**Practical File**

***Lab Name……………………………DS LAB….………………….………. Lab Code………KCS-351…………………***

.



**Name…………………………SAMARPIT DUA……………..…………………………………………………..**

**Adm.No…………2019B111068…………………….. Univ. Roll No………1900321290050….…...…**

**Course ………B.TECH……………………..……Branch……CEIT ………………..……….…………..**

**Sem…………………..……3……….…… Section……………A……..………….…..……..**



**ABES Engineering College, Ghaziabad**

**Department of Computer Science**

**B.Tech (CS) III Semester**

**Session 2020-21 (Odd Semester)**

**Course Title: DS using C**

**Name of Faculty : Sugandha Satija**

**List of Experiments**

1. Implementation of basic operations on Arrays.
2. Implementation of basic operations on Linked List.
3. Implementation of basic operations on Stacks using arrays and linked list.
4. Implementation of basic operations on Queues using arrays and linked list.
5. Implementation of circular Queue.
6. Implementation of Searching techniques.
7. Implementation of Recursive Sorting techniques
8. Implementation of Non-Recursive sorting techniques
9. Implementation of Binary trees.
10. Implementation of Graphs.

**(Sign of faculty) Dr. (Prof.) Pankaj Sharma**

**HOD(CS)**

**ABES Engineering College, Ghaziabad**

**Department of Computer Science**

**List of Experiments**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. **S.No.** | **Experiment Name** | **Date Held** | **Submission Date** | **Grade** |
| 1 | |  | | --- | | Array | | (a)   Program for Matrix Addition | | (b)   Program for Matrix Multiplication | | (c)   Program for Matrix transposition | |  |  |  |
| 2 | |  | | --- | | Linked List | | (a)   Creation | | (b)   Insertion | | (c)   Deletion | | (d)   Searching | | (e)   Sorting | |  |  |  |
| 3 | Implementation of Stack Operation using Array and Linked List. |  |  |  |
| 4 | Implementation of Queue using Array and Linked List. |  |  |  |
| 5 | Implementation of Circular Queue using Array.  Implementation of Circular Queue using Linked List. |  |  |  |
| 6 | |  |  |  |  | | --- | --- | --- | --- | | |  | | --- | | Searching Algorithm. | | (a)   Linear Search | | (b)   Binary Search | | |  |  |  |
| 7 | |  | | --- | | Sorting Algorithms-Recursive. | | (a)   Quick Sort | | (b)   Merge Sort | | (c)   Heap Sort | |  |  |  |
| 8 | |  | | --- | | Sorting Algorithms-Non-Recursive. | | (a)  Bubble | | (b)  Selection | | (c)   Insertion Sort | | (d)   Counting Sort | | (e)  Radix Sort | |  |  |  |
| 9 | Implementation of Tree Structures, Binary Tree, Tree Traversal, Binary Search Tree, Insertion and Deletion in BST. |  |  |  |
| 10 | Graph Implementation, BFS, DFS, Minimum cost spanning tree, shortest path algorithm. |  |  |  |

**(Sign of Faculty) Dr. (Prof.) Pankaj Sharma**

**HOD(CS)**

**PRACTICAL -1(A)**

1. **PRACTICAL STATEMENT OF PRACTICAL:Write a program to perform addition of two matrices.**
2. **OBJECTIVE OF PRACTICAL : Addition of two matrices.**
3. **IMPLEMENTATION :**

#include <stdio.h>

int main()

{

    printf("SAMARPIT DUA 1900321290050 \n ADDITION OF TWO MATRICES");

    int r, c, a[100][100], b[100][100], sum[100][100], i, j;

    printf("\nEnter the number of rows : ");

    scanf("%d", &r);

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

    scanf("%d", &c);

    printf("\nEnter elements of 1st matrix:\n");

    for (i = 0; i < r; ++i)

        for (j = 0; j < c; ++j) {

            printf("Enter element a%d%d: ", i + 1, j + 1);

            scanf("%d", &a[i][j]);

        }

    printf("Enter elements of 2nd matrix:\n");

    for (i = 0; i < r; ++i)

        for (j = 0; j < c; ++j) {

            printf("Enter element a%d%d: ", i + 1, j + 1);

            scanf("%d", &b[i][j]);

        }

    for (i = 0; i < r; ++i)

        for (j = 0; j < c; ++j) {

            sum[i][j] = a[i][j] + b[i][j];

        }

    // printing the result

    printf("\nSum of two matrices: \n");

    for (i = 0; i < r; ++i)

        for (j = 0; j < c; ++j) {

            printf("%d   ", sum[i][j]);

            if (j == c - 1) {

                printf("\n\n");

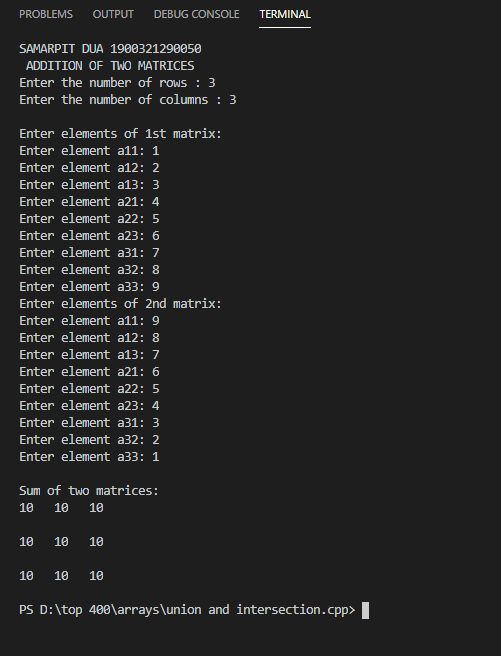
            }

        }

    return 0;

}

**Result /Output :**

****

**PRACTICAL -1(B)**

**PRACTICAL STATEMENT OF PRACTICAL:Write a program to implement multiplication of two matrices.**

**OBJECTIVE OF PRACTICAL : Implementing multiplication of two matrices.**

**IMPLEMENTATION :**

**#include <stdio.h>**

**int main()**

**{**

**printf("SAMARPIT DUA 1900321290050 \n ");**

**int m, n, p, q, c, d, k, sum = 0;**

**int first[10][10], second[10][10], multiply[10][10];**

**printf("Enter the number of rows and columns of first matrix\n");**

**scanf("%d%d", &m, &n);**

**printf("Enter the elements of first matrix\n");**

**for (c = 0; c < m; c++)**

**for (d = 0; d < n; d++)**

**scanf("%d", &first[c][d]);**

**printf("Enter the number of rows and columns of second matrix\n");**

**scanf("%d%d", &p, &q);**

**if (n != p)**

**printf("Matrices with entered orders can't be multiplied with each other.\n");**

**else**

**{**

**printf("Enter the elements of second matrix\n");**

**for (c = 0; c < p; c++)**

**for (d = 0; d < q; d++)**

**scanf("%d", &second[c][d]);**

**for (c = 0; c < m; c++)**

**{**

**for (d = 0; d < q; d++)**

**{**

**for (k = 0; k < p; k++)**

**{**

**sum = sum + first[c][k] \* second[k][d];**

**}**

**multiply[c][d] = sum;**

**sum = 0;**

**}**

**}**

**printf("Product of entered matrices:-\n");**

**for (c = 0; c < m; c++)**

**{**

**for (d = 0; d < q; d++)**

**printf("%d\t", multiply[c][d]);**

**printf("\n");**

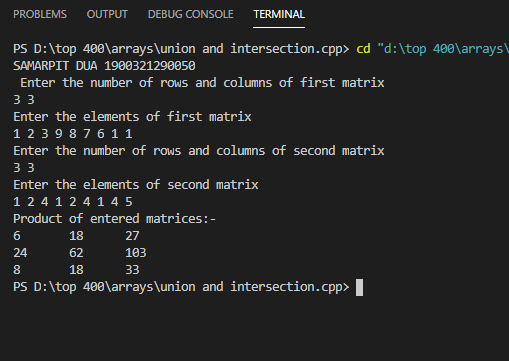
**}**

**}**

**return 0;**

**}**

**Result /Output :**

****

**PRACTICAL -1(C)**

**PRACTICAL STATEMENT OF PRACTICAL: Write a program to implement transpose of a matrix.**

**OBJECTIVE OF PRACTICAL :Calculate Transpose of matrix.**

**IMPLEMENTATION :**

#include <stdio.h>

int main()

{

    printf("\nSAMARPIT DUA 1900321290050\n");

    int a[10][10], transpose[10][10], r, c, i, j;

    printf("Enter rows and columns: ");

    scanf("%d %d", &r, &c);

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

    for (i = 0; i < r; ++i)

        for (j = 0; j < c; ++j)

        {

            printf("Enter element a%d%d: ", i + 1, j + 1);

            scanf("%d", &a[i][j]);

        }

    printf("\nEntered matrix: \n");

    for (i = 0; i < r; ++i)

    {

        for (j = 0; j < c; ++j)

        {

            printf("%d ", a[i][j]);

        }

        printf("\n");

    }

    for (i = 0; i < r; ++i)

        for (j = 0; j < c; ++j)

        {

            transpose[j][i] = a[i][j];

        }

    printf("\nTranspose of the matrix:\n");

    for (i = 0; i < c; ++i)

    {

        for (j = 0; j < r; ++j)

        {

            printf("%d ", transpose[i][j]);

        }

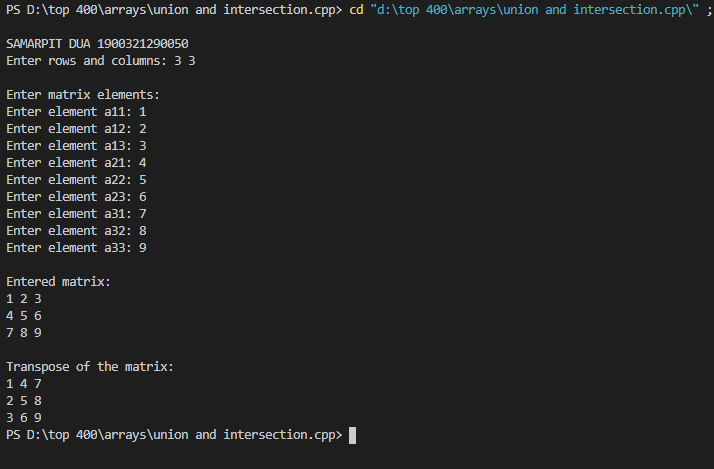
        printf("\n");

    }

    return 0;

}

**Result /Output :**

****

**PRACTICAL-2**

**PRACTICAL STATEMENT OF PRACTICAL: Create a linked list and perform different operations like insertion,deletion etc. on it.**

**OBJECTIVE OF PRACTICAL : Implementation of Linked list.**

**IMPLEMENTATION :**

**#include<iostream>**

**#include<stdlib.h>**

**struct node**

**{**

**int data;**

**struct node \*next;**

**};**

**struct node \*temp,\*ptr;**

**struct node \*start=NULL;**

**//Display Function**

**void display()**

**{**

**ptr=start;**

**while(ptr!=NULL)**

**{**

**printf("\n%d",ptr->data);**

**ptr=ptr->next;**

**}**

**}**

**//Creation of Singly linked list**

**void create()**

**{**

**int ch;**

**printf("\nPress 2 to create : ");**

**scanf("%d",&ch);**

**while(ch!=1)**

**{**

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

**printf("\nEnter the data-");**

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

**temp->next=NULL;**

**if(start==NULL)**

**{**

**start=temp;**

**ptr=temp;**

**}**

**else**

**{**

**ptr->next=temp;**

**ptr=temp;**

**}**

**printf("\nPress 2 to keep adding nodes and 1 to exit-");**

**scanf("%d",&ch);**

**}**

**}**

**//Insert at beginning of the list**

**void insert\_at\_beg()**

**{**

**if(start==NULL)**

**{**

**printf("Create list first ");**

**return;**

**}**

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

**printf("\nEnter the Data:-");**

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

**temp->next=start;**

**start=temp;**

**}**

**//Insert At the end**

**void insert\_at\_end()**

**{**

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

**printf("\nEnter the Data:-");**

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

**temp->next=NULL;**

**ptr=start;**

**while(ptr->next!=NULL)**

**{**

**ptr=ptr->next;**

**}**

**ptr->next=temp;**

**}**

**//Insert at any position**

**void insert\_at\_pos()**

**{**

**struct node \*q;**

**int pos;int c=1;**

**printf("\nEnter the position at which you want to insert a node: ");**

**scanf("%d",&pos);**

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

**printf("\nEnter the Data:-");**

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

**ptr=start;**

**while(ptr->next!=NULL&&c!=pos)**

**{**

**q=ptr;**

**c++;**

**ptr=ptr->next;**

**}**

**if(c==pos)**

**{**

**temp->next=ptr;**

**q->next=temp;**

**}**

**else**

**{**

**printf("\nYour entered position is not defined");**

**}**

**}**

**//Deletion in Singly linked list at beginning**

**void delete\_at\_beg()**

**{**

**if(start==NULL)**

**{**

**printf("\nUnderflow");**

**return;**

**}**

**ptr=start;**

**start=ptr->next;**

**free(ptr);**

**}**

**//Deletion at end of linked list**

**void delete\_at\_end()**

**{**

**if(start==NULL)**

**{**

**printf("\nUnderflow");**

**return;**

**}**

**if(start->next==NULL)**

**{**

**ptr=start;**

**start=NULL;**

**free(ptr);**

**}**

**else**

**{**

**ptr=start;**

**while(ptr->next!=NULL)**

**{**

**temp=ptr;**

**ptr=ptr->next;**

**}**

**temp->next=NULL;**

**free(ptr);**

**}**

**}**

**//Delete element from any position**

**void delete\_at\_pos()**

**{**

**int pos;int c=1;**

**printf("\nEnter the position of node you want to delete: ");**

**scanf("%d",&pos);**

**ptr=start;**

**while(ptr->next!=NULL&&c!=pos)**

**{**

**temp=ptr;**

**c++;**

**ptr=ptr->next;**

**}**

**if(c==pos)**

**{**

**temp->next=ptr->next;**

**free(ptr);**

**}**

**else**

**{**

**printf("\nYour entered position is not defined");**

**}**

**}**

**int main()**

**{**

**printf("SAMARPIT DUA (1900321290050) IMPLEMENTATION OF LINKED LIST");**

**int ch,c;**

**do**

**{**

**printf("\n\tMENU DRIVEN\n\t1.Creation of linked list\n\t2.Display of linkedlist\n\t");**

**printf("3.Insertion at beginning\n\t4.Insertion at the end\n\t5.Insertion at any position\n\t");**

**printf("6.Deletion at beginning\n\t7.Deletion at the end\n\t8.Deletion at any position\n\t9Exit");**

**printf("\n\tEnter Your Choice:-");**

**scanf("%d",&c);**

**switch(c)**

**{**

**case 1: create();**

**break;**

**case 2: display();**

**break;**

**case 3: insert\_at\_beg();**

**break;**

**case 4: insert\_at\_end();**

**break;**

**case 5: insert\_at\_pos();**

**break;**

**case 6: delete\_at\_beg();**

**break;**

**case 7: delete\_at\_end();**

**break;**

**case 8: delete\_at\_pos();**

**break;**

**case 9: exit(0);**

**default: printf("Wrong Choice!!");**

**}**

**printf("Do you want to continue ? ");**

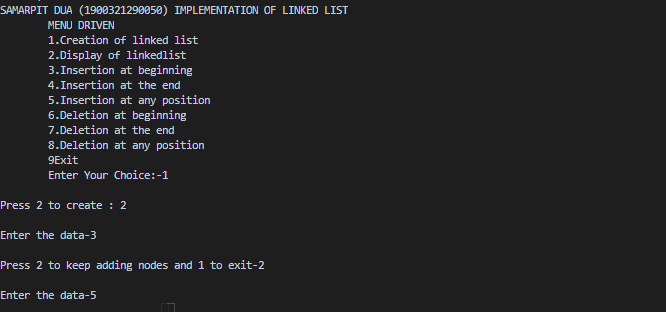
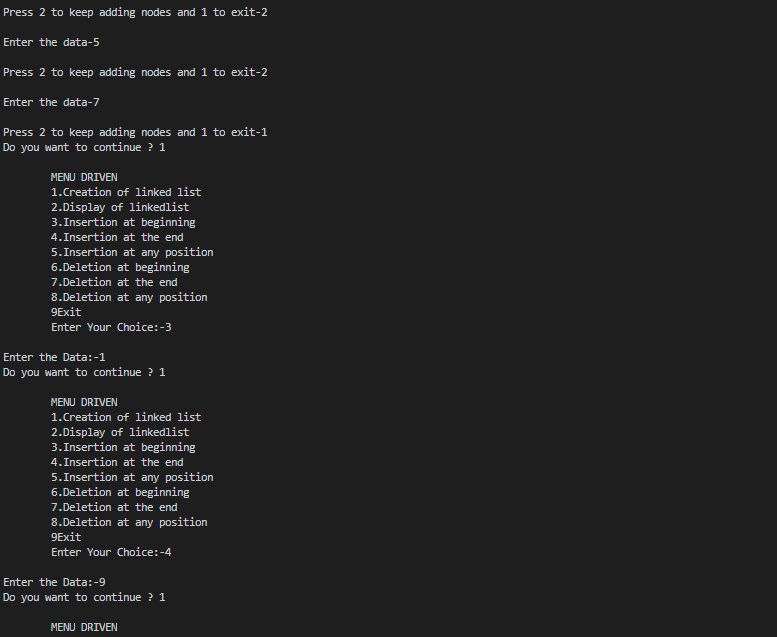
**scanf("%d",&ch);**

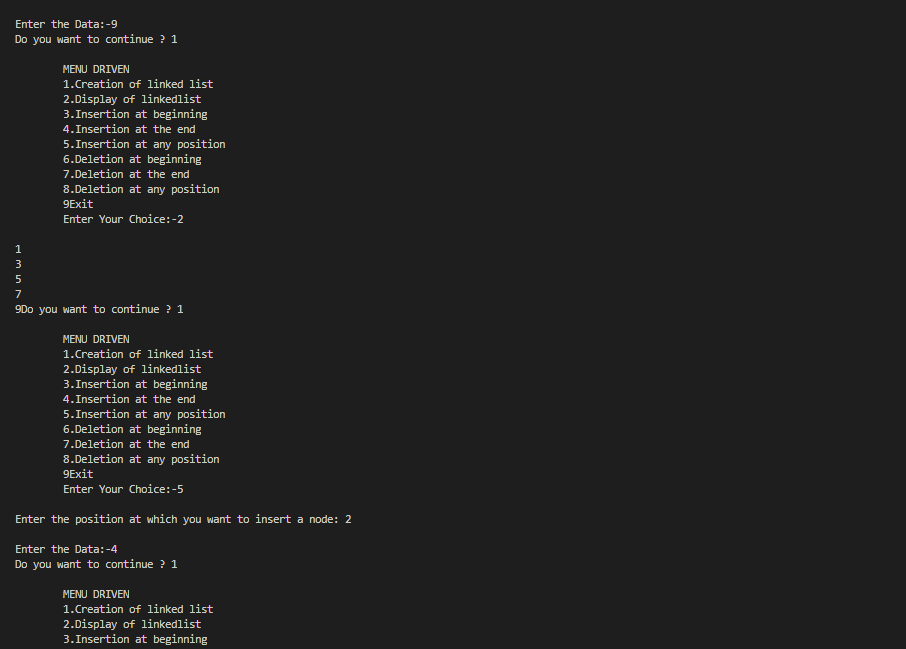
**}while(ch!=0);**

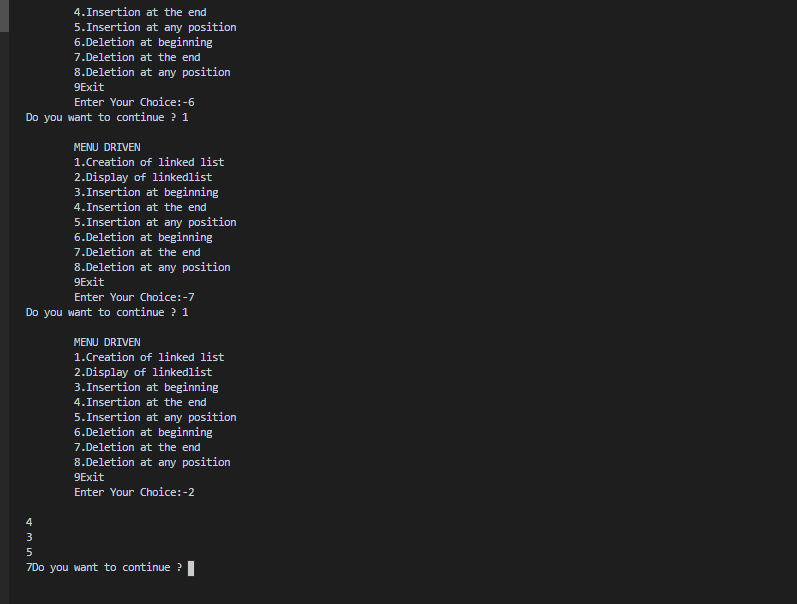
**return 0;**

**}**

**Result /Output :**

** **

****

****

**Practical-3**

1. **PRACTICAL STATEMENT OF PRACTICAL: (a) Implement stack using array and perform operations like insertion , deletion and display.**

**(b) Implement stack using LINKED LIST and perform operations like insertion, deletion and display.**

1. **OBJECTIVE OF PRACTICAL : Perform different operations on stack using array and linked list.**
2. **ALGORITHM / FLOW CHART:**

**(a) Using array:**

**Begin:**

1. **Create an array with fixed size.**
2. **Define an integer top and initialize it with -1.**
3. **Write a push function for insertion-Check if stack is full or not .If it is full then print overflow else insert element in stack.**
4. **Write a pop function check whether top =-1.If top=-1 print underflow else delete the element on top of stack and decrement the value of top with -1.**
5. **Write a display function and run a loop from 0 to top and print all the elements in the stack.**

**6. END**

**(b) Using linked list:**

**Begin:**

**Step 1 - Define a 'Node' structure with two members data and next.**

**Step 2 - Define a Node pointer 'top' and set it to NULL.**

**Step 3 - Implement the main method by displaying Menu with list of operations and make suitable function calls in the main method.**

**Step-4 – Create push function-**

**(a) Define a 'Node' structure with two members data and next.**

**(b) Define a Node pointer 'top' and set it to NULL.**

**(c) Implement the main method by displaying Menu with list of operations and make suitable function calls in the main method.**

**Step-5-Create a pop function:**

**(a) - Check whether stack is Empty (top == NULL).**

**(b) - If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the function**

**(c) - If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.**

**(d) - Then set 'top = top → next'.**

**(e) - Finally, delete 'temp'. (free(temp)).**

**Step-6-Display a function:**

**(a)- Check whether stack is Empty (top == NULL).**

**(b) - If it is Empty, then display 'Stack is Empty!!!' and terminate the function.**

**(c) - If it is Not Empty, then define a Node pointer 'temp' and initialize with top.**

**(d) - Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to the first node in the stack. (temp → next != NULL).**

**(e) – Finally Display 'temp → data ---> NULL'.**

1. **IMPLEMENTATION of stack with array :**

#include<iostream>

int stk[50],top=-1,max,ele;

int c,i;

void push()

{

    printf("Enter the element to be inserted ");

    scanf("%d",&ele);

    if(top==-1)

    {

        top=0;

        stk[top]=ele;

    }

    else if(top+1<max)

    {

        stk[++top]=ele;

    }

    else

    {

        printf(" Overflow \n");

    }

}

void pop()

{

    if(top==-1)

    {

        printf(" Underflow \n");

    }

    else

    {

        ele=stk[top--];

    }

}

void display()

{

    printf(" \nStack = ");

    for(i=top;i>=0;i--)

    {

        printf(" %d ",stk[i]);

    }

}

int main()

{

    printf("\tCode by SAMARPIT DUA (1900321290050)\n\t STACK IMPLEMENTATION\n");

    printf("Enter the maximum size of stack \n");

    scanf("%d",&max);

    int ch;

    do

    {

    printf("Enter 1 for insertion , 2 for deletion and 3 for display : ");

    scanf("%d",&ch);

    switch(ch)

    {

        case 1:

            push();

            break;

        case 2:

            pop();

            break;

        case 3:

            display();

            break;

        default:

            printf(" invalid input\n");

            break;

    }

    printf("\n\n Press 2 to continue and 0 to exit ");

    scanf("%d",&c);

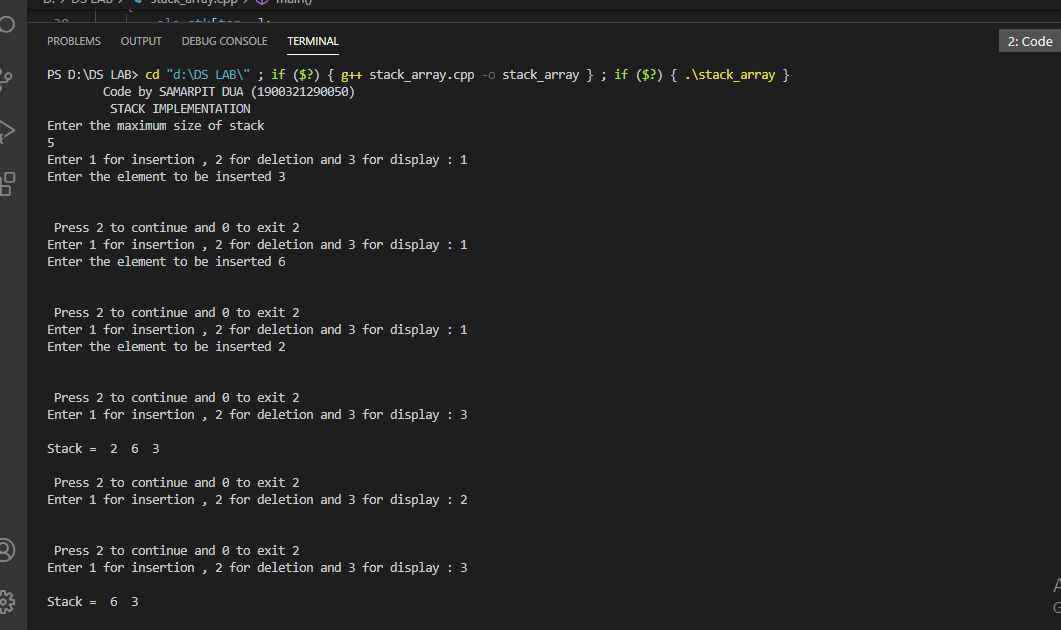
    }

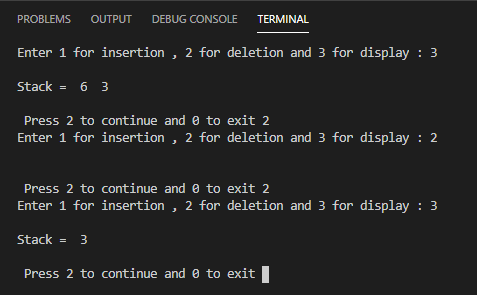
    while(c==2);

    return 0;

}

1. **Result /Output of stack with array:**





**3(b). Implementation of stack using linked list:**

**#include<iostream>**

**#include<stdlib.h>**

**typedef struct stack**

**{**

**int data;**

**struct stack \*next;**

**}stk;**

**stk \*top=NULL,\*temp,\*start=NULL;**

**void push()**

**{**

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

**printf("Enter the data\n");**

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

**temp->next=top;**

**top=temp;**

**}**

**void pop()**

**{**

**stk\*ptr;**

**if(top == NULL)**

**printf("Stack is empty\n");**

**else**

**{**

**ptr=top;**

**printf("Popped item is %d\n",ptr->data);**

**top=top->next;**

**free(ptr);**

**}**

**}**

**void display()**

**{**

**stk\*ptr;**

**ptr=top;**

**if(top==NULL)**

**printf("Stack is empty\n");**

**else**

**{**

**printf("\nStack = ");**

**while(ptr!= NULL)**

**{**

**printf("%d ",ptr->data);**

**ptr = ptr->next;**

**}**

**}**

**}**

**int c,ch;**

**int main()**

**{**

**printf("SAMARPIT DUA (1900321290050)\n\t STACK IMPLEMENTATION WITH LINKED LIST \n");**

**do**

**{**

**printf("Enter 1 for insertion 2 for deletion and 3 for display : \n ");**

**scanf("%d",&ch);**

**switch(ch)**

**{**

**case 1:**

**push();**

**break;**

**case 2:**

**pop();**

**break;**

**case 3:**

**display();**

**break;**

**default:**

**printf("Invalid input");**

**break;**

**}**

**printf("\n Press 2 to continue");**

**scanf("%d",&c);**

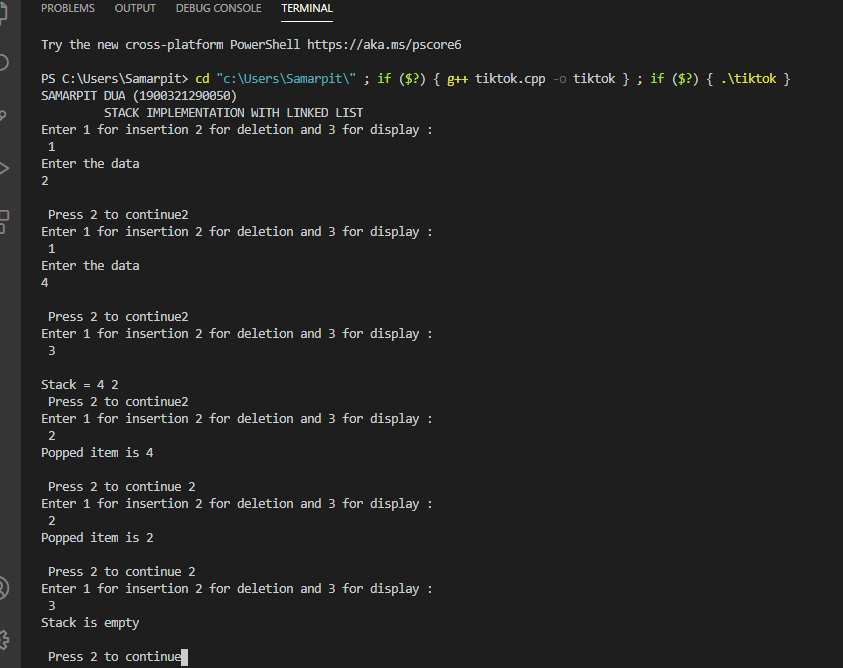
**}**

**while(c==2);**

**return 0;**

**}**

**5(b). Result/Output:**

****

**Practical-4**

1. **PRACTICAL STATEMENT OF PRACTICAL: Implementation of Queue using Array and Linked List.**
2. **OBJECTIVE OF PRACTICAL : Implementation of Queue using Array and Linked List.**
3. **LANGUAGE REQUIRED: C Language**
4. **ALGORITHM:**

