Shri Vaishnav Vidyapeeth Vishwavidyala, Indore
Shri Vaishnav Institute of InformationTechnology
Department of Computer Science & Engineering

LAB FILE

Jan-Jun 2023

Name of the Student: JHILMIL GEETE

Enrollment: 2210DMTCSE12010

Program: B.Tech+M.Tech(CSE)

Section: H

Year/Sem: I / II

Subject Code: BTCS201N

Subject Name: Data Structure and Algorithms

Name of Subject Teacher: Dr. Sandeep Kumar Jain

Shri Vaishnav Institute of Information Technology

Department of Computer Science and Engineering

INDEX

NAME-JHILMIL GEETE
ENROLLMENT NO. – 2210DMTCSE12010

13	WAP to develop an algorithm for binary search, implement and test it.	16/05/2023	
14	WAP for implementation of Bubble Sort.	23/05/2023	
15	WAP for implementation of Insertion Sort.	20/06/2023	

Program No: 1

Aim/Title: WAP to find maximum and minimum number in an array.

Date: 14/03/2023

Tool: Dev c++

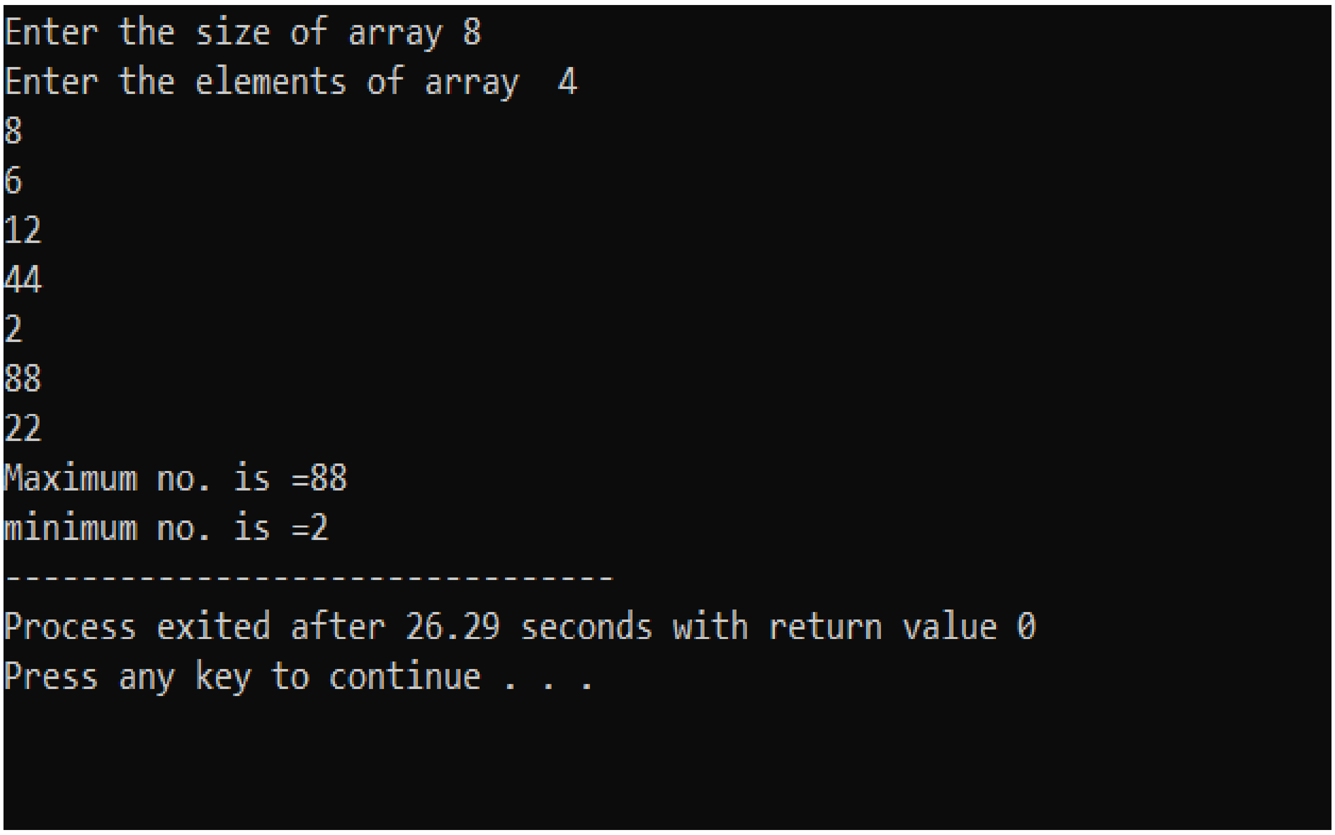
Algorithm/FlowChart:

1. Iintialize array and maximum size of array
2. Initialize variables i,n, for elements of array and max , min to store values of maximum and minimum no. of array
3. Take a loop to iterate elements of array
4. If (arr[i]>max) we update max with arr[i] to get max number
5. If (arr[i]<min) we update min with a[i] to get minimum value
6. After iterating through entire array the max and min will contain maximum and minimum numbers respectively

Source Code:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[100];
int i,max,min,n;
printf("Enter the size of array");
scanf("%d",&n);
printf("Enter the elements of array");
for(i=0;i<n;i++)
        {
                scanf("%d",&arr[i]);
                max=arr[0];

                min=arr[0];
        }
        for(i=1;i<n;i++)
        {
                if(arr[i]>max)
                {
                        max=arr[i];
                }
                else if(arr[i]<min)
                {
                        min=arr[i];
}
printf("Maximum no. is =%d",max);
        printf("\nminimum no. is =%d",min);
        return 0;
}
```

Output:-



```
Enter the size of array 8
Enter the elements of array  4
8
6
12
44
2
88
22
Maximum no. is =88
minimum no. is =2
--------------------------------
Process exited after 26.29 seconds with return value 0
Press any key to continue . . .
```

Program No: 2

Aim/Title: Perform searching and sorting in an array.

Date: 14/03/2023
Tool: Dev c++

Algorithm/FlowChart:

// to search element in array

```
1 Iterate the array using the loop.
2. Check whether the given key present in the array i.e. arr[i] == key.
3. If yes,
    print "Search Found".
4. Else
    print "Search Not Found".
```

// **to sort array**

```
1. Create an array of fixed size (maximum capacity), lets say 8.
2. Take n, a variable which stores the number of elements of the array, less than maximum capacity of array.
3. Iterate via for loop to take array elements as input, and print them.
4. The array elements are in unsorted fashion, to sort them, make a nested loop.
5. In the nested loop, the each element will be compared to all the elements below it.
6. In case the element is greater than the element present below it, then they are interchanged
7. After executing the nested loop, we will obtain an array in ascending order arranged elements.
```

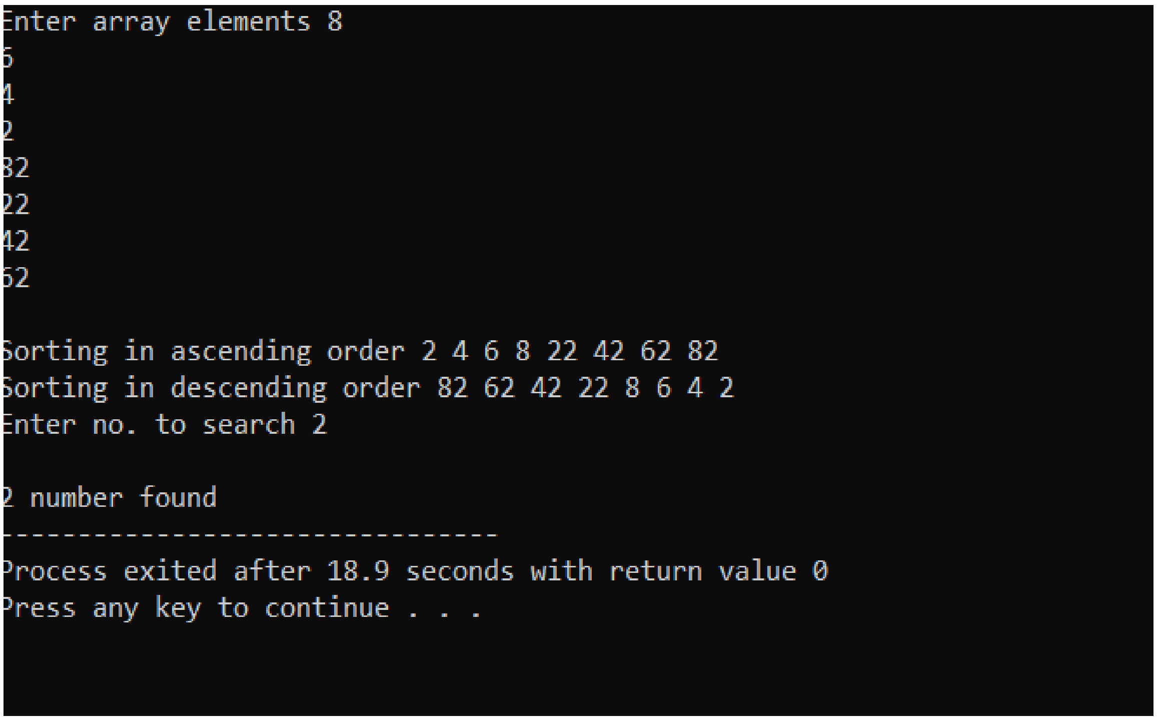
Source code:

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int a[8];
        int i,j,temp,n;
        printf("Enter array elements");
        for(i=0;i<8;i++)
        {
                scanf("%d",&a[i]);

        }
        //Ascending order
        for(i=0;i<8;i++)
        {

        for(j=i+1;j<8;j++)
        {
                if(a[i]>a[j])
       {
                        temp=a[i];
                        a[i]=a[j];
                        a[j]=temp;
            }
          }
}
        printf("\nSorting in ascending order");
```

```
    //descending order

    for(i=0;i<8;i++)
    {
        printf(" %d",a[i]);

    }
        for(i=0;i<8;i++)
    {

    for(j=i+1;j<8;j++)
    {
        if(a[i]<a[j])
   {
            temp=a[i];
            a[i]=a[j];
            a[j]=temp;
      }
     }
}
    printf("\nSorting in descending order");
    for(i=0;i<8;i++)
    {
        printf(" %d",a[i]);

    }
    //Searching of element
    printf("\nEnter no. to search ");
        scanf("%d",&n);

    for(i=0;i<8;i++)
```

```
        {
                if(a[i]==n)
                {
                        printf("\n%d number found ",n);
                }


        }




        return 0;
}
```

Output :



```
nter array elements 8
6
4
2
82
22
42
62

orting in ascending order 2 4 6 8 22 42 62 82
orting in descending order 82 62 42 22 8 6 4 2
nter no. to search 2

2 number found
--------------------------------
rocess exited after 18.9 seconds with return value 0
ress any key to continue . . .
```

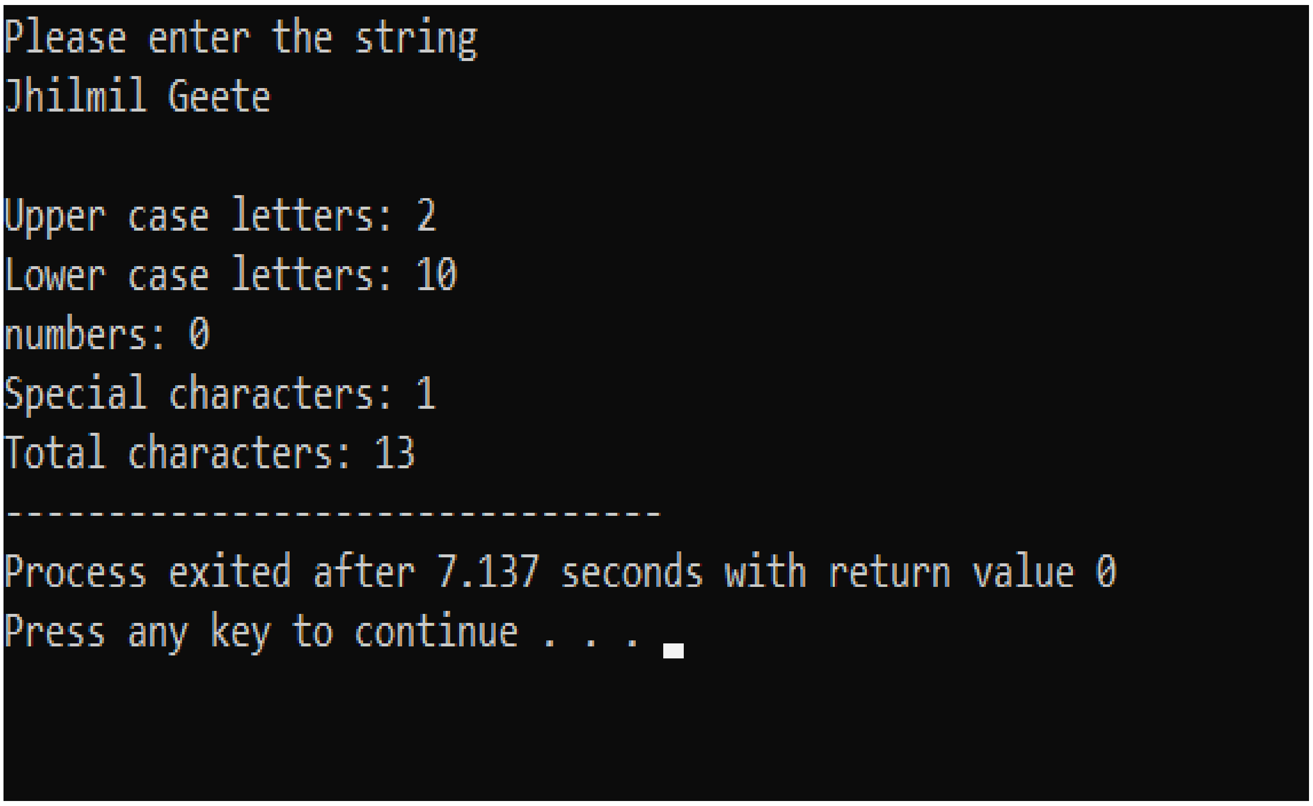
Program No: 3

AIM:- WAP to find lowercase, uppercase, wordcount, total characters and numericdigits in a
string.
DATE:- 21/03/2023
TOOL:- Dev C++
ALGORITHM:-
1. Scan string str from 0 to length-1.
2. check one character at a time
• if(str[i] >= ‘A’ and str[i] <=’Z’), then it is uppercase letter,
• if(str[i] >= ‘a’ and str[i] <=’z’), then it is lowercase letter,
• if(str[i] >= ‘0’ and str[i] <=’9’), then it is number,
else it is a special character
3. Print all the counters

SOURCE CODE :-

```
//WAP to find lowercase, uppercase, wordcount, total characters and numericdigits in a
string.
#include <stdio.h>
#include <stdlib.h>
int main()
{
 char str[50];
 int i;
 int upper=0,lower=0,num=0,special=0,
chars=0;
 printf("Please enter the string \n");
 gets(str);
 i=0;
 for( i = 0; str[i] != '\0'; i++) {
 //check for uppercase
 if(str[i]>='A' && str[i]<='Z') {
 upper++;
 //ceck lower case
 }else if(str[i]>='a' && str[i]<='z') {
 lower++;
 //Check numeric
 }else if(str[i]>='0' && str[i]<='9') {
 num++;
 }
 else{//check special character
 special++;
 }
 chars++;
 }
 printf("\nUpper case letters: %d",upper);
 printf("\nLower case letters: %d",lower);
 printf("\nnumbers: %d",num);
 printf("\nSpecial characters: %d",special);
 printf("\nTotal characters: %d",chars);
 return 0;
```

OUTPUT:-



```
Please enter the string
Jhilmil Geete

Upper case letters: 2
Lower case letters: 10
numbers: 0
Special characters: 1
Total characters: 13
--------------------------------
Process exited after 7.137 seconds with return value 0
Press any key to continue . . .
```

.

Program No: 4

AIM:- Perform insertion and deletion operation in 1D array.
DATE:- 21/03/2023
TOOL:- Dev C++

ALGORITHM OF DELETION :-

```
Step 01: Start
Step 02: [Initialize counter variable. ] Set i = index - 1
Step 03: Repeat Step 04 and 05 for i = index - 1 to i < size
Step 04: [Move ith element backward (left). ] set arr[i] = arr[i+1]
Step 05: [Increase counter. ] Set i ++;
Step 06: [End of step 03 loop. ]
Step 07: [Increase size of the array. ] set size++;
Step 08: Stop
```

SOURCE CODE :-

```
//Perform deletion operation in 1D array.
#include <stdio.h>
int main() {
 int i,arr[50], size, index;
 printf("Enter the size of the array: ");
 scanf("%d", &size);
 printf("Enter %d elements:\n", size
 for (i = 0; i < size; i++) {
 scanf("%d", &arr[i]);
 }
printf("Enter the index of the element you want to delete: ");
 scanf("%d", &index);
 // Shift all elements after the index to the left
 for (i = index; i < size - 1; i++) {
 arr[i] = arr[i + 1];
 }
 // Decrease the size of the array by 1
 size--;
 printf("The updated array is:\n");
 for (i = 0; i < size; i++) {
 printf("%d ", arr[i]);
 }
 printf("\n");
 return 0;
}
```

OUTPUT:-

ALGORITHM OF INSERTION:-

AT BEGINNING –

1.First get the element to be inserted.
2.Increase size of array.
3.Then right sift all element of array a[i-1]=a[i-2].
4.Then insert the new element at index 0.

AT THE END –

1.First get the element to be inserted.
2.Increase size of array.
3.Then insert the new element at index of size-1.

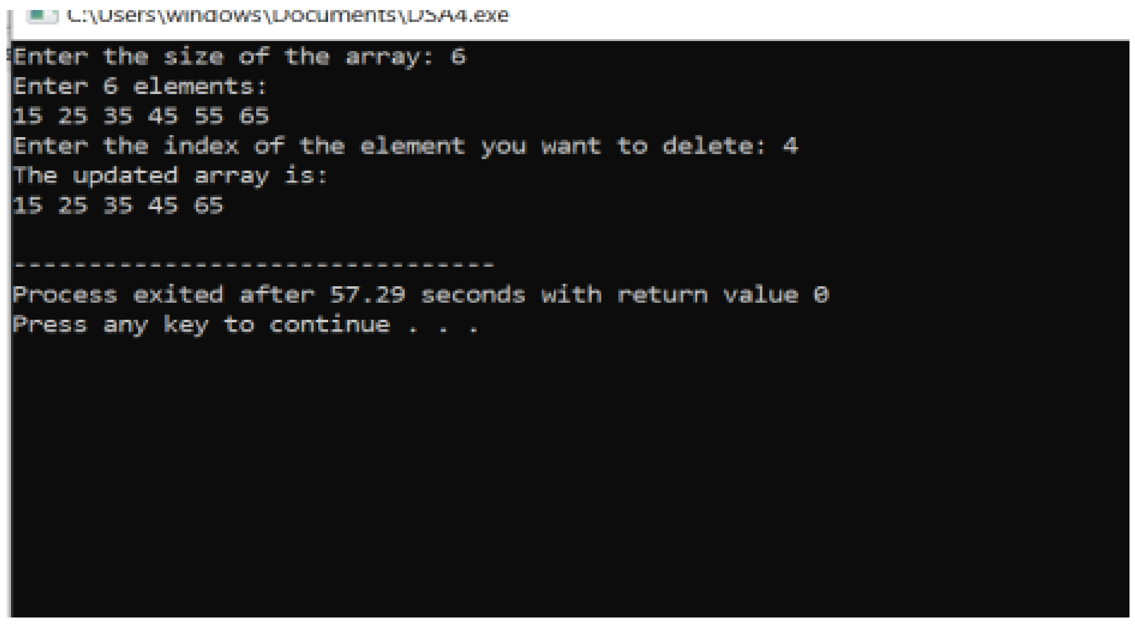
At any index –

1. First get the element to be inserted, say x.
2. Then get the position at which this element is to be inserted, say index and increase the size of array.
3. Then shift the array elements from this position to one position forward(towards right), and do this for all the other elements next to index.
4. Insert the element x now at the position index, as this is now empty.

SOURCE CODE:-

```
#include <stdio.h>
int main()
{
 int size, i, new;
 printf("enter the size of array\n");
 scanf("%d", &size);
 int a[size];
 printf("enter the elements in array \n");
 for (i = 0; i < size; i++)
 {
 scanf("%d", &a[i]);
 }
//At the beginning
 printf("Insert the new element at beggining\n");
 scanf("%d",&new);
 size++;
 for(i=size; i>1; i--)
 {
 a[i-1]=a[i-2];
 }
 a[0]=new;
 //At the end
 printf("Insert the new element at end\n");
 scanf("%d",&new);
 size++;
 a[size-1]=new;
//At any position
 int index,element;
 printf("Enter the position where you want to insert the new element\n");
 scanf("%d",&index);
 printf("Enter the new element\n");
 scanf("%d",&element);
 size ++;
 for(i=size-1; i>=index; i--){
 a[i+1]=a[i];
 }
 a[index]=element;
 printf("Resultant array elements\n");
 for (i = 0; i < size; i++)
 {
 printf("%d\n", a[i]);
 }
 return 0;
}
```

OUTPUT:



```
C:\Users\windows\Documents\DSA4.exe
Enter the size of the array: 6
Enter 6 elements:
15 25 35 45 55 65
Enter the index of the element you want to delete: 4
The updated array is:
15 25 35 45 65

--------------------------------
Process exited after 57.29 seconds with return value 0
Press any key to continue . . .
```

EXPERIMENT NO.-05

AIM:- WAP to perform various operation in the single linked list.

(g) CreateFirstNode

(h) InsertAtBeginning

(i) InsertAtLast

(j) DeleteFromBeginning

(k) DeleteFromLast

(l) DisplayList

DATE:- 26/04/2023

TOOL:- Dev C++

ALGORITHM:- 1. Declare a structure which contain data , and a pointer(next).

2. Declare a starting point as head.

3. For create first node

i. Take a pointer and assign DMA.

ii. new-> data = data; iii. new->next=head; iv. assign new to head

4. For insertion at start

i. take a new pointer and assign DMA.

ii. new->data=data; iii. new->next=head; iv. assign new to head;

5. For insert at end

i. take a new pointer and assign DMA.

ii. take another pointer which start from head.

iii. new->data= data; iv. traverse the list untill you reach the last node.

v. set the next pointer of the last node to new node.

vi. new-> next = NULL;

6. Delete at start
i. take a pointer (ptr) which start from head.

ii. head=head->next; iii. And then free(ptr);

7. Delete at end.
i. take pointer (p) which start from head.

ii. take another pointer (q) which start from head->next. iii.

traverse the list untill a reach last node.

iv. assign p-> next=NULL;

v. free (q); 8. For print

i. take a pointer (ptr).

ii. traverse list from head to last node. iii. and every

time print element.

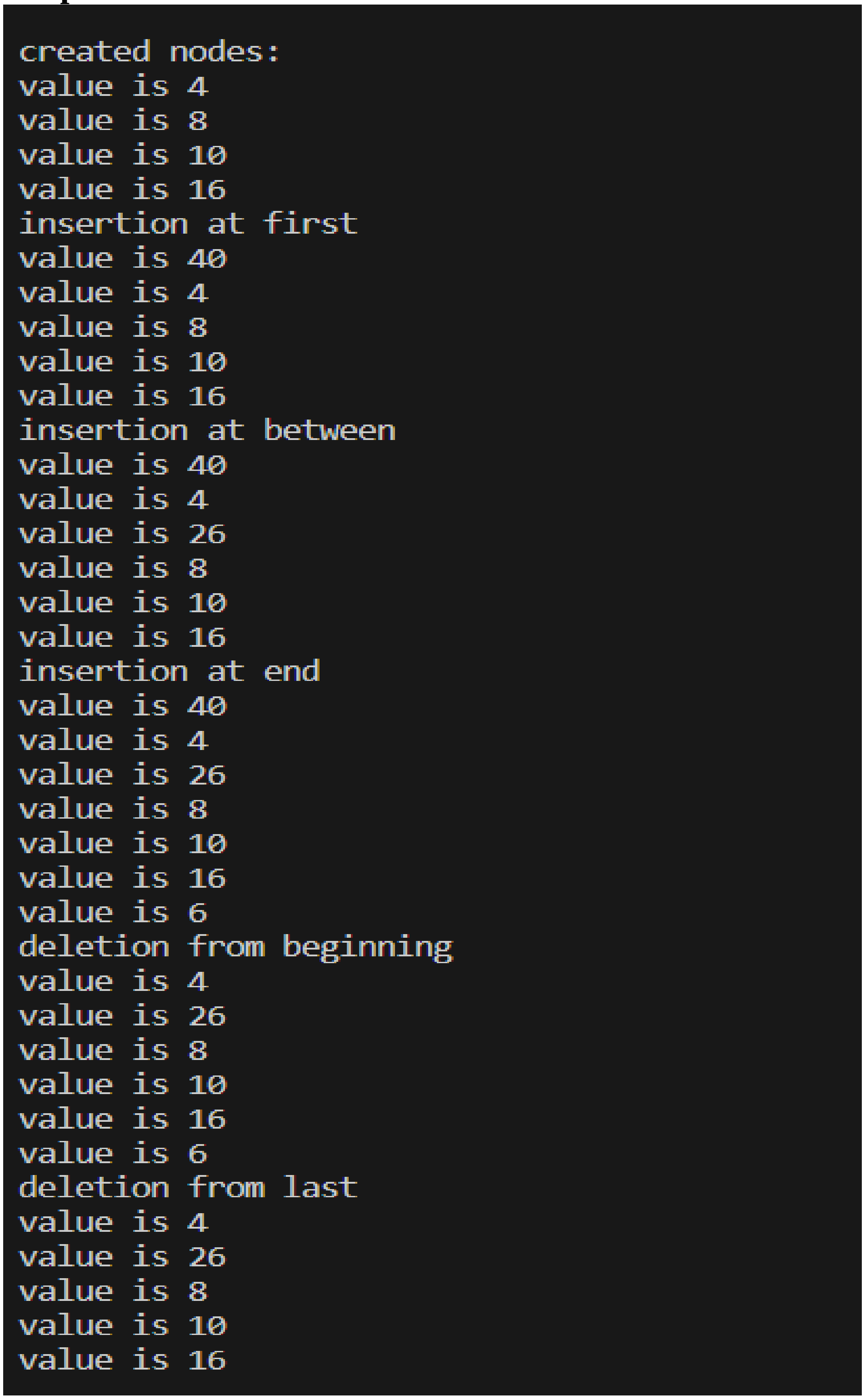
iii. ptr=ptr->next

Source code :

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

struct node* deleteFromBeginning(struct node* head);
struct node* deleteFromLast(struct node* head);

//to create node
struct node
{
   int data;
   struct node *next;
   int *p;
};
//To display linklist
void tra(struct node*ptr)
{
   while(ptr!=0) {
   printf("\nvalue is %d",ptr->data);
   ptr=ptr->next;
    }
}//insert at first
struct node*insertatfirst(struct node *head,int data)
{
struct node*ptr=(struct node*)malloc(sizeof(struct node));
   ptr->next=head;
   ptr->data=data;
   return ptr;
}
//insert at between
struct node*insertatbtw(struct node*head,int data,int index)
{
struct node*ptr=(struct node*)malloc(sizeof(struct node));
struct node*p=head;
int i=0;
while(i!=index-1)
{
   p=p->next;
   i++;
}
   ptr->data=data;
   ptr->next=p->next;
   p->next=ptr;
   return head;
}
```

```
//insert at end
struct node*insertatend(struct node*head,int data)
{
struct node*ptr=(struct node*)malloc(sizeof(struct node));
ptr->data=data;
struct node*p=head;

while(p->next!=NULL){
   p=p->next;
   }
p->next=ptr;
ptr->next=0;
return head;
}

// Delete the first node
struct node* deleteFromBeginning(struct node* head)
{
   if (head == NULL)
      return NULL;

   struct node* temp = head;
   head = head->next;
   free(temp);

   return head;
}

// Delete the last node
struct node* deleteFromLast(struct node* head)
{
   if (head == NULL)
      return NULL;

   if (head->next == NULL) {
      free(head);
      return NULL;
   }

   struct node* temp = head;
   while (temp->next->next != NULL)
      temp = temp->next;

   free(temp->next);
   temp->next = NULL;

   return head;
}

int main()
{
struct node *head;
struct node *second;
struct node *third;
struct node *fourth;
head=(struct node*)malloc(sizeof(struct node));
second=(struct node*)malloc(sizeof(struct node));
third=(struct node*)malloc(sizeof(struct node));
fourth=(struct node*)malloc(sizeof(struct node));
head->data=4;
head->next=second;
second->data=8;
second->next=third;
third->data=10;
third->next=fourth;
fourth->data=16;
fourth->next=NULL;
printf("created nodes:");
tra(head);
printf("\ninsertion at first");
head=insertatfirst(head,40);
tra(head);
printf("\ninsertion at between");
head=insertatbtw(head,26,2);
tra(head);
printf("\ninsertion at end");
head=insertatend(head,6);
tra(head);
printf("\ndeletion from beginning");
head = deleteFromBeginning(head);
tra(head);
printf("\ndeletion from last");
head = deleteFromLast(head);
tra(head);

return 0;
}
```

Output:



```
created nodes:
value is 4
value is 8
value is 10
value is 16
insertion at first
value is 40
value is 4
value is 8
value is 10
value is 16
insertion at between
value is 40
value is 4
value is 26
value is 8
value is 10
value is 16
insertion at end
value is 40
value is 4
value is 26
value is 8
value is 10
value is 16
value is 6
deletion from beginning
value is 4
value is 26
value is 8
value is 10
value is 16
value is 6
deletion from last
value is 4
value is 26
value is 8
value is 10
value is 16
```

EXPERIMENT NO.-06

AIM:- WAP to perform insertion and deletion operation in circular linked list.

DATE:- 04/05/2023

TOO:-Dev C++

ALGORITHM:- 1)Declare a structure which contain data field and a pointer(next).

2)Create a pointer (tail)=NULL.

3)Create first node

1.take a pointer and assign memeory using malloc function(dynamic memory allocation).

2.Set a pointer new -> data = data;

3.The next field of new point its self.

4)Insert at start

1..take a pointer and assign memeory using malloc function(dynamic memory allocation).

2.new->data=data;

3.new->next=tail; 4.tail-
>next=new;

5)Update the tail of the list to point to the new.

6)Insert at end

1.allocate memory for the new node.

2.assign data to new node.

3.set the next pointer of the new to the next of the tail of the list.

4.set the next of the tail to new.

5.assign new to tail.

6)Delete at first

1.create a pointer (p) and intialise it to the next of the tail.

2.assign p->next to the tail->next.

3.then free to p.

7)delete at end

1. take two

pointer

p=tail->next;

q=tail->next-

>next;

2. traverse the list unitl q do not reach tail.

3. assign p->next to tail->next.

4. assign next of q is NULL.

5. free(q);

6. assign p to tail.

8. for Traverse the list

1. first check it tail->next=NULL,so list is empty.

2. take a pointer(temp) which is start from next of tail.

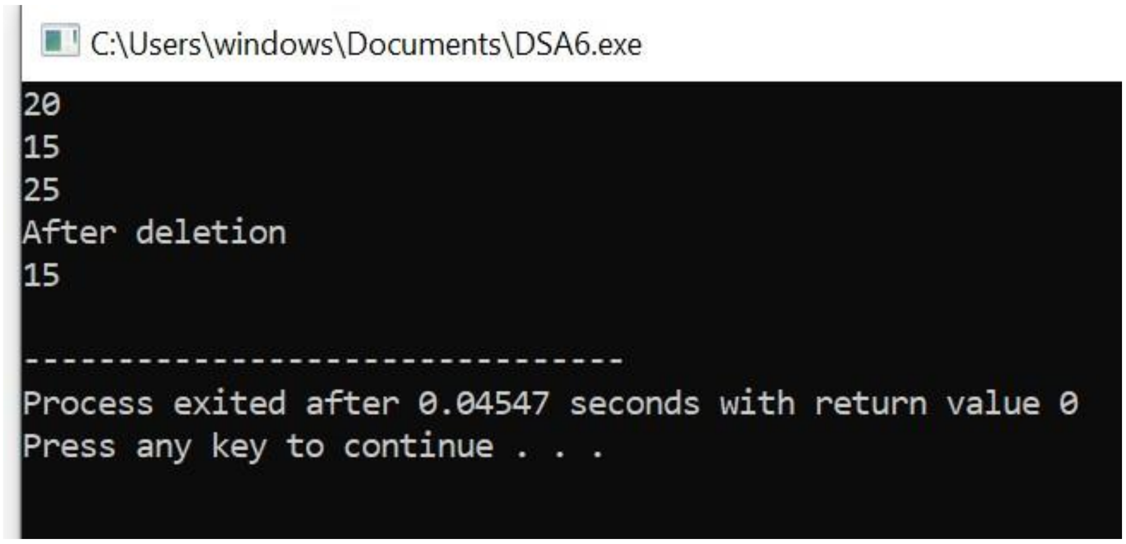
3. traverse the list unitl temp not equal tail->next.

4. print the element of list.

SOURCE CODE:-

```
//All operation of circular linked list
#include<stdio.h

>

#include<stdlib.

h> struct node {

int data;     struct

node * next;

};
typedef struct node node;

node* createfirstnode(int data)

{
    node *new;     new=(struct

node*)malloc(sizeof(struct node));     new-

>data=data;     new->next=new;     return new;

}
node*insert_at_start(node*tail,int data)
{
    node*new =(node*)malloc(sizeof(node));

new->data=data;     new->next=tail;     tail-

>next=new;     return tail;

}
node*insert_at_end(node*tail,int data)
{
    node*new =(node*)malloc(sizeof(node));

new->data=data;     new->next=tail->next;

tail->next=new;     tail=new;     return tail;

}
node*delete_at_first(node*tail)

{     node*p=tail->next;     tail-

>next=p->next;     free(p);

return tail;

}
node*delete_at_last(node*tail)

{     node*p=tail->next;

node*q=tail->next->next;
while(q!=tail) {        p=p-

>next;        q=q->next;

    }
    p->next=tail->next;    q-

>next=NULL;

    free(q);

tail=p;

return

tail;

 }
node*printlist(node*tail) {      if(tail-

>next==NULL){      printf("There is no

node in linked list\n");    return 0;

    }
    node*temp=tail->next;
    do {
        printf("%d\n",temp-

>data);        temp=temp-

>next; } while(temp!=tail-

>next);     return temp;

    }
int main()
{
    struct node *tail=NULL;

tail=createfirstnode(15);

tail=insert_at_start(tail,20);

tail=insert_at_end(tail,25);

printlist(tail);

printf("After deletion\n");

tail=delete_at_first(tail);

tail=delete_at_last(tail);

printlist(tail);        return 0;

}
```

OUTPUT:-



C:\Users\windows\Documents\DSA6.exe

```
20
15
25
After deletion
15

--------------------------------
Process exited after 0.04547 seconds with return value 0
Press any key to continue . . .
```

EXPERIMENT NO.-07

AIM:- Write a menu driven program to implement various operations as push, pop, display, isFull and isEmpty in a stack with the help of static memory allocation.

DATE:- 11/05/2023

TOOL:- Dev C++

ALGORITHM:- 1. Define size 5;

2. Initialize global vairiable top = -1 and integer arr.

3. Push

 1 First we check the stack is full or not .
 2. If empty so we increment top to top +1 and insert data at top.
4. Pop
1. First we check the stack in empty or not.

2. If it is not empty we do pop operation .

3. Initialize data (variable).

4. We transfer the data which is at top in data.

5. Them we increase top from top-1 .

 print
 1.Initialize i .
2. With the help of for loop condition (i=top,i>-1 ,i--)

3. Print element of array

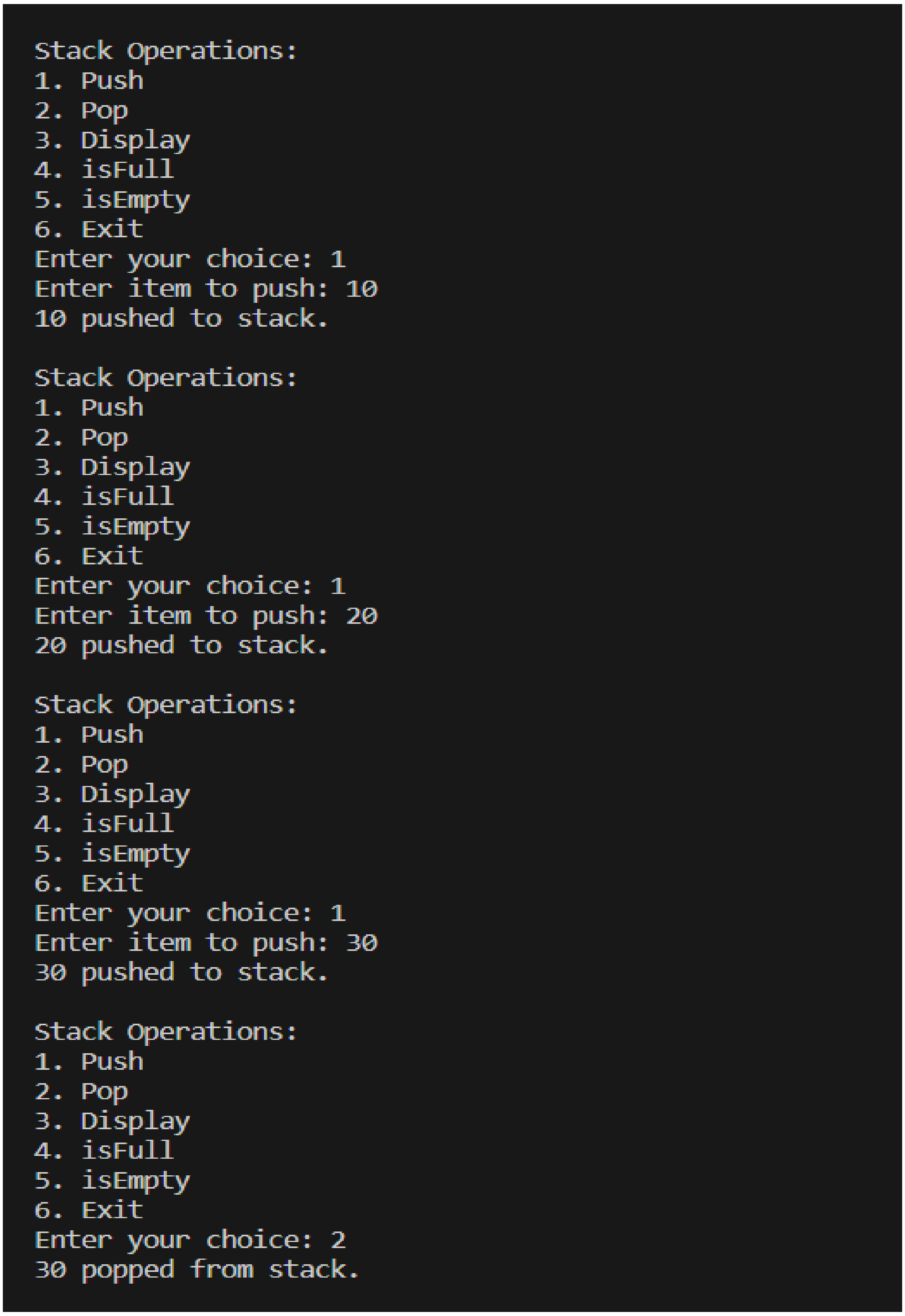
6. Return data;\

7. Them we can print which element get deleted .

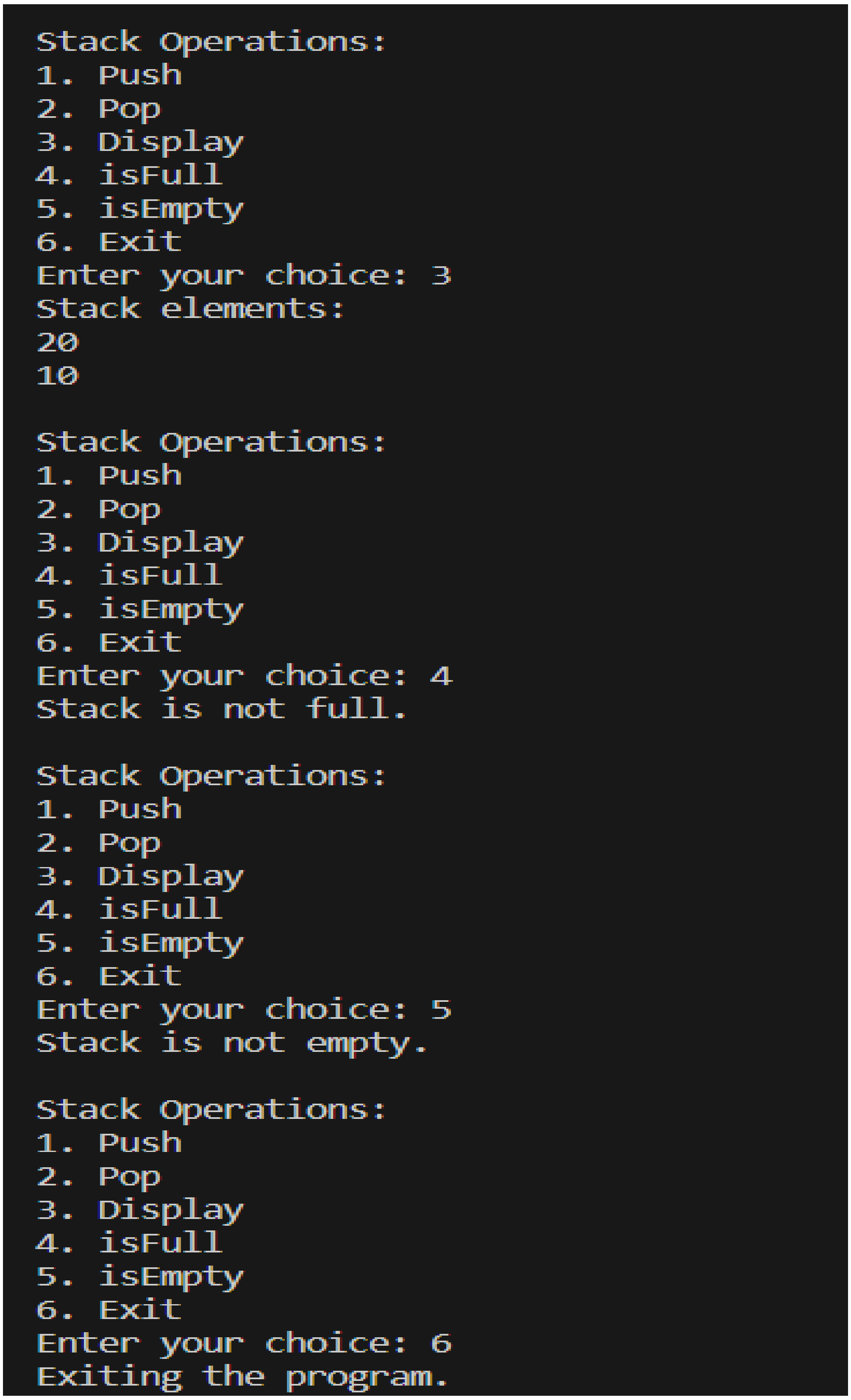
SOURCE CODE:-

```
#include <stdio.h>
#define SIZE 5

int top = -1;
int stack[SIZE];

// Push operation
void push(int data) {
    if (top == SIZE - 1) {
        printf("Stack Overflow! Cannot push item.\n");
        return;
    }
    stack[++top] = data;
    printf("%d pushed to stack.\n", data);
}

// Pop operation
void pop() {
    if (top == -1) {
        printf("Stack Underflow! Cannot pop item.\n");
        return;
    }
    printf("%d popped from stack.\n", stack[top--]);
}


// Print operation
void print() {
    if (top == -1) {
        printf("Stack is empty.\n");
        return;
    }
    printf("Stack elements:\n");
    for (int i = top; i >= 0; i--) {
        printf("%d\n", stack[i]);
    }
}
int main() {
    int choice, item;
    while (1) {
        printf("\nStack Operations:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Print\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter item to push: ");
                scanf("%d", &item);
                push(item);
                break;
            case 2:
                pop();
                break;
            case 3:
                print();
                break;
            case 4:
                printf("Exiting the program.\n");
                return 0;
            default:
                printf("Invalid choice. Please enter a valid option.\n");    }
    }
    return 0;
}
```

OUTPUT:-



```
Stack Operations:
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
Enter your choice: 1
Enter item to push: 10
10 pushed to stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
Enter your choice: 1
Enter item to push: 20
20 pushed to stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
Enter your choice: 1
Enter item to push: 30
30 pushed to stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
Enter your choice: 2
30 popped from stack.
```



```
Stack Operations:
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
Enter your choice: 3
Stack elements:
20
10

Stack Operations:
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
Enter your choice: 4
Stack is not full.

Stack Operations:
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
Enter your choice: 5
Stack is not empty.

Stack Operations:
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
Enter your choice: 6
Exiting the program.
```

EXPERIMENT NO.-08

AIM:- Write a menu driven program to implement various operations as push, pop, display, isFull and isEmpty in a stack with the help of dynamic memory allocation.

DATE:- 18/05/2023

TOOL:- Dev C++

ALGORITHM:- 1. First create a node which contains data and a pointer (next) and a another pointer top.

2. Create a node

Initialize a pointer and assign memory with the help of malloc.

Assign data in his data field.

Next of the new is equal to NULL.

3.Push

Initialize a pointer and assign memory with the help of malloc.

Assign data in his data field.

next of the new is equal to top.

assign new in top.

4.Pop

First we check the linked list is empty or not.

If it is not empty we pop.

Take a pointer (P) = top;

Then increase top is equal to top's next.

Then free P.

5.Print

Take a pointer (ptr) = top.

Traverse the list until ptr reaches NULL.

Then print elements of list.

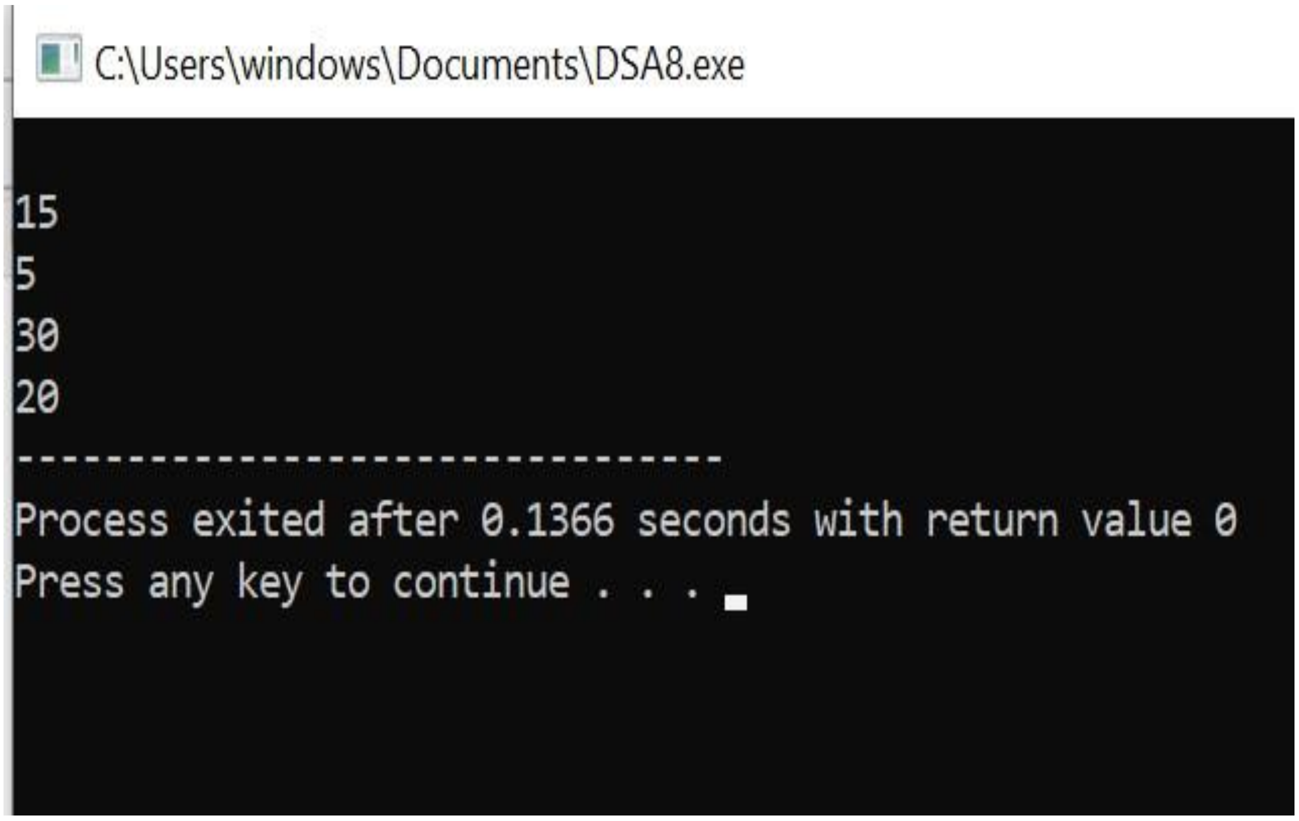
Increase ptr = ptr->next;

SOURCE CODE:-

```
//STACK BY LINKED LIST
#include<stdio.h>

#include<stdlib.h>

struct node{

            int data;

struct node * next;

}*top; typedef struct node node;

node*createnode(int data){

node*new=(node*)malloc(sizeof(node));      new-

>data=data;         new->next=NULL;

return new;

}
node*push(node*top,int data){

node*new=(node*)malloc(sizeof(node));      new-

>data=data;         new->next=top;   top=new;

return top;

}
node*pop(node*top){
                        if(isempty()){

            return 0;

            }
   node*p=top;
            top=top-

>next;    free(p);

return top;

}
int isempty(){
            if(top==NULL){

printf("Stack is Underflow\n");

            }
            else{

return 0;

            }
}
```

```
void printlist(node*top)
 {
                node*ptr=top;
while(ptr!=NULL){

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

     ptr=ptr->next;
    }
}
int main()
{
             top=createnode(20);

top=push(top,30);

top=push(top,5);

top=push(top,15);

top=push(top,8);

top=pop(top);

                printlist(top);
               return 0;
}
```

OUTPUT:-



C:\Users\windows\Documents\DSA8.exe

```
15
5
30
20
--------------------------------
Process exited after 0.1366 seconds with return value 0
Press any key to continue . . .
```

Program No:- 09

Aim/Title: WAP for Tower of Hanoi using recursion.

Date: 25/05/2023

Tool: Dev C++

Algorithm/Flow Chart:-

Algorithm:-

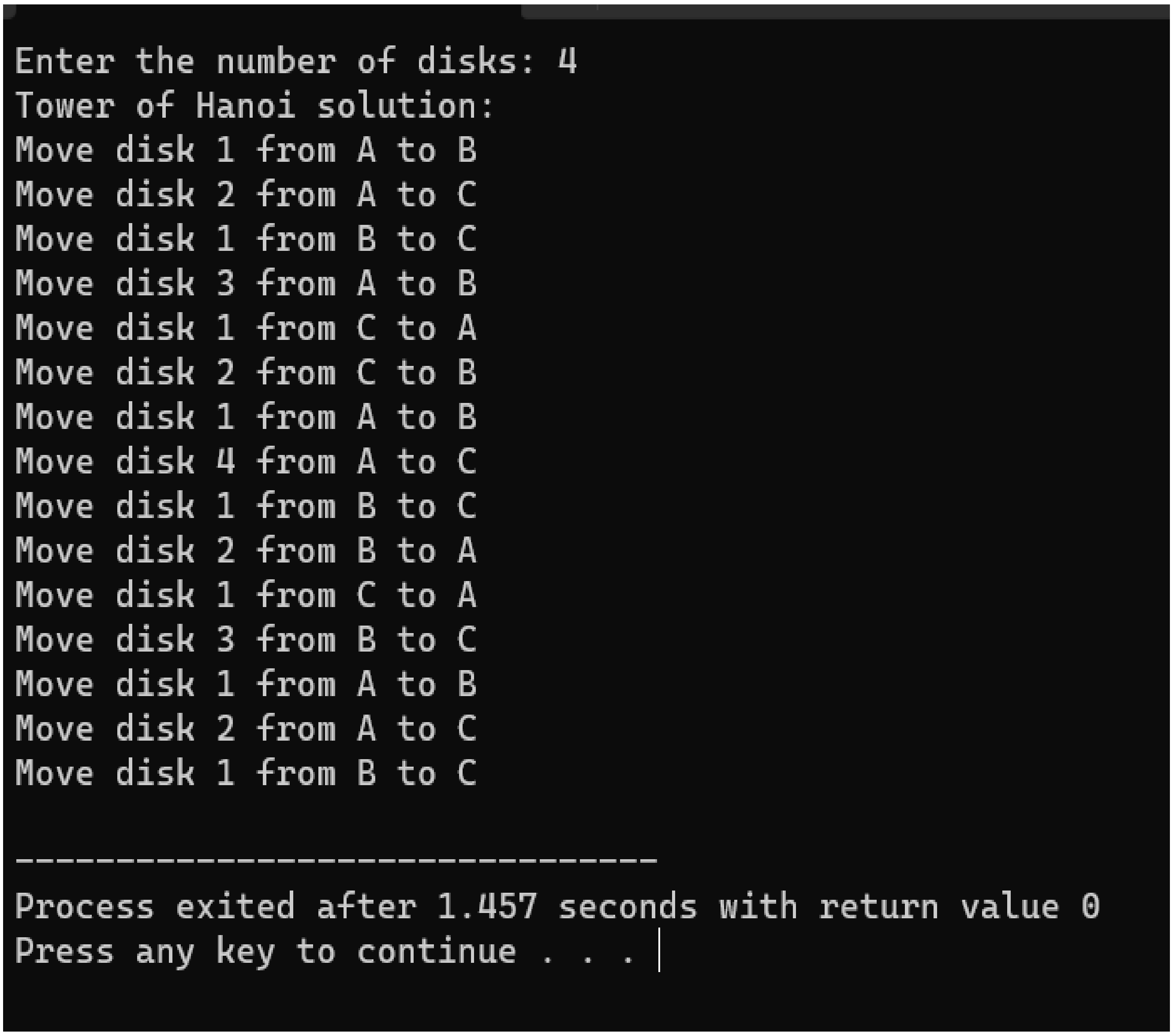
1. Define a function towerOfHanoi that takes four parameters:
- **n**: the number of disks to be moved
- **source**: the tower from which disks are initially placed
- **auxiliary**: the auxiliary tower used for intermediate moves
- **destination**: the tower where disks are to be moved
2. Inside the towerOfHanoi function:
- If n is equal to 1 (base case), then:
  I. Print the move of the disk from the source tower to the destination tower
  II. Return from the function.
- Otherwise (recursive case), do the following:
  I. Recursively call the towerOfHanoi function with n-1 disks, moving them from the source tower to the auxiliary tower.
  II. Print the move of the nth disk from the source tower to the destination tower.
  III. Recursively call the towerOfHanoi function with n-1 disks, moving them from the auxiliary tower to the destination tower.

3. In the main function:
   - Prompt the user to enter the number of disks.
   - Read the input value and store it in a variable **numDisks**.
   - Print the message indicating the start of the Tower of Hanoi solution.
   - Call the **towerOfHanoi** function with **numDisks** disks, using towers A, B, and C as the source, auxiliary, and destination towers, respectively.

Source Code:-

```
#include <stdio.h>

void towerOfHanoi(int n, char source, char auxiliary, char destination) {
   if (n == 1) {
      printf("Move disk 1 from %c to %c\n", source, destination);
      return;
   }

   towerOfHanoi(n - 1, source, destination, auxiliary);
   printf("Move disk %d from %c to %c\n", n, source, destination);
   towerOfHanoi(n - 1, auxiliary, source, destination);
}

int main() {
   int numDisks;
   printf("Enter the number of disks: ");
   scanf("%d", &numDisks);

   printf("Tower of Hanoi solution:\n");
   towerOfHanoi(numDisks, 'A', 'B', 'C');

   return 0;
}
```

Outcomes :-



```
Enter the number of disks: 4
Tower of Hanoi solution:
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Move disk 3 from A to B
Move disk 1 from C to A
Move disk 2 from C to B
Move disk 1 from A to B
Move disk 4 from A to C
Move disk 1 from B to C
Move disk 2 from B to A
Move disk 1 from C to A
Move disk 3 from B to C
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C

--------------------------------
Process exited after 1.457 seconds with return value 0
Press any key to continue . . .
```

Program No:- 10

Aim/Title: iilustrate queue implementation using array in c with following operation as enQueue, deQueue, isEmpty, displayQueue.

Date: 09/05/2023

Tool: Dev C++

Algorithm/Flow Chart:-

Algorithm:-

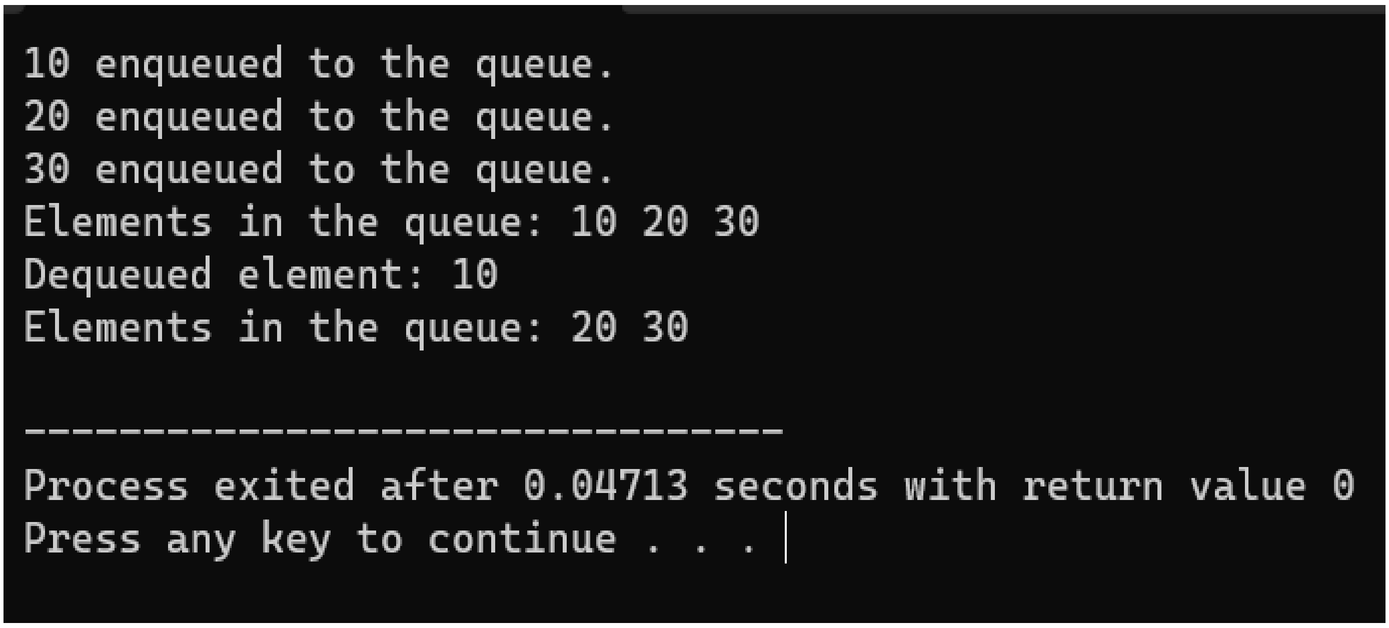
1. Create a structure Queue with an array items, front, and rear variables.
2. Initialize the queue by setting front and rear to -1.
3. Implement the isEmpty function to check if the queue is empty by comparing front with -1.
4. Implement the isFull function to check if the queue is full by comparing rear with the maximum size minus 1.
5. Implement the enQueue function to add an element to the queue:
    - Check if the queue is full. If it is, display an error message and return.
    - If the queue is empty, set **front** to 0.
    - Increment **rear** by 1 and assign the given value to **items[rear]**.
6. Implement the deQueue function to remove an element from the queue:
    - Check if the queue is empty. If it is, display an error message and return -1.
    - Retrieve the value at items[front].
    - Increment front by 1.
    - If front is greater than rear, reset the queue by setting front and rear to -1.
    - Return the dequeued value.
7. Implement the displayQueue function to display the elements in the queue:
    - Check if the queue is empty. If it is, display an appropriate message.
    - Iterate from **front** to **rear** and print each element.

Source Code:-

```
#include <stdio.h>

#define MAX_SIZE 100

// Structure to represent a queue
typedef struct {
   int items[MAX_SIZE];
   int front;
   int rear;
} Queue;

// Function to initialize a queue
void initializeQueue(Queue* queue) {
   queue->front = -1;
   queue->rear = -1;
}

// Function to check if the queue is empty
int isEmpty(Queue* queue) {
   return queue->front == -1;
}

// Function to check if the queue is full
int isFull(Queue* queue) {
   return queue->rear == MAX_SIZE - 1;
}

// Function to add an element to the queue (enQueue)
void enQueue(Queue* queue, int value) {
   if (isFull(queue)) {
       printf("Queue is full. Cannot enqueue %d\n", value);
       return;
   }

   if (isEmpty(queue)) {
       queue->front = 0;
   }

   queue->rear++;
   queue->items[queue->rear] = value;
   printf("%d enqueued to the queue.\n", value);
}

// Function to remove an element from the queue (deQueue)
int deQueue(Queue* queue) {
   if (isEmpty(queue)) {
       printf("Queue is empty. Cannot dequeue.\n");
       return -1;
   }

   int value = queue->items[queue->front];
   queue->front++;

   if (queue->front > queue->rear) {
       // Reset the queue
       queue->front = -1;
       queue->rear = -1;
   }

   return value;
}
```

```
// Function to display the elements in the queue
void displayQueue(Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return;
    }

    printf("Elements in the queue: ");
    for (int i = queue->front; i <= queue->rear; i++) {
        printf("%d ", queue->items[i]);
    }
    printf("\n");
}
int main() {
    Queue queue;
    initializeQueue(&queue);
    enQueue(&queue, 10);
    enQueue(&queue, 20);
    enQueue(&queue, 30);
    displayQueue(&queue);

    int dequeuedElement = deQueue(&queue);
    if (dequeuedElement != -1) {
        printf("Dequeued element: %d\n", dequeuedElement);
    }
    displayQueue(&queue);

    return 0;
}
```

Outcomes :-



```
10 enqueued to the queue.
20 enqueued to the queue.
30 enqueued to the queue.
Elements in the queue: 10 20 30
Dequeued element: 10
Elements in the queue: 20 30

--------------------------------
Process exited after 0.04713 seconds with return value 0
Press any key to continue . . .
```

Program No:- 11

Aim/Title: Illustrate queue implementation using linked list with following operation as enQueue, deQueue, isEmpty, displayQueue.

Date: 09/05/2023

Tool: Dev C++

Algorithm/Flow Chart:-

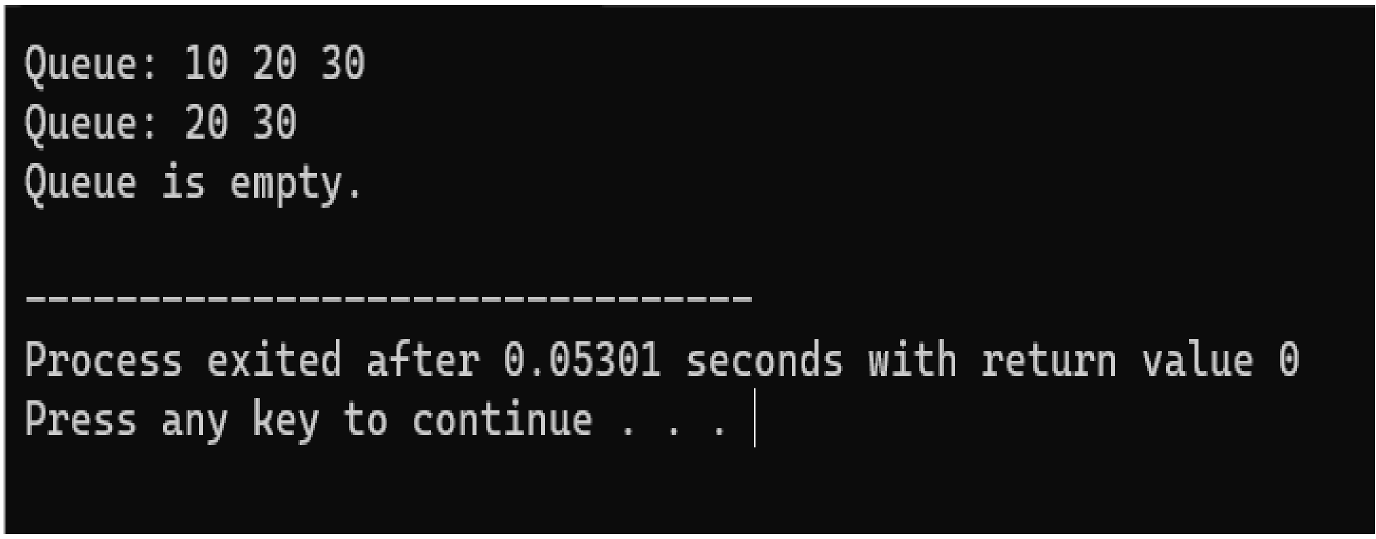
Algorithm:-

1. Start
2. Create a structure for each node in the linked list (Node) with data and next pointer.
3. Create a structure for the Queue with front and rear pointers.
4. Create a function ‘createQueue’ to create an empty queue:
- Allocate memory for the Queue structure.
- Set the front and rear pointers to NULL.
- Return the created queue.
5. Create a function isEmpty to check if the queue is empty:
- If the front pointer is NULL, return 1 (true); otherwise, return 0 (false).
6. Create a function enQueue to insert an element into the queue:
- Create a new node.
- Set the data of the new node to the given element.
- Set the next pointer of the new node to NULL.
- If the queue is empty, set both front and rear pointers to the new node.
- Otherwise, set the next pointer of the current rear node to the new node.
- Set the rear pointer to the new node.
7. Create a function ‘deQueue’ to remove an element from the queue:
- If the queue is empty, print an appropriate message and return.
- Store the front node in a temporary variable.
- Move the front pointer to the next node.
- If the front becomes NULL, set the rear pointer to NULL as well.
- Free the memory occupied by the dequeued node.
8. Create a function displayQueue to print the elements of the queue:
- If the queue is empty, print an appropriate message and return.
- Traverse the queue from the front to the rear.
- Print each element.
9. Main function:
- Create an empty queue using createQueue.
- Perform enqueue, dequeue, and display operations on the queue to demonstrate its functionality.
10. End

Source Code:-

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

// Structure for each node in the linked list
struct Node {
   int data;
   struct Node* next;
};

// Structure for the Queue
struct Queue {
   struct Node* front;
   struct Node* rear;
};

// Function to create an empty queue
struct Queue* createQueue() {
   struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
   queue->front = NULL;
   queue->rear = NULL;
   return queue;
}

// Function to check if the queue is empty
int isEmpty(struct Queue* queue) {
   return (queue->front == NULL);
}

// Function to enqueue an element into the queue
void enQueue(struct Queue* queue, int data) {
   // Create a new node
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;

   // If the queue is empty, make the new node both front and rear
   if (isEmpty(queue)) {
       queue->front = newNode;
       queue->rear = newNode;
       return;
   }

   // Otherwise, add the new node at the end and update rear
   queue->rear->next = newNode;
   queue->rear = newNode;
}

// Function to dequeue an element from the queue
void deQueue(struct Queue* queue) {
   // If the queue is empty, no elements to dequeue
   if (isEmpty(queue)) {
       printf("Queue is empty. Cannot dequeue.\n");
       return;
   }

   // Store the front node and move front to the next node
   struct Node* temp = queue->front;
   queue->front = queue->front->next;

   // If the front becomes NULL, update rear as NULL as well
   if (queue->front == NULL) {
       queue->rear = NULL;
   }

   // Free the memory occupied by the dequeued node
   free(temp);
}

// Function to display the elements of the queue
void displayQueue(struct Queue* queue) {
   // If the queue is empty, display appropriate message
   if (isEmpty(queue)) {
       printf("Queue is empty.\n");
       return;
   }

   // Traverse the queue and print each element
   struct Node* current = queue->front;
   printf("Queue: ");
   while (current != NULL) {
       printf("%d ", current->data);
       current = current->next;
   }
   printf("\n");
}

// Main function
int main() {
   // Create an empty queue
   struct Queue* queue = createQueue();

   // Demonstrate queue operations
   enQueue(queue, 10);
   enQueue(queue, 20);
   enQueue(queue, 30);

   displayQueue(queue);  // Output: Queue: 10 20 30

   deQueue(queue);

   displayQueue(queue);  // Output: Queue: 20 30

   deQueue(queue);
   deQueue(queue);

   displayQueue(queue);  // Output: Queue is empty.

   return 0;
}
```

Outcomes :-



```
Queue: 10 20 30
Queue: 20 30
Queue is empty.

--------------------------------
Process exited after 0.05301 seconds with return value 0
Press any key to continue . . .
```

Program No:- 12

Aim/Title: WAP to construct a binary search tree and perform deletion and inorder traversal on it.

Date: 16/05/2023

Tool: Dev C++

Algorithm/Flow Chart:-

Algorithm:-

Start with an empty binary search tree (BST).
Create a function createNode to allocate memory for a new node and initialize its data and pointers.
Create a function insertNode to insert a node into the BST:
If the tree is empty, create a new node with the given data and return it as the root.
If the data is less than the root's data, recursively insert it into the left subtree.
If the data is greater than the root's data, recursively insert it into the right subtree.
Return the modified root.
Create a function inorderTraversal to perform inorder traversal of the BST:
If the current node is not null:
Recursively traverse the left subtree.
Print the data of the current node.
Recursively traverse the right subtree.
Create a function minValueNode to find the minimum value node in a given BST:
Start from the given node.
Traverse down the left subtree until reaching the leftmost node.
Return the leftmost node.
Create a function deleteNode to delete a node from the BST:
If the root is null, return null.
If the data to be deleted is less than the root's data, recursively delete it from the left subtree.
If the data to be deleted is greater than the root's data, recursively delete it from the right subtree.
If the data to be deleted is equal to the root's data:
If the node has no left child, replace it with its right child (if any) and free the node.
If the node has no right child, replace it with its left child and free the node.
If the node has both left and right children, find the minimum value node from the right subtree.
Copy the data of the minimum value node to the node to be deleted.
Recursively delete the minimum value node from the right subtree.
Return the modified root.
Create the main function:

Initialize the root of the BST as null.
Insert nodes into the BST using the insertNode function.
Perform inorder traversal before deletion using the inorderTraversal function.
Read the key to be deleted from the user.
Delete the node with the given key using the deleteNode function.
Perform inorder traversal after deletion using the inorderTraversal function.
Return 0 to exit the program.

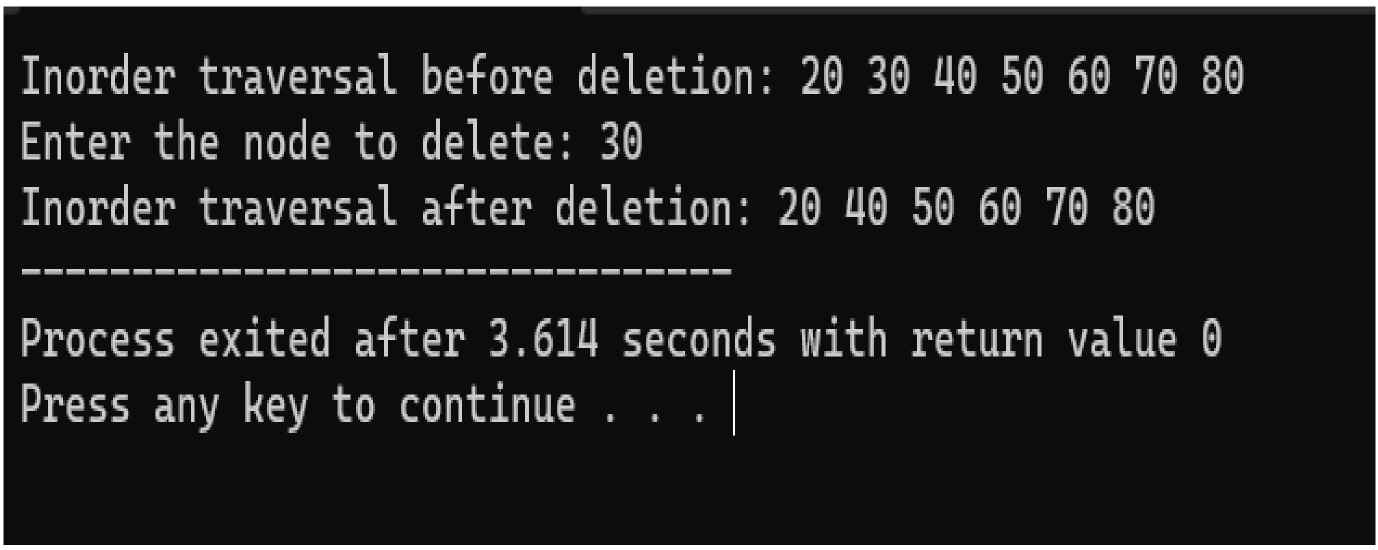
Source Code:-

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

// Structure for a node in the binary search tree
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node with the given data
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Function to insert a node into the binary search tree
struct Node* insertNode(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insertNode(root->left, data);
    }
    else if (data > root->data) {
        root->right = insertNode(root->right, data);
    }
    return root;
```

```
}

// Function to perform inorder traversal of the binary search tree
void inorderTraversal(struct Node* root) {
   if (root != NULL) {
      inorderTraversal(root->left);
      printf("%d ", root->data);
      inorderTraversal(root->right);
   }
}

// Function to find the minimum value node in a given binary search tree
struct Node* minValueNode(struct Node* node) {
   struct Node* current = node;
   while (current && current->left != NULL) {
      current = current->left;
   }
   return current;
}

// Function to delete a node from the binary search tree
struct Node* deleteNode(struct Node* root, int data) {
   if (root == NULL) {
      return root;
   }
   if (data < root->data) {
      root->left = deleteNode(root->left, data);
   }
   else if (data > root->data) {
      root->right = deleteNode(root->right, data);
   }
```

```
    else {
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }
        struct Node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

int main() {
    struct Node* root = NULL;
    root = insertNode(root, 50);
    insertNode(root, 30);
    insertNode(root, 20);
    insertNode(root, 40);
    insertNode(root, 70);
    insertNode(root, 60);
    insertNode(root, 80);

    printf("Inorder traversal before deletion: ");
    inorderTraversal(root);
    int key;
    printf("\nEnter the node to delete: ");
    scanf("%d", &key);
    root = deleteNode(root, key);
    printf("Inorder traversal after deletion: ");
    inorderTraversal(root);

    return 0;
}
```

Outcomes :-



```
Inorder traversal before deletion: 20 30 40 50 60 70 80
Enter the node to delete: 30
Inorder traversal after deletion: 20 40 50 60 70 80
--------------------------------
Process exited after 3.614 seconds with return value 0
Press any key to continue . . .
```

Program No:- 13

Aim/Title:- WAP to develop an algorithm for binary search, implement and test it.

Date: 16/05/2023

Tool: Dev C++

Algorithm/Flow Chart:-

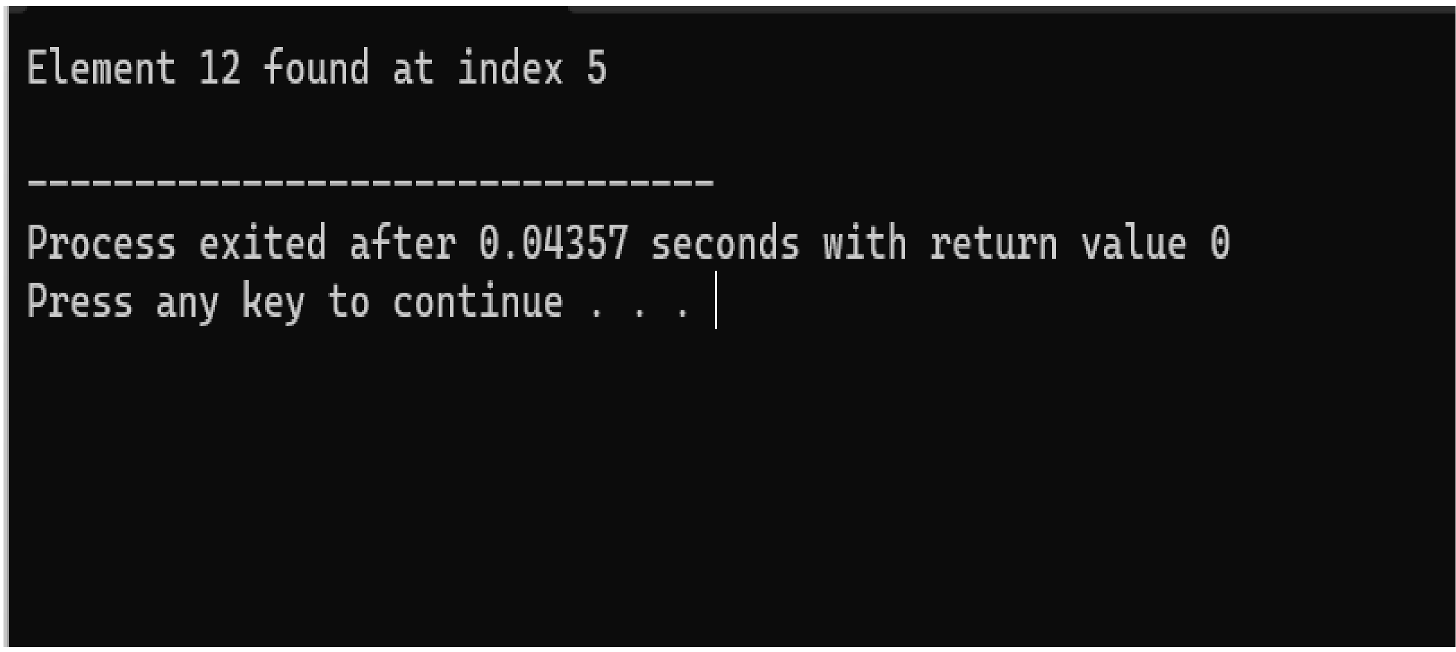
Algorithm:-

```
Steps:
    1. Set left = 0 and right = length of arr - 1.
    2. Repeat the following steps until left <= right:
        a. Compute mid as (left + right) / 2.
        b. If arr[mid] equals key, return mid.
        c. If arr[mid] is less than key, set left = mid + 1.
        d. If arr[mid] is greater than key, set right = mid - 1.
    3. If the key is not found after the loop, return -1.
```

Source Code:-

```
#include <stdio.h>

int binarySearch(int arr[], int left, int right, int key) {
    while (left <= right) {
        int mid = left + (right - left) / 2;

        // Check if the key is present at the middle
        if (arr[mid] == key)
            return mid;

        // If the key is greater, ignore the left half
        if (arr[mid] < key)
            left = mid + 1;

        // If the key is smaller, ignore the right half
        else
            right = mid - 1;
    }

    // Key not found
    return -1;
}

int main() {
    int arr[] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 12;

    int index = binarySearch(arr, 0, n - 1, key);
    if (index != -1)
        printf("Element %d found at index %d\n", key, index);
    else
        printf("Element %d not found in the array\n", key);

    return 0;
}
```

Outcomes :-



```
Element 12 found at index 5

--------------------------------
Process exited after 0.04357 seconds with return value 0
Press any key to continue . . .
```

Program No:- 14

Aim/Title:- WAP for implementation of Bubble Sort.

Date: 23/05/2023

Tool: Dev C++

Algorithm/Flow Chart:-

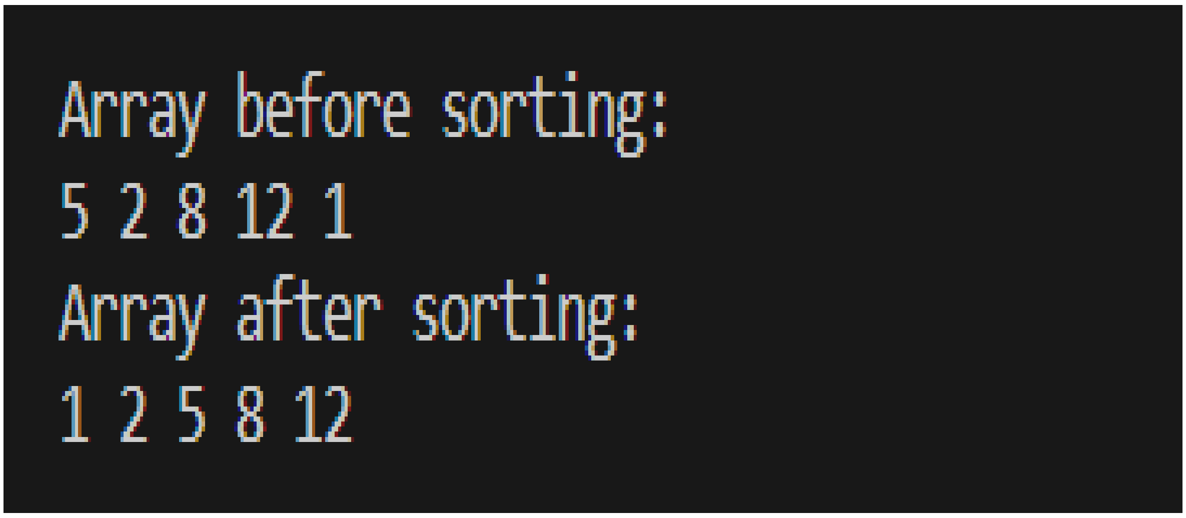
Algorithm:-

1) Start with an unsorted array of elements.
2) Compare the first element with the second element. If the first element is greater than the second element, swap them.
3) Move to the next pair of elements (i.e., the second and third elements) and compare them. Swap them if they are in the wrong order.
4) Continue this process until you reach the end of the array. At this point, the largest element will be in its correct position at the end of the array.
5) Repeat steps 2-4 for the remaining elements, excluding the last element that was already sorted in the previous pass.
6) Keep repeating steps 2-5 until the entire array is sorted.

Source Code:-

```
#include <stdio.h>

void bubbleSort(int arr[], int n) {
    int i, j, temp;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap arr[j] and arr[j+1]
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main() {
    int arr[] = {5, 2, 8, 12, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    int i;

    printf("Array before sorting:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    bubbleSort(arr, n);

    printf("\nArray after sorting:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

Outcomes :-



```
Array before sorting:
5 2 8 12 1
Array after sorting:
1 2 5 8 12
```

Program No:- 15

Aim/Title:- WAP for implementation of Insertion Sort.

Date: 20/06/2023
Tool: Dev C++

Algorithm/Flow Chart:-

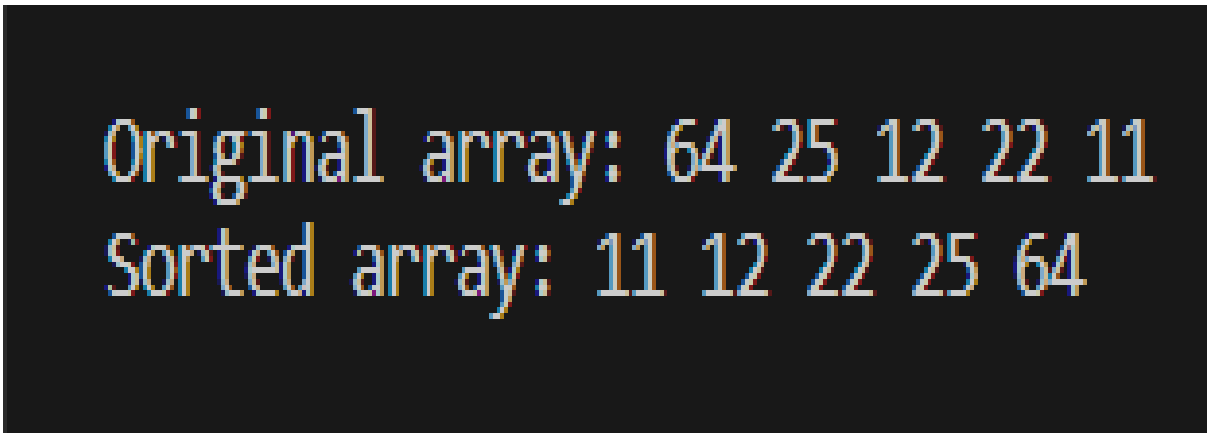
Algorithm:-

1) Start with an unsorted array of elements.
2) Iterate over the array starting from the second element (index 1) to the last element (index n-1), where n is the number of elements in the array.
3) For each element, compare it with the elements before it in the sorted subarray.
4) If the element is smaller than the previous element, shift the previous element one position to the right.
5) Repeat step 4 until you find the correct position for the current element in the sorted subarray.
6) Insert the current element into its correct position in the sorted subarray.
7) Repeat steps 2-6 until all elements in the array are sorted.

Source Code:-

```
#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[] = { 64, 25, 12, 22, 11 };
   int n = sizeof(arr) / sizeof(arr[0]);

   printf("Original array: ");
   printArray(arr, n);

   insertionSort(arr, n);

   printf("Sorted array: ");
   printArray(arr, n);

   return 0;
}
```

Outcomes :-



```
Original array: 64 25 12 22 11
Sorted array: 11 12 22 25 64
```