1 Write a program to implement job scheduling algorithm using queue in the following process

Process	Burst	Time Priority
P1	10	3
P2	12	1
P3	8	2

```
#include<conio.h>
#include<iostream.h>
#include<stdio.h>
int main()
{
int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
printf("Enter Total Number of Process:");
scanf("%d",&n);

printf("\nEnter Burst Time and Priority\n");
for(i=0;i<n;i++) //entering the burst time
{
printf("\nP[%d]\n",i+1);
printf("Burst Time:");
scanf("%d",&bt[i]);
printf("Priority:");

scanf("%d",&pr[i]);
p[i]=i+1; //contains process number
}

//sorting burst time, priority and process number in ascending order using selection sort
for(i=0;i<n;i++)
{
pos=i;
for(j=i+1;j<n;j++)
{
if(pr[j]<pr[pos])
pos=j;
}

temp=pr[i];
pr[i]=pr[pos];
pr[pos]=temp;

temp=bt[i];
bt[i]=bt[pos];
bt[pos]=temp;

temp=p[i];
p[i]=p[pos];
p[pos]=temp;
}
wt[0]=0; //waiting time for first process is zero

//calculate waiting time
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
total+=wt[i];
}

avg_wt=total/n; //average waiting time
total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i]; //calculate turnaround time
total+=tat[i];
printf("\nP[%d]\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
}

avg_tat=total/n; //average turnaround time
printf("\n\nAverage Waiting Time=%d",avg_wt);
printf("\nAverage Turnaround Time=%d\n",avg_tat);
getch();

}
```

2 Create a binary tree and perform postorder traversal for given data
// this prog contain pre post and inorder

```
        #include<stdio.h>
        #include<conio.h>
        #include<stdlib.h>
         struct node{
                int data;
                struct node* leftchild;
                struct node* rightchild;
         }*root=NULL;

         void insert(int data){
                struct node *tempnode=(struct node*)malloc(sizeof(struct node));
                 struct node *current;
                 struct node *parent;
                tempnode->data=data;
                tempnode->leftchild=NULL;
                tempnode->rightchild=NULL;

                if(root==NULL){
                        root=tempnode;
                 }
                 else{
                        current=root;
                        parent=NULL;

                        while(1){
```

```
                    parent=current;

        }           if(tempnode->data < parent->data){

                              current=current->leftchild;

                              if(current==NULL){
                                        parent->leftchild=tempnode;
                                        return;
                               }
                     }
                     else{
                              current=current->rightchild;
                              if(current==NULL){
                                        parent->rightchild=tempnode;
                                        return;
                               }
                     }
          }
         }
}

void pre_order_traversal(struct node* root){
         if(root!=NULL){
                   printf("%d ",root->data);
                   pre_order_traversal(root->leftchild);
                   pre_order_traversal(root->rightchild);
          }
}

void inorder_traversal(struct node* root){
         if(root!=NULL){
                   inorder_traversal(root->leftchild);
                   printf("%d ",root->data);
                   inorder_traversal(root->rightchild);
          }
}

void post_order_traversal(struct node* root){
         if(root!=NULL){
                   post_order_traversal(root->leftchild);
                   post_order_traversal(root->rightchild);
                   printf("%d ",root->data);
          }
}

struct node* search(int data){
        struct node *current=root;
        while(current->data!=data){
                if(data<current->data)
                current=current->leftchild;

                else
                current=current->rightchild;

                if(current==NULL)
                return NULL;
         }
         return current;
}


int main(){
        int i;
        int array[9]={45,15,79,90,10,55,12,20,50};
        struct node* temp;

        for(i=0;i<9;i++)
        insert(array[i]);

        printf("\nPreOrder Traversal : ");
        pre_order_traversal(root);

        printf("\nInOrder Traversal : ");
        inorder_traversal(root);

        printf("\nPostOrder Traversal : ");
        post_order_traversal(root);
        return 0;
}
```

3 create doubly Linked list, perform insert first, delete_at_pos, display operations on it.