**⦁ALGORITHM INTIALIZE(QUEUE Q)**

**BEGIN:**

**Q.REAR=0 Q.FRONT=1**

**END;**

**⦁ALGORITHM ENQUEUE(QUEUE Q,key)**

**BEGIN:**

**IF Q.REAR ==SIZE THEN**

**write (Queue overflow) Exit 1**

**Q.REAR=Q.REAR+1**

**Q.item[Q.REAR]=key**

**END;**

**ALGORITHM DEQUEUE(QUEUE Q)**

**BEGIN:**

**IF Q.REAR-Q.FRONT+1==0 THEN**

**Exit(1) x=Q.item[Q.FRONT] Q.FRONT =Q.FRONT +1 RETURN x**

**END;**

**⦁ALGORITHM EMPTY(QUEUE Q)**

**BEGIN:**

**IF Q.REAR – Q.FRONT +1 == 0 THEN RETURN TRUE**

**ELSE RETURN FALSE**

**END;**

**For all the above ALGORITHM**

**Time Complexity: ϴ (1)**

**Space Complexity: ϴ (1)**

1. **IMPLEMENTATION :**

**USING ARRAY---->**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX 500**

**int front, rear;**

**int q[MAX];**

**//INSERT ELEMENT IN QUEUE**

**void insertq(int element)**

**{**

**if (rear == MAX)**

**{**

**printf("Overflow");**

**return;**

**}**

**if (front == -1 && rear == -1)**

**{**

**front = rear = 0;**

**q[rear] = element;**

**}**

**else**

**{**

**rear++;**

**q[rear] = element;**

**}**

**}**

**//DELETION IN QUEUE**

**void deleteq()**

**{**

**if (front == -1)**

**{**

**printf("\nUnderflow");**

**return;**

**}**

**int k = q[front];**

**if (front == rear)**

**{**

**rear = front = -1;**

**}**

**else**

**{**

**front++;**

**}**

**printf("\nDeleted Element id %d", k);**

**}**

**//DISPLAY OF QUEUE**

**void display()**

**{**

**if (front == -1)**

**{**

**printf("\nQueue is empty");**

**return;**

**}**

**for (int i = front; i <= rear; i++)**

**{**

**printf("\t%d", q[i]);**

**}**

**}**

**int main()**

**{**

**printf("\n\tSAMARPIT DUA 1900321290050\n\tIMPLEMENTATON OF QUEUE USING ARRAYS ");**

**rear = front = -1;**

**int ch, element;**

**while (1)**

**{**

**printf("\n\tMAIN MENU");**

**printf("\n\t1.Insert element in queue.\n\t2.Deletion of element in queue.\n\t3.Display queue.\n\t4.EXIT.");**

**printf("\nEnter your choice:-");**

**scanf("%d", &ch);**

**switch (ch)**

**{**

**case 1:**

**printf("\nEnter the element to be inserted:-");**

**scanf("%d", &element);**

**insertq(element);**

**break;**

**case 2:**

**deleteq();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**exit(0);**

**break;**

**default:**

**printf("\nWRONG CHOICE!!");**

**};**

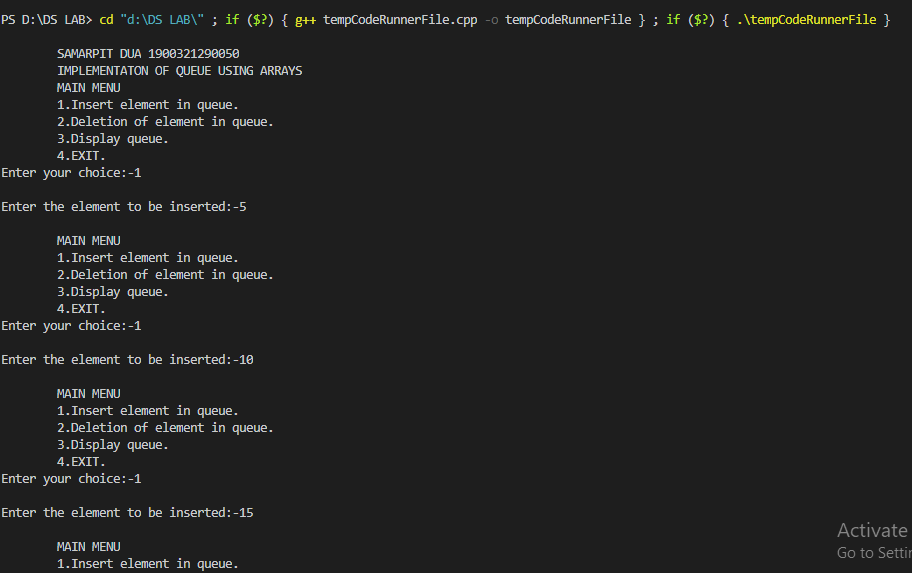
**}**

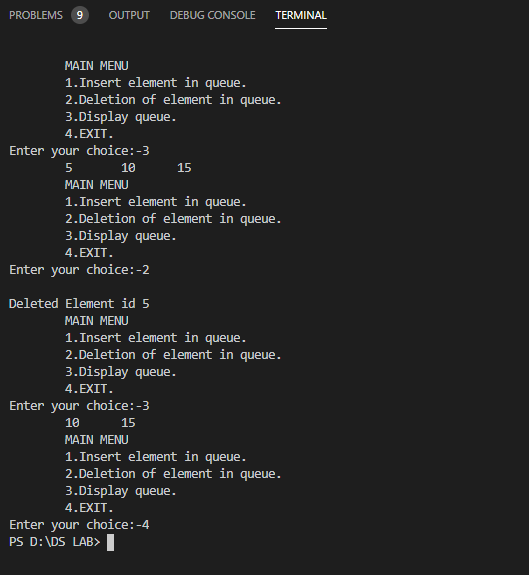
**return 0;**

**}**

**Result /Output :**

**Using array--🡪**

****

****

**IMPLEMENTATION:**

**USING LINKED LIST --🡪**

**#include <stdio.h>**

**#include <stdlib.h>**

**typedef struct queue**

**{**

**int info;**

**struct queue \*next;**

**} pq;**

**pq \*rear = NULL, \*temp = NULL, \*front = NULL;**

**void insertq()**

**{**

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

**if (temp == NULL)**

**{**

**printf("\nUnderflow");**

**return;**

**}**

**printf("\nEnter Info:-");**

**scanf("%d", &temp->info);**

**temp->next = NULL;**

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

**{**

**front = temp;**

**rear = temp;**

**}**

**else**

**{**

**rear->next = temp;**

**rear = temp;**

**}**

**}**

**void deleteq()**

**{**

**if (front == NULL)**

**{**

**printf("\nUnderflow");**

**return;**

**}**

**pq \*ptr = front;**

**printf("\nDeleted element is %d", front->info);**

**if (front == rear)**

**{**

**front = rear = NULL;**

**}**

**else**

**{**

**front = front->next;**

**}**

**free(ptr);**

**}**

**void display()**

**{**

**pq \*pyr = front;**

**printf("\nElements in queue:-");**

**while (pyr != NULL)**

**{**

**printf("\t%d", pyr->info);**

**pyr = pyr->next;**

**}**

**}**

**int main()**

**{**

**printf("\n\tSAMARPIT DUA 1900321290050\n\tIMPLEMENTATON OF QUEUE USING LINKED LIST");**

**int ch;**

**while (1)**

**{**

**printf("\n\t\tMAIN MENU\n\t1.Insertion\n\t2.Deletion\n\t3.Display.\n\t4.Exit");**

**printf("\nEnter your choice:-");**

**scanf("%d", &ch);**

**switch (ch)**

**{**

**case 1:**

**insertq();**

**break;**

**case 2:**

**deleteq();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**exit(0);**

**};**

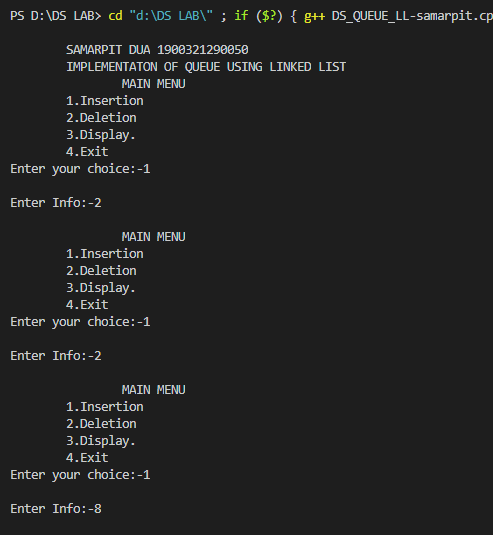
**}**

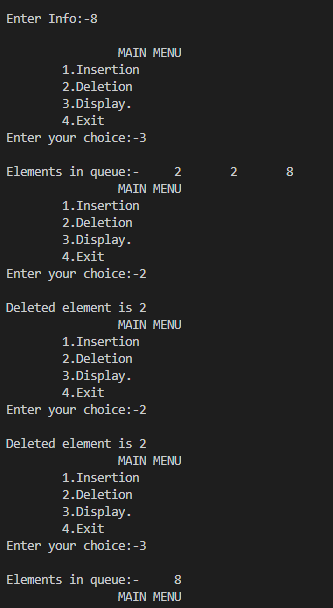
**return 0;**

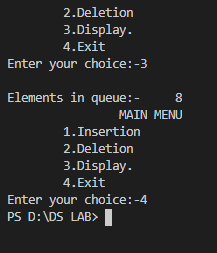
**}**

**RESULT/OUTPUT:**

**USING LINKED LIST--🡪**

****

****

****

**Practical NO. 5**

**Write a program in C to implement Circular Queue using array.**

**Objective : Implement Circular Queue using array  
Implementation :**

**#include<stdio.h>**

**int que[50],front=-1,rear=-1,max,ele;**

**int i,cc;**

**void main()**

**{**

**printf("enter the maximum size of queue\n");**

**scanf("%d",&max);**

**int ch;**

**do**

**{**

**printf("enter 1 for insertion and 2 for deletion \n");**

**scanf("%d",&ch);**

**switch(ch)**

**{**

**case 1:**

**insert\_rear();**

**display();**

**break;**

**case 2:**

**delete();**

**display();**

**break;**

**default:**

**printf("invalid input\n");**

**break;**

**}**

**printf("\n press 2 to continue ");**

**scanf("%d",&cc);**

**}**

**while(cc==2);**

**}**

**void insert\_rear()**

**{**

**printf("enter the element to be inserted ");**

**scanf("%d",&ele);**

**if(rear==-1&&front==-1)**

**{**

**front=0;**

**rear=0;**

**que[rear]=ele;**

**}**

**else if((rear+1)%max!=front)**

**{**

**rear=(rear+1)%max;**

**que[rear]=ele;**

**}**

**else{**

**printf("overflow");**

**}**

**}**

**void delete()**

**{**

**if(rear==-1&&front==-1)**

**printf("not possible");**

**else if(front==rear)**

**{**

**ele=que[front];**

**front=-1;**

**rear=-1;**

**}**

**else**

**{**

**ele=que[front];**

**front=(front+1)%max;**

**}**

**}**

**void display()**

**{**

**int front\_pos = front,rear\_pos = rear;**

**if(front == -1)**

**{**

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

**return;**

**}**

**printf("Queue elements :\n");**

**if( front\_pos <= rear\_pos )**

**while(front\_pos <= rear\_pos)**

**{**

**printf("%d ",que[front\_pos]);**

**front\_pos++;**

**}**

**else**

**{**

**while(front\_pos <= max-1)**

**{**

**printf("%d ",que[front\_pos]);**

**front\_pos++;**

**}**

**front\_pos = 0;**

**while(front\_pos <= rear\_pos)**

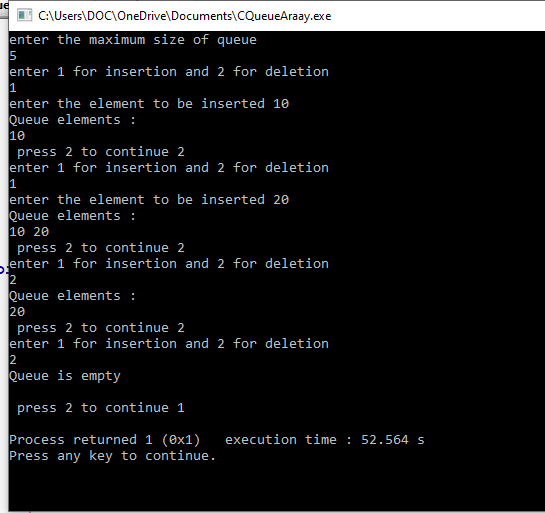
**{**

**printf("%d ",que[front\_pos]);**

**front\_pos++;**

**}}}**

**OUTPUT:**

****

**Practical No. 5(b)**

**Implementation of Circular Queue using Linked List**

**Implementation :**

**#include<stdio.h>**

**#include<stdlib.h>**

**typedef struct queue**

**{**

**int data;**

**struct queue \*next;**

**}que;**

**que \*front=NULL,\*rear=NULL,\*temp;**

**int c,ch;**

**void main()**

**{**

**do**

**{**

**printf("enter 1 for insertion and 2 for deletion\n ");**

**scanf("%d",&ch);**

**switch(ch)**

**{**

**case 1:**

**insert();**

**display();**

**break;**

**case 2:**

**delete();**

**display();**

**break;**

**default:**

**printf("invalid input");**

**break;**

**}**

**printf("\n press 2 to continue");**

**scanf("%d",&c);**

**}**

**while(c==2);**

**}**

**void insert()**

**{**

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

**printf("enter the element to be inserted ");**

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

**if(front==NULL)**

**{**

**front=temp;**

**rear=temp;**

**temp->next=front;**

**}**

**else**

**{**

**rear->next=temp;**

**temp->next=front;**

**rear=temp;**

**}**

**}**

**void delete()**

**{**

**que\* ptr;**

**int ele;**

**if(front==NULL)**

**{**

**printf(" EMPTY QUEUE \n");**

**return;**

**}**

**ptr=front;**

**if(rear->next==rear)**

**{**

**ele=ptr->data;**

**front=NULL;**

**rear=NULL;**

**printf("\n %d IS DELETED!",ele);**

**}**

**else**

**{**

**ele=ptr->data;**

**rear->next=front->next;**

**front=front->next;**

**printf("\n %d IS DELETED!",ele);**

**}**

**free(ptr);**

**}**

**void display()**

**{**

**que\* ptr=front;**

**if(front==NULL)**

**{**

**printf(" EMPTY QUEUE \n");**

**}**

**printf("\nQueue is =");**

**ptr=rear->next;**

**do**

**{**

**printf("%d ",ptr->data);**

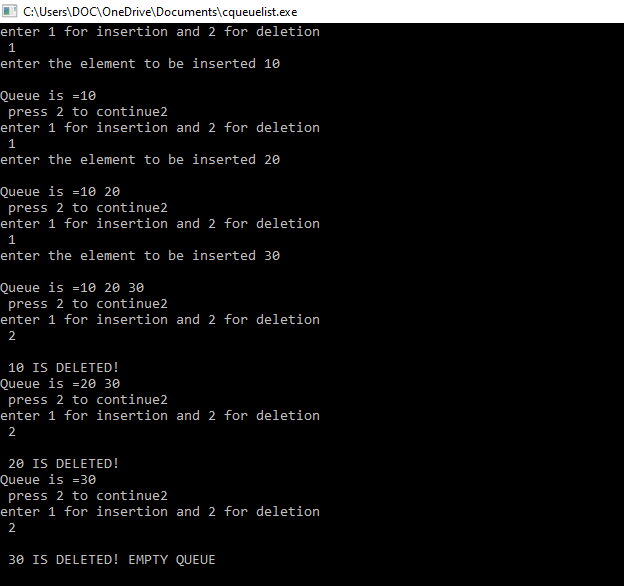
**ptr=ptr->next;**

**}**

**while(ptr!=rear->next);**

**}**

**OUTPUT**

****

**PRACTICAL - 6**

1. **PRACTICAL STATEMENT OF PRACTICAL:** Search an element in array.
2. **OBJECTIVE OF PRACTICAL :**  Fine an element in given array using different algorithms & calculate time and space complexity.
3. **ALGORITHM / FLOW CHART:**
4. **Linear Search**

**Linear\_search(A[], N, key) BEGIN:**

FOR i=1 TO N DO

IF A[i]==key

THEN RETURN i

RETURN -1

### END;

**Worst Case Time Complexity: O(N) ,Best Case Time Complexity: (1) and Space Complexity: ϴ(1)**

1. **Binary Search**

Begin:

HIGH=N-1, LOW=0

WHILE (LOW<=HIGH) DO

MID=(LOW+HIGH)/2

IF A[MID]==key THEN

RETURN MID

ELSE IF (key<A[MID]) THEN

HIGH=MID-1

ELSE

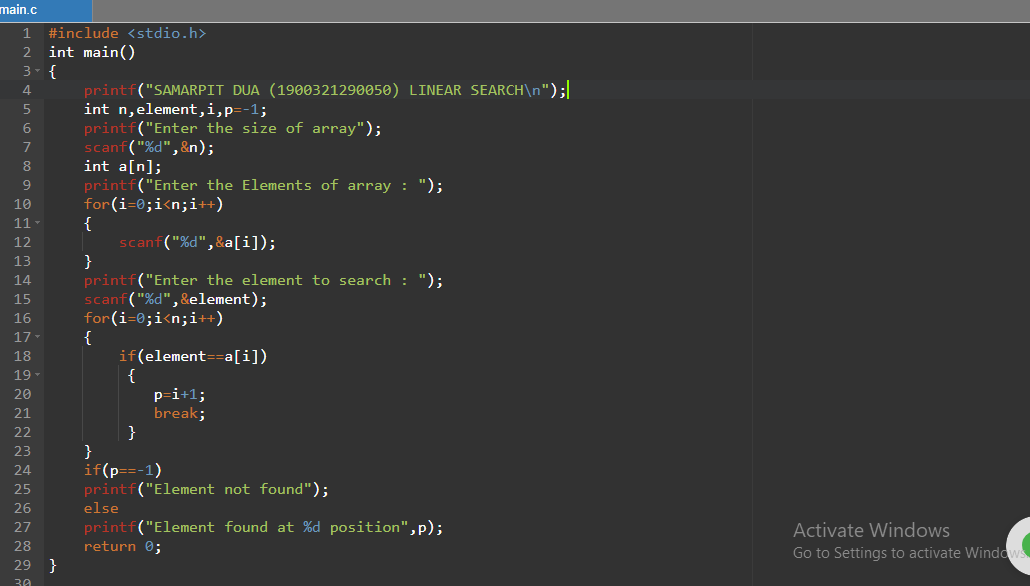
LOW=LOW=MID+1

END

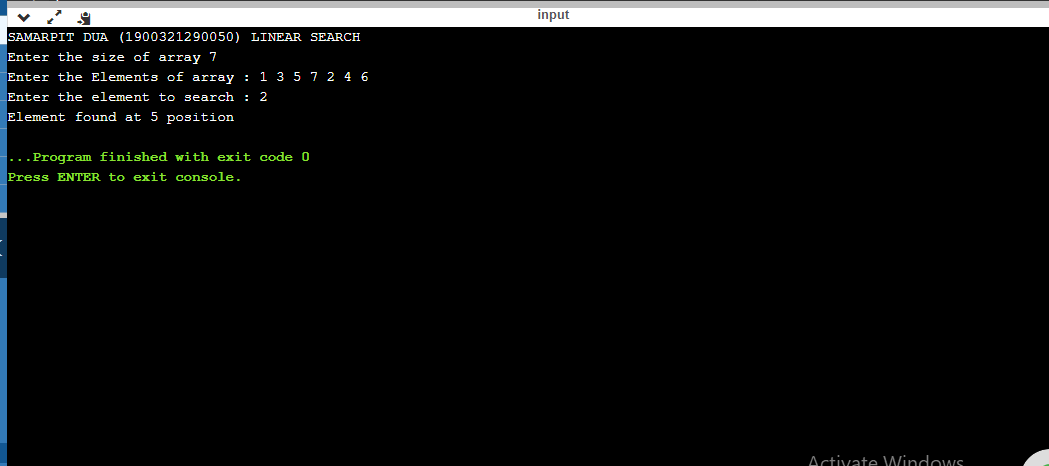
### Worst Case Time Complexity: O(logN)

### Best Case Time Complexity: (1) Space Complexity: ϴ(1)

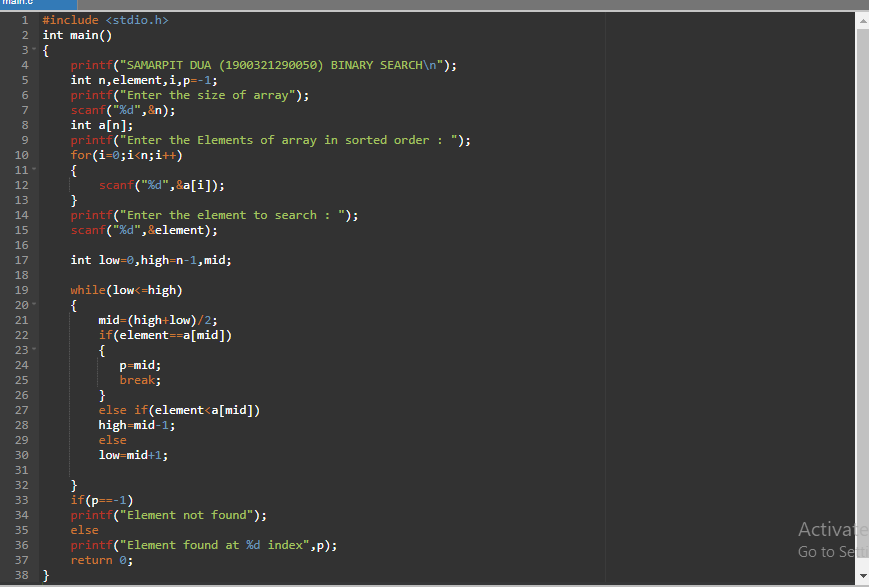
1. **(a)IMPLEMENTATION(Linear Search) :**

****

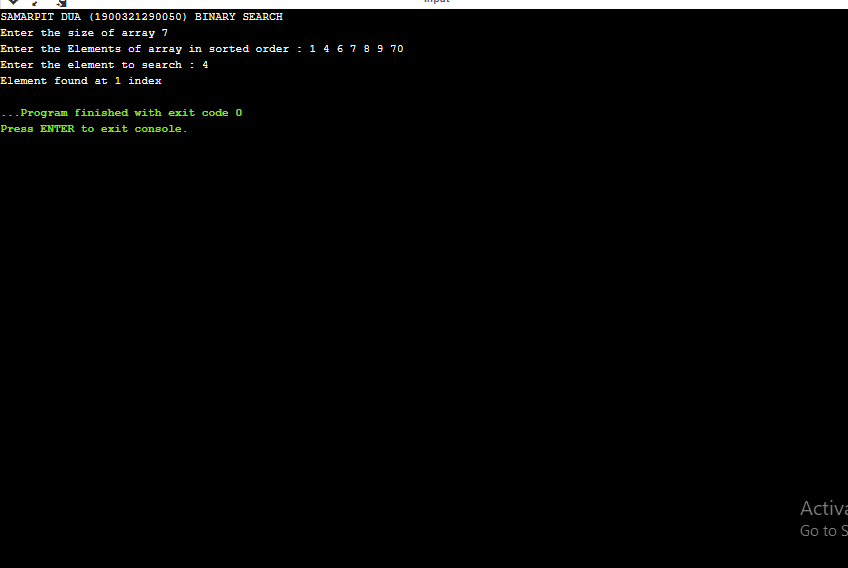
1. **(a)Result/OUTPUT(Linear Search) :**



1. **Implementation(Binary Search):**



1. **Result(Binary Search) :**



**Practical -7(a)**

1. **PRACTICAL STATEMENT OF PRACTICAL: Sort the given array.**
2. **OBJECTIVE OF PRACTICAL : Sort the given array using quick sorting algorithm & calculate time and space complexity.**
3. **IMPLEMENTATION :**

#include <stdio.h>

void quicksort(int number[25], int first, int last)

{

    int i, j, pivot, temp;

    if (first < last)

    {

        pivot = first;

        i = first;

        j = last;

        while (i < j)

        {

            while (number[i] <= number[pivot] && i < last)

                i++;

            while (number[j] > number[pivot])

                j--;

            if (i < j)

            {

                temp = number[i];

                number[i] = number[j];

                number[j] = temp;

            }

        }

        temp = number[pivot];

        number[pivot] = number[j];

        number[j] = temp;

        quicksort(number, first, j - 1);

        quicksort(number, j + 1, last);

    }

}

int main()

{

    int i, count, number[25];

    printf("\n\tSAMARPIT DUA 1900321290050\n\tQUICK SORT\n\t");

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

    scanf("%d", &count);

    printf("Enter %d elements: ", count);

    for (i = 0; i < count; i++)

        scanf("%d", &number[i]);

    quicksort(number, 0, count - 1);

    printf("The Sorted Order is: ");

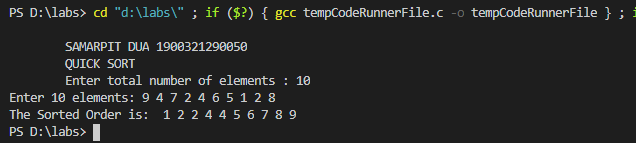
    for (i = 0; i < count; i++)

        printf(" %d", number[i]);

    return 0;

}

**Result /Output :**

****

**Practical-7(b)**

1. **PRACTICAL STATEMENT OF PRACTICAL: Sorting Algorithms-Recursive.**
2. **OBJECTIVE OF PRACTICAL : Sort the given array using MERGE SORT algorithm & calculate time and space complexity.**
3. **IMPLEMENTATION :**

#include <stdio.h>

int a[20], b[20];

int n, i, j, k;

void Merge(int a[], int lb, int mid, int ub)

{

    i = lb;

    j = mid + 1;

    k = 0;

    while (i <= mid && j <= ub)

    {

        if (a[i] < a[j])

        {

            b[k++] = a[i++];

        }

        else

        {

            b[k++] = a[j++];

        }

    }

    if (i > mid)

    {

        while (j <= ub)

        {

            b[k++] = a[j++];

        }

    }

    else

    {

        while (i <= mid)

        {

            b[k++] = a[i++];

        }

    }

    for (i = lb, j = 0; i <= ub; i++, j++)

        a[i] = b[j];

}

void Merge\_sort(int a[], int lb, int ub)

{

    if (lb < ub)

    {

        int mid = (lb + ub) / 2;

        Merge\_sort(a, lb, mid);

        Merge\_sort(a, mid + 1, ub);

        Merge(a, lb, mid, ub);

    }

}

void main()

{

    printf("SAMARPIT DUA 1900321290050\n\tMERGE SORT");

    printf("\nEnter size of array:-");

    scanf("%d", &n);

    printf("\nEnter Array elements:-");

    for (int i = 0; i < n; i++)

        scanf("%d", &a[i]);

    printf("\nMerge Sort:-\n");

    Merge\_sort(a, 0, n - 1);

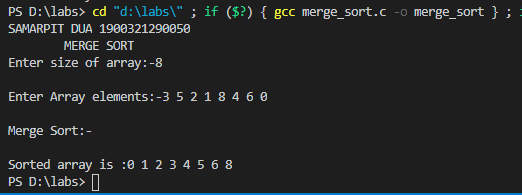
    printf("\nSorted array is :");

    for (i = 0; i < n; i++)

        printf("%d ", a[i]);

}

1. **Result /Output :**

****

**PRACTICAL -7(c)**

1. **PRACTICAL STATEMENT OF PRACTICAL: Write a program to implement Heap Sort.**
2. **OBJECTIVE OF PRACTICAL : Implementation of Heap Sort.**
3. **IMPLEMENTATION :**

#include <stdio.h>

void main()

{

    int heap[10], no, i, j, c, root, temp;

    printf("\nSAMARPIT DUA 1900321290050\n");

    printf("\n Enter no of elements :");

    scanf("%d", &no);

    printf("\n Enter the nos : ");

    for (i = 0; i < no; i++)

       scanf("%d", &heap[i]);

    for (i = 1; i < no; i++)

    {

        c = i;

        do

        {

            root = (c - 1) / 2;

            if (heap[root] < heap[c])

            {

                temp = heap[root];

                heap[root] = heap[c];

                heap[c] = temp;

            }

            c = root;

        } while (c != 0);

    }

    printf("Heap array : ");

    for (i = 0; i < no; i++)

        printf("%d\t ", heap[i]);

    for (j = no - 1; j >= 0; j--)

    {

        temp = heap[0];

        heap[0] = heap[j];

        heap[j] = temp;

        root = 0;

        do

        {

            c = 2 \* root + 1;

            if ((heap[c] < heap[c + 1]) && c < j-1)

                c++;

            if (heap[root]<heap[c] && c<j)             {

                temp = heap[root];

                heap[root] = heap[c];

                heap[c] = temp;

            }

            root = c;

        } while (c < j);

    }

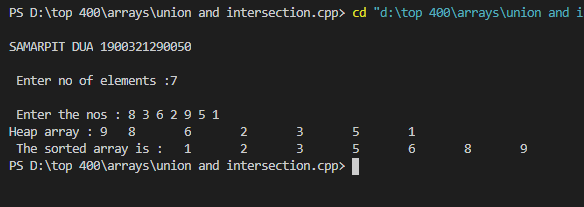
    printf("\n The sorted array is : ");

    for (i = 0; i < no; i++)

       printf("\t %d", heap[i]);

}

**Result /Output :**

****

**Practical – 8(a)**

1. **PRACTICAL STATEMENT OF PRACTICAL:** **Sort the given Array using Bubble Sort.**
2. **OBJECTIVE OF PRACTICAL : Sort the given array using different sorting algorithms & calculate time and space complexity.**

### ALGORITHM:

**BubbleSort(A[], N)**

BEGIN: FOR i=1 TO N-1 DO

FOR j=1 TO N-i DO

IF A[j]>A[j+1]

k=A[j]

A[j]=A[j+1]

A[j+1]=k

END;

Worst Case Time Complexity:O(N2 )

Best Case Time Complexity: Omega(N)

Space Complexity:ϴ(1)

1. **IMPLEMENTATION :**

#include <stdio.h>

int main()

{

    int a[10], n, i, j, temp = 0;

    printf("\n\tSAMARPIT DUA 1900321290050\n\t BUBBLE SORT");

    printf("Enter size of array-");

    scanf("%d", &n);

    printf("\nEnter Array-");

    for (i = 0; i < n; i++)

        scanf("%d", &a[i]);

    for (i = 0; i < n - 1; i++)

    {

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

        {

            if (a[j] > a[j + 1])

            {

                temp = a[j];

                a[j] = a[j + 1];

                a[j + 1] = temp;

            }

        }

    }

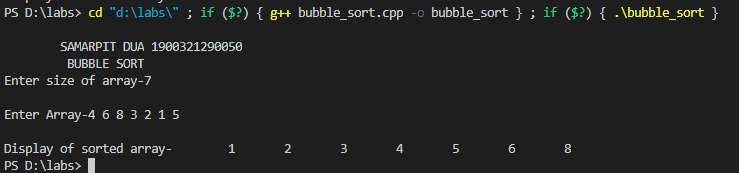
    printf("\nDisplay of sorted array-");

    for (i = 0; i < n; i++)

        printf("\t%d", a[i]);

        return 0;

}

**Result /Output : **

**Practical-8(b)**

1. **PRACTICAL STATEMENT OF PRACTICAL : Sort the given Array using selection sort.**
2. **OBJECTIVE OF PRACTICAL : Sort the given array using different sorting algorithms & calculate time and space complexity.**
3. **ALGORITHM :**

**SelectionSort(A[], N)**

**BEGIN:**

**FOR i=1 TO N-1 DO**

**min=i**

**FOR j=i+1 TO N DO**

**IF A[j] <A[min] then**

**Min=j**

**Exchange(A[min],A[i])**

**Time Complexity:O(N2 )**

**Space Complexity:ϴ(1)**

1. **IMPLEMENTATION :**

#include<stdio.h>

int main()

{

    printf("\n\tCODE BY SAMARPIT DUA (1900321290050)\n\tSELECTION SORT");

    int n;

    printf("\nEnter the total no. of elements of array : ");

    scanf("%d",&n);

    int a[n];

    printf("\nEnter the elements of array : ");

    for(int i=0;i<n;i++)

    scanf("%d",&a[i]);

    for(int i=0;i<n;i++)

    {

        for(int j=i+1;j<n;j++)

        {

              if(a[j]<a[i])

              {

                  int temp=a[j];

                  a[j]=a[i];

                  a[i]=temp;

              }

        }

    }

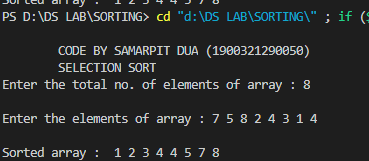
    printf("\nSorted array : ");

    for(int i=0;i<n;i++)

    printf(" %d",a[i]);

}

1. **Result /Output :**

****

**Practical-8(c)**

1. **PRACTICAL STATEMENT OF PRACTICAL:** **Sort the given array using insertion sort.**
2. **OBJECTIVE OF PRACTICAL: Sort the given array using different sorting algorithms & calculate time and space complexity.**
3. **ALGORITHM :**

**InsertionSort(A[], N)**

**BEGIN: FOR i=2 TO N DO**

**key=A[i] j=i-1**

**WHILE j>=1 AND A[j]>key DO**

**A[j+1]=A[j]**

**j=j-1 A[j+1]=key**

**END;**

**Worst Case Time Complexity:O(N2 )**

**Best Case Time Complexity: Omega(N)**

**Space Complexity:ϴ(1)**

1. **IMPLEMENTATION :**

#include<stdio.h>

int main()

{

    printf("\n\tCODE BY SAMARPIT DUA (1900321290050)\n\t INSERTION SORT");

    int n;

    printf("\nEnter the total no. of elements of array : ");

    scanf("%d",&n);

    int a[n];

    printf("\nEnter the elements of array : ");

    for(int i=0;i<n;i++)

    scanf("%d",&a[i]);

    for(int i=1;i<n;i++)ert

    {

        int curr=a[i];

        int j=i-1;

        while(a[j]>curr&&j>=0)

        {

            a[j+1]=a[j];

            j--;

        }

        a[j+1]=curr;

    }

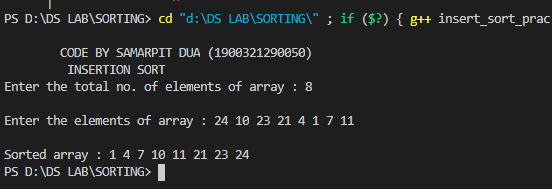
    printf("\nSorted array : ");

    for(int i=0;i<n;i++)

    printf("%d ",a[i]);

    return 0;

}

**Result /Output :** 

**PRACTICAL -8(d)**

1. **PRACTICAL STATEMENT OF PRACTICAL: Write a program to implement Count Sort.**
2. **OBJECTIVE OF PRACTICAL : Implementation of Count Sort.**
3. **IMPLEMENTATION :**

**#include <stdio.h>**

**void counting\_sort(int A[], int k, int n)**

**{**

**int i, j;**

**int B[15], C[100];**

**for (i = 0; i <= k; i++)**

**C[i] = 0;**

**for (j = 1; j <= n; j++)**

**C[A[j]] = C[A[j]] + 1;**

**for (i = 1; i <= k; i++)**

**C[i] = C[i] + C[i-1];**

**for (j = n; j >= 1; j--)**

**{**

**B[C[A[j]]] = A[j];**

**C[A[j]] = C[A[j]] - 1;**

**}**

**printf("The Sorted array is : ");**

**for (i = 1; i <= n; i++)**

**printf("%d ", B[i]);**

**}**

**int main()**

**{**

**printf("SAMARPIT DUA 1900321290050 \n ");**

**int n, k = 0, A[15], i;**

**printf("Enter the number of input : ");**

**scanf("%d", &n);**

**printf("\nEnter the elements to be sorted :\n");**

**for (i = 1; i <= n; i++)**

**{**

**scanf("%d", &A[i]);**

**if (A[i] > k) {**

**k = A[i];**

**}**

**}**

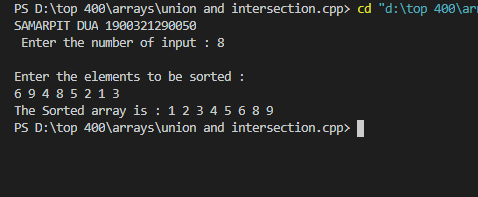
**counting\_sort(A, k, n);**

**printf("\n");**

**return 0;**

**}**

**Result /Output :**

****

**Practical -8(e)**

1. **PRACTICAL STATEMENT OF PRACTICAL:** **Sort the given Array.**
2. **OBJECTIVE OF PRACTICAL :** **Sort the given array using RADIX SORT & calculate time and space complexity.**
3. **ALGORITHM:**

**RadixSort(A[],N,d)**

BEGIN:

FOR i=1 TO d DO

Apply counting Sort on A[] at radix i

END;

**Time Complexity**: ϴ(N)

**Space Complexity**:ϴ(N)

1. **IMPLEMENTATION :**

#include <stdio.h>

int get\_max(int a[], int n)

{

    int max = a[0];

    for (int i = 1; i < n; i++)

        if (a[i] > max)

            max = a[i];

    return max;

}

void radix\_sort(int a[], int n)

{

    int bucket[10][10], bucket\_cnt[10];

    int i, j, k, r, NOP = 0, divisor = 1, lar, pass;

    lar = get\_max(a, n);

    while (lar > 0)

    {

        NOP++;

        lar /= 10;

    }

    for (pass = 0; pass < NOP; pass++)

    {

        for (i = 0; i < 10; i++)

        {

            bucket\_cnt[i] = 0;

        }

        for (i = 0; i < n; i++)

        {

            r = (a[i] / divisor) % 10;

            bucket[r][bucket\_cnt[r]] = a[i];

            bucket\_cnt[r] += 1;

        }

        i = 0;

        for (k = 0; k < 10; k++)

        {

            for (j = 0; j < bucket\_cnt[k]; j++)

            {

                a[i] = bucket[k][j];

                i++;

            }

        }

        divisor \*= 10;

        printf("After pass %d : ", pass + 1);

        for (i = 0; i < n; i++)

            printf("%d ", a[i]);

        printf("\n");

    }

}

int main()

{

    int i, n, a[100];

    printf("SAMARPIT DUA 1900321290050 \n\t RADIX SORT\n\t");

    printf("Enter the number of items to be sorted: ");

    scanf("%d", &n);

    printf("Enter items: ");

    for (i = 0; i < n; i++)

    {

        scanf("%d", &a[i]);

    }

    radix\_sort(a, n);

    printf("Sorted items : ");

    for (i = 0; i < n; i++)

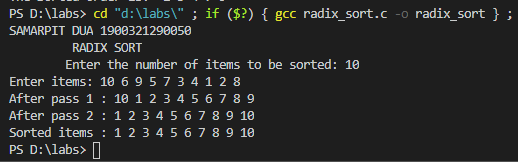
        printf("%d ", a[i]);

    printf("\n");

    return 0;

}

**Result /Output** :



**Practical-9**

1. **PRACTICAL STATEMENT OF PRACTICAL: Implementation of Tree Structures, Binary Tree and Tree Traversal using arrays and linked list.**
2. **OBJECTIVE OF PRACTICAL :** **Implementation of Binary Trees using arrays and linked lists.**
3. **IMPLEMENTATION :**

**USING ARRAYS:**

#include <stdio.h>

#include <stdlib.h>

struct tree

{

  struct tree \*lchild;

  char data[10];

  struct tree \*rchild;

};

typedef struct tree node;

int ctr;

node \*tree[100];

node \*getnode()

{

  node \*temp;

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

  printf("\n Enter Data: ");

  scanf("%s", temp->data);

  temp->lchild = NULL;

  temp->rchild = NULL;

  return temp;

}

void create\_fbinarytree()

{

  int j, i = 0;

  printf("\n Enter Number of nodes you want: ");

  scanf("%d", &ctr);

  tree[0] = getnode();

  j = ctr;

  j--;

  do

  {

    if (j > 0)

    {

      tree[i \* 2 + 1] = getnode();

      tree[i]->lchild = tree[i \* 2 + 1];

      j--;

    }

    if (j > 0)

    {

      tree[i \* 2 + 2] = getnode();

      j--;

      tree[i]->rchild = tree[i \* 2 + 2];

    }

    i++;

  } while (j > 0);

}

void inorder(node \*root)

{

  if (root != NULL)

  {

    inorder(root->lchild);

    printf("%3s", root->data);

    inorder(root->rchild);

  }

}

void preorder(node \*root)

{

  if (root != NULL)

  {

    printf("%3s", root->data);

    preorder(root->lchild);

    preorder(root->rchild);

  }

}

void postorder(node \*root)

{

  if (root != NULL)

  {

    postorder(root->lchild);

    postorder(root->rchild);

    printf("%3s", root->data);

  }

}

void levelorder()

{

  int j;

  for (j = 0; j < ctr; j++)

  {

    if (tree[j] != NULL)

      printf("%3s", tree[j]->data);

  }

}

void print\_leaf(node \*root)

{

  if (root != NULL)

  {

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

      printf("%3s ", root->data);

    print\_leaf(root->lchild);

    print\_leaf(root->rchild);

  }

}

void main()

{

  int i;

  printf("\n\tSAMARPIT DUA 1900321290050 Binary Tree Implementation Using Arrays\n");

  create\_fbinarytree();

  printf("\n Inorder Traversal: ");

  inorder(tree[0]);

  printf("\n Preorder Traversal: ");

  preorder(tree[0]);

  printf("\n Postorder Traversal: ");

  postorder(tree[0]);

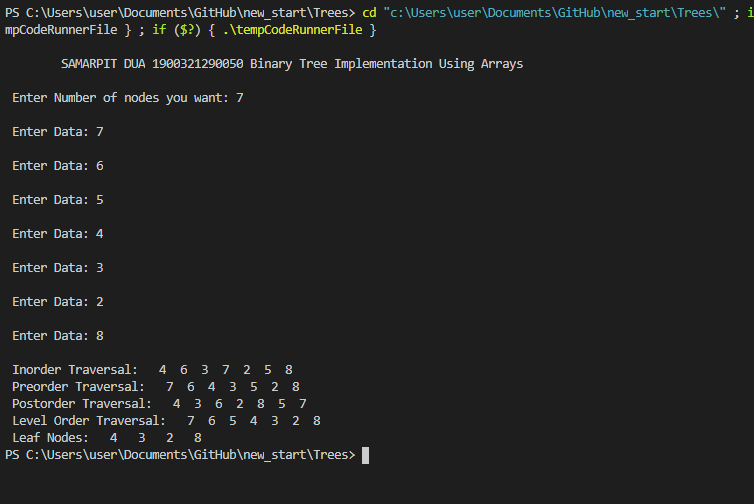
  printf("\n Level Order Traversal: ");

  levelorder();

  printf("\n Leaf Nodes: ");

  print\_leaf(tree[0]);

}

****

**USING LINKED LISTS**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct tree**

**{**

**struct tree \*lchild;**

**char data[10];**

**struct tree \*rchild;**

**};**

**typedef struct tree node;**

**node \*Q[50];**

**int node\_ctr;**

**node \*getnode()**

**{**

**node \*temp;**

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

**printf("\n Enter Data: ");**

**scanf("%s", temp->data);**

**temp->lchild = NULL;**

**temp->rchild = NULL;**

**return temp;**

**}**

**void create\_binarytree(node \*root)**

**{**

**char option;**

**node\_ctr = 1;**

**if (root != NULL)**

**{**

**printf("\n Node %s has Left SubTree(Y/N)", root->data);**

**scanf("%c", &option);**

**if (option == 'Y' || option == 'y')**

**{**

**root->lchild = getnode();**

**node\_ctr++;**

**create\_binarytree(root->lchild);**

**}**

**else**

**{**

**root->lchild = NULL;**

**create\_binarytree(root->lchild);**

**}**

**printf("\n Node %s has Right SubTree(Y/N) ", root->data);**

**scanf("%c", &option);**

**if (option == 'Y' || option == 'y')**

**{**

**root->rchild = getnode();**

**node\_ctr++;**

**create\_binarytree(root->rchild);**

**}**

**else**

**{**

**root->rchild = NULL;**

**create\_binarytree(root->rchild);**

**}**

**}**

**}**

**void make\_Queue(node \*root, int parent)**

**{**

**if (root != NULL)**

**{**

**node\_ctr++;**

**Q[parent] = root;**

**make\_Queue(root->lchild, parent \* 2 + 1);**

**make\_Queue(root->rchild, parent \* 2 + 2);**

**}**

**}**

**delete\_node(node \*root, int parent)**

**{**

**int index = 0;**

**if (root == NULL)**

**printf("\n Empty TREE ");**

**else**

**{**

**node\_ctr = 0;**

**make\_Queue(root, 0);**

**index = node\_ctr - 1;**

**Q[index] = NULL;**

**parent = (index - 1) / 2;**

**if (2 \* parent + 1 == index)**

**Q[parent]->lchild = NULL;**

**else**

**Q[parent]->rchild = NULL;**

**}**

**printf("\n Node Deleted ..");**

**}**

**void inorder(node \*root)**

**{**

**if (root != NULL)**

**{**

**inorder(root->lchild);**

**printf("%3s", root->data);**

**inorder(root->rchild);**

**}**

**}**

**void preorder(node \*root)**

**{**

**if (root != NULL)**

**{**

**printf("%3s", root->data);**

**preorder(root->lchild);**

**preorder(root->rchild);**

**}**

**}**

**void postorder(node \*root)**

**{**

**if (root != NULL)**

**{**

**postorder(root->lchild);**

**postorder(root->rchild);**

**printf("%3s", root->data);**

**}**

**}**

**void print\_leaf(node \*root)**

**{**

**if (root != NULL)**

**{**

**if (root->lchild == NULL && root->rchild == NULL)**

**printf("%3s ", root->data);**

**print\_leaf(root->lchild);**

**print\_leaf(root->rchild);**

**}**

**}**

**void print\_tree(node \*root, int line)**

**{**

**int i;**

**if (root != NULL)**

**{**

**print\_tree(root->rchild, line + 1);**

**printf("\n");**

**for (i = 0; i < line; i++)**

**printf(" ");**

**printf("%s", root->data);**

**print\_tree(root->lchild, line + 1);**

**}**

**}**

**void level\_order(node \*Q[], int ctr)**

**{**

**int i;**

**for (i = 0; i < ctr; i++)**

**{**

**if (Q[i] != NULL)**

**printf("%5s", Q[i]->data);**

**}**

**}**

**int menu()**

**{**

**int ch;**

**printf("\n 1. Create Binary Tree ");**

**printf("\n 2. Inorder Traversal ");**

**printf("\n 3. Preorder Traversal ");**

**printf("\n 4. Postorder Traversal ");**

**printf("\n 5. Level Order Traversal");**

**printf("\n 6. Leaf Node ");**

**printf("\n 7. Print Height of Tree ");**

**printf("\n 8. Print Binary Tree ");**

**printf("\n 9. Delete a node ");**

**printf("\n 10. Quit ");**

**printf("\n Enter Your choice: ");**

**scanf("%d", &ch);**

**return ch;**

**}**

**void main()**

**{**

**printf("\n\tSAMARPIT DUA 1900321290050 BINARY TREE THROUGH LINKED LIST");**

**int i, ch;**

**node \*root = NULL;**

**do**

**{**

**ch = menu();**

**switch (ch)**

**{**

**case 1:**

**if (root == NULL)**

**{**

**root = getnode();**

**create\_binarytree(root);**

**}**

**else**

**{**

**printf("\n Tree is already Created ..");**

**}**

**break;**

**case 2:**

**printf("\n Inorder Traversal: ");**

**inorder(root);**

**break;**

**case 3:**

**printf("\n Preorder Traversal: ");**

**preorder(root);**

**break;**

**case 4:**

**printf("\n Postorder Traversal: ");**

**postorder(root);**

**break;**

**case 5:**

**printf("\n Level Order Traversal ..");**

**make\_Queue(root, 0);**

**level\_order(Q, node\_ctr);**

**break;**

**case 6:**

**printf("\n Leaf Nodes: ");**

**print\_leaf(root);**

**break;**

**case 8:**

**printf("\n Print Tree \n");**

**print\_tree(root, 0);**

**break;**

**case 9:**

**delete\_node(root, 0);**

**break;**

**case 10:**

**exit(0);**

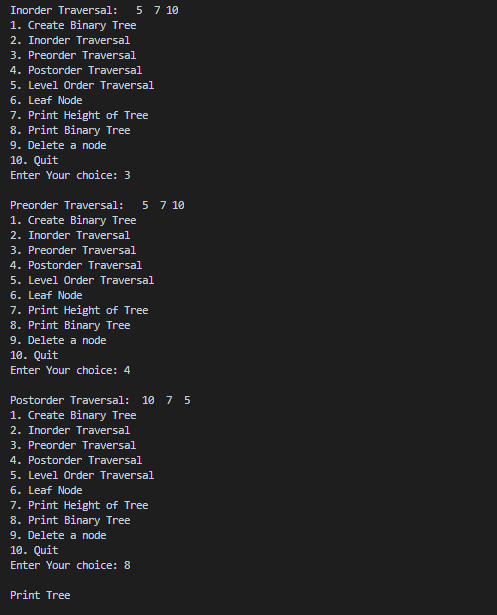
**}**

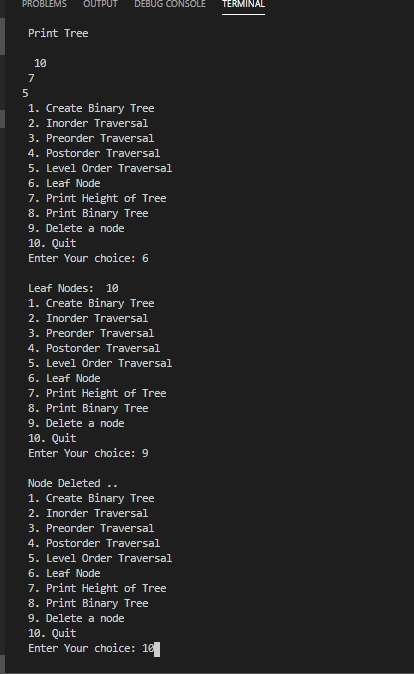
**} while (1);**

**}**

**Result /Output :**

****

****

****

**PRACTICAL-10(a)**

1. **PRACTICAL STATEMENT OF PRACTICAL: Implementation of BFS,DFS.**
2. **OBJECTIVE OF PRACTICAL : Implementation of different traversal technique BFS and DFS.**
3. **LANGUAGE REQUIRED: C Language.**
4. **Algorithm :**

BFS (G, s)

BEGIN:

QUEUE Q

Initialize (Q)

For all u ɛ V[G] DO

Π[u] ← Nil

Color[u] ← White

EnQueue (Q, s)

Color[s] ← Grey

d[s] ← 0

WHILE (!Empty(Q)) Do

u ← DeQueue (Q) Do

For each V ɛ Adj [u]

If Color[ u] = = White

Color [ u] ← Grey

EnQueue (Q, u)

d[v] ← d[u]+1

Π[u] ← u Color[u] ← Black

WRITE (u)

END;

Time Complexity: ϴ(|V|+|E|)

Space Complexity: ϴ(|V|)

**ALGORITHM DFS (G)**

BEGIN:

For all u ɛ V [G] Do

Color[u] ← White

Time ← 0

For all u ɛ V[u] Do

IF Color[u] ← White

DFS ← Visit (u)

END;

**ALGORITHM DFS-VISIT (U)**

BEGIN:

Color[u] ← Grey

S[u] ← Time+1

For all V ɛ Adj [U] Do

IF Color [V] ← White

Π[V] ← U

DFS ← Visit [V]

Color [U] ← Black

F[U] ← Time+1

Time ← Time+1

END;

Time Complexity: ϴ(|V|+|E|)

Space Complexity: ϴ(|V|)

1. **IMPLEMENTATION :**

#include <stdio.h>

int q[20], top = -1, front = -1, rear = -1, a[20][20], vis[20], stack[20];

int delete ();

void add(int item);

void bfs(int s, int n);

void dfs(int s, int n);

void push(int item);

int pop();

void main()

{

    printf("SAMARPIT DUA 1900321290050 \n BFS and DFS\n");

    int n, i, s, ch, j;

    char c, dummy;

    printf("ENTER THE NUMBER VERTICES ");

    scanf("%d", &n);

    for (i = 1; i <= n; i++)

    {

        for (j = 1; j <= n; j++)

        {

            printf("ENTER 1 IF %d HAS A NODE WITH %d ELSE 0 ", i, j);

            scanf("%d", &a[i][j]);

        }

    }

    printf("THE ADJACENCY MATRIX IS\n");

    for (i = 1; i <= n; i++)

    {

        for (j = 1; j <= n; j++)

        {

            printf(" %d", a[i][j]);

        }

        printf("\n");

    }

    do

    {

        for (i = 1; i <= n; i++)

            vis[i] = 0;

        printf("\nMENU");

        printf("\n1.B.F.S");

        printf("\n2.D.F.S");

        printf("\nENTER YOUR CHOICE");

        scanf("%d", &ch);

        printf("ENTER THE SOURCE VERTEX :");

        scanf("%d", &s);

        switch (ch)

        {

        case 1:

            bfs(s, n);

            break;

        case 2:

            dfs(s, n);

            break;

        }

        printf("\nDO U WANT TO CONTINUE(Y/N) ? ");

        scanf("%c", &dummy);

        scanf("%c", &c);

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

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*BFS(breadth-first search) code\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void bfs(int s, int n)

{

    int p, i;

    add(s);

    vis[s] = 1;

    p = delete ();

    if (p != 0)

        printf(" %d", p);

    while (p != 0)

    {

        for (i = 1; i <= n; i++)

            if ((a[p][i] != 0) && (vis[i] == 0))

            {

                add(i);

                vis[i] = 1;

            }

        p = delete ();

        if (p != 0)

            printf(" %d ", p);

    }

    for (i = 1; i <= n; i++)

        if (vis[i] == 0)

            bfs(i, n);

}

void add(int item)

{

    if (rear == 19)

        printf("QUEUE FULL");

    else

    {

        if (rear == -1)

        {

            q[++rear] = item;

            front++;

        }

        else

            q[++rear] = item;

    }

}

int delete ()

{

    int k;

    if ((front > rear) || (front == -1))

        return (0);

    else

    {

        k = q[front++];

        return (k);

    }

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DFS(depth-first search) code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void dfs(int s, int n)

{

    int i, k;

    push(s);

    vis[s] = 1;

    k = pop();

    if (k != 0)

        printf(" %d ", k);

    while (k != 0)

    {

        for (i = 1; i <= n; i++)

            if ((a[k][i] != 0) && (vis[i] == 0))

            {

                push(i);

                vis[i] = 1;

            }

        k = pop();

        if (k != 0)

            printf(" %d ", k);

    }

    for (i = 1; i <= n; i++)

        if (vis[i] == 0)

            dfs(i, n);

}

void push(int item)

{

    if (top == 19)

        printf("Stack overflow ");

    else

        stack[++top] = item;

}

int pop()

{

    int k;

    if (top == -1)

        return (0);

    else

    {

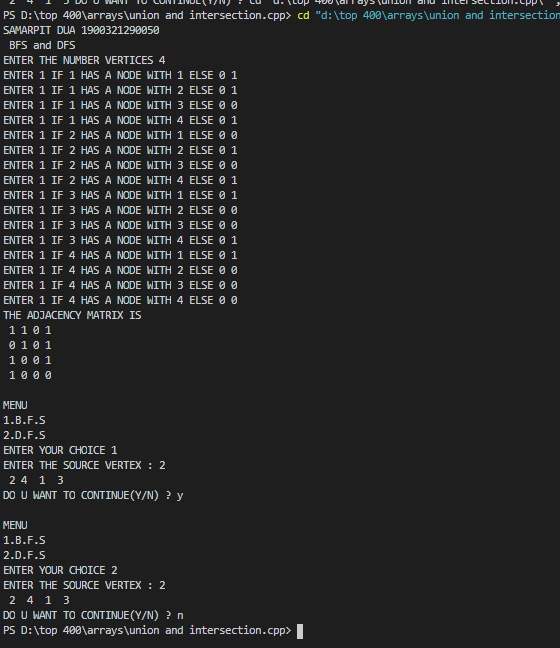
        k = stack[top--];

        return (k);

    }

}

**Result /Output :**

****

**Practical-10(b)**

1. **PRACTICAL STATEMENT OF PRACTICAL:** **Write a program in C for minimum cost spanning tree.**
2. **THEORY : Prim's Algorithm is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.Prim's algorithm starts with the single node and explore all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.**
3. **ALGORITHM / FLOW CHART :**

**1) Create a set *mstSet* that keeps track of vertices already included in MST.**

**2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.**

**3) While mstSet doesn’t include all vertices   
….a) Pick a vertex *u* which is not there in *mstSet*and has minimum key value.**

**….b) Include *u*to mstSet.**

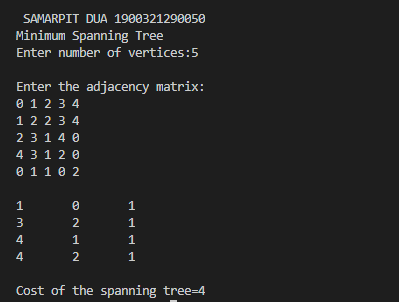
**….c) Update key value of all adjacent vertices of *u*. To update the key values, iterate through**

**All adjacent vertices. For every adjacent vertex *v*, if weight of edge *u-v* is less than the previous key value of *v*, update the key value as weight of *u-v***

**The idea of using key values is to pick the minimum weight edge from**[**cut**](http://en.wikipedia.org/wiki/Cut_(graph_theory))**. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.**

