

**Laboratory Manual**

**Data Structures & Algorithms**

**VIDUSH SOMANY INSTITUTE OF TECHNOLOGY AND RESEARCH**

**Name : \_**

**Semester :\_ Branch :**

**Enrolment No. :**



CERTIFICATE

*This is to certify that the practical/term work carried in subject of Data Structures And Algorithms and recorded in this journal is the bonafide work of Mr./Ms \_*

*Enrolment No. : of B.E.*

*Year \_ Semester in the branch of during the academic year 20*

*-20 within the four walls of this institute*.

# Faculty in Charge Date Head of the Department



**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.**  **No.** | **TITLE** | **Date** | **Sign.** |
| 1 | Write a menu driven program to perform the following operations on the **STACK** using an array.  1. Push 2. Pop 3.Peep 4. Change 5. Display the contents 6. Exit. |  |  |
| 2 | Write a program to convert an infix expression into reverse polish (postfix) notation with Parenthesis. |  |  |
| 3 | Write a program to solve the problem of Tower of Hanoi (Application of stack) |  |  |
| 4 | Write a menu driven program to perform the following operations on the **QUEUE** using an Array.  1. Insert 2. Delete 3.Search 4. Change 5. Display the contents 6. Exit |  |  |
| 5 | Write a menu driven program to perform the following operations on the **CIRCULARQUEUE** Using an array.1. Insert 2. Delete 3. Search 4. Change 5. Display the contents 6. Exit |  |  |
| 6 | Write a menu driven program to perform the following operations on a **Singly Linked list**. Insert Search Insend Sort Insat Count Delete Display Reverse Exit. |  |  |
| 7 | Write a menu driven program to perform the following operations on a **Doubly Linked list**.1. Insert 2. Insend 3. Insat 4. Delete 5. Display 6. Exit |  |  |
| 8 | Write a program to implement Searching Algorithms Sequential search Binary search |  |  |
| 9 | Write a program to implement following sorting algorithms Selection sort Bubble sort Merge sort Quick sort |  |  |
| 10 | Write a program to implement breadth first search (BFS) graph traversal algorithm. |  |  |
| 11 | Write a program to implement depth first search (DFS) graph traversal algorithm. |  |  |

**Practical No. 1**

**Date:**

**֍ Aim:** Write a menu driven program to perform the following

operations on the STACK using an array. 1. Push 2. Pop 3. Peep 4. Change

5. Display the contents 6. Exit

**CODE:**

#include<stdio.h> #include<stdlib.h> #define size 5

int S[size], top=- 1;

void push(int x)

{

if(top==size-1){

printf("Stack is full\n"); return;

}

else{

top++;

S[top]=x;

printf("Element pushed : %d\n",S[top]);

}

}

void pop(void)

{

if(top==-1)

{

printf("Stack is empty"); return;

}

else

{

printf("Element popped : %d\n",S[top]); top--;

}

}

int peep()

{

printf("Element peeped:%d\n",S[top]);

}

int isFull()

{

if(top==size-1) printf("Stack is Full\n"); else

printf("Stack is NOT Full\n");

}

int isEmpty()

{

if(top==-1)

printf("Stack is empty\n"); else printf("Stack is NOT empty\n");

}

int main()

{

int choice, element; while(choice!=6)

{

printf("\n1: Push");

printf("\n2: Pop");

printf("\n3: Peep");

printf("\n4: IsFull");

printf("\n5: IsEmpty");

printf("\n6: Exit");

printf("\nEnter your choice:

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

{

case 1:

printf("You have selected PUSH option\n"); printf("Enter the element to push: ");

scanf("%d",&element); push(element); break;

case 2:

printf("You have selected POP option\n"); pop();

break;

case 3:

printf("You have selected PEEP option\n"); peep();

break;

case 4:

printf("You have selected IsFull option\n"); isFull();

break;

case 5:

printf("You have selected IsEmpty option\n"); isEmpty();

break;

case 6:

printf("You have selected exit"); exit (0);

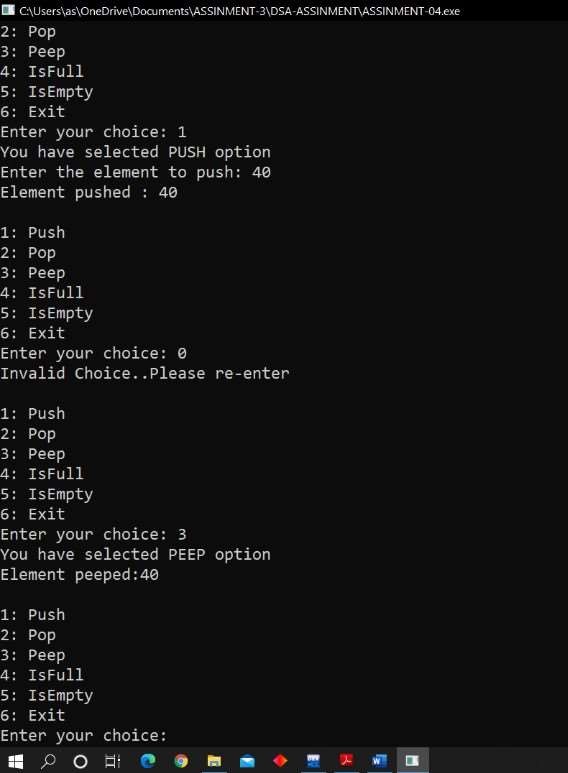
default: printf("Invalid Choice..Please re-enter\n");

} }

return 0;

}

**OUTPUT:**



**Practical No. 2**

**Date:**

**֍ Aim:** Write a program to convert an infix expression into reverse

polish (postfix) notation with parenthesis

# CODE:

#include<stdio.h

>

#include<ctype.h>

char stack[20]; int top=-1; void push(char x)

