Data Structures Lab File



Submitted By

PRATHMESH PATIL

BCA/II Semester

A71004819033

AMITY UNIVERSITY MAHARASHTRA

Amity Institute of Information Technology

|  |  |  |  |
| --- | --- | --- | --- |
| Exp no | Experiment name | Date | Sign |
| 1 | Write a Menu driven program to implement following array operations (i) Creation of the Array (ii) Deletion of an element from the kth position of the Array (iii)Insertion of an element to the kth position of the Array (iii) Linear Search for an element of the Array (iv)Reverse the elements of the Array (v)Merge two Sorted Arrays | 14/01/2020 |  |
| 2 | Write a Menu Driven Program to implement the following Sorting Techniques (i) Insertion Sort (ii) Bubble Sort (iii) Selection Sort | 21/01/2020 |  |
| 3 | Write a Menu driven program to implement stack using Arrays. Implement the following Stack Operations. (i) Display (ii) Push (iii)Pop | 28/01/2020 |  |
| 4 | Write a Menu driven program to implement Queues using Arrays. Implement the following Queue Operations. (i) Insertion (ii) Deletion | 4/02/2020 |  |
| 5 | Write a Menu driven program to implement Circular Queues using Arrays. | 11/02/2020 |  |
| 6 | Write a Menu driven program to implement Linked Lists. Implement the following operations. (i) Creation (ii) Insertion (iii)Deletion (iv) Display (v) Searching | 18/02/2020 |  |
| 7 | Write a Menu driven program to implement Doubly Linked Lists. Implement the following operations. (i) Creation (ii) Insertion (iii)Deletion (iv) Display | 25/02/2020 |  |
| 8 | Write a Menu driven program to implement stack using Linked Lists. Implement the following Stack Operations. (i) Display (ii) Push (iii)Pop | 24/03/2020 |  |
| 9 | Write a Menu driven program to implement Queues using Linked Lists. Implement the following Queue Operations. (i) Insertion (ii) Deletion | 31/03/2020 |  |
| 10 | Write a Program to convert Infix expression to Postfix expression using Stack | 7/04/2020 |  |
| 11 | Write a Menu Driven Program to implement the following Searching Techniques (i) Linear Search (ii) Binary Search | 7/04/2020 |  |
| 12 | Write a Menu Driven Program to implement the following Sorting Techniques (i) Merge Sort (ii) Quick Sort | 14/04/2020 |  |
| 13 | Write a Program to perform various operations on BST. | 22/04/2020 |  |

**Experiment no :-1**

**Aim :-** write a program in c to implement array that includes:-

1. Creation of an array
2. Deletion of an element in an array
3. Insertion of an element in an array
4. Searching for a particular element in an array
5. Display an array
6. Reverse an array
7. Merge of 2 array

**Theory :-**

An array is defined as a set of finite number of homogeneous elements or same data items.  An array is stored such that the position of each element can be computed from its index tuple by a mathematical formula. The simplest type of data structure is a linear array, also called one-dimensional array.

**Algorithm:-**

1. Traversing Linear Array

• Consider a Linear Array(LA) list given with lower bound(LB) and upper bound(UB) that contains 'n' number of elements. We can traverse the list with the given algorithm.

• (initialized counter) K = LB

• Repeat

• while k <= UB

• Process LA[k]

• K = K + 1(End of loop)

• Exit

1. Inserting into a Linear Array INSERT(LA,N,K,ITEM)

• Here LA is a linear array with N elements and K is a positive Integer such that K<=N. This algorithm inserts an element ITEM into the Kth position in LA

1.[Initialize counter] Set J:=N

2.Repeat step 3 and 4 while J>=K

3.[Move J th element Downwards ] Set LA[J+1] := LA[J]

4.[Decrease Counter] Set J=J-1 [end of step 2 loop]

5.[Insert Element] Set LA[K]:= ITEM

6.[Reset N] Set N:=N+1

7.EXIT

1. Deleting from a Linear Array DELETE(LA,N,K,ITEM)

• Here LA is a linear array with N elements and K is a positive Integer such that K<=N. This algorithm deletes an element ITEM from the Kth position in LA.

1. Set ITEM:=LA[K]

2. Repeat for J=K to N-1

[Move J+1st element upwards]

Set LA[J] :=LA[J+1]

[End of Loop]

3.[Reset the number N of elements in LA] Set N:=N-1

4. Exit

4. Searching an Array – Linear Search

(Linear Search) LINEAR (DATA,N,ITEM,LOC) • Here DATA is a linear array with N Elements and ITEM is a given item of information. This Algorithm finds the location LOC of ITEM in DATA or sets LOC:=0 if the search is unsuccessful.

1.[Insert ITEM at the end of Data] Set DATA[N+1]:=ITEM

2.[Initialize Counter] Set LOC:=1

3.[Search for ITEM] Repeat while DATA[LOC] ≠ ITEM Set LOC=LOC+1. [End of Loop]

4.[Successful ? ] If LOC=N+1 then Set LOC=0

5.Exit

5. Merging Algorithm

• MERGING (A,R,B,S,C) –Let Array A and B be sorted arrays respectively. This algorithm Merges A and B into C with N=R+S elements.

• 1.[Initialize] Set NA:=1,NB:=1,PTR:=1

• 2.[Compare] Repeat while NA<=R and NB<=S

• If A[NA]R Then • Repeat for K =0,1,2,3,4,5…S-NB:

• Set C[PTR+K] := B[NB+K]

• [end of Loop]

• Else

• Repeat for K =0,1,2,3,4,5…R-NA:

• Set C[PTR+K] := A[NA+K]

• [end of Loop]

• [End of If Structure]

• Exit

**Code:-**

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

void main()

{

void create(int arr[],int n);

void display(int arr[],int n);

int del(int arr[],int n,int num);

int insert(int arr[],int n,int num,int num2);

void search(int arr[],int n,int num);

void rev(int arr[],int b[],int n);

void merge(int arr[],int b[],int c[],int r,int s);

int arr[10],b[10],n,option,num,num2,ch=1,c[20],r,s,x;

A:

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

printf("\nMenu \n 1.Create a Array \n 2.Insert a ELEMENT \n 3.Delete a ELEMENT \n 4.Searching a element\n 5.Display a Array \n 6. reverse a Array \n 7.merge \n 8.Exit \n ENTER THE CHOICE:- ");

scanf("%d",&option);

switch(option)

{

case 1:

printf("\n ENTER THE SIZE OF ARRAY");

scanf("%d",&n);

create(arr,n);

printf("\n DO YOU WANT TO CONTINUE(y=1 or n=0) \n ENTER THE CHOICE:-");

scanf("%d",&ch);

if(ch==1)

goto A;

else

break;

case 2:

printf("\nenter the Number and index to insert");

scanf("%d %d",&num2,&num);

n=insert(arr,n,num,num2);

printf("THE INSERTION DONE SUCCESSFULLY!!!");

printf("\nDO YOU WANT TO CONTINUE(y=1 or n=0) \n ENTER THE CHOICE:-");

scanf("%d",&ch);

if(ch==1)

goto A;

else

break;

case 3:

printf("\nenter the index to delete");

scanf("%d",&num);

n=del(arr,n,num);

printf("THE ELEMENT DELETED SUCCESSFULLY!!!");

printf("\nDO YOU WANT TO CONTINUE(y=1 or n=0) \n ENTER THE CHOICE:-");

scanf("%d",&ch);

if(ch==1)

goto A;

else

break;

case 4:

printf("\nenter the Number to be searched");

scanf("%d",&num);

search(arr,n,num);

printf("\nDO YOU WANT TO CONTINUE(y=1 or n=0) \n ENTER THE CHOICE:-");

scanf("%d",&ch);

if(ch==1)

goto A;

else

break;

case 5:

display(arr,n);

printf("\nDO YOU WANT TO CONTINUE(y=1 or n=0) \n ENTER THE CHOICE:-");

scanf("%d",&ch);

if(ch==1)

goto A;

else

break;

case 6:

rev(arr,b,n);

display(b,n);

printf("\nDO YOU WANT TO CONTINUE(y=1 or n=0) \n ENTER THE CHOICE:-");

scanf("%d",&ch);

if(ch==1)

goto A;

else

break;

case 7:

merge(arr,b,c,r,s);

printf("\nDO YOU WANT TO CONTINUE(y=1 or n=0) \n ENTER THE CHOICE:-");

scanf("%d",&ch);

if(ch==1)

goto A;

else

break;

case 8:

printf("THANK YOU");

exit(0);

default:

printf("\nwrong choice !!!!!!!");

printf("\nDO YOU WANT TO CONTINUE(y or n) \n ENTER THE CHOICE:-");

scanf("%c",&ch);

}

}

void create(int arr[],int n)

{

int i=0;

printf("\nENTER THE NUMBER OF ARRAY");

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

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

printf("THE ARRAY CREATED SUCCESSFULLY!!!");

}

void display(int arr[],int n)

{

int i;

printf("\n The Number of ARRAY are \n");

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

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

printf("THE ARRAY DISPLAY SUCCESSFULLY!!!");

}

int del(int arr[],int n,int num)

{

int i;

if(n<num)

printf("\n index does not exist \n");

else

{

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

{

arr[i]=arr[i+1];

}

}

return n-1;

}

int insert(int arr[],int n,int num,int num2)

{

int i=n,x=num-1;

while(i>=x-1)

{

arr[i+1]=arr[i];

i--;

}

arr[num-1]=num2;

return n+1;

}

void search(int arr[],int n,int num)

{

int i=0,k,flag=0;

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

{

if(arr[i]==num)

{

flag=1;

k=i;

}

else

;

}

if(flag==1)

printf("\nELEMENT founded at %d\n",k+1);

else

printf("ELEMENT not founded");

}

void rev(int arr[],int b[],int n)

{

int i,j;

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

b[j]=arr[i];

}

void merge(int arr[],int b[],int c[],int r, int s)

{

int d[20];

printf("ENTER THE SIZE OF FIRST ARRAY");

scanf("%d",&r);

create(arr,r);

printf("ENTER THE SIZE OF SECOND ARRAY");

scanf("%d",&s);

create(b,s);

int i,j,x,temp;

x=r+s;

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

{

c[i]=arr[i];

}

for(j=r,i=0;i<s;i++,j++)

c[j]=b[i];

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

{

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

if(c[i]<c[j])

{

temp=c[i];

c[i]=c[j];

c[j]=temp;

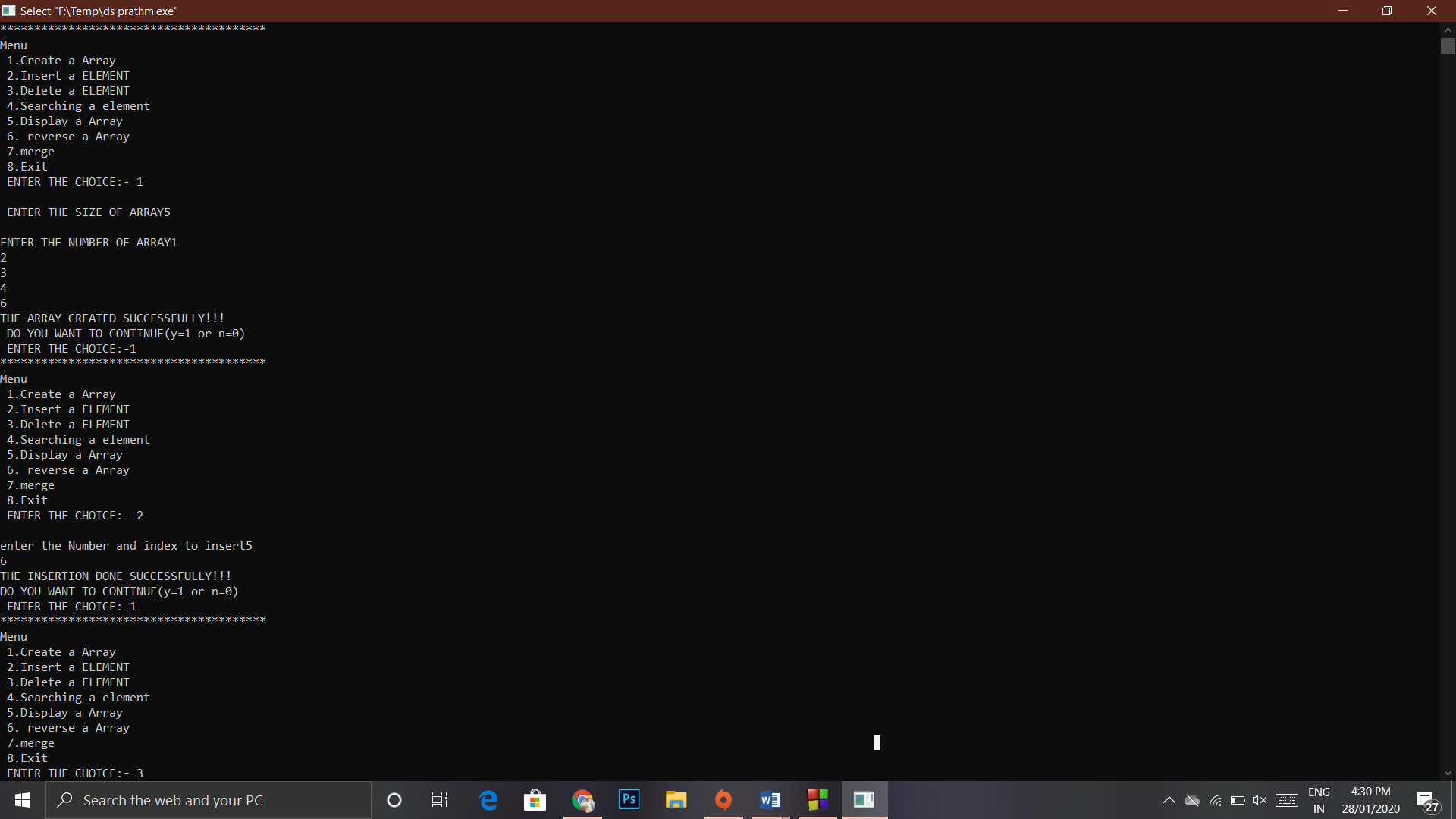
}

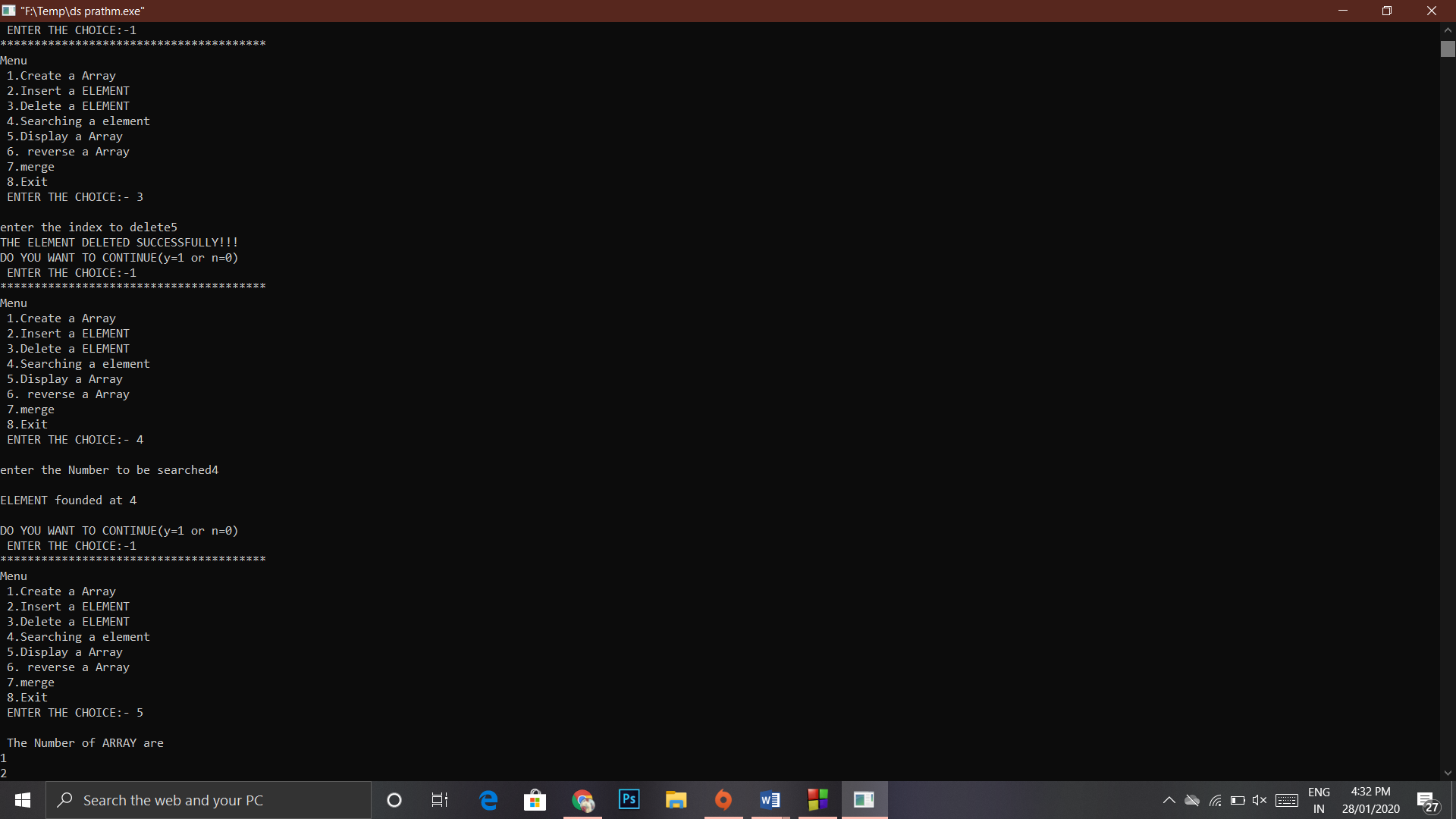
}

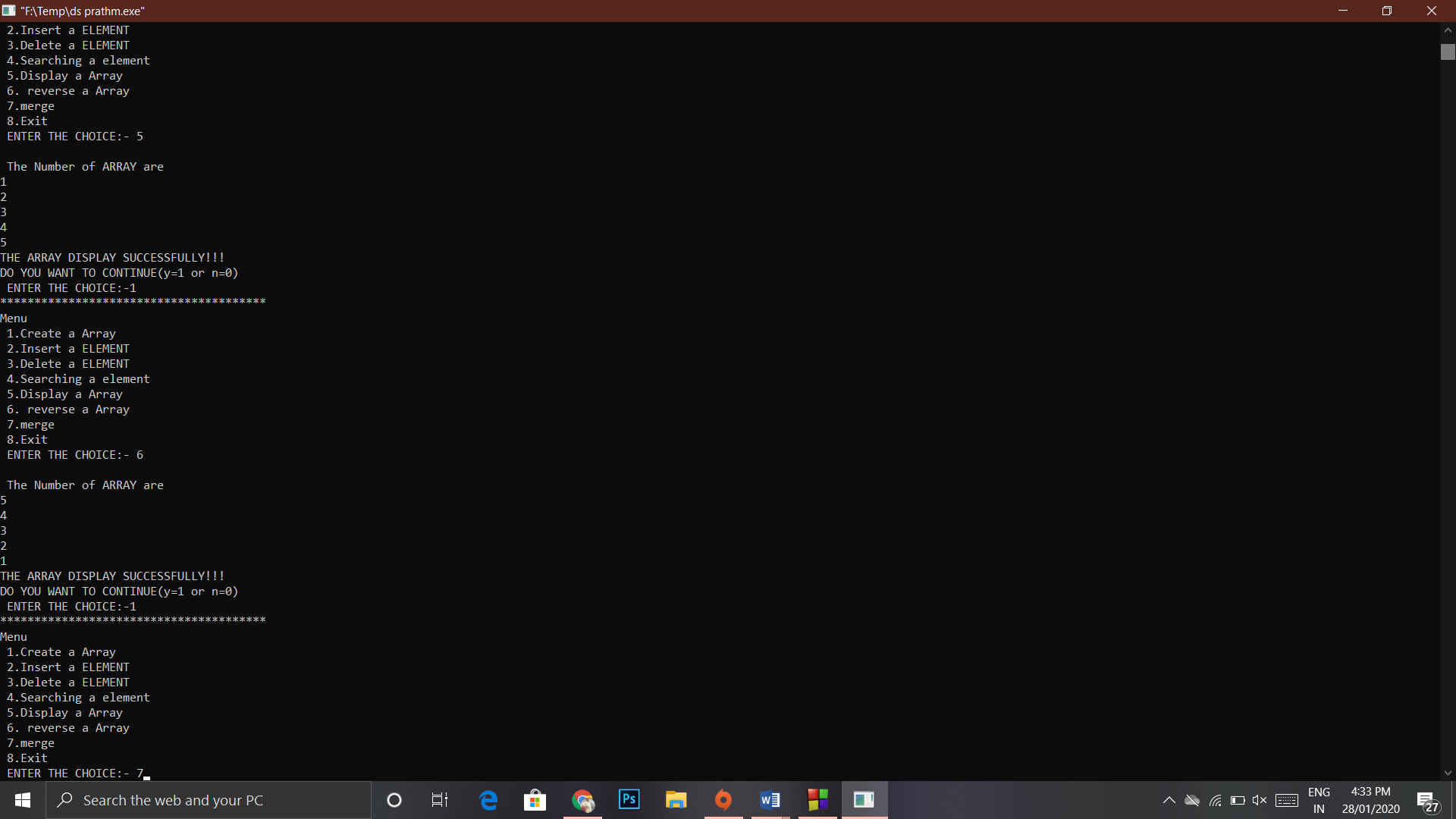
display(c,x);

