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

DATA STRUCTURES & ALGORITHMS PROGRAMMING LAB

Prepared by:

Name of Student:Ashlin Lee George

Roll No: **150096723011**

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. / Ms. Ashlin Lee George Roll No. **150096723011** Semester 2nd of B.Tech Computer Science & Engineering, ITM Skills University, Kharghar, Navi Mumbai , has completed the term work satisfactorily in subject Data Structures for the academic year 2023 - 2027 as prescribed in the curriculum.

Place:

Date: 08/04/2024

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: Ashlin Lee George

Roll Number: 150096723011

Experiment No: 01

Title:

Implement Array and write a menu driven program to perform all the operation on array element.

Theory:

In the provided menu-driven program, an array is implemented using a class `Menu`. This class encapsulates various operations that can be performed on the array, such as insertion, deletion, and display. Each operation is implemented as a member function of the class, providing a modular and organised approach to array manipulation.

Code:

```
#include <iostream>

using namespace std;


class Menu

{

public:

   int array[100];

   int n;

   void insertatbeg()

   {

      int num;

      cout << "\nEnter the element to insert at [0] index:";

      cin >> num;

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

      {

         array[i] = array[i - 1];
```

```
    }
    array[0] = num;
    n++;
    cout << "\nElements after Insertion:" << endl;
    display();
}
void insertatend()
{
    int num;
    cout << "\nEnter the element to insert at [" << n << "] index:";
    cin >> num;
    array[n] = num;
    n++;
    cout << "\nElements after Insertion:" << endl;
    display();
}
void insertatindex()
{
    int pos, num;
    cout << "\nEnter the index: ";
    cin >> pos;

    if (pos < 0 || pos > n)
    {
        cout << "Invalid index. Cannot insert element." << endl;
        return;
    }
    cout << "\nEnter the element to insert at index " << pos << ": ";
    cin >> num;
    for (int i = n; i > pos; i--)
    {
        array[i] = array[i - 1];
    }
    array[pos] = num;
    n++;
    cout << "\nElements after Insertion:" << endl;
    display();
}

void insertbeforeval()
{
    int val, num;
    cout << "\nEnter the Value before which to insert: ";
    cin >> val;
    int pos = -1;
    for (int i = 0; i < n; i++)
    {
        if (array[i] == val)
        {
            pos = i;
            break;
        }
    }
    if (pos == -1)
    {
        cout << "Value not found. Cannot insert before." << endl;
        return;
    }
    cout << "\nEnter the element to insert before " << val << ": ";
    cin >> num;
    for (int i = n; i > pos; i--)
    {
        array[i] = array[i - 1];
    }
    array[pos] = num;
    n++;
    cout << "\nElements after Insertion:" << endl;
    display();
}
void insertafterval()
{
    int val, num;

    cout << "\nEnter the Value after which to insert: ";
    cin >> val;
    int pos = -1;
    for (int i = 0; i < n; i++)
    {
        if (array[i] == val)
        {
            pos = i;
            break;
        }
    }
    if (pos == -1)
    {
        cout << "Value not found. Cannot insert after." << endl;
        return;
    }
    cout << "\nEnter the element to insert after " << val << ": ";
    cin >> num;
    for (int i = n; i > pos + 1; i--)
    {
        array[i] = array[i - 1];
    }
    array[pos + 1] = num;
    n++;

    cout << "\nElements after Insertion:" << endl;
    display();
}
void deletebyval()
{
    int val;
    cout << "\nEnter the Value to delete: ";
    cin >> val;
    int pos = -1;
    for (int i = 0; i < n; i++)
    {
        if (array[i] == val)
        {
            pos = i;
            break;
        }
    }
    if (pos == -1)
    {
        cout << "Value not found. Cannot delete." << endl;
        return;
    }
    for (int i = pos; i < n - 1; i++)
    {
        array[i] = array[i + 1];
    }

    n--;
    cout << "\nElements after Deletion:" << endl;
    display();
```

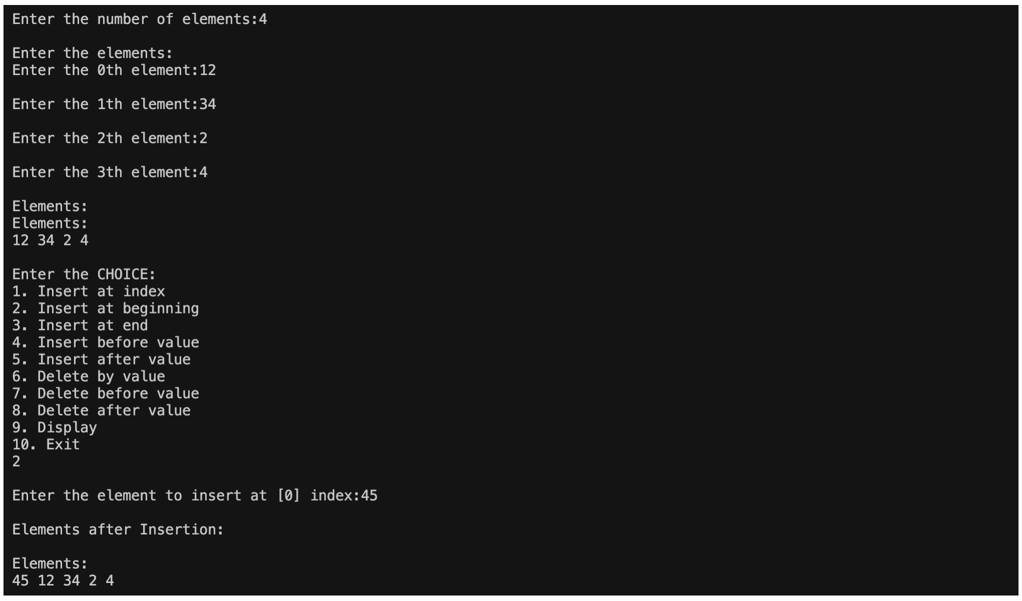
```
}
void deletebeforeval()
{
    int val;
    cout << "\nEnter the Value before which to delete: ";
    cin >> val;
    int pos = -1;
    for (int i = 0; i < n; i++)
    {
        if (array[i] == val)
        {
            pos = i;
            break;
        }
    }
    if (pos <= 0)
    {
        cout << "Value not found or it's the first element. Cannot delete before." << endl;
        return;
    }
    for (int i = pos - 1; i < n - 1; i++)
    {
        array[i] = array[i + 1];
    }
    n--;
    cout << "\nElements after Deletion:" << endl;
    display();
}
void deleteafterval()
{
    int val;
    cout << "\nEnter the Value after which to delete: ";
    cin >> val;
    int pos = -1;
    for (int i = 0; i < n; i++)
    {
        if (array[i] == val)
        {
            pos = i;
            break;
        }
    }
    if (pos == -1 || pos == n - 1)
    {
        cout << "Value not found or it's the last element. Cannot delete after." << endl;
        return;
    }
    for (int i = pos + 1; i < n - 1; i++)
    {
        array[i] = array[i + 1];
    }
    n--;
    cout << "\nElements after Deletion:" << endl;
    display();
}
void display()
{
    cout << "\nElements:" << endl;
    for (int i = 0; i < n; i++)
    {
        cout << array[i] << " ";
    }
    cout << endl;
}
```

```cpp
};
int main()
{
    int n;
    cout << "\nEnter the number of elements:";
    cin >> n;
    Menu oprs;
    oprs.n = n;

    cout << "\nEnter the elements:";
    for (int i = 0; i < n; i++)
    {
        cout << "\nEnter the " << i << "th element:";
        cin >> oprs.array[i];
    }
    cout << "\nElements:";
    oprs.display();
    int cho;
    char choo;
    do
    {
        cout << "\nEnter the CHOICE: ";
        cout << "\n1. Insert at index";
        cout << "\n2. Insert at beginning";
        cout << "\n3. Insert at end";
        cout << "\n4. Insert before value";
        cout << "\n5. Insert after value";
        cout << "\n6. Delete by value";
        cout << "\n7. Delete before value";
        cout << "\n8. Delete after value";
        cout << "\n9. Display";
        cout << "\n10. Exit" << endl;
```

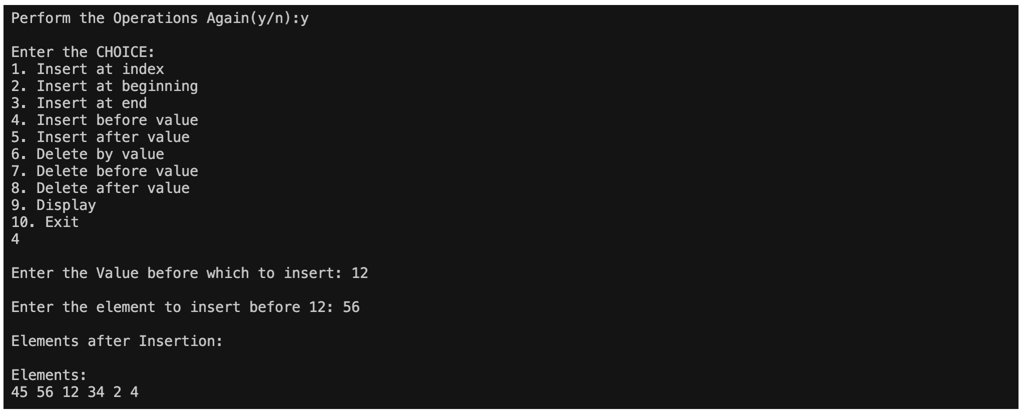
```
cin >> cho;
switch (cho)
{
case 1:
   oprs.insertatindex();
   break;
case 2:
   oprs.insertatbeg();
   break;
case 3:
   oprs.insertatend();
   break;
case 4:
   oprs.insertbeforeval();
   break;
case 5:
   oprs.insertafterval();
   break;
case 6:
   oprs.deletebyval();
   break;
case 7:
   oprs.deletebeforeval();
   break;
case 8:
   oprs.deleteafterval();
   break;
case 9:
   oprs.display();
   break;
case 10:
   cout << "\nExiting...";
```

```
            return 0;
        default:
            cout << "\nInvalid choice. Please enter a valid choice.";
            break;
        }
        cout << "\nPerform the Operations Again(y/n):";
        cin >> choo;
    } while (choo == 'y' || choo == 'Y');
    return 0;
}
```

Output: (screenshot)

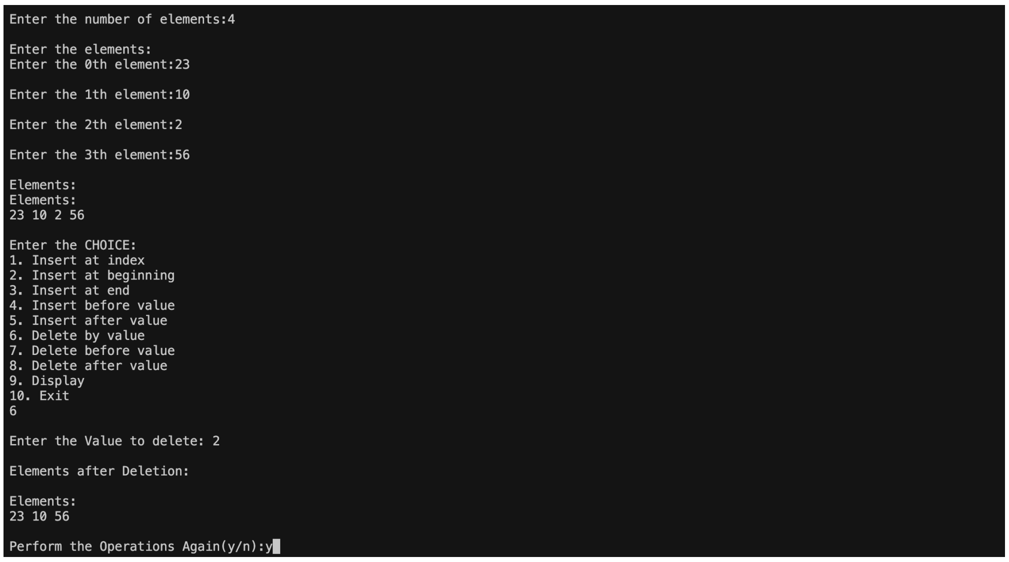


```
Enter the number of elements:4

Enter the elements:
Enter the 0th element:12

Enter the 1th element:34

Enter the 2th element:2

Enter the 3th element:4

Elements:
Elements:
12 34 2 4

Enter the CHOICE:
1. Insert at index
2. Insert at beginning
3. Insert at end
4. Insert before value
5. Insert after value
6. Delete by value
7. Delete before value
8. Delete after value
9. Display
10. Exit
2

Enter the element to insert at [0] index:45

Elements after Insertion:

Elements:
45 12 34 2 4
```

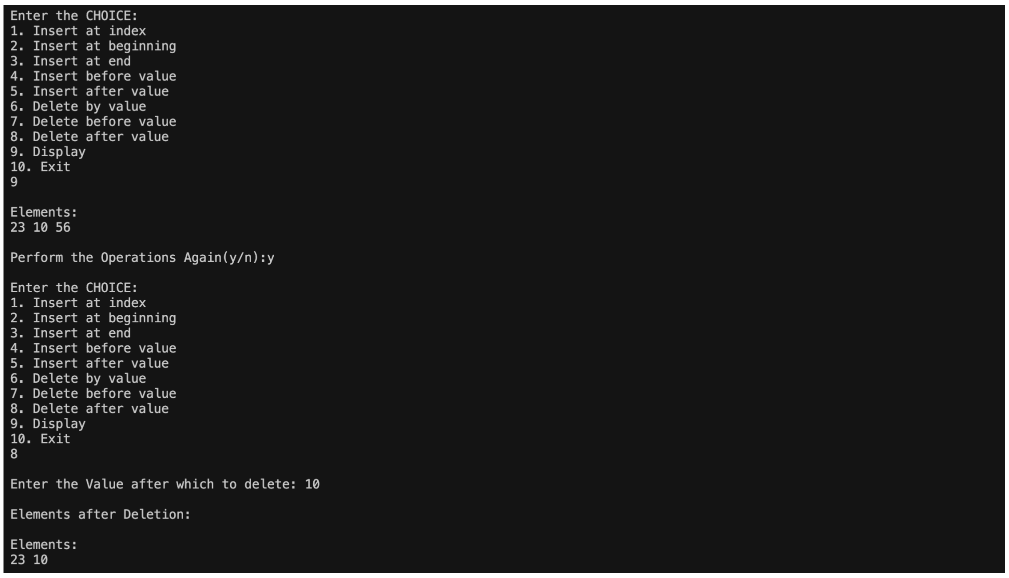


```
Perform the Operations Again(y/n):y

Enter the CHOICE:
1. Insert at index
2. Insert at beginning
3. Insert at end
4. Insert before value
5. Insert after value
6. Delete by value
7. Delete before value
8. Delete after value
9. Display
10. Exit
4

Enter the Value before which to insert: 12

Enter the element to insert before 12: 56

Elements after Insertion:

Elements:
45 56 12 34 2 4
```

Test Case: Any two (screenshot)



```
Enter the number of elements:4

Enter the elements:
Enter the 0th element:23

Enter the 1th element:10

Enter the 2th element:2

Enter the 3th element:56

Elements:
Elements:
23 10 2 56

Enter the CHOICE:
1. Insert at index
2. Insert at beginning
3. Insert at end
4. Insert before value
5. Insert after value
6. Delete by value
7. Delete before value
8. Delete after value
9. Display
10. Exit
6

Enter the Value to delete: 2

Elements after Deletion:

Elements:
23 10 56

Perform the Operations Again(y/n):y
```



```
Enter the CHOICE:
1. Insert at index
2. Insert at beginning
3. Insert at end
4. Insert before value
5. Insert after value
6. Delete by value
7. Delete before value
8. Delete after value
9. Display
10. Exit
9

Elements:
23 10 56

Perform the Operations Again(y/n):y

Enter the CHOICE:
1. Insert at index
2. Insert at beginning
3. Insert at end
4. Insert before value
5. Insert after value
6. Delete by value
7. Delete before value
8. Delete after value
9. Display
10. Exit
8

Enter the Value after which to delete: 10

Elements after Deletion:

Elements:
23 10
```

Conclusion:

In conclusion, the menu-driven program for array operations demonstrates the versatility and practicality of arrays in implementing various data manipulation tasks. By encapsulating array operations within a class and providing a user-friendly interface through a menu-driven approach, the program enables users to easily perform operations such as insertion, deletion, and display on array elements. Overall, the program showcases the flexibility and utility of arrays as essential data structures in programming.

Name of Student: Ashlin Lee George

Roll Number: 150096723011

Experiment No: 02

Title:

Implement Stack ADT using array.

Theory:

A stack is a fundamental data structure that follows the Last In, First Out (LIFO) principle. It is commonly used to store and manage data in a way that supports two primary operations: push and pop. The push operation adds an element to the top of the stack, while the pop operation removes the top element from the stack.

In the provided code, a stack data structure is implemented using an array. The class `Stack` encapsulates the stack operations and maintains an array to store the elements. The `top` variable keeps track of the index of the top element in the stack.

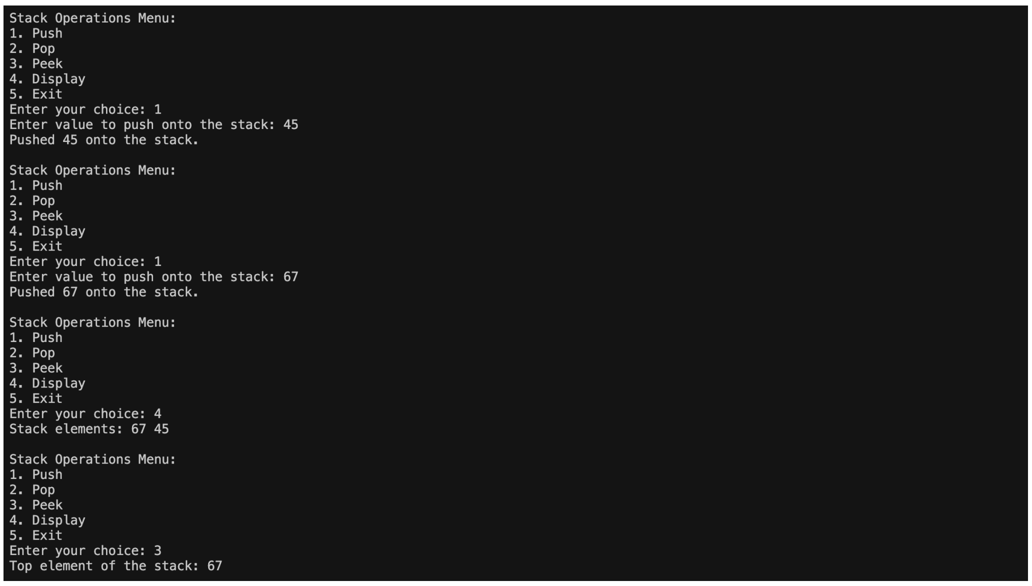
The main function of the program presents a menu-driven interface, allowing users to choose stack operations interactively. Depending on the user's choice, the corresponding stack operation is performed, and appropriate messages or results are displayed.

Code:

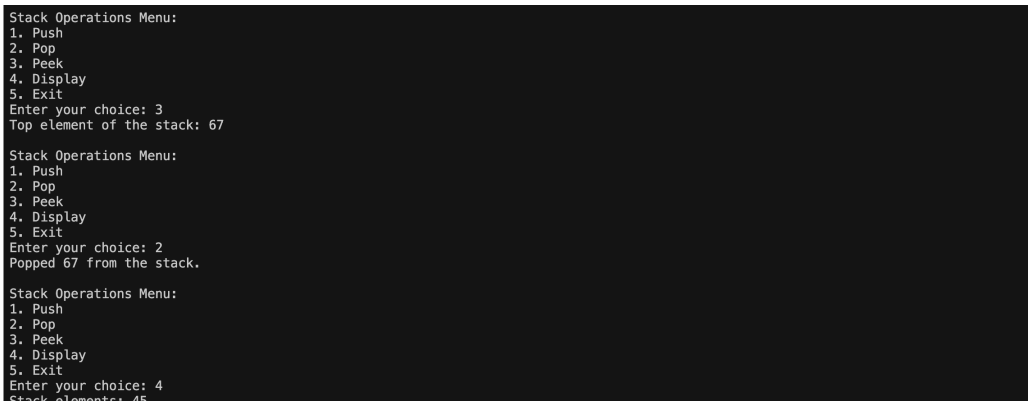
```
#include <iostream>

using namespace std;


const int MAX_SIZE = 100;

class Stack

{

private:

   int arr[MAX_SIZE];

   int top;

public:

   Stack()

   {

      top = -1;

   }
```

```
bool isEmpty()

{

    return top == -1;

}


bool isFull()

{

    return top == MAX_SIZE - 1;

}

void push(int value)

{

    if (isFull())

    {

        cout << "Stack overflow. Cannot push element." << endl;

        return;

    }

    arr[++top] = value;

    cout << "Pushed " << value << " onto the stack." << endl;

}

void pop()

{

    if (isEmpty())

    {

        cout << "Stack underflow. Cannot pop element." << endl;

        return;

    }

    cout << "Popped " << arr[top--] << " from the stack." << endl;

}

int peek()

{

    if (isEmpty())

    {
            cout << "Stack is empty." << endl;
            return -1;
        }
        return arr[top];
    }
    void display()
    {
        if (isEmpty())
        {
            cout << "Stack is empty." << endl;
            return;
        }
        cout << "Stack elements: ";
        for (int i = top; i >= 0; i--)
        {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

int main()
{
    Stack stack;
    int choice, value;
    do
    {
        cout << "\nStack Operations Menu:" << endl;
        cout << "1. Push" << endl;
        cout << "2. Pop" << endl;
        cout << "3. Peek" << endl;
        cout << "4. Display" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            cout << "Enter value to push onto the stack: ";
            cin >> value;
            stack.push(value);
            break;
        case 2:
            stack.pop();
            break;
        case 3:
            value = stack.peek();
            if (value != -1)
            {
                cout << "Top element of the stack: " << value << endl;
            }
            break;
        case 4:
            stack.display();
            break;
        case 5:
            cout << "Exiting..." << endl;
            break;
        default:
            cout << "Invalid choice. Please enter a valid choice." << endl;
        }
    } while (choice != 5);
    return 0;
```

Output: (screenshot)

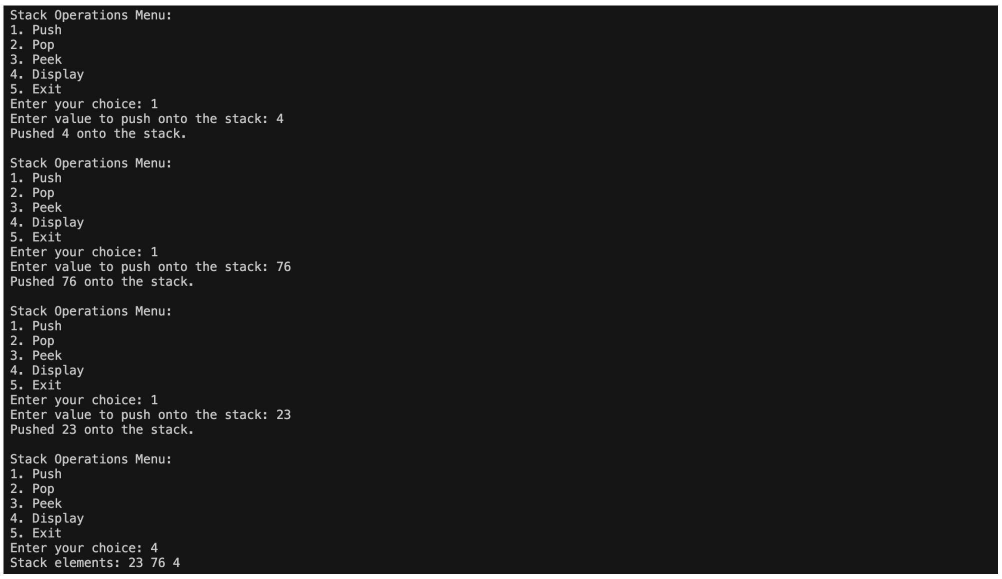


```
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push onto the stack: 45
Pushed 45 onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push onto the stack: 67
Pushed 67 onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 4
Stack elements: 67 45

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 3
Top element of the stack: 67
```

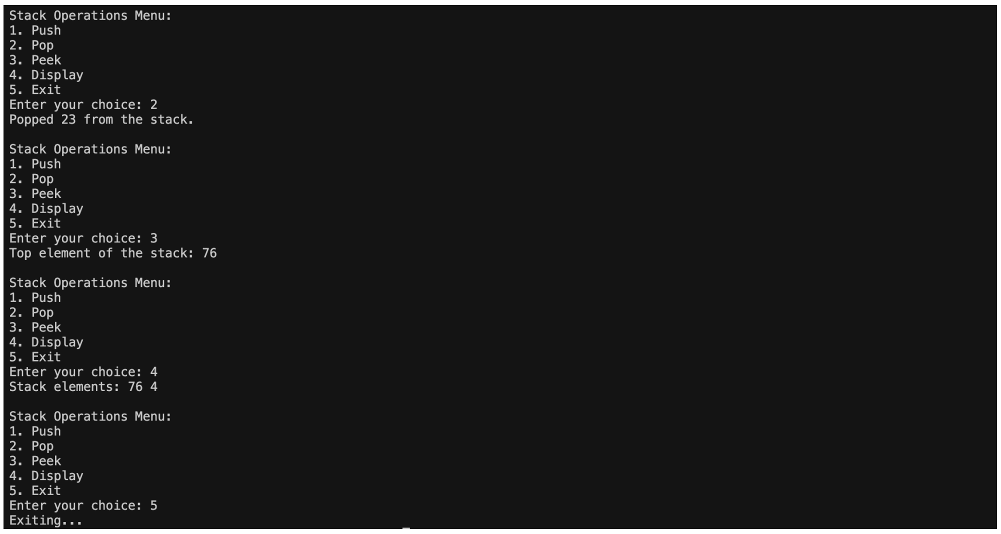


```
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 3
Top element of the stack: 67

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 2
Popped 67 from the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 4
Stack elements: 45
```

Test Case: Any two (screenshot)



```
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push onto the stack: 4
Pushed 4 onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push onto the stack: 76
Pushed 76 onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push onto the stack: 23
Pushed 23 onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 4
Stack elements: 23 76 4
```



```
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 2
Popped 23 from the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 3
Top element of the stack: 76

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 4
Stack elements: 76 4

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 5
Exiting...
```

Conclusion:

In conclusion, the provided code offers a practical implementation of a stack data structure using an array in C++. It demonstrates the core concepts of stacks and how they can be implemented using arrays. By providing a menu-driven interface, the program allows users to interactively perform stack operations, enhancing understanding and facilitating experimentation.

This code serves as a useful tool for learning and practicing stack operations in C++, enabling users to gain hands-on experience with stack manipulation. Moreover, it can be adapted and extended to suit various applications that involve stack-based algorithms or data management tasks. Overall, the program provides a solid foundation for understanding stacks and their applications in computer science.

Name of Student: Ashlin Lee George

Roll Number: 150096723011

Experiment No: 03

Title:

Convert an Infix expression to Postfix expression using stack ADT.

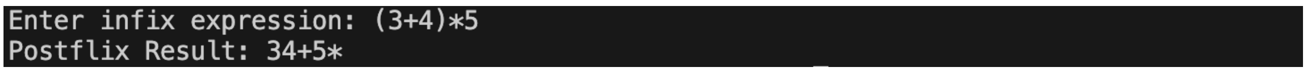
Theory:

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.

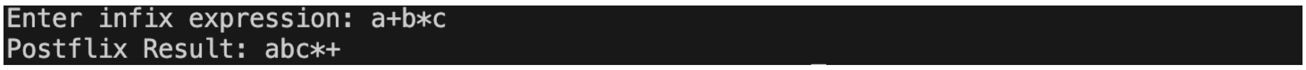
Code:

```
#include <iostream>

using namespace std;


int precedence(char op)

{

  if (op == '+' || op == '-')

  {

    return 1;

  }

  else if (op == '*' || op == '/' || op == '%')

  {

    return 2;

  }

  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)

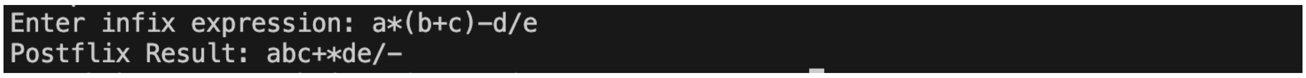


```
Enter infix expression: (3+4)*5
Postflix Result: 34+5*
```

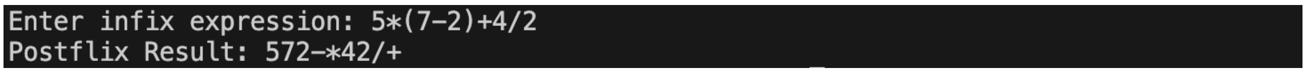


```
Enter infix expression: a+b*c
Postflix Result: abc*+
```

Test Case: Any two (screenshot)



```
Enter infix expression: a*(b+c)-d/e
Postflix Result: abc+*de/-
```



```
Enter infix expression: 5*(7-2)+4/2
Postflix Result: 572-*42/+
```

Conclusion:

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

Name of Student: Ashlin Lee George

Roll Number: 150096723011

Experiment No: 04

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.

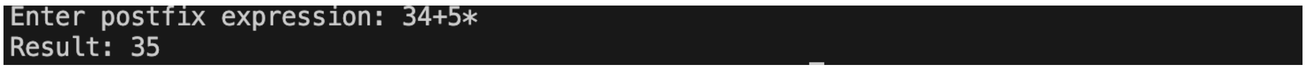
Code:

```
#include <iostream>

#include <string>

using namespace std;

int main()

{

  string expression;

  char stack[100];

  int stack1[100];

  int top = -1, a, b, result = 0;

  cout << "Enter postfix expression: ";

  getline(cin, expression);

  for (int i = 0; i < expression.length(); i++)

  {

    stack[i] = expression[i];

  }

  stack[expression.length()] = ')';

  int i = 0;
```

```
while (stack[i] != ')')
{
    if (stack[i] == '*' || stack[i] == '/' || stack[i] == '%' || stack[i] == '-' || stack[i] == '+')
    {
        a = stack1[top];
        top--;
        b = stack1[top];
        top--;
        if (stack[i] == '*')
        {
            result = b * a;
        }
        else if (stack[i] == '/')
        {
            if (a != 0)
            {
                result = b / a;
            }
            else
            {
                cout << "Error: Division by zero." << endl;
                return 1;
            }
        }
        else if (stack[i] == '%')
        {
            result = b % a;
        }
        else if (stack[i] == '+')
        {
            result = b + a;
        }
```

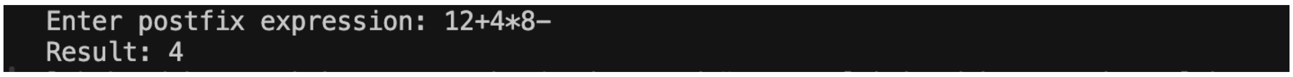
```
            else
            {
                result = b - a;
            }
            top++;
            stack1[top] = result;
        }
        else
        {
            top++;
            stack1[top] = int(stack[i]) - 48;
        }
        i++;
    }
    cout << "Result: " << stack1[top] << endl;
    return 0;
}
```

Output: (screenshot)

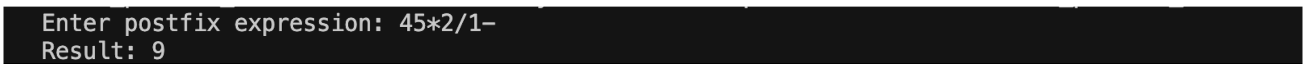


```
Enter postfix expression: 34+5*
Result: 35
```

Test Case: Any two (screenshot)



```
Enter postfix expression: 12+4*8-
Result: 4
```



```
Enter postfix expression: 45*2/1-
Result: 9
```

Conclusion:

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

Name of Student: Ashlin Lee George

Roll Number: 150096723011

Experiment No: 05

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).

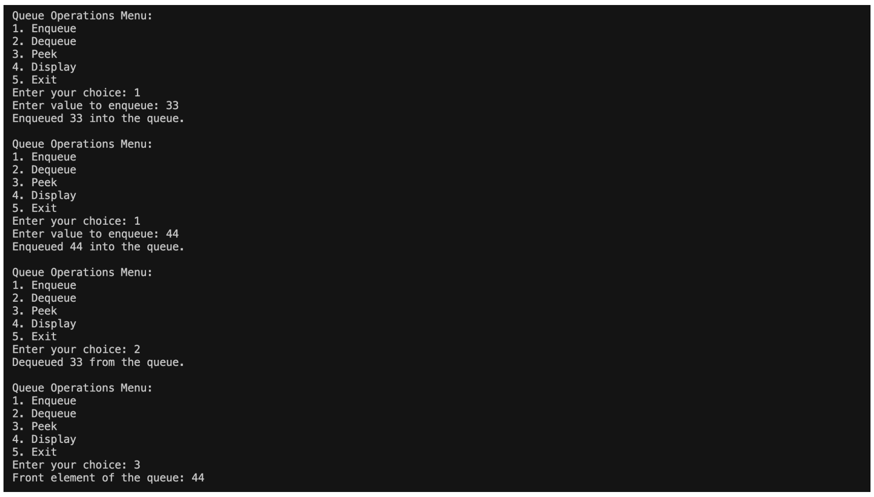
Queue is an Abstract Data Type which can be implemented using Linked List or Array. It consists of two variables named Front and Rear which point to the first and last elements of the stack, respectively. Queue follows FIFO principle(First In, First Out) which means that the element which is inserted 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:

```
#include <iostream>

using namespace std;


const int MAX_SIZE = 100;


class Queue

{

private:

  int arr[MAX_SIZE];

  int front, rear;


public:

  Queue()

  {

    front = -1;

    rear = -1;
```

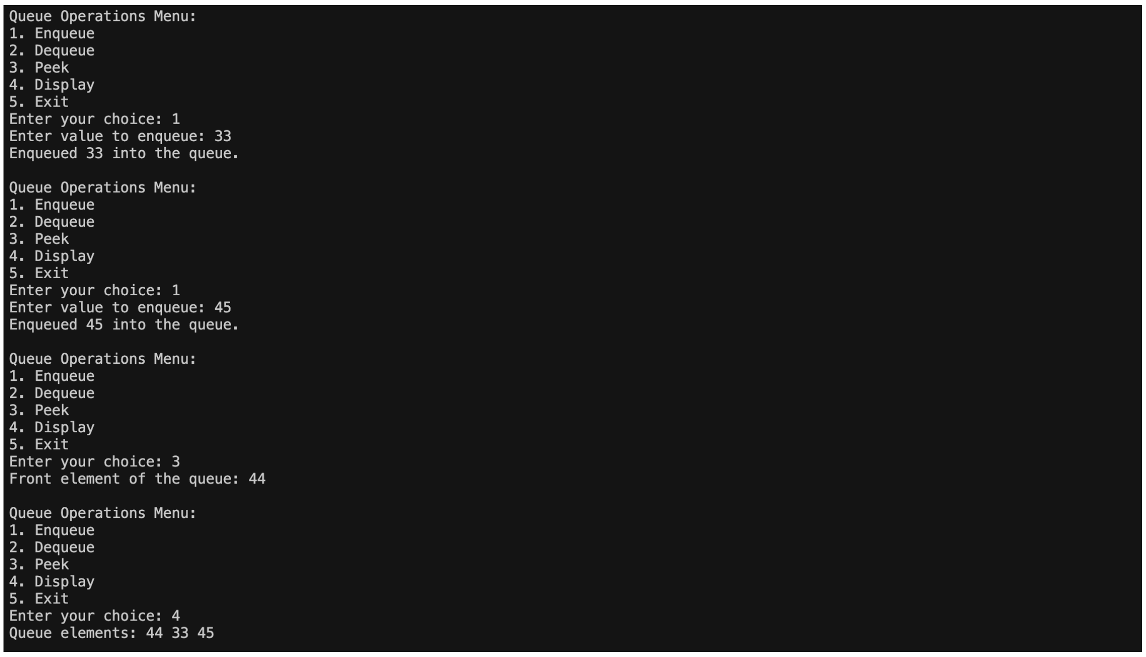
```
}

bool is_empty()
{
    return (front == -1 && rear == -1);
}

bool is_full()
{
    return (rear == MAX_SIZE - 1);
}

void enqueue(int value)
{
    if (is_full())
    {
        cout << "Queue overflow. Cannot enqueue element." << endl;
        return;
    }
    if (is_empty())
    {
        front = 0;
    }
    arr[++rear] = value;
    cout << "Enqueued " << value << " into the queue." << endl;
}

int dequeue()
{
    if (is_empty())
    {
        cout << "Queue underflow. Cannot dequeue element." << endl;
        return -1;
    }
    int dequeued_element = arr[front];
    if (front == rear)
    {
        front = -1;
        rear = -1;
    }
    else
    {
        front++;
    }
    cout << "Dequeued " << dequeued_element << " from the queue." << endl;
    return dequeued_element;
}

int peek()
{
    if (is_empty())
    {
        cout << "Queue is empty." << endl;
        return -1;
    }
    return arr[front];
}

void display()
{
    if (is_empty())
    {
        cout << "Queue is empty." << endl;
        return;
        }
        cout << "Queue elements: ";
        for (int i = front; i <= rear; i++)
        {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

int main()
{
    Queue queue;
    int choice, value;

    do
    {
        cout << "\nQueue Operations Menu:" << endl;
        cout << "1. Enqueue" << endl;
        cout << "2. Dequeue" << endl;
        cout << "3. Peek" << endl;
        cout << "4. Display" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "Enter value to enqueue: ";
            cin >> value;
            queue.enqueue(value);
            break;
        case 2:
            queue.dequeue();
            break;
        case 3:
            value = queue.peek();
            if (value != -1)
            {
                cout << "Front element of the queue: " << value << endl;
            }
            break;
        case 4:
            queue.display();
            break;
        case 5:
            cout << "Exiting..." << endl;
            break;
        default:
            cout << "Invalid choice. Please enter a valid choice." << endl;
        }
    } while (choice != 5);

    return 0;
}
```

Output: (screenshot)

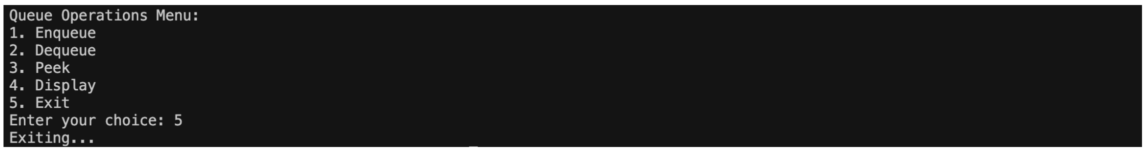


```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 33
Enqueued 33 into the queue.

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 44
Enqueued 44 into the queue.

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 2
Dequeued 33 from the queue.

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 3
Front element of the queue: 44
```

Test Case: Any two (screenshot)



```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 33
Enqueued 33 into the queue.

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 45
Enqueued 45 into the queue.

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 3
Front element of the queue: 44

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Queue elements: 44 33 45
```



```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 5
Exiting...
```

Conclusion:

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

Name of Student: Ashlin Lee George

Roll Number: 150096723011

Experiment No: 06

Title:

Implement Circular Queue ADT using array.

Theory:

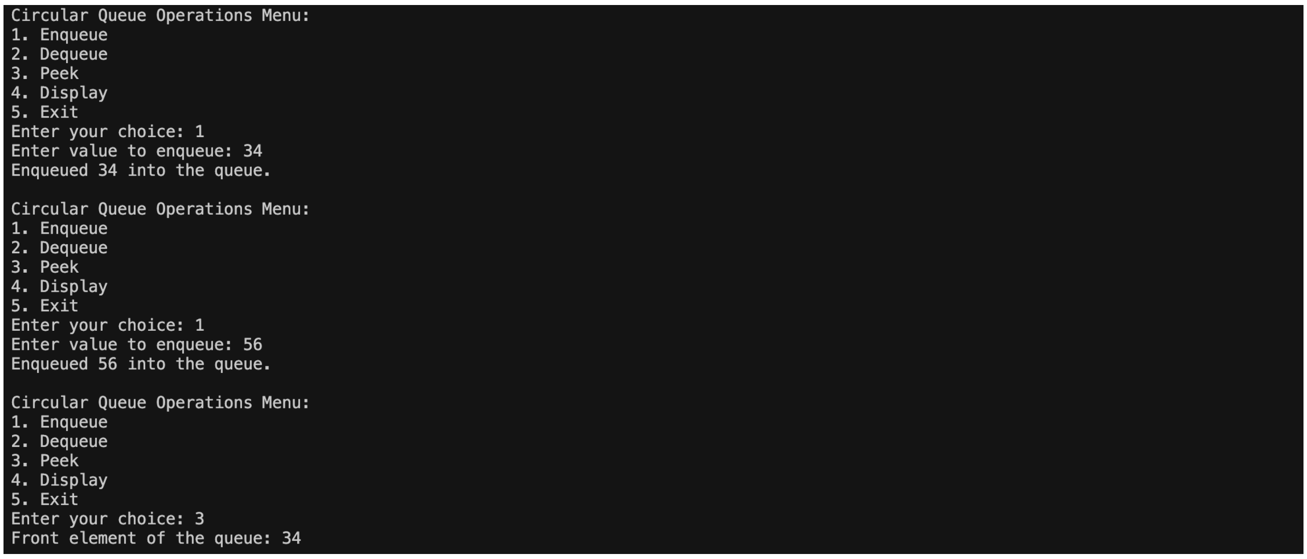
Array is a collection of elements of similar data types and has a fixed size. We can access an element of the array through it's index. Indexing starts from 0 till n-1(where n=size of array).

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:

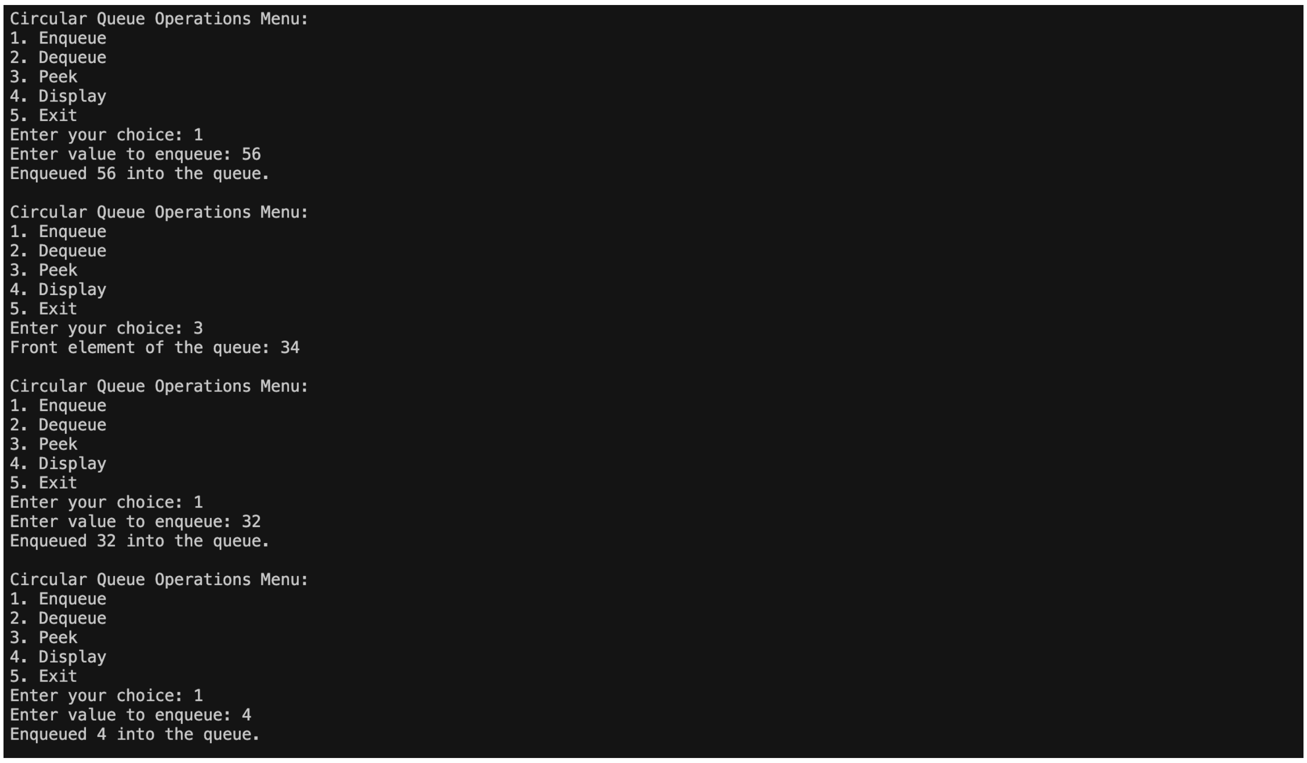
```
#include <iostream>

using namespace std;


const int MAX_SIZE = 100;


class Queue

{

private:

   int arr[MAX_SIZE];

   int front, rear;


public:

   Queue()

   {
       front = -1;
       rear = -1;
   }

   bool is_empty()
   {
       return (front == -1 && rear == -1);
   }

   bool is_full()
   {
       return (rear == MAX_SIZE - 1);
   }

   void enqueue(int value)
   {
       if (is_full())
       {
           cout << "Queue overflow. Cannot enqueue element." << endl;
           return;
       }
       if (is_empty())
       {
           front = 0;
       }
       arr[++rear] = value;
       cout << "Enqueued " << value << " into the queue." << endl;
   }

   int dequeue()
   {
       if (is_empty())
       {
           cout << "Queue underflow. Cannot dequeue element." << endl;
           return -1;
       }
       int dequeued_element = arr[front];
       if (front == rear)
       {
           front = -1;
           rear = -1;
       }
       else
       {
           front++;
       }
       cout << "Dequeued " << dequeued_element << " from the queue." << endl;
       return dequeued_element;
   }

   int peek()
   {
       if (is_empty())
       {
           cout << "Queue is empty." << endl;
           return -1;
       }
       return arr[front];
   }

   void display()
   {
       if (is_empty())
       {
               cout << "Queue is empty." << endl;
               return;
           }
           cout << "Queue elements: ";
           for (int i = front; i <= rear; i++)
           {
               cout << arr[i] << " ";
           }
           cout << endl;
       }
   };

   int main()
   {
       Queue queue;
       int choice, value;

       do
       {
           cout << "\nQueue Operations Menu:" << endl;
           cout << "1. Enqueue" << endl;
           cout << "2. Dequeue" << endl;
           cout << "3. Peek" << endl;
           cout << "4. Display" << endl;
           cout << "5. Exit" << endl;
           cout << "Enter your choice: ";
           cin >> choice;

           switch (choice)
           {
           case 1:
               cout << "Enter value to enqueue: ";
               cin >> value;
               queue.enqueue(value);
               break;
           case 2:
               queue.dequeue();
               break;
           case 3:
               value = queue.peek();
               if (value != -1)
               {
                   cout << "Front element of the queue: " << value << endl;
               }
               break;
           case 4:
               queue.display();
               break;
           case 5:
               cout << "Exiting..." << endl;
               break;
           default:
               cout << "Invalid choice. Please enter a valid choice." << endl;
           }
       } while (choice != 5);

       return 0;
   }
```

Output: (screenshot)

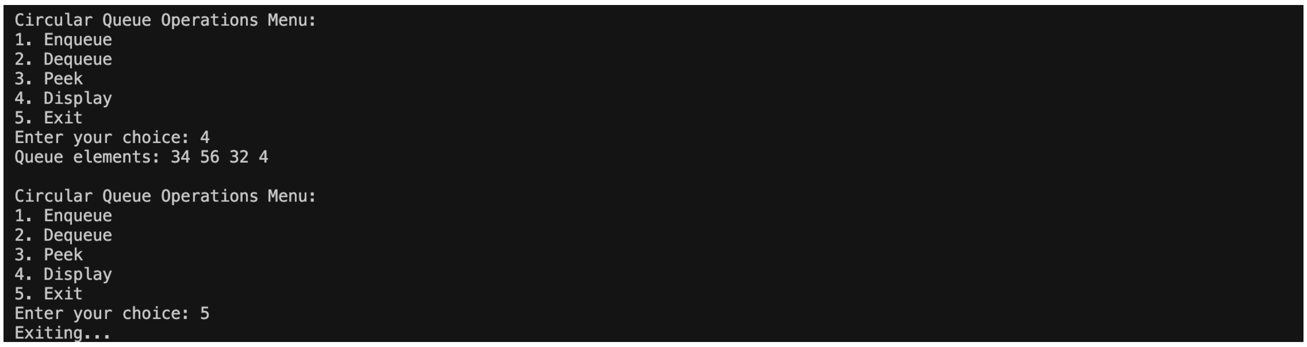


```
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 34
Enqueued 34 into the queue.

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 56
Enqueued 56 into the queue.

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 3
Front element of the queue: 34
```

Test Case: Any two (screenshot)



```
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 56
Enqueued 56 into the queue.

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 3
Front element of the queue: 34

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 32
Enqueued 32 into the queue.

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 4
Enqueued 4 into the queue.
```



```
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Queue elements: 34 56 32 4

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 5
Exiting...
```

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: Ashlin Lee George

Roll Number: 150096723011

Experiment No: 07

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.

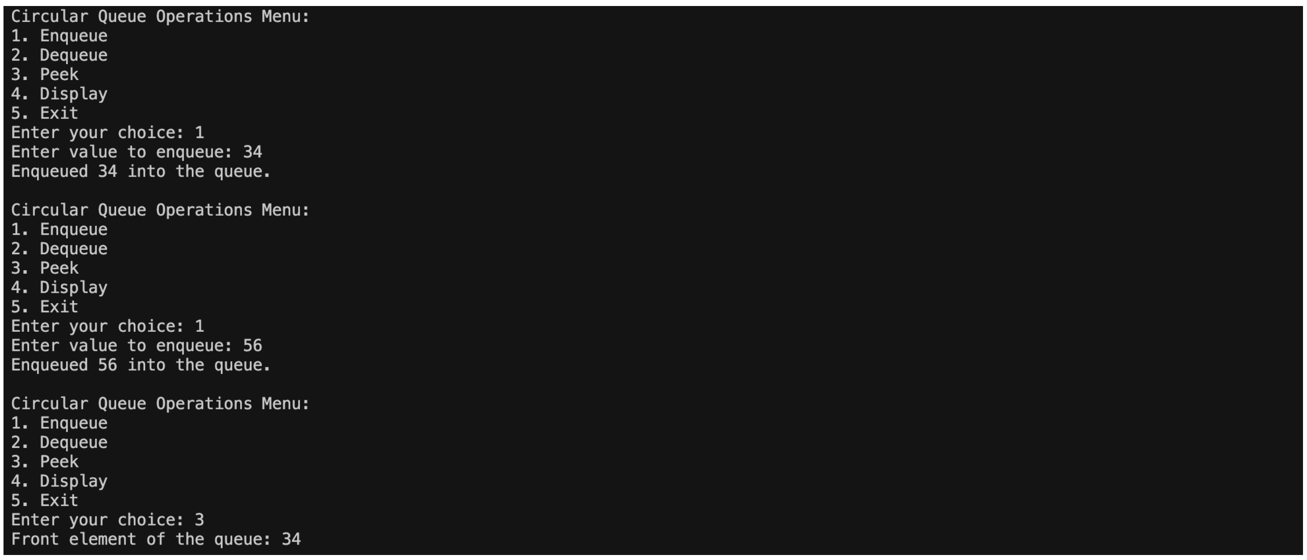
Code:

```
#include <iostream>

using namespace std;


const int MAX_SIZE = 100;


class Queue

{

private:

    int arr[MAX_SIZE];

    int front, rear;


public:

    Queue()

    {

        front = -1;

        rear = -1;

    }
```

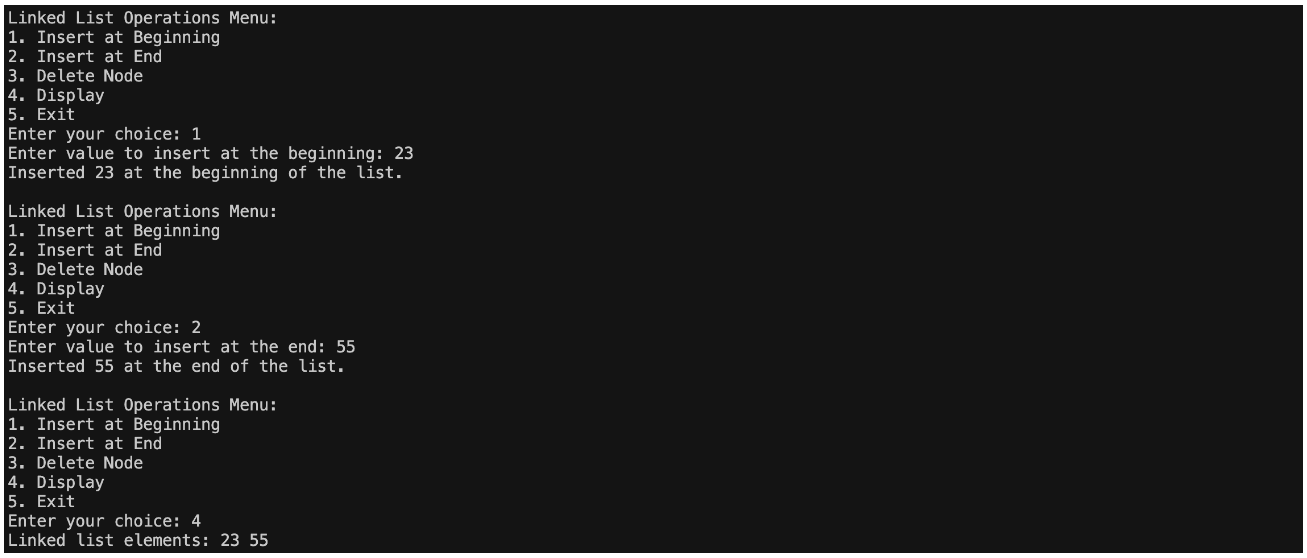
```
bool is_empty()

{

    return (front == -1 && rear == -1);

}


bool is_full()

{

    return (rear == MAX_SIZE - 1);

}


void enqueue(int value)

{

    if (is_full())

    {

        cout << "Queue overflow. Cannot enqueue element." << endl;

        return;

    }

    if (is_empty())

    {

        front = 0;

    }

    arr[++rear] = value;

    cout << "Enqueued " << value << " into the queue." << endl;

}


int dequeue()

{

    if (is_empty())

    {

        cout << "Queue underflow. Cannot dequeue element." << endl;

        return -1;

    }
    int dequeued_element = arr[front];
    if (front == rear)
    {
        front = -1;
        rear = -1;
    }
    else
    {
        front++;
    }
    cout << "Dequeued " << dequeued_element << " from the queue." << endl;
    return dequeued_element;
}

int peek()
{
    if (is_empty())
    {
        cout << "Queue is empty." << endl;
        return -1;
    }
    return arr[front];
}

void display()
{
    if (is_empty())
    {
        cout << "Queue is empty." << endl;
        return;
    }
    cout << "Queue elements: ";
```

```
        for (int i = front; i <= rear; i++)
        {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

int main()
{
    Queue queue;
    int choice, value;

    do
    {
        cout << "\nQueue Operations Menu:" << endl;
        cout << "1. Enqueue" << endl;
        cout << "2. Dequeue" << endl;
        cout << "3. Peek" << endl;
        cout << "4. Display" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "Enter value to enqueue: ";
            cin >> value;
            queue.enqueue(value);
            break;
        case 2:
            queue.dequeue();
            break;
        case 3:
            value = queue.peek();
            if (value != -1)
            {
                cout << "Front element of the queue: " << value << endl;
            }
            break;
        case 4:
            queue.display();
            break;
        case 5:
            cout << "Exiting..." << endl;
            break;
        default:
            cout << "Invalid choice. Please enter a valid choice." << endl;
        }
    } while (choice != 5);

    return 0;
}
```

Output: (screenshot)

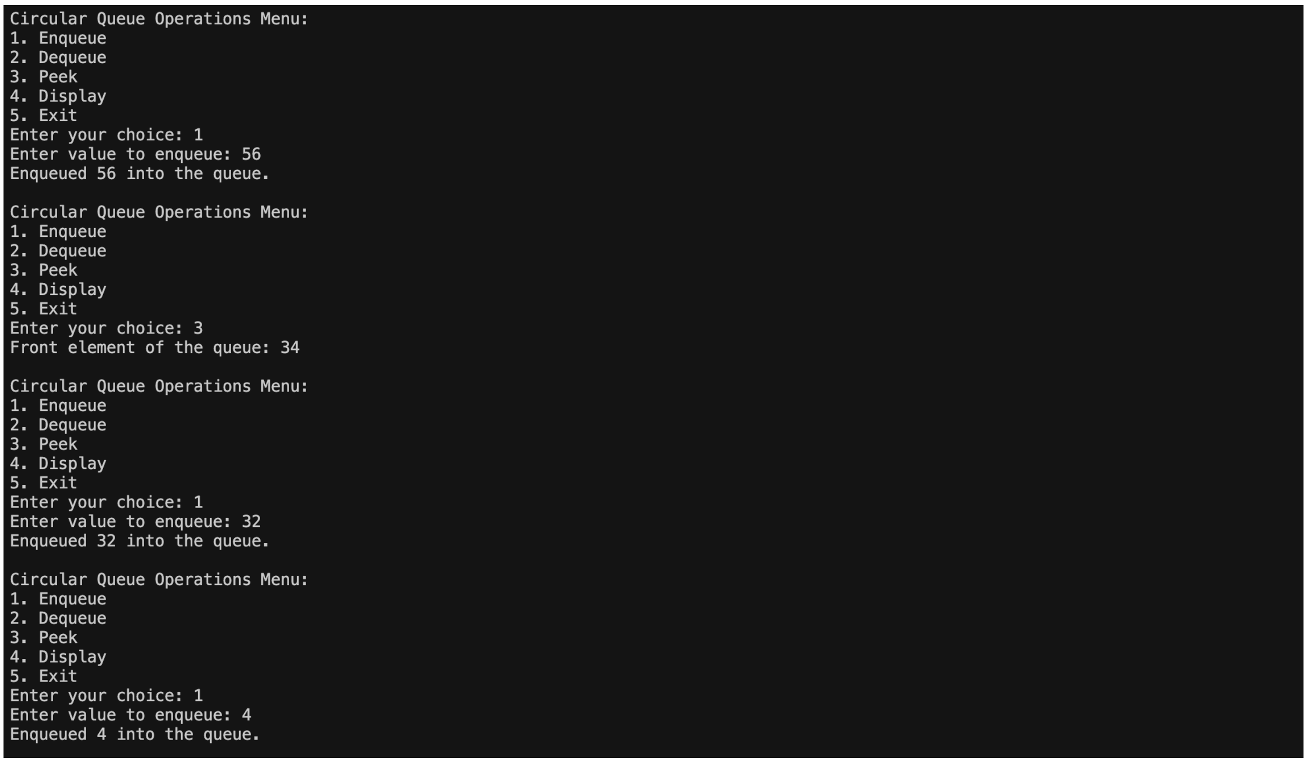


```
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 34
Enqueued 34 into the queue.

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 56
Enqueued 56 into the queue.

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 3
Front element of the queue: 34
```

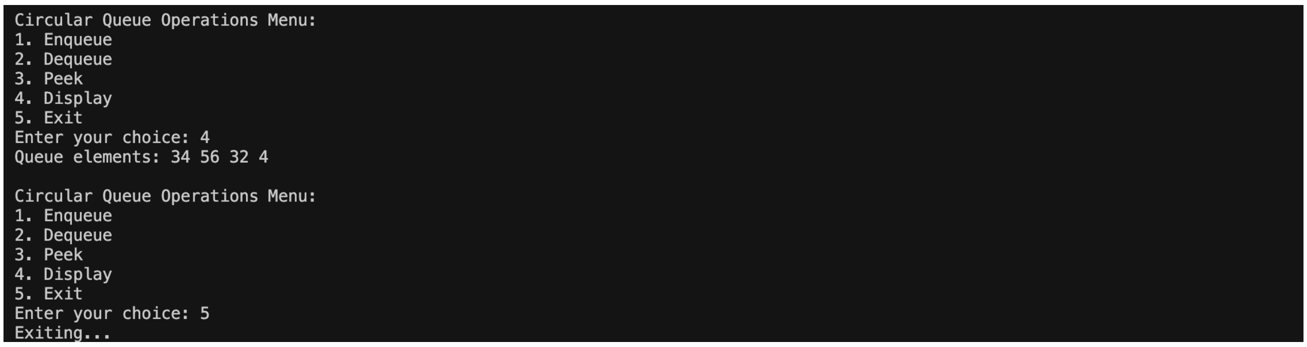


```
Linked List Operations Menu:
1. Insert at Beginning
2. Insert at End
3. Delete Node
4. Display
5. Exit
Enter your choice: 1
Enter value to insert at the beginning: 23
Inserted 23 at the beginning of the list.

Linked List Operations Menu:
1. Insert at Beginning
2. Insert at End
3. Delete Node
4. Display
5. Exit
Enter your choice: 2
Enter value to insert at the end: 55
Inserted 55 at the end of the list.

Linked List Operations Menu:
1. Insert at Beginning
2. Insert at End
3. Delete Node
4. Display
5. Exit
Enter your choice: 4
Linked list elements: 23 55
```

Test Case: Any two (screenshot)



```
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 56
Enqueued 56 into the queue.

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 3
Front element of the queue: 34

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 32
Enqueued 32 into the queue.

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 4
Enqueued 4 into the queue.
```



```
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Queue elements: 34 56 32 4

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 5
Exiting...
```

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: Ashlin Lee George

Roll Number: 150096723011

Experiment No: 08

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.

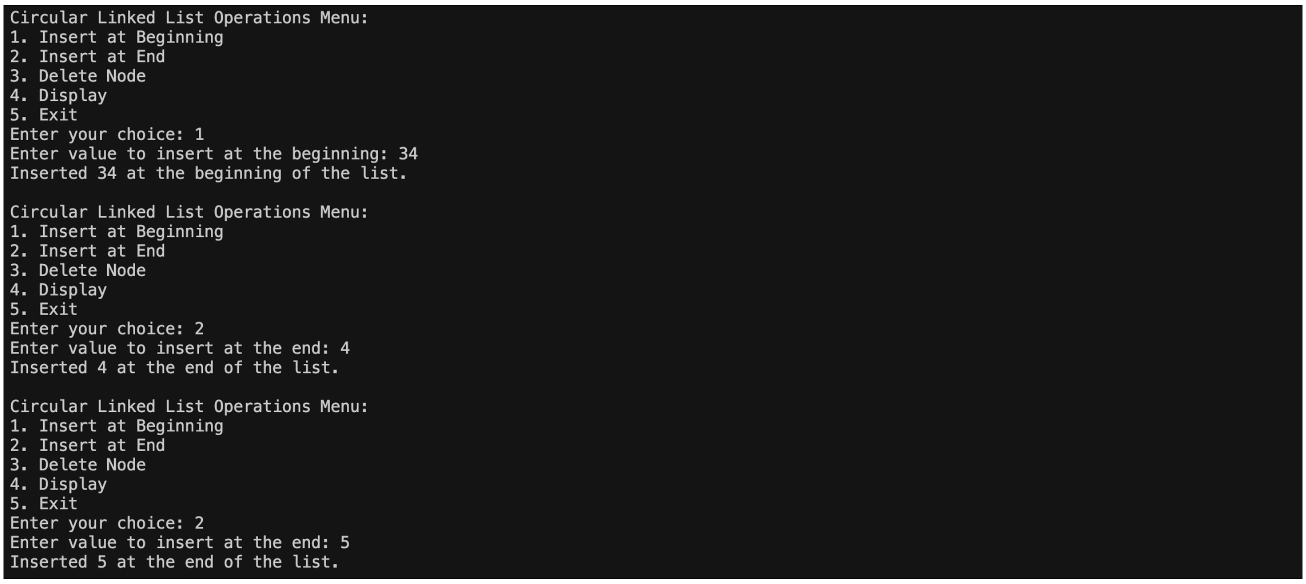
Code:

```
#include <iostream>
using namespace std;

class Node
{
public:
   int data;
   Node *next;

   Node(int value)
   {
      data = value;
      next = nullptr;
   }
};

class CircularLinkedList
```

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

    ~CircularLinkedList()
    {
        if (head == nullptr)
            return;
        Node *current = head->next;
        while (current != head)
        {
            Node *temp = current;
            current = current->next;
            delete temp;
        }
        delete head;
        head = nullptr;
    }

    bool is_empty()
    {
        return head == nullptr;
    }

    void insert_at_beginning(int value)
    {
        Node *newNode = new Node(value);
        if (head == nullptr)
        {
            newNode->next = newNode;
            head = newNode;
        }
        else
        {
            Node *current = head;
            while (current->next != head)
            {
                current = current->next;
            }
            newNode->next = head;
            head = newNode;
            current->next = head;
        }
        cout << "Inserted " << value << " at the beginning of the list." << endl;
    }

    void insert_at_end(int value)
    {
        Node *newNode = new Node(value);
        if (head == nullptr)
        {
            newNode->next = newNode;
            head = newNode;
        }
        else
        {
            Node *current = head;
            while (current->next != head)
            {
                current = current->next;
            }
            current->next = newNode;
            newNode->next = head;
        }
        cout << "Inserted " << value << " at the end of the list." << endl;
    }

    void delete_node(int value)
    {
        if (head == nullptr)
        {
            cout << "Circular linked list is empty. Cannot delete." << endl;
            return;
        }

        Node *current = head;
        Node *prev = nullptr;

        do
        {
            if (current->data == value)
            {
                if (current == head)
                {
                    Node *lastNode = head;
                    while (lastNode->next != head)
                    {
                        lastNode = lastNode->next;
                    }
                    if (head == head->next)
                    {
                        delete head;
                        head = nullptr;
                    }
                    else
                    {
                        lastNode->next = head->next;
                        delete head;
                        head = lastNode->next;
                    }
                    cout << "Deleted " << value << " from the list." << endl;
                    return;
                }
                else
                {

                    prev->next = current->next;
                    delete current;
                    cout << "Deleted " << value << " from the list." << endl;
                    return;
                }
            }
            prev = current;
            current = current->next;
        } while (current != head);

        cout << "Value " << value << " not found in the list. Cannot delete." << endl;
    }

    void display()
    {
        if (is_empty())
        {
            cout << "Circular linked list is empty." << endl;
            return;
        }
        cout << "Circular linked list elements: ";
        Node *current = head;
        do
        {
            cout << current->data << " ";
            current = current->next;
        } while (current != head);
        cout << endl;
    }
};

int main()
{
    CircularLinkedList list;
    int choice, value;

    do
    {
        cout << "\nCircular Linked List Operations Menu:" << endl;
        cout << "1. Insert at Beginning" << endl;
        cout << "2. Insert at End" << endl;
        cout << "3. Delete Node" << endl;
        cout << "4. Display" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice)
        {
```

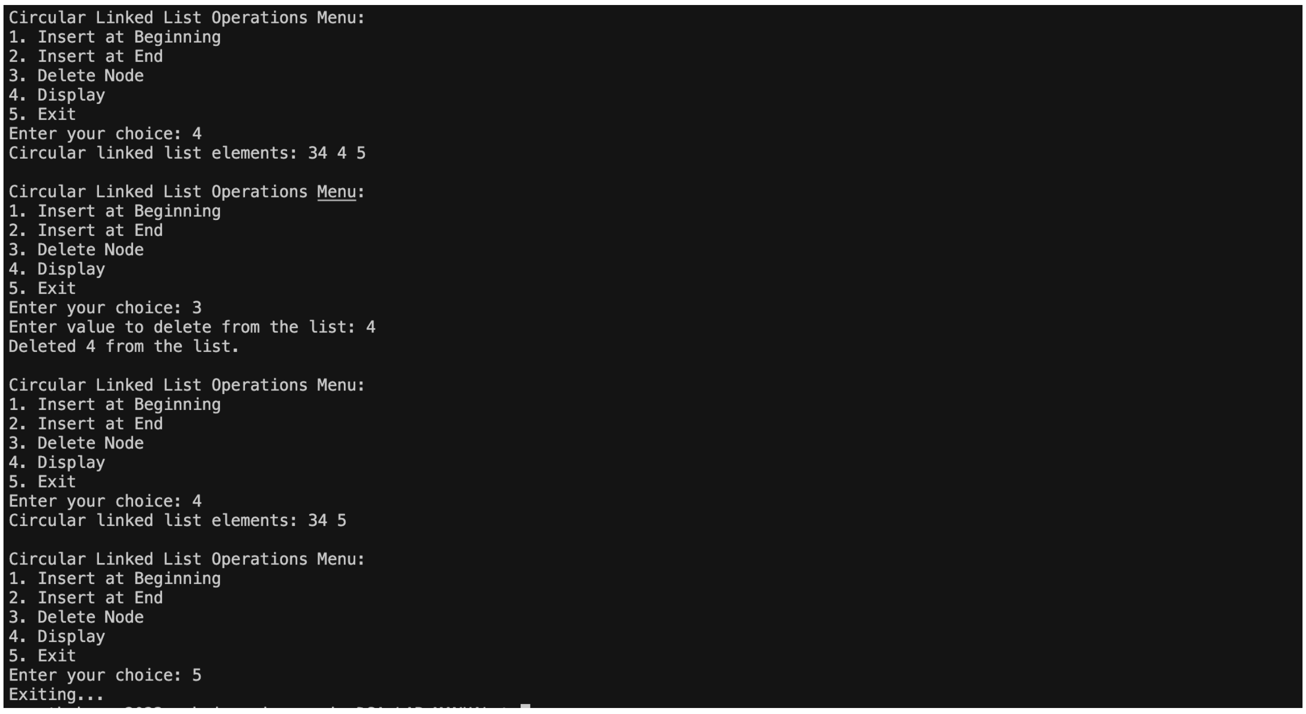
```
        case 1:
            cout << "Enter value to insert at the beginning: ";
            cin >> value;
            list.insert_at_beginning(value);
            break;
        case 2:
            cout << "Enter value to insert at the end: ";
            cin >> value;
            list.insert_at_end(value);
            break;
        case 3:
            cout << "Enter value to delete from the list: ";
            cin >> value;
            list.delete_node(value);
            break;
        case 4:
            list.display();
            break;
        case 5:
            cout << "Exiting..." << endl;
            break;
        default:
            cout << "Invalid choice. Please enter a valid choice." << endl;
        }
    } while (choice != 5);

    return 0;
}
```

Output: (screenshot)



```
Circular Linked List Operations Menu:
1. Insert at Beginning
2. Insert at End
3. Delete Node
4. Display
5. Exit
Enter your choice: 1
Enter value to insert at the beginning: 34
Inserted 34 at the beginning of the list.

Circular Linked List Operations Menu:
1. Insert at Beginning
2. Insert at End
3. Delete Node
4. Display
5. Exit
Enter your choice: 2
Enter value to insert at the end: 4
Inserted 4 at the end of the list.

Circular Linked List Operations Menu:
1. Insert at Beginning
2. Insert at End
3. Delete Node
4. Display
5. Exit
Enter your choice: 2
Enter value to insert at the end: 5
Inserted 5 at the end of the list.
```

Test Case: Any two (screenshot)



```
Circular Linked List Operations Menu:
1. Insert at Beginning
2. Insert at End
3. Delete Node
4. Display
5. Exit
Enter your choice: 4
Circular linked list elements: 34 4 5

Circular Linked List Operations Menu:
1. Insert at Beginning
2. Insert at End
3. Delete Node
4. Display
5. Exit
Enter your choice: 3
Enter value to delete from the list: 4
Deleted 4 from the list.

Circular Linked List Operations Menu:
1. Insert at Beginning
2. Insert at End
3. Delete Node
4. Display
5. Exit
Enter your choice: 4
Circular linked list elements: 34 5

Circular Linked List Operations Menu:
1. Insert at Beginning
2. Insert at End
3. Delete Node
4. Display
5. Exit
Enter your choice: 5
Exiting...
```

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: Ashlin Lee George

Roll Number: 150096723011

Experiment No: 09

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.

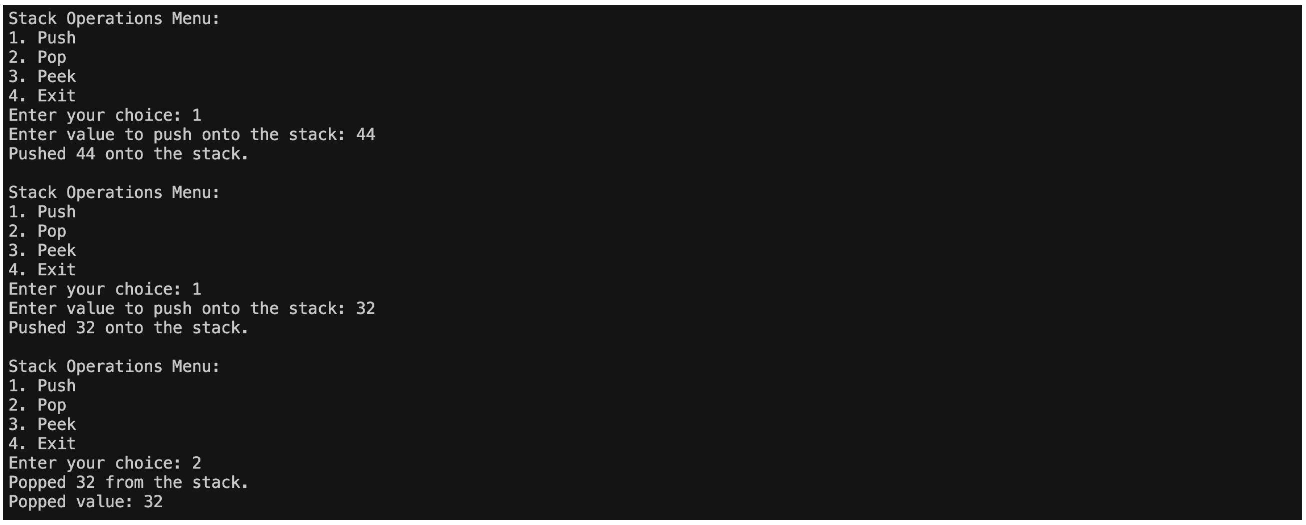
Code:

```
#include <iostream>

using namespace std;


class Node

{

public:

   int data;

   Node *next;


   Node(int value)

   {

      data = value;
```

```
        next = nullptr;
    }
};

class Stack
{
private:
    Node *top;

public:
    Stack()
    {
        top = nullptr;
    }

    ~Stack()
    {
        Node *current = top;
        Node *next;
        while (current != nullptr)
        {
            next = current->next;
            delete current;
            current = next;
        }
        top = nullptr;
    }

    bool is_empty()
    {
        return top == nullptr;
    }
```

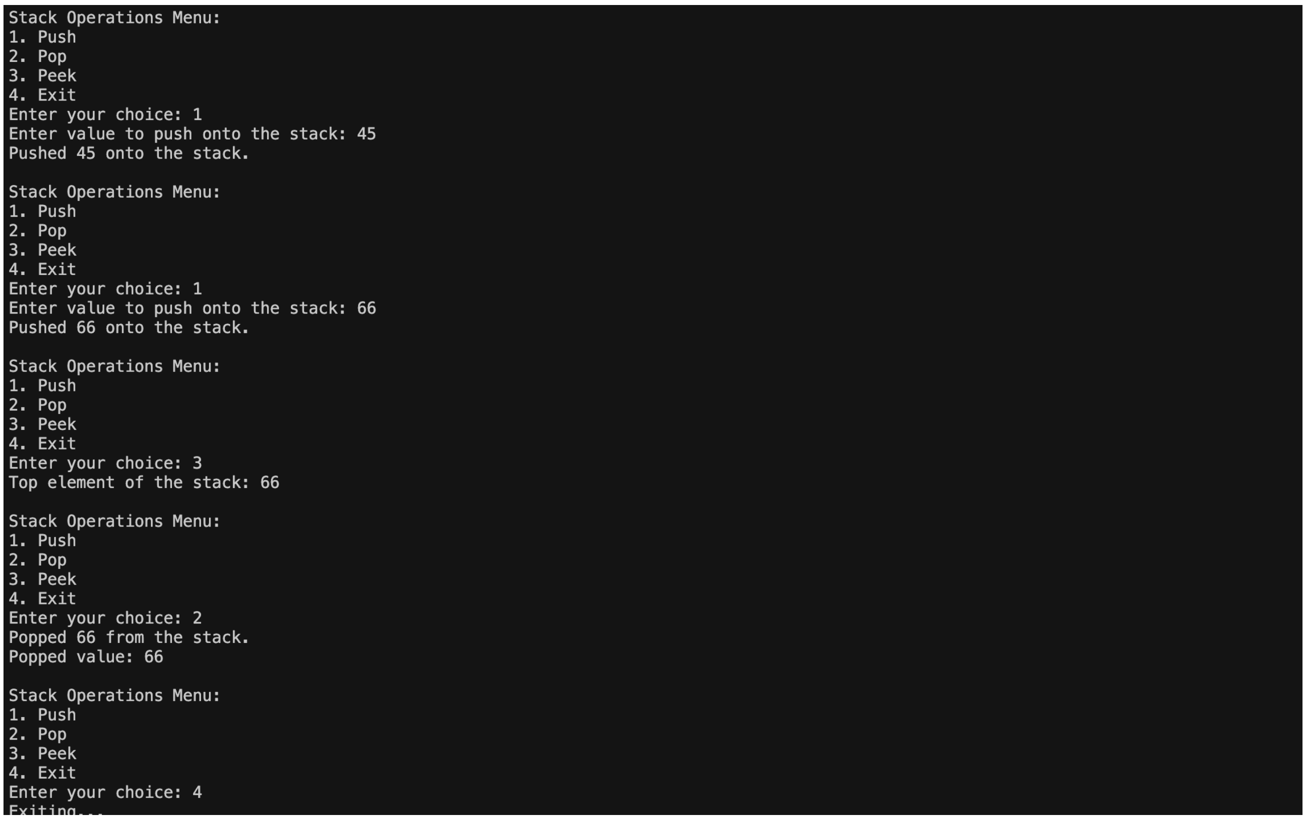
```
void push(int value)
{
    Node *newNode = new Node(value);
    newNode->next = top;
    top = newNode;
    cout << "Pushed " << value << " onto the stack." << endl;
}

int pop()
{
    if (is_empty())
    {
        cout << "Stack underflow. Cannot pop element." << endl;
        return -1;
    }
    int popped_element = top->data;
    Node *temp = top;
    top = top->next;
    delete temp;
    cout << "Popped " << popped_element << " from the stack." << endl;
    return popped_element;
}

int peek()
{
    if (is_empty())
    {
        cout << "Stack is empty. No top element to peek." << endl;
        return -1;
    }
    return top->data;
    }
};

int main()
{
    Stack stack;
    int choice, value;

    do
    {
        cout << "\nStack Operations Menu:" << endl;
        cout << "1. Push" << endl;
        cout << "2. Pop" << endl;
        cout << "3. Peek" << endl;
        cout << "4. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "Enter value to push onto the stack: ";
            cin >> value;
            stack.push(value);
            break;
        case 2:
            value = stack.pop();
            if (value != -1)
            {
                cout << "Popped value: " << value << endl;
            }
            break;
        case 3:
            value = stack.peek();
            if (value != -1)
            {
                cout << "Top element of the stack: " << value << endl;
            }
            break;
        case 4:
            cout << "Exiting..." << endl;
            break;
        default:
            cout << "Invalid choice. Please enter a valid choice." << endl;
        }
    } while (choice != 4);

    return 0;
}
```

Output: (screenshot)

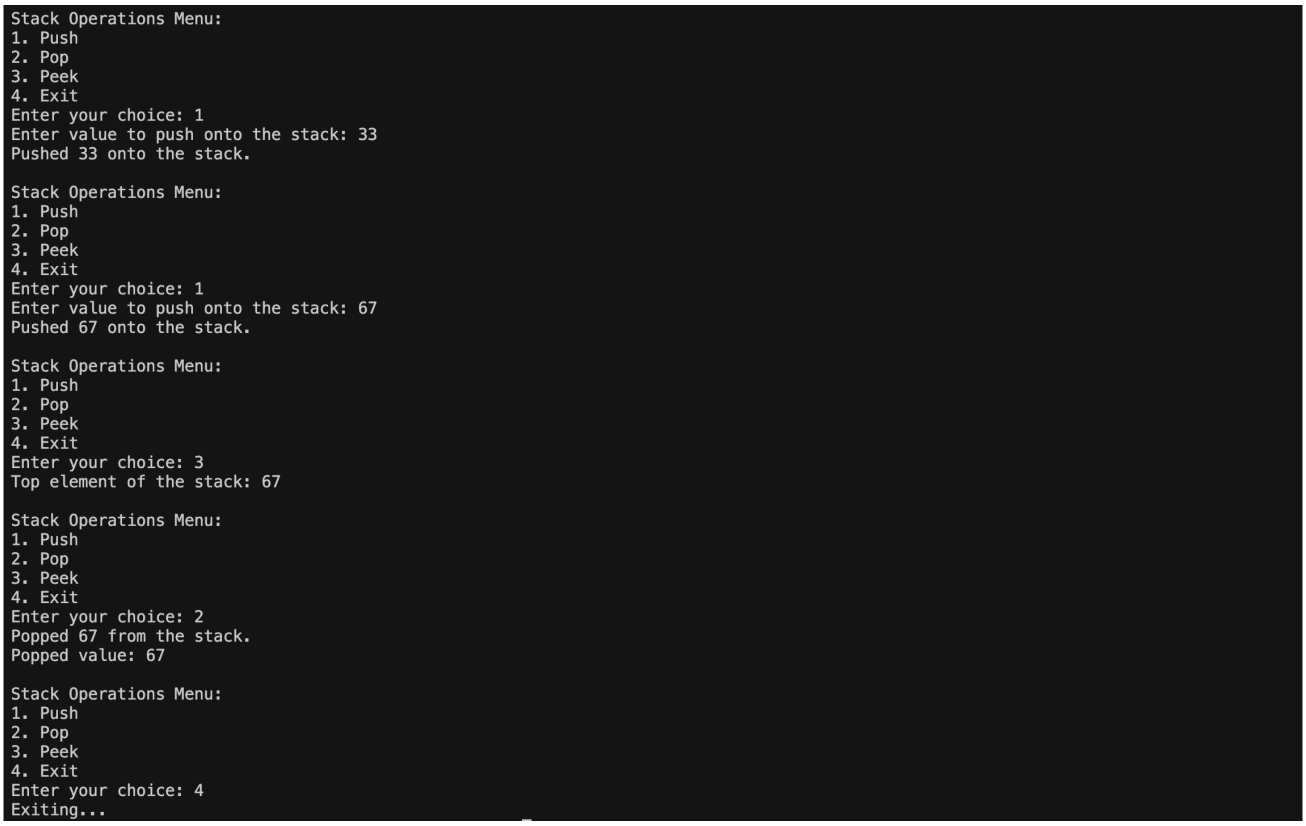


```
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 1
Enter value to push onto the stack: 44
Pushed 44 onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 1
Enter value to push onto the stack: 32
Pushed 32 onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 2
Popped 32 from the stack.
Popped value: 32
```

Test Case: Any two (screenshot)



```
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 1
Enter value to push onto the stack: 45
Pushed 45 onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 1
Enter value to push onto the stack: 66
Pushed 66 onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 3
Top element of the stack: 66

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 2
Popped 66 from the stack.
Popped value: 66

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 4
Exiting...
```



```
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 1
Enter value to push onto the stack: 33
Pushed 33 onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 1
Enter value to push onto the stack: 67
Pushed 67 onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 3
Top element of the stack: 67

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 2
Popped 67 from the stack.
Popped value: 67

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 4
Exiting...
```

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: Ashlin Lee George

Roll Number: 150096723011

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.

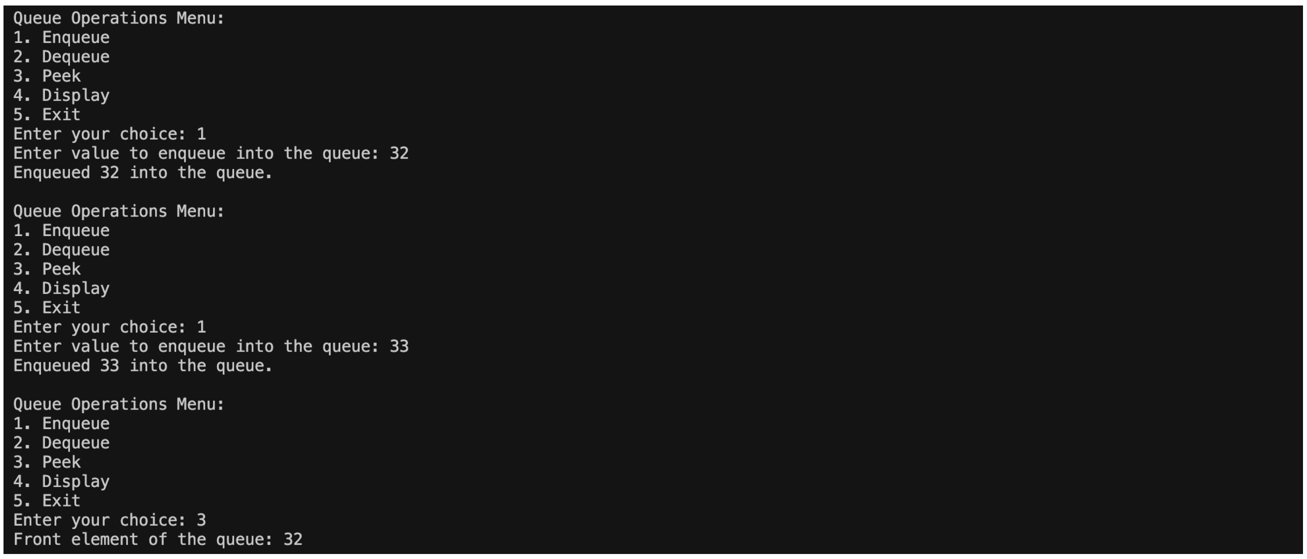
Code:

```
#include <iostream>

using namespace std;


class Node

{

public:

    int data;

    Node *next;


    Node(int value)

    {

        data = value;

        next = nullptr;
    }
};

class Queue
{
private:
    Node *front;
    Node *rear;

public:
    Queue()
    {
        front = nullptr;
        rear = nullptr;
    }

    ~Queue()
    {
        Node *current = front;
        Node *next;
        while (current != nullptr)
        {
            next = current->next;
            delete current;
            current = next;
        }
        front = nullptr;
        rear = nullptr;
    }

    bool is_empty()
    {
```

```
    return front == nullptr;
}

void enqueue(int value)
{
    Node *newNode = new Node(value);
    if (is_empty())
    {
        front = newNode;
        rear = newNode;
    }
    else
    {
        rear->next = newNode;
        rear = newNode;
    }
    cout << "Enqueued " << value << " into the queue." << endl;
}

int dequeue()
{
    if (is_empty())
    {
        cout << "Queue underflow. Cannot dequeue element." << endl;
        return -1;
    }
    int dequeued_element = front->data;
    Node *temp = front;
    front = front->next;
    delete temp;
    cout << "Dequeued " << dequeued_element << " from the queue." << endl;
    return dequeued_element;
    }

    int peek()
    {
        if (is_empty())
        {
            cout << "Queue is empty. No front element to peek." << endl;
            return -1;
        }
        return front->data;
    }

    void display()
    {
        if (is_empty())
        {
            cout << "Queue is empty." << endl;
            return;
        }
        cout << "Queue elements: ";
        Node *current = front;
        while (current != nullptr)
        {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};

int main()
{
```

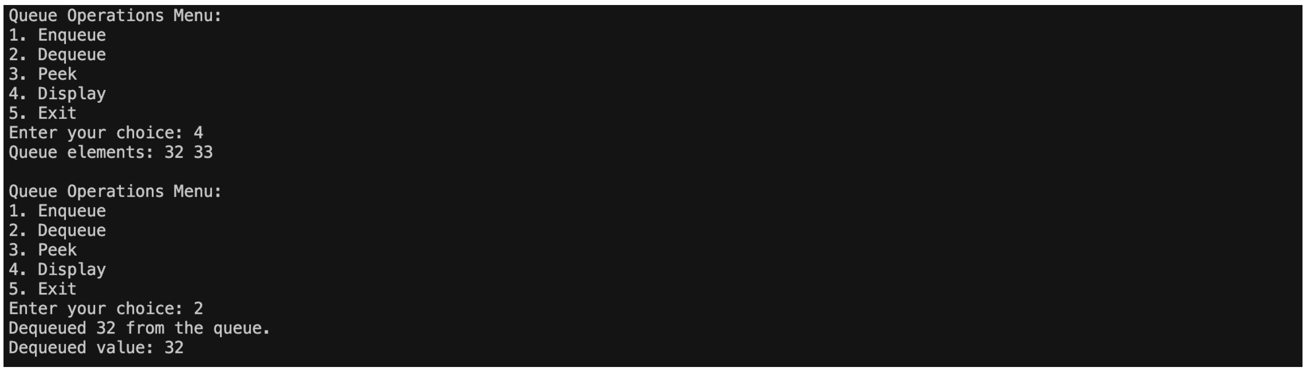
```
Queue queue;
int choice, value;

do
{
    cout << "\nQueue Operations Menu:" << endl;
    cout << "1. Enqueue" << endl;
    cout << "2. Dequeue" << endl;
    cout << "3. Peek" << endl;
    cout << "4. Display" << endl;
    cout << "5. Exit" << endl;
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice)
    {
    case 1:
        cout << "Enter value to enqueue into the queue: ";
        cin >> value;
        queue.enqueue(value);
        break;
    case 2:
        value = queue.dequeue();
        if (value != -1)
        {
            cout << "Dequeued value: " << value << endl;
        }
        break;
    case 3:
        value = queue.peek();
        if (value != -1)
        {
          cout << "Front element of the queue: " << value << endl;
        }
        break;
      case 4:
        queue.display();
        break;
      case 5:
        cout << "Exiting..." << endl;
        break;
      default:
        cout << "Invalid choice. Please enter a valid choice." << endl;
      }
  } while (choice != 5);

  return 0;
}
```

Output: (screenshot)