1. **SOURCE CODE:**
2. #include<stdio.h>
3. #define MAX 30
4. typedef struct edge
5. {
6. int u,v,w;
7. }edge;
8. typedef struct edgelist
9. {
10. edge data[MAX];
11. int n;
12. }edgelist;
13. edgelist elist;
14. int G[MAX][MAX],n;
15. edgelist spanlist;
16. void kruskal();
17. int find(int belongs[],int vertexno);
18. void union1(int belongs[],int c1,int c2);
19. void sort();
20. void print();
21. void main()
22. {
23. int i,j,total\_cost;
24. printf("\n SAMARPIT DUA 1900321290050 \nMinimum Spanning Tree ");
25. printf("\nEnter number of vertices:");
26. scanf("%d",&n);
27. printf("\nEnter the adjacency matrix:\n");
28. for(i=0;i<n;i++)
29. for(j=0;j<n;j++)
30. scanf("%d",&G[i][j]);
31. kruskal();
32. print();
33. }
34. void kruskal()
35. {
36. int belongs[MAX],i,j,cno1,cno2;
37. elist.n=0;
38. for(i=1;i<n;i++)
39. for(j=0;j<i;j++)
40. {
41. if(G[i][j]!=0)
42. {
43. elist.data[elist.n].u=i;
44. elist.data[elist.n].v=j;
45. elist.data[elist.n].w=G[i][j];
46. elist.n++;
47. }
48. }
49. sort();
50. for(i=0;i<n;i++)
51. belongs[i]=i;
52. spanlist.n=0;
53. for(i=0;i<elist.n;i++)
54. {
55. cno1=find(belongs,elist.data[i].u);
56. cno2=find(belongs,elist.data[i].v);
57. if(cno1!=cno2)
58. {
59. spanlist.data[spanlist.n]=elist.data[i];
60. spanlist.n=spanlist.n+1;
61. union1(belongs,cno1,cno2);
62. }
63. }
64. }
65. int find(int belongs[],int vertexno)
66. {
67. return(belongs[vertexno]);
68. }
69. void union1(int belongs[],int c1,int c2)
70. {
71. int i;
72. for(i=0;i<n;i++)
73. if(belongs[i]==c2)
74. belongs[i]=c1;
75. }
76. void sort()
77. {
78. int i,j;
79. edge temp;
80. for(i=1;i<elist.n;i++)
81. for(j=0;j<elist.n-1;j++)
82. if(elist.data[j].w>elist.data[j+1].w)
83. {
84. temp=elist.data[j];
85. elist.data[j]=elist.data[j+1];
86. elist.data[j+1]=temp;
87. }
88. }
89. void print()
90. {
91. int i,cost=0;
92. for(i=0;i<spanlist.n;i++)
93. {
94. printf("\n%d\t%d\t%d",spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);
95. cost=cost+spanlist.data[i].w;
96. }
97. printf("\n\nCost of the spanning tree=%d",cost);
98. }

**OUTPUT:**

****

**PRACTICAL -10(c)**

OBJECTIVE: Write a program in C for finding shortest path in a graph.

THEORY:One algorithm for finding the shortest path from a starting node to a target node in a weighted graph is Dijkstra’s algorithm. The algorithm creates a tree of shortest paths from the starting vertex, the source, to all other points in the graph.

ALGORITHM:

**Dijkstra’s Algorithm:**

1. Create cost matrix C[ ][ ] from adjacency matrix adj[ ][ ]. C[i][j] is the cost of going from vertex i to vertex j. If there is no edge between vertices i and j then C[i][j] is infinity.

2. Array visited[ ] is initialized to zero.

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

visited[i]=0;

3. If the vertex 0 is the source vertex then visited[0] is marked as 1.

4. Create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1 from the source vertex 0.

for(i=1;i<n;i++)

distance[i]=cost[0][i];

Initially, distance of source vertex is taken as 0. i.e. distance[0]=0;

5. for(i=1;i<n;i++)

Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark visited[w] as 1.

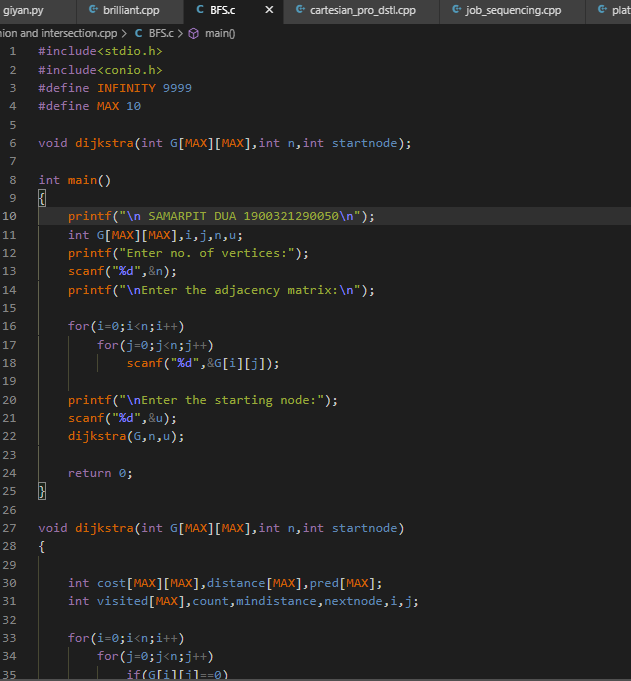
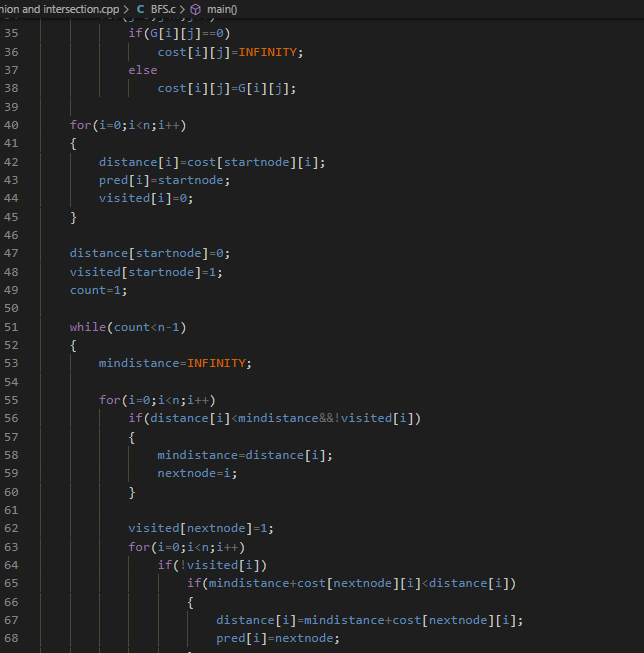
Recalculate the shortest distance of remaining vertices from the source.

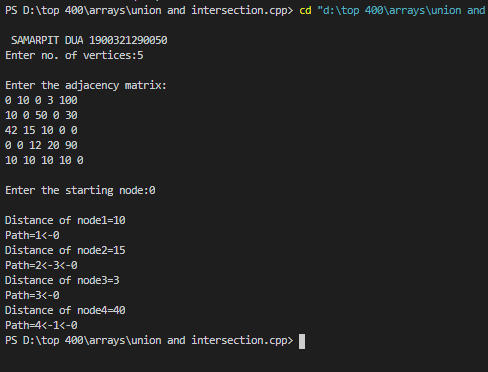
Only, the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance. i.e. for each vertex v

if(visited[v]==0)

distance[v]=min(distance[v],

distance[w]+cost[w][v])

1. **IMPLEMENTATION:**
2. ****
3. ****
4. ****
5. **Result /Output:**

****

**#include<stdio.h>**

**#include<conio.h>**

**#define INFINITY 9999**

**#define MAX 10**

**void dijkstra(int G[MAX][MAX],int n,int startnode);**

**int main()**

**{**

**int G[MAX][MAX],i,j,n,u;**

**printf("Enter no. of vertices:");**

**scanf("%d",&n);**

**printf("\nEnter the adjacency matrix:\n");**

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

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

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

**printf("\nEnter the starting node:");**

**scanf("%d",&u);**

**dijkstra(G,n,u);**

**return 0;**

**}**

**void dijkstra(int G[MAX][MAX],int n,int startnode)**

**{**

**int cost[MAX][MAX],distance[MAX],pred[MAX];**

**int visited[MAX],count,mindistance,nextnode,i,j;**

**//pred[] stores the predecessor of each node**

**//count gives the number of nodes seen so far**

**//create the cost matrix**

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

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

**if(G[i][j]==0)**

**cost[i][j]=INFINITY;**

**else**

**cost[i][j]=G[i][j];**

**//initialize pred[],distance[] and visited[]**

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

**{**

**distance[i]=cost[startnode][i];**

**pred[i]=startnode;**

**visited[i]=0;**

**}**

**distance[startnode]=0;**

**visited[startnode]=1;**

**count=1;**

**while(count<n-1)**

**{**

**mindistance=INFINITY;**

**//nextnode gives the node at minimum distance**

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

**if(distance[i]<mindistance&&!visited[i])**

**{**

**mindistance=distance[i];**

**nextnode=i;**

**}**

**//check if a better path exists through nextnode**

**visited[nextnode]=1;**

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

**if(!visited[i])**

**if(mindistance+cost[nextnode][i]<distance[i])**

**{**

**distance[i]=mindistance+cost[nextnode][i];**

**pred[i]=nextnode;**

**}**

**count++;**

**}**

**//print the path and distance of each node**

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

**if(i!=startnode)**

**{**

**printf("\nDistance of node%d=%d",i,distance[i]);**

**printf("\nPath=%d",i);**

**j=i;**

**do**

**{**

**j=pred[j];**

**printf("<-%d",j);**

**}while(j!=startnode);**

**}**

**}**

// Prim's Algorithm in C

#include<stdio.h>

#include<stdbool.h>

#define INF 9999999

// number of vertices in graph

#define V 5

// create a 2d array of size 5x5

//for adjacency matrix to represent graph

int G[V][V] = {

{0, 9, 75, 0, 0},

{9, 0, 95, 19, 42},

{75, 95, 0, 51, 66},

{0, 19, 51, 0, 31},

{0, 42, 66, 31, 0}};

int main() {

int no\_edge; // number of edge

// create a array to track selected vertex

// selected will become true otherwise false

int selected[V];

// set selected false initially

memset(selected, false, sizeof(selected));

// set number of edge to 0

no\_edge = 0;

// the number of egde in minimum spanning tree will be

// always less than (V -1), where V is number of vertices in

//graph

// choose 0th vertex and make it true

selected[0] = true;

int x; // row number

int y; // col number

// print for edge and weight

printf("Edge : Weight\n");

while (no\_edge < V - 1) {

//For every vertex in the set S, find the all adjacent vertices

// , calculate the distance from the vertex selected at step 1.

// if the vertex is already in the set S, discard it otherwise

//choose another vertex nearest to selected vertex at step 1.

int min = INF;

x = 0;

y = 0;

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

if (selected[i]) {

for (int j = 0; j < V; j++) {

if (!selected[j] && G[i][j]) { // not in selected and there is an edge

if (min > G[i][j]) {

min = G[i][j];

x = i;

y = j;

}

}

}

}

}

printf("%d - %d : %d\n", x, y, G[x][y]);

selected[y] = true;

no\_edge++;

}

return 0;

}

**// Iterative C program to reverse a linked list**

**#include <stdio.h>**

**#include <stdlib.h>**

**/\* Link list node \*/**

**struct Node {**

**int data;**

**struct Node\* next;**

**};**

**/\* Function to reverse the linked list \*/**

**static void reverse(struct Node\*\* head\_ref)**

**{**

**struct Node\* prev = NULL;**

**struct Node\* current = \*head\_ref;**

**struct Node\* next = NULL;**

**while (current != NULL) {**

**// Store next**

**next = current->next;**

**// Reverse current node's pointer**

**current->next = prev;**

**// Move pointers one position ahead.**

**prev = current;**

**current = next;**

**}**

**\*head\_ref = prev;**

**}**

**/\* Function to push a node \*/**

**void push(struct Node\*\* head\_ref, int new\_data)**

**{**

**struct Node\* new\_node**

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

**new\_node->data = new\_data;**

**new\_node->next = (\*head\_ref);**

**(\*head\_ref) = new\_node;**

**}**

**/\* Function to print linked list \*/**

**void printList(struct Node\* head)**

**{**

**struct Node\* temp = head;**

**while (temp != NULL) {**

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

**temp = temp->next;**

**}**

**}**

**/\* Driver code\*/**

**int main()**

**{**

**/\* Start with the empty list \*/**

**struct Node\* head = NULL;**

**push(&head, 20);**

**push(&head, 4);**

**push(&head, 15);**

**push(&head, 85);**

**printf("Given linked list\n");**

**printList(head);**

**reverse(&head);**

**printf("\nReversed Linked list \n");**

**printList(head);**

**getchar();**

**}**

#include <stdio.h>

int main()  
{  
   char s[1000], r[1000];  
   int begin, end, count = 0;

   printf("Input a string**\n**");  
   gets(s);

*// Calculating string length*

   while (s[count] != '**\0**')  
      count++;

   end = count - 1;

   for (begin = 0; begin < count; begin++) {  
      r[begin] = s[end];  
      end--;  
   }

   r[begin] = '**\0**';

   printf("%s**\n**", r);

   return 0;  
}