INSTITUTE OF TECHNOLOGY AND MANAGEMENT SKILLS UNIVERSITY, KHARGHAR, NAVI MUMBAI

DATA STRUCTURES & ALGORITHMS

PROGRAMMING LAB

DATA
STRUCTURES

Prepared by:

Name of Student :   Prabodh Badimi

Roll No: 13

Batch: 2023-27

Dept. of CSE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INSTITUTE OF TECHNOLOGY AND MANAGEMENT SKILLS UNIVERSITY, KHARGHAR, NAVI MUMBAI

CERTIFICATE

This is to certify that Mr._____Badimi Prabodh_____   Roll No. **13** Semester __2__ of
B.Tech Computer Science & Engineering, ITM Skills University, Kharghar, Navi
Mumbai , has completed the term work satisfactorily in subject DSA for the academic
year __2023 - 2027__ as prescribed in the curriculum.

Place: ___Kharghar___
Date: ____07/04/24___

Subject I/C HOD

Exp. No	List of Experiment	Date of Submission	Sign
1	Implement Array and write a menu driven program to perform all the operation on array elements		
2	Implement Stack ADT using array.		

3	Convert an Infix expression to Postfix expression using stack ADT.		
4	Evaluate Postfix Expression using Stack ADT.		
5	Implement Linear Queue ADT using array.		
6	Implement Circular Queue ADT using array.		
7	Implement Singly Linked List ADT.		
8	Implement Circular Linked List ADT.		
9	Implement Stack ADT using Linked List		
10	Implement Linear Queue ADT using Linked List		
11	Implement Binary Search Tree ADT using Linked List.		
12	Implement Graph Traversal techniques: a) Depth First Search b) Breadth First Search		
13	Implement Binary Search algorithm to search an element in an array		
14	Implement Bubble sort algorithm to sort elements of an array in ascending and descending order		

Name of Student: Prabodh Badimi

Roll Number: 13

Experiment No: 1

Title: Implement Array and write a menu driven program to perform all the operation on array elements

Theory: Array is a collection of elements of similar data types and has a fixed size. We can access an element of the array through it's index. Indexing starts from 0 till n-1(where n=size of array). An element can be inserted in the array by shifting all the elements of the array to the right and making space for the element. Similarly, to delete an element, we need to shift all the elements from the right of the deleted element to the left side in order to overwrite the deleted element. In order to search for an element, we need to traverse through the array and print the appropriate message if the element is found or not.