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

struct node{
        int data;
        struct node *prev;
        struct node *next;
}*head=NULL, *end=NULL;

void create(){
        struct node *temp;
        temp = (struct node*)malloc(sizeof(struct node));
```

```
		printf("\n Enter the element to insert: ");
		scanf("%d",&temp->data);
		temp->prev = NULL;
		temp->next = NULL;
		if(head==NULL){
			head = temp;
			end = temp;
		}
		else{
			end->next = temp;
			temp->prev = end;
			end = temp;
		}
}

void insert_first(){
		struct node *temp;
		temp = (struct node*)malloc(sizeof(struct node));
		printf("\n Enter the element to insert: ");
		scanf("%d",&temp->data);
		temp->prev = NULL;
		temp->next = head;
		head->prev = temp;
		head = temp;
}

void insert_end(){
		struct node *temp;
		temp = (struct node*)malloc(sizeof(struct node));
		printf("\n Enter the element to insert: ");
		scanf("%d",&temp->data);
		temp->next = NULL;
		temp->prev = end;
		end->next = temp;
		end = temp;
}

void insert_atpos(){
		int pos,i;
		struct node *temp,*t1,*t2;
		temp = (struct node*)malloc(sizeof(struct node));
		printf("\n Enter the element to insert: ");
		scanf("%d",&temp->data);
		printf("\n Enter the position where you want to insert the data: ");
		scanf("%d",&pos);
		temp->prev = NULL;
		temp->next = NULL;
		t2 = head;
		for(i=0;i<pos-1;i++){
			t1 = t2;
			t2 = t2->next;
		}
		t1->next = temp;
		temp->prev = t1;
		temp->next = t2;
		t2->prev = temp;
}

void delete_first(){
		struct node *temp;
```

```
        temp = head;
        head = head->next;
        head->prev = NULL;
        free(temp);
}

void delete_last(){
        struct node *temp;
        temp = end;
        end = end->prev;
        end->next = NULL;
        free(temp);
}

void delete_atpos(){
        int pos,i;
        struct node *t1,*t2,*t3;
        printf("\n Enter the position where you want to delete the data: ");
        scanf("%d",&pos);
        t2 = head;
        for(i=1;i<pos;i++){
                t1 = t2;
                t2 = t2->next;
        }
        t2->next->prev = t1;
        t1->next = t2->next;
        free(t2);
}

void display(){
        struct node *temp1, *temp2;
        temp1 = head;
        temp2 = end;
        printf("NULL->");
        while(temp1!=NULL){
                printf("%d->",temp1->data);
                temp1 = temp1->next;
        }
        printf("NULL\n");
        printf("NULL->");
        while(temp2!=NULL){
                printf("%d->",temp2->data);
                temp2 = temp2->prev;
        }
        printf("NULL\n");
}

int main(){
        int n,i,choice;
   int ch;
   printf("\n Enter the initial number of nodes you want to create: ");
   scanf("%d",&n);
   for(i=0;i<n;i++){
        create();
        }
        display();
        do{
                printf("\n 1. Insert at first position");
                printf("\n 2. Insert at last position");
                printf("\n 3. Insert at your choice position");
                printf("\n 4. Delete node from first");
                printf("\n 5. Delete node from last");
                printf("\n 6. Delete node at your choice position");
                printf("\n Enter your choice: ");
                scanf("%d",&choice);
                switch(choice){
                        case 1:
                                insert_first();
                                display();
                                break;
                        case 2:
                                insert_end();
                                display();
                                break;
                        case 3:
                                insert_atpos();
                                display();
                                break;
                        case 4:
                                delete_first();
                                display();
                                break;
                        case 5:
                                delete_last();
                                display();
                                break;
                        case 6:
                                delete_atpos();
                                display();
                                break;
                        case 7:
                                exit(0);
                        default:
                                printf("\n Invalid Input!!!");
                }
                printf("\n Would you like to continue press 1:");
                scanf("%d",&ch);
        }while(ch==1);
        return 0;
}
```

4 convert a decimal number to its binary equivalent using stack