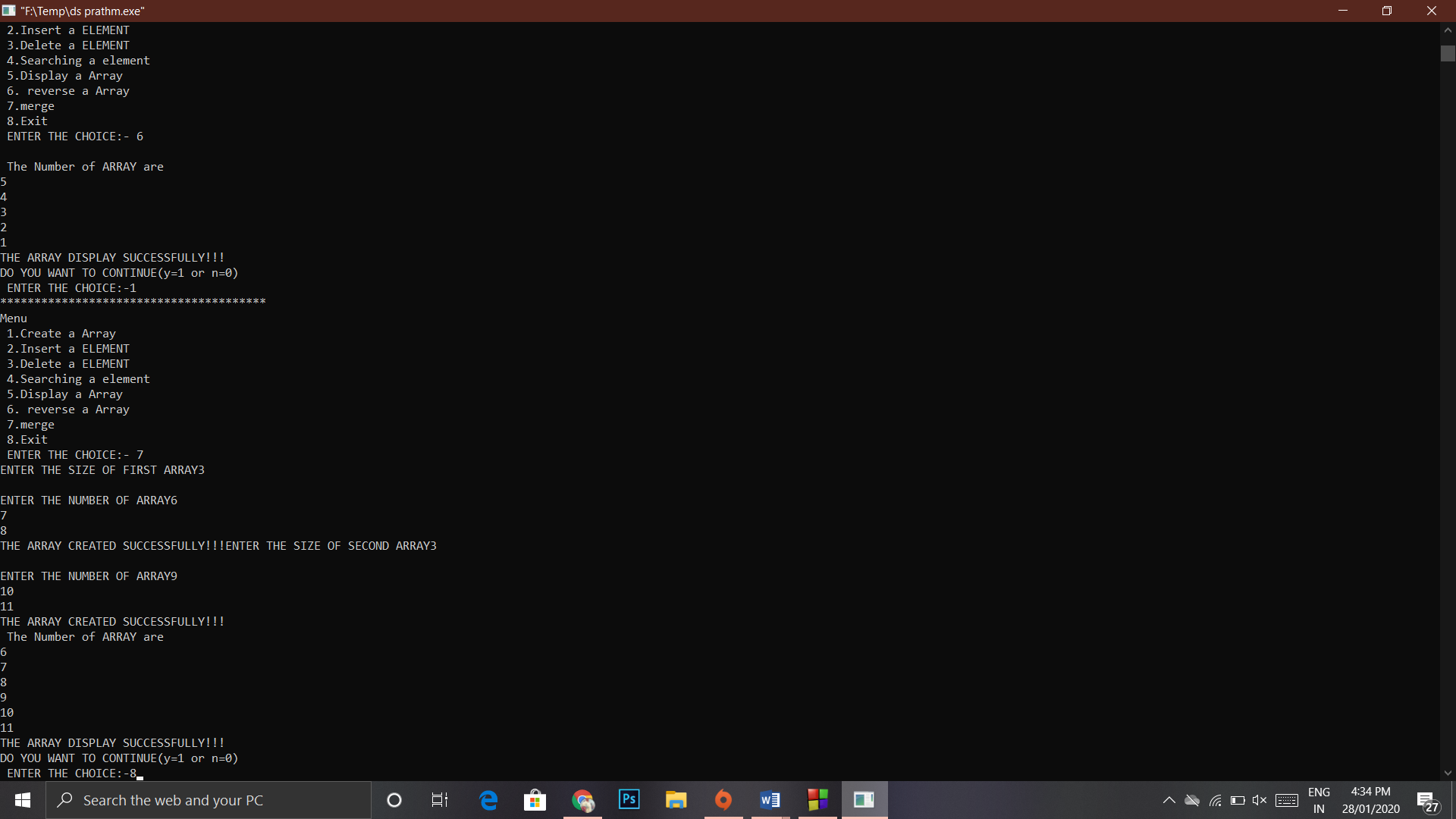
}

**Output:-**









**Experiment 2**

**Aim:-** write a menu driven program in c to implement sorting on array that includes:-

* Insertion sort
* Bubble sort
* Selection sort

**Theory:-**

a. Bubble Sort BUBBLE(DATA,N) – Here DATA is the array with N elements. This algorithm sorts the elements in DATA

1. Repeat Step 2 and 3 for K=1 to N-1

2. Set PTR:=1 [Initialize the Pass pointer]

3. Repeat while PTR<=N-K [Execute passes]

(a) If DATA[PTR]>DATA[PTR+1] then Interchange DATA[PTR] and DATA[PTR+1] [end of If Structure]

(b) Set PTR=PTR+1 [End of inner Loop] [End of Step 1 Outer Loop]

4. Exit

b. Selection Sort

• The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

• 1) The subarray which is already sorted.

2) Remaining subarray which is unsorted.

• In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

Selection Sort SELECTION(A,N) This algorithm Sorts the element A with N elements.

• 1.Repeat Step 2 and 3 for K =1,2,3…,N-1

• 2.Call MIN(A,K,N,LOC)

• 3.[Interchange A[K] and A[LOC]

• Set TEMP:=A[K];

• A[K]:=A[LOC]; • A[LOC]:=TEMP;

• [End of Step 1 Loop]

• 4.Exit

Selection Sort SELECTION(A,N) This algorithm Sorts the element A with N elements.

• 1.Repeat Step 2 and 3 for K =1,2,3…,N-1

• 2.Call MIN(A,K,N,LOC)

• 3.[Interchange A[K] and A[LOC]

• Set TEMP:=A[K];

• A[K]:=A[LOC];

• A[LOC]:=TEMP;

• [End of Step 1 Loop]

• 4.Exit

c.Insertion sort

• Insertion sort is based on the idea that one element from the input elements is consumed in each iteration to find its correct position i.e, the position to which it belongs in a sorted array

(Insertion Sort) INSERTION (A,N) -This Algorithm sorts the arrays A with N elements

1.Set A[0]:=-∞ [Initialize Sentinel Elements]

2.Repeat Step 3 to 5 for K =2,3,4..N

3. Set TEMP := A[K] and PTR=K-1

4. Repeat While TEMP

**Code:-**

#include<stdio.h>

#include<stdlib.h>

void create(int ar[],int n)

{

printf("Enter the %d elements :- ",n); for(int i=0;i<n;i++)

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

}

void insertionSort(int ar[], int n)

{ int i, key, j;

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

{ key = ar[i];

j = i - 1;

while (j >= 0 && ar[j] > key)

{ ar[j + 1] = ar[j];

j = j - 1;

}

ar[j + 1] = key;

}

}

void printArray(int ar[], int n)

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

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

printf("\n");

}

void swap(int \*xp, int \*yp)

{ int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selectionSort(int ar[], int n)

{

int i, j, min\_idx;

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

{

min\_idx = i;

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

if (ar[j] < ar[min\_idx])

min\_idx = j;

swap(&ar[min\_idx],

&ar[i]);

}

}

void bubbleSort(int arr[], int n)

{

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

{

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

{

if (arr[j] > arr[j+1]) swap(&arr[j], &arr[j+1]);

}

}

}

int main()

{

int ar[20],n,ch,k;

A:

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

printf("Menu:- \n 1. bubble sort \n 2. insertion sort \n 3. selection sort \n 4. Exit \n Enter your choice :-");

scanf("%d",&ch);

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

switch(ch)

{

case 1:

printf("Enter the size of array :- "); scanf("%d",&n);

create(ar,n);

bubbleSort(ar, n);

printf("Sorted array: \n"); printArray(ar, n);

printf("want to continue(y=1 or n=0):-");

scanf("%d",&k);

if(k==1)

goto A;

else

break;

case 2:

printf("Enter the size of array :- "); scanf("%d",&n);

create(ar,n);

insertionSort(ar, n);

printArray(ar, n);

printf("want to continue(y=1 or n=0):-");

scanf("%d",&k);

if(k==1)

goto A;

else

break;

case 3:

printf("enter the size of array :- "); scanf("%d",&n);

create(ar,n);

selectionSort(ar, n);

printf("Sorted array: \n"); printArray(ar, n);

printf("want to continue(y=1 or n=0):- ");

scanf("%d",&k);

if(k==1)

goto A;

else

break;

case 4:

exit(0);

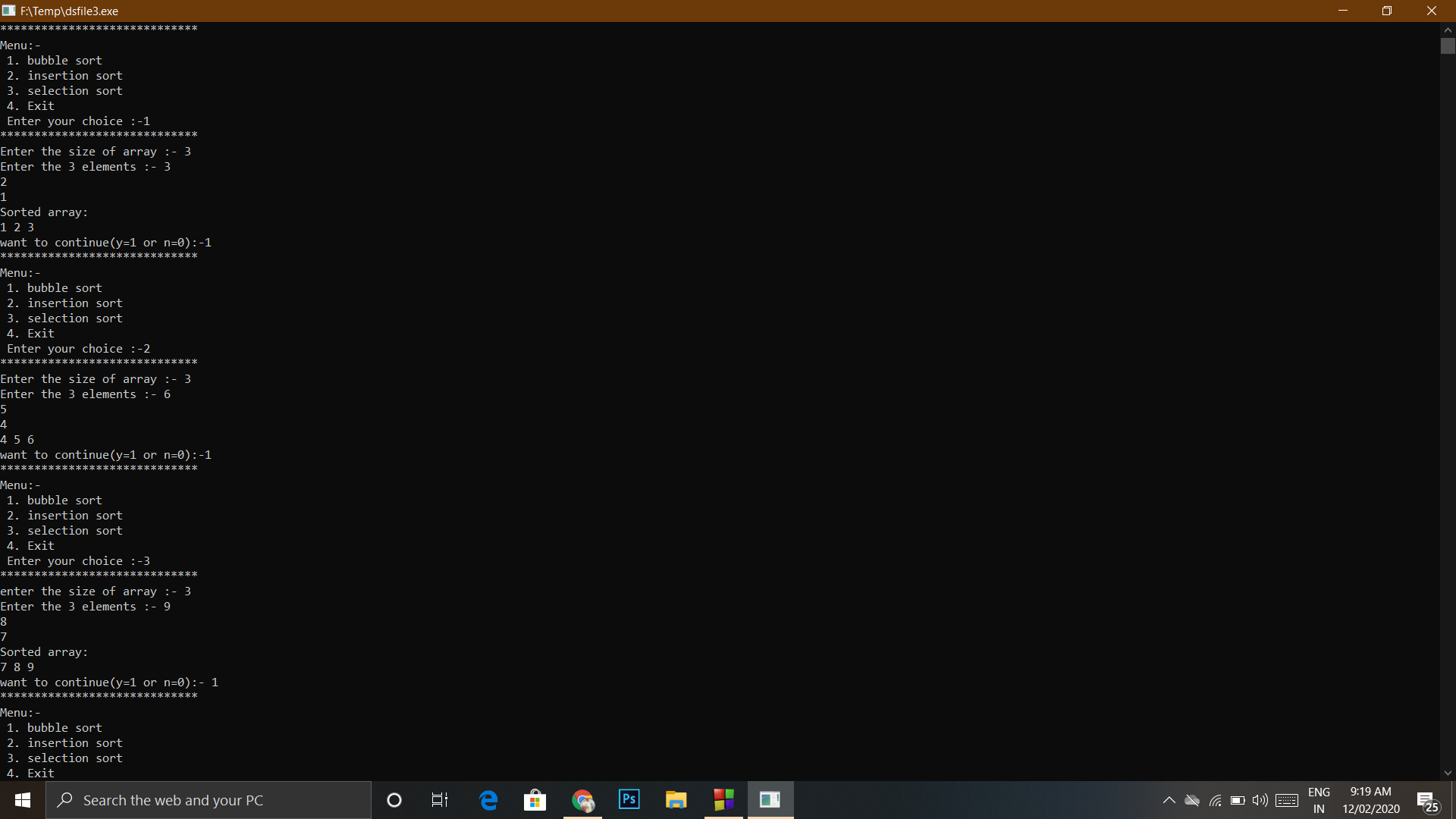
default:

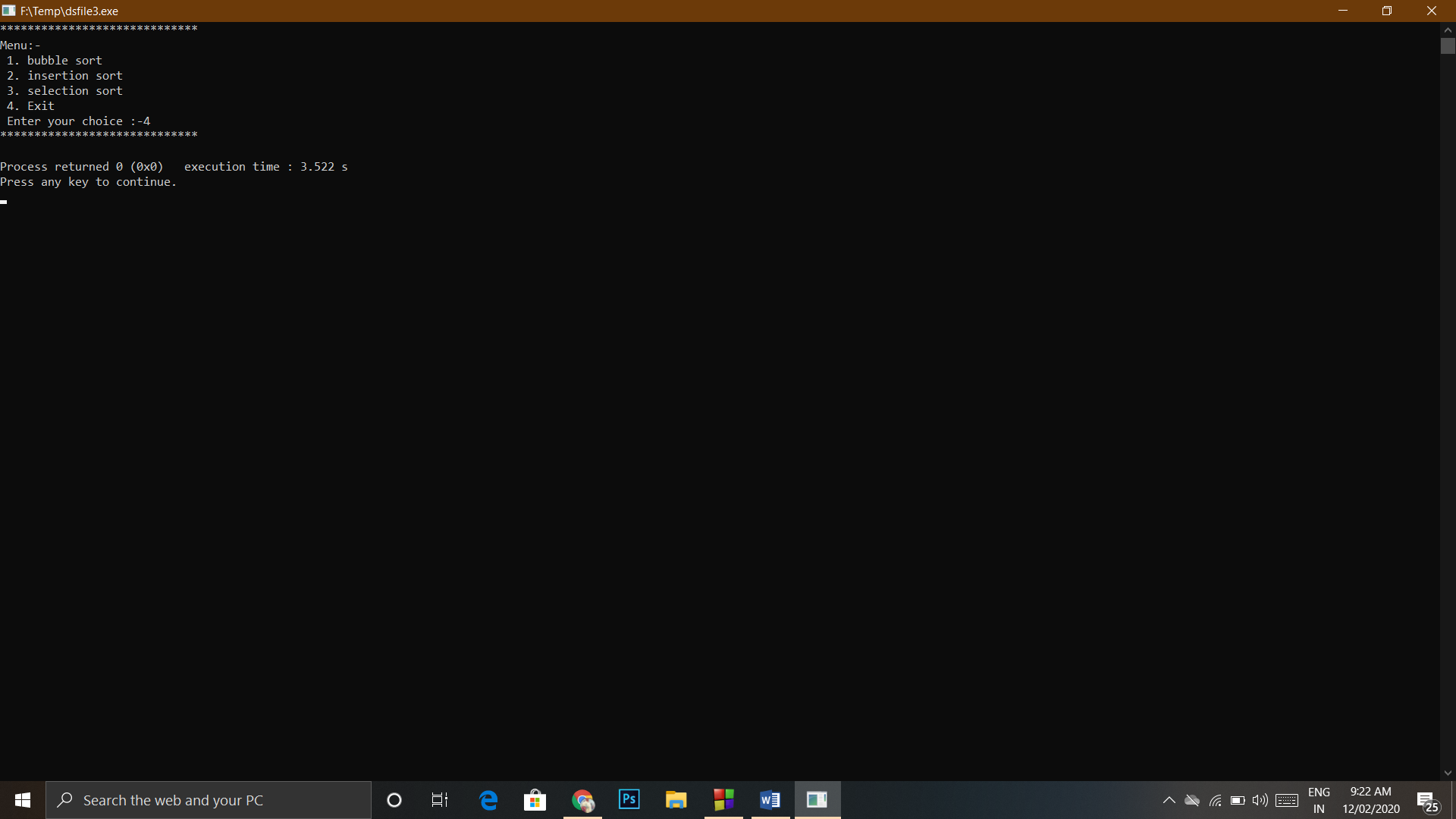
printf("wrong choice! !");

}

return 0; }

**Output:-**





**Experiment no 3**

**Aim :-** write a menu driven program in c to implement stack operation using array that includes

1. Push
2. Pop
3. Display

**Theory:-**

Push Operation

• The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps –

• Step 1 − Checks if the stack is full.

• Step 2 − If the stack is full, produces an error and exit.

• Step 3 − If the stack is not full, increments top to point next empty space.

• Step 4 − Adds data element to the stack location, where top is pointing.

• Step 5 − Returns success.

Pop Operation

• Accessing the content while removing it from the stack, is known as a Pop Operation

• A Pop operation may involve the following steps –

• Step 1 − Checks if the stack is empty.

• Step 2 − If the stack is empty, produces an error and exit.

• Step 3 − If the stack is not empty, accesses the data element at which top is pointing.

• Step 4 − Decreases the value of top by 1.

• Step 5 − Returns success.

**Code:-**

#include<stdio.h>

#include<stdlib.h>

#define MAX 10

void push(int stack[],int element);

void pop(int stack[]);

void display(int stack[]);

int top=-1;

void main()

{

int ch,stack[MAX],element,opt=1;

A:

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

printf("menu \n 1. push \n 2. pop \n 3.display \n 4. exit\n ");

scanf("%d",&ch);

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

switch(ch)

{

case 1:

printf("enter the element to be inserted :- ");

scanf("%d",&element);

if(top>=MAX)

printf("stack overflow \n");

else

push(stack,element);

printf("enter (1 for y) or (0 for n )=");

scanf("%d",&opt);

if(opt==1)

goto A;

else

break;

case 2:

if(MAX<=top)

printf("stack underflow \n");

else

pop(stack);

printf("enter (1 for y) or (0 for n )");

scanf("%d",&opt);

if(opt==1)

goto A;

else

break;

case 3:

display(stack);

printf("enter (1 for y) or (0 for n) ");

scanf("%d",&opt);

if(opt==1)

goto A;

else

break;

case 4:

exit(0);

default:

printf("wrong choice!!!!!!");

}

}

void push(int stack[],int element)

{

stack[top+1]=element;

top=top+1;

}

void pop(int stack[])

{

stack[top]=0;

top=top-1;

}

void display(int stack[])

{

if(top==-1)

printf("stack is empty");

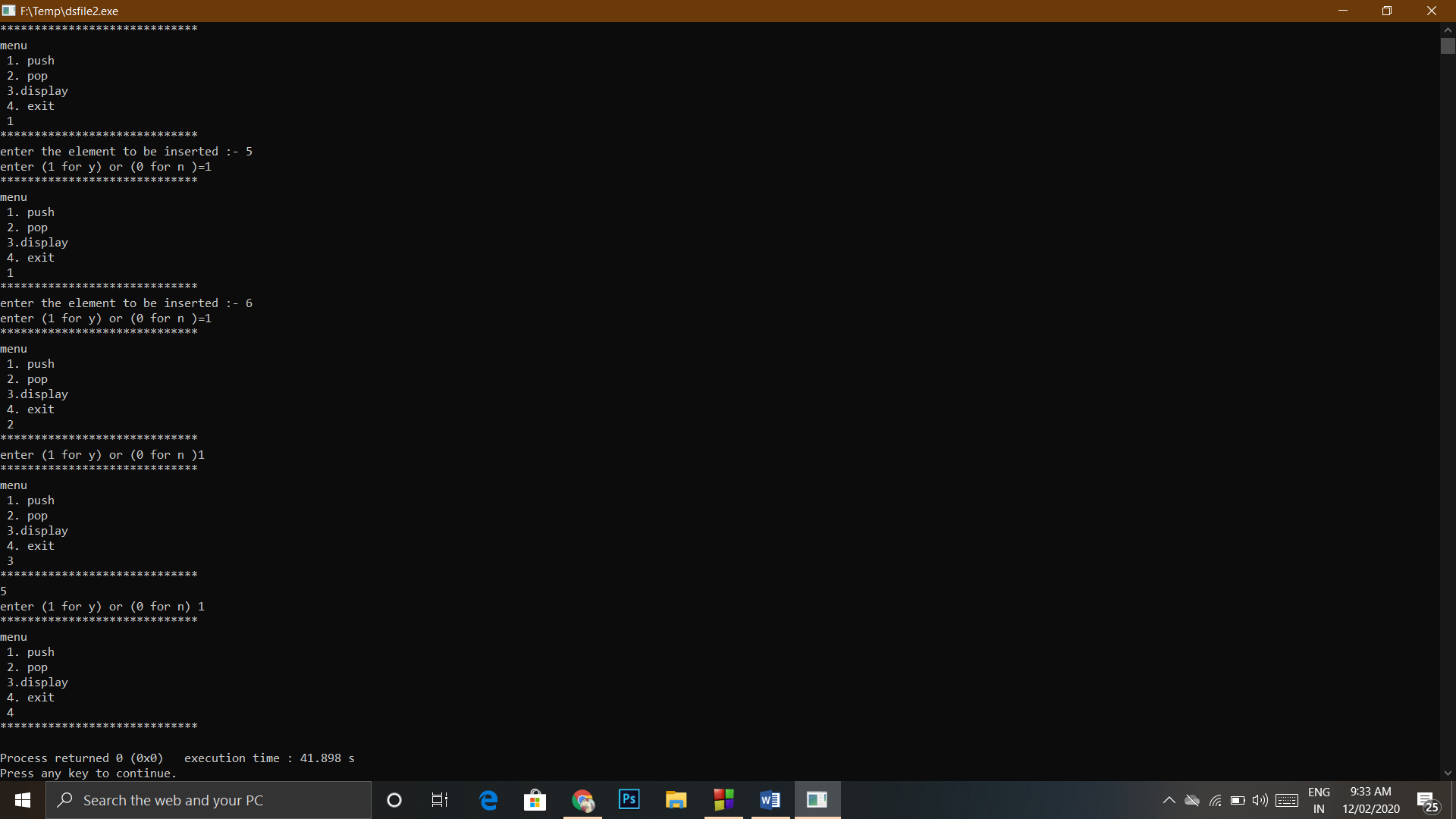
else

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

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

}

**Output:-**



**Experiment no :- 4**

**Aim :-** Write a menu driven program to implement the following operation on linear queue :-

1. Enqueue
2. Dequeue

Using arrays.

**Theory :-**

In queue, insertion and deletion happen at the opposite ends, so implementation is not as simple as stack.

To implement a queue using array, create an array arr of size n and take two variables front and rear both of which will be initialized to 0 which means the queue is currently empty. Element rear is the index upto which the elements are stored in the array and front is the index of the first element of the array.

Algorithm to insert any element in a queue :-

• Step 1: IF REAR = MAX - 1

Write OVERFLOW

Go to step 4

[END OF IF]

• Step 2: IF FRONT = -1 and REAR = -1

SET FRONT = REAR = 0

ELSE SET REAR = REAR + 1

[END OF IF]

• Step 3: Set QUEUE[REAR] = NUM

• Step 4: EXIT

Algorithm to delete any element in a queue :-

• Step 1: IF FRONT = -1 or FRONT > REAR

Write UNDERFLOW

ELSE SET VAL = QUEUE[FRONT]

SET FRONT = FRONT + 1

[END OF IF]

• Step 2: EXIT

**Code:-**

#include<stdio.h>

#include<stdlib.h>

int rear=-1;

int front=-1;

#define size 10

void main()

{

void Enqueue(int arr[],int ele);

void display(int arr[]);

void Dequeue(int arr[]);

int arr[size],n,ele,ch=1;

while(ch==1)

{

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

printf("Menu \n 1.Enqueue \n 2. Dequeue \n 3. display \n 4. Exit \n Enter your choice :- ");

scanf("%d",&n);

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

switch(n)

{

case 1:

printf("Enter the element to insert :- ");

scanf("%d",&ele);

Enqueue(arr,ele);

printf("\n Do you want to continue (Y=1 or N=0)");

scanf("%d",&ch);

if(ch==1)

continue;

else

break;

case 2:

Dequeue(arr);

printf("\n Do you want to continue (Y=1 or N=0)");

scanf("%d",&ch);

if(ch==1)

continue;

else

break;

case 3:

display(arr);

printf("\n Do you want to continue (Y=1 or N=0)");

scanf("%d",&ch);

if(ch==1)

continue;

else

break;

case 4:

exit(0);

default:

printf("wrong choice !!!");

}

}

}

void Enqueue(int arr[],int ele)

{

if(rear==size-1)

printf("The queue is overflow");

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

{

rear=front=0;

arr[rear]=ele;

}

else

{

rear=rear+1;

arr[rear]=ele;

}

}

void Dequeue(int arr[])

{

int val;

if(front==-1)

printf("queue underflow");

else if(front > rear)

printf("queue is underflow \n");

else

{

val=arr[front];

front=front+1;

}

}

void display(int arr[])

{

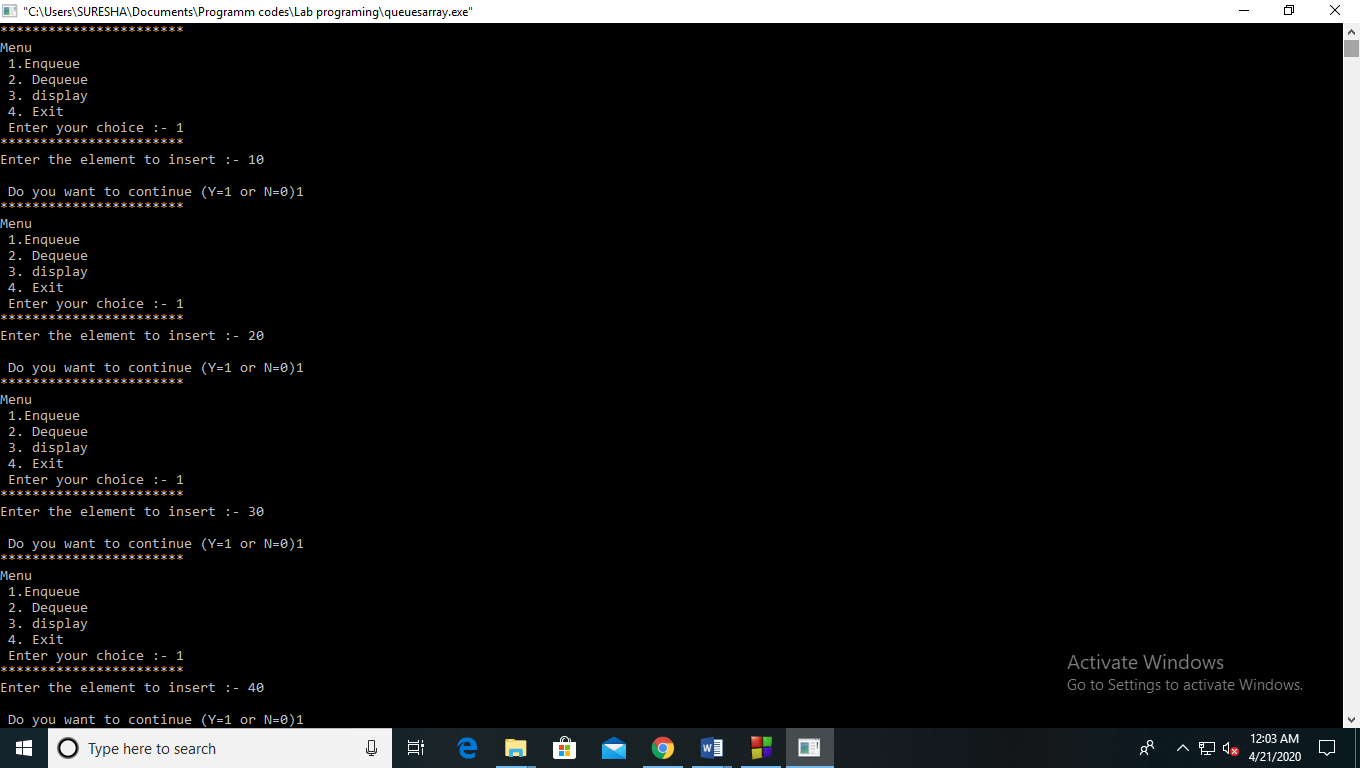
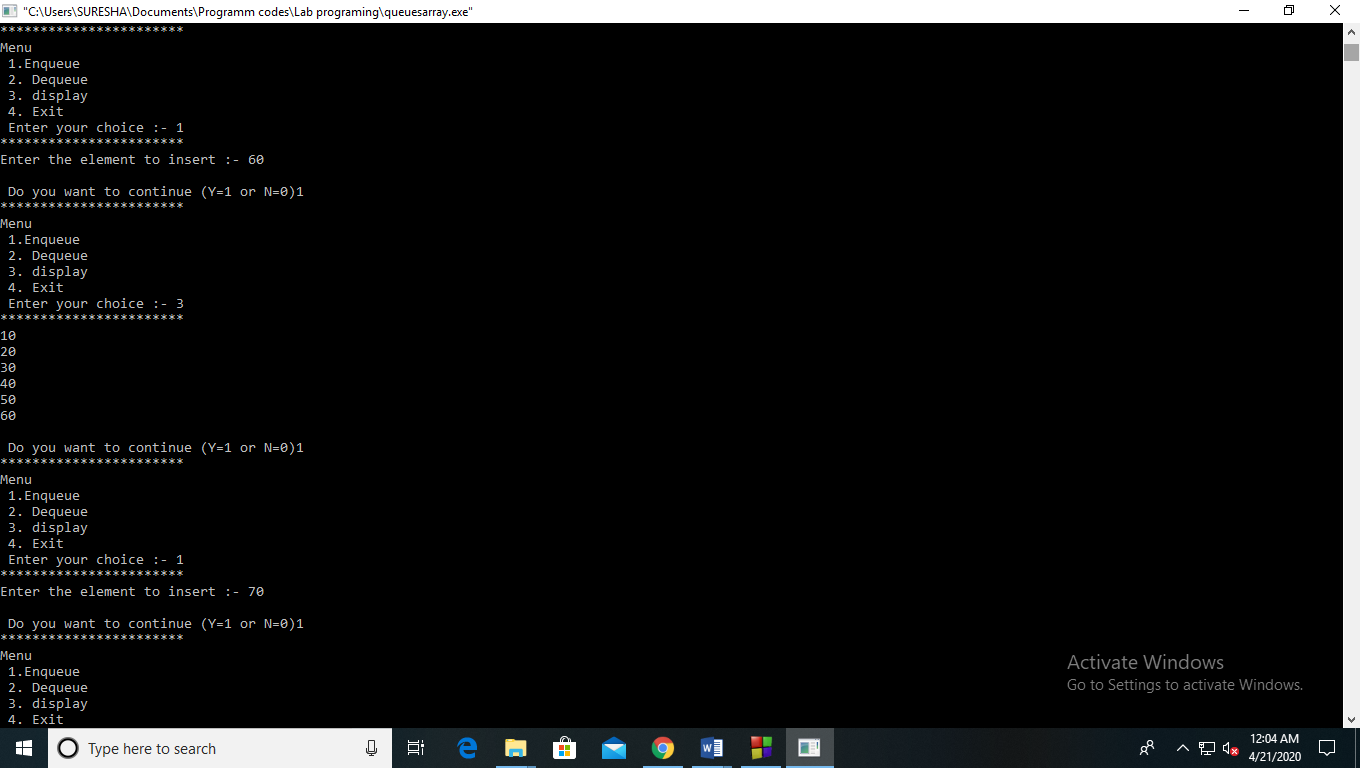
for(int i=front;i<=rear;i++)

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

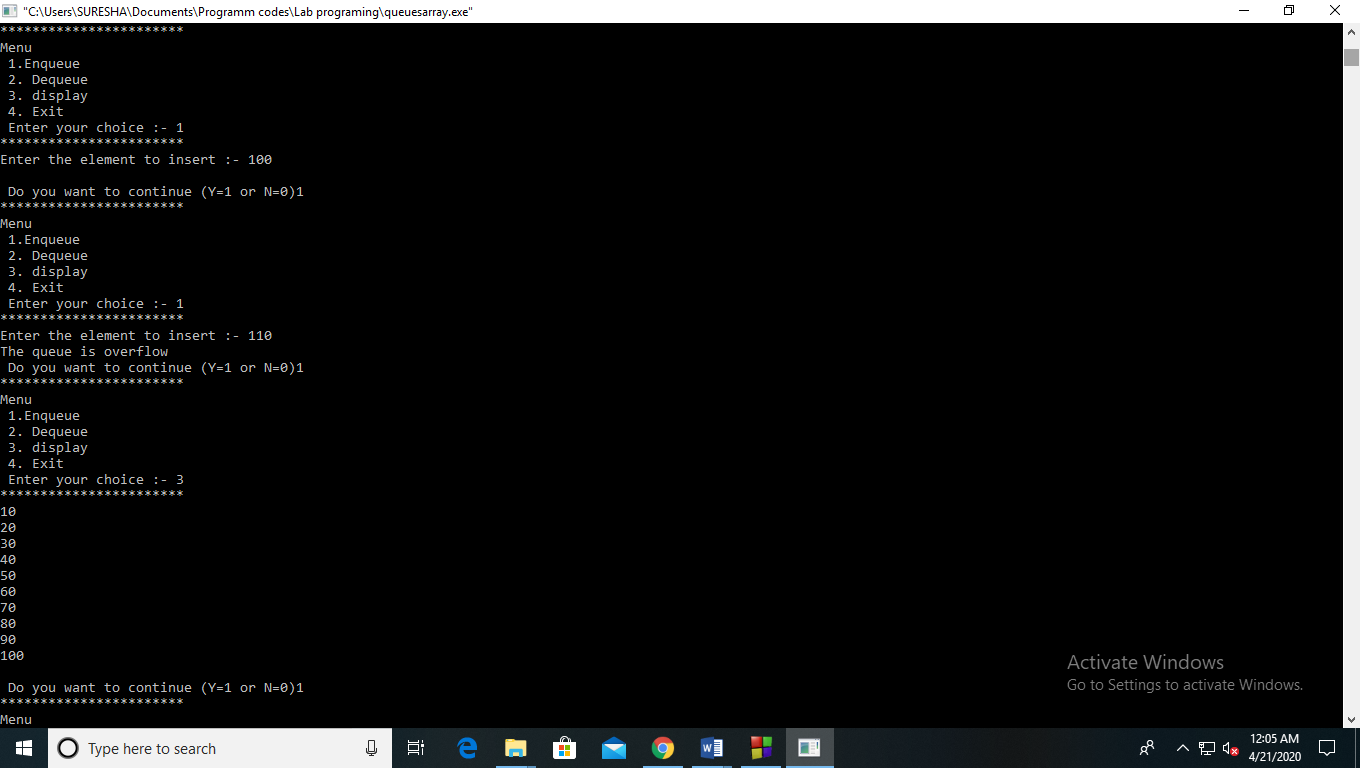
}

**Output:-**

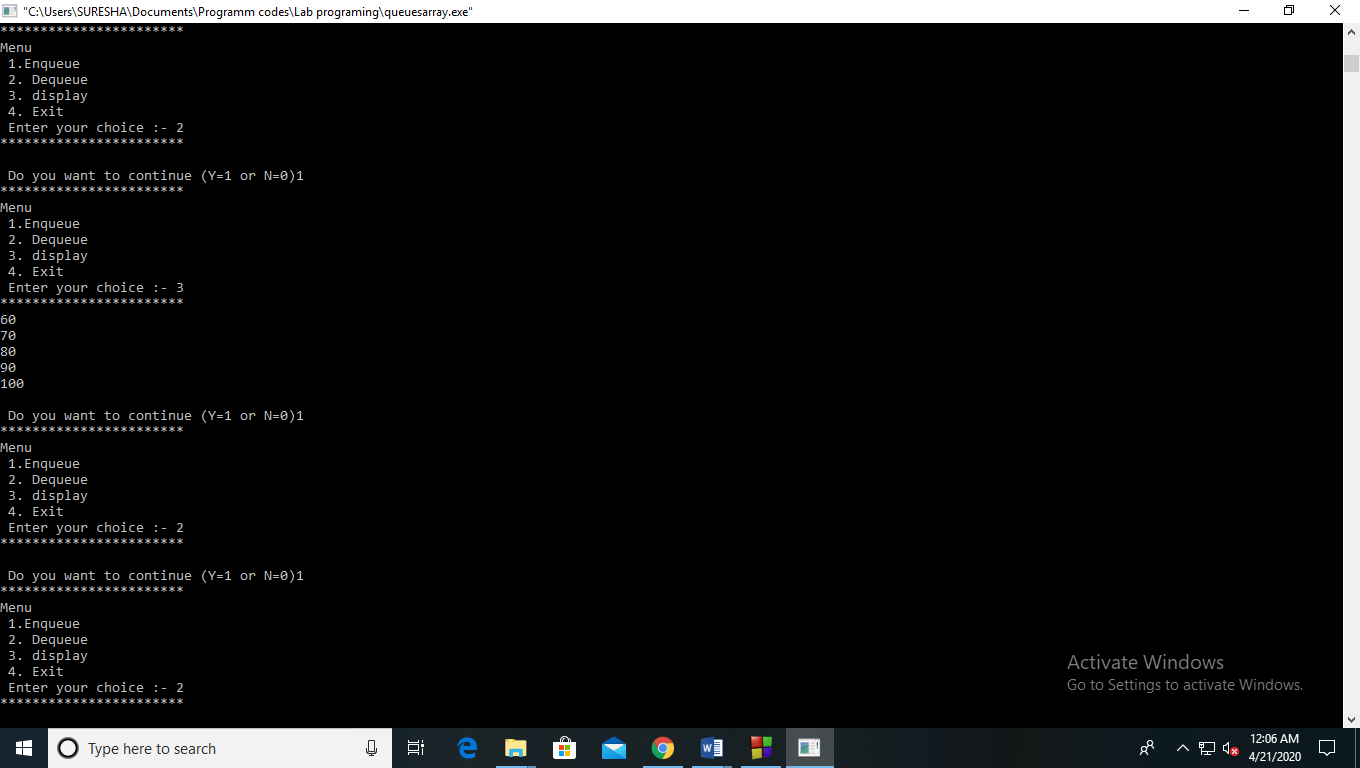
1)Enqueue



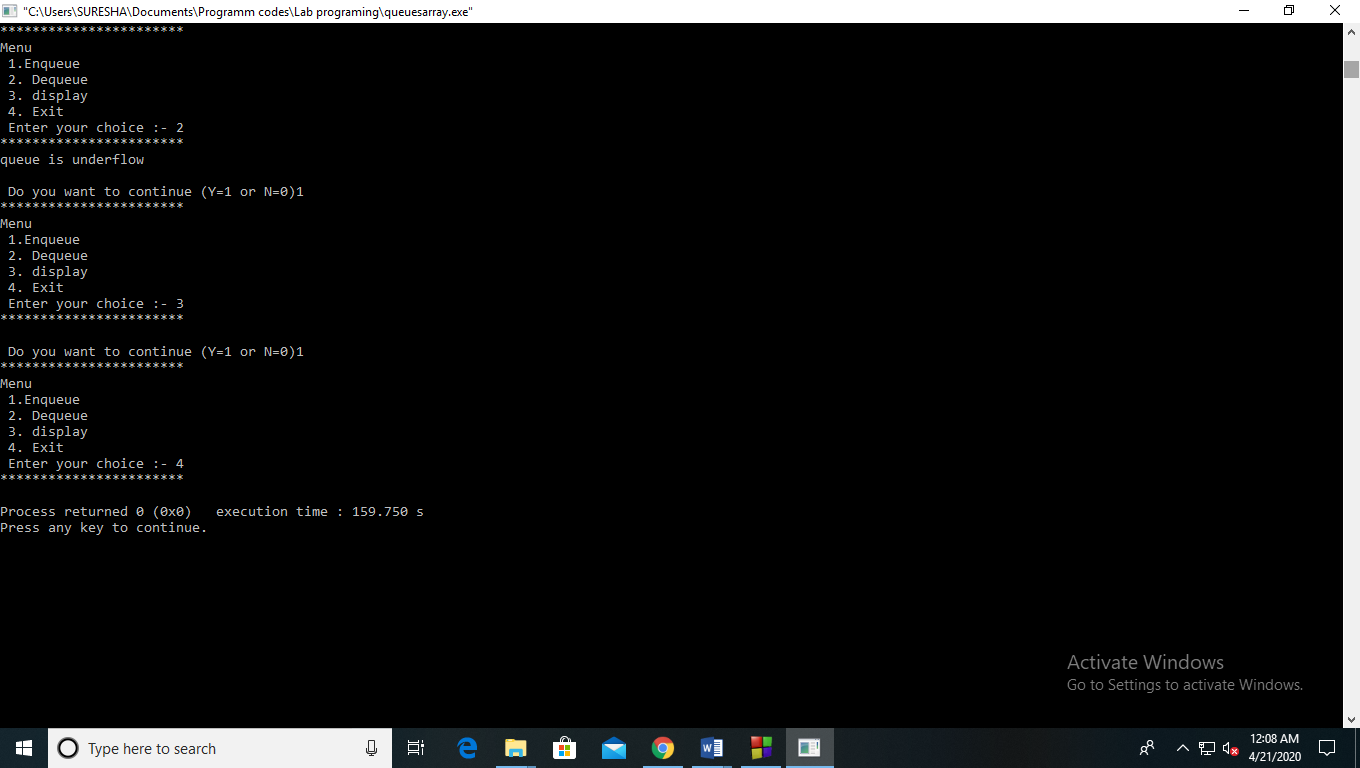
Overflow condition:-



2)Dequeue:-



Underflow condition:-



**Experiment no :- 5**

**Aim :-** Write a menu driven program to implement the following operations on circular queue

1. Insertion of a element
2. Deletion of a element

Using arrays.

**Theory :-**

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called **‘Ring Buffer’**.

Insertion in Circular queue

• There are three scenario of inserting an element in a queue.

• If (rear + 1)%maxsize = front,the queue is full. In that case, overflow occurs and therefore,

insertion can not be performed in the queue.

• If rear != max - 1, then rear will be incremented to the mod(maxsize) and the new value will be

inserted at the rear end of the queue.

• If front != 0 and rear = max - 1, then it means that queue is not full therefore, set the value of rear

to 0 and insert the new element there.

Algorithm to insert an element in circular queue

• Step 1: IF (REAR+1)%MAX = FRONT

Write " OVERFLOW "

Goto step 4

[End OF IF]

• Step 2: IF FRONT = -1 and REAR = -1

SET FRONT = REAR = 0

ELSE IF REAR = MAX - 1 and FRONT ! = 0

SET REAR = 0

ELSE

SET REAR = (REAR + 1) % MAX

[END OF IF]

• Step 3: SET QUEUE[REAR] = VAL

• Step 4: EXIT

Algorithm to delete an element from a circular queue

• To delete an element from the circular queue, we must check for the three following conditions.

• If front = -1, then there are no elements in the queue and therefore this will be the case of an

underflow condition.

• If there is only one element in the queue, in this case, the condition rear = front holds and

therefore, both are set to -1 and the queue is deleted completely.

• If front = max -1 then, the value is deleted from the front end the value of front is set to 0.

• Otherwise, the value of front is incremented by 1 and then delete the element at the front end

Algorithm Deletion of an element from a circular queue

• Step 1: IF FRONT = -1

Write " UNDERFLOW "

Goto Step 4

[END of IF]

• Step 2: SET VAL = QUEUE[FRONT]

• Step 3: IF FRONT = REAR

SET FRONT = REAR = -1

ELSE IF FRONT = MAX -1

SET FRONT = 0

ELSE SET FRONT = FRONT + 1

[END of IF]

[END OF IF]

• Step 4: EXIT

**Code:-**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int front = -1;

int rear = -1;

#define size 5

void main()

{

int cenqueue(int cqueue[],int ele);

void display(int cqueue[]);

int cdequeue(int cqueue[]);

int ele,ch=1,opt,ret,cqueue[size];

while(ch==1)

{

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

printf(" Menu : \n 1. enqueue \n 2. dequeue \n 3. display \n 4. Exit \n Enter your choice :- ");

scanf("%d",&opt);

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

switch(opt)

{

case 1:

printf("enter the element to be inserted :- ");

scanf("%d",&ele);

ret=cenqueue(cqueue,ele);

if(ret==-1)

printf(" Overflow !!!! \n");

else

printf(" insertion is succesfull ! \n ");

printf(" Do you want to continue (yes = 1 and no = 0 ) :- ");

scanf("%d",&ch);

if(ch==1)

continue;

else

break;

case 2:

ret=cdequeue(cqueue);

if(ret==-1)

printf(" underflow !!!! \n");

else

printf(" %d deleted from queue succesfull ! \n ",ret);

printf(" Do you want to continue (yes = 1 and no = 0 ) :- ");

scanf("%d",&ch);

if(ch==1)

continue;

else

break;

case 3:

display(cqueue);

printf(" Do you want to continue (yes = 1 and no = 0 ) :- ");

scanf("%d",&ch);

if(ch==1)

continue;

else

break;

case 4:

exit(0);

default :

printf("wrong choice !!!");

}

}

}

int cenqueue(int cqueue[],int ele)

{

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

{

rear=front=0;

}

else if(rear == size-1 && front != 0)

{

rear=0;

}

else if(rear<size-1 )

{

rear=rear+1;

}

else if(rear==size-1 && front==0 )

{

return -1;

}

cqueue[rear]=ele;

return 0;

}

int cdequeue(int cqueue[])

{

int item;

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

{

item=cqueue[front];

front=rear=-1;

}

else if(front==size-1)

{

item=cqueue[front];

front=0;

}

else if(front<size-1 && front != -1)

{

item=cqueue[front];

front=front+1;

}

else

{

return -1;

}

return item;

}

void display(int cqueue[])

{

int i;

if(front<=rear)

{

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

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

}

else

{

for(i=front;i<=size-1;i++)

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

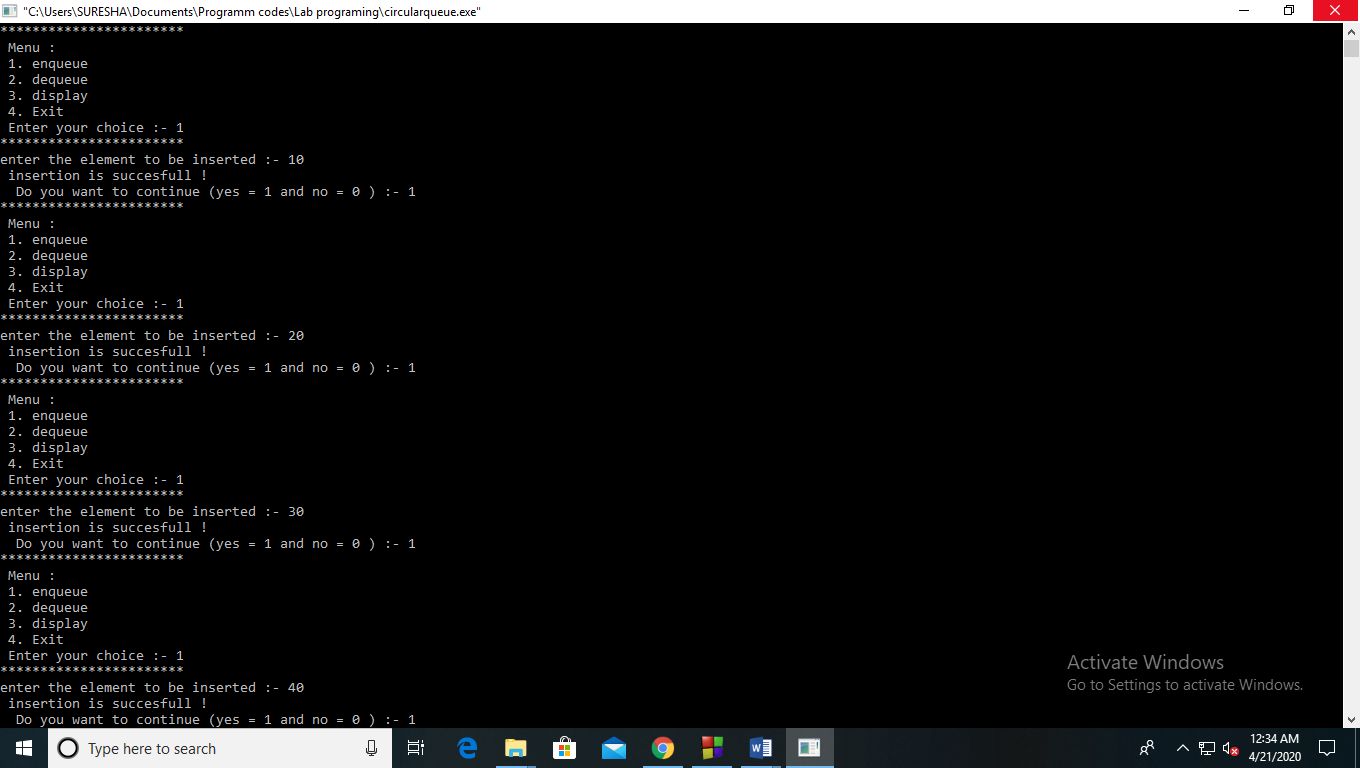
for(i=0;i<=rear;i++)

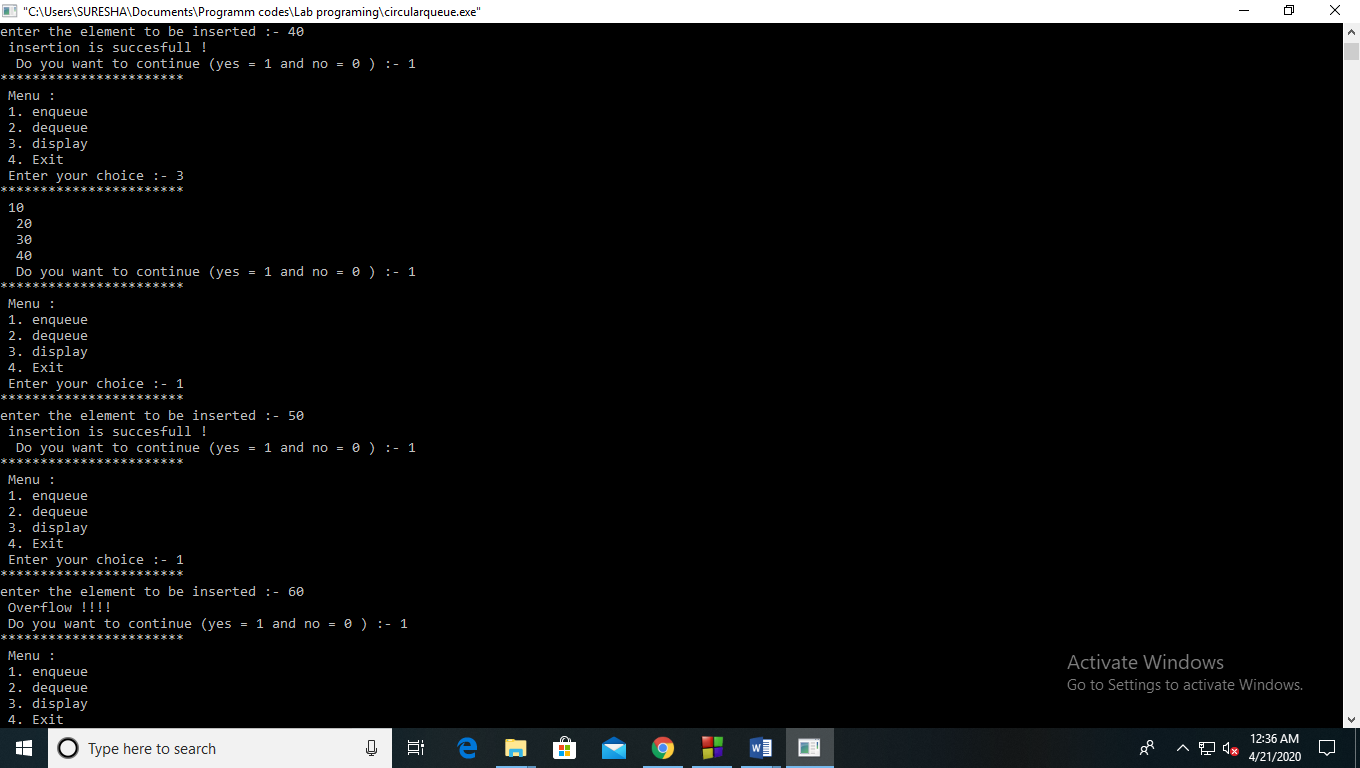
printf(" %d \n",cqueue[i]);

}

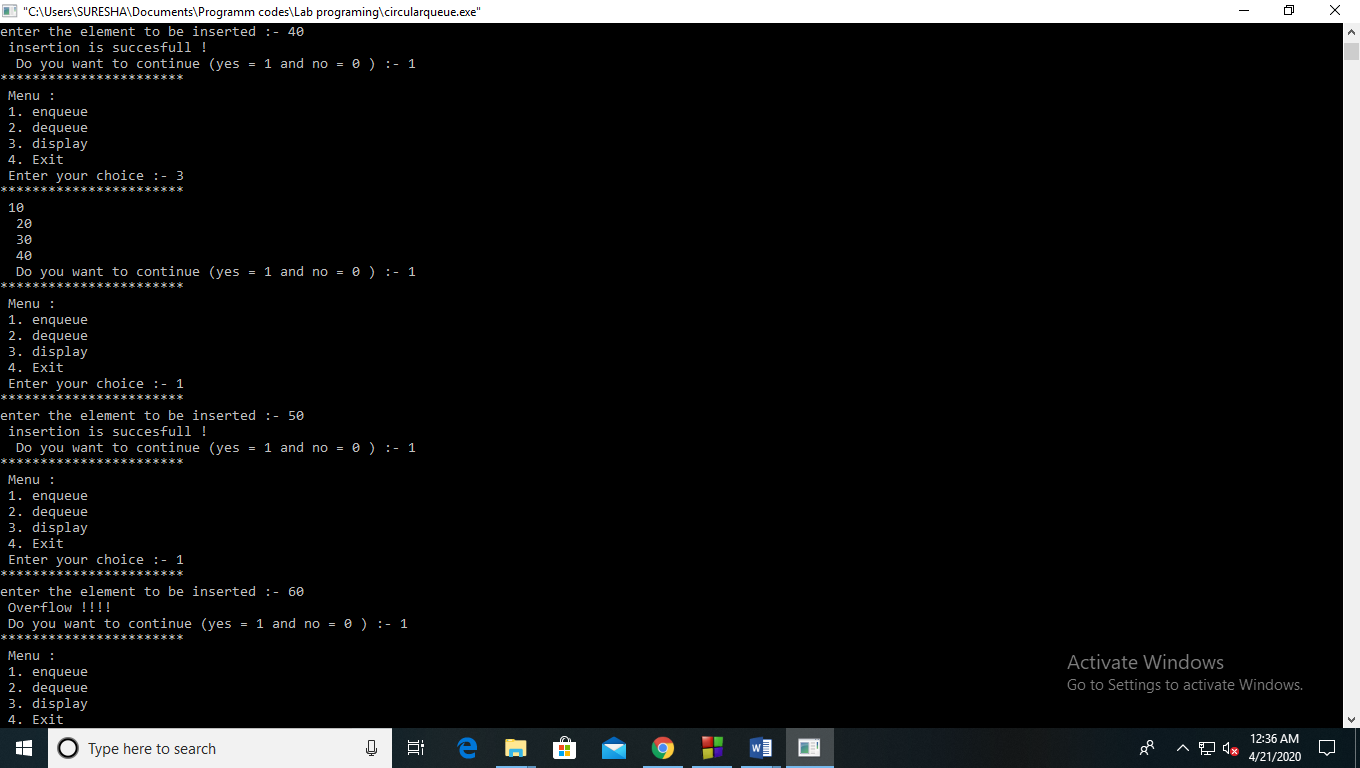
}

**Output:-**

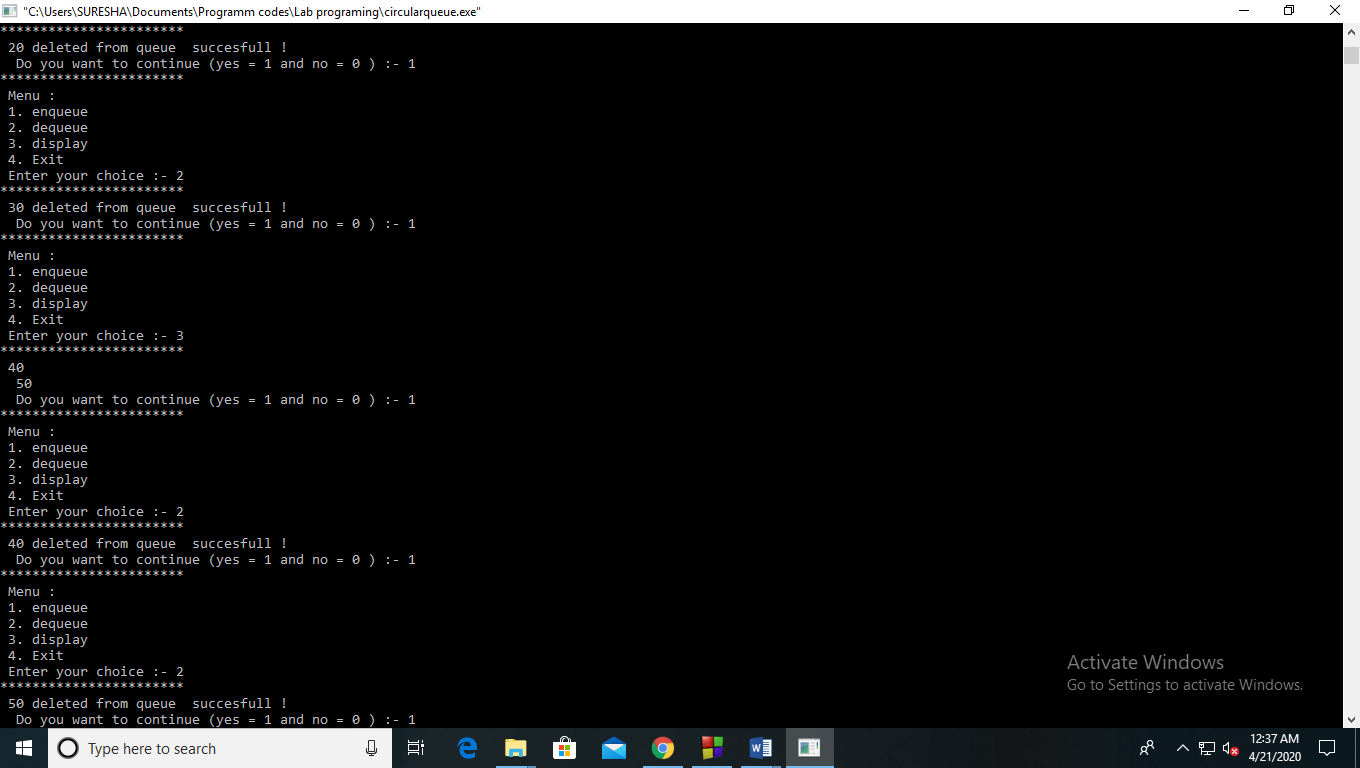
1)Enqueue:- 



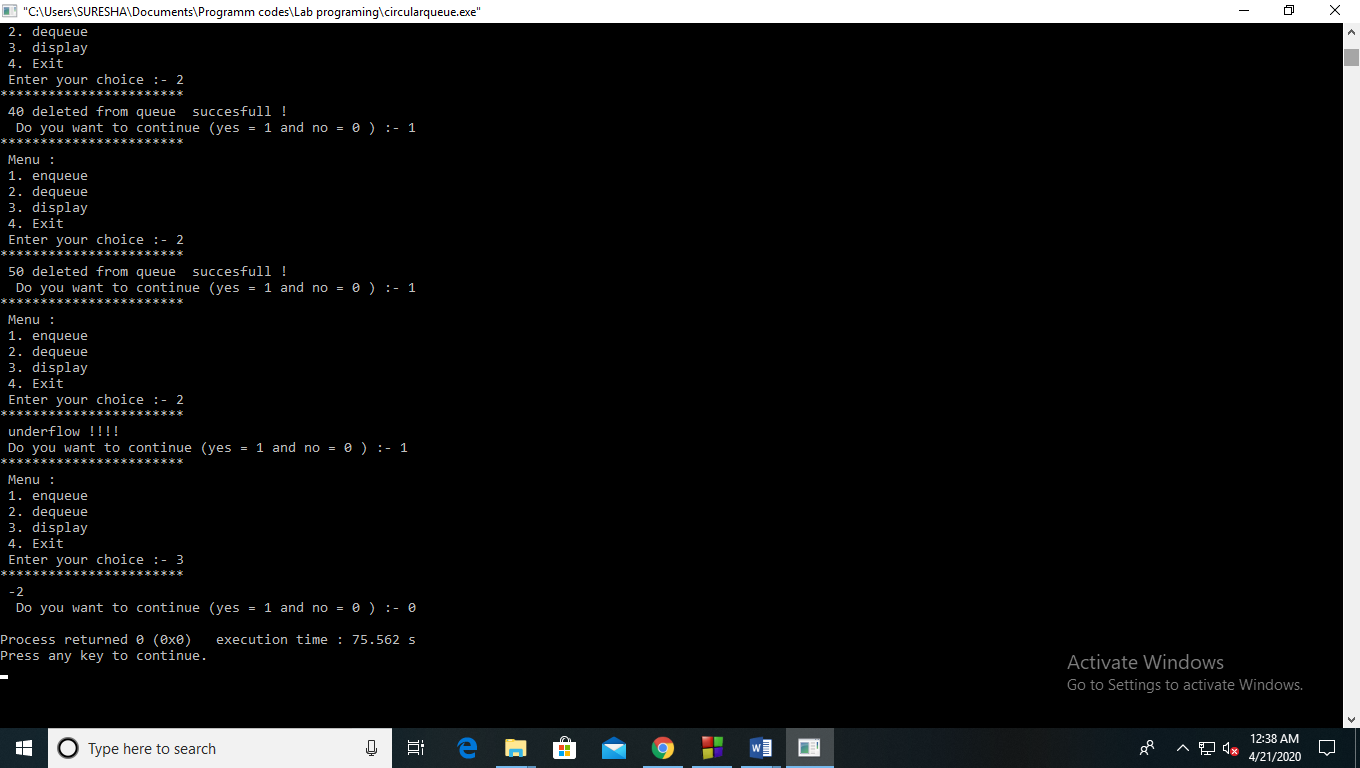
Overflow condition:-



2)Dequeue:-



Underflow condition:-



**Experiment 6**

**Aim:-** Write a menu driven programme to implement linked list. Implement the following

Addition at the start.

Addition at the end.

Addition in between.

Deletion at the start.

Deletion at the end.

Deletion in between.

**Theory:-** A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers.

A node can be added in three ways  
**1)** At the front of the linked list  
**2)**After a given node.  
**3)** At the end of the linked list.

Addition of a node at the start

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

{

    /\* 1. allocate node \*/

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

    /\* 2. put in the data  \*/

    new\_node->data  = new\_data;

    /\* 3. Make next of new node as head \*/

    new\_node->next = (\*head\_ref);

    /\* 4. move the head to point to the new node \*/

    (\*head\_ref)    = new\_node;

}

Addition of  node After a given node.

void insertAfter(struct Node\* prev\_node, int new\_data)

{

    /\*1. check if the given prev\_node is NULL \*/

    if (prev\_node == NULL)

    {

       printf("the given previous node cannot be NULL");

       return;

    }

    /\* 2. allocate new node \*/

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

    /\* 3. put in the data  \*/

    new\_node->data  = new\_data;

    /\* 4. Make next of new node as next of prev\_node \*/

    new\_node->next = prev\_node->next;

    /\* 5. move the next of prev\_node as new\_node \*/

    prev\_node->next = new\_node;

}

Addition of a node at the end

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

{

    /\* 1. allocate node \*/

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

    struct Node \*last = \*head\_ref;  /\* used in step 5\*/

      /\* 2. put in the data  \*/

    new\_node->data  = new\_data;

      /\* 3. This new node is going to be the last node, so make next

          of it as NULL\*/

    new\_node->next = NULL;

      /\* 4. If the Linked List is empty, then make the new node as head \*/

    if (\*head\_ref == NULL)

    {

       \*head\_ref = new\_node;

       return; }

    /\* 5. Else traverse till the last node \*/

    while (last->next != NULL)

        last = last->next;

    /\* 6. Change the next of last node \*/

    last->next = new\_node;

    return;

}

Deletion Operation – from the beginning of the Linked List

•      Deleting a node from a linked list is straight-forward, but there are some cases that you need to account for:

•      the list is empty; or

•      the node to remove is the only node in the linked list; or

•      we are removing the head node; or

•      we are removing the tail node; or

•      the node to remove is somewhere in between the head and tail; or

•      the item to remove doesn't exist in the linked list

**Code:-**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

void display(struct node \*head);

struct node\* insert\_front(struct node \*new\_head,int val);

struct node\* insert\_end(struct node \*new\_end,int val);

struct node\* insert\_between(struct node \*new\_head,int val);

struct node\* delete\_end(struct node\* head);

struct node\* delete\_front(struct node\* head );

struct node\* delete\_between(struct node \*new\_head);

void search(struct node\* new\_head,int n);

int main()

{

int val;

struct node \*head=NULL;

struct node \*second=NULL;

struct node \*third=NULL;

struct node \*p=NULL;

struct node \*ptr=NULL;

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

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

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

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

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

head->data=1;

head->next=second;

second->data=2;

second->next=third;

third->data=3;

third->next=NULL;

display(head);

int ch,opt=1,opt1,n;

while(opt==1)

{

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

printf("Main Menu \n 1.insertion \n 2. deletion \n 3.searching \n Enter your choice :- ");

scanf("%d",&ch);

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

switch(ch)

{

case 1:

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

printf("Menu of insertion \n 1. insertion in front \n 2. insertion in end \n 3. insertion in between \n Enter your choice :-");

scanf("%d",&opt1);

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

if(opt1==1)

{

printf("Enter the value to be inserted in the front :- ");

scanf("%d",&val);

head=insert\_front(head,val);

printf("after inserting in front \n");

display(head);

}

else if(opt1==2)

{

printf("Enter the value to be inserted in the end :- ");

scanf("%d",&val);

third->next=insert\_end(head,val);

printf("after inserting in End \n");

display(head);

}

else if(opt1==3)

{

printf("Enter the value to be inserted in the between :- ");

scanf("%d",&val);

head=insert\_between(head,val);

printf("after inserting in between \n");

display(head);

}

else

printf("wrong choice!!!");

printf("Do you want to continue (Y=1 or N=0)");

scanf("%d",&opt);

case 2:

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

printf("Menu of deletion \n 1. deletion in front \n 2. deletion in end \n 3. deletion in between \n Enter your choice :-");

scanf("%d",&opt1);

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

if(opt1==1)

{

head=delete\_front(head);

printf(" \n after deleting front \n ");

display(head);

}

else if(opt1==2)

{

head=delete\_end(head);

printf(" \n after deleting end \n ");

display(head);

}

else if(opt1==3)

{

head=delete\_between(head);

printf("after deleting between element \n");

display(head);

}

else

printf("wrong choice!!!");

printf("Do you want to continue (Y=1 or N=0)");

scanf("%d",&opt);

case 3:

printf("enter the element to be searched :- ");

scanf("%d",&n);

search(head,n);

printf("Do you want to continue (Y=1 or N=0)");

scanf("%d",&opt);

default:

printf("wrong choice !!");

break;

}

}

return 0;

}

void display(struct node \*head)

{

printf("Linked list is :- \n");

while(head!=NULL)

{

printf(" %d ",head->data);

head=head->next;

}

printf("\n");

}

struct node\* insert\_front(struct node \*new\_head,int val)

{

struct node \*new\_node=NULL;

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

new\_node->data=val;

new\_node->next=new\_head;

return(new\_node);

}

struct node\* insert\_end(struct node \*new\_end,int val)

{

struct node \*p=NULL;

struct node \*new\_node1=NULL;

new\_node1=(struct node\*)malloc(sizeof(struct node));

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

p=new\_end;

while(new\_end!=NULL)

{

printf(" %d ",new\_end->data);

new\_end=new\_end->next;

}

new\_node1->data=val;

new\_node1->next=NULL;

new\_end=new\_node1;

return(new\_end);

}

struct node\* insert\_between(struct node \*new\_head,int val)

{

int ch,i;

printf("enter where you to insert :- ");

scanf("%d",&ch);

struct node \*p=NULL;

struct node \*new\_node1=NULL;

struct node \*ptr=NULL;

new\_node1=(struct node\*)malloc(sizeof(struct node));

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

p=new\_head;

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

ptr=new\_head;

for(i=1;i<=ch-1;i++)

{

p=p->next;

}

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

{

ptr=ptr->next;

}

new\_node1->data=val;

p->next=new\_node1;

new\_node1->next=ptr;

return(new\_head);

}

struct node\* delete\_front(struct node\* head )

{

struct node \*ptr=NULL;

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

ptr=head;

head=ptr->next;

return (head);

}

struct node\* delete\_end(struct node\* head)

{

struct node \*ptr=head;

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

//ptr=head;

int l=0;

while(ptr!=NULL)

{

ptr=ptr->next;

l++;

}

ptr=head;

for(int i=1;i<l-1;i++)

{

ptr=ptr->next;

}

ptr->next=NULL;

return head;

}

struct node\* delete\_between(struct node \*new\_head)

{

struct node \*ptr=new\_head;

struct node \*p=new\_head;

int l=0,i,m,n=0;

printf("which position element you want to delete ?");

scanf("%d",&m);

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

{

ptr=ptr->next;

l++;

}

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

{

p=p->next;

n++;

}

ptr=ptr->next;

p->next=ptr;

return new\_head;

}

void search(struct node\* new\_head,int n)

{

int i=1;

struct node \*p=new\_head;

while(p!=NULL)

{

if(p->data==n)

{

printf("element founded at %d\n",i);

break;

}

else

{

p=p->next;

i++;

}

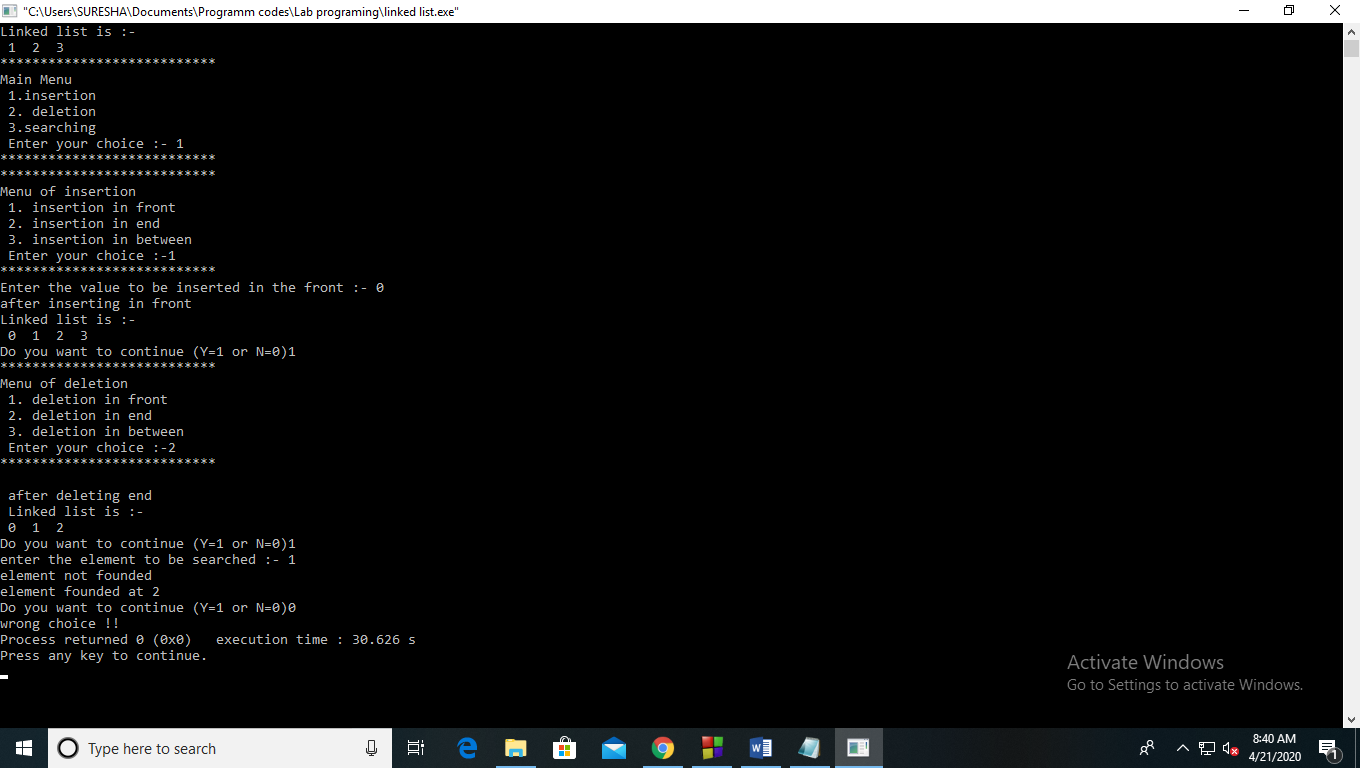
if(i!=1)

printf("element not founded\n");

}

}

**Output:-**



**Experiment 7**

**Aim:-** Write a menu driven program to implement doubly linked list, implement the following operations

* Addition at the start.
* Addition at the end.
* Addition in between.
* Deletion at the start.
* Deletion at the end.
* Deletion in between.

**Theory:-** Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List. Following are the important terms to understand the concept of doubly linked list.

        **Link** − Each link of a linked list can store a data called an element.

        **Next** − Each link of a linked list contains a link to the next link called Next.

        **Prev** − Each link of a linked list contains a link to the previous link called Prev.

        **LinkedList** − A Linked List contains the connection link to the first link called First and to the last link called Last.

As per the above illustration, following are the important points to be considered.

        Doubly Linked List contains a link element called first and last.

        Each link carries a data field(s) and two link fields called next and prev.

        Each link is linked with its next link using its next link.

        Each link is linked with its previous link using its previous link.

        The last link carries a link as null to mark the end of the list.

Basic Operations

Following are the basic operations supported by a list.

        **Insertion** − Adds an element at the beginning of the list.

        **Deletion** − Deletes an element at the beginning of the list.

        **Insert Last** − Adds an element at the end of the list.

        **Delete Last** − Deletes an element from the end of the list.

        **Insert After** − Adds an element after an item of the list.

        **Delete** − Deletes an element from the list using the key.

        **Display forward** − Displays the complete list in a forward manner.

        **Display backward** − Displays the complete list in a backward manner.

**Code:-**

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \*next;

struct Node \*prev;

};

typedef struct Node \*NodePtr;

NodePtr head = NULL;

NodePtr tail = NULL;

int isEmpty()

{

return head == NULL;

}

void insertAtFront(int value)

{

NodePtr node = malloc(sizeof(NodePtr));

node->data = value;

node->next = NULL;

node->prev = NULL;

if (isEmpty())

{

head = node;

tail = node;

}

else

{

node->next = head;

head->prev = node;

head = node;

}

}

void search(int data)

{

NodePtr current;

int pos = 0;

if(head==NULL)

{

printf("Linked List not initialized");

return;

}

current = head;

while(current!=NULL)

{

pos++;

if(current->data == data)

{

printf("%d found at position %d\n", data, pos);

return;

}

if(current->next != NULL)

current = current->next;

else

break;

}

printf("%d does not exist in the list\n", data);

}

void insertAtBack(int value)

{

NodePtr node = malloc(sizeof(NodePtr));

node->data = value;

node->next = NULL;

node->prev = NULL;

if (isEmpty())

{

head = node;

tail = node;

}

else

{

tail->next = node;

node->prev = tail;

tail = node;

}

}

NodePtr insertAfter(int key, int value)

{

NodePtr node = malloc(sizeof(NodePtr));

node->data = value;

node->next = NULL;

node->prev = NULL;

NodePtr currPtr = head;

while (currPtr != NULL && currPtr->data != key)

{

currPtr = currPtr->next;

}

if (currPtr == NULL)

{

printf("key not found");

}

else if (currPtr->next == NULL)

{

currPtr->next = node;

node->prev = currPtr;

tail = node;

}

else

{

NodePtr nextPtr = currPtr->next;

currPtr->next = node;

node->prev = currPtr;

node->next = nextPtr;

nextPtr->prev = node;

}

}

int topFront()

{

if (isEmpty())

{

printf("%s", "List is empty");

}

else

{

return head->data;

}

}

int topBack()

{

if (isEmpty())

{

printf("%s", "List is empty");

}

else if (head->next == NULL)

{

return head->data;

}

else

{

NodePtr currPtr = head;

while (currPtr->next != NULL)

{

currPtr = currPtr->next;

}

return currPtr->data;

}

}

int popFront()

{

int item;

if (isEmpty())

{

printf("%s", "List is empty");

return -99999;

}

else

{

item = head->data;

if (head->next != NULL)

{

head->next->prev = NULL;

}

NodePtr next = head->next;

free(head);

head = next;

}

return item;

}

int popBack()

{

int item;

if (isEmpty())

{

printf("%s", "List is empty");

return -99999;

}

else

{

item = tail->data;

if (tail->prev != NULL)

{

tail->prev->next = NULL;

}

NodePtr prev = tail->prev;

tail->prev = NULL;

free(tail);

tail = prev;

}

return item;

}

void delete(int key)

{

if (isEmpty())

{

printf("%s", "List is empty");

return;

}

NodePtr currPtr = head;

while(currPtr != NULL && currPtr->data != key)

{

currPtr = currPtr->next;

}

if (currPtr == NULL)

{

printf("%s", "Key is not found in the list");

return;

}

if (currPtr->prev == NULL)

{

popFront();

}

else if (currPtr->next == NULL)

{

popBack();

}

else

{

NodePtr nextPtr = currPtr->next;

NodePtr prevPtr = currPtr->prev;

nextPtr->prev = prevPtr;

prevPtr->next = nextPtr;

currPtr->next = NULL;

currPtr->prev = NULL;

free(currPtr);

currPtr = NULL;

}

}

void print()

{

NodePtr currPtr = head;

while (currPtr != NULL)

{

printf("%d", currPtr->data);

printf(" ");

currPtr = currPtr->next;

}

printf("\n");

}

void printReverse()

{

NodePtr currPtr = tail;

while (currPtr != NULL)

{

printf("%d", currPtr->data);

printf(" ");

currPtr = currPtr->prev;

}

printf("\n");

}

int find(int key)

{

if (isEmpty())

{

return 0;

}

NodePtr currPtr = head;

while (currPtr != NULL && currPtr->data != key)

{

currPtr = currPtr->next;

}

if (currPtr == NULL)

{

return 0;

}

return 1;

}

int main()

{

int opt,val,opt1,pos,item;

int ch=1;

while(ch==1)

{

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

printf("Menu : \n 1. insertion \n 2. deletion \n 3. search \n 4. display \n Enter your choice :- ");

scanf("%d",&opt);

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

switch(opt)

{

case 1:

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

printf("insertion Menu : \n 1. Insertion at the head \n 2.Insert at the tail \n 3.Insert at the middle \n Enter your choice :- ");

scanf("%d",&opt1);

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

if(opt1==1)

{

printf("Enter the value to be insert :-");

scanf("%d",&val);

insertAtFront(val);

}

else if(opt1==2)

{

printf("Enter the value to be insert :-");

scanf("%d",&val);

insertAtBack(val);

}

else if(opt1==3)

{

printf("Enter the value to be insert :-");

scanf("%d",&val);

printf("Enter after which element :-");

scanf("%d",&pos);

insertAfter(pos,val);

}

else

printf("wrong choie!!! \n");

printf("Want to continue (y=1 or n=0)");

scanf("%d",&ch);

if(ch==1)

continue;

else

break;

case 2:

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

printf("deletion Menu : \n 1. Delete at the head \n 2.Delete at the tail \n 3.Delete at the middle \n Enter your choice :-");

scanf("%d",&opt1);

if(opt1==1)

{

int item = popFront();

printf(" the element deleted is :- %d\n", item);

}

else if(opt1==2)

{

item = popBack();

printf("the element deleted is :- %d\n", item);

}

else if(opt1==3)

{

printf("Enter the position:- ");

scanf("%d",&pos);

delete(pos);

}

else

printf("wrong choice!!! \n");

printf("Want to continue (y=1 or n=0)");

scanf("%d",&ch);

if(ch==1)

continue;

else

break;

case 3:

printf("Enter the element to be searched :-");

scanf("%d",&pos);

search(pos);

printf("Want to continue (y=1 or n=0)");

scanf("%d",&ch);

if(ch==1)

continue;

else

break;

case 4:

print();

printf("Want to continue (y=1 or n=0)");

scanf("%d",&ch);

if(ch==1)

continue;

else

break;

}

}

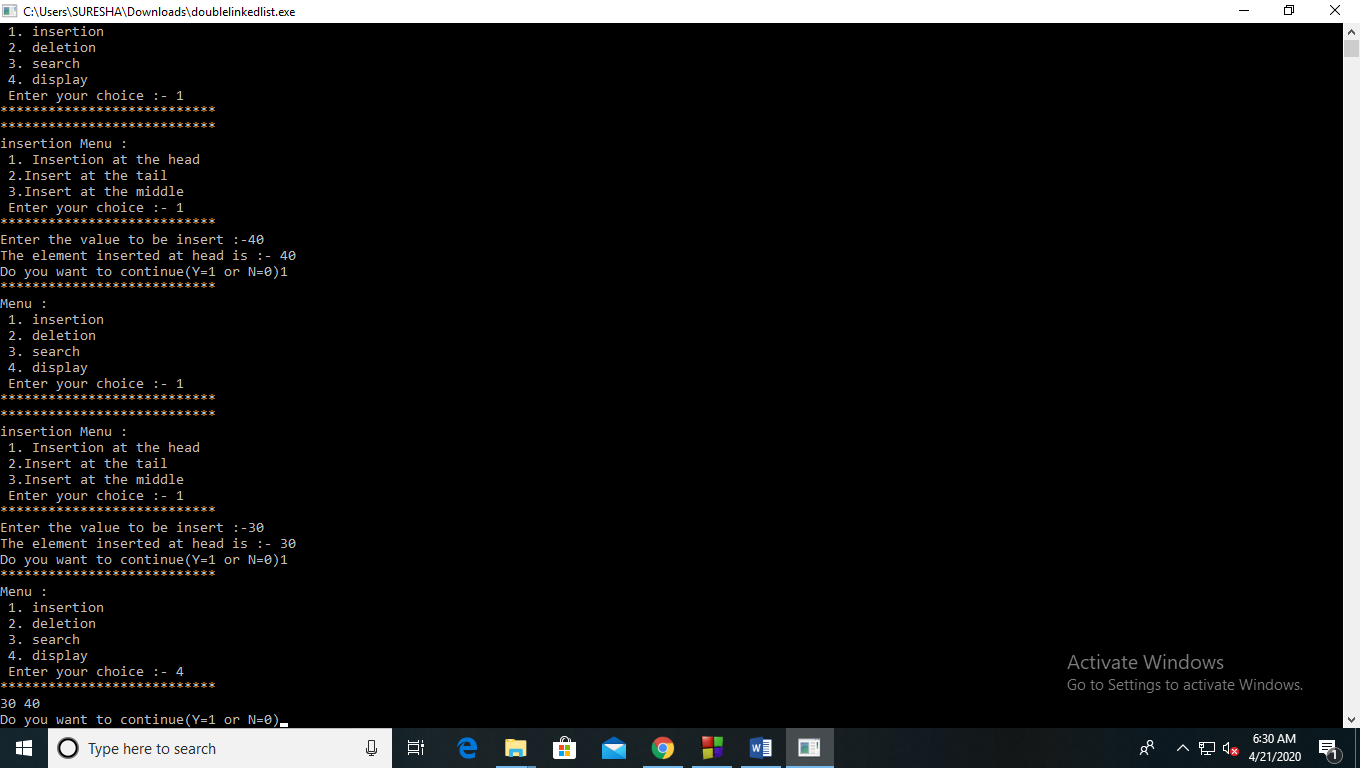
return 0;

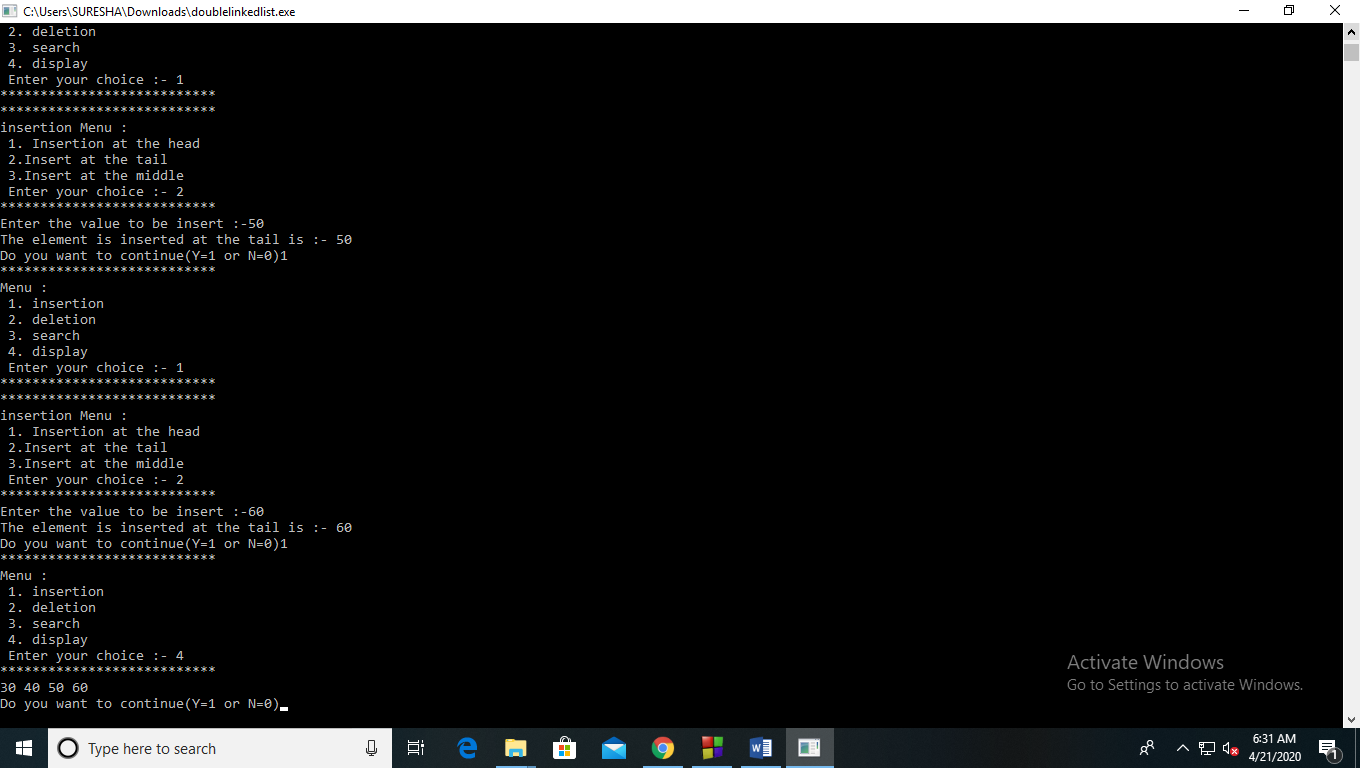
}

**Output:-**

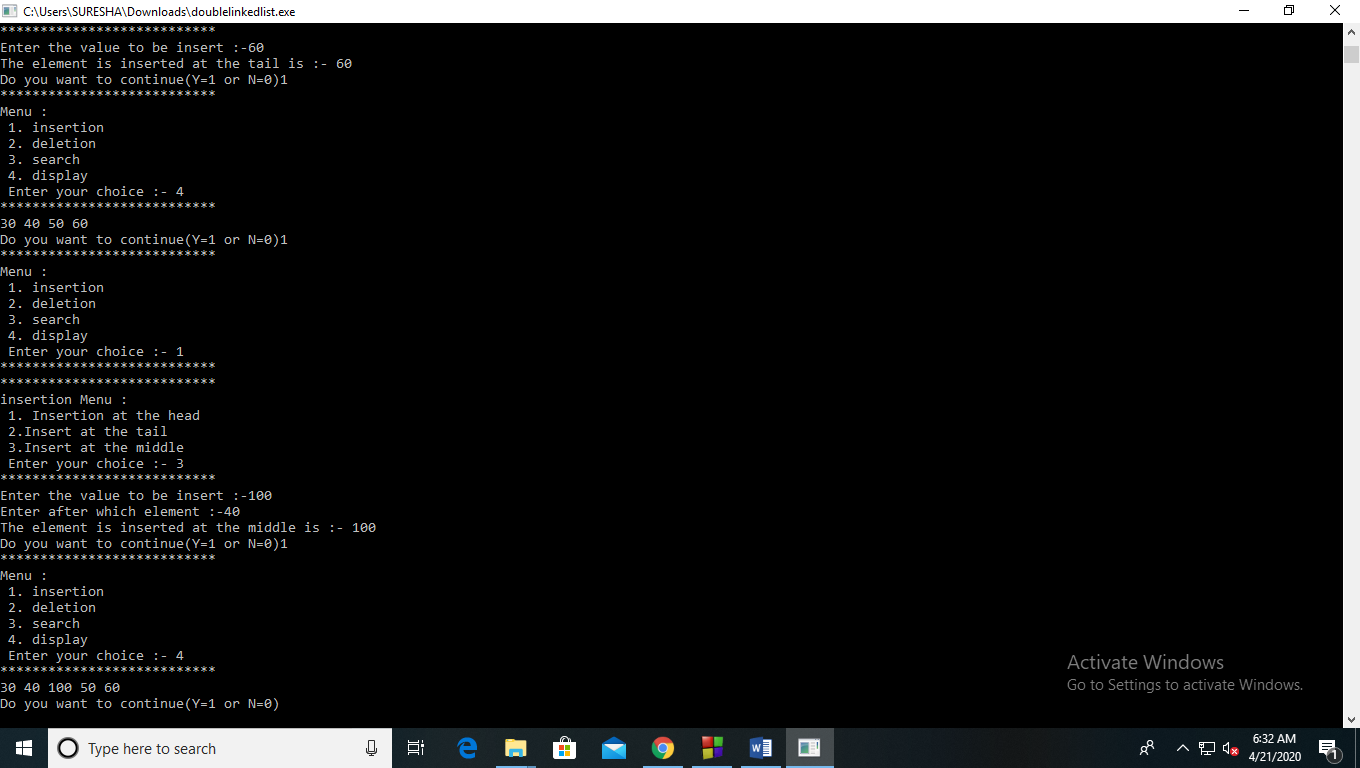
1)insertion:-

Insertion at head:



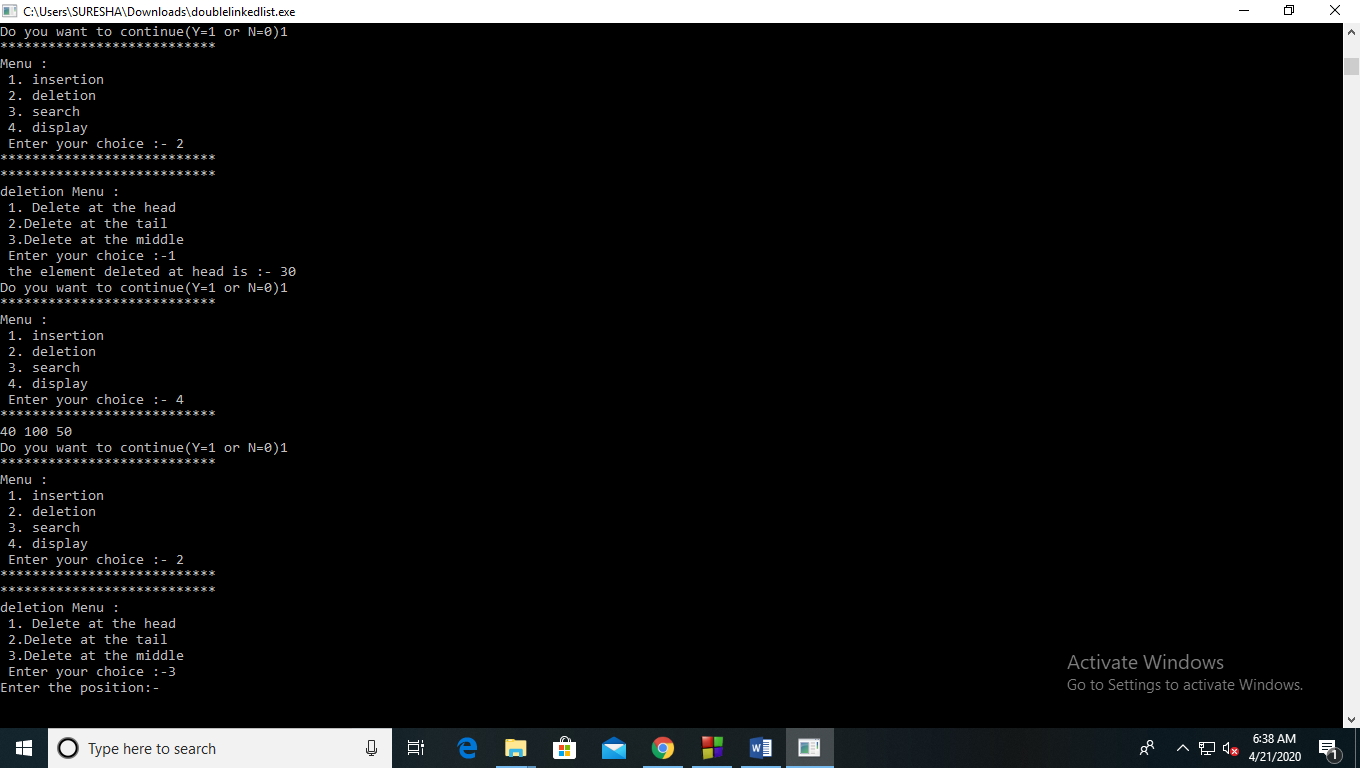
Insertion at tail: 

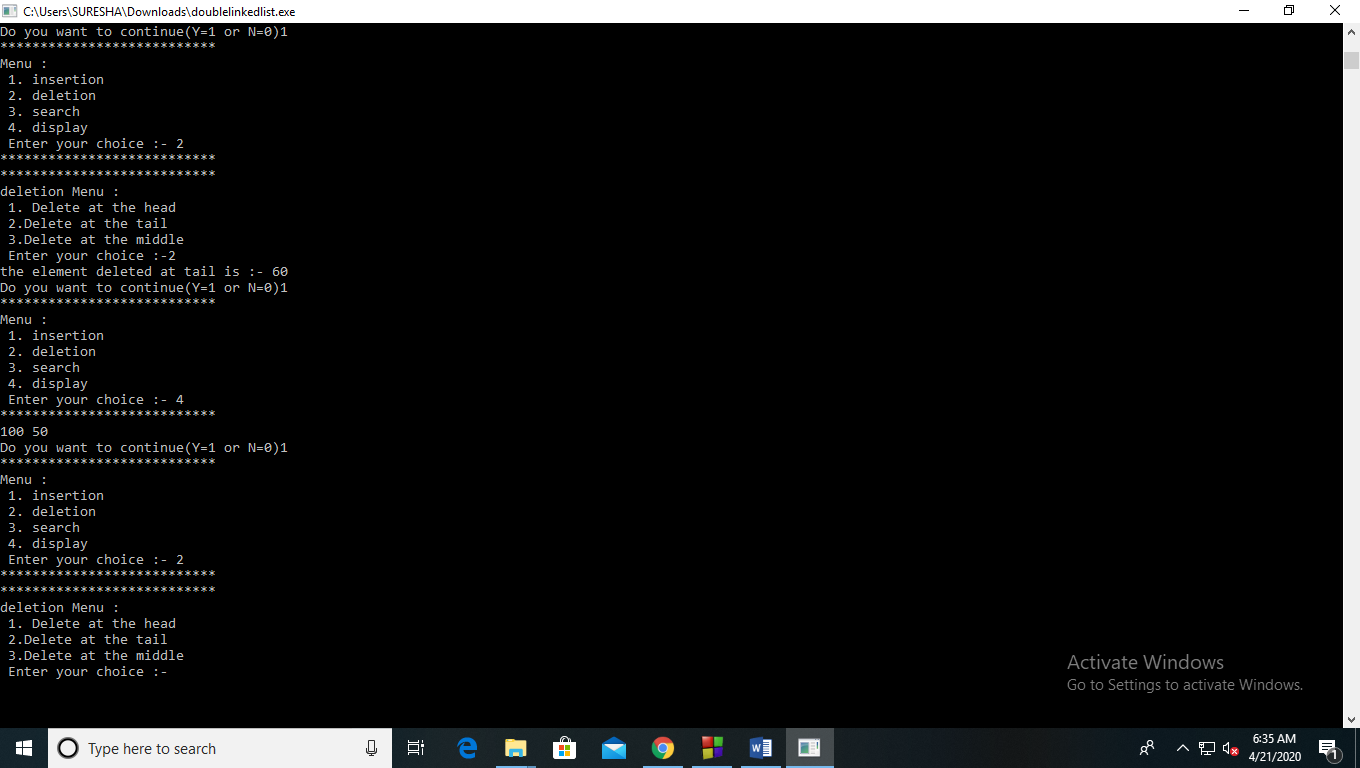
Insertion at middle:



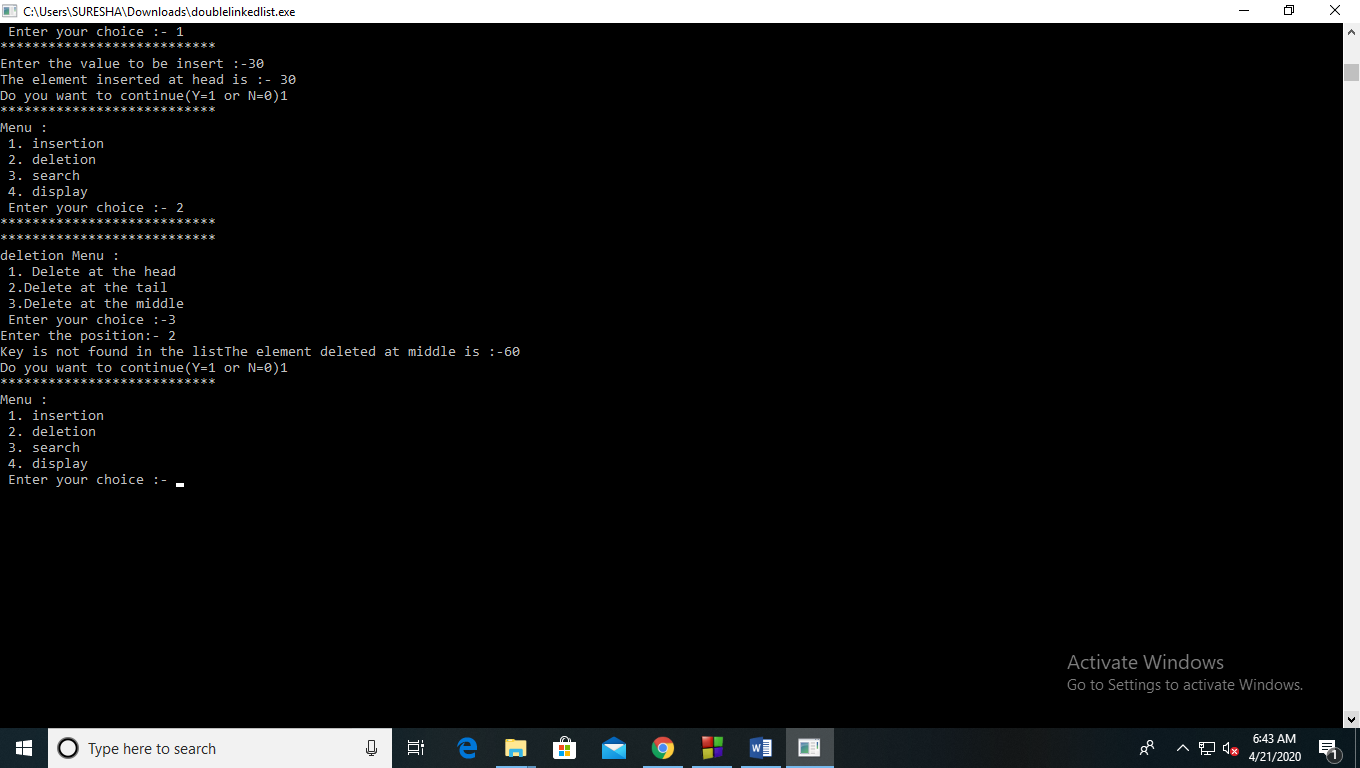
2) Deletion:-

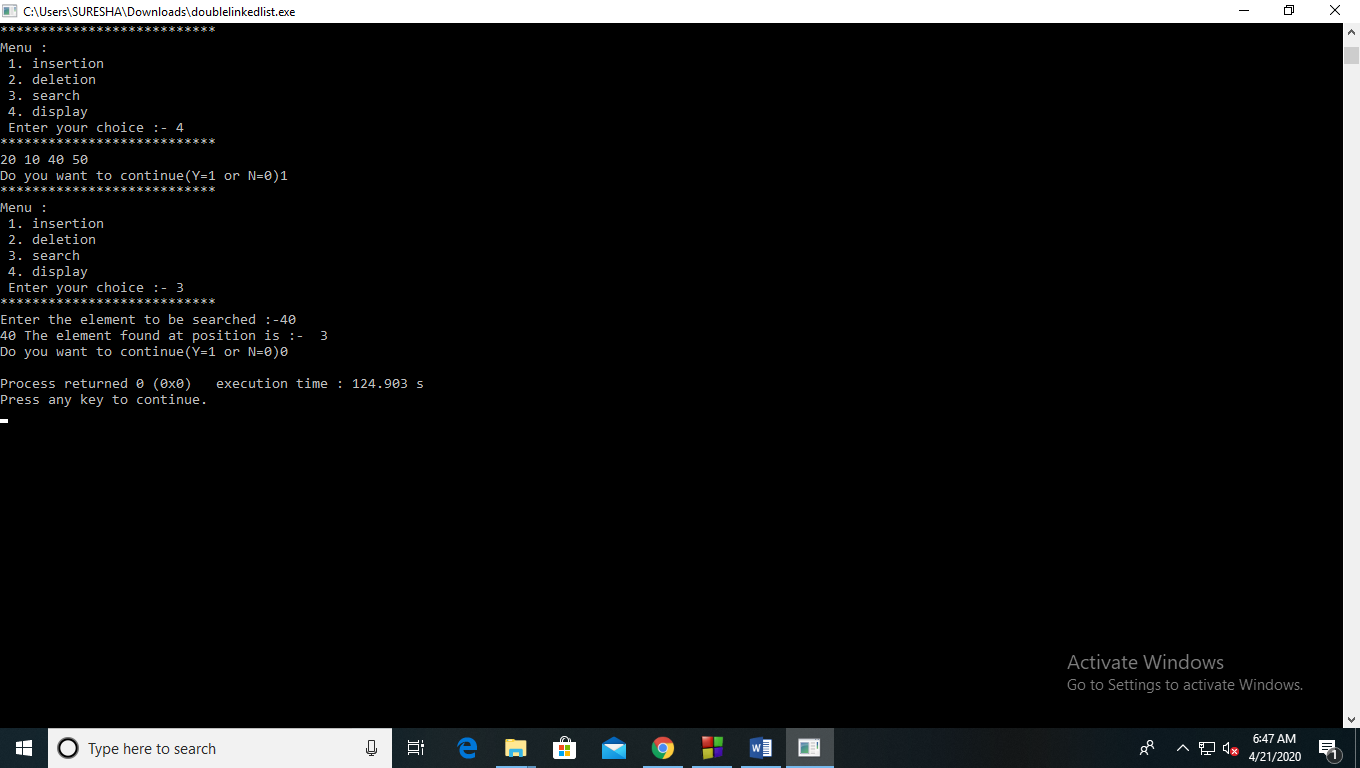
Deletion at head:



Deletion at tail:- 

Deletion at middle:-



3)Search:- 

**Experiment 8**

**Aim:-**Write a menu driven program to implement the following Stack operations using Linked List.

(i) Push()

(ii)Pop()

(iii)display()

**THEORY:** The main advantage of using linked list over an arrays is that it is possible to implements a stack that can shrink or grow as much as needed.

**ALGORITHM FOR PUSHING AN ELEMENT**

1. Create a **newNode** with given value.
2. Check whether stack is **Empty** (**top** == **NULL**)
3. If it is **Empty**, then set **newNode → next** = **NULL**.
4. If it is **Not Empty**, then set **newNode → next** = **top**.
5. Finally, set **top** = **newNode**.

**ALGORITHM FOR POPPING AN ELEMENT-**

1. Check whether **stack** is **Empty** (**top == NULL**).
2. If it is **Empty**, then display **"Stack is Empty!!! Deletion is not possible!!!"** and terminate the function
3. If it is **Not Empty**, then define a **Node** pointer '**temp**' and set it to '**top**'.
4. Then set '**top** = **top → next**'.
5. Finally, delete '**temp**'. (**free(temp)**).

**ALGORITHM FOR DISPLAYING-**

1. Check whether stack is **Empty** (**top** == **NULL**).
2. If it is **Empty**, then display **'Stack is Empty!!!'** and terminate the function.
3. If it is **Not Empty**, then define a Node pointer **'temp'** and initialize with **top**.
4. 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**).
5. Finally! Display '**temp → data** ---> **NULL**'.

**Code:-**

#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 c=0;

int ch;

while(c != 4)

{

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

printf("\n MENU \n");

printf("\n1.Push\n2.Pop\n3.display");

printf("\n enter your choice:");

scanf("%d",&c);

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

switch(c)

{

case 1:

{

push();

printf("\n Do want to continue (yes = 1 and no = 0 ) :-\n");

scanf("%d",&ch);

if(ch==1)

continue;

else

printf("THANK YOU\n");

break;

}

case 2:

{

pop();

printf("\n Do want to continue (yes = 1 and no = 0 ) :-\n");

scanf("%d",&ch);

if(ch==1)

continue;

else

printf("THANK YOU\n");

break;

}

case 3:

{

display();

printf("\n Do want to continue (yes = 1 and no = 0 ) :-\n");

scanf("%d",&ch);

if(ch==1)

continue;

else

printf("THANK YOU\n");

break;

}

default:

{

printf("Invalid choice ");

}

};

}

}

void push ()

{

int val;

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

if(ptr == NULL)

{

printf("Overflow");

}

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("The value has been pushed:%d",val);

}

}

void pop()

{

int item;

struct node \*ptr;

if (head == NULL)

{

printf("Underflow");

}

else

{

item = head->val;

ptr = head;

head = head->next;

free(ptr);

printf("The value has been popped:%d",item);

}

}

void display()

{

int i;

struct node \*ptr;

ptr=head;

if(ptr == NULL)

{

printf("Stack is empty\n");

}

else

{

printf("The Stack elements are:- \n");

while(ptr!=NULL)

{

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

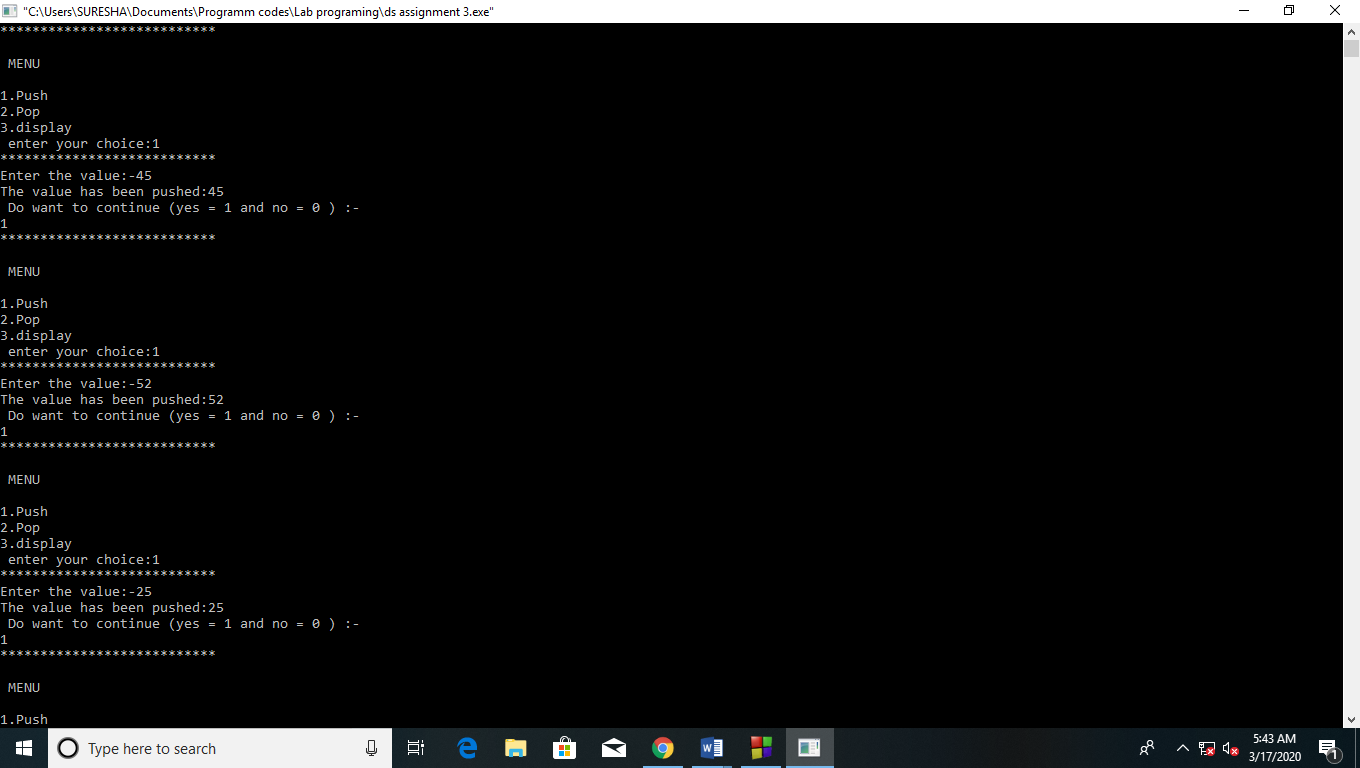
ptr = ptr->next;

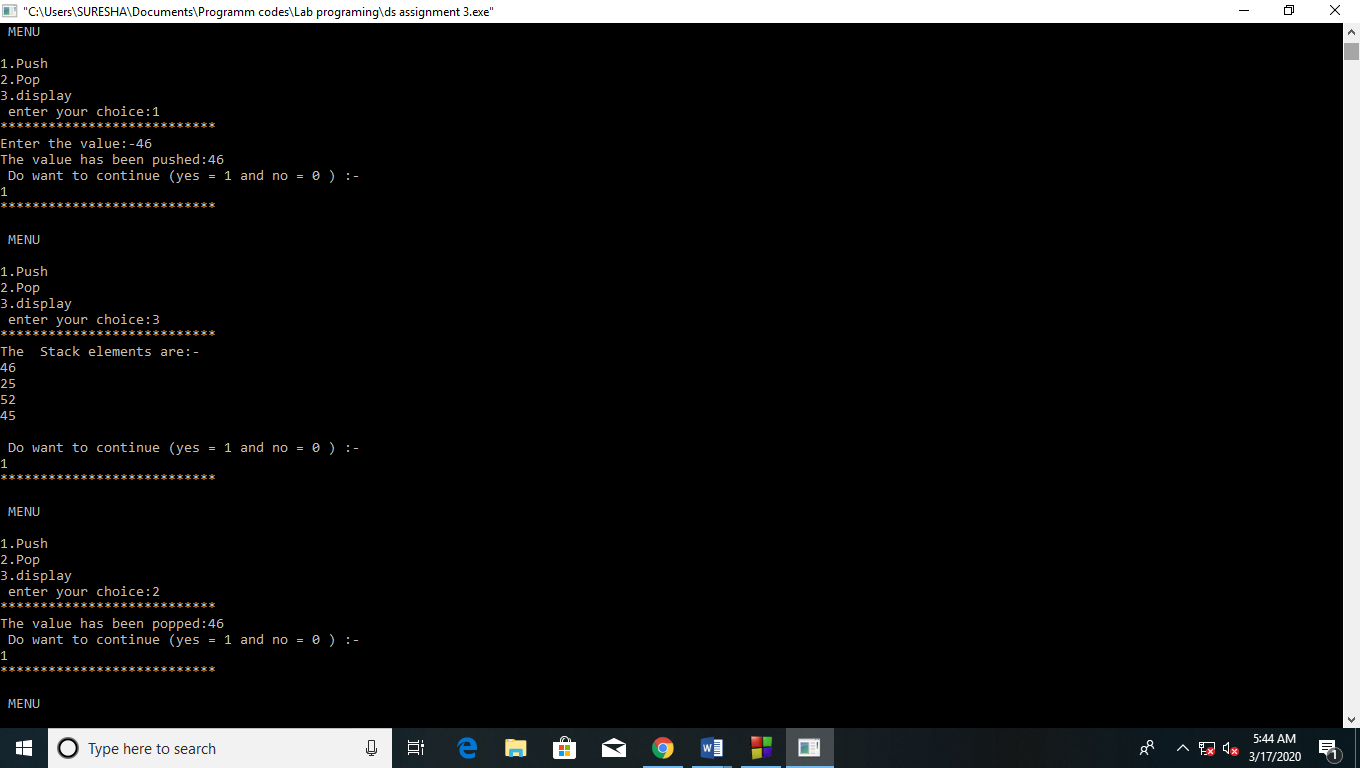
}

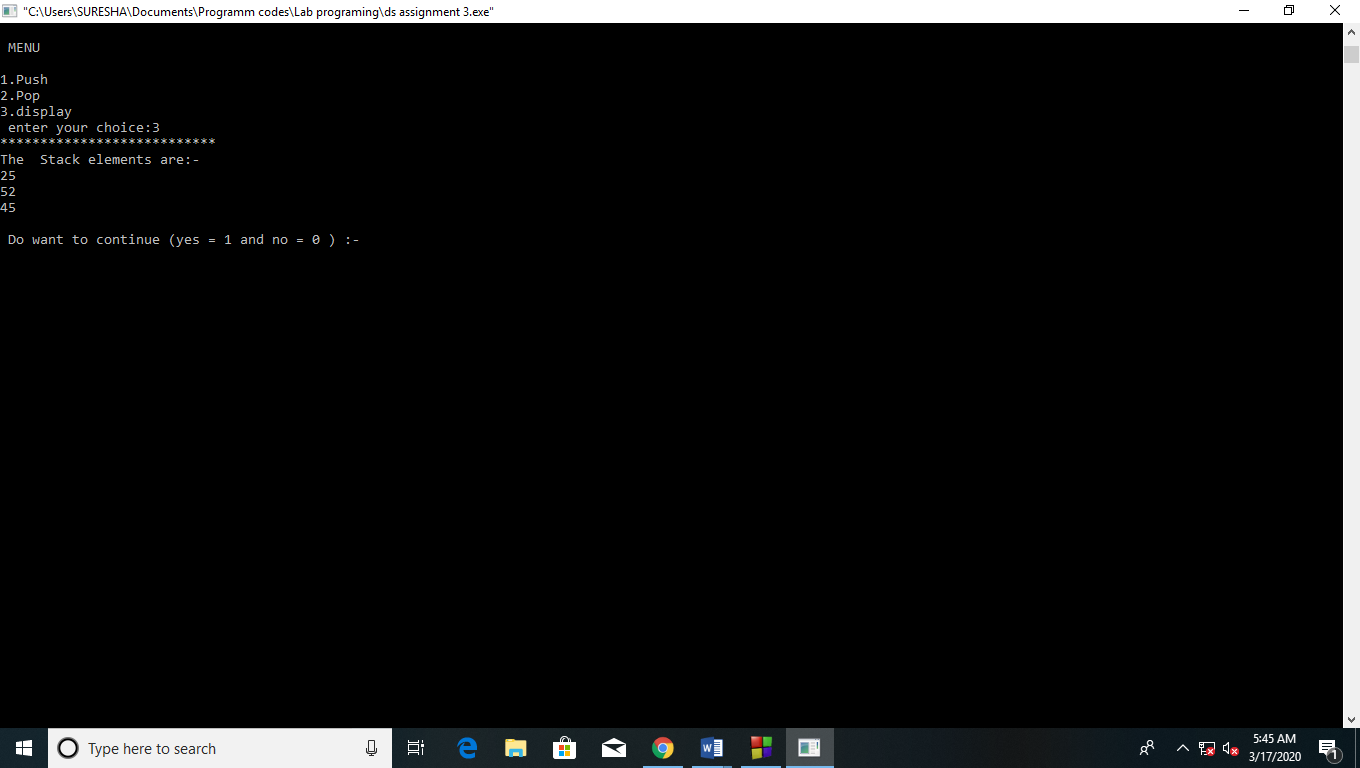
}

}

**Output:-**







**Experiment 9**

**Aim:-**Implement the following Queue operations using Linked List.

•(i) Enqueue()

•(ii)Dequeue()

•(iii)display()

**Theory:**

Enqueue Operation

The following steps should be taken to enqueue (insert) data into a queue −

* **Step 1** − Check if the queue is full.
* **Step 2** − If the queue is full, produce overflow error and exit.
* **Step 3** − If the queue is not full, increment **rear** pointer to point the next empty space.
* **Step 4** − Add data element to the queue location, where the rear is pointing.
* **Step 5** − return success.

## Dequeue Operation

Accessing data from the queue is a process of two tasks − access the data where **front** is pointing and remove the data after access. The following steps are taken to perform **dequeue** operation −

* **Step 1** − Check if the queue is empty.
* **Step 2** − If the queue is empty, produce underflow error and exit.
* **Step 3** − If the queue is not empty, access the data where **front** is pointing.
* **Step 4** − Increment **front** pointer to point to the next available data element.
* **Step 5** − Return success.

**Code:**-

#include<stdio.h>

#include<stdlib.h>

struct queue

{

int data;

struct queue \*next;

}\*ptr,\*rear,\*front,\*temp,\*newptr;

struct queue\* create(int val)

{

newptr=(struct queue\*)malloc(sizeof(struct queue\*));

newptr->data=val;

newptr->next=NULL;

return newptr;

}

void enqueue(struct queue\* a)

{

if(front==NULL)

{

front=rear=a;

}

else

{

rear->next=a;

rear=a;

}

}

void dequeue(struct queue \*a)

{

if(front==NULL)

{

printf("The Queue Is Empty!!");

}

else

{

temp=front;

front=front->next;

free(temp);

}

}

void display(struct queue \*a)

{

printf("The Queue is :- \n");

while(a!=NULL)

{

printf("%d ->",a->data);

a=a->next;

}

}

void main()

{

int ch=1,opt,val;

while(ch==1)

{

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

printf("Menu : \n 1. Enqueue \n 2. Dequeue \n 3. Display \n 4. Exit \n Enter your choice:- ");

scanf("%d",&opt);

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

switch(opt)

{

case 1:

printf("Enter The Value in Queue :-");

scanf("%d",&val);

ptr=create(val);

enqueue(ptr);

printf("Do You Want to continue (y=1 and N=0)");

scanf("%d",&ch);

if(ch==1)

continue;

else

break;

case 2:

dequeue(ptr);

printf("DO You Want to continue (y=1 and N=0)");

scanf("%d",&ch);

if(ch==1)

continue;

else

break;

case 3:

display(front);

printf("Do You Want to continue (y=1 and N=0)");

scanf("%d",&ch);

if(ch==1)

continue;

else

break;

case 4:

printf("!!!!! All Operations are Done !!!!!");

exit(0);

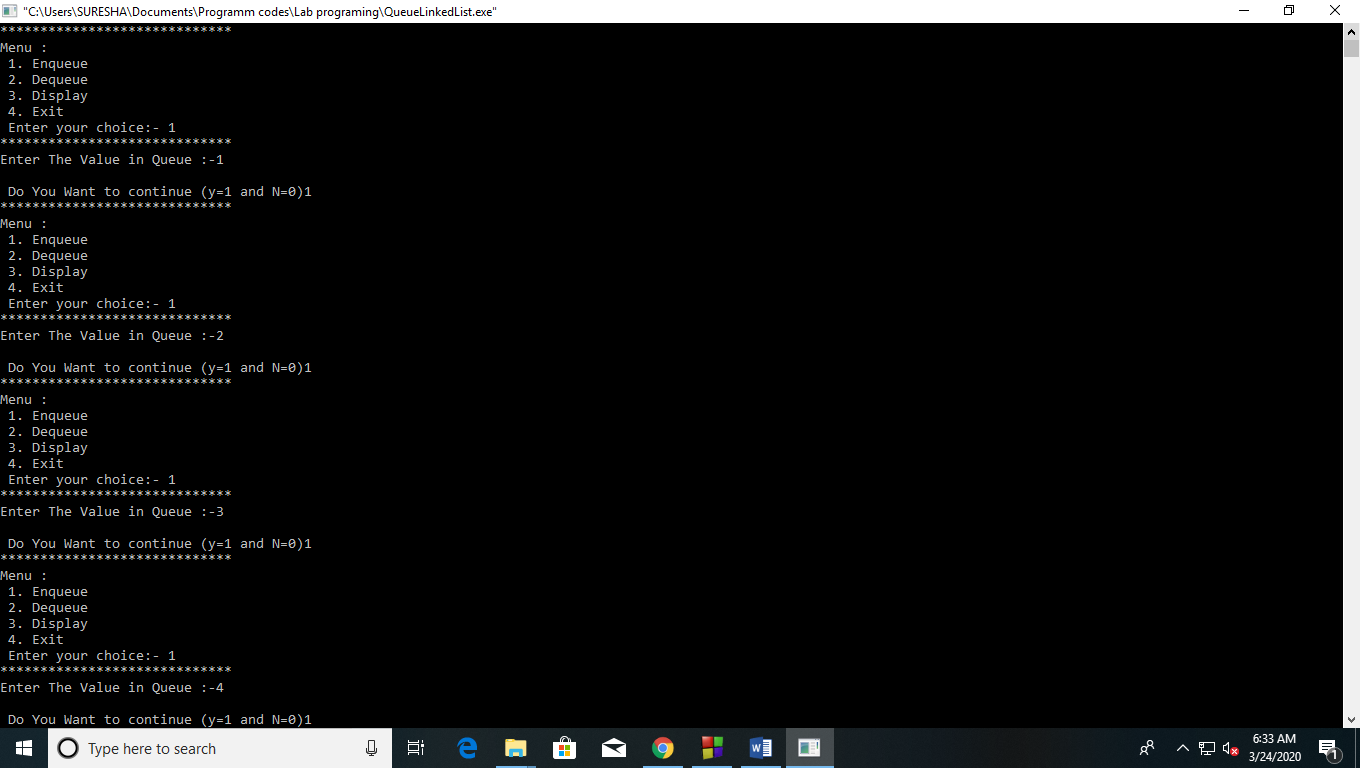
default:

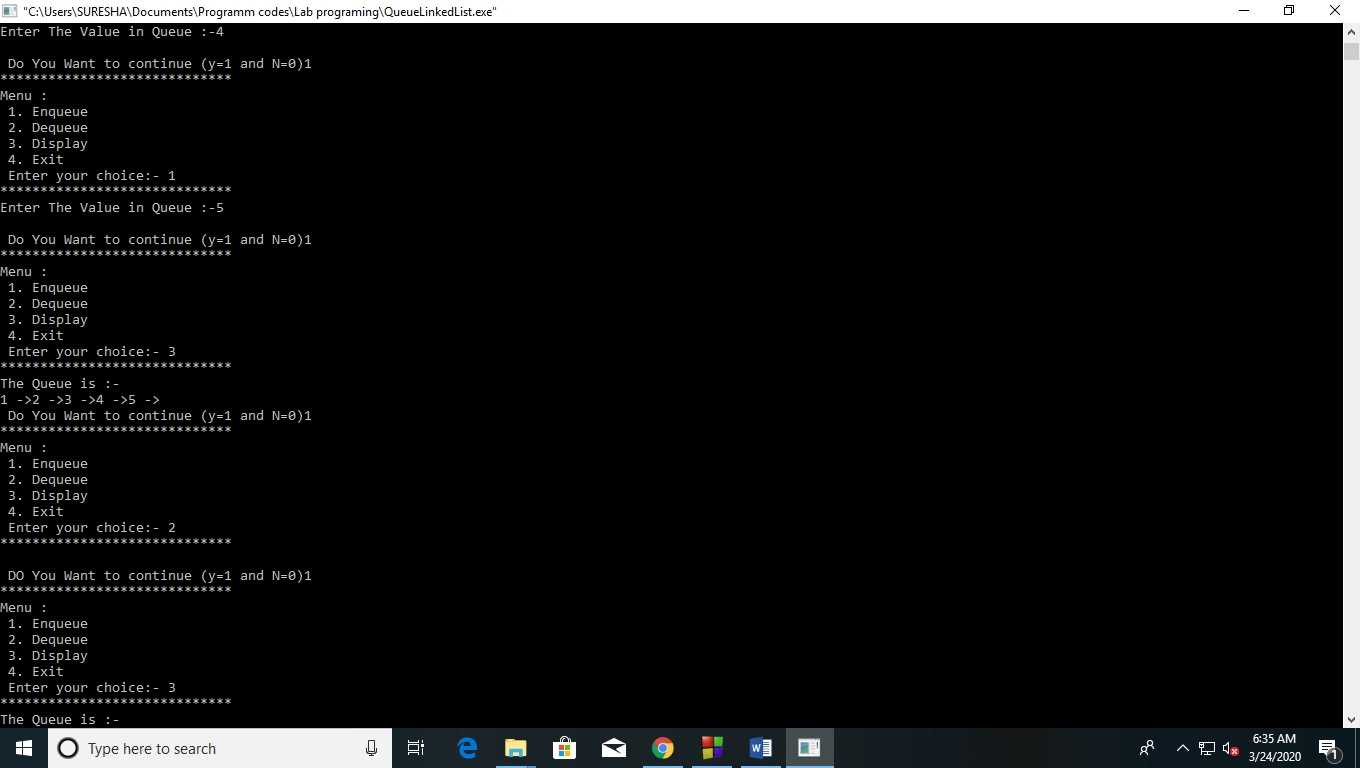
printf("Invalid choice !!!");

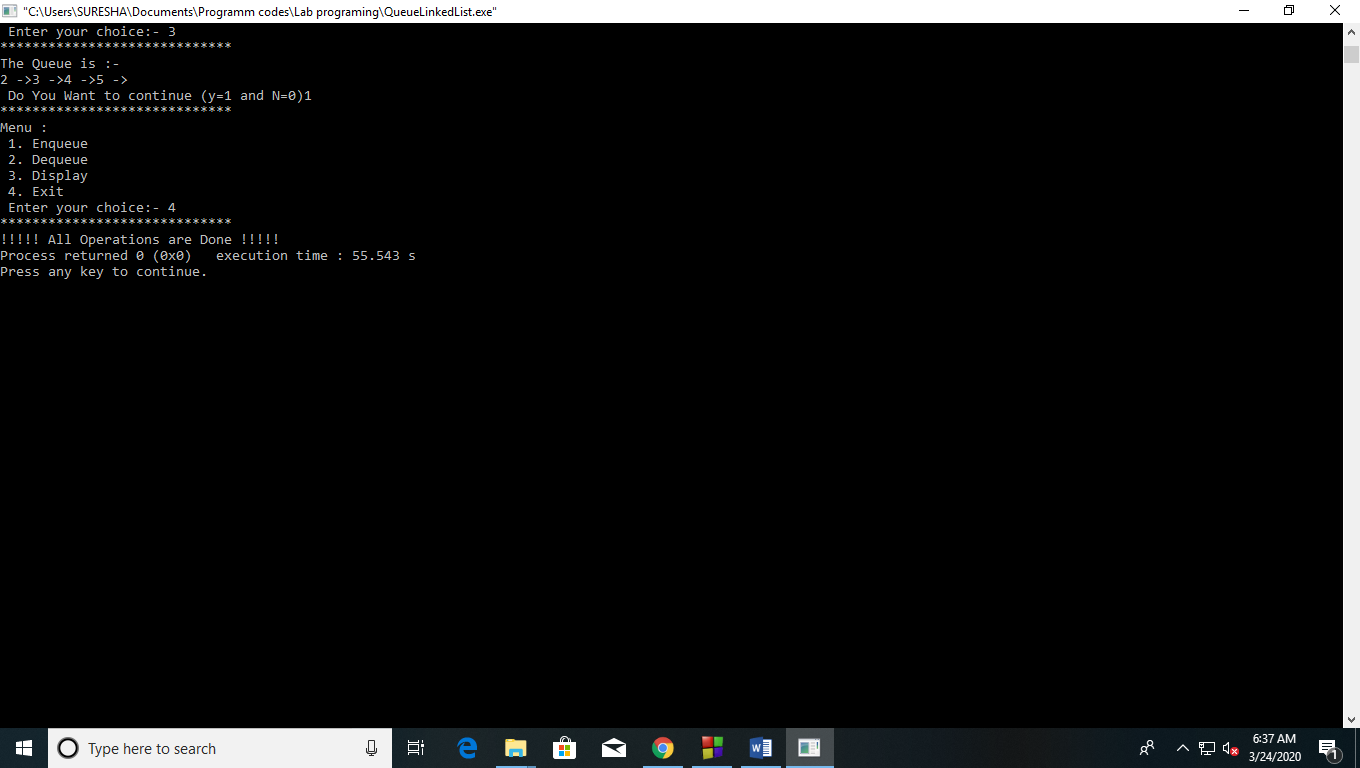
}

}

}

**Output:-** 





**Experiment 10**

**Aim :-** Write a C Program to [convert Infix expression to Postfix form using Stack](http://lms6.amizone.net/mod/assign/view.php?id=82003)

**Theory :-**Postfix  :- An expression is called the postfix expression if the operator appears in the expression after the operands. Simply of the form (operand1 operand2 operator).

Infix :- where operators are used **in**-between operands. It is easy for us humans to read, write, and speak in infix notation but the same does not go well with computing devices.

Transforming Infix to Postfix expression

• POLISH(Q,P) Suppose Q is an arithmetic expression written in Infix notation. This algorithm finds the equivalent postfix expression P.

• 1.Push “(“ onto STACK and add “)” to the end of Q.

• 2.Scan Q from left to right and repeat step 3 to 6 for each statement of Q until the stack is empty.

• 3.If an operand is encountered, add it to P.

• 4. If a left parenthesis is encountered, push it onto stack.

• 5.If an operator OP is encountered , then:

• (a) Repeatedly pop from STACK and add P each operator(on the top of the STACK) which has the

same precedence as or higher precedence precedence than OP

• (b) Add OP to STACK

• [End of If Structure]

• 6. If a right parenthesis is encountered , then:

• (a) Repeatedly pop from STACK and add to p each operator (on top of the STACK) until a left

parenthesis is encountered.

• (b) Remove the left parenthesis [ Do not add the left parenthesis to P]

• [End of If Structure] [End of Step 2 loop]

• 7.Exit

**Code :-**

#include<stdio.h>

char stack[20];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

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;

if(x == '^')

return 3;

}

main()

{

char exp[20];

char \*e, x;

printf("Enter the expression :- ");

scanf("%s",exp);

e = exp;

while(\*e != '\0')

{

if(isalnum(\*e))

printf("%c",\*e);

else if(\*e == '(')

push(\*e);

else if(\*e == ')')

{

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

printf("%c", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

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

push(\*e);

}

e++;

}

while(top != -1)

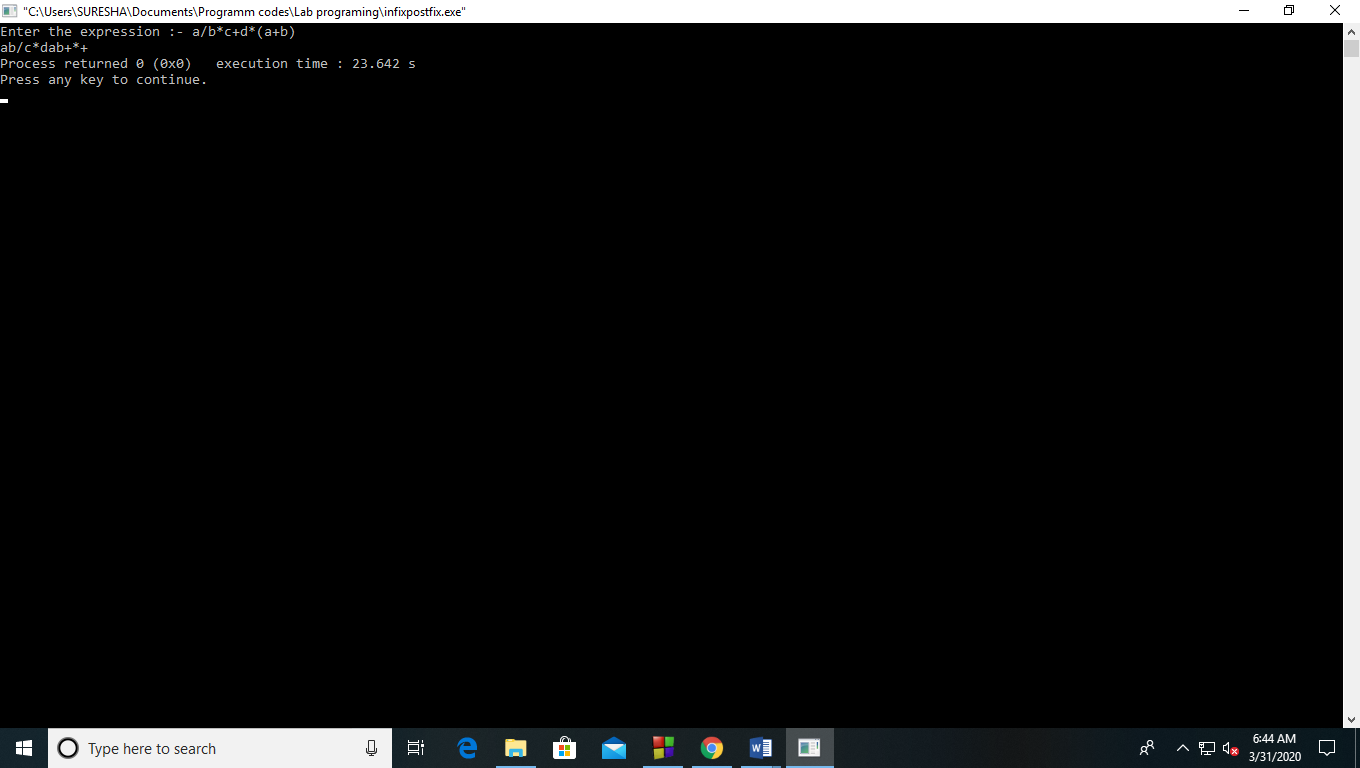
{

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

}

}

**Output:-**



**Experiment 11**

**AIM:** Write a menu driven program implementing the following techniques.

Linear search.

Binary search.

**THEORY:**

LINEAR SEARCH- Suppose DATA is a Linear Array with n elements. The most intuitive way to search for an ITEM in DATA is to compare ITEM with each element of DATA one by one. This method that traverses data sequentially is called as linear or sequential search.

ALGORITHM-

(Linear Search) LINEAR (DATA,N,ITEM,LOC)

Here DATA is a linear array with N Elements and ITEM is a given item of information. This Algorithm finds the location LOC of ITEM in DATA or sets LOC:=0 if the search is unsuccessful.

[Insert ITEM at the end of Data] Set DATA[N+1]:=ITEM

[Initialize Counter] Set LOC:=1

[Search for ITEM] Repeat while DATA[LOC] ≠ ITEM Set LOC=LOC+1. [End of Loop]

[Successful?] If LOC=N+1 then Set LOC=0

Exit.

BINARY SEARCH- Binary Search is a divide and conquer algorithm. Like all divide and conquer algorithms, Binary Search first divides a large array into two smaller sub-arrays and then recursively operate the sub-arrays. But instead of operating on both sub-arrays, it discards one sub-array and continues on the second sub-array. This decision of discarding one sub-array is made in just one comparison.

ALGORITHM-

Compare x with the middle element.

If x matches with middle element, we return the mid index.

Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.

Else (x is smaller) recur for the left half.

**Code:-**

#include<stdio.h>

#include<conio.h>

void linearsearch( )

{

int a[10],i,e,flag=0,n;

printf("Enter the size of the array:");

scanf("%d",&n);

printf("Enter the elements in the array:\n");

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

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

}

printf("Enter the element you want to search:");

scanf("%d",&e);

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

{if(e==a[i])

{flag=1;

break;

}

else

flag=0;

}

if(flag==1)

printf("The element is present.\n");

else

printf("The element is not present\n");

}

void binarysearch( )

{

int a[20],i,n,lb,ub,mid,m,f=0,q=0,v=0;

printf("Enter the size of the array:");

scanf("%d",&n);

printf("Enter the elements in the array:\n");

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

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

}

printf("Enter the element you want to search:");

scanf("%d",&m);

lb=0;

ub=n-1;

while(lb<=ub)

{ mid=(lb+ub)/2;

if(m<a[mid])

{

ub=mid-1;

}

else if(m>a[mid])

{

lb=mid+1;

}

else if(m==a[mid])

{printf("The element is present");

f=1;

break;

}

}

if(lb>ub)

printf("The element is not present.");

}

void main( )

{

int ch,k;

do

{

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

printf("MENU :\n 1.Linear search \n 2.Binary search \n Enter your choice:");

scanf("%d",&ch);

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

switch(ch)

{

case 1:

linearsearch( );

break;

case 2:

binarysearch( );

break;

default:printf("Please selaect a valid choice.");

}

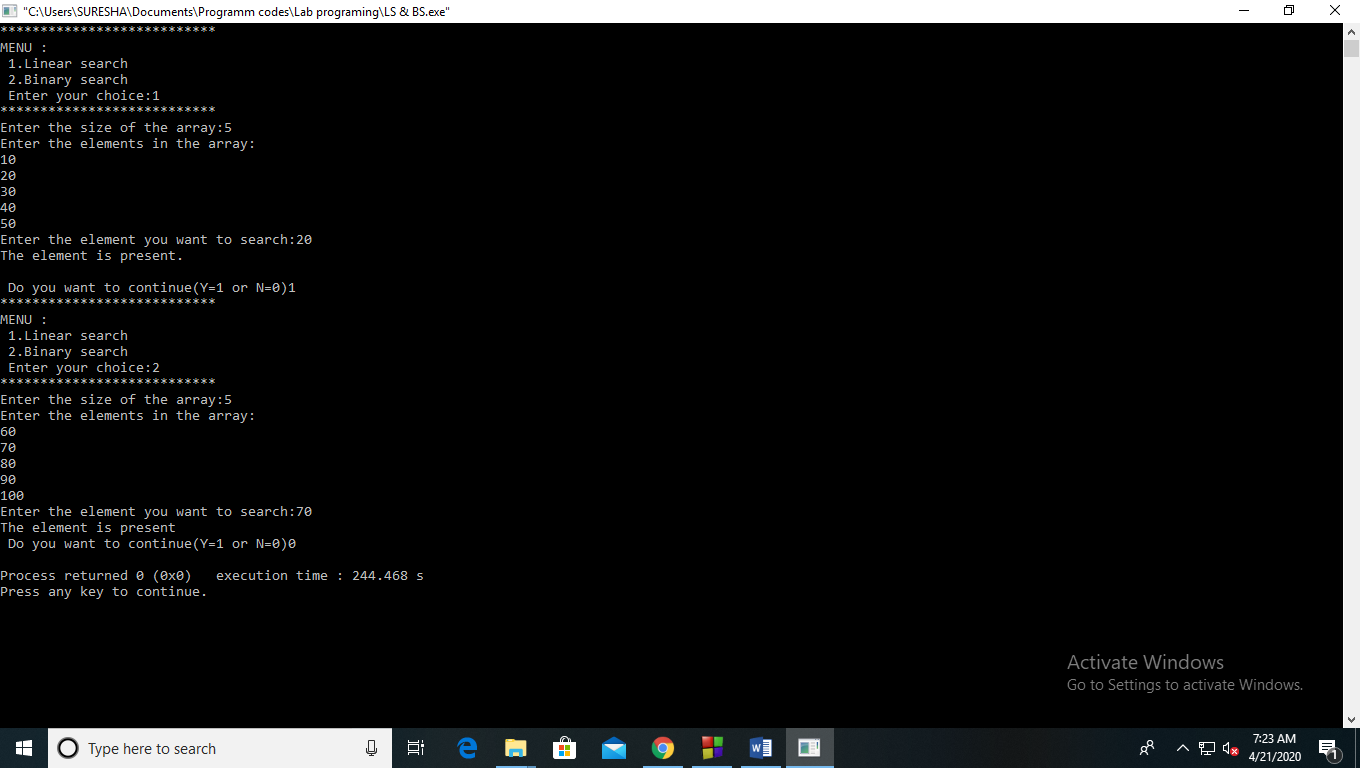
printf("\n Do you want to continue(Y=1 or N=0)");

scanf("%d",&k);

}while(k==1);

}

**Output:-**



**Experiment 12**

**Aim:-** WRITE A MENU DRIVEN PROGRAM TO IMPLEMENT.

a. Merge sort

b. Quick sort

**Theory:-**

Merge Sort :-

* The basic concept of merge sort is divides the list into two smaller sub-lists of approximately equal size.
* Recursively repeat this procedure till only one element is left in the sub-list.
* After this, various sorted sub-lists are merged to form sorted parent list. This process goes on recursively till the original sorted list arrived.

Quick Sort:-

* Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays.
* A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value.

• Quick sort partitions an array and then calls itself recursively twice to sort the two resulting subarrays. This algorithm is quite efficient for large-sized data sets.

**Code :-**

#include<stdio.h>

#define max 10

int a[11];

int b[10];

void merging(int low, int mid, int high) {

int l1, l2, i;

for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {

if(a[l1] <= a[l2])

b[i] = a[l1++];

else

b[i] = a[l2++];

}

while(l1 <= mid)

b[i++] = a[l1++];

while(l2 <= high)

b[i++] = a[l2++];

for(i = low; i <= high; i++)

a[i] = b[i];

}

void sort(int low, int high) {

int mid;

if(low < high) {

mid = (low + high) / 2;

sort(low, mid);

sort(mid+1, high);

merging(low, mid, high);

} else {

return;

}

}

void quickSort(int list[10],int first,int last){

int pivot,i,j,temp;

if(first < last){

pivot = first;

i = first;

j = last;

while(i < j){

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

i++;

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

j--;

if(i <j){

temp = list[i];

list[i] = list[j];

list[j] = temp;

}

}

temp = list[pivot];

list[pivot] = list[j];

list[j] = temp;

quickSort(list,first,j-1);

quickSort(list,j+1,last);

}

}

void main()

{

int ch=1;

int list[20],size,i,opt;

while(ch==1)

{

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

printf("Menu : \n 1. merge sort \n 2. quick sort \n 3. Exit \n Enter your choice :- ");

scanf("%d",&opt);

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

switch(opt)

{

case 1:

printf("Enter %d integer values: ",max);

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

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

printf("List before sorting\n");

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

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

sort(0, max);

printf("\nList after sorting\n");

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

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

printf(" Do you want to continue (YES=1 and NO=0)");

scanf("%d",&ch);

if(ch==1)

continue;

else

break;

case 2:

printf("Enter size of the list: ");

scanf("%d",&size);

printf("Enter %d integer values: ",size);

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

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

quickSort(list,0,size-1);

printf("List after sorting is: ");

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

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

printf("want to continue (YES=1 and NO=0)");

scanf("%d",&ch);

if(ch==1)

continue;

else

break;

case 3:

printf("All operations are done\n");

printf(" Do you want to continue (YES=1 and NO=0)");

scanf("%d",&ch);

if(ch==1)

continue;

else

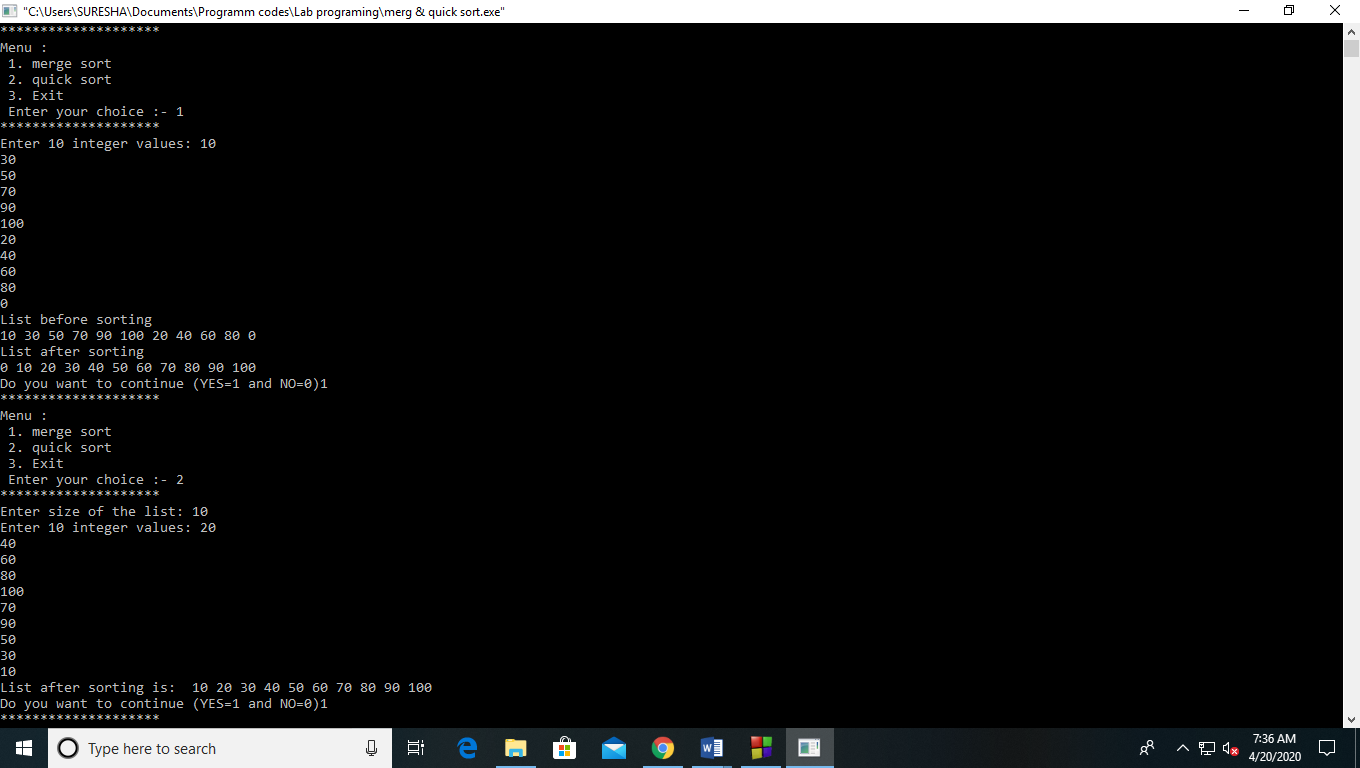
break;

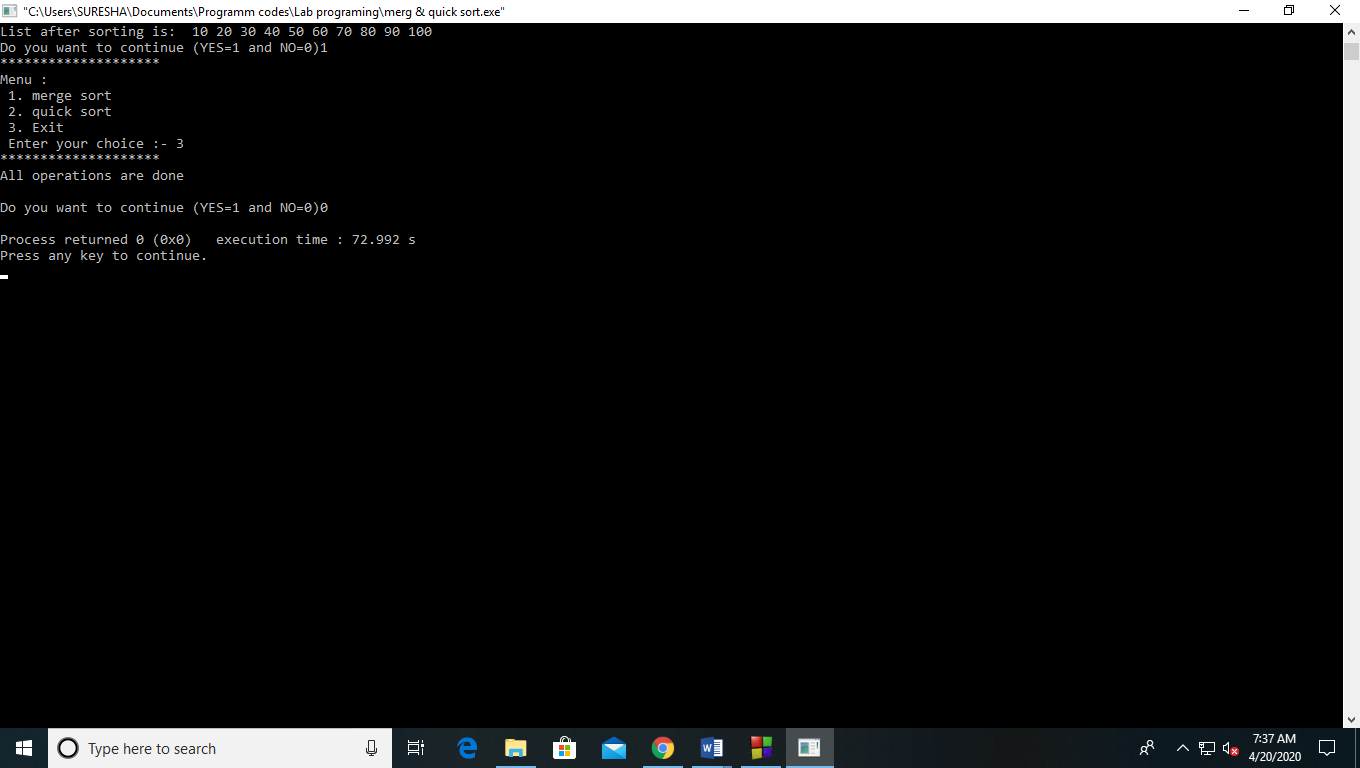
}

}

}

**Output:-**





**Experiment 13**

**Aim:** Write a Program to perform various operations on Binary Search Tree.

Search

Insert

delete

**Theory:**

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties −

• The left sub-tree of a node has a key less than or equal to its parent node's key.

• The right sub-tree of a node has a key greater than to its parent node's key.

• Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as −

• left\_subtree (keys) ≤ node (key) < right\_subtree (keys)

• Following are the basic operations of a BST −

• Search − Searches an element in a tree.

• Insert − Inserts an element in a tree.

• Delete - Delete an element from a tree

• Pre-order Traversal − Traverses a tree in a pre-order manner.

• In-order Traversal − Traverses a tree in an in-order manner.

• Post-order Traversal − Traverses a tree in a post-order manner.

**Code:-**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int info;

struct node\*left;

struct node\*right;

};

typedef struct node BST;

BST \*LOC, \*PAR;

void search(BST \*root, int item)

{

BST \*save,\*ptr;

if (root == NULL)

{

LOC = NULL;

PAR=NULL;

}

if (item == root -> info)

{

LOC = root;

PAR = NULL;

return;

}

if (item < root->info)

{

save = root;

ptr = root->left;

}

else

{

save = root;

ptr = root -> right;

}

while( ptr != NULL)

{

if (ptr -> info == item)

{

LOC = ptr;

PAR = save;

return;

}

if(item < ptr->info)

{

save = ptr;

ptr = ptr->left;

}

else

{

save = ptr;

ptr = ptr->right;

}

}

LOC = NULL;

PAR = save;

return;

}

struct node\* findmin(struct node\*r)

{

if (r == NULL)

return NULL;

else if (r->left!=NULL)

return findmin(r->left);

else if (r->left == NULL)

return r;

}

struct node\*insert(struct node\*r, int x)

{

if (r == NULL)

{

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

r->info = x;

r->left = r->right = NULL;

return r;

}

else if (x < r->info)

r->left = insert(r->left, x);

else if (x > r->info)

r->right = insert(r->right, x);

return r;

}

struct node\* del(struct node\*r, int x)

{

struct node \*t;

if(r == NULL)

printf("\nElement not found");

else if (x < r->info)

r->left = del(r->left, x);

else if (x > r->info)

r->right = del(r->right, x);

else if ((r->left != NULL) && (r->right != NULL))

{

t = findmin(r->right);

r->info = t->info;

r->right = del(r->right, r->info);

}

else

{

t = r;

if (r->left == NULL)

r = r->right;

else if (r->right == NULL)

r = r->left;

free(t);

}

return r;

}

int main()

{

struct node\* root = NULL;

int x, c = 1, z;

int element;

char ch;

printf("\nEnter an element: ");

scanf("%d", &x);

root = insert(root, x);

printf("\nDo you want to enter another element (Y=1 orN=0)");

scanf(" %c",&ch);

while (ch == '1')

{

printf("\nEnter an element:");

scanf("%d", &x);

root = insert(root,x);

printf("\nPress Y or N to insert another element: Y or N: ");

scanf(" %c", &ch);

}

while(1)

{

printf("Menu:");

printf("\n1 Insert an element ");

printf("\n2 Delete an element");

printf("\n3 Search for an element ");

printf("\n4 Exit ");

printf("\nEnter your choice: ");

scanf("%d", &c);

switch(c)

{

case 1:

printf("\nEnter the item:");

scanf("%d", &z);

root = insert(root,z);

break;

case 2:

printf("\nEnter the info to be deleted:");

scanf("%d", &z);

root = del(root, z);

break;

case 3:

printf("\nEnter element to be searched: ");

scanf("%d", &element);

search(root, element);

if(LOC != NULL)

printf("\n%d Found in Binary Search Tree \n",element);

else

printf("\nIt is not present in Binary Search Tree\n");

break;

case 4:

printf("\n");

return 0;

default:

printf("Enter a valid choice: ");

}

}

return 0;

}

**Output:-** 