Code:

```cpp
// menu driven array operations program - 1.display array 2.insert at beginning 3.insert at end 4.insert before an element 5.insert after an element 6.delete at beginning 7.delete at end 8.delete before an element 9.delete after an element 10.search an element 11.number of elements
#include <iostream>
#include <algorithm>
using namespace std;
void displayArray(int &a, int arr[]) // function to display array and number of elements in array
{
int count = 0;
cout << "Array: ";
for (int i = 0; i < a; i++)
{
if (arr[i] != -1)
{
cout << arr[i] << " ";
```

```cpp
count++;
}
else
{
break;
}
}
cout << endl
<< "Number of elements: " << count << endl; }
void insertAtBegin(int &a, int arr[]) // function to insert element
at the beginning of the array
{
if (a >= 45)
{
cout << "Array is full. Cannot insert at the beginning." << endl;
return;
}
int b, count = 0;
cout << "Enter beginning detail: ";
cin >> b;
for (int i = a - 1; i >= 0; i--)
{
arr[i + 1] = arr[i];
}
arr[0] = b;
a++;
for (int i = 0; i < a; i++)
{
if (arr[i] == -1)
{
break;
```

```cpp
}
else
{
count++;
}
}
for (int i = 0; i < count; i++)
{
cout << arr[i] << " ";
}
}
void insertAtEnd(int &a, int arr[]) // function to insert element
at the end of the array
{
if (a >= 45)
{
cout << "Array is full. Cannot insert at the end." << endl;
return;
}
int b, count = 0;
cout << "Enter end detail: ";
cin >> b;
for (int i = 0; i < a; i++)
{
if (arr[i] == -1)
{
break;
}
else
 {
count++;
```

```cpp
}
}
arr[count] = b;
count++;
cout << "Size of array: " << count << endl;
a = count;
for (int i = 0; i < count; i++)
{
cout << arr[i] << " ";
}
}
void insertAtIndexLocation(int &c, int arr[]) // function to insert
element at specified index location in the array
{
int a, b;
cout << "Enter updated detail: ";
cin >> b;
cout << "Enter index location: ";
cin >> a;
int i = c - 1;
while (i >= a)
{
arr[i + 1] = arr[i];
i--;
}
c++;
arr[a] = b;
for (int i = 0; i < c; i++)
{
cout << arr[i] << " ";
}
```

```cpp
}
void insertBeforeElement(int &c, int arr[]) // function to insert
element before specified element in the array
{
int b, a, pos;
cout << "Enter updated detail: ";
cin >> b;
cout << "Enter element to insert before: ";
cin >> a;
for (int i = 0; i < c; i++)
{
if (arr[i] == a)
{
pos = i;
}
}
for (int i = c - 1; i >= pos; i--)
{
arr[i + 1] = arr[i];
 }
c++;
arr[pos] = b;
for (int i = 0; i < c; i++)
{
cout << arr[i] << " ";
}
}
void insertAfterElement(int &c, int arr[]) // function to insert
element after specified element in the array
{
int b, a, pos;
```

```cpp
cout << "Enter updated detail: ";
cin >> b;
cout << "Enter element to insert after: ";
cin >> a;
for (int i = 0; i < c; i++)
{
if (arr[i] == a)
{
pos = i;
}
}
for (int i = c - 1; i > pos; i--)
{
arr[i + 1] = arr[i];
}
c++;
arr[pos + 1] = b;
for (int i = 0; i < c; i++)
{
cout << arr[i] << " ";
}
}
void deleteFromBegin(int &c, int arr[]) // function to delete
element from beginning of the array
{
for (int i = 0; i < c; i++)
{
arr[i] = arr[i + 1];
}
c--;
for (int i = 0; i < c; i++)
```

```cpp
{
cout << arr[i] << " ";
}
}
void deleteFromEnd(int &c, int arr[]) // function to delete element
from end of the array
{
if (c <= 0)
 {
cout << "Array is empty. Cannot delete from the end." << endl;
return;
}
arr[c - 1] = arr[c];
c--;
for (int i = 0; i < c; i++)
{
cout << arr[i] << " ";
}
}
void deleteBeforeElement(int &c, int arr[]) // function to delete
element before specified element in the array
{
int b, pos;
cout << "Enter element to delete after it: ";
cin >> b;
for (int i = 0; i < c; i++)
{
if (arr[i] == b)
{
pos = i;
}
```

```cpp
}
for (int i = pos - 1; i < c; i++)
{
arr[i] = arr[i + 1];
}
c--;
for (int i = 0; i < c; i++)
{
cout << arr[i] << " ";
}
}
void deleteAfterElement(int &c, int arr[]) // function to delete
{
int b, pos;
cout << "Enter element to delete after it: ";
cin >> b;
for (int i = 0; i < c; i++)
{
if (arr[i] == b)
{
pos = i;
}
}
for (int i = pos + 1; i < c; i++)
{
arr[i] = arr[i + 1];
}
 c--;
for (int i = 0; i < c; i++)
{
cout << arr[i] << " ";
```

```cpp
}
}
void deleteFromArray(int &a, int arr[]) // function to delete
elements from array
{
int b, pos;
cout << "Enter element to delete: ";
cin >> b;
for (int i = 0; i < a; i++)
{
if (arr[i] == b)
{
pos = i;
}
}
for (int i = pos; i < a; i++)
{
arr[i] = arr[i + 1];
}
a--;
for (int i = 0; i < a; i++)
{
cout << arr[i] << " ";
}
}
void searchElement(int &a, int arr[]) // function to search an
element in the array
{
int b, count = 0;
cout << "Enter element to search: ";
cin >> b;
```

```cpp
for (int i = 0; i < a; i++)
{
if (arr[i] == b) {
cout << "Element found at index " << i << endl;  count++; }
}
if (count == 0) {
cout << "Element not found" << endl; }
}
int main() {
int arr[13], n, choice;
fill_n(arr, 13, -1);
cout << "Enter number of details you want to enter (less than  45): ";
cin >> n;
while (n >= 45 || n <= 0)
{
cout << "Invalid size. Enter a valid size" << endl;  cin >> n;
}
for (int i = 0; i < n; i++)
{
cout << "Enter detail: ";
cin >> arr[i];
}
char ans = 'y';
while (ans == 'y')
{
cout << "Enter your choice:\n1. Insert element at  beginning\n2.
Insert element at end\n3. Insert element at a  particular index
position\n4. Insert element before an element\n5.  Insert element
after an element\n6. Delete element from  beginning\n7. Delete
element from end\n8. Delete element before a  particular
```

```cpp
element\n9. Delete element after a particular  element\n10. Search
an element\n11. Delete element from array\n12.  Display array\n13.
Exit\n";
cin >> choice;
switch (choice)
{
case 1:
insertAtBegin(n, arr);
break;
case 2:
insertAtEnd(n, arr);
break;
case 3:
insertAtIndexLocation(n, arr);
break;
case 4:
insertBeforeElement(n, arr);
break;
case 5:
insertAfterElement(n, arr);
break;
case 6:
deleteFromBegin(n, arr);
break;
case 7:
deleteFromEnd(n, arr);
break;
case 8:
deleteBeforeElement(n, arr);
break;
case 9:
```

```cpp
deleteAfterElement(n, arr);

break;

case 10:

searchElement(n, arr);

break;

case 11:

deleteFromArray(n, arr);

break;

case 12:

displayArray(n, arr);

break;

case 13:

cout << "Exiting..." << endl;

return 0;

default:

cout << "Invalid choice\n";

}

cout << "Want to perform another operation? (y/n): ";  cin >> ans;

}

return 0;

}
```

Output: (screenshot)

```
dh/Desktop/DSALabManual/"1st
Enter number of details you want to enter (less than  45): 3
Enter detail: 1234567
Enter detail: 567899
Enter detail: 123456789
Enter your choice:
1. Insert element at  beginning
2. Insert element at end
3. Insert element at a  particular index position
4. Insert element before an element
5.  Insert element after an element
6. Delete element from  beginning
7. Delete element from end
8. Delete element before a  particular element
9. Delete element after a particular  element
10. Search an element
11. Delete element from array
12.  Display array
13. Exit
1
Enter beginning detail: 0
0 1234567 567899 123456789 Want to perform another operation? (y/n): y
Enter your choice:
1. Insert element at  beginning
2. Insert element at end
3. Insert element at a  particular index position
4. Insert element before an element
5.  Insert element after an element
6. Delete element from  beginning
7. Delete element from end
8. Delete element before a  particular element
9. Delete element after a particular  element
10. Search an element
11. Delete element from array
12.  Display array
13. Exit
2
Enter end detail: 5
Size of array: 5
0 1234567 567899 123456789 5 Want to perform another operation? (y/n): y
Enter your choice:
1. Insert element at  beginning
2. Insert element at end
3. Insert element at a  particular index position
```

```
dh/Desktop/DSALabManual/"1st
Enter number of details you want to enter (less than  45): 2
Enter detail: 23456
Enter detail: 23456
Enter your choice:
1. Insert element at  beginning
2. Insert element at end
3. Insert element at a  particular index position
4. Insert element before an element
5.  Insert element after an element
6. Delete element from  beginning
7. Delete element from end
8. Delete element before a  particular element
9. Delete element after a particular  element
10. Search an element
11. Delete element from array
12.  Display array
13. Exit
6
23456 Want to perform another operation? (y/n): y
Enter your choice:
1. Insert element at  beginning
2. Insert element at end
3. Insert element at a  particular index position
4. Insert element before an element
5.  Insert element after an element
6. Delete element from  beginning
7. Delete element from end
8. Delete element before a  particular element
9. Delete element after a particular  element
10. Search an element
11. Delete element from array
12.  Display array
13. Exit
7
```

Conclusion: Therefore, using switch cases, we can perform multiple operations like insertion, deletion, and searching for an element in an array through traversal using index.

Name of Student: Prabodh Badimi

Roll Number: 13

Experiment No: 2

Title: Implement Stack ADT using Array.

Theory: Array is a collection of elements of similar data types and has a fixed size. We can access an element of the array through it's index. Indexing starts from 0 till n-1(where n=size of array).

Stack is an Abstract Data Type which can be implemented using Linked List or Array. It consists of a variable named Top which points to the topmost element of the stack. Stack follows LIFO principle(Last In, First Out) which means that the element which is inserted last will be deleted first. There are three operations in Stack: Push- insertion from top, Pop- deletion from top, Peek- returning the topmost element from the stack.

Code:

```cpp
// stack operations(array)
#include <iostream>
using namespace std;

int main()
{
int top = -1, element, op,n;
cout<<"Enter size of stack: ";
cin>>n;
int stack[n];
while (true)
{
cout << "\nStack operation:
\n1.Push\n2.Pop\n3.Peek\n4.Exit\n";
```

```cpp
cin >> op;
switch (op)
{
case 1:
if (top == n - 1)
{
cout << "Stack is full. Cannot add more elements. \n";
break;
}
else
{
cout << "Enter element: ";
cin >> element;
top++;
stack[top] = element;
cout << "Element added in stack\n"; }
break;
case 2:
if (top == -1)
{
cout << "Stack is empty.\n";
break;
}
else
{
cout << stack[top] << " is popped from stack\n"; top--;
}
break;
case 3:
if (top == -1)
{
cout << "Stack is empty.\n";
break;
}
else
{
cout << "Top element: " << stack[top] << "\n"; }
break;
case 4:
cout << "Exiting...\n";
return 0;
default:
cout << "Wrong choice\n";
break;
}
}
return 0;
}
```

Output: (screenshot)

Conclusion: Therefore, using switch cases, we can perform multiple operations like push, pop, and peek in a stack using

Name of Student: Prabodh Badimi

Roll Number: 13

Experiment No: 3

Title: Convert an Infix expression to Postfix expression using Stack ADT.

Theory: Stack is an Abstract Data Type which can be implemented using Linked List or Array. It consists of a variable named Top which points to the topmost element of the stack. Stack follows LIFO principle(Last In, First Out) which means that the element which is inserted last will be deleted first. There are three operations in Stack: Push- insertion from top, Pop- deletion from top, Peek returning the topmost element from the stack. Using stack, we can convert an infix expression to postfix expression by pushing the operators and brackets in the stack and the operands to the expression and popping the elements to the expression through operator precedence after encountering a closing bracket.

Code:

```cpp
// conversion of infix to postfix expression using stack(array)

#include <iostream>
using namespace std;

int precedence(char op)
{
if (op == '+' || op == '-')
{
return 1;
}
else if (op == '*' || op == '/' || op == '%')
{
return 2;
}
```

```cpp
    else
    {
        return 0;
    }
}

int main()
{
    string exp, result = "";
    char stack[100];
    int top = -1;
    cout << "Enter infix expression: ";
    getline(cin, exp);
    int n = exp.length();
    char express[n + 2];
    express[0] = '(';
    for (int i = 0; i < n; i++)
    {
        express[i + 1] = exp[i];
    }
    express[n + 1] = ')';
    for (int i = 0; i < n + 2; i++)
    {
        if (express[i] == '(')
        {
            top++;
            stack[top] = express[i];
        }
        else if (express[i] == ')')
        {
            while (stack[top] != '(' && top > -1)
            {
                result += stack[top];
                top--;
            }
            top--;
        }
        else if ((express[i] >= 'a' && express[i] <= 'z') || (express[i] >= 'A' && express[i] <= 'Z') ||
        (express[i] >= '0' && express[i] <= '9'))
        {
            result += express[i];
        }
        else
        {
            while (top > -1 && precedence(stack[top]) >= precedence(express[i]))
            {
                result += stack[top];
                top--;
            }
            top++;
            stack[top] = express[i];
        }
    }
}
```

```
while (top > -1)
{
result += stack[top];
top--;
}
cout << "Postflix Result: " << result << endl;
return 0;
}
```

Output: (screenshot)

Conclusion: Therefore, using stack ADT, we can convert infix expression to postfix expression by operations like Push and Pop.

Name of Student: Prabodh Badimi

Roll Number: 13

Experiment No: 4

Title: Evaluate Postfix expression using Stack ADT.

Theory: Stack is an Abstract Data Type which can be implemented using Linked List or Array. It consists of a variable named Top which points to the topmost element of the stack. Stack follows LIFO principle(Last In, First Out) which means that the element which is inserted last will be deleted first. There are three operations in Stack: Push- insertion from top, Pop- deletion from top, Peek returning the topmost element from the stack. Using stack, we can evaluate a postfix expression by pushing the operands in the stack and popping them and evaluating them when an operator is encountered and popping the result back in the stack and printing the topmost element after the whole expression is evaluated.

Code:

```cpp
// evaluating postfix expression using stack(array)

#include <iostream>

using namespace std;

// Stack implementation

class Stack {

private:

    int top;

    int *arr;

    int capacity;

public:

    Stack(int size) {
```

```cpp
        capacity = size;

        arr = new int[size];

        top = -1;

    }

    ~Stack() {

        delete[] arr;

    }

    void push(int data) {

        if (isFull()) {

            cout << "Stack Overflow!" << endl;

            return;

        }

        arr[++top] = data;

    }

    int pop() {

        if (isEmpty()) {

            cout << "Stack Underflow!" <<
endl;

            return -1;

        }

        return arr[top--];

    }

    int peek() {
```

```cpp
        if (isEmpty()) {

            cout << "Stack is empty!" << endl;

            return -1;

        }

        return arr[top];

    }

    bool isEmpty() {

        return top == -1;

    }

    bool isFull() {

        return top == capacity - 1;

    }
};

int evaluatePostfix(char* expression) {

    Stack operandStack(100);

    for (int i = 0; expression[i]; ++i) {

        if (isdigit(expression[i])) {

            operandStack.push(expression[i] -
'0');

        } else if (expression[i] == '+' ||
expression[i] == '-' || expression[i] == '*'
|| expression[i] == '/' || expression[i] ==
'^') {

            int operand2 = operandStack.pop();
```

```cpp
            int operand1 = operandStack.pop();

            switch(expression[i]) {

                case '+':

                    operandStack.push(operand1
+ operand2);

                    break;

                case '-':

                    operandStack.push(operand1
- operand2);

                    break;

                case '*':

                    operandStack.push(operand1
* operand2);

                    break;

                case '/':

                    operandStack.push(operand1
/ operand2);

                    break;

                case '^':


operandStack.push(pow(operand1, operand2));

                    break;

                default:

                    cout << "Invalid operator"
```

```cpp
        << endl;

                        exit(1);

                }

        }

    }

    return operandStack.peek();

}

int main() {

    char postfixExpression[100];

    cout << "Enter postfix expression: ";

    cin.getline(postfixExpression, 100);

    int result =
evaluatePostfix(postfixExpression);

    cout << "Result: " << result << endl;

    return 0;

}
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion: Therefore, using stack ADT, we can evaluate a postfix expression by operations like Push and Pop.

Name of Student: Prabodh Badimi

Roll Number: 13

Experiment No: 5

Title: Implement Linear Queue ADT using array.

Theory: Array is a collection of elements of similar data types and has a fixed size. We can access an element of the array through it's index. Indexing starts from 0 till n-1(where n=size of array).

Queue is an Abstract Data Type which can be implemented using Linked List or Array. It consists of two variables named Front and Rear which point to the first and last elements of the stack, respectively. Queue follows FIFO principle(First In, First Out) which means that the element which is inserted first will be deleted first. There are three operations in Stack: Enqueue- insertion from rear, Dequeue- deletion from front, Peek- returning the frontmost element from the queue.

Code:

```cpp
// queue menu driven program(array)
#include <iostream>
using namespace std;
const int MAX_SIZE = 100;
class Queue {
private:
    int front, rear;
    int arr[MAX_SIZE];
public:
    Queue() {
        front = -1;
        rear = -1;
```

```cpp
    }
    bool isEmpty() {
        return front == -1 && rear == -1;
    }
    bool isFull() {
        return rear == MAX_SIZE - 1;
    }
    void enqueue(int data) {
        if (isFull()) {
            cout << "Queue is full. Cannot enqueue." << endl;
            return;
        }
        if (isEmpty()) {
            front = 0;
        }
        rear++;
        arr[rear] = data;
        cout << data << " enqueued to queue." << endl;
    }
    void dequeue() {
        if (isEmpty()) {
            cout << "Queue is empty. Cannot dequeue." << endl;
            return;
        }
        cout << arr[front] << " dequeued from queue." << endl;
        if (front == rear) {
            front = rear = -1;
        } else {
            front++;
        }
    }
    int peek() {
        if (isEmpty()) {
            cout << "Queue is empty. Cannot peek." << endl;
            return -1;
        }
        return arr[front];
    }
    void display() {
        if (isEmpty()) {
            cout << "Queue is empty." << endl;
            return;
        }
        cout << "Queue elements: ";
        for (int i = front; i <= rear; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};
int main() {
    Queue q;
    int choice, data;
```

```cpp
    do {
        cout << "\n1. Enqueue\n2. Dequeue\n3. Peek\n4. Display\n5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "Enter data to enqueue: ";
                cin >> data;
                q.enqueue(data);
                break;
            case 2:
                q.dequeue();
                break;
            case 3:
                cout << "Front element: " << q.peek() << endl;
                break;
            case 4:
                q.display();
                break;
            case 5:
                cout << "Exiting program." << endl;
                break;
            default:
                cout << "Invalid choice. Please enter a valid option." << endl;
                break;
        }
    } while (choice != 5);
    return 0;
}
```

Output: (screenshot)

```
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter data to enqueue: 23
23 enqueued to queue.

1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 2
23 dequeued from queue.

1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 3
Front element: 45
```

Conclusion: Therefore, using array, we can implement a linear queue and perform operations like Enqueue, Dequeue and Peek.

Name of Student: Prabodh Badimi

Roll Number: 13

Experiment No: 6

Title: Implement Circular Queue ADT using array.

Theory: Array is a collection of elements of similar data types and has a fixed size. We can access an element of the array through it's index. Indexing starts from 0 till n-1(where n=size of array).

Queue is an Abstract Data Type which can be implemented using Linked List or Array. It consists of two variables named Front and Rear which point to the first and last elements of the stack, respectively. Queue follows FIFO principle(First In, First Out) which means that the element which is inserted first will be deleted first. There are three operations in Stack: Enqueue- insertion from rear, Dequeue- deletion from front, Peek- returning the frontmost element from the queue. As size of array is fixed, in order to overcome the challenges, we can move the rear pointer to the start of the array if rear=n-1 and front is not at first index, so we can continue to insert elements.

Code:

```cpp
// circular queue menu driven operations(array)
#include <iostream>
using namespace std;
const int MAX_SIZE = 100;
class CircularQueue {
private:
    int front, rear;
    int arr[MAX_SIZE];
public:
    CircularQueue() {
        front = -1;
        rear = -1;
    }
    bool isEmpty() {
```

```cpp
        return front == -1;
    }
    bool isFull() {
        return (front == 0 && rear == MAX_SIZE - 1) || (rear ==
front - 1);
    }
    void enqueue(int data) {
        if (isFull()) {
            cout << "Queue is full. Cannot enqueue." << endl;
            return;
        }
        if (isEmpty()) {
            front = 0;
        }
        rear = (rear + 1) % MAX_SIZE;
        arr[rear] = data;
        cout << data << " enqueued to queue." << endl;
    }
    void dequeue() {
        if (isEmpty()) {
            cout << "Queue is empty. Cannot dequeue." << endl;
            return;
        }
        int data = arr[front];
        if (front == rear) {
            front = rear = -1;
        } else {
            front = (front + 1) % MAX_SIZE;
        }
        cout << data << " dequeued from queue." << endl;
    }
```

```cpp
    int peek() {
        if (isEmpty()) {
            cout << "Queue is empty. Cannot peek." << endl;
            return -1;
        }
        return arr[front];
    }
    void display() {
        if (isEmpty()) {
            cout << "Queue is empty." << endl;
            return;
        }
        cout << "Queue elements: ";
        int i = front;
        while (i != rear) {
            cout << arr[i] << " ";
            i = (i + 1) % MAX_SIZE;
        }
        cout << arr[rear] << endl;
    }
};
int main() {
    CircularQueue q;
    int choice, data;
    do {
        cout << "\n1. Enqueue\n2. Dequeue\n3. Peek\n4. Display\n5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
```

```cpp
                cout << "Enter data to enqueue: ";

                cin >> data;

                q.enqueue(data);

                break;

            case 2:

                q.dequeue();

                break;

            case 3:

                cout << "Front element: " << q.peek() << endl;

                break;

            case 4:

                q.display();

                break;

            case 5:

                cout << "Exiting program." << endl;

                break;

            default:

                cout << "Invalid choice. Please enter a valid
option." << endl;

                break;

        }

    } while (choice != 5);

    return 0;

}
```

Output: (screenshot)

```
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter data to enqueue: 234
234 enqueued to queue.

1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter data to enqueue: 7654
7654 enqueued to queue.

1. Enqueue
```

Conclusion: Therefore, using array, we can implement a circular queue and perform operations like Enqueue, Dequeue and Peek without being constrained by the limitation of the fixed size of the array.

Name of Student: Prabodh Badimi

Roll Number: 13

Experiment No: 7

Title: Implement Singly Linked List ADT.

Theory: Linked List is a data type which consists of nodes which contain data and a next pointer which points to the next node in the list. It stores the address of the next node. There is a start pointer in stack memory which points to the first node in the heap memory. It utilises dynamic memory and allocates heap memory to the nodes in the list. The last node's next pointer has NULL value to indicate it's the last node in the list.

Code:

```cpp
// menu driven linked list
#include <iostream>
using namespace std;
class Node {
public:
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }
};
class SinglyLinkedList {
private:
    Node* head;
public:
    SinglyLinkedList() {
        head = nullptr;
```

```cpp
    }
    void insertAtBeginning(int value) {
        Node* newNode = new Node(value);
        newNode->next = head;
        head = newNode;
    }
    void insertAtEnd(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    void display() {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
    ~SinglyLinkedList() {
        Node* temp = head;
        while (head != nullptr) {
            head = head->next;
            delete temp;
```

```cpp
            temp = head;
        }
    }
};
int main() {
    SinglyLinkedList list;
    int choice, value;
    do {
        cout << "\n1. Insert at Beginning\n2. Insert at End\n3.
Display\n4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "Enter value to insert at the beginning: ";
                cin >> value;
                list.insertAtBeginning(value);
                break;
            case 2:
                cout << "Enter value to insert at the end: ";
                cin >> value;
                list.insertAtEnd(value);
                break;
            case 3:
                cout << "Linked List elements: ";
                list.display();
                break;
            case 4:
                cout << "Exiting program." << endl;
                break;
            default:
```

```
                cout << "Invalid choice. Please enter a valid
option." << endl;

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

    return 0;
}
```

Output: (screenshot)

```
1. Insert at Beginning
2. Insert at End
3. Display
4. Exit
Enter your choice: 1
Enter value to insert at the beginning: 1234

1. Insert at Beginning
2. Insert at End
3. Display
4. Exit
Enter your choice: 1
Enter value to insert at the beginning: 2345

1. Insert at Beginning
2. Insert at End
3. Display
4. Exit
Enter your choice: 1
Enter value to insert at the beginning: 3456

1. Insert at Beginning
2. Insert at End
3. Display
4. Exit
Enter your choice: 3
Linked List elements: 3456 2345 1234

1. Insert at Beginning
2. Insert at End
3. Display
4. Exit
Enter your choice: 2
Enter value to insert at the end: 345
```

Test Case: Any two (screenshot)

```
1. Insert at Beginning
2. Insert at End
3. Display
4. Exit
Enter your choice: 1
Enter value to insert at the beginning: 1234

1. Insert at Beginning
2. Insert at End
3. Display
4. Exit
Enter your choice: 1
Enter value to insert at the beginning: 2345

1. Insert at Beginning
2. Insert at End
3. Display
4. Exit
Enter your choice: 1
Enter value to insert at the beginning: 3456

1. Insert at Beginning
2. Insert at End
3. Display
4. Exit
Enter your choice: 3
Linked List elements: 3456 2345 1234

1. Insert at Beginning
2. Insert at End
3. Display
4. Exit
Enter your choice: 2
Enter value to insert at the end: 345
```

Conclusion: Therefore, we can implement a linked list by using class or structure and allocate heap memory for the node by using new operator or malloc function. We can deallocate memory for the node by using free function or delete operator.

Name of Student: Prabodh Badimi

Roll Number: 13

Experiment No: 8

Title: Implement Circular Linked List ADT.

Theory: Linked List is a data type which consists of nodes which contain data and a next pointer which points to the next node in the list. It stores the address of the next node. There is a start pointer in stack memory which points to the first node in the heap memory. It utilises dynamic memory and allocates heap memory to the nodes in the list. The last node's next pointer has the address of first node, hence it's called circular linked list.

Code:

```cpp
// circular linked list menu
#include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node* next;

    Node(int value) {

        data = value;

        next = nullptr;

    }

};

class CircularLinkedList {

private:
```

```cpp
    Node* head;
public:
    CircularLinkedList() {
        head = nullptr;
    }

    void insertAtBeginning(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            newNode->next = newNode; // Point to itself
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
        head = newNode;
    }

    void insertAtEnd(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
```

```cpp
        newNode->next = newNode; // Point to itself

        head = newNode;

        return;

    }

    Node* temp = head;

    while (temp->next != head) {

        temp = temp->next;

    }

    temp->next = newNode;

    newNode->next = head;

}

void display() {

    if (head == nullptr) {

        cout << "Circular Linked List is empty" << endl;

        return;

    }

    Node* temp = head;

    do {

        cout << temp->data << " ";

        temp = temp->next;

    } while (temp != head);

    cout << endl;

}
```

```cpp
    ~CircularLinkedList() {

        if (head == nullptr) return;

        Node* temp = head;

        while (temp->next != head) {

            Node* toDelete = temp;

            temp = temp->next;

            delete toDelete;

        }

        delete temp;

    }

};

int main() {

    CircularLinkedList list;

    int choice, value;

    do {

        cout << "\n1. Insert at Beginning\n2. Insert at End\n3.
Display\n4. Exit\n";

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter value to insert at the beginning: ";

                cin >> value;
```

```cpp
                    list.insertAtBeginning(value);

                    break;

            case 2:

                    cout << "Enter value to insert at the end: ";

                    cin >> value;

                    list.insertAtEnd(value);

                    break;

            case 3:

                    cout << "Circular Linked List elements: ";

                    list.display();

                    break;

            case 4:

                    cout << "Exiting program." << endl;

                    break;

            default:

                    cout << "Invalid choice. Please enter a valid
option." << endl;

                    break;

        }

    } while (choice != 4);

    return 0;

}
```

Output: (screenshot)

```
1. Insert at Beginning
2. Insert at End
3. Display
4. Exit
Enter your choice: 1
Enter value to insert at the beginning: 234

1. Insert at Beginning
2. Insert at End
3. Display
4. Exit
Enter your choice: 5678
Invalid choice. Please enter a valid option.

1. Insert at Beginning
2. Insert at End
3. Display
4. Exit
Enter your choice: 4567
Invalid choice. Please enter a valid option.

1. Insert at Beginning
2. Insert at End
3. Display
4. Exit
Enter your choice: 345
Invalid choice. Please enter a valid option.

1. Insert at Beginning
2. Insert at End
3. Display
4. Exit
Enter your choice: 2
Enter value to insert at the end: 34566
```

Conclusion: Therefore, we can implement a circular linked list by using class or structure and allocate heap memory for the node by using new operator or malloc function. We can deallocate memory for the node by using free function or delete operator.

Name of Student: Prabodh Badimi

Roll Number: 13

Experiment No: 9

Title: Implement Stack ADT using Linked List.

Theory: Stack is an Abstract Data Type which can be implemented using Linked List or Array. It consists of a variable named Top which points to the topmost element of the stack. Stack follows LIFO principle(Last In, First Out) which means that the element which is inserted last will be deleted first. There are three operations in Stack: Push- insertion from top, Pop- deletion from top, Peek returning the topmost element from the stack. We can implement insertion at beginning, deletion from beginning algorithms to implement Stack using Linked List.

Linked List is a data type which consists of nodes which contain data and a next pointer which points to the next node in the list. It stores the address of the next node. There is a start pointer in stack memory which points to the first node in the heap memory. It utilises dynamic memory and allocates heap memory to the nodes in the list. The last node's next pointer has the address of first node, hence it's called circular linked list.

Code:

```cpp
// stack operations(linked list)

#include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node* next;

    Node(int value) {

        data = value;
```

```cpp
        next = nullptr;

    }

};

class Stack {

private:

    Node* top;

public:

    Stack() {

        top = nullptr;

    }

    void push(int value) {

        Node* newNode = new Node(value);

        newNode->next = top;

        top = newNode;

    }

    int pop() {

        if (isEmpty()) {

            cout << "Stack is empty. Cannot pop." << endl;

            return -1; // Return some invalid value

        }

        int poppedValue = top->data;

        Node* temp = top;

        top = top->next;
```

```cpp
        delete temp;

        return poppedValue;

    }

    int peek() {

        if (isEmpty()) {

            cout << "Stack is empty. Cannot peek." << endl;

            return -1; // Return some invalid value

        }

        return top->data;

    }

    bool isEmpty() {

        return top == nullptr;

    }

    ~Stack() {

        while (!isEmpty()) {

            pop();

        }

    }

};

int main() {

    Stack stack;

    int choice, value;

    do {
```

```cpp
        cout << "\n1. Push\n2. Pop\n3. Peek\n4. Exit\n";

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter value to push: ";

                cin >> value;

                stack.push(value);

                break;

            case 2:

                cout << "Popped value: " << stack.pop() << endl;

                break;

            case 3:

                cout << "Top of stack value: " << stack.peek() <<
endl;

                break;

            case 4:

                cout << "Exiting program." << endl;

                break;

            default:

                cout << "Invalid choice. Please enter a valid
option." << endl;

                break;
```

```
        }

    } while (choice != 4);

    return 0;

}
```

Output: (screenshot)

```
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 1
Enter value to push: 23

1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 1
Enter value to push: 2

1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 1
Enter value to push: 23

1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 1
Enter value to push: 567

1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 4
Exiting program.
```

Test Case: Any two (screenshot)

```
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 2
Popped value: 23

1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 2
Popped value: 234

1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 2
Popped value: 23

1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 2
Popped value: Stack is empty. Cannot pop.
-1
```

Conclusion: Therefore, we can implement Stack by linked list by using class or structure and allocate heap memory for the node by using new operator or malloc function. We can deallocate memory for the node by using free function or delete operator. We can implement push and pop operations through insertion at beginning and deletion from beginning algorithms.

Name of Student: Prabodh Badimi

Roll Number: 13

Experiment No: 10

Title: Implement Linear Queue ADT using Linked List.

Theory: Queue is an Abstract Data Type which can be implemented using Linked List or Array. It consists of two variables named Front and Rear which point to the first and last elements of the stack, respectively. Queue follows FIFO principle(First In, First Out) which means that the element which is inserted first will be deleted first. There are three operations in Stack: Enqueue- insertion from rear, Dequeue- deletion from front, Peek- returning the frontmost element from the queue. It can be implemented by insertion at end and deletion from beginning algorithms.

Linked List is a data type which consists of nodes which contain data and a next pointer which points to the next node in the list. It stores the address of the next node. There is a start pointer in stack memory which points to the first node in the heap memory. It utilises dynamic memory and allocates heap memory to the nodes in the list. The last node's next pointer has the address of first node, hence it's called circular linked list.

Code:

```
// queue menu driven program(linked list)
```

```cpp
#include <iostream>
using namespace std;
class Node {
public:
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }
};
```

```cpp
class Queue {
private:
    Node* front;
    Node* rear;
public:
    Queue() {
        front = nullptr;
        rear = nullptr;
    }
    void enqueue(int value) {
        Node* newNode = new Node(value);
        if (isEmpty()) {
            front = rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
        cout << value << " enqueued to queue." << endl;
    }
    int dequeue() {
        if (isEmpty()) {
            cout << "Queue is empty. Cannot dequeue." << endl;
            return -1; // Return some invalid value
        }
        int dequeuedValue = front->data;
        Node* temp = front;
        if (front == rear) {
            front = rear = nullptr;
        } else {
            front = front->next;
        }
    }
```

```cpp
            delete temp;
            return dequeuedValue;
        }
    int peek() {
        if (isEmpty()) {
            cout << "Queue is empty. Cannot peek." << endl;
            return -1; // Return some invalid value
        }
        return front->data;
    }
    bool isEmpty() {
        return front == nullptr;
    }
    ~Queue() {
        while (!isEmpty()) {
            dequeue();
        }
    }
};
int main() {
    Queue queue;
    int choice, value;
    do {
        cout << "\n1. Enqueue\n2. Dequeue\n3. Peek\n4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "Enter value to enqueue: ";
                cin >> value;
                queue.enqueue(value);
```

```
                break;
            case 2:
                cout << "Dequeued value: " << queue.dequeue() <<
endl;
                break;
            case 3:
                cout << "Front of queue value: " << queue.peek() <<
endl;
                break;
            case 4:
                cout << "Exiting program." << endl;
                break;
            default:
                cout << "Invalid choice. Please enter a valid
option." << endl;
                break;
        }
    } while (choice != 4);
    return 0;
}
```

Output: (screenshot)

```
1. Enqueue
2. Dequeue
3. Peek
4. Exit
Enter your choice: 1
Enter value to enqueue: 334
334 enqueued to queue.

1. Enqueue
2. Dequeue
3. Peek
4. Exit
Enter your choice: 1
Enter value to enqueue: 3456
3456 enqueued to queue.

1. Enqueue
2. Dequeue
3. Peek
4. Exit
Enter your choice: 4
Exiting program.
```

Conclusion: Therefore, we can implement Linear Queue by linked list by using class or structure and allocate heap memory for the node by using new operator or malloc function. We can deallocate memory for the node by using free function or delete operator. We can implement enqueue and dequeue operations through insertion at end and deletion from beginning algorithms.

Name of Student: Prabodh Badimi

Roll Number: 13

Experiment No: 11

Title: Implement Binary Search Tree ADT using Linked List.

Theory:

A binary tree is a non-linear data structure in which there is a root node and each parent node has 0,1 or 2 child nodes at most. In binary search tree, all the nodes having values less than that of the root node are present in the left subtree of the root node and all the nodes having values greater than or equal to that of the root node are present in the right subtree of the root node.

Code:

```cpp
// Binary Search Tree using Linked List

#include <iostream>

using namespace std;

class TreeNode {

public:

    int data;

    TreeNode* left;

    TreeNode* right;

    TreeNode(int value) {
```

```cpp
        data = value;

        left = nullptr;

        right = nullptr;

    }

};

class BinarySearchTree {

private:

    TreeNode* root;

    TreeNode* insertRecursively(TreeNode* root, int value) {

        if (root == nullptr) {

            return new TreeNode(value);

        }

        if (value < root->data) {

            root->left = insertRecursively(root->left, value);

        } else if (value > root->data) {

            root->right = insertRecursively(root->right, value);

        }

        return root;

    }

    void inorderTraversal(TreeNode* root) {

        if (root != nullptr) {

            inorderTraversal(root->left);

            cout << root->data << " ";
```

```cpp
            inorderTraversal(root->right);

        }

    }
public:
    BinarySearchTree() {

        root = nullptr;

    }

    void insert(int value) {

        root = insertRecursively(root, value);

    }

    void display() {

        inorderTraversal(root);

        cout << endl;

    }

    ~BinarySearchTree() {

        // Perform post-order traversal and delete nodes

        deleteTree(root);

    }

    void deleteTree(TreeNode* root) {

        if (root != nullptr) {

            deleteTree(root->left);

            deleteTree(root->right);

            delete root;
```

```cpp
        }
    }
};
int main() {
    BinarySearchTree bst;
    int choice, value;
    do {
        cout << "\n1. Insert\n2. Display\n3. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "Enter value to insert: ";
                cin >> value;
                bst.insert(value);
                break;
            case 2:
                cout << "Binary Search Tree elements (inorder): ";
                bst.display();
                break;
            case 3:
                cout << "Exiting program." << endl;
                break;
```

```
        default:

            cout << "Invalid choice. Please enter a valid
option." << endl;

            break;

    }

} while (choice != 3);

return 0;

}
```

Output: (screenshot)

```
1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter value to insert: 23

1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter value to insert: 23

1. Insert
2. Display
3. Exit
Enter your choice: 156
Invalid choice. Please enter a valid option.

1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter value to insert: 56

1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter value to insert: 478

1. Insert
2. Display
3. Exit
Enter your choice: 3
Exiting program.
```

```
1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter value to insert: 567

1. Insert
2. Display
3. Exit
Enter your choice: 2
Binary Search Tree elements (inorder): 23 345 567

1. Insert
2. Display
3. Exit
Enter your choice: 2
Binary Search Tree elements (inorder): 23 345 567

1. Insert
2. Display
3. Exit
Enter your choice: 2
Binary Search Tree elements (inorder): 23 345 567

1. Insert
2. Display
3. Exit
Enter your choice: 3
Exiting program.
```

Conclusion: Therefore, we can implement Binary Search Tree ADT using Linked List.

Name of Student: Prabodh Badimi

Roll Number: 13

Experiment No: 12

Title: Implement Graph Traversal techniques: a) Depth First Search b) Breadth First Search

Theory: A Graph is a non-linear data structure which can have parent-child as well as other complex relationships between the nodes. It is a set of edges and vertices, where vertices are the nodes, and the edges are the links connecting the nodes. We can implement a graph using adjacency matrix or adjacency list.

Code:

```cpp
// Graph Traversal techniques: a) Depth First Search b) Breadth First Search

#include <iostream>

#include <list>

using namespace std;

class Graph {

private:

    int vertices;

    list<int>* adjList;

    void DFSUtil(int v, bool visited[]) {

        visited[v] = true;

        cout << v << " ";

        for (auto i = adjList[v].begin(); i !=
adjList[v].end(); ++i) {
```

```cpp
            if (!visited[*i]) {

                DFSUtil(*i, visited);

            }

        }

    }

public:

    Graph(int V) : vertices(V) {

        adjList = new list<int>[V];

    }

    void addEdge(int u, int v) {

        adjList[u].push_back(v);

    }

    void DFS(int start) {

        bool* visited = new bool[vertices];

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

            visited[i] = false;

        }

        DFSUtil(start, visited);

    }

    void BFS(int start) {
```

```cpp
        bool* visited = new bool[vertices];

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

            visited[i] = false;

        }

        visited[start] = true;

        cout << start << " ";

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

            if (!visited[i]) {

                visited[i] = true;

                cout << i << " ";

            }

        }

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

            for (auto j = adjList[i].begin(); j !=
adjList[i].end(); ++j) {

                if (!visited[*j]) {

                    visited[*j] = true;

                    cout << *j << " ";

                }

            }
```

```cpp
        }

    }

    ~Graph() {

        delete[] adjList;

    }

};

int main() {

    int vertices, edges, start;

    cout << "Enter number of vertices: ";

    cin >> vertices;

    cout << "Enter number of edges: ";

    cin >> edges;

    Graph graph(vertices);

    cout << "Enter edges (u v):" << endl;

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

        int u, v;

        cin >> u >> v;

        graph.addEdge(u, v);

    }

    cout << "Enter starting vertex for traversal: ";
```

```cpp
    cin >> start;

    cout << "Depth First Search (DFS): ";

    graph.DFS(start);

    cout << endl;

    cout << "Breadth First Search (BFS): ";

    graph.BFS(start);

    cout << endl;

    return 0;

}
```

Output: (screenshot)

Conclusion: Therefore, we can implement Graph Traversal techniques by Depth First and Breadth First using adjacency matrix.

Name of Student: Prabodh Badimi

Roll Number: 13

Experiment No: 13

Title: Implement Binary Search algorithm to search an element in the array.

Theory:

Binary Search is a searching algorithm which is used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(log N).

Code:

```cpp
// Binary Search algorithm to search an element in an array
#include <iostream>
#include <algorithm>
using namespace std;
int binarySearch(int arr[], int left, int right, int target) {
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target)
            return mid;
        if (arr[mid] < target)
            left = mid + 1;
        else
            right = mid - 1;
    }
```

```cpp
        return -1;
}

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;
    int arr[n];
    cout << "Enter the elements of the array in sorted
order: ";
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }
    int target;
    cout << "Enter the element to search: ";
    cin >> target;
    int index = binarySearch(arr, 0, n - 1, target);
    if (index != -1)
        cout << "Element found at index: " << index <<
endl;
    else
        cout << "Element not found in the array." <<
endl;
    return 0;
}
```
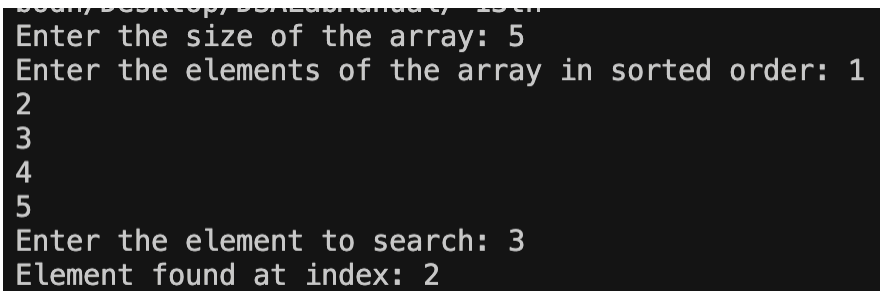
Output: (screenshot)

```
bodh/Desktop/DSALabPrandaC/ 15th
Enter the size of the array: 5
Enter the elements of the array in sorted order: 1
2
3
4
5
Enter the element to search: 3
Element found at index: 2
```

Conclusion: Therefore, we can implement Binary Search algorithm in a sorted array to search the index location of an element present in the array in an efficient manner.

Name of Student: Prabodh Badimi

Roll Number: 13

Experiment No: 14

Title: Implement Bubble Sort algorithm to sort elements of an array in ascending and descending order.

Theory:

In Bubble Sort algorithm, we traverse from left and compare adjacent elements and the higher one is placed at right side. In this way, the largest element is moved to the rightmost end at first. This process is then continued to find the second largest and place it and so on until the data is sorted.

Code:

```cpp
// bubble sort algorithm to sort array in ascending and descending order

#include <iostream>

using namespace std;

void bubbleSortAscending(int arr[], int n) {

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

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

            if (arr[j] > arr[j + 1]) {

                // Swap arr[j] and arr[j+1]

                int temp = arr[j];

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

                arr[j + 1] = temp;

            }

        }

    }
```

```cpp
}
void bubbleSortDescending(int arr[], int n) {
    for (int i = 0; i < n - 1; ++i) {
        for (int j = 0; j < n - i - 1; ++j) {
            if (arr[j] < arr[j + 1]) {
                // Swap arr[j] and arr[j+1]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;
    int arr[n];
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }
    // Sort in ascending order
```

```cpp
    bubbleSortAscending(arr, n);

    cout << "Sorted array in ascending order: ";

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

        cout << arr[i] << " ";

    }

    cout << endl;

    // Sort in descending order

    bubbleSortDescending(arr, n);

    cout << "Sorted array in descending order: ";

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

        cout << arr[i] << " ";

    }

    cout << endl;

    return 0;

}
```

Output: (screenshot)

```
bodn/Desktop/DSALabManual/ 14th
Enter the size of the array: 6
Enter the elements of the array: 1
2
3
4
5
6
Sorted array in ascending order: 1 2 3 4 5 6
Sorted array in descending order: 6 5 4 3 2 1
```

Conclusion: Therefore, we can implement Bubble Sort algorithm to sort the array in ascending or descending order by traversing through the array and comparing the elements to the adjacent elements.