```
#include<iostream>
#include<stack>
using namespace std;
void dec_to_bin(int number) {
  stack<int> stk;
  while(number > 0) {
    int rem = number % 2; //take remainder
    number = number / 2;
    stk.push(rem);
  }
  while(!stk.empty()) {
    int item;
    item = stk.top();
    stk.pop();
    cout << item;
  }
}
main() {
  int num;
  cout << "Enter a number: ";
  cin >> num;
  dec_to_bin(num);
}
```

5 Write a program to implement the following graph using DFS traversal.

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

int total,graph[30][30],arr[30],visited[30];
static int k=0,count=0; //maintain k value even if changed
void DFS(int);

main(){
   int i,j;
   printf("\nEnter total number of vertices in graph:");
   scanf("%d",&total);

   /Adjacency matrix input/
   printf("\nEnter the adjacency matrix:\n");
   for(i=0;i<total;i++){
      for ( j = 0; j< total; j++)
      {
         scanf("%d",&graph[i][j]);
      }
   }
   for(i=0;i<total;i++){
      visited[i]=0;
   }
   //calling dfs for first vertex
   DFS(0);
}

void DFS(int vertex){
   int j,c=0;
   count++;
   printf("%d\t",vertex);
   visited[vertex]=1;
```

```
    for(j=0;j<total;j++){
        if(!visited[j] && graph[vertex][j]==1) //visit the unvisited adjacent vertices of current vertex
        {
            arr[++k]=j;
            c=1;
        }
        if(count==total){
            exit(0);
        }
    }
    if(c==1)//adjacent vetex is present
    {
        DFS(arr[k]);
    }else{
        k--;
        DFS(arr[k]);
    }
}
```

6 Represent polynomial using singly linked list

```
#include <iostream>
using namespace std;

// Node structure for singly linked list
struct Node {
    int coefficient;
    int exponent;
    Node* next;
};

// Function to create a new node with given coefficient and exponent
Node* createNode(int coefficient, int exponent) {
    Node* node = new Node;
    node->coefficient = coefficient;
    node->exponent = exponent;
    node->next = NULL;
    return node;
}

// Function to add a new term to the polynomial
void addTerm(Node** head, int coefficient, int exponent) {
    Node* node = createNode(coefficient, exponent);
    if (*head == NULL) {
        *head = node;
    }
    else {
        Node* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = node;
    }
}

// Function to display the polynomial
void display(Node* head) {
    Node* current = head;
    while (current != NULL) {
        cout << current->coefficient << "x^" << current->exponent;
        if (current->next != NULL) {
            cout << " + ";
        }
        current = current->next;
    }
    cout << endl;
}

int main() {
    Node* polynomial = NULL;
    addTerm(&polynomial, 2, 3);
    addTerm(&polynomial, 5, 2);
    addTerm(&polynomial, 1, 0);
    display(polynomial);
    return 0;
}
```

7 Representation of a polynomial using singly circular linked list

```
#include <iostream>
using namespace std;

// Node structure for circular linked list
struct Node {
    int coefficient;
    int exponent;
    Node* next;
};

// Function to create a new node with given coefficient and exponent
Node* createNode(int coefficient, int exponent) {
    Node* node = new Node;
    node->coefficient = coefficient;
    node->exponent = exponent;
    node->next = NULL;
    return node;
}

// Function to add a new term to the polynomial
void addTerm(Node** head, int coefficient, int exponent) {
    Node* node = createNode(coefficient, exponent);
    if (*head == NULL) {
        node->next = node;
        *head = node;
    }
    else {
        Node* current = *head;
        while (current->next != *head) {
            current = current->next;
        }
        current->next = node;
        node->next = *head;
    }
}

// Function to display the polynomial
void display(Node* head) {
    Node* current = head;
    do {
        cout << current->coefficient << "x^" << current->exponent;
        if (current->next != head) {
            cout << " + ";
        }
        current = current->next;
    } while (current != head);
    cout << endl;
}

int main() {
    Node* polynomial = NULL;
    addTerm(&polynomial, 2, 3);
    addTerm(&polynomial, 5, 2);
    addTerm(&polynomial, 1, 0);
    display(polynomial);
    return 0;
}
```

8 write a program to make singly linked list to perform create insert at position and display function

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

```
#include<conio.h>
// Linked List Node
struct node
{
        int info;
        struct node* link;
}*head=NULL,*end;

// Function to create list with n nodes initially//code by Anuja
void createList()
{
        if (head == NULL)
        {
                int n;
                printf("\nEnter the number of nodes: ");
                scanf("%d", &n);
                if (n >0)
                 {
                        int info;
                        struct node* newnode;
                        struct node* temp;

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

                        head= newnode;
                        temp = head;
                        printf("\nEnter number to be inserted : ");
                        scanf("%d", &head -> info);


                        for (int i = 2; i <= n; i++)
                        {
                                newnode = (struct node*)malloc(sizeof(struct node));
                                printf("\nEnter number to be inserted : ");
                                scanf("%d", &newnode->info);
                        newnode->link=NULL;
                        temp->link=newnode;
                                temp = temp->link;
                        }
                }
                printf("\nThe list is created\n");
        } else
                printf("\nThe list is already created\n");
}
void reverseL()
{
    struct node *t1, *t2, *temp;
    t1 = t2 = NULL;
    if (head == NULL)  //If list is empty
        printf("List is empty\n");

    else {
        while (head!= NULL)
                {
            t2 = head->link;
            head->link = t1;
            t1 = head;
            head= t2;
        }
        head= t1;
        temp = head;

        printf("Reverse linked list is : ");
                    while (temp != NULL) {
            temp = temp->link;
        }
    }
}
void search()
{
 struct node *ptr;
 int item,i=0,flag=1;
 ptr = head;
 if(ptr == NULL)
 {
 printf("\nEmpty List\n");
 }
 else
 {
 printf("\nEnter item which you want to search?\n");   scanf("%d",&item);
 if(head ->info== item)
 {
 printf("item found at location %d",i+1);   flag=0;
 }
 else
 {
 while (ptr->link != head)
 {
 if(ptr->info== item)
 {
 printf("item found at location %d ",i+1);
 flag=0;
 break;
 }
 else
 {  //printf("item is not in the list");
 flag=1;
 }
 i++;
 ptr = ptr -> link;
 }
 }
 if(flag != 0)
 {
 printf("Item not found\n");
 }
 }

}


void count(){
        struct node *temp=head;
        int count=0,num;
        temp = head;
        printf("\n Enter the element whos count you want to know in Linked lsit:");
        scanf("%d",&num);
        while(temp!=NULL){
            if(temp->info==num){
                count++;
```

```
                        }
                        temp = temp->link;
                    }
                    if(count==0){
                        printf("\n %d is not present in the Linked List",num);
                    }
                    else{
                        printf("\n %d is present %d times in the linked list",num);
                    }

            }

            void display()
                    {
                    struct node *disp;
                    disp=head;
                            while(disp!=NULL)
                            {
                                    printf("%d->",disp->info);
                                    disp=disp->link;
                            }
                            printf("NULL");
              }
            // Driver Code
                    int main()
                     {
                            int choice,pos,i;
                            while (choice !=5)
                            {
                                    printf("\n\n1.Create list \n2.Reverse \n3.Search \n4.Count \n5.Display \n6.Exit");
                                    printf("\nEnter your choice: ");
                                    scanf("%d",&choice);
                                    switch(choice)
                                    {
                                    case 1: createList();
                                                    display();
                                                    break;
                                    case 2:
                                    reverseL();
                                              display();
                                                     break;
                                    case 3:
                                                    search();
                                                    display();

        break;//##################################################################

                                    case 4:
                                                    printf("\nLinked list is: ");
                                            display();

                                    case 5: exit(0);
                                            default:printf("\nYour choice");

                                            }
                                            }
                                            return 0;
                     }
```

9 Convert following matrix into sparse matrix

```
// C++ program to convert a Matrix
// into Sparse Matrix

#include <iostream>
using namespace std;

// Maximum number of elements in matrix
#define MAX 100

// Array representation
// of sparse matrix
//[][0] represents row
//[][1] represents col
//[][2] represents value
int data[MAX][3];

// total number of elements in matrix
int len;

// insert elements into sparse matrix
void insert(int r, int c, int val)
{
	// insert row value
	data[len][0] = r;

	// insert col value
	data[len][1] = c;

	// insert element's value
	data[len][2] = val;

	// increment number of data in matrix
	len++;
}

// printing Sparse Matrix
void print()
{
	cout << "\nDimension of Sparse Matrix: "
		<< len << " x " << 3;
	cout << "\nSparse Matrix: \nRow Column Value\n";

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

		cout << data[i][0] << " "
			<< data[i][1] << " "
			<< data[i][2] << "\n";
	}
}

// Driver code
int main()
{
	int i, j;
	int r = 5, c = 4;

	// Get the matrix
	int a[r] = { { 0, 1, 0, 0 },
						{ 0, 0, 2, 0 },
						{ 0, 3, 0, 0 },
						{ 0, 0, 5, 0 },
						{ 0, 0, 0, 4 } };

	// print the matrix
	cout << "\nMatrix:\n";
	for (i = 0; i < r; i++) {
		for (j = 0; j < c; j++) {
			cout << a[i][j] << " ";
		}
		cout << endl;
	}

	// iterate through the matrix and
	// insert every non zero elements
	// in the Sparse Matrix
	for (i = 0; i < r; i++)
		for (j = 0; j < c; j++)
			if (a[i][j] > 0)
				insert(i, j, a[i][j]);

	// Print the Sparse Matrix
	print();

	return 0;
}
```

10 Write a program to create a stack of following characters using a linked list and perform Push()and Pop() operations on it. G,F,E,D,C,B,A

```
#include <iostream>
using namespace std;

struct Node {
    char data;
    Node* next;
};

class Stack {
private:
    Node* top;
public:
    Stack() {
        top = nullptr;
    }

    void push(char x) {
        Node* temp = new Node;
        temp->data = x;
        temp->next = top;
        top = temp;
    }

    char pop() {
        char x = '\0';
        if (top == nullptr) {
            cout << "Stack is empty";
            return x;
        }
        Node* temp = top;
        top = top->next;
        x = temp->data;
        delete temp;
        return x;
    }
};

int main() {
    Stack s;
    s.push('A');
    s.push('B');
    s.push('C');
    s.push('D');
    s.push('E');
    s.push('F');
    s.push('G');
    cout << s.pop() << endl; // output: G
    cout << s.pop() << endl; // output: F
    cout << s.pop() << endl; // output: E
    cout << s.pop() << endl; // output: D
    cout << s.pop() << endl; // output: C
    cout << s.pop() << endl; // output: B
    cout << s.pop() << endl; // output: A
    cout << s.pop() << endl; // output: Stack is empty
    return 0;
}
```

11 Circular queue using linked list

```
// C++ program for insertion and
// deletion in Circular Queue
#include <bits/stdc++.h>
using namespace std;

// Structure of a Node
struct Node {
	int data;
	struct Node* link;
};

struct Queue {
	struct Node *front, *rear;
};

// Function to create Circular queue
void enQueue(Queue* q, int value)
{
	struct Node* temp = new Node;
	temp->data = value;
	if (q->front == NULL)
		q->front = temp;
	else
		q->rear->link = temp;

	q->rear = temp;
	q->rear->link = q->front;
}

// Function to delete element from Circular Queue
int deQueue(Queue* q)
{
	if (q->front == NULL) {
		cout << "Queue is empty";
		return INT_MIN;
	}

	// If this is the last node to be deleted
	int value; // Value to be dequeued
```

```
        if (q->front == q->rear) {
                value = q->front->data;
                free(q->front);
                q->front = NULL;
                q->rear = NULL;
        }
        else // There are more than one nodes
        {
                struct Node* temp = q->front;
                value = temp->data;
                q->front = q->front->link;
                q->rear->link = q->front;
                free(temp);
        }

        return value;
}

// Function displaying the elements of Circular Queue
void displayQueue(struct Queue* q)
{
        struct Node* temp = q->front;
        cout << endl << "Elements in Circular Queue are: ";
        while (temp->link != q->front) {
                cout << temp->data << " ";
                temp = temp->link;
        }
        cout << temp->data;
}

/* Driver of the program */
int main()
{
        // Create a queue and initialize front and rear
        Queue* q = new Queue;
        q->front = q->rear = NULL;

        // Inserting elements in Circular Queue
        enQueue(q, 14);
        enQueue(q, 22);
        enQueue(q, 6);

        // Display elements present in Circular Queue
        displayQueue(q);

        // Deleting elements from Circular Queue
        cout << endl << "Deleted value = " << deQueue(q);
        cout << endl << "Deleted value = " << deQueue(q);
        // Remaining elements in Circular Queue
        displayQueue(q);

        enQueue(q, 9);
        enQueue(q, 20);
        displayQueue(q);

        return 0;
}
```

12 write a program to create a doubly linked list of no perform insert at last,delete first and display.

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

struct node{
        int data;
        struct node *prev;
        struct node *next;
}*head=NULL, *end=NULL;

void create(){
        struct node *temp;
        temp = (struct node*)malloc(sizeof(struct node));
        printf("\n Enter the element to insert: ");
        scanf("%d",&temp->data);
        temp->prev = NULL;
        temp->next = NULL;
        if(head==NULL){
                head = temp;
                end = temp;
        }
        else{
                end->next = temp;
                temp->prev = end;
                end = temp;
        }
}

void insert_first(){
        struct node *temp;
        temp = (struct node*)malloc(sizeof(struct node));
        printf("\n Enter the element to insert: ");
        scanf("%d",&temp->data);
        temp->prev = NULL;
        temp->next = head;
        head->prev = temp;
        head = temp;
}

void insert_end(){
        struct node *temp;
        temp = (struct node*)malloc(sizeof(struct node));
        printf("\n Enter the element to insert: ");
```

```
		scanf("%d",&temp->data);
		temp->next = NULL;
		temp->prev = end;
		end->next = temp;
		end = temp;
}

void insert_atpos(){
		int pos,i;
		struct node *temp,*t1,*t2;
		temp = (struct node*)malloc(sizeof(struct node));
		printf("\n Enter the element to insert: ");
		scanf("%d",&temp->data);
		printf("\n Enter the position where you want to insert the data: ");
		scanf("%d",&pos);
		temp->prev = NULL;
		temp->next = NULL;
		t2 = head;
		for(i=0;i<pos-1;i++){
				t1 = t2;
				t2 = t2->next;
		}
		t1->next = temp;
		temp->prev = t1;
		temp->next = t2;
		t2->prev = temp;
}

void delete_first(){
		struct node *temp;
		temp = head;
		head = head->next;
		head->prev = NULL;
		free(temp);
}

void delete_last(){
		struct node *temp;
		temp = end;
		end = end->prev;
		end->next = NULL;
		free(temp);
}

void delete_atpos(){
		int pos,i;
		struct node *t1,*t2,*t3;
		printf("\n Enter the position where you want to delete the data: ");
		scanf("%d",&pos);
		t2 = head;
		for(i=1;i<pos;i++){
				t1 = t2;
				t2 = t2->next;
		}
		t2->next->prev = t1;
		t1->next = t2->next;
		free(t2);
}

void display(){
```

```
            struct node *temp1, *temp2;
            temp1 = head;
            temp2 = end;
            printf("NULL->");
            while(temp1!=NULL){
                        printf("%d->",temp1->data);
                        temp1 = temp1->next;
            }
            printf("NULL\n");
            printf("NULL->");
            while(temp2!=NULL){
                        printf("%d->",temp2->data);
                        temp2 = temp2->prev;
            }
            printf("NULL\n");
}

int main(){
            int n,i,choice;
   int ch;
   printf("\n Enter the initial number of nodes you want to create: ");
   scanf("%d",&n);
   for(i=0;i<n;i++){
            create();
            }
            display();
            do{
                        printf("\n 1. Insert at first position");
                        printf("\n 2. Insert at last position");
                        printf("\n 3. Insert at your choice position");
                        printf("\n 4. Delete node from first");
                        printf("\n 5. Delete node from last");
                        printf("\n 6. Delete node at your choice position");
                        printf("\n Enter your choice: ");
                        scanf("%d",&choice);
                        switch(choice){
                                    case 1:
                                                insert_first();
                                                display();
                                                break;
                                    case 2:
                                                insert_end();
                                                display();
                                                break;
                                    case 3:
                                                insert_atpos();
                                                display();
                                                break;
                                    case 4:
                                                delete_first();
                                                display();
                                                break;
                                    case 5:
                                                delete_last();
                                                display();
                                                break;
                                    case 6:
                                                delete_atpos();
                                                display();
                                                break;
```

```
                                        case 7:
                                                exit(0);
                                        default:
                                                printf("\n Invalid Input!!!");
                                }
                                printf("\n Would you like to continue press 1:");
                                scanf("%d",&ch);
                        }while(ch==1);
                        return 0;
                }
```

13 Implement Quick sort 38,10,9,2,3,2,27,40

```
/* C++ implementation of QuickSort */
#include <bits/stdc++.h>
using namespace std;

/* This function takes last element as pivot, places
the pivot element at its correct position in sorted
array, and places all smaller (smaller than pivot)
to left of pivot and all greater elements to right
of pivot */
int partition(int arr[], int low, int high)
{
        int pivot = arr[high]; // pivot
        int i
                = (low
                - 1); // Index of smaller element and indicates
                                // the right position of pivot found so far

        for (int j = low; j <= high - 1; j++) {
                // If current element is smaller than the pivot
                if (arr[j] < pivot) {
                        i++; // increment index of smaller element
                        swap(arr[i], arr[j]);
                }
        }
        swap(arr[i + 1], arr[high]);
        return (i + 1);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
        if (low < high) {
                /* pi is partitioning index, arr[p] is now
                at right place */
                int pi = partition(arr, low, high);
                // Separately sort elements before
                // partition and after partition
                quickSort(arr, low, pi - 1);
                quickSort(arr, pi + 1, high);
        }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
        int i;
        for (i = 0; i < size; i++)
                cout << arr[i] << " ";
        cout << endl;
}
int main()
{
        int arr[] = { 10, 7, 8, 9, 1, 5 };
        int n = sizeof(arr) / sizeof(arr[0]);
        quickSort(arr, 0, n - 1);
        cout << "Sorted array: \n";
        printArray(arr, n);
        return 0;
}
```

14 WAP to print the postorder of following tree
Go check 2 que
15 Write a program to check whether following Matrix is a sparse matrix

```
#include <stdio.h>

int main()
{
    int rows, cols, size, count = 0;

    //Initialize matrix a
    int a[][3] = {
                    {4, 0, 0},
                    {0, 5, 0},
                    {0, 0, 6}
                };

    //Calculates number of rows and columns present in given matrix
    rows = (sizeof(a)/sizeof(a[0]));
    cols = (sizeof(a)/sizeof(a[0][0]))/rows;

    //Calculates the size of array
    size = rows * cols;

    //Count all zero element present in matrix
    for(int i = 0; i < rows; i++){
        for(int j = 0; j < cols; j++){
          if(a[i][j] == 0)
            count++;
        }
    }

    if(count > (size/2))
        printf("Given matrix is a sparse matrix");
    else
        printf("Given matrix is not a sparse matrix");

    return 0;
}
```

16 Write a program to sort the following series using the merge sort method. 38,12,25,18,45,32

```
#include <iostream>

using namespace std;

/* Function to merge the subarrays of a[] */
void merge(int a[], int beg, int mid, int end)
{
    int i, j, k;
    int n1 = mid - beg + 1;
    int n2 = end - mid;

    int LeftArray[n1], RightArray[n2]; //temporary arrays

    /* copy data to temp arrays */
    for (int i = 0; i < n1; i++)
    LeftArray[i] = a[beg + i];
    for (int j = 0; j < n2; j++)
    RightArray[j] = a[mid + 1 + j];

    i = 0; /* initial index of first sub-array */
    j = 0; /* initial index of second sub-array */
    k = beg;  /* initial index of merged sub-array */

    while (i < n1 && j < n2)
    {
        if(LeftArray[i] <= RightArray[j])
        {
            a[k] = LeftArray[i];
            i++;
        }
        else
        {
            a[k] = RightArray[j];
            j++;
        }
        k++;
    }
    while (i<n1)
    {
        a[k] = LeftArray[i];
        i++;
        k++;
    }
```

```

    while (j<n2)
    {
        a[k] = RightArray[j];
        j++;
        k++;
    }
}

void mergeSort(int a[], int beg, int end)
{
    if (beg < end)
    {
        int mid = (beg + end) / 2;
        mergeSort(a, beg, mid);
        mergeSort(a, mid + 1, end);
        merge(a, beg, mid, end);
    }
}

/* Function to print the array */
void printArray(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        cout<<a[i]<<" ";
}

int main()
{
    int a[] = { 11, 30, 24, 7, 31, 16, 39, 41 };
    int n = sizeof(a) / sizeof(a[0]);
    cout<<"Before sorting array elements are - \n";
    printArray(a, n);
```

```
mergeSort(a, 0, n - 1);
cout<<"\nAfter sorting array elements are - \n";
printArray(a, n);
return 0;
}
```

17 Implementation of Fibonacci search

```
#include <bits/stdc++.h>
using namespace std;

int min(int x, int y) { return (x <= y) ? x : y; }
int fibMonaccianSearch(int arr[], int x, int n)
{
        int fibMMm2 = 0; // (m-2)'th Fibonacci No.
        int fibMMm1 = 1; // (m-1)'th Fibonacci No.
        int fibM = fibMMm2 + fibMMm1; // m'th Fibonacci

        while (fibM < n) {
                fibMMm2 = fibMMm1;
                fibMMm1 = fibM;
                fibM = fibMMm2 + fibMMm1;
        }
        int offset = -1;
        while (fibM > 1) {

                int i = min(offset + fibMMm2, n - 1);

                if (arr[i] < x) {
                        fibM = fibMMm1;
                        fibMMm1 = fibMMm2;
                        fibMMm2 = fibM - fibMMm1;
                        offset = i;
                }


                else if (arr[i] > x) {
                        fibM = fibMMm2;
                        fibMMm1 = fibMMm1 - fibMMm2;
                        fibMMm2 = fibM - fibMMm1;
                }


                else
                        return i;
        }


        if (fibMMm1 && arr[offset + 1] == x)
                return offset + 1;
        return -1;
}

int main()
{
        int arr[]
                = { 11,45,25,46,36,25};
        int n = sizeof(arr) / sizeof(arr[0]);
        int x = 36;
        int ind = fibMonaccianSearch(arr, x, n);
if(ind>=0)
        cout << "Found at index: " << ind;
else
        cout << x << " isn't present in the array";

        return 0;
}
```

18 create queue using linked list

```
#include <iostream>
using namespace std;
struct node {
  int data;
  struct node *next;
};
struct node* front = NULL;
struct node* rear = NULL;
struct node* temp;
void Insert() {
  int val;
  cout<<"Insert the element in queue : "<<endl;
  cin>>val;
  if (rear == NULL) {
    rear = (struct node *)malloc(sizeof(struct node));
    rear->next = NULL;
    rear->data = val;
    front = rear;
  } else {
    temp=(struct node *)malloc(sizeof(struct node));
    rear->next = temp;
    temp->data = val;
    temp->next = NULL;
    rear = temp;
  }
}
void Delete() {
  temp = front;
  if (front == NULL) {
    cout<<"Underflow"<<endl;
    return;
  }
  else
  if (temp->next != NULL) {
    temp = temp->next;
    cout<<"Element deleted from queue is : "<<front->data<<endl;
    free(front);
    front = temp;
  } else {
    cout<<"Element deleted from queue is : "<<front->data<<endl;
    free(front);
    front = NULL;
    rear = NULL;
  }
}
void Display() {
  temp = front;
  if ((front == NULL) && (rear == NULL)) {
    cout<<"Queue is empty"<<endl;
    return;
  }
  cout<<"Queue elements are: ";
  while (temp != NULL) {
    cout<<temp->data<<" ";
    temp = temp->next;
  }
  cout<<endl;
}
int main() {
  int ch;
  cout<<"1) Insert element to queue"<<endl;
  cout<<"2) Delete element from queue"<<endl;
  cout<<"3) Display all the elements of queue"<<endl;
  cout<<"4) Exit"<<endl;
  do {
    cout<<"Enter your choice : "<<endl;
    cin>>ch;
    switch (ch) {
       case 1: Insert();
       break;
       case 2: Delete();
       break;
       case 3: Display();
       break;
       case 4: cout<<"Exit"<<endl;
       break;
       default: cout<<"Invalid choice"<<endl;
    }
  } while(ch!=4);
```

```
    return 0;
}
```