# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Laboratory Manual

DATA STRUCTURES

PROGRAMMING LAB MANUAL

FOR

Second year Students (CSE)

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## Vision:

To achieve Academic Excellence in Computer Science and Engineering by imparting knowledge to the students towards fulfilling the ever changing Industrial demands and Societal needs

## Mission:

**M1:** To produce professionally competitive Computer Science Engineers with Excellent skill set. **M2:** To fulfill Global demands, State of the Art Laboratories and Research facilities are developed to impart knowledge based education.

**M3:** Through Career Development Training, skills required for Employability and Societal needs are developed.

## Program Specific Outcomes:

**PSO1:Professional Skills:** The ability to understand, analyze and develop computer programs in the areas related to Algorithms, System Software, Multimedia, Web design, Big Data Analytics, and networking for efficient design of computer-based systems of varying complexity.

**PSO2: Problem-Solving Skills:** The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.

**PSO3: Successful Career:** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and a zest for higher studies.

## Program Educational Objectives:

**PEO 1:** To Produce Graduates with a Strong foundation in Mathematics, Science and Computer Engineering fundamentals to solve Engineering problems and also pursue Higher studies.

**PEO 2:** Graduates with Ability to Analyze, Design and Synthesize Data, Create Novel Products to satisfy Industry needs.

**PEO 3:** Ability to Understand and Analyze Engineering issues in a broader perspective with Ethical responsibility towards Sustainable Development and Societal needs.

**PEO 4:** Graduates with Managerial, Soft Skills, Entrepreneurship and Leadership Qualities in order to be Competent Professional.

## Programme Outcomes:

After completion of the Computer Science and Engineering program students will have:

**PO1: Engineering Knowledge:** Apply knowledge of mathematics and science, with fundamentals of Computer Science & Engineering to be able to solve complex engineering problems related to CSE.

**PO2: Problem Analysis:** Identify, formulate, review research literature and analyze complex engineering problems related to CSE and reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.

**PO3: Design/Development of Solutions:** Design solutions for complex engineering problems related to CSE and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety and the cultural societal and environmental considerations.

**PO4: Conduct Investigations of Complex Problems:** Use research–based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**PO5: Modern Tool Usage:** Create, select and apply appropriate techniques, resources and modern Engineering and IT tools including prediction and modeling to computer science related complex Engineering activities with an understanding of the limitations.

**PO6: The Engineer and Society:** Apply Reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the CSE professional engineering practice.

**PO7: Environment and Sustainability:** Understand the impact of the CSE professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of, and need for sustainable development.

**PO8: Ethics:** Apply Ethical Principles and Commit to professional ethics and responsibilities and norms of the engineering practice.

**PO9: Individual and Team Work:** Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary settings.

**PO10: Communication:** Communicate effectively on complex engineering activities with the engineering community and society at large such as , able to comprehend, write effective reports, design documentation, and to make effective presentations .

**PO11: Project Management and Finance:** Demonstrate knowledge, understanding of the engineering management principles and apply these to one’s own work, as a member and leader in a team to manage projects and in multi disciplinary environments.

**PO12: Life-long Learning:** Recognize the need for preparation and ability to engage in independent, lifelong learning the broadest context of technological change.

## DOs and DON’Ts in Laboratory:

1. Make entry in the Log Book as soon as you enter the Laboratory.
2. All the students should sit according to their roll numbers starting from their left to right.
3. All the students are supposed to enter the terminal number in the log book.
4. Do not change the terminal on which you are working.
5. All the students are expected to get at least the algorithm of the program/concept to be implement.
6. Strictly follow the instructions given by the teacher/Lab Instructor.

## Name of the Course: Course Code:

**Regulation: Academic Year: Year/ Semester:**

1. **PRE-REQUISITES:**
2. **COURSE OBJECTIVES: The students will be able to learn**
3. **COURSE OUTCOMES: At the end of the course, students will be able to**
4. **List of Programs**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Programs to be Covered** | **Mapping CO’s** |
| **1** |  |  |
| **2** |  |  |
| **3** |  |  |
| **4** |  |  |
| **5** |  |  |
| **6** |  |  |
| **7** |  |  |
| **8** |  |  |
| **9** |  |  |
| **10** |  |  |
| **11** |  |  |
| **12** |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |

## I. Exercise programs on basic control structures & loops

a) Write **a recursive program to find the factorial of number.**

**AIM: To write a recursive program to find the factorial of number.**

**ALGORTIHM:**

**Step 1: Start**

**Step 2: Read number n**

**Step 3: Call factorial(n)**

**Step 4: Print factorial f**

**Step 5: Stop**

**factorial(n)**

**Step 1: If n==1 then return 1**

**Step 2: Else**

**f=n\*factorial(n-1)**

**Step 3: Return f**

**PROGRAM:**

**#include<stdio.h>**

**#include<stdlib.h>**

**int rfact(int n);**

**int main()**

**{**

**int n,r;**

**printf(“\n Enter a number :\n”);**

**scanf(“\n %d”,&n);**

**r=rfact(n);**

**printf(“The factorial of a given number is %d\n”,r);**

**return 0;**

**}**

**int rfact(int n)  
{**

**if(n>=1)**

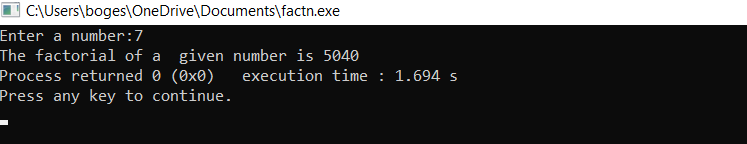
**return n\*fact(n-1);**

**else**

**return 1;**

**}**

**OUTPUT:**



**b.)** **Write a recursive C program to display the Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, …N.**

**Aim: To Write a recursive C program to display the Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, …N**

**PROGRAM:**

**#include<stdio.h>**

**int fib(int n)**

**{**

**if (n==0||n==1)**

**return n;**

**else**

**return fib(n-1) + fib(n-2);**

**}**

**int main ()**

**{**

**int n ;**

**printf(“Enter a numbers:”);**

**scanf(“%d\n”,&n);**

**printf(“The fibanocci series are\n”);**

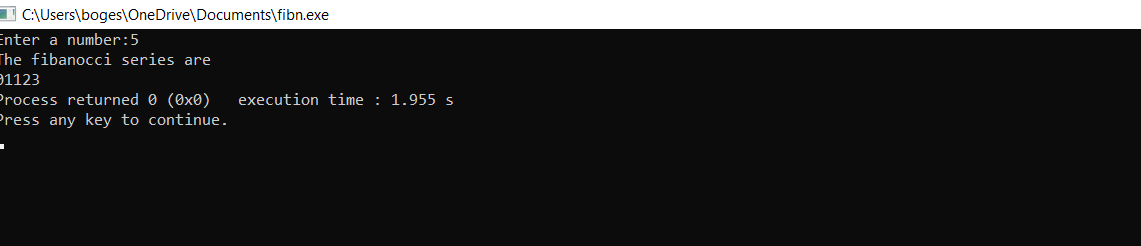
**for(int i=0; i<n;i++)**

**printf("%d", fib(i));**

**return 0;**

}

**OUTPUT:**



**C.)** **Write a recursive C program to calculate the GCD of two numbers.**

**AIM: To find the gcd of two numbers using recursion**

**PROGRAM:**

**int GCD(int n1, int n2);**

**int main() {**

**int n1, n2;**

**printf("Enter two positive integers: ");**

**scanf("%d %d", &n1, &n2);**

**printf("G.C.D of %d and %d is %d.", n1, n2, GCD(n1, n2));**

**return 0;**

**}**

**int GCD(int n1, int n2)**

**{**

**if (n2 != 0)**

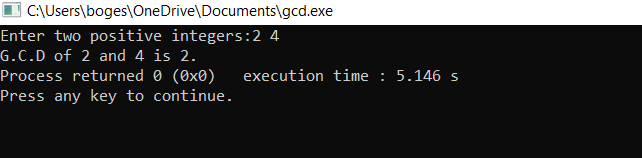
**return GCD(n2, n1 % n2);**

**else**

**return n1;**

**}**

**OUTPUT:**



**d.)** **Write a recursive C program for Towers of Hanoi: N disks are to be transferred from peg S to peg D with Peg I as the intermediate peg.**

**AIM : TO Write a recursive C program for Towers of Hanoi: N disks are to be transferred from peg S to peg D with Peg I as the intermediate peg.**

**PROGRAM:**

**#include <stdio.h>**

**void towershanoi(int, char, char, char);**

**int main()**

**{**

**int num;**

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

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

**printf("The sequence of moves involved in the Tower of Hanoi are :\n");**

**towershanoi(num, 'A', 'C', 'B');**

**return 0;**

**}**

**void towershanoi(int num, char fromrod, char torod, char auxrod)**

**{**

**if (num == 1)**

**{**

**printf("\n Move disk 1 from rod %c to rod %c", fromrod, torod);**

**return;**

**}**

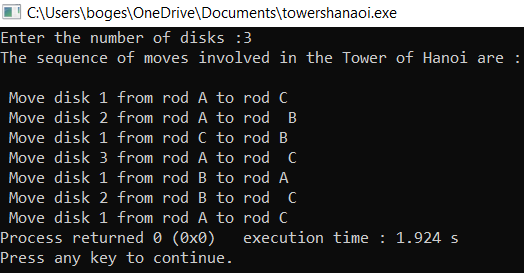
**towershanoi(num - 1, fromrod, auxrod, torod);**

**printf("\n Move disk %d from rod %c to rod %c", num, fromrod, torod);**

**towershanoi(num - 1, auxrod, torod, fromrod);**

**}**

**OUTPUT:**



**Exercise2-**

1. **Write a recursive and non-recursive C program to implement Linear Search technique.**

**AIM:TO** **Write a recursive and non-recursive C program to implement Linear Search technique.**

**PROGRAM:**

**#include <stdio.h>**

**int RecursiveLS(int arr[], int value, int index, int n)**

**{**

**int pos = 0;**

**if(index >= n)**

**{**

**return 0;**

**}**

**else if (arr[index] == value)**

**{**

**pos = index + 1;**

**return pos;**

**}**

**else**

**{**

**return RecursiveLS(arr, value, index+1, n);**

**}**

**return pos;**

**}**

**int main()**

**{**

**int n, value, pos, m = 0, arr[100];**

**printf("Enter the total elements in the array ");**

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

**printf("Enter the array elements\n");**

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

**{**

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

**}**

**printf("Enter the element to search ");**

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

**pos = RecursiveLS(arr, value, 0, n);**

**if (pos != 0)**

**{**

**printf("Element found at pos %d ", pos);**

**}**

**else**

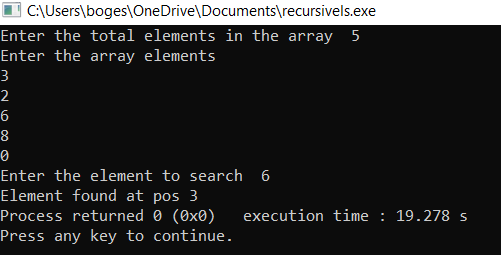
**{**

**printf("Element not found");**

**} return 0;**

**}**

**OUTPUT:**



**WITHOUT RECURSION:**

**#include<stdio.h>**

**int main()**

**{**

**int array[100], search, c, n;**

**printf("Enter number of elements in array\n");**

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

**printf("Enter %d integer(s)\n", n);**

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

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

**printf("Enter a number to search\n");**

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

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

**{**

**if (array[c] == search)**

**{**

**printf("%d is present at location %d.\n", search, c+1);**

**break;**

**}**

**}**

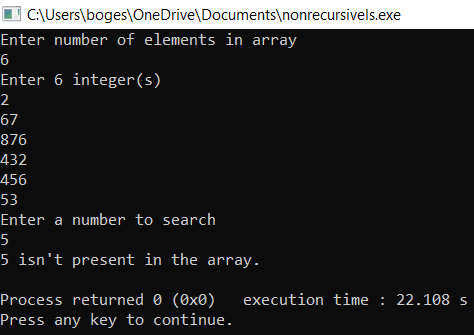
**if (c == n)**

**printf("%d isn't present in the array.\n", search);**

**return 0;**

**}**

**OUTPUT:**



**2.** **Write** **a recursive and non-recursive C program to implement Binary Search technique**

**AIM: To write a recursive and non-recursive C program to implement Binary Search technique**

**USING RECURSION:**

**#include<stdio.h>**

**int bs(int arr[],int l,int h,int x)**

**{**

**if(h>=1)**

**{**

**int mid=1+(h-1)/2;**

**if(arr[mid]==x)**

**return mid;**

**if(arr[mid]>x)**

**return bs(arr,l,mid-1,x);**

**return bs(arr,mid+1,h,x);**

**}**

**return -1;**

**}**

**int main(void)**

**{**

**int arr[100],n,i,x;**

**printf("enter no.of elements :");**

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

**printf("enter array elaments:");**

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

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

**printf("enter search element :");**

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

**int res=bs(arr,0,n-1,x);**

**if(res==-1)**

**printf("element is not found");**

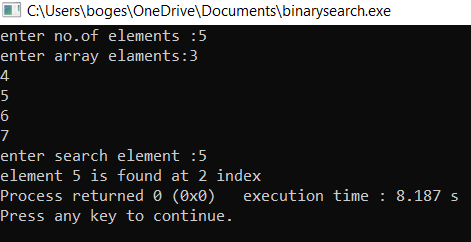
**else**

**printf("element %d is found at %d index ",x,res);**

**return 0;**

**}**

**OUTPUT :**



**WITHOUT RECURSION**

**#include<stdio.h>**

**int bs(int arr[],int l,int h,int x)**

**{**

**while(l<=h)**

**{**

**int mid=(h+l)/2;**

**if(arr[mid]==x)**

**return mid;**

**if(arr[mid]<x)**

**l=mid+1;**

**else**

**h=mid-1;**

**}**

**return -1;**

**}**

**int main(void)**

**{**

**int arr[100],n,i,x;**

**printf("enter no.of elements :");**

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

**printf("enter array elaments:");**

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

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

**printf("enter search element :");**

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

**int res=bs(arr,0,n-1,x);**

**if(res==-1)**

**printf("element is not found");**

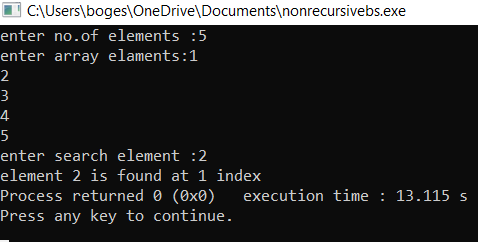
**else**

**printf("element %d is found at %d index ",x,res);**

**return 0;**

**}**

**OUTPUT:**



**Exercise-3:**

1. **Write C program that implement Insertion sort, to sort elements in an ascending order.**

**AIM:TO Write C program that implement Insertion sort, to sort elements in an ascending order.**

**PROGRAM:**

**#include <math.h>**

**#include <stdio.h>**

**void insertionSort(int arr[], int n)**

**{**

**int i, key, j;**

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

**key = arr[i];**

**j = i - 1;**

**while (j >= 0 && arr[j] > key)**

**{**

**arr[j + 1] = arr[j];**

**j = j - 1;**

**}**

**arr[j + 1] = key;**

**}**

**}**

**void printArray(int arr[], int n)**

**{**

**int i;**

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

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

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

**}**

**int main()**

**{**

**int arr[10] ,i,n;**

**printf("enter size of an array:\n");**

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

**printf("enter the elements into array:\n");**

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

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

**printf("Given array is \n");**

**printArray(arr, n);**

**insertionSort(arr, n);**

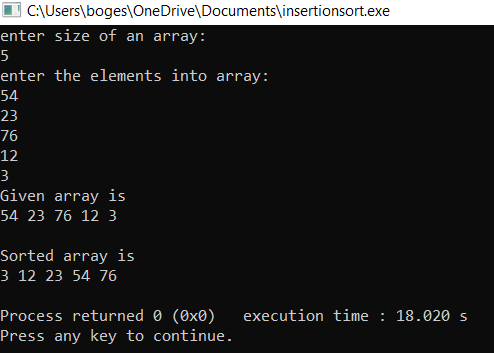
**printf("\nSorted array is \n");**

**printArray(arr, n);**

**return 0;**

**}**

**OUTPUT:**

**.**

**b. Write C program that implement Merge sort, to sort elements in an ascending order.**

**AIM: To b. Write C program that implement Merge sort, to sort elements in an ascending order.**

**PROGRAM:**

**#include <stdio.h>**

**#include <stdlib.h>**

**void merge(int arr[], int l, int m, int r)**

**{**

**int i, j, k;**

**int n1 = m - l + 1;**

**int n2 = r - m;**

**int L[n1], R[n2];**

**for (i = 0; i < n1; i++)**

**L[i] = arr[l + i];**

**for (j = 0; j < n2; j++)**

**R[j] = arr[m + 1 + j];**

**i = 0;**

**j = 0;**

**k = l;**

**while (i < n1 && j < n2)**

**{**

**if (L[i] <= R[j])**

**{**

**arr[k] = L[i];**

**i++;**

**}**

**else**

**{**

**arr[k] = R[j];**

**j++;**

**}**

**k++;**

**}**

**while (i < n1)**

**{**

**arr[k] = L[i];**

**i++;**

**k++;**

**}**

**while (j < n2)**

**{**

**arr[k] = R[j];**

**j++;**

**k++;**

**}**

**}**

**void mergeSort(int arr[], int l, int h)**

**{**

**if (l < h)**

**{**

**int m = (l + h)/ 2;**

**mergeSort(arr, l, m);**

**mergeSort(arr, m + 1, h);**

**merge(arr, l, m, h);**

**}**

**}**

**void printArray(int A[], int size)**

**{**

**int i;**

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

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

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

**}**

**int main()**

**{**

**int arr[10],i,n;**

**printf("enter the size of an array:\n");**

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

**printf("enter the elements into array:\n");**

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

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

**printf("Given array is :\n");**

**printArray(arr, n);**

**mergeSort(arr, 0, n - 1);**

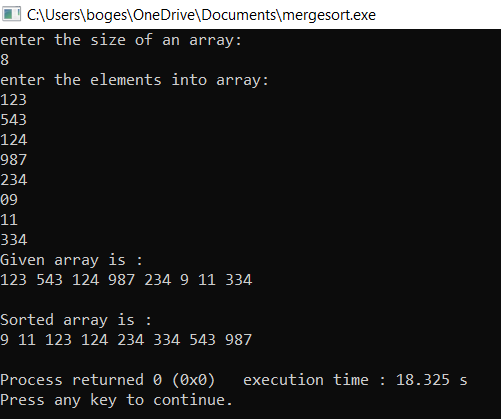
**printf("\nSorted array is :\n");**

**printArray(arr, n);**

**return 0;**

**}**

**OUTPUT:**



**c.)Write C program that implement Quick sort, to sort elements in an ascending order.**

**AIM:To Write C program that implement Quick sort, to sort elements in an ascending order**

**PROGRAM:**

**#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("How many elements are u going to enter?: ");**

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

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

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

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

**quicksort(number,0,count-1);**

**printf("Order of Sorted elements: ");**

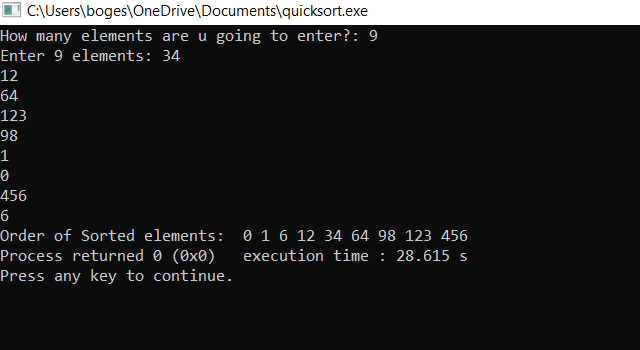
**for(i=0;i<count;i++)**

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

**return 0;**

**}**

**OUTPUT:**



**Exercise-4**

**a.)** **Write a C program to insert a node in a Single Linked List.**

**b.) Write a C program to delete a node in a Single Linked List.**

**AIM: To Write a C program to insert a node and to delete a node in a single linked list.**

**PROGRAM:**

**#include<stdio.h>**

**#include<conio.h>**

**#include<process.h>**

**struct node**

**{**

**int data;**

**struct node \*next;**

**}\*start=NULL,\*q,\*t;**

**int main()**

**{**

**int ch;**

**void insert\_beg();**

**void insert\_end();**

**int insert\_pos();**

**void display();**

**void delete\_beg();**

**void delete\_end();**

**int delete\_pos();**

**while(1)**

**{**

**printf("\n\n---- Singly Linked List(SLL) Menu ----");**

**printf("\n1.Insert\n2.Display\n3.Delete\n4.Exit\n\n");**

**printf("Enter your choice(1-4):");**

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

**switch(ch)**

**{**

**case 1:**

**printf("\n---- Insert Menu ----");**

**printf("\n1.Insert at beginning\n2.Insert at end\n3.Insert at specified position\n4.Exit");**

**printf("\n\nEnter your choice(1-4):");**

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

**switch(ch)**

**{**

**case 1: insert\_beg();**

**break;**

**case 2: insert\_end();**

**break;**

**case 3: insert\_pos();**

**break;**

**case 4: exit(0);**

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

**}**

**break;**

**case 2: display();**

**break;**

**case 3: printf("\n---- Delete Menu ----");**

**printf("\n1.Delete from beginning\n2.Delete from end\n3.Delete from specified position\n4.Exit");**

**printf("\n\nEnter your choice(1-4):");**

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

**switch(ch)**

**{**

**case 1: delete\_beg();**

**break;**

**case 2: delete\_end();**

**break;**

**case 3: delete\_pos();**

**break;**

**case 4: exit(0);**

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

**}**

**break;**

**case 4: exit(0);**

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

**}**

**}**

**return 0;**

**}**

**void insert\_beg()**

**{**

**int num;**

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

**printf("Enter data:");**

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

**t->data=num;**

**if(start==NULL) //If list is empty**

**{**

**t->next=NULL;**

**start=t;**

**}**

**else**

**{**

**t->next=start;**

**start=t;**

**}**

**}**

**void insert\_end()**

**{**

**int num;**

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

**printf("Enter data:");**

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

**t->data=num;**

**t->next=NULL;**

**if(start==NULL) //If list is empty**

**{**

**start=t;**

**}**

**else**

**{**

**q=start;**

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

**q=q->next;**

**q->next=t;**

**}**

**}**

**int insert\_pos()**

**{**

**int pos,i,num;**

**if(start==NULL)**

**{**

**printf("List is empty!!");**

**return 0;**

**}**

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

**printf("Enter data:");**

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

**printf("Enter position to insert:");**

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

**t->data=num;**

**q=start;**

**for(i=1;i<pos-1;i++)**

**{**

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

**{**

**printf("There are less elements!!");**

**return 0;**

**}**

**q=q->next;**

**}**

**t->next=q->next;**

**q->next=t;**

**return 0;**

**}**

**void display()**

**{**

**if(start==NULL)**

**{**

**printf("List is empty!!");**

**}**

**else**

**{**

**q=start;**

**printf("The linked list is:\n");**

**while(q!=NULL)**

**{**

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

**q=q->next;**

**}**

**}**

**}**

**void delete\_beg()**

**{**

**if(start==NULL)**

**{**

**printf("The list is empty!!");**

**}**

**else**

**{**

**q=start;**

**start=start->next;**

**printf("Deleted element is %d",q->data);**

**free(q);**

**}**

**}**

**void delete\_end()**

**{**

**if(start==NULL)**

**{**

**printf("The list is empty!!");**

**}**

**else**

**{**

**q=start;**

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

**q=q->next;**

**t=q->next;**

**q->next=NULL;**

**printf("Deleted element is %d",t->data);**

**free(t);**

**}**

**}**

**int delete\_pos()**

**{**

**int pos,i;**

**if(start==NULL)**

**{**

**printf("List is empty!!");**

**return 0;**

**}**

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

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

**q=start;**

**for(i=1;i<pos-1;i++)**

**{**

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

**{**

**printf("There are less elements!!");**

**return 0;**

**}**

**q=q->next;**

**}**

**t=q->next;**

**q->next=t->next;**

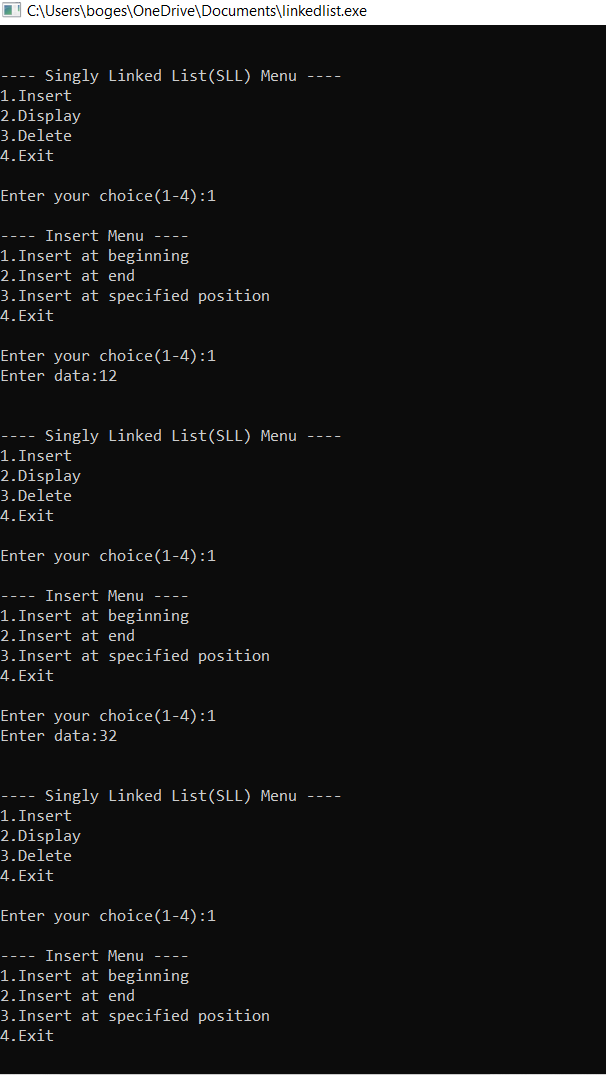
**printf("Deleted element is %d",t->data);**

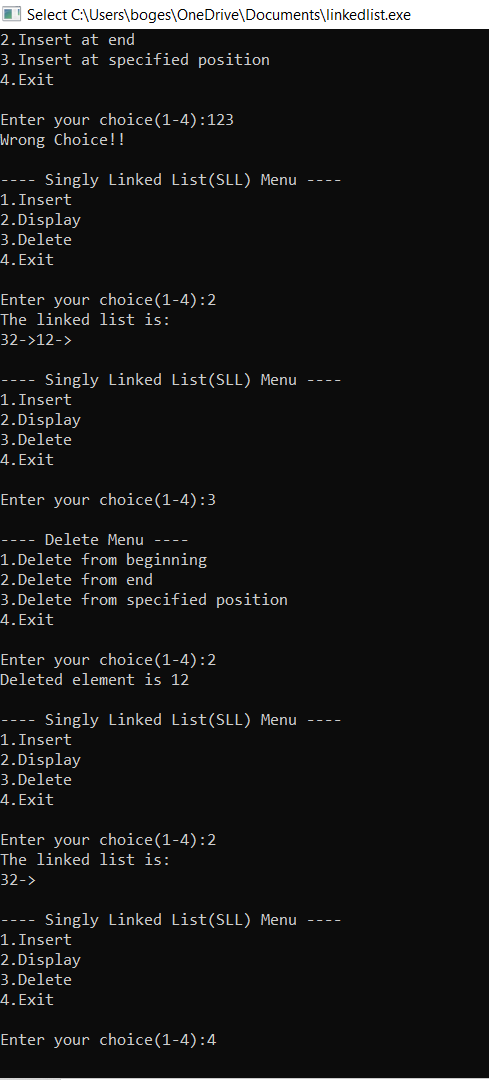
**free(t);**

**return 0;**

**}**

**OUTPUT:**





**c.)Write a C program to reverse elements in a Single Linked List.**

**AIM: To Write a C program to reverse elements in a Single Linked List.**

**PROGRAM:**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct node**

**{**

**int num;**

**struct node \*next;**

**};**

**void create(struct node \*\*);**

**void reverse(struct node \*\*);**

**void release(struct node \*\*);**

**void display(struct node \*);**

**int main()**

**{**

**struct node \*p = NULL;**

**int n;**

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

**create(&p);**

**printf("Displaying the nodes in the list:\n");**

**display(p);**

**printf("Reversing the list...\n");**

**reverse(&p);**

**printf("Displaying the reversed list:\n");**

**display(p);**

**release(&p);**

**return 0;**

**}**

**void reverse(struct node \*\*head)**

**{**

**struct node \*p, \*q, \*r;**

**p = q = r = \*head;**

**p = p->next->next;**

**q = q->next;**

**r->next = NULL;**

**q->next = r;**

**while (p != NULL)**

**{**

**r = q;**

**q = p;**

**p = p->next;**

**q->next = r;**

**}**

**\*head = q;**

**}**

**void create(struct node \*\*head)**

**{**

**int c, ch;**

**struct node \*temp, \*rear;**

**do**

**{**

**printf("Enter number: ");**

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

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

**temp->num = c;**

**temp->next = NULL;**

**if (\*head == NULL)**

**{**

**\*head = temp;**

**}**

**else**

**{**

**rear->next = temp;**

**}**

**rear = temp;**

**printf("Do you wish to continue [1/0]: ");**

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

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

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

**}**

**void display(struct node \*p)**

**{**

**while (p != NULL)**

**{**

**printf("%d\t", p->num);**

**p = p->next;**

**}**

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

**}**

**void release(struct node \*\*head)**

**{**

**struct node \*temp = \*head;**

**\*head = (\*head)->next;**

**while ((\*head) != NULL)**

**{**

**free(temp);**

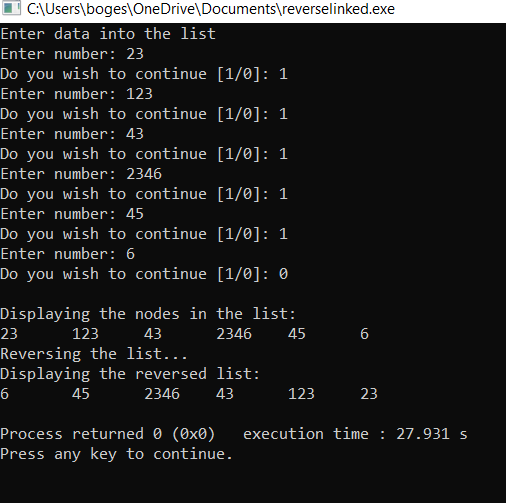
**temp = \*head;**

**(\*head) = (\*head)->next;**

**}**

**}**

**OUTPUT:**



**d. Write a C program to insert a node in a Doubly Linked List.**

**AIM: To Write a C program to insert a node in a Doubly Linked List**

**PROGRAM:**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct node {**

**int data;**

**struct node \* prev;**

**struct node \* next;**

**}\*head, \*last;**

**void createList(int n);**

**void displayList();**

**void insertAtBeginning(int data);**

**int main()**

**{**

**int n, data, choice=1;**

**head = NULL;**

**last = NULL;**

**printf("\nEnter the total number of nodes in list : ");**

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

**createList(n);**

**displayList();**

**printf("\n\nEnter data of first node : ");**

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

**insertAtBeginning(data);**

**displayList();**

**return 0;**

**}**

**void createList(int n)**

**{**

**int i, data;**

**struct node \*newNode;**

**if(n >= 1)**

**{**

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

**printf("\nEnter data of node 1 : ");**

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

**head->data = data;**

**head->prev = NULL;**

**head->next = NULL;**

**last = head;**

**for(i=2; i<=n; i++)**

**{**

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

**printf("\nEnter data of node %d : ", i);**

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

**newNode->data = data;**

**newNode->prev = last;**

**newNode->next = NULL;**

**last->next = newNode;**

**last = newNode;**

**}}}**

**void displayList()**

**{**

**struct node \* temp;**

**int n = 1;**

**if(head == NULL)**

**{**

**printf("\nList is empty.\n");**

**}**

**else**

**{**

**temp = head;**

**printf("\nTHE DOUBLY LINKED LIST IS :\n\n");**

**// Print the list**

**while(temp != NULL)**

**{**

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

**n++;**

**/\* Move the current pointer to next node \*/**

**temp = temp->next;**

**}}}**

**void insertAtBeginning(int data)**

**{**

**struct node \* newNode;**

**if(head == NULL)**

**{**

**printf("Error, List is Empty!\n");**

**}**

**else**

**{**

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

**newNode->data = data;**

**newNode->next = head;**

**newNode->prev = NULL;**

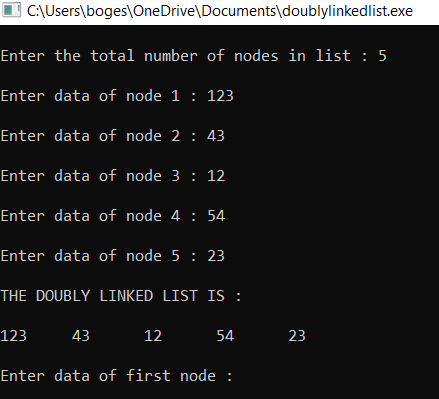
**head->prev = newNode;**

**head = newNode;**

**}**

**}**

**OUTPUT:**



**Exercise-5:**

1. **Write C program that implement Stack (its operations) using arrays.**

**AIM: To Write C program that implement Stack (its operations) using arrays**

**PROGRAM:**

**#include<stdio.h>**

**int stack[100],choice,n,top,x,i;**

**void push(void);**

**void pop(void);**

**void display(void);**

**int main()**

**{**

**top=-1;**

**printf("\n Enter the size of STACK:");**

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

**printf("\n\t STACK OPERATIONS USING ARRAY");**

**printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");**

**do**

**{**

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

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

**switch(choice)**

**{**

**case 1:**

**{**

**push();**

**break;**

**}**

**case 2:**

**{**

**pop();**

**break;**

**}**

**case 3:**

**{**

**display();**

**break;**

**}**

**case 4:**

**{**

**printf("\n\t EXIT POINT ");**

**break;**

**}**

**default:**

**{**

**printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");**

**}**

**}**

**}**

**while(choice!=4);**

**return 0;**

**}**

**void push()**

**{**

**if(top>=n-1)**

**{**

**printf("\n\tSTACK is over flow");**

**}**

**else**

**{**

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

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

**top++;**

**stack[top]=x;**

**}**

**}**

**void pop()**

**{**

**if(top<=-1)**

**{**

**printf("\n\t Stack is under flow");**

**}**

**else**

**{**

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

**top--;**

**}**

**}**

**void display()**

**{**

**if(top>=0)**

**{**

**printf("\n The elements in STACK \n");**

**for(i=top; i>=0; i--)**

**printf("\n%d",stack[i]);**

**printf("\n Press Next Choice");**

**}**

**else**

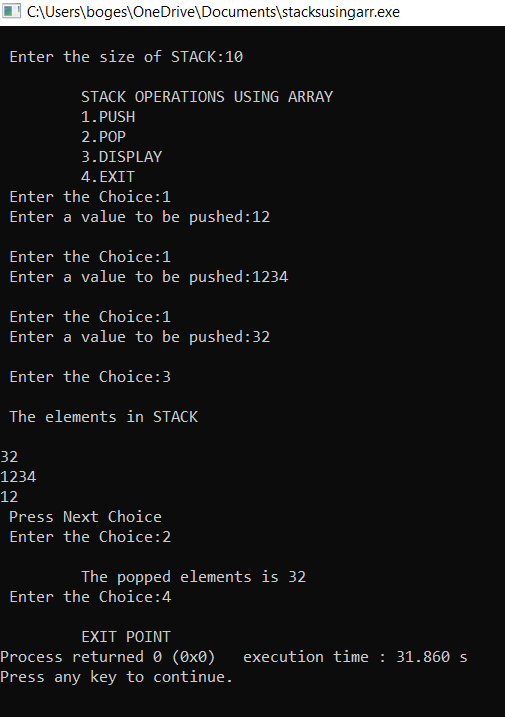
**{**

**printf("\n The STACK is empty");**

**}**

**}**

**OUTPUT:**



1. **Write C program that implement Queue (its operations) using arrays.**

**AIM: TO Write C program that implement Queue (its operations) using arrays**

**PROGRAM:**

**#include<stdio.h>**

**#define n 5**

**int main()**

**{**

**int queue[n],ch=1,front=0,rear=0,i,j=1,x=n;**

**printf("Queue using Array");**

**printf("\n1.Insertion \n2.Deletion \n3.Display \n4.Exit");**

**while(ch)**

**{**

**printf("\nEnter the Choice:");**

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

**switch(ch)**

**{**

**case 1:**

**if(rear==x)**

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

**else**

**{**

**printf("\n Enter no %d:",j++);**

**scanf("%d",&queue[rear++]);**

**}**

**break;**

**case 2:**

**if(front==rear)**

**{**

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

**}**

**else**

**{**

**printf("\n Deleted Element is %d",queue[front++]);**

**x++;**

**}**

**break;**

**case 3:**

**printf("\nQueue Elements are:\n ");**

**if(front==rear)**

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

**else**

**{**

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

**{**

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

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

**}**

**break;**

**case 4:**

**exit(0);**

**default:**

**printf("Wrong Choice: please see the options");**

**}**

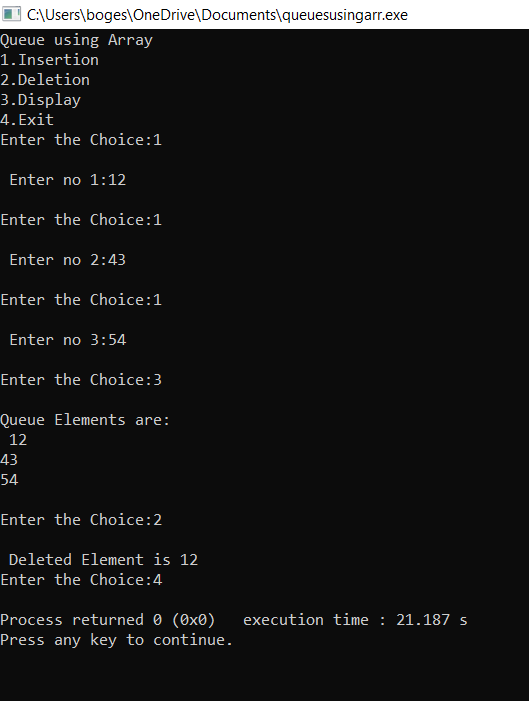
**}**

**}**

**return 0;**

**}**

**OUTPUT:**



1. **Write C program that implement Queue using Two Stacks.**

**AIM: To Write C program that implement Queue using Two Stacks**

**PROGRAM:**

**#include <stdio.h>**

**#include <stdlib.h>**

**void push1(int);**

**void push2(int);**

**int pop1();**

**int pop2();**

**void enqueue();**

**void dequeue();**

**void display();**

**void create();**

**int st1[100], st2[100];**

**int top1 = -1, top2 = -1;**

**int count = 0;**

**void main()**

**{**

**int ch;**

**printf("\n1 - Enqueue element into queue");**

**printf("\n2 - Dequeu element from queue");**

**printf("\n3 - Display from queue");**

**printf("\n4 - Exit");**

**create();**

**while (1)**

**{**

**printf("\nEnter choice");**

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

**switch (ch)**

**{**

**case 1:**

**enqueue();**

**break;**

**case 2:**

**dequeue();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**exit(0);**

**default:**

**printf("Wrong choice");**

**}**

**}**

**}**

**void create()**

**{**

**top1 = top2 = -1;**

**}**

**void push1(int data)**

**{**

**st1[++top1] = data;**

**}**

**int pop1()**

**{**

**return(st1[top1--]);**

**}**

**void push2(int data)**

**{**

**st2[++top2] = data;**

**}**

**int pop2()**

**{**

**return(st2[top2--]);**

**}**

**void enqueue()**

**{**

**int data, i;**

**printf("Enter data into queue");**

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

**push1(data);**

**count++;**

**}**

**void dequeue()**

**{**

**int i;**

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

**{**

**push2(pop1());**

**}**

**pop2();**

**count--;**

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

**{**

**push1(pop2());**

**}**

**}**

**void display()**

**{**

**int i;**

**for (i = 0;i <= top1;i++)**

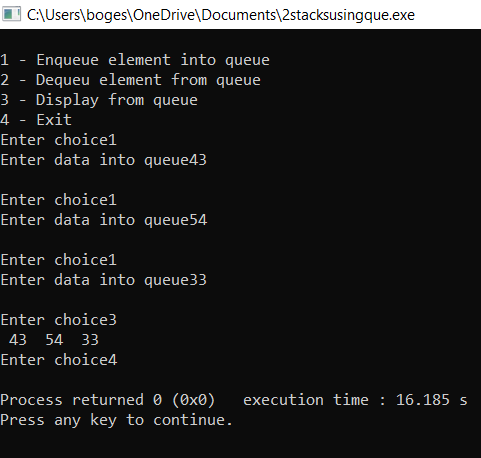
**{**

**printf(" %d ", st1[i]);**

**}**

**}**

**OUTPUT:**



**Exercise-6:**

1. **Write C program that implement Stack using Linked List.**

**AIM: To Write C program that implement Stack using Linked List**

**PROGRAM:**

**#include <stdio.h>**

**#include <stdlib.h>**

**void push();**

**void pop();**

**void display();**

**struct node**

**{**

**int val;**

**struct node \*next;**

**};**

**struct node \*head;**

**void main ()**

**{**

**int choice=0;**

**printf("\nStack operations using linked list\n");**

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

**while(choice != 4)**

**{**

**printf("\n\nChose one from the below options...\n");**

**printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");**

**printf("\n Enter your choice \n");**

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

**switch(choice)**

**{**

**case 1:**

**{**

**push();**

**break;**

**}**

**case 2:**

**{**

**pop();**

**break;**

**}**

**case 3:**

**{**

**display();**

**break;**

**}**

**case 4:**

**{**

**printf("Exiting....");**

**break;**

**}**

**default:**

**{**

**printf("Please Enter valid choice ");**

**}**

**};**

**}**

**}**

**void push ()**

**{**

**int val;**

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

**if(ptr == NULL)**

**{**

**printf("not able to push the element");**

**}**

**else**

**{**

**printf("Enter the value");**

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

**if(head==NULL)**

**{**

**ptr->val = val;**

**ptr -> next = NULL;**

**head=ptr;**

**}**

**else**

**{**

**ptr->val = val;**

**ptr->next = head;**

**head=ptr;**

**}**

**printf("Item pushed");**

**}**

**}**

**void pop()**

**{**

**int item;**

**struct node \*ptr;**

**if (head == NULL)**

**{**

**printf("Underflow");**

**}**

**else**

**{**

**item = head->val;**

**ptr = head;**

**head = head->next;**

**free(ptr);**

**printf("Item popped");**

**}**

**}**

**void display()**

**{**

**int i;**

**struct node \*ptr;**

**ptr=head;**

**if(ptr == NULL)**

**{**

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

**}**

**else**

**{**

**printf("Printing Stack elements \n");**

**while(ptr!=NULL)**

**{**

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

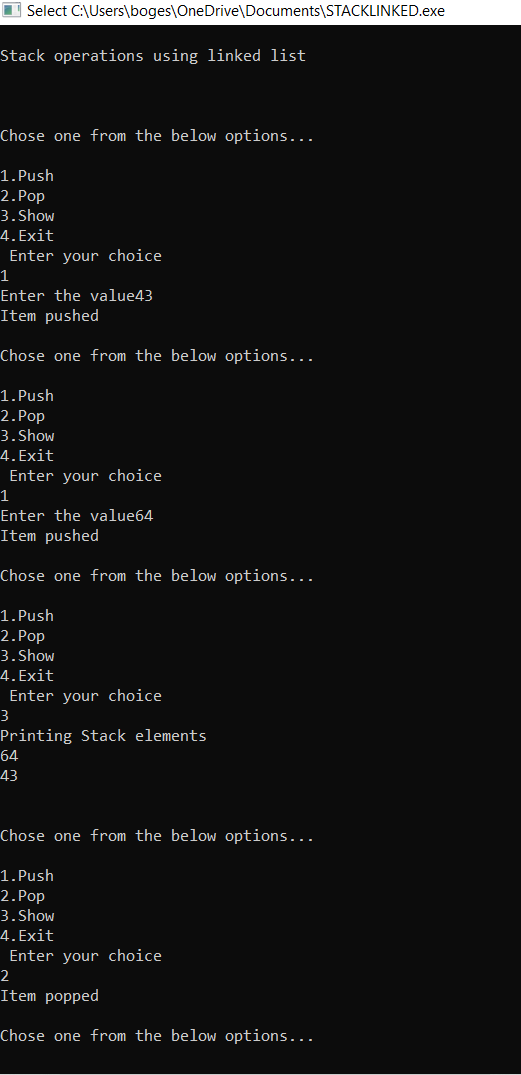
**ptr = ptr->next;**

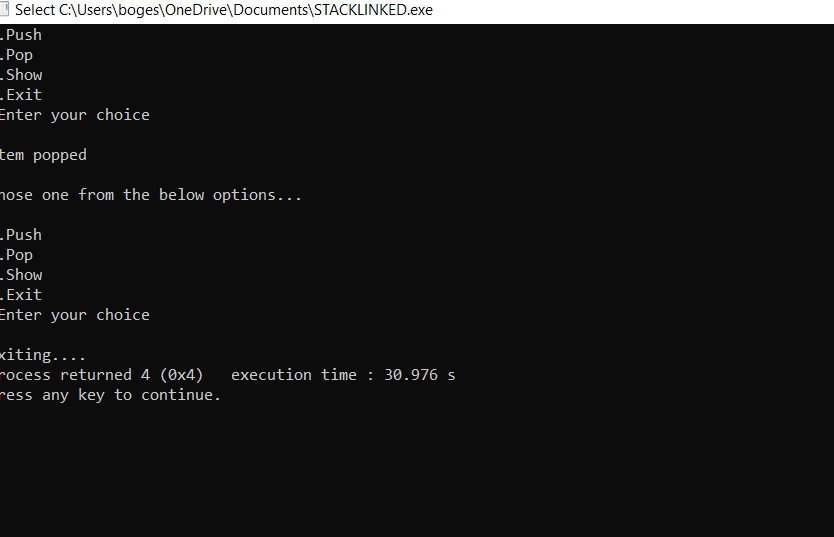
**}**

**}**

**}**

**OUTPUT:**





**B.)** **Write C program that implement Queue using Linked List.**

**AIM: To Write C program that implement Queue using Linked List.**

**PROGRAM:**

**#include<stdlib.h>**

**struct node**

**{**

**int data;**

**struct node \*next;**

**};**

**struct node \*front;**

**struct node \*rear;**

**void insert();**

**void delete();**

**void display();**

**void main ()**

**{**

**int choice;**

**while(choice != 4)**

**{**

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

**printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");**

**printf("\nEnter your choice ?");**

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

**switch(choice)**

**{**

**case 1:**

**insert();**

**break;**

**case 2:**

**delete();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**exit(0);**

**break;**

**default:**

**printf("\nEnter valid choice??\n");**

**}**

**}**

**}**

**void insert()**

**{**

**struct node \*ptr;**

**int item;**

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

**if(ptr == NULL)**

**{**

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

**return;**

**}**

**else**

**{**

**printf("\nEnter value?\n");**

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

**ptr -> data = item;**

**if(front == NULL)**

**{**

**front = ptr;**

**rear = ptr;**

**front -> next = NULL;**

**rear -> next = NULL;**

**}**

**else**

**{**

**rear -> next = ptr;**

**rear = ptr;**

**rear->next = NULL;**

**}**

**}**

**}**

**void delete ()**

**{**

**struct node \*ptr;**

**if(front == NULL)**

**{**

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

**return;**

**}**

**else**

**{**

**ptr = front;**

**front = front -> next;**

**free(ptr);**

**}**

**}**

**void display()**

**{**

**struct node \*ptr;**

**ptr = front;**

**if(front == NULL)**

**{**

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

**}**

**else**

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

**while(ptr != NULL)**

**{**

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

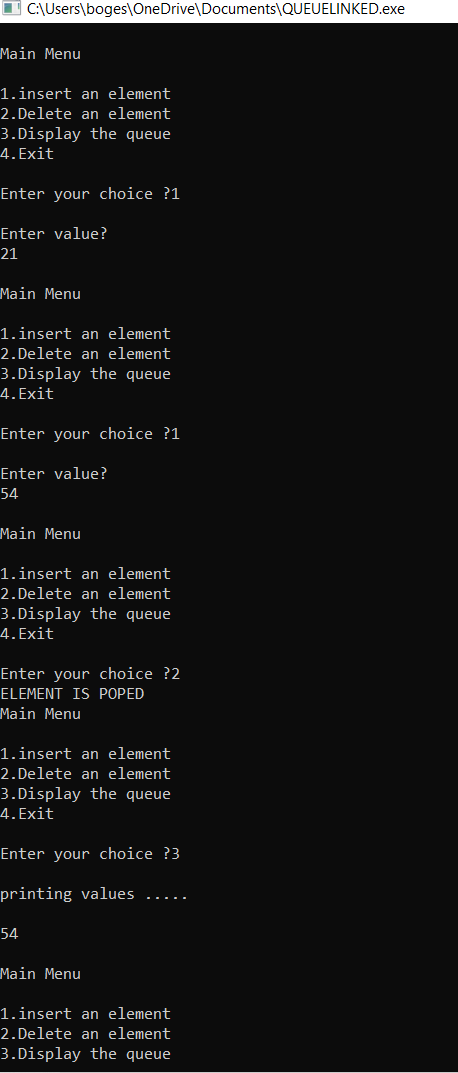
**ptr = ptr -> next;**

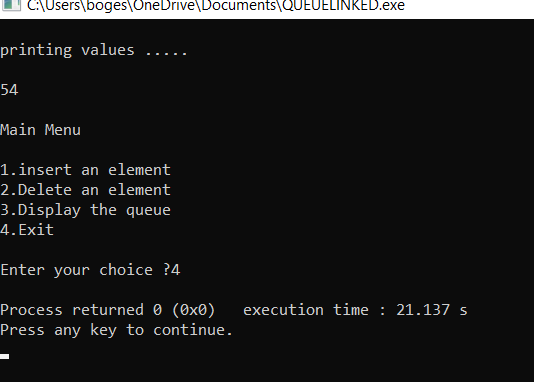
**}**

**}**

**}**

**OUTPUT:**





1. **Write a C program to implement the Circular Queue.**

**AIM: To Write a C program to implement the Circular Queue**

**PROGRAM:**

**#include<stdio.h>**

**# define MAX 5**

**int cqueue\_arr[MAX];**

**int front = -1;**

**int rear = -1;**

**void insert(int item)**

**{**

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

**{**

**printf("Queue Overflow n");**

**return;**

**}**

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

**{**

**front = 0;**

**rear = 0;**

**}**

**else**

**{**

**if(rear == MAX-1)**

**rear = 0;**

**else**

**rear = rear+1;**

**}**

**cqueue\_arr[rear] = item ;**

**}**

**void deletion()**

**{**

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

**{**

**printf("Queue Underflown");**

**return ;**

**}**

**printf("Element deleted from queue is : %dn",cqueue\_arr[front]);**

**if(front == rear)**

**{**

**front = -1;**

**rear=-1;**

**}**

**else**

**{**

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

**front = 0;**

**else**

**front = front+1;**

**}**

**}**

**void display()**

**{**

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

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

**{**

**printf("Queue is emptyn");**

**return;**

**}**

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

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

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

**{**

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

**front\_pos++;**

**}**

**else**

**{**

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

**{**

**printf("%d ",cqueue\_arr[front\_pos])**

**front\_pos++;**

**}**

**front\_pos = 0;**

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

**{**

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

**front\_pos++;**

**}**

**}**

**printf("n");**

**}**

**int main()**

**{**

**int choice,item;**

**do**

**{**

**printf("1.Insertn");**

**printf("2.Deleten");**

**printf("3.Displayn");**

**printf("4.Quitn");**

**printf("Enter your choice : ");**

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

**switch(choice)**

**{**

**case 1 :**

**printf("Input the element for insertion in queue : ");**

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

**insert(item);**

**break;**

**case 2 :**

**deletion();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**break;**

**default:**

**printf("Wrong choicen");**

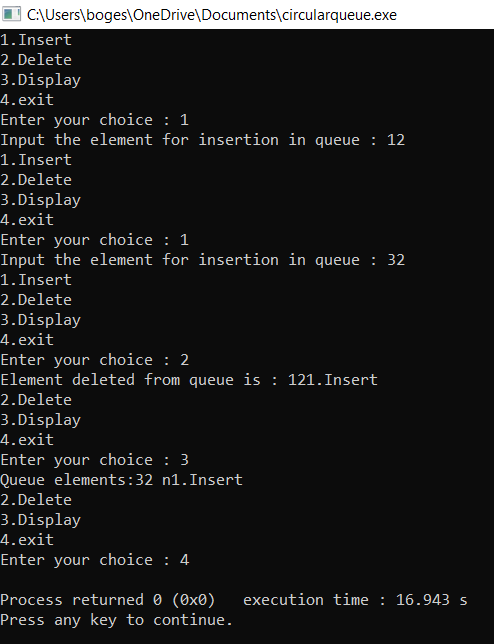
**}**

**}while(choice!=4);**

**return 0;**

**}**

**OUTPUT:**



**Exercise 7:**

**a. Write a C program to insert elements in a Binary Search Tree (BST).**

**b. Write a C program to delete element in a Binary Search Tree (BST).**

**c. Write a C program to implement BST traversals: Inorder, Preorder, and postorder.**

**AIM: To write a program to insert elements ,to delete elements in a binary search tree and to implement traversals: Inorder, Preorder,Postorder.**

**PRORGAM:**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct btnode**

**{**

**int value;**

**struct btnode \*l;**

**struct btnode \*r;**

**}\*root = NULL, \*temp = NULL, \*t2, \*t1;**

**void delete1();**

**void insert();**

**void delete();**

**void inorder(struct btnode \*t);**

**void create();**

**void search(struct btnode \*t);**

**void preorder(struct btnode \*t);**

**void postorder(struct btnode \*t);**

**void search1(struct btnode \*t,int data);**

**int smallest(struct btnode \*t);**

**int largest(struct btnode \*t);**

**int flag = 1;**

**void main()**

**{**

**int ch;**

**printf("\nOPERATIONS");**

**printf("\n1 - Insert an element into tree\n");**

**printf("2 - Delete an element from the tree\n");**

**printf("3 - Inorder Traversal\n");**

**printf("4 - Preorder Traversal\n");**

**printf("5 - Postorder Traversal\n");**

**printf("6 - Exit\n");**

**while(1)**

**{**

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

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

**switch (ch)**

**{**

**case 1:**

**insert();**

**break;**

**case 2:**

**delete();**

**break;**

**case 3:**

**inorder(root);**

**break;**

**case 4:**

**preorder(root);**

**break;**

**case 5:**

**postorder(root);**

**break;**

**case 6:**

**exit(0);**

**default :**

**printf("Wrong choice, Please enter correct choice ");**

**break;**

**}**

**}**

**}**

**void insert()**

**{**

**create();**

**if (root == NULL)**

**root = temp;**

**else**

**search(root);**

**}**

**void create()**

**{**

**int data;**

**printf("Enter data of node to be inserted : ");**

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

**temp = (struct btnode \*)malloc(1\*sizeof(struct btnode));**

**temp->value = data;**

**temp->l = temp->r = NULL;**

**}**

**void search(struct btnode \*t)**

**{**

**if ((temp->value > t->value) && (t->r != NULL))**

**search(t->r);**

**else if ((temp->value > t->value) && (t->r == NULL))**

**t->r = temp;**

**else if ((temp->value < t->value) && (t->l != NULL))**

**search(t->l);**

**else if ((temp->value < t->value) && (t->l == NULL))**

**t->l = temp;**

**}**

**void inorder(struct btnode \*t)**

**{**

**if (root == NULL)**

**{**

**printf("No elements in a tree to display");**

**return;**

**}**

**if (t->l != NULL)**

**inorder(t->l);**

**printf("%d -> ", t->value);**

**if (t->r != NULL)**

**inorder(t->r);**

**}**

**void delete()**

**{**

**int data;**

**if (root == NULL)**

**{**

**printf("No elements in a tree to delete");**

**return;**

**}**

**printf("Enter the data to be deleted : ");**

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

**t1 = root;**

**t2 = root;**

**search1(root, data);**

**}**

**void preorder(struct btnode \*t)**

**{**

**if (root == NULL)**

**{**

**printf("No elements in a tree to display");**

**return;**

**}**

**printf("%d -> ", t->value);**

**if (t->l != NULL)**

**preorder(t->l);**

**if (t->r != NULL)**

**preorder(t->r);**

**}**

**void postorder(struct btnode \*t)**

**{**

**if (root == NULL)**

**{**

**printf("No elements in a tree to display ");**

**return;**

**}**

**if (t->l != NULL)**

**postorder(t->l);**

**if (t->r != NULL)**

**postorder(t->r);**

**printf("%d -> ", t->value);**

**}**

**void search1(struct btnode \*t, int data)**

**{**

**if ((data>t->value))**

**{**

**t1 = t;**

**search1(t->r, data);**

**}**

**else if ((data < t->value))**

**{**

**t1 = t;**

**search1(t->l, data);**

**}**

**else if ((data==t->value))**

**{**

**delete1(t);**

**}**

**}**

**void delete1(struct btnode \*t)**

**{**

**int k;**

**/\* To delete leaf node \*/**

**if ((t->l == NULL) && (t->r == NULL))**

**{**

**if (t1->l == t)**

**{**

**t1->l = NULL;**

**}**

**else**

**{**

**t1->r = NULL;**

**}**

**t = NULL;**

**free(t);**

**return;**

**}**

**else if ((t->r == NULL))**

**{**

**if (t1 == t)**

**{**

**root = t->l;**

**t1 = root;**

**}**

**else if (t1->l == t)**

**{**

**t1->l = t->l;**

**}**

**else**

**{**

**t1->r = t->l;**

**}**

**t = NULL;**

**free(t);**

**return;**

**}**

**else if (t->l == NULL)**

**{**

**if (t1 == t)**

**{**

**root = t->r;**

**t1 = root;**

**}**

**else if (t1->r == t)**

**t1->r = t->r;**

**else**

**t1->l = t->r;**

**t == NULL;**

**free(t);**

**return;**

**}**

**else if ((t->l != NULL) && (t->r != NULL))**

**{**

**t2 = root;**

**if (t->r != NULL)**

**{**

**k = smallest(t->r);**

**flag = 1;**

**}**

**else**

**{**

**k =largest(t->l);**

**flag = 2;**

**}**

**search1(root, k);**

**t->value = k;**

**}**

**}**

**int smallest(struct btnode \*t)**

**{**

**t2 = t;**

**if (t->l != NULL)**

**{**

**t2 = t;**

**return(smallest(t->l));**

**}**

**else**

**return (t->value);**

**}**

**int largest(struct btnode \*t)**

**{**

**if (t->r != NULL)**

**{**

**t2 = t;**

**return(largest(t->r));**

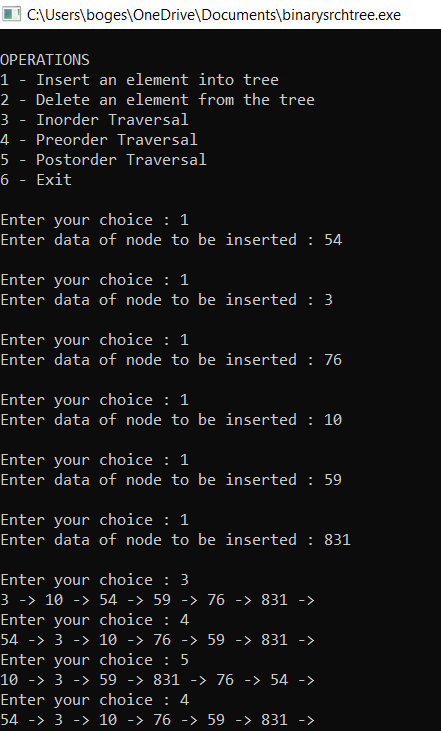
**}**

**else**

**return(t->value);**

**}**

**OUTPUT:**



**Exercise-8**

**a.)** **a. Write a C program to implement the Max Heap.**

**b. Write C program that implement Heap sort, to sort elements in an ascending order.**

**AIM: To a. Write a C program to implement the Max Heap and to implement Heap sort, to sort elements in an ascending order.**

**PROGRAM:**

**#include <stdio.h>**

**int main()**

**{**

**int arr[10], no, i, j, c, heap\_root, temp;**

**printf("Input number of elements: ");**

**scanf("%d", &no);**

**printf("\nInput array values one by one : ");**

**for (i = 0; i < no; i++)**

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

**for (i = 1; i < no; i++)**

**{**

**c = i;**

**do**

**{**

**heap\_root = (c - 1) / 2;**

**if (arr[heap\_root] < arr[c])**

**{**

**temp = arr[heap\_root];**

**arr[heap\_root] = arr[c];**

**arr[c] = temp;**

**}**

**c = heap\_root;**

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

**}**

**printf("Heap array : ");**

**for (i = 0; i < no; i++)**

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

**for (j = no - 1; j >= 0; j--)**

**{**

**temp = arr[0];**

**arr[0] = arr[j];**

**arr[j] = temp;**

**heap\_root = 0;**

**do**

**{**

**c = 2 \* heap\_root + 1;**

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

**c++;**

**if (arr[heap\_root]<arr[c] && c<j)**

**{**

**temp = arr[heap\_root];**

**arr[heap\_root] = arr[c];**

**arr[c] = temp;**

**}**

**heap\_root = c;**

**} while (c < j);**

**}**

**printf("\nSorted array : ");**

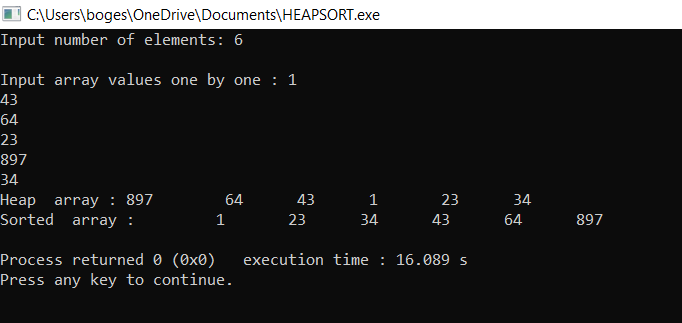
**for (i = 0; i < no; i++)**

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

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

**}**

**OUTPUT**



**Exercise-9**

**A.) Write a C program to implement the Breadth First Search technique on a Graph**

**AIM: To a. Write a C program to implement the Breadth First Search technique on a Graph**

**PROGRAM:**

**#include<stdio.h>**

**#include<conio.h>**

**int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;**

**void bfs(int v) {**

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

**if(a[v][i] && !visited[i])**

**q[++r]=i;**

**if(f<=r) {**

**visited[q[f]]=1;**

**bfs(q[f++]);**

**}**

**}**

**void main() {**

**int v;**

**printf("\n Enter the number of vertices:");**

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

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

**q[i]=0;**

**visited[i]=0;**

**}**

**printf("\n Enter graph data in matrix form:\n");**

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

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

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

**printf("\n Enter the starting vertex:");**

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

**bfs(v);**

**printf("\n The node which are reachable are:\n");**

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

**if(visited[i])**

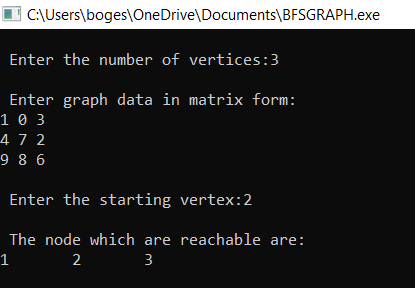
**printf("%d\t",i); else**

**printf("\n Bfs is not possible");**

**getch();**

**}**

**OUTPUT:**



**B.)** **Write a C program to implement the Depth First Search technique on a Graph.**

**AIM: To Write a C program to implement the Depth First Search technique on a Graph.**

**PROGRAM:**

**#include<stdio.h>**

**#include<conio.h>**

**int a[20][20],reach[20],n;**

**void dfs(int v) {**

**int i;**

**reach[v]=1;**

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

**if(a[v][i] && !reach[i]) {**

**printf("\n %d->%d",v,i);**

**dfs(i);**

**}**

**}**

**void main() {**

**int i,j,count=0;**

**clrscr();**

**printf("\n Enter number of vertices:");**

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

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

**reach[i]=0;**

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

**a[i][j]=0;**

**}**

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

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

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

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

**dfs(1);**

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

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

**if(reach[i])**

**count++;**

**}**

**if(count==n)**

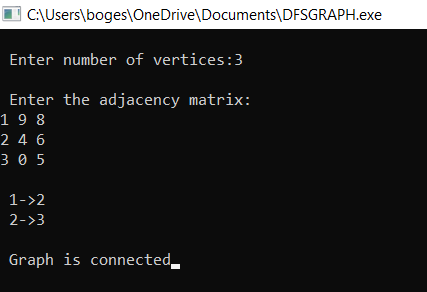
**printf("\n Graph is connected"); else**

**printf("\n Graph is not connected");**

**getch();**

**}**

**OUTPUT:**



**Exercise-10**

1. **Write a C program to implement the Prim’s algorithm to construct Minimum Spanning Tree.**

**AIM: To a. Write a C program to implement the Prim’s algorithm to construct Minimum Spanning Tree.**

**PROGRAM:**

**#include<stdio.h>**

**#include<conio.h>**

**int a,b,u,v,n,i,j,ne=1;**

**int visited[10]= {**

**0**

**}**

**,min,mincost=0,cost[10][10];**

**void main() {**

**printf("\n Enter the number of nodes:");**

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

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

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

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

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

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

**cost[i][j]=999;**

**}**

**visited[1]=1;**

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

**while(ne<n) {**

**for (i=1,min=999;i<=n;i++)**

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

**if(cost[i][j]<min)**

**if(visited[i]!=0) {**

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

**a=u=i;**

**b=v=j;**

**}**

**if(visited[u]==0 || visited[v]==0) {**

**printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);**

**mincost+=min;**

**visited[b]=1;**

**}**

**cost[a][b]=cost[b][a]=999;**

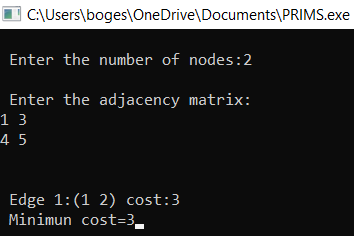
**}**

**printf("\n Minimun cost=%d",mincost);**

**getch();**

**}**

**OUTPUT:**



1. **Write a C program to implement the Kruskal’s algorithm to construct Minimum Spanning Tree.**

**AIM: To b. Write a C program to implement the Kruskal’s algorithm to construct Minimum Spanning Tree**

**PROGRAM:**

**#include<stdio.h>**

**#include<conio.h>**

**#include<stdlib.h>**

**int i,j,k,a,b,u,v,n,ne=1;**

**int min,mincost=0,cost[9][9],parent[9];**

**int find(int);**

**int uni(int,int);**

**void main()**

**{**

**printf("\n\tImplementation of Kruskal's algorithm\n");**

**printf("\nEnter the no. of vertices:");**

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

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

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

**{**

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

**{**

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

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

**cost[i][j]=999;**

**}**

**}**

**printf("The edges of Minimum Cost Spanning Tree are\n");**

**while(ne < n)**

**{**

**for(i=1,min=999;i<=n;i++)**

**{**

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

**{**

**if(cost[i][j] < min)**

**{**

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

**a=u=i;**

**b=v=j;**

**}**

**}**

**}**

**u=find(u);**

**v=find(v);**

**if(uni(u,v))**

**{**

**printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);**

**mincost +=min;**

**}**

**cost[a][b]=cost[b][a]=999;**

**}**

**printf("\n\tMinimum cost = %d\n",mincost);**

**getch();**

**}**

**int find(int i)**

**{**

**while(parent[i])**

**i=parent[i];**

**return i;**

**}**

**int uni(int i,int j)**

**{**

**if(i!=j)**

**{**

**parent[j]=i;**

**return 1;**

**}**

**return 0;**

**}**

**OUTPUT:**

