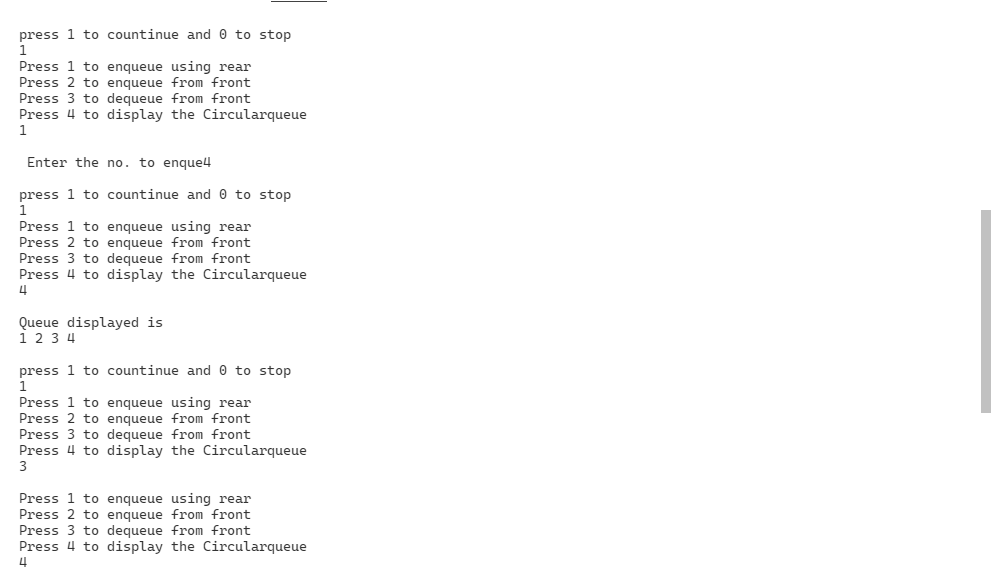
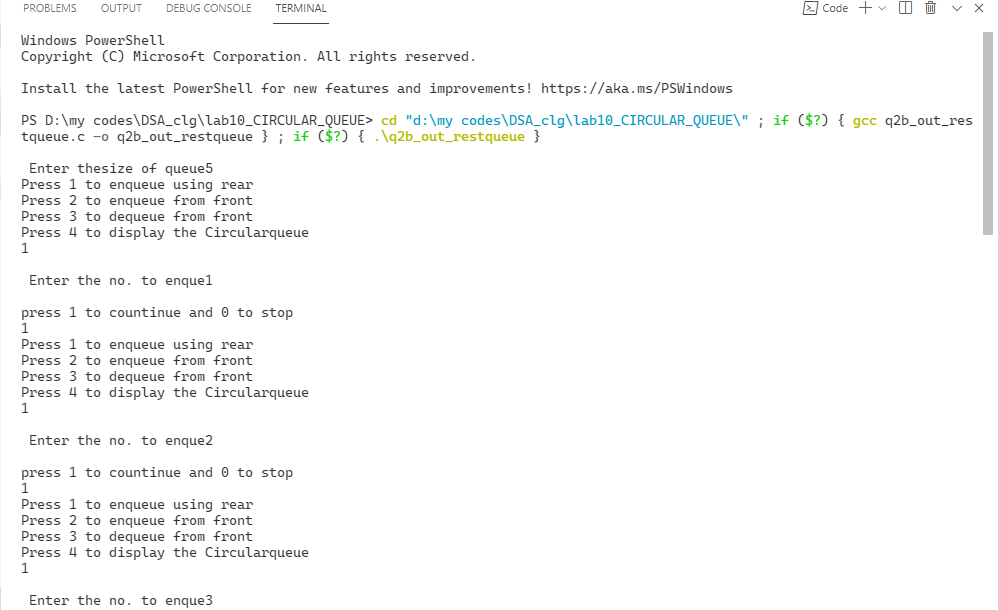
        printf("\npress 1 to countinue and 0 to stop\n");

        scanf("%d", &n);

    }

    return 0;

}



Lab 11

/\*•q1   WAP to find the height of a binary tree and to

 display the total no of nodes in a binary tree using recursion.\*/

#include <stdio.h>

#include <stdlib.h>

struct node\_025

{

    struct node\_025 \*lchild;

    int data;

    struct node\_025 \*rchild;

} \*root = NULL;

struct queue

{

    int size;

    int front;

    int rear;

    struct node\_025 \*\*Q;

};

void create(struct queue \*q, int size)

{

    q->size = size;

    q->front = q->rear = 0;

    q->Q = (struct node\_025 \*\*)malloc(q->size \* sizeof(struct node\_025 \*));

}

void enqueue(struct queue \*q, struct node\_025 \*x)

{

    if ((q->rear + 1) % q->size == q->front)

        printf("queue is Full");

    else

    {

        q->rear = (q->rear + 1) % q->size;

        q->Q[q->rear] = x;

    }

}

struct node\_025 \*dequeue(struct queue \*q)

{

    struct node\_025 \*x = NULL;

    if (q->front == q->rear)

        printf("queue is Empty\n");

    else

    {

        q->front = (q->front + 1) % q->size;

        x = q->Q[q->front];

    }

    return x;

}

int isempty(struct queue q)

{

    return q.front == q.rear;

}

void createtree()

{

    struct node\_025 \*p, \*t;

    int x;

    struct queue q;

    create(&q, 100);

    printf("Enter root value ");

    scanf("%d", &x);

    root = (struct node\_025 \*)malloc(sizeof(struct node\_025));

    root->data = x;

    root->lchild = root->rchild = NULL;

    enqueue(&q, root);

    while (!isempty(q))

    {

        p = dequeue(&q);

        printf("Enter left child of %d ", p->data);

        scanf("%d", &x);

        if (x != -1)

        {

            t = (struct node\_025 \*)malloc(sizeof(struct

                                             node\_025));

            t->data = x;

            t->lchild = t->rchild = NULL;

            p->lchild = t;

            enqueue(&q, t);

        }

        printf("Enter right child of %d ", p->data);

        scanf("%d", &x);

        if (x != -1)

        {

            t = (struct node\_025 \*)malloc(sizeof(struct

                                             node\_025));

            t->data = x;

            t->lchild = t->rchild = NULL;

            p->rchild = t;

            enqueue(&q, t);

        }

    }

}

int maxDepth(struct node\_025\* node\_025)

{

    if (node\_025 == NULL)

        return -1;

    else {

        int lnodeDepth\_025 = maxDepth(node\_025->lchild);

        int rnodeDepth\_025 = maxDepth(node\_025->rchild);

        if (lnodeDepth\_025 > rnodeDepth\_025)

            return (lnodeDepth\_025 + 1);

        else

            return (rnodeDepth\_025 + 1);

    }

}

void preorder(struct node\_025 \*p)

{

    if (p)

    {

        printf("%d ", p->data);

        preorder(p->lchild);

        preorder(p->rchild);

    }

}

void inorder(struct node\_025 \*p)

{

    if (p)

    {

        inorder(p->lchild);

        printf("%d ", p->data);

        inorder(p->rchild);

    }

}

void postorder(struct node\_025 \*p)

{

    if (p)

    {

        postorder(p->lchild);

        postorder(p->rchild);

        printf("%d ", p->data);

    }

}

int main()

{

    createtree();

     printf("\nPre Order ");

    preorder(root);

    printf("\nPost Order ");

    postorder(root);

     printf("\nIn Order ");

    inorder(root);

    printf("Height of tree is %d", maxDepth(root));

    return 0;

}



/\*•q2   Write the following menu driven program for the binary tree

----------------------------------------

2.Create\_BinaryTree\_Linked (using linked representation)

3. In-Order Traversal

4. Pre-Order Traversal

5. Post-Order traversal

\*/

#include <stdio.h>

#include <stdlib.h>

struct node\_025

{

    struct node\_025 \*lchild;

    int data;

    struct node\_025 \*rchild;

} \*root = NULL;

struct queue

{

    int size;

    int front;

    int rear;

    struct node\_025 \*\*Q;

};

void create(struct queue \*q, int size)

{

    q->size = size;

    q->front = q->rear = 0;

    q->Q = (struct node\_025 \*\*)malloc(q->size \* sizeof(struct node\_025 \*));

}

void enqueue(struct queue \*q, struct node\_025 \*x)

{

    if ((q->rear + 1) % q->size == q->front)

        printf("queue is Full");

    else

    {

        q->rear = (q->rear + 1) % q->size;

        q->Q[q->rear] = x;

    }

}

struct node\_025 \*dequeue(struct queue \*q)

{

    struct node\_025 \*x = NULL;

    if (q->front == q->rear)

        printf("queue is Empty\n");

    else

    {

        q->front = (q->front + 1) % q->size;

        x = q->Q[q->front];

    }

    return x;

}

int isempty(struct queue q)

{

    return q.front == q.rear;

}

void createtree()

{

    struct node\_025 \*p, \*t;

    int x;

    struct queue q;

    create(&q, 100);

    printf("Enter root value ");

    scanf("%d", &x);

    root = (struct node\_025 \*)malloc(sizeof(struct node\_025));

    root->data = x;

    root->lchild = root->rchild = NULL;

    enqueue(&q, root);

    while (!isempty(q))

    {

        p = dequeue(&q);

        printf("Enter left child of %d ", p->data);

        scanf("%d", &x);

        if (x != -1)

        {

            t = (struct node\_025 \*)malloc(sizeof(struct

                                             node\_025));

            t->data = x;

            t->lchild = t->rchild = NULL;

            p->lchild = t;

            enqueue(&q, t);

        }

        printf("Enter right child of %d ", p->data);

        scanf("%d", &x);

        if (x != -1)

        {

            t = (struct node\_025 \*)malloc(sizeof(struct

                                             node\_025));

            t->data = x;

            t->lchild = t->rchild = NULL;

            p->rchild = t;

            enqueue(&q, t);

        }

    }

}

void preorder(struct node\_025 \*p)

{

    if (p)

    {

        printf("%d ", p->data);

        preorder(p->lchild);

        preorder(p->rchild);

    }

}

void inorder(struct node\_025 \*p)

{

    if (p)

    {

        inorder(p->lchild);

        printf("%d ", p->data);

        inorder(p->rchild);

    }

}

void postorder(struct node\_025 \*p)

{

    if (p)

    {

        postorder(p->lchild);

        postorder(p->rchild);

        printf("%d ", p->data);

    }

}

int main()

{

    createtree();

     printf("\nPre Order ");

    preorder(root);

    printf("\nPost Order ");

    postorder(root);

     printf("\nIn Order ");

    inorder(root);

    return 0;

}



LAB 12

/\*• WAP Write the following menu driven program for the binary search tree

----------------------------------------

Binary search Tree Menu

----------------------------------------

0. Quit

1. Create

2. In-Order Traversal

3. Pre-Order Traversal

4. Post-Order traversal

5. search

6. Find Smallest Element

7. Find Largest Element

----------------------------------------

Enter your choice:

\*/

#include <stdio.h>

#include <stdlib.h>

struct BSTnode

{

    struct BSTnode \*lchild;

    int data;

    struct BSTnode \*rchild;

} \*root = NULL;

void insert(int key)

{

    struct BSTnode \*prev = root;

    struct BSTnode \*rear = NULL, \*newnode;

    if (root == NULL)

    {

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

        newnode->data = key;

        newnode->lchild = newnode->rchild = NULL;

        root = newnode;

        return;

    }

    while (prev != NULL)

    {

        rear = prev;

        if (key < prev->data)

            prev = prev->lchild;

        else if (key > prev->data)

            prev = prev->rchild;

        else

            return;

    }

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

    newnode->data = key;

    newnode->lchild = newnode->rchild = NULL;

    if (key < rear->data)

        rear->lchild = newnode;

    else

        rear->rchild = newnode;

}

void Inorder(struct BSTnode \*newnode)

{

    if (newnode)

    {

        Inorder(newnode->lchild);

        printf("%d ", newnode->data);

        Inorder(newnode->rchild);

    }

}

void preorder(struct BSTnode \*newnode)

{

    if (newnode)

    {

        printf("%d ", newnode->data);

        preorder(newnode->lchild);

        preorder(newnode->rchild);

    }

}

void postorder(struct BSTnode \*newnode)

{

    if (newnode)

    {

        postorder(newnode->lchild);

        postorder(newnode->rchild);

        printf("%d ", newnode->data);

    }

}

struct BSTnode \*search(int key)

{

    struct BSTnode \*prev = root;

    while (prev != NULL)

    {

        if (key == prev->data)

            return prev;

        else if (key < prev->data)

            prev = prev->lchild;

        else

            prev = prev->rchild;

    }

    return NULL;

}

struct BSTnode\*  search\_smal()

{       int key=root->data;

    struct BSTnode \*prev = root;

     struct BSTnode \*temp=prev;

    while (prev != NULL)

    {  temp=prev;

        prev=prev->lchild;

    }

    return temp;

}

struct BSTnode\*  search\_largest()

{       int key=root->data;

    struct BSTnode \*prev = root;

     struct BSTnode \*temp=prev;

    while (prev != NULL)

    {  temp=prev;

        prev=prev->rchild;

    }

    return temp;

}

int main()

{

    int n = 1, c,l=1;

    struct BSTnode \*temp;

    while (n)

    {

        printf("\n 0. Quit \n");

        printf("1. Create \n");

        printf("2. In-Order Traversal \n");

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

        printf(" 4. Post-Order traversal \n");

        printf(" 5. search \n");

        printf(" 6. Find Smallest Element \n");

        printf(" 7. Find Largest Element \n");

        scanf("%d", &c);

        switch (c)

        {

        case 1:

        while (l)

        {

            int j;

             printf("Enter the element ");

            scanf("%d", &j);

            insert(j);

              printf("\nPress 0 to exit and 1 to continue \n");

             scanf("%d", &l);

        }

            break;

        case 2:

            Inorder(root);

            break;

        case 3:

            preorder(root);

            break;

        case 4:

            postorder(root);

            break;

        case 5:

        {

            int j;

            printf("which element u want to search ");

            scanf("%d", &j);

            temp = search(j);

            if (temp != NULL)

                printf("element %d is found\n", temp->data);

            else

                printf("element is not found\n");

            break;

        }

        case 6:

        {

            temp = search\_smal();

            if (temp != NULL)

                printf("element %d is the smallest element\n", temp->data);

            break;

        }

        case 7:

        {

            temp = search\_largest();

            if (temp != NULL)

                printf("element %d is the largest element\n", temp->data);

            break;

        }

        default:

            break;

            printf("\nPress 0 to exit and 1 to continue \n");

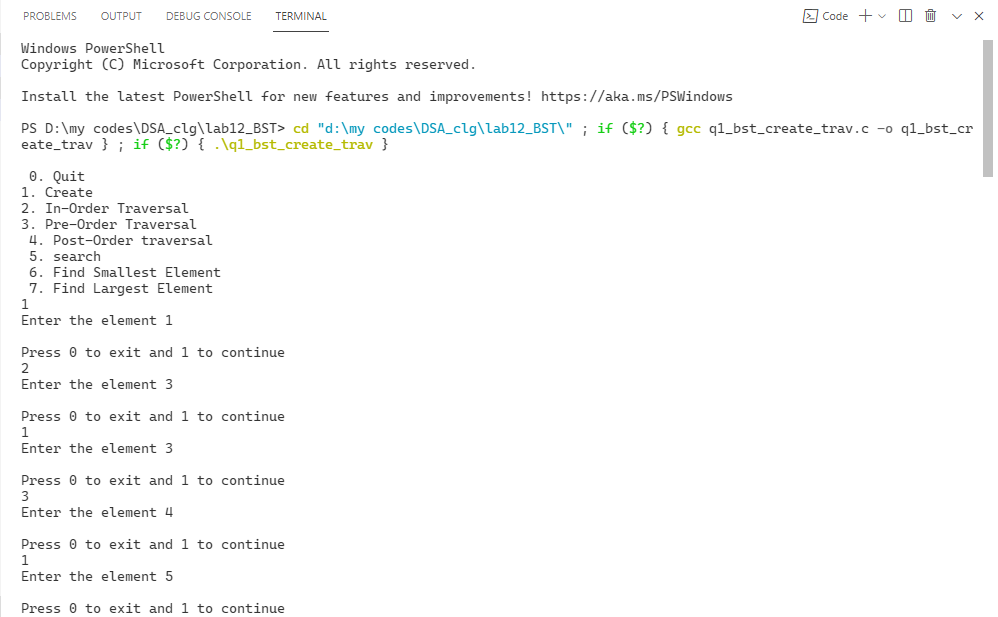
            scanf("%d", &n);

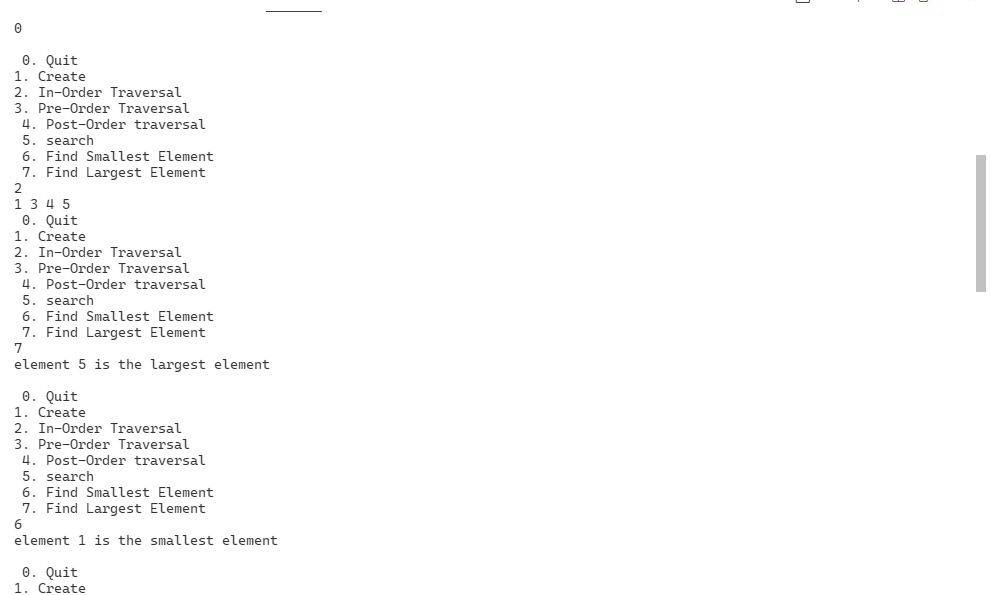
        }

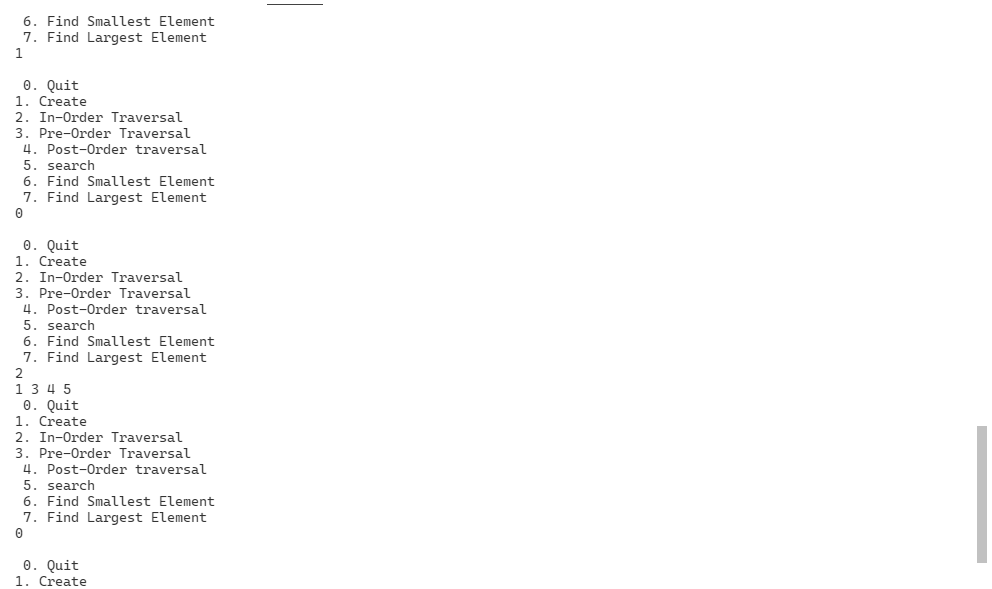
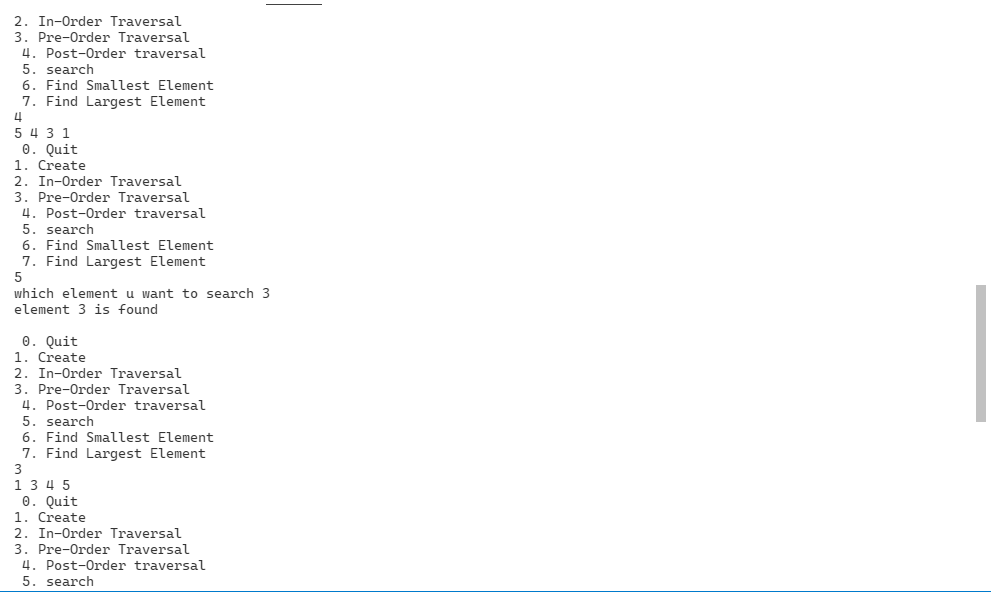
    }

    return 0;

}







/\*Q2•   Extend the above program by providing more options as follows:

a) To count number of leaf nodes in the tree.

b) To count number of non-leaf nodes in the tree.

c) To find number of nodes in the tree.

d) To find sum of all nodes of the tree.

e) To print depth of the tree.

f) To find nodes which are at maximum depth in the tree?

g) To print all the elements of kth level in single line.

h) To find the common ancestor and print the paths.

i) To check whether a tree is a binary search tree or not.

\*/

#include <stdio.h>

#include <stdlib.h>

int count = 0;

struct node

{

    struct node \*left;

    int data;

    struct node \*right;

};

struct node \*New (int x)

{

    struct node \*Temp;

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

    Temp->data = x;

    Temp->left = NULL;

    Temp->right = NULL;

    return Temp;

}

struct node \*insert(struct node \*root, int x)

{

    if (root == NULL)

        return New (x);

    else if (x > root->data)

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

    else

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

    return root;

}

int countLeaf(struct node \*root)

{

    if (root == NULL)

        return 0;

    if (root->left == NULL && root->right == NULL)

        return 1;

    else

        return countLeaf(root->left) + countLeaf(root->right);

}

int countNonleaf(struct node \*root)

{

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

        return 0;

    return 1 + countNonleaf(root->left) + countNonleaf(root->right);

}

int countNode(struct node \*root)

{

    return countLeaf(root) + countNonleaf(root);

}

int sumNodes(struct node \*root)

{

    if (root == NULL)

        return 0;

    return root->data + sumNodes(root->left) + sumNodes(root->right);

}

int maxDepth(struct node \*node)

{

    if (node == NULL)

        return -1;

    else

    {

        int lDepth = maxDepth(node->left);

        int rDepth = maxDepth(node->right);

        if (lDepth > rDepth)

            return (lDepth + 1);

        else

            return (rDepth + 1);

    }

}

void printlevel(struct node \*n, int desired, int current)

{

    if (n)

    {

        if (desired == current)

            printf("%d ", n->data);

        else

        {

            printlevel(n->left, desired, current + 1);

            printlevel(n->right, desired, current + 1);

        }

    }

}

int isBST(struct node \*root)

{

    static struct node \*prev = NULL;

    if (root)

    {

        if (!isBST(root->left))

            return 0;

        if (prev != NULL && root->data <= prev->data)

            return 0;

        prev = root;

        return isBST(root->right);

    }

    return 1;

}

void path(struct node \*root, int num)

{

    if (num > root->data)

    {

        printf("%d ", root->data);

        path(root->right, num);

    }

    else if (num < root->data)

    {

        printf("%d ", root->data);

        path(root->left, num);

    }

    else if (num == root->data)

    {

        printf("%d \n", root->data);

    }

}

struct node \*lca(struct node \*root, int n1, int n2)

{

    if (root == NULL)

        return NULL;

    if (root->data > n1 && root->data > n2)

        return lca(root->left, n1, n2);

    if (root->data < n1 && root->data < n2)

        return lca(root->right, n1, n2);

    return root;

}

struct node \*NewNode(int data)

{

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

    node->data = data;

    node->left = node->right = NULL;

    return (node);

}

int main()

{

    struct node \*root;

    int n1, n2;

    root = NULL;

    int c = 0;

    int temp = 0;

    int k = 0;

    while (1)

    {

        printf("\n1 - Insert a Node in BST.\n");

        printf("2 - Count number of leaf nodes.\n");

        printf("3 - Count number of non-leaf nodes.\n");

        printf("4 - Total number of nodes.\n");

        printf("5 - Sum of all nodes.\n");

        printf("6 - Depth of tree.\n");

        printf("7 - Nodes at maximum depth.\n");

        printf("8 - All elements at k-th level.\n");

        printf("9 - Find common ancestors and print the paths.\n");

        printf("10 - Check if BST or not.\n");

        printf("11 - Exit.\n");

        printf("Enter your choice : ");

        scanf("%d", &c);

        switch (c)

        {

            case 1:

                printf("Enter Value of Node to Insert in BST : ");

                scanf("%d", &temp);

                root = insert(root, temp);

                printf("Insertion Done.\n");

                break;

            case 2:

                printf("Number of leaf nodes in BST : ");

                printf("%d", countLeaf(root));

                break;

            case 3:

                printf("Number of non-leaf nodes in BST : ");

                printf("%d", countNonleaf(root));

                break;

            case 4:

                printf("Total number of nodes in BST : ");

                printf("%d", countNonleaf(root) + countLeaf(root));

                break;

            case 5:

                printf("Sum of all nodes in BST : ");

                printf("%d", sumNodes(root));

                break;

            case 6:

                printf("Depth of BST : ");

                printf("%d", maxDepth(root));

                break;

            case 7:

                printf("Nodes present at maximum depth in BST : ");

                printlevel(root, maxDepth(root), 0);

                break;

            case 8:

                printf("Enter value of k : ");

                scanf("%d", &k);

                printf("\nNodes present at %d-th level in BST : ");

                printlevel(root, k, 0);

                break;

            case 9:

               { printf("Enter value of Nodes : ");

                scanf("%d", &n1);

                scanf("%d", &n2);

                struct node \*t = lca(root, n1, n2);

                printf("LCA of %d and %d is %d \n", n1, n2, t->data);

                printf("Path between %d (LCA) and %d is : ",t->data,n1);

                path(t,n1);

                printf("Path between %d (LCA) and %d is : ",t->data,n2);

                path(t,n2);

                break;

               }

            case 10:

                if (isBST(root) == 1)

                {

                    printf("Given tree is a BST.\n");

                }

                else

                {

                    printf("Given tree is not a BST.\n");

                }

                break;

            case 11:

                printf("Code Exited.\n");

                exit(1);

            default:

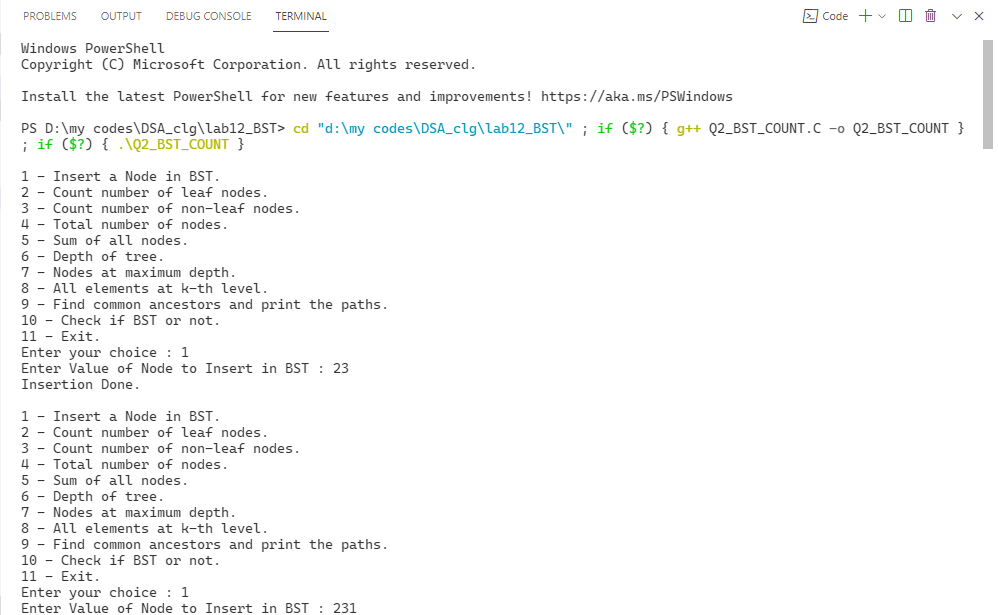
                printf("Wrong Choice, Try again!\n");

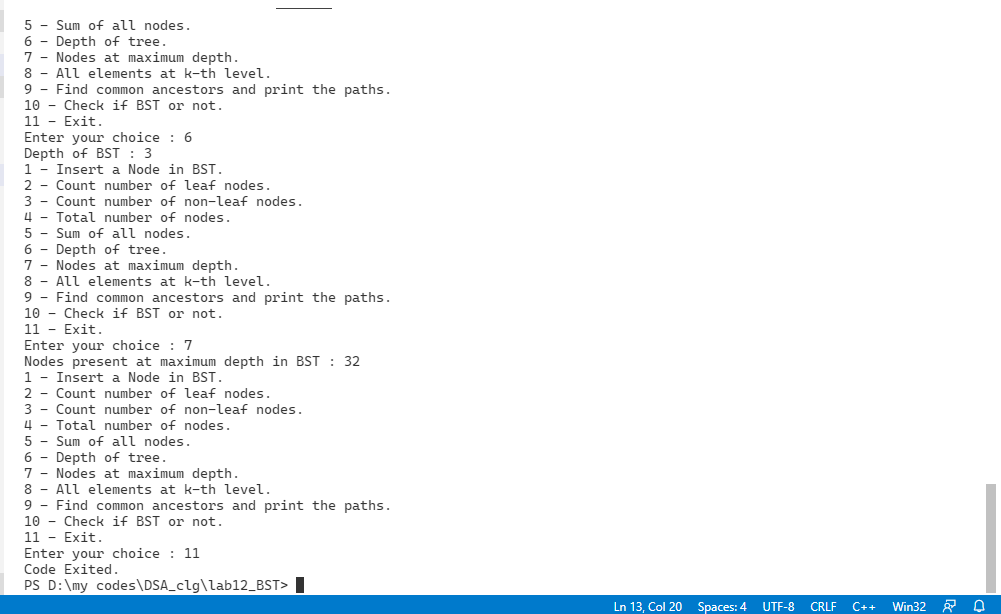
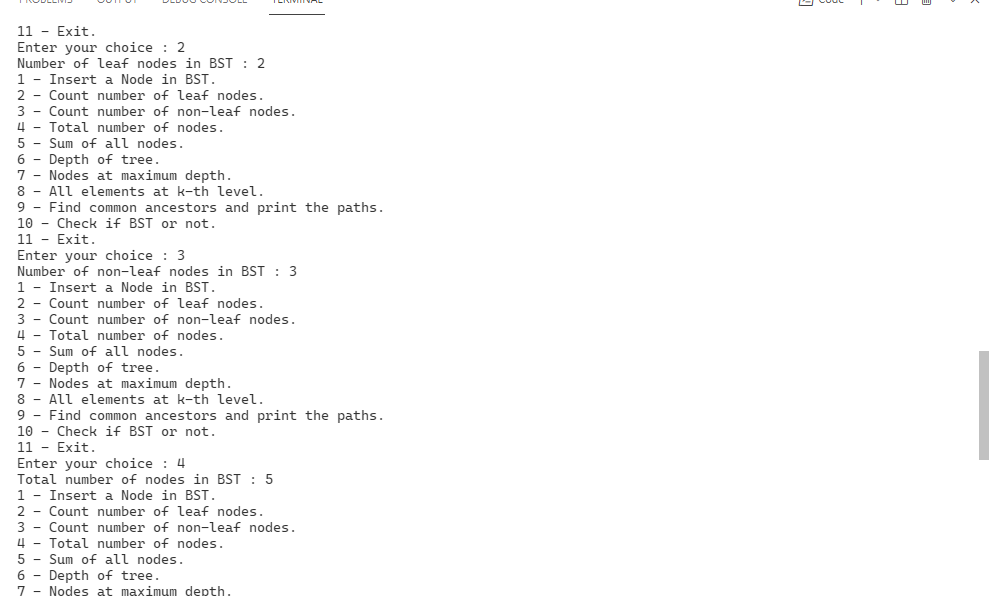
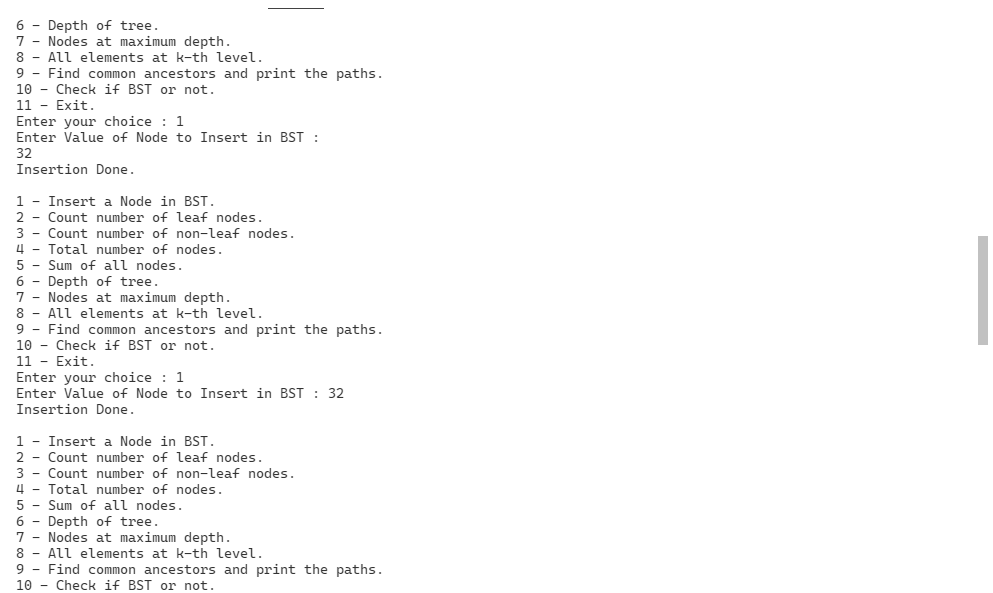
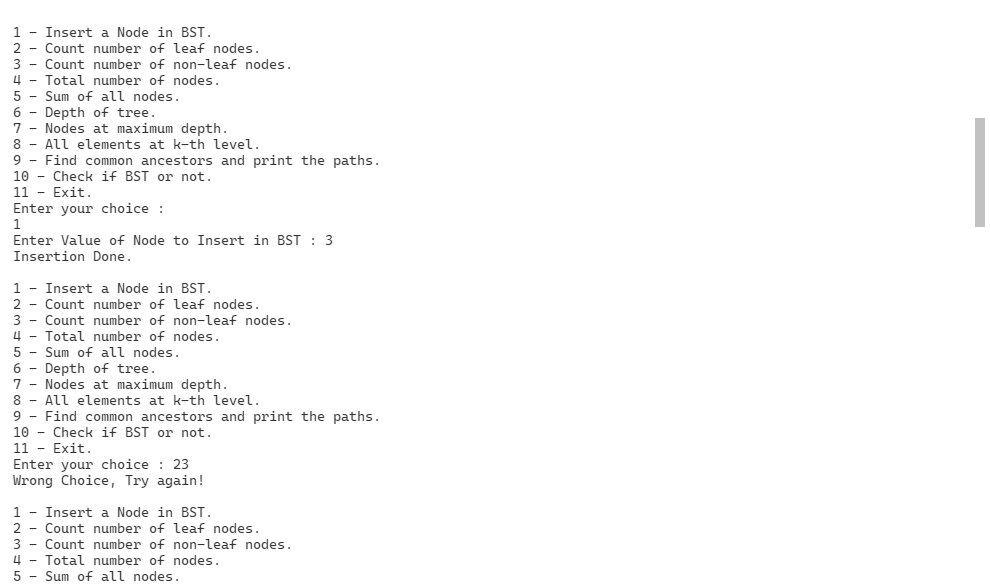
        }

    }

    return 0;

}





LAB 13

/\*1.WAP using C

I. To create a BST

II.  Display the elements using Level order Traversal

III. Delete the leaf node and print it .

IV. Delete a node which has only one child and readjust the BST

V. Delete the node whose degree is 2 and display the Deleted node,it’s inorder predecessor and inorder successor and display all nodes in Inorder traversal after readjustment of BST

\*/

/\* level order traversal \*/

#include <stdio.h>

#include <stdlib.h>

struct Node

{

    struct Node \*lchild;

    int data;

    struct Node \*rchild;

} \*root = NULL;

struct Queue

{

    int size;

    int front;

    int rear;

    struct Node \*\*Q;

};

void create(struct Queue \*q, int size)

{

    q->size = size;

    q->front = q->rear = 0;

    q->Q = (struct Node \*\*)malloc(q->size \* sizeof(struct Node \*));

}

void enqueue(struct Queue \*q, struct Node \*x)

{

    if ((q->rear + 1) % q->size == q->front)

        printf("Queue is full\n");

    else

    {

        q->rear = q->rear + 1 % q->size;

        q->Q[q->rear] = x;

    }

}

struct Node \*dequeue(struct Queue \*q)

{

    struct Node \*x = NULL;

    if (q->front == q->rear)

        printf("Queue is empty\n");

    else

    {

        q->front = (q->front + 1) % q->size;

        x = q->Q[q->front];

    }

    return x;

}

int isEmpty(struct Queue q)

{

    return q.front == q.rear;

}

struct Node \*insert(struct Node \*p, int key)

{

    struct Node \*t;

    if (p == NULL)

    {

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

        t->data = key;

        t->lchild = t->rchild = NULL;

        return t;

    }

    if (key < p->data)

        p->lchild = insert(p->lchild, key);

    else if (key > p->data)

        p->rchild = insert(p->rchild, key);

    return p;

}

void levelorder(struct Node \*r)

{

    struct Queue q;

    create(&q, 100);

    printf("%d ", r->data);

    enqueue(&q, root);

    while (!isEmpty(q))

    {

        root = dequeue(&q);

        if (root->lchild)

        {

            printf("%d ", root->lchild->data);

            enqueue(&q, root->lchild);

        }

        if (root->rchild)

        {

            printf("%d ", root->rchild->data);

            enqueue(&q, root->rchild);

        }

    }

}

void Inorder(struct Node \*p)

{

    if (p)

    {

        Inorder(p->lchild);

        printf("%d ", p->data);

        Inorder(p->rchild);

    }

}

int Height(struct Node \*p)

{

    int x, y;

    if (p == NULL)

        return 0;

    x = Height(p->lchild);

    y = Height(p->rchild);

    return x > y ? x + 1 : y + 1;

}

struct Node \*InPredecessor(struct Node \*p)

{

    while (p && p->rchild)

        p = p->rchild;

    return p;

}

struct Node \*InSuccessor(struct Node \*p)

{

    while (p && p->lchild)

        p = p->lchild;

    return p;

}

struct Node \*Delete (struct Node \*p, int key)

{

    struct Node \*q;

    if (p == NULL)

        return NULL;

    if (p->lchild == NULL && p->rchild == NULL)

    {

        if (p == root)

            root = NULL;

        free(p);

        return NULL;

    }

    if (key < p->data)

        p->lchild = Delete (p->lchild, key);

    else if (key > p->data)

        p->rchild = Delete (p->rchild, key);

    else

    {

        if (Height(p->lchild) > Height(p->rchild))

        {

            q = InPredecessor(p->lchild);

            p->data = q->data;

            p->lchild = Delete (p->lchild, q->data);

        }

        else

        {

            q = InSuccessor(p->rchild);

            p->data = q->data;

            p->rchild = Delete (p->rchild, q->data);

        }

    }

    return p;

}

int main()

{

    int ch, x;

    printf("Enter root node data: ");

    scanf("%d", &x);

    root = insert(root, x);

    do

    {

        printf("Enter data: ");

        scanf("%d", &x);

        insert(root, x);

        printf("Do you want more nodes? (1/0): ");

        scanf("%d", &ch);

    } while (ch != 0);

    printf("\nLevel order traversal is\n");

    levelorder(root);

    do

    {

        printf("\nEnter node to be Deleted: ");

        scanf("%d", &x);

        Delete (root, x);

        printf("\nInorder traversal after deletion\n");

        Inorder(root);

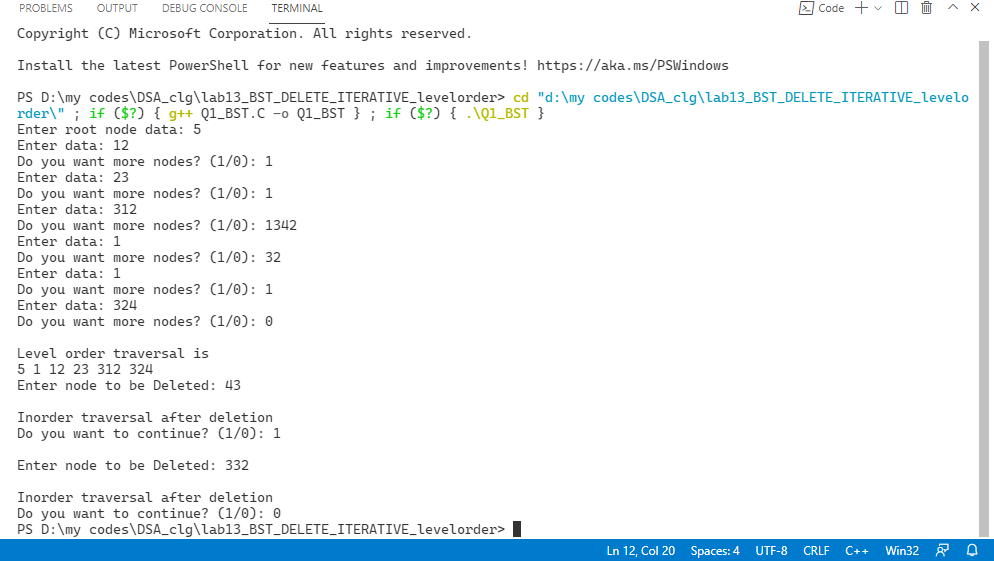
        printf("Do you want to continue? (1/0): ");

        scanf("%d", &ch);

    } while (ch != 0);

    return 0;

}



/\*2.WAP to create a Binary Tree and

 display all the nodes using Iterative Version of all types of traversals using Stack data structure.

\*/

#include <stdio.h>

#include <stdlib.h>

struct Node

{

    struct Node \*lchild;

    int data;

    struct Node \*rchild;

} \*root = NULL;

struct Stack

{

    int size;

    int top;

    struct Node \*\*S;

};

struct Queue

{

    int size;

    int front;

    int rear;

    struct Node \*\*Q;

};

void stackCreate(struct Stack \*st, int size)

{

    st->size = size;

    st->top = -1;

    st->S = (struct Node \*\*)malloc(st->size \* sizeof(struct Node \*));

}

void push(struct Stack \*st, struct Node \*x)

{

    if (st->top == st->size - 1)

        printf("Stack overflow\n");

    else

    {

        st->top++;

        st->S[st->top] = x;

    }

}

struct Node \*pop(struct Stack \*st)

{

    struct Node \*x = NULL;

    if (st->top == -1)

        printf("Stack underflow\n");

    else

        x = st->S[st->top--];

    return x;

}

int isEmptyStack(struct Stack st)

{

    if (st.top == -1)

        return 1;

    return 0;

}

void create(struct Queue \*q, int size)

{

    q->size = size;

    q->front = q->rear = 0;

    q->Q = (struct Node \*\*)malloc(q->size \* sizeof(struct Node \*));

}

void enqueue(struct Queue \*q, struct Node \*x)

{

    if ((q->rear + 1) % q->size == q->front)

        printf("Queue is full\n");

    else

    {

        q->rear = q->rear + 1 % q->size;

        q->Q[q->rear] = x;

    }

}

struct Node \*dequeue(struct Queue \*q)

{

    struct Node \*x = NULL;

    if (q->front == q->rear)

        printf("Queue is empty\n");

    else

    {

        q->front = (q->front + 1) % q->size;

        x = q->Q[q->front];

    }

    return x;

}

int isEmpty(struct Queue q)

{

    return q.front == q.rear;

}

void treeCreate()

{

    struct Node \*p, \*t;

    int x;

    struct Queue q;

    create(&q, 100);

    printf("Enter root value: ");

    scanf("%d", &x);

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

    root->data = x;

    root->lchild = root->rchild = NULL;

    enqueue(&q, root);

    while (!isEmpty(q))

    {

        p = dequeue(&q);

        printf("Enter left child of %d: ", p->data);

        scanf("%d", &x);

        if (x != -1)

        {

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

            t->data = x;

            t->lchild = t->rchild = NULL;

            p->lchild = t;

            enqueue(&q, t);

        }

        printf("Enter right child of %d: ", p->data);

        scanf("%d", &x);

        if (x != -1)

        {

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

            t->data = x;

            t->lchild = t->rchild = NULL;

            p->rchild = t;

            enqueue(&q, t);

        }

    }

}

void preorder(struct Node \*p)

{

    struct Stack stk;

    stackCreate(&stk, 100);

    while (p || !isEmptyStack(stk))

    {

        if (p)

        {

            printf("%d ", p->data);

            push(&stk, p);

            p = p->lchild;

        }

        else

        {

            p = pop(&stk);

            p = p->rchild;

        }

    }

}

void inorder(struct Node \*p)

{

    struct Stack stk;

    stackCreate(&stk, 100);

    while (p || !isEmptyStack(stk))

    {

        if (p)

        {

            push(&stk, p);

            p = p->lchild;

        }

        else

        {

            p = pop(&stk);

            printf("%d ", p->data);

            p = p->rchild;

        }

    }

}

void postorder(struct Node \*p)

{

    struct Stack stk;

    stackCreate(&stk, 100);

    long int temp;

    while (p || !isEmptyStack(stk))

    {

        if (p)

        {

            push(&stk, p);

            p = p->lchild;

        }

        else

        {

            temp = (long int)pop(&stk);

            if (temp > 0)

            {

                push(&stk, (struct Node \*)(-temp));

                p = ((struct Node \*)temp)->rchild;

            }

            else

            {

                printf("%d ", ((struct Node \*)-temp)->data);

                p = NULL;

            }

        }

    }

}

int main()

{

    treeCreate();

    printf("\nPreorder traversal is\n");

    preorder(root);

    printf("\nInorder traversal is\n");

    inorder(root);

    printf("\nPostorder traversal is\n");

    postorder(root);

    return 0;

}



LAB -14

// q1 sort day month and year(structre type) using bubble sort

#include <stdio.h>

struct date\_sort

{

    int day;

    int month;

    int year;

};

void sort\_year(struct date\_sort a[], int n)

{

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n - 1 - i; j++)

        {

            if (a[j].year > a[j + 1].year)

            {

                struct date\_sort temp = a[j];

                a[j] = a[j + 1];

                a[j + 1] = temp;

            }

            else if (a[j].month > a[j + 1].month&& a[j].year == a[j + 1].year)

            {

                struct date\_sort temp = a[j];

                a[j] = a[j + 1];

                a[j + 1] = temp;

            }

            else if (a[j].day > a[j + 1].day&& a[j].year == a[j + 1].year&&a[j].month == a[j + 1].month)

            {

                struct date\_sort temp = a[j];

                a[j] = a[j + 1];

                a[j + 1] = temp;

            }

        }

    }

}

int main()

{

    struct date\_sort a[5];

    for (int i = 0; i < 5; i++)

    {

        scanf("%d", &a[i].day);

        scanf("%d", &a[i].month);

        scanf("%d", &a[i].year);

    }

  /\* 2 3 2003

    4 5 2008

    5 6 2001

    6 6 2001

    1 6 2001\*/

    sort\_year(a, 5);

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

    for (int i = 0; i < 5; i++)

    {

        printf("%d %d %d \n", a[i].day, a[i].month, a[i].year);

    }

    return 0;

}



/\*q2•WAP to sort an array of n dates in an ascending order using Bubble sort. Date structure is {day, month, year }

\*/

#include <stdio.h>

int  main()

{

    int a[100];

   int num;

    printf("Enter the value of num \n");

    scanf("%d", &num);

    printf("Enter the elements \n");

    for (int i = 0; i < num; i++)

    {

        scanf(" %d",&a[i]);

    }

       for (int i = 0; i < num-1; i++)

    {

        for (int j = i; j<num-1; j++)

        {

            if (a[i]> a[j + 1])

            {

              int temp = a[i];

                a[i] = a[j + 1];

                a[j + 1] = temp;

            }

        }

    }

    printf("after seelection sort ascending  array is \n");

    for (int i = 0; i < num; i++)

    {

        printf("%d ", a[i]);

    }

    return 0;

}



// q3 WAP to sort an array of n integers in a descending order using insertion sort.

#include <stdio.h>

int main()

{

    int a[100];

    int num;

    printf("Enter the value of num \n");

    scanf("%d", &num);

    printf("Enter the elements \n");

    for (int i = 0; i < num; i++)

    {

        scanf(" %d", &a[i]);

    }

    for (int i = 0; i < num ; i++)

    {

        int back = a[i+1];

        int j = i+1;

        for (; j > -1; j--)

        {

            if (a[j] > back)

            {

                a[j+1] = a[j];

            }

        }

        a[j+1] = back;

    }

    printf("after insertion sort ascending  array is \n");

    for (int i = 0; i < num; i++)

    {

        printf("%d ", a[i]);

    }

    return 0;

}



//4-WAP demonstrating bubble sort using linked list.

#include<stdio.h>

#include<stdlib.h>

struct Node\_025

{

    int data;

    struct Node\_025 \*next;

};

void insertAtTheBegin(struct Node\_025 \*\*start\_ref, int data);

void bubbleSort(struct Node\_025 \*start);

void swap(struct Node\_025 \*a, struct Node\_025 \*b);

void printList(struct Node\_025 \*start);

int main()

{

    int arr[50],n,i;

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

    scanf("%d",&n);

    struct Node\_025 \*start = NULL;

    printf("Insert those elements");

    for (i = 0; i< n; i++)

    {

        scanf("%d",&arr[i]);

        insertAtTheBegin(&start, arr[i]);

    }

    printf("\nLinked list before sorting ");

    printList(start);

    bubbleSort(start);

    printf("\nLinked list after sorting ");

    printList(start);

    getchar();

    return 0;

}

void insertAtTheBegin(struct Node\_025 \*\*start\_ref, int data)

{

    struct Node\_025 \*ptr1 = (struct Node\_025\*)malloc(sizeof(struct Node\_025));

    ptr1->data = data;

    ptr1->next = \*start\_ref;

    \*start\_ref = ptr1;

}

void printList(struct Node\_025 \*start)

{

    struct Node\_025 \*temp = start;

    printf("\n");

    while (temp!=NULL)

    {

        printf("%d ", temp->data);

        temp = temp->next;

    }

}

void bubbleSort(struct Node\_025 \*start)

{

    int swapped, i;

    struct Node\_025 \*ptr1;

    struct Node\_025 \*lptr = NULL;

    if (start == NULL)

        return;

    do

    {

        swapped = 0;

        ptr1 = start;

        while (ptr1->next != lptr)

        {

            if (ptr1->data > ptr1->next->data)

            {

                swap(ptr1, ptr1->next);

                swapped = 1;

            }

            ptr1 = ptr1->next;

        }

        lptr = ptr1;

    }

    while (swapped);

}

void swap(struct Node\_025 \*a, struct Node\_025 \*b)

{

    int temp = a->data;

    a->data = b->data;

    b->data = temp;

}



//6-WAP sort the n names in an alphabetical order.

#include <stdio.h>

#include <string.h>

void main()

{

    char name[10][8], tname[10][8], temp[8];

    int i, j, n;

    printf("Enter the number of names to enter: ");

    scanf("%d", &n);

    printf("Enter %d names: \n", n);

    for (i = 0; i < n; i++)

    {

        scanf("%s", name[i]);

        strcpy(tname[i], name[i]);

    }

    for (i = 0; i < n - 1 ; i++)

    {

        for (j = i + 1; j < n; j++)

        {

            if (strcmp(name[i], name[j]) > 0)

            {

                strcpy(temp, name[i]);

                strcpy(name[i], name[j]);

                strcpy(name[j], temp);

            }

        }

    }

    printf("\n");

    printf("Input Names\tSorted names\n");

    printf("\n");

    for (i = 0; i < n; i++)

    {

        printf("%s\t\t%s\n", tname[i], name[i]);

    }

    printf("\n");

}