{

stack[++top]=x;

}

char pop(void)

{

if(top==-1){

return - 1;

}

Else{

return stack[top--];

}

}

int priority(char x)

{ if(x=='(')

return 0;

if(x=='+'||x=='-') return 1;

if(x=='\*'||x=='/')

return 2;

}

int main()

{

char str[20],x;

int i=0;

printf("ENTER YOUR STRING:");

scanf("%s",str);

while(str[i]!='\0')

{

if(isalnum(str[i]))

printf("%c",str[i]);

else if(str[i]=='(') push(str[i]); else if(str[i]==')')

{

while((x=pop())!='(')

printf("%c",x);

}

else

{

while(priority(stack[top])>=priority(str[i])) printf("%c",pop());

push(str[i]);

}i++;

}

while(top!=-1)

{

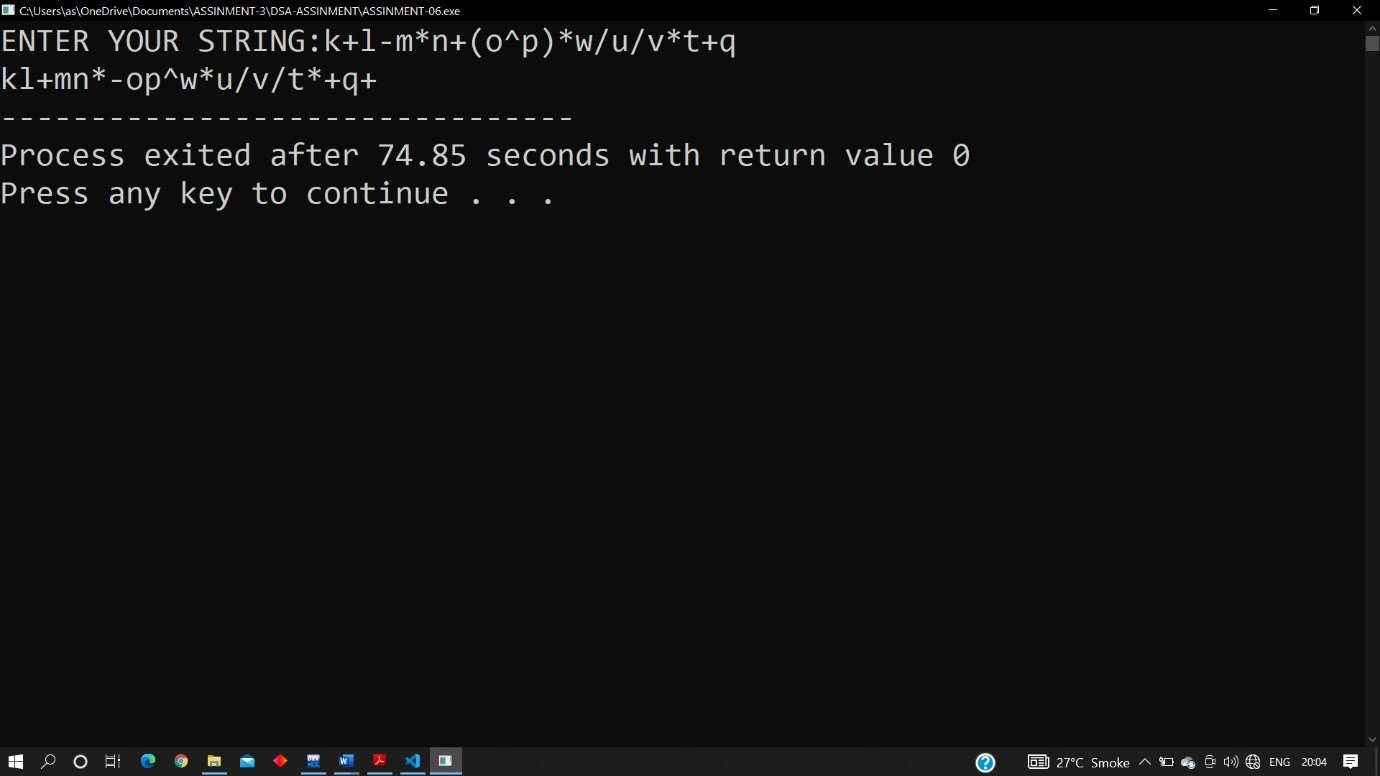
printf("%c",pop());

}

return 0;

}

**OUTPUT:**



**Practical No. 3**

**Date:**

**֍ Aim:** Write a program to solve the problem of Tower of Hanoi

(Application of stack)

# CODE:

#include<stdio.h> int S,D,T;

void TOH(int n, char source, char temp, char dest)

{

if(n>0)

{

TOH(n-1,source,dest,temp);

printf("move disk # %d from %c to %c\n",n,source,temp); TOH(n-1,dest,temp,source);

}

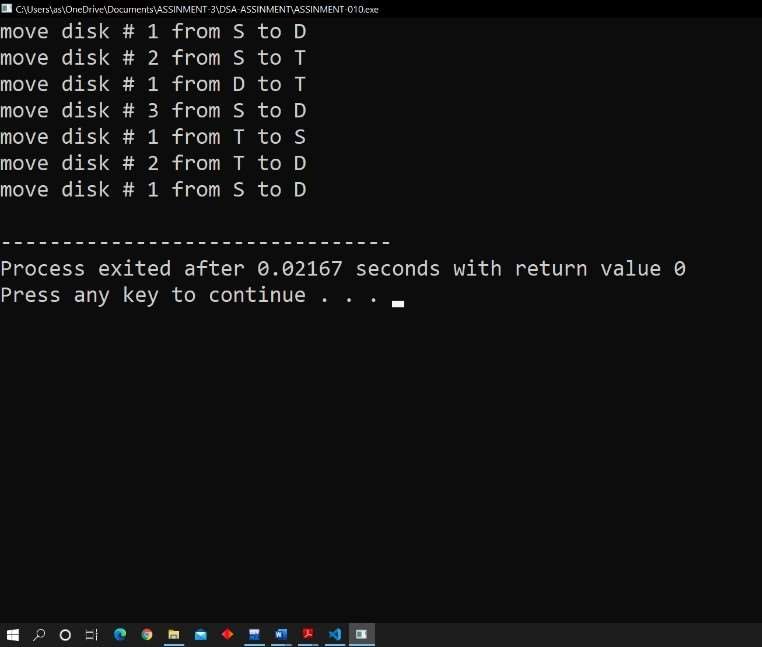
}

int main()

{ int n=3; TOH(n,'S','D','T');

}

**OUTPUT:**



**Practical No. 4**

**Date:**

**֍ Aim:** Write a menu driven program to perform the following

operations on the QUEUE using an array. 1. Insert 2. Delete 3. Search

4. Change 5. Display the contents 6. Exit

# CODE:

#include<stdio.h> #define size 5

int queue[size];

int front=-1,rear=-1; void enqueue(int element)

{

if(rear==size-1)

{printf("Queue is Over flow=\n");} else if(front==-1 &&rear=1)

{front=rear=0;queue[rear]=element;} else{rear++;queue[rear]=element;}

}

void dequeue(void)

{

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

{printf("Queue is Under flow=\n");} else if(front==rear)

else

}

{printf("Element Deleted=%d\n",queue[front]); front=rear=-1;}

{printf("Element Deleted=%d\n",queue[front]);front++;}

void display(void)

{

Int i;

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

{printf("Queue is Empty\n");} else{for(i=front;i<=rear;++i)

printf("%d\t",queue[i]);printf("\n");}

}

void peep(void)

{

if(front==-1 && rear==-1) printf("Queue is Empty\n");

else

}

{printf("Data at front=%d\n",queue[front]);}

int main()

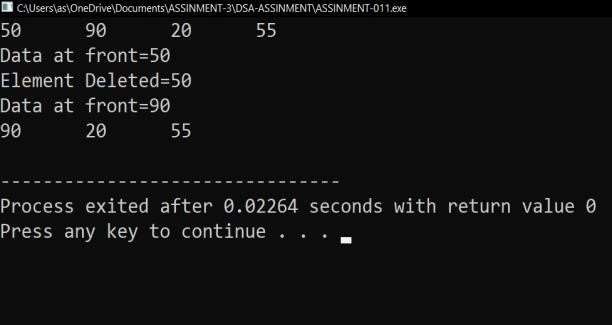
{

enqueue(50); enqueue(90); enqueue(20); enqueue(55);

display(); peep(); dequeue(); peep(); display(); return 0;

}

**OUTPUT:**



**Practical No. 5**

**Date:**

**֍ Aim:** Write a menu driven program to perform the following

operations on the CIRCULARQUEUE using an array. 1. Insert 2. Delete

3. Search 4. Change 5. Display the contents 6. Exit

# CODE:

#include<stdio.h> #define size 5 int queue[size]; int front=- 1,rear=-1; void enqueue(int element)

{

if((rear+1)%size==front){printf("Queue is Over flow=\n");} else if(front==-1 && rear==-

1){front=rear=0;queue[rear]=element;} else{rear=(rear+1)%size;queue[rear]=element;}

}

void dequeue(void)

{

if(front==-1 && rear==-1){printf("Queue is Under flow=\n");}

else if(front==rear){printf("Element Deleted=%d\n",queue[front]);front=rear=- 1;} else{printf("Element

Deleted=%d\n",queue[front]);front=(front+1)%size;}

}

void display(void)

{ int i=front;

if(front==-1 && rear==-1){printf("Queue is Empty\n");} else{while(i!=rear){printf("%d\t",queue[i]);i=(i+1)%size;} printf("\n");}

}

void peep(void)

{

if(front==-1 && rear==-1)printf("Queue is Empty\n"); else{printf("Data at front=%d\n",queue[front]);}

} int main()

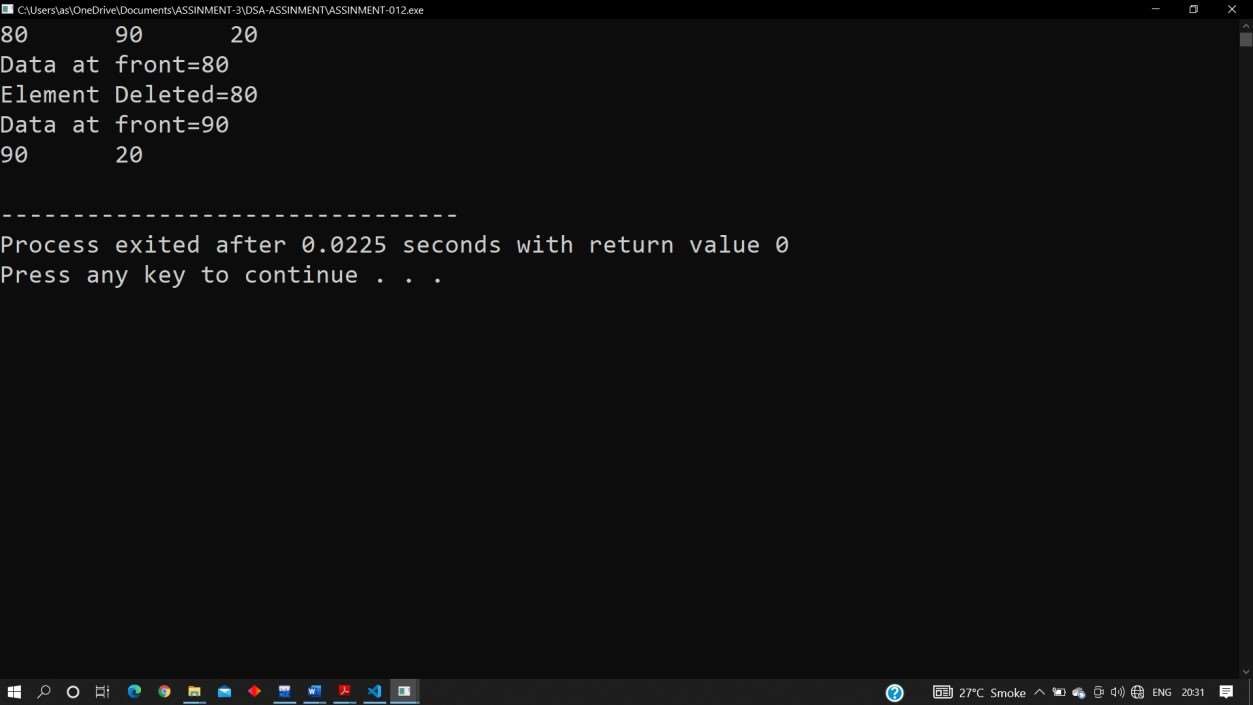
{

enqueue(80); enqueue(90); enqueue(20); enqueue(55); display(); peep();

dequeue(); peep(); display(); return 0;

}

# OUTPUT:



**Practical No. 6**

**Date:**

**֍ Aim:** Write a menu driven program to perform the following

operations on a Singly Linked list 1. Insert, 2. Insend, 3. Insat, 4. Delete,

6. Search,7. Sort, 8. Count, 9. Display, 10. Exit

**CODE:**

#include <stdio.h> #include<stdlib.h> struct node { int data; struct node \*next;

}; int main ()

{

struct node \*start=NULL, \*newnode, \*temp; int choice; do {

newnode = (struct node\*)malloc (sizeof(struct node)); printf("Enter data : "); scanf("%d",&newnode->data); newnode->next=NULL; if(start==NULL) {start = temp = newnode;} else {

temp->next = newnode; temp=newnode;

} printf("Do you wish to continue? (1 For Yes / 0 For No) : ");

scanf("%d",&choice);

}while(choice != 0); temp

= start; while (temp != NULL)

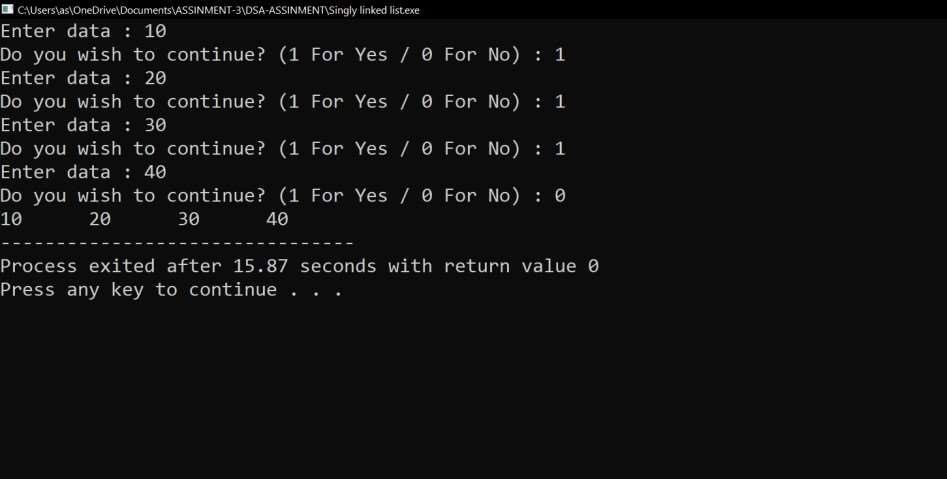
{

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

}

}

# OUTPUT:



**Practical No. 7**

**Date:**

**֍ Aim:** Write a menu driven program to perform the following

operations on a Doubly Linked list.1. Insert 2. Insend 3. Insat 4. Delete 5. Display 6. Exit

**CODE:**

#include<stdio.h> #include<stdlib.h> struct node

{

int data;

struct node

\*next,\*prev;

};

Struct node\*start=NULL,\*temp,

\*end=NULL;

void createDoublyLinkedList(void)

{

struct node\*newnode;

int choice=1; while(choice!=0)

{

newnode=(struct node\*)malloc(sizeof(struct node)); printf("Enter data to insert:");

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

newnode->next=newnode->prev=NULL; if(start==NULL)

{

start=end=newnode;

start->next=start;

start->prev=start;

}

else

{

end->next=newnode;

newnode->prev=end;

newnode->next=start;

start-

>prev=newnode;

end=newnode;

}

printf("Continue? 1: Yes,0:No=>"); scanf("%d",&choice);

}

}

void forwarddisplay()

{

temp=start;

do {

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

}

while(temp!=start); printf("\n");

}

void backwarddisplay()

{

temp=end; do

{

printf("%d\t",temp->data); temp=temp->prev; } while(temp!=end);

printf("\n");

}

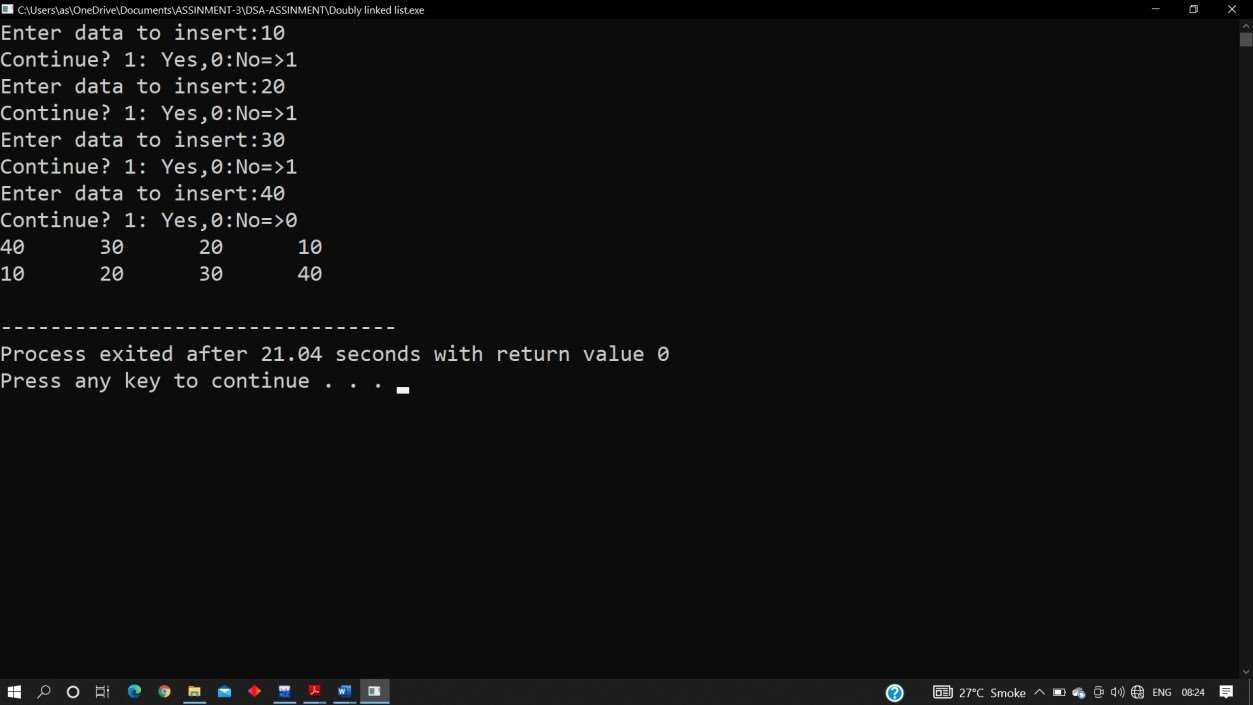
int main()

{

createDoublyLinkedList(); backwarddisplay(); forwarddisplay();

}

**OUTPUT:**



**Practical No. 8**

**Date:**

**֍ Aim:** Write a program to implement Searching Algorithms

# Sequential search CODE:

#include<stdio.h>

int searchLinear (int A[], int n, int x)

{ int I;

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

{if (A[i]==x)return i;} return -1;

}

int main()

{

int A[]={77,12,8,39,27,21,44,18,6,47,11,37,60,56};

int n=14, x=39;

int searchLinear (int[],int,int);

int searchResult = searchLinear (A,n,x); if (searchResult == -1)

printf("Element does not exist");

else

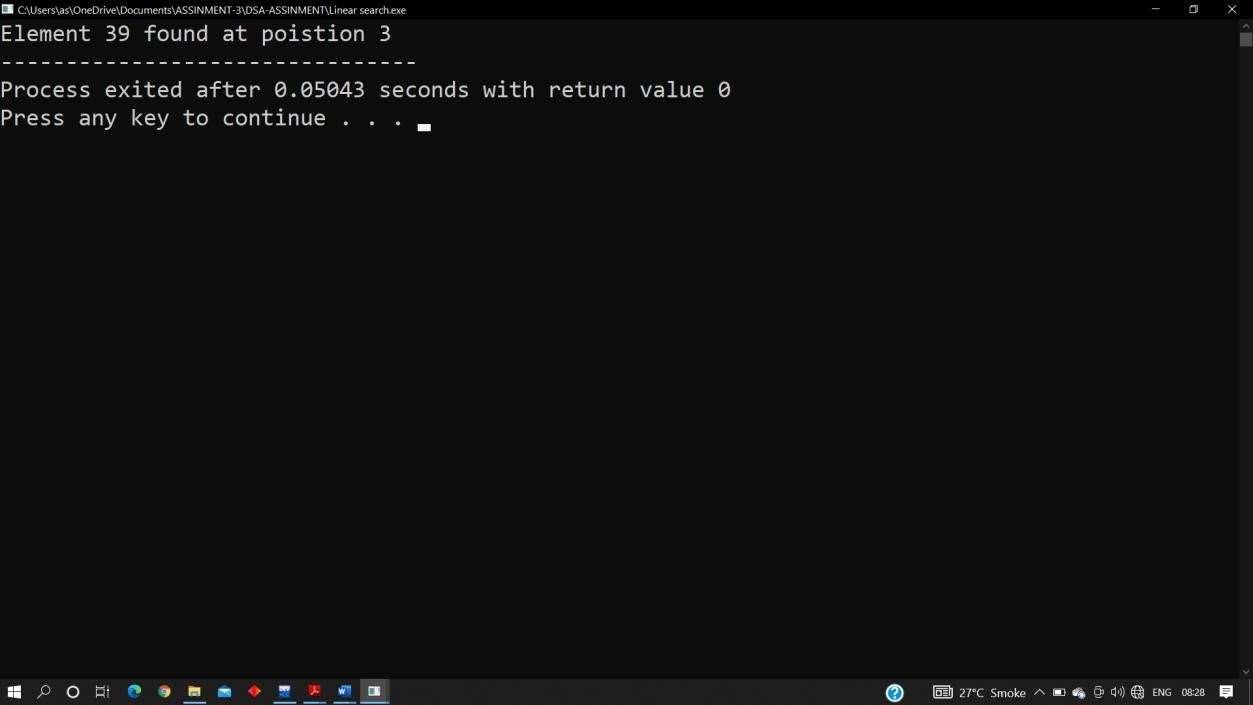
printf("Element %d found at poistion

%d",x,searchResult);

return 0;

}

**OUTPUT:**



# Binary search CODE:

#include<stdio.h>

int searchBinary (int A[], int n, int x) { int i, start=0, end=n-1, middle; while (start <= end)

{ middle = (start + end)/2;

if (A[middle]==x)return middle; else if (A[middle]>x) end=middle- 1; else start=middle+1;

}

return -1;

}

int main()

{ int A[]={6,8,11,12,18,21,27,37,39,44,47,58,60,77}, n=14, x=39;

int searchBinary (int[],int,int);

int searchResult = searchBinary (A,n,x); if (searchResult == -1)

printf("Element does not exist");

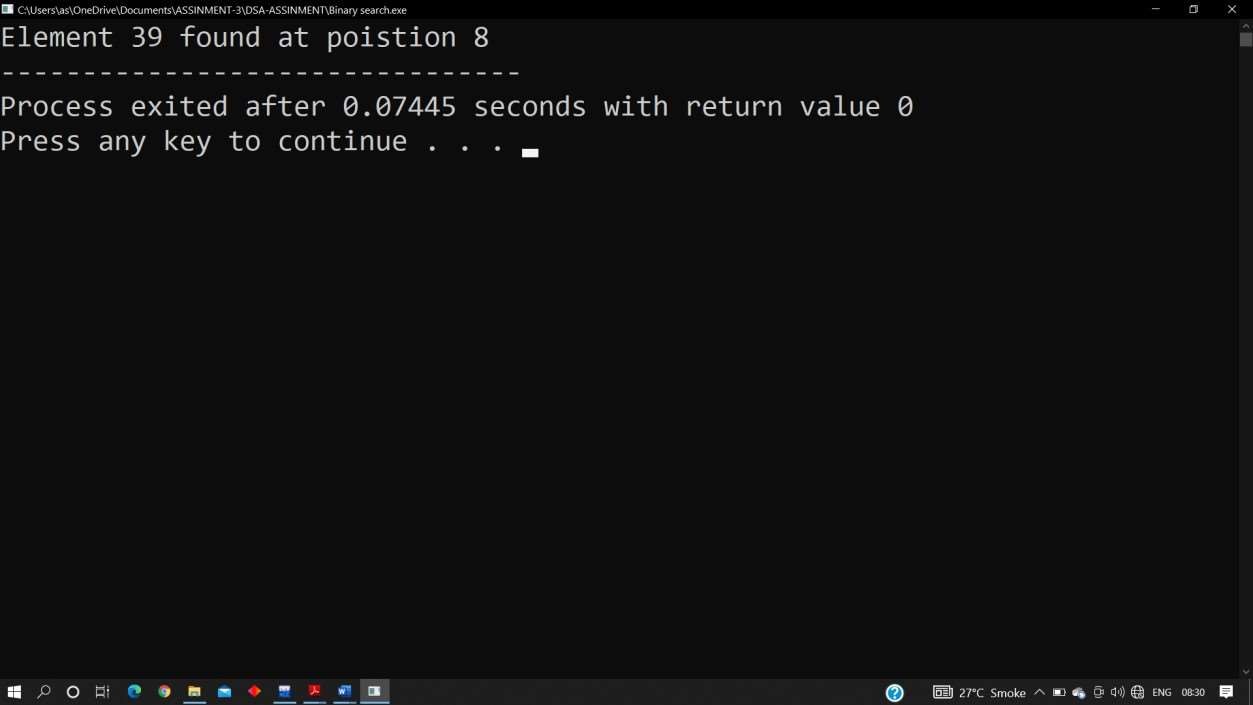
else

}

printf("Element %d found at poistion

%d",x,searchResult); return 0;

**OUTPUT:**



**Practical No. 9**

**Date:**

**֍ Aim:** Write a program to implement following sorting

algorithms 1. Selection sort, 2.Bubble sort, 3. Merge sort, 4.Quick sort

# Selection sort

**CODE:**

#include<stdio.h> int main()

{

int i, j, iMin, temp, count=10, A[25]={42, 23, 74, 11, 65, 58, 94, 36, 99, 87};

for(i=0;i<=count-2;i++)

{

iMin = i;

for(j=i+1; j<=count-1; ++j)

{

if(A[j]<A[iMin]) iMin = j;

}

temp=A[i]; A[i]=A[iMin];

A[iMin]=temp;

}

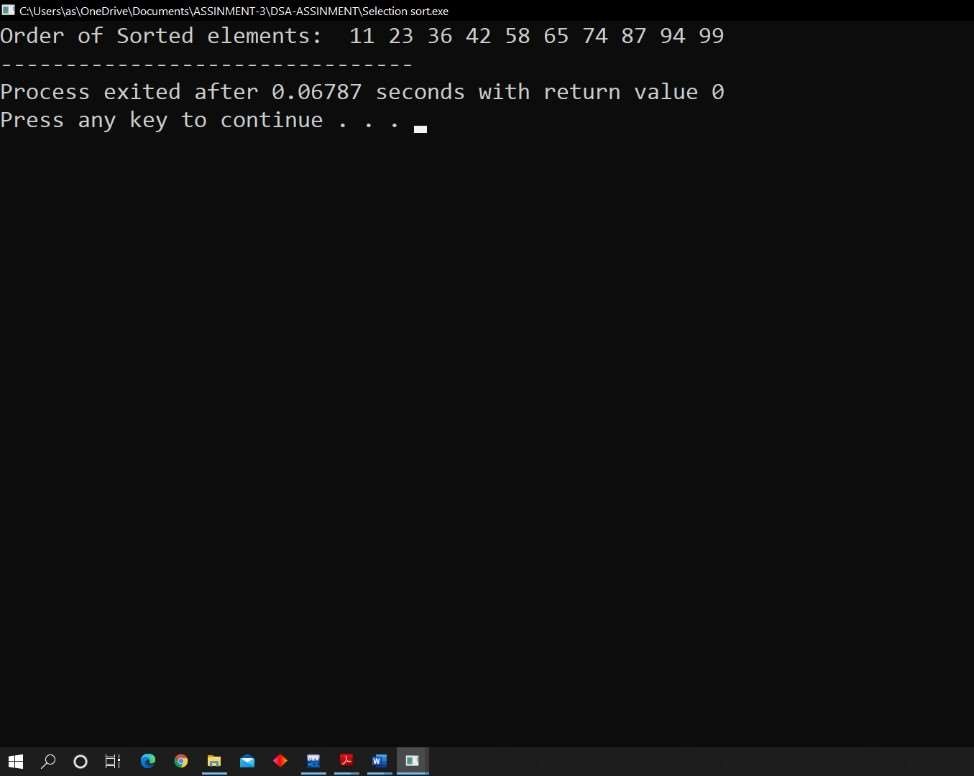
printf("Order of Sorted elements: "); for(i=0;i<count;i++)

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

return 0;

}

**OUTPUT:**



# Bubble sort CODE:

#include<stdio.h> int main()

{

int i, j, temp, count=6;

int A[25]={42, 23, 74, 11, 65, 58};

for(i=0;i<count-1;i++)

{

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

{

if(A[j]>A[j+1])

{ temp=A[j];

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

A[j+1]=temp;

}

}

}

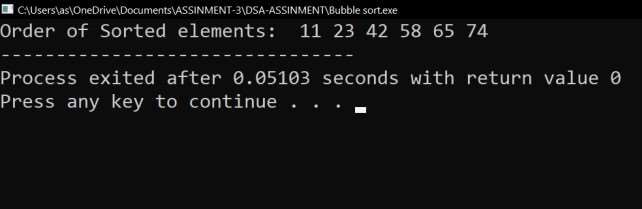
printf("Order of Sorted elements: ");

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

printf(" %d",A[i]); return 0;

}

**OUTPUT:**



# Merge sort

**CODE:**

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

void merge(int A[], int left, int mid, int right)

{

int i, j, k;

int nL = mid - left + 1; int nR = right - mid;

int L[nL], R[nR];

for (i = 0; i < nL; i++) L[i] = A[left + i];

for (j = 0; j < nR; j++) R[j] = A[mid + 1 + j]; i = 0;

// Initial index of first subarray j = 0; // Initial index of second subarray k =

left; // Initial index of merged subarray while (i < nL && j < nR) {

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

{

A[k] = L[i]; i++; }

else

{ A[k] = R[j]; j++; }

k++;

}

while (i < nL)

{

A[k] =L[i]; i++;

k++;

}

while (j < nR)

{ A[k] = R[j]; j++;

k++;

}

}

void mergeSort(int A[], int left, int right)

{

if (left < right)

{

int mid = left + (right - left) / 2; mergeSort(A, left, mid);

mergeSort(A, mid + 1, right);

merge(A, left, mid, right);

}

}

int main()

{

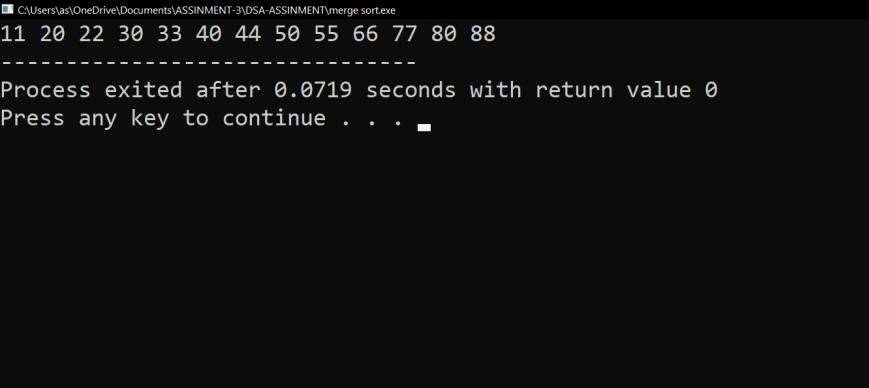
int i, A[] = {66, 33, 40, 22, 55, 88, 11, 80, 20, 50, 44, 77, 30}, size=13;

mergeSort(A, 0, size - 1); for(i=0; i<size; ++i)

printf("%d ",A[i]); return 0;

}

**OUTPUT:**



* 1. **Quick sort**

**CODE**:

#include<stdio.h>

int Partition (int \*A, int start, int end)

{

int i, k, temp;

int pivot = A[end];

int partitionIndex = start;

for (i=start; i<end; ++i)

{

printf ("i=%d A[i]=%d Pivot=%d Partition Index=%d\n",i,A[i], pivot, partitionIndex);

if(A[i]<pivot)

{

temp = A[i];

A[i] = A[partitionIndex]; A[partitionIndex] = temp;

partitionIndex++;

}

for(k=start; k<=end; ++k) printf("\*%d ",A[k]); printf("\n");

}

temp = A[end];

A[end] = A[partitionIndex]; A[partitionIndex] = temp;

printf("Partition Index = %d\n",partitionIndex); for(k=start; k<=end; ++k)

printf("#%d ",A[k]);

printf("\n");

return partitionIndex;

}

int QuickSort (int \*A, int start, int end)

{

int i; if(start< end) {

int partitionIndex = Partition (A, start, end); QuickSort (A, start, partitionIndex-1);

QuickSort (A, partitionIndex+1, end);

}

return 0;

}

int main()

{

int A[] = {34, 90, 21, 43, 87, 2, 67, 53, 9, 23, 82}, i; QuickSort (A,

0, 10);

for(i=0; i<11; ++i) printf("%d ",A[i]); return 0;

}

# OUTPUT:

**Practical No. 10**

**Date:**

**֍ Aim:** Write a program to implement breadth first search (BFS) graph

traversal algorithm**.**

**CODE:**

#include <stdio.h> #define size 8

int output[size], outputPtr=0, queue[size], front=-1, rear=-1; void enqueue(int node)

{

if(front==-1 && rear==- 1)front=rear=0; else rear++; queue[rear]=node;

}

int dequeue(void)

{

int temp=queue[front];

if(front==0 && rear==0){front=rear=- 1;} else front++; return temp;

}

int main()

{int matrix[size][size]=

{

{0,1,1,0,0,0,1,0},

{1,0,1,1,1,0,0,0},

{1,1,0,0,1,0,0,0},

{0,1,0,0,0,1,0,1},

{0,1,1,0,0,1,1,0},

{0,0,0,1,1,0,0,0},

{1,0,0,0,1,0,0,1},

{0,0,0,1,0,0,1,0}

};

int i, j, k, start=5,temp,isthere; void enqueue(int); enqueue(start); while(!(front==- 1 && rear==-1))

{temp=dequeue(); printf("%d\t",temp); output[outputPtr++]=temp; if(outputPtr>=size)

break;

for(j=0; j<size; ++j)

{if(matrix[temp-1][j]==1)

{isthere=0;

for(k=0; k<outputPtr; ++k)

{if(output[k]==j+1)isthere=1

;} if(!(front==-1 && rear==- 1)) for(k=front; k<=rear;

++k)

{if(queue[k]==j+1)isthere=1;

}

if(isthere==0)enqueue(j+1);

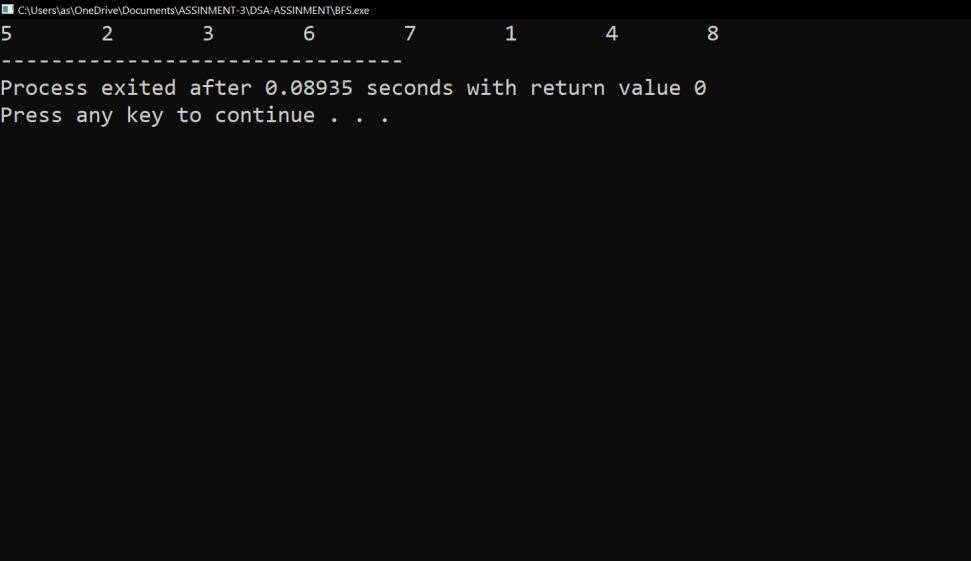
}

}

}

}

**OUTPUT:**



**Practical No. 11**

**Date:**

**֍ Aim:** Write a program to implement depth first search (DFS) graph

traversal algorithm**.**

# CODE:

#include <stdio.h> #define size 8

int output[size], outputPtr=0, stack[size], top=-1; void push(int node)

{

stack[++top]=node;

}

int pop(void)

{

return stack[top--];

}

int main()

{int matrix[size][size]={

{0,1,1,0,0,0,1,0},

{1,0,1,1,1,0,0,0},

{1,1,0,0,1,0,0,0},

{0,1,0,0,0,1,0,1},

{0,1,1,0,0,1,1,0},

{0,0,0,1,1,0,0,0},

{1,0,0,0,1,0,0,1},

{0,0,0,1,0,0,1,0}

};

int i, j, k, start=1,temp,isthere,alreadyAnElementPushed; void push(int); push(start);

while(top!=-1) {temp=pop(); printf("%d\t",temp); output[outputPtr++]=temp; if(outputPtr>=size)break; for(j=0; j<size; ++j)

{alreadyAnElementPushed= 0; if(matrix[temp-1][j]==1)

{isthere=0; for(k=0; k<outputPtr; ++k)

{if(output[k]==j+1)isthere=1;

} if(top!=-1) for(k=0; k<=top;

++k)

{if(stack[k]==j+1)isthere=1;} if(isthere==0) {push(j+1); alreadyAnElementPushed=1;

}

}

if(alreadyAnElementPushed==1) break;

}

}

}

# OUTPUT:

