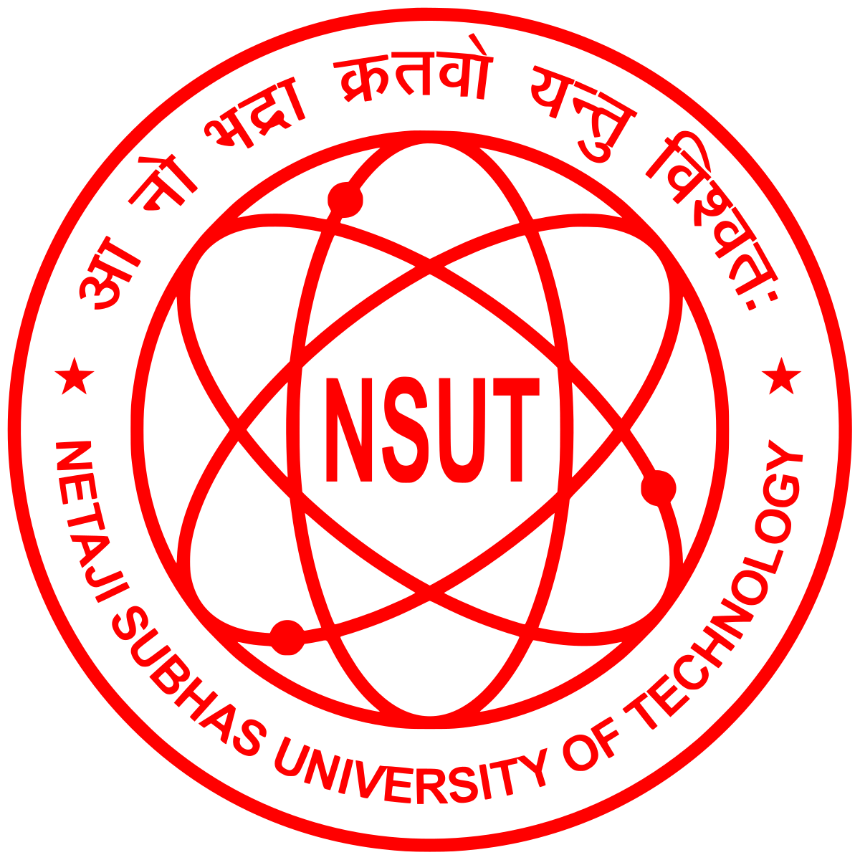
**Data Structures Lab File**



Name: Prajwal Krishnamurthy Bhat

Class: CSE-1

Roll number: 2021UCS1609

1. **Write a program to find the mean and the median of the numbers stored in an array.**

Source Code:

#include<iostream>

#include<algorithm>

using namespace std;

double Mean(int a[], int n){

    int sum = 0;

    for (int i = 0; i < n; i++)

        sum += a[i];

    return (double)sum / (double)n;

}

double Median(int a[], int n){

    sort(a, a + n);

    if (n % 2 != 0)

        return (double)a[n / 2];

    return (double)(a[(n - 1) / 2] + a[n / 2]) / 2.0;

}

int main(){

    int a[] = {10,31,34,56,77,85,23,19};

    int n = sizeof(a) / sizeof(a[0]);

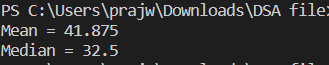
    cout << "Mean = " << Mean(a, n) <<endl;

    cout << "Median = " << Median(a, n) <<endl;

    return 0;

}

Output:



1. **Write a program to insert one element in an array and delete an element from an array.**

Source Code:

#include<bits/stdc++.h>

using namespace std;

int main(){

    int arr[100];

    cout<<"Enter size of array ";

    int n;cin>>n;cout<<endl;

    cout<<"Enter array- ";

    for(int i=0;i<n;i++){

        cin>>arr[i];

    }

    cout<<endl;

    cout<<"Enter element to insert ";

    int k;cin>>k;

    arr[n]=k;

    cout<<endl;

    cout<<"Array is-"<<endl;

    for(int i=0;i<=n;i++){

        cout<<arr[i]<<" ";

    }

    cout<<endl;

    cout<<"Enter element to delete ";

    int del;cin>>del;

    int pos;

    for(int i=0;i<=n;i++){

        if(arr[i]==del){

            pos=i;

        }

    }

    for(int i=pos;i<n;i++){

        arr[i]=arr[i+1];

    }

    cout<<"Array is-"<<endl;

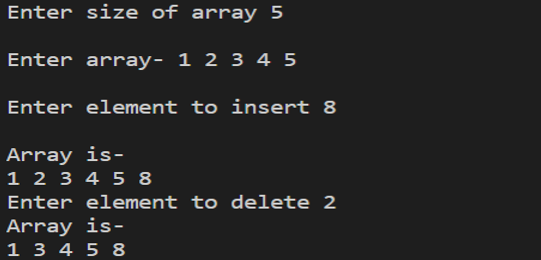
    for(int i=0;i<n;i++){

        cout<<arr[i]<<" ";

    }

    return 0;

Output:



1. **Write a program to search for a number in an array.**

Source Code:

#include<bits/stdc++.h>

using namespace std;

int main(){

    int n;

    cout<<"Enter size of array ";

    cin>>n;

    cout<<"Enter array "<<endl;

    int arr[n];

    for(int i=0;i<n;i++){

        cin>>arr[i];

    }

    cout<<endl;

    int key;

    cout<<"Enter elements to search ";

    cin>>key;

    int pos=-1;

    for(int i=0;i<n;i++){

        if(arr[i]==key){

            pos=i;

        }

    }

    if(pos==-1){

        cout<<"Element not found"<<endl;

    }

    else{

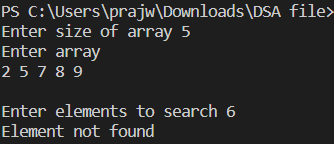
        cout<<"Element found at "<<pos+1<<endl;

    }

    return 0;

}

Output:



1. **Write a program to sort an array.**

Source Code:

#include<iostream>

using namespace std;

int main(){

    cout<<"Enter number of elements ";

    int n;

    cin>>n;

    cout<<"Enter array "<<endl;

    int arr[n];

    for (int i = 0; i<n; i++){

        cin>>arr[i];

    }

    for (int i = 0; i<n; i++){

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

            if (arr[j] > arr[j+1])

            {

                int temp = arr[j];

                arr[j] = arr[j+1];

                arr[j+1] = temp;

            }

        }

    }

    cout<<"Sorted array "<<endl;

    for (int k = 0; k<n; k++){

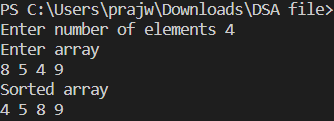
        cout<<arr[k]<<" ";

    }

    return 0;

}

Output:



1. **Write a program to merge two sorted arrays.**

Source Code:

#include<iostream>

using namespace std;

void mergeArrays(int arr1[], int arr2[], int n1,

                            int n2, int arr3[]){

    int i = 0, j = 0, k = 0;

    while (i<n1 && j <n2){

        if (arr1[i] < arr2[j])

            arr3[k++] = arr1[i++];

        else

            arr3[k++] = arr2[j++];

    }

    while (i < n1){

        arr3[k++] = arr1[i++];

    }

    while (j < n2){

        arr3[k++] = arr2[j++];

    }

}

int main(){

    int arr1[] = {12, 34, 65, 87, 98};

    int n1 = sizeof(arr1) / sizeof(arr1[0]);

    int arr2[] = {23, 44, 45, 76, 80};

    int n2 = sizeof(arr2) / sizeof(arr2[0]);

    int arr3[n1+n2];

    mergeArrays(arr1, arr2, n1, n2, arr3);

    cout << "Merged array: " <<endl;

    for (int i=0; i < n1+n2; i++)

        cout << arr3[i] << " ";

    return 0;

}

Output:



1. **Write a program to store the marks obtained by 10 students in 5 courses in a two-dimensional array.**

Source Code:

#include<bits/stdc++.h>

using namespace std;

int main(){

    int arr[10][5];

    for(int i=0;i<10;i++){

        for(int j=0;j<5;j++){

            cout<<"Enter marks of student "<<i+1<<" in subject "<<j+1<<" ";

            cin>>arr[i][j];

        }

    }

    return 0;

}

1. **Write a program to implement a linked list.**

Source Code:

#include<iostream>

using namespace std;

class node{

    public:

    int data;

    node\* next;

    node(int val){

        data = val;

        next = NULL;

    }

};

void AddtoTail(node\* head,int val){

    node \*n = new node(val);

    if (head==NULL){

        head = n;

        return;

    }

    node \*temp = head;

    while(temp->next!=NULL){

        temp = temp->next;

    }

    temp->next = n;

}

void display(node\* head){

    node\* temp = head;

    while(temp!=NULL){

        cout<<(temp->data)<<"->";

        temp = temp->next;

    }

    cout<<"NULL";

}

int main(){

    node \* head = new node(5);

    AddtoTail(head,3);

    AddtoTail(head,2);

    AddtoTail(head,1);

    AddtoTail(head,2);

    cout<<"initial list-"<<endl;

    display(head);

    cout<<endl;

    return 0;

}

Output:



1. **Write a program to insert a node in a linked list and delete a node from a linked**

**list.**

Source Code:

#include<iostream>

using namespace std;

class node{

    public:

    int data;

    node\* next;

    node(int val){

        data = val;

        next = NULL;

    }

};

void AddtoTail(node\* head,int val){

    node \*n = new node(val);

    if (head==NULL){

        head = n;

        return;

    }

    node \*temp = head;

    while(temp->next!=NULL){

        temp = temp->next;

    }

    temp->next = n;

}

void InsertVal(node\* &head,int pos,int val){

    node\* n= new node(val);

    node\* temp= head;

    int ind=1;

    while(ind<(pos-1) && temp!=NULL){

        temp=temp->next;

        ind++;

    }

    n->next=temp->next;

    temp->next=n;

}

void deleteNode(node\* head, int val){

    node\* temp = head;

    if (head==NULL){

        return;

    }

    if (head->next == NULL){

        deleteHead(head);

        return;

    }

    while(temp->next->data!=val){

        temp = temp->next;

    }

    node\* todelete = temp->next;

    temp->next = temp->next->next;

    delete todelete;

}

int main(){

    node \* head = new node(5);

    AddtoTail(head,3);

    AddtoTail(head,2);

    AddtoTail(head,1);

    AddtoTail(head,2);

    cout<<"initial list-"<<endl;

    display(head);

    cout<<endl;

    int val,pos;

    cout<<"enter position to insert element "<<endl;

    cin>>pos;

    cout<<"enter value "<<endl;

    cin>>val;

    InsertVal(head,pos,val);

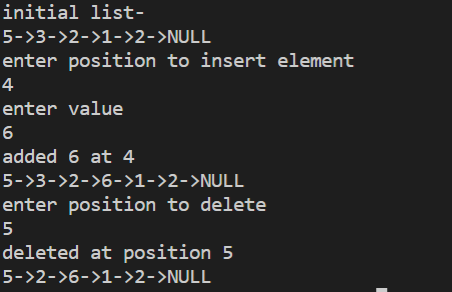
    cout<<"added "<<val<<" at "<<pos<<endl;

    display(head);

return 0;

}

Output:



1. **Write a program to print the elements of a linked list in reverse order without**

**disturbing the linked list.**

Source Code:

node\* reverse(node\* head){

    node\* curr = head;

    node\* prev = NULL;

    node\* second;

    while(curr!=NULL){

        second = curr->next;

        curr->next = prev;

        prev = curr;

        curr = second;

    }

    return prev;

}

int main(){

    node \* head = new node(5);

    AddtoTail(head,3);

    AddtoTail(head,2);

    AddtoTail(head,1);

    AddtoTail(head,2);

    cout<<"initial list-"<<endl;

    display(head);

    cout<<endl;

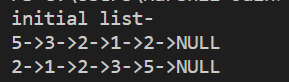
    node\* rev=reverse(head);

    display(rev);

    return 0;

}

Output:



1. **Write a program to add two polynomials using linked lists.**

Source Code:

#include <iostream>

using namespace std;

struct Node {

    int data;

    struct Node\* next;

    Node(int data)

    {

        this->data = data;

        next = NULL;

    }

};

struct LinkedList {

    Node\* head;

    LinkedList() { head = NULL; }

    void reverse(){

        Node\* current = head;

        Node \*prev = NULL, \*next = NULL;

        while (current != NULL) {

            next = current->next;

            current->next = prev;

            prev = current;

            current = next;

        }

        head = prev;

    }

    void print(){

        struct Node\* temp = head;

        while(temp!=NULL){

        cout<<(temp->data)<<"->";

        temp = temp->next;

    }

    cout<<"NULL";

}

    void push(int data){

        Node\* temp = new Node(data);

        temp->next = head;

        head = temp;

    }

};

int main()

{

    LinkedList ll;

    ll.push(5);

    ll.push(4);

    ll.push(3);

    ll.push(2);

    cout << "Given linked list\n";

    ll.print();

    ll.reverse();

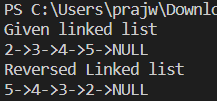
    cout << "\nReversed Linked list \n";

    ll.print();

    return 0;

}

Output:



1. **Write a program to add two polynomials using linked lists.**

Source Code:

#include<iostream>

using namespace std;

struct Node {

    int cf;

    int pow;

    struct Node\* next;

};

void create\_node(int x, int y, struct Node\*\* temp){

    struct Node \*r, \*z;

    z = \*temp;

    if (z == NULL){

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

        r->cf = x;

        r->pow = y;

        \*temp = r;

        r->next = (struct Node\*)malloc(sizeof(struct Node));

        r = r->next;

        r->next = NULL;

    }

    else{

        r->cf = x;

        r->pow = y;

        r->next = (struct Node\*)malloc(sizeof(struct Node));

        r = r->next;

        r->next = NULL;

    }

}

void polyadd(struct Node\* a, struct Node\* b,

            struct Node\* poly){

    while (a->next && b->next){

        if (a->pow > b->pow) {

            poly->pow = a->pow;

            poly->cf = a->cf;

            a = a->next;

        }

        else if (a->pow < b->pow) {

            poly->pow = b->pow;

            poly->cf = b->cf;

            b = b->next;

        }

        else{

            poly->pow = a->pow;

            poly->cf = a->cf + b->cf;

            a = a->next;

            b = b->next;

        }

        poly->next

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

        poly = poly->next;

        poly->next = NULL;

    }

    while (a->next || b->next){

        if (a->next){

            poly->pow = a->pow;

            poly->cf = a->cf;

            a = a->next;

        }

        if (b->next)

            poly->pow = b->pow;

            poly->cf = b->cf;

            b = b->next;

        }

        poly->next

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

        poly = poly->next;

        poly->next = NULL;

    }

void show(struct Node\* node){

    while (node->next != NULL){

        printf("%dx^%d", node->cf, node->pow);

        node = node->next;

        if (node->cf >= 0){

            if (node->next != NULL)

                printf("+");

        }

    }

}

int main()

{

    struct Node \*a = NULL, \*b = NULL, \*poly = NULL;

    create\_node(7, 2, &a);

    create\_node(4, 4, &a);

    create\_node(2, 5, &a);

    create\_node(-5, 2, &b);

    create\_node(-9, 0, &b);

    printf("1st Polynomial: ");

    show(a);

    printf("\n2nd Polynomial: ");

    show(b);

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

    polyadd(a, b, poly);

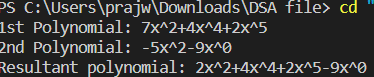
    printf("\nResultant polynomial: ");

    show(poly);

    return 0;

}

Output:



1. **Write a program to implement a doubly-linked list.**

Source Code:

#include <iostream>

using namespace std;

class node{

    public:

    node\* prev;

    int data;

    node\* next;

    node(int value){

        prev=NULL;

        data=value;

        next=NULL;

    }

};

void insert\_head(node\* &head, int value){

    node\* n = new node(value);

    n->next=head;

    if(head!=NULL){

        head->prev=n;

    }

    head=n;

}

void insert\_tail(node\* &head, int value){

    if(head==NULL){

        insert\_head(head, value);

        return;

    }

    node\* n = new node(value);

    node\* temp=head;

    while(temp->next!=NULL){

        temp=temp->next;

    }

    temp->next=n;

    n->prev=temp;

}

void display(node\* head){

    node\* temp=head;

    while(temp!=NULL){

        cout<<temp->data<<"->";

        temp=temp->next;

    }

    cout<<"NULL"<<endl;

}

int main()

{

    node\* head=NULL;

    insert\_tail(head,3);

    insert\_tail(head,2);

    insert\_tail(head,1);

    insert\_tail(head,5);

    insert\_tail(head,4);

    cout<<"Insertion at tail: ";

    display(head);

    cout<<"Insertion at head: ";

    insert\_head(head,7);

    insert\_head(head,9);

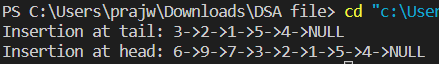
    insert\_head(head,6);

    display(head);

    return 0;

}

Output:



1. **Write a program to implement a stack using an array.**

Source Code:

#include<stdio.h>

int stack[100],c,n,top,x,i;

void push(void);

void pop(void);

void display(void);

int main()

{

    top=-1;

    scanf("%d",&n);

    printf("\n\t 1.Push\n\t 2.Pop\n\t 3.Display\n\t 4.Exit");

    do

    {

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

        scanf("%d",&c);

        switch(c)

        {

            case 1:

            {

                push();

                break;

            }

            case 2:

            {

                pop();

                break;

            }

            case 3:

            {

                display();

                break;

            }

            case 4:

            {

                break;

            }

            default:

            {

                printf ("\n\t Enter a c: ");

            }

        }

    }

    while(c!=4);

    return 0;

}

void push(){

    if(top>=n-1){

        printf("\n Overflow");

    }

    else{

        printf(" Enter a value to be pushed:");

        scanf("%d",&x);

        top++;

        stack[top]=x;

    }

}

void pop(){

    if(top<=-1){

        printf("\n Underflow");

    }

    else{

        printf("\n The popped element is %d",stack[top]);

        top--;

    }

}

void display(){

    if(top>=0){

        printf("\n Stack elements: \n");

        for(i=top; i>=0; i--)

            printf("\n%d",stack[i]);

        printf("\n Next Choice: ");

    }

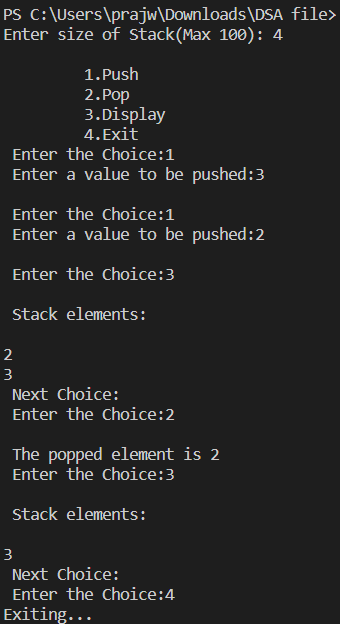
    else{

        printf("\n Stack is empty");

    }

}

Output:



1. **Write a program to implement a stack using a linked list.**

Source Code:

#include<iostream>

using namespace std;

struct node{

    int data;

    node\* link;

};

node\* top;

void push(int data){

    node\* temp = new node();

    if (!temp){

        cout << "\nStack Overflow";

        exit(1);

    }

    temp->data = data;

    temp->link = top;

    top = temp;

}

int isEmpty(){

    return top == NULL;

}

int peek(){

    if (!isEmpty())

        return top->data;

    else

        exit(1);

}

void pop(){

    node\* temp;

    if (top == NULL){

        cout << "\nStack Underflow" << endl;

        exit(1);

    }

    else{

        temp = top;

        top = top->link;

        free(temp);

    }

}

void display(){

    node\* temp;

    if (top == NULL){

        cout << "\nStack Underflow";

        exit(1);

    }

    else{

        temp = top;

        while (temp != NULL){

            cout << temp->data << "-> ";

            temp = temp->link;

        }

    }

}

int main(){

    push(15);

    push(43);

    push(78);

    push(56);

    display();

    cout << "\nTop element is "

        << peek() << endl;

    pop();

    pop();

    display();

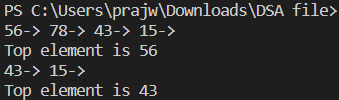
    cout << "\nTop element is "

        << peek() << endl;

    return 0;

}

Output:



1. **Write a program to implement a queue using an array.**

Source Code:

#include <stdio.h>

#include <stdlib.h>

#define MAX 50

int arr[MAX];

int front = -1;

int rear = -1;

void printQueue(){

    if(front == -1||front>rear){

        printf("Overflow\n");

    }

    else{

    for(int i = front; i<=rear; i++){

        printf("%d ",arr[i]);

    }

    printf("\n");

    }

}

void enqueue(int data){

    if (rear == MAX - 1){

        printf("Overflow\n");

    }

    if (rear == -1 && front == -1){

        rear += 1;

        front += 1;

        arr[front] = data;

    }

    else{

        rear += 1;

        arr[rear] = data;

    }

}

void dequeue(){

    if(front == -1||front>rear){

        printf("Underflow\n");

    }

    else{

      printf("%d\n",arr[front]);

      front+=1;

    }

}

int main(){

    enqueue(4);

    enqueue(7);

    enqueue(9);

    dequeue();

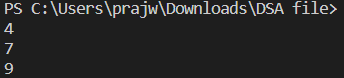
    dequeue();

    printQueue();

    return 0;

}

Output:



1. **Write a program to implement a queue using a linked list.**

Source Code:

#include <stdio.h>

#include <stdlib.h>

struct node {

    int key;

    struct node\* next;

};

struct Queue {

    struct node \*front, \*rear;

};

struct node\* newNode(int k)

{

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

    temp->key = k;

    temp->next = NULL;

    return temp;

}

struct Queue\* createQueue()

{

    struct Queue\* q = (struct Queue\*)malloc(sizeof(struct Queue));

    q->front = q->rear = NULL;

    return q;

}

void enQueue(struct Queue\* q, int k)

{

    struct node\* temp = newNode(k);

    if (q->rear == NULL) {

        q->front = q->rear = temp;

        return;

    }

    q->rear->next = temp;

    q->rear = temp;

}

void deQueue(struct Queue\* q)

{

    if (q->front == NULL)

        return;

    struct node\* temp = q->front;

    q->front = q->front->next;

    if (q->front == NULL)

        q->rear = NULL;

    free(temp);

}

int main()

{

    struct Queue\* q = createQueue();

    enQueue(q, 1);

    enQueue(q, 2);

    deQueue(q);

    enQueue(q, 3);

    enQueue(q, 4);

    enQueue(q, 5);

    deQueue(q);

    deQueue(q);

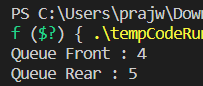
    printf("Queue Front : %d \n", q->front->key);

    printf("Queue Rear : %d", q->rear->key);

    return 0;

}

Output:



1. **Write a program to implement a circular queue using an array.**

Source Code:

#include <stdio.h>

#define MAX 6

int items[MAX];

int front = -1, rear = -1;

int isFull() {

  if ((front == rear + 1) || (front == 0 && rear == MAX - 1)) return 1;

  return 0;

}

int isEmpty() {

  if (front == -1) return 1;

  return 0;

}

void enQueue(int element) {

  if (isFull())

    printf("\n Overflow \n");

  else {

    if (front == -1) front = 0;

    rear = (rear + 1) % MAX;

    items[rear] = element;

    printf("\n Inserted  %d", element);

  }

}

int deQueue() {

  int element;

  if (isEmpty()) {

    printf("\n Underflow \n");

    return (-1);

  } else {

    element = items[front];

    if (front == rear) {

      front = -1;

      rear = -1;

    }

    else {

      front = (front + 1) % MAX;

    }

    printf("\n Deleted  %d \n", element);

    return (element);

  }

}

void display() {

  int i;

  if (isEmpty())

    printf(" \n Empty Queue\n");

  else {

    printf("\n Front: %d ", front);

    printf("\n Items: ");

    for (i = front; i != rear; i = (i + 1) % MAX) {

      printf("%d ", items[i]);

    }

    printf("%d ", items[i]);

    printf("\n Rear:  %d \n", rear);

  }

}

int main() {

  deQueue();

  enQueue(1);

  enQueue(2);

  enQueue(3);

  enQueue(4);

  enQueue(5);

  enQueue(6);

  display();

  deQueue();

  deQueue();

  display();

  enQueue(7);

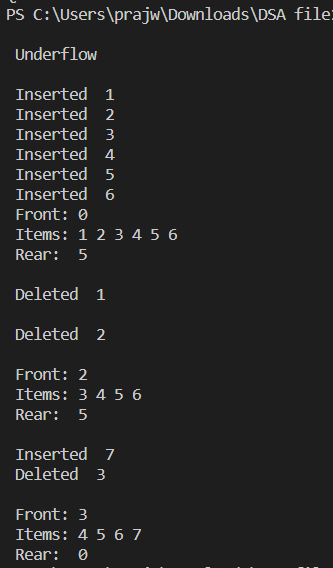
  deQueue();

  display();

  return 0;

}

Output:



1. **Write a program to implement a priority queue using a linked list.**

Source Code:

#include <stdio.h>

#include <stdlib.h>

typedef struct node{

    int item;

    int priority;

    struct node \*next;

}

node;

node \*newnode(int d, int p){

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

    temp->item = d;

    temp->priority = p;

    temp->next = NULL;

    return temp;

}

int peek(node \*\*head){

    return (\*head)->item;

}

void pop(node \*\*head){

    node \*temp = \*head;

    (\*head) = (\*head)->next;

    free(temp);

}

void push(node \*\*head, int d, int p){

    node \*start = (\*head);

    node \*temp = newnode(d, p);

    if ((\*head)->priority > p){

        temp->next = \*head;

        (\*head) = temp;

    }

    else{

        while (start->next != NULL &&

               start->next->priority < p)

        {

            start = start->next;

        }

        temp->next = start->next;

        start->next = temp;

    }

}

int isEmpty(node \*\*head){

    return (\*head) == NULL;

}

int main(){

    node \*pq = newnode(5, 2);

    push(&pq, 1, 6);

    push(&pq, 3, 3);

    push(&pq, 7, 0);

    while (!isEmpty(&pq)){

        printf("%d, ", peek(&pq));

        pop(&pq);

    }

    return 0;

}

Output:



1. **Write a program to construct a binary tree and display its preorder, inorder and postorder traversals.**

Source Code:

#include <stdio.h>

#include <stdlib.h>

struct node{

    int item;

    struct node \*left;

    struct node \*right;

};

struct node \*newNode(int item){

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

    node->item = item;

    node->left = NULL;

    node->right = NULL;

    return (node);

}

void postorder(struct node \*node){

    if (node == NULL)

        return;

    postorder(node->left);

    postorder(node->right);

    printf("%d ", node->item);

}

void inorder(struct node \*node)

{    if (node == NULL)

        return;

    inorder(node->left);

    printf("%d ", node->item);

    inorder(node->right);

}

void preorder(struct node \*node){

    if (node == NULL)

        return;

    printf("%d ", node->item);

    preorder(node->left);

    preorder(node->right);

}

int main(){

    struct node \*root = newNode(5);

    root->left = newNode(3);

    root->right = newNode(1);

    root->left->left = newNode(2);

    root->left->right = newNode(4);

    printf("\nPreorder traversal of binary tree is \n");

    preorder(root);

    printf("\nInorder traversal of binary tree is \n");

    inorder(root);

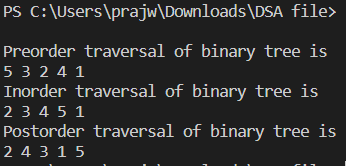
    printf("\nPostorder traversal of binary tree is \n");

    postorder(root);

    return 0;

}

Output:



1. **Write a program to construct a binary search tree**

Source Code:

#include <iostream>

using namespace std;

class BST {

    int data;

    BST \*left, \*right;

public:

    BST();

    BST(int);

    BST\* insert(BST\*, int);

    void Inorder(BST\*);

};

BST ::BST()

    : data(0)

    , left(NULL)

    , right(NULL)

{

}

BST ::BST(int value)

{

    data = value;

    left = right = NULL;

}

BST\* BST ::insert(BST\* root, int value)

{

    if (!root) {

        return new BST(value);

    }

    if (value > root->data) {

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

    }

    else {

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

    }

    return root;

}

void BST ::Inorder(BST\* root)

{

    if (!root) {

        return;

    }

    Inorder(root->left);

    cout << root->data << endl;

    Inorder(root->right);

}

int main(){

    BST b, \*root = NULL;

    root = b.insert(root, 5);

    b.insert(root, 3);

    b.insert(root, 2);

    b.insert(root, 4);

    b.insert(root, 7);

    b.insert(root, 6);

    b.insert(root, 8);

    b.Inorder(root);

    return 0;

}

Output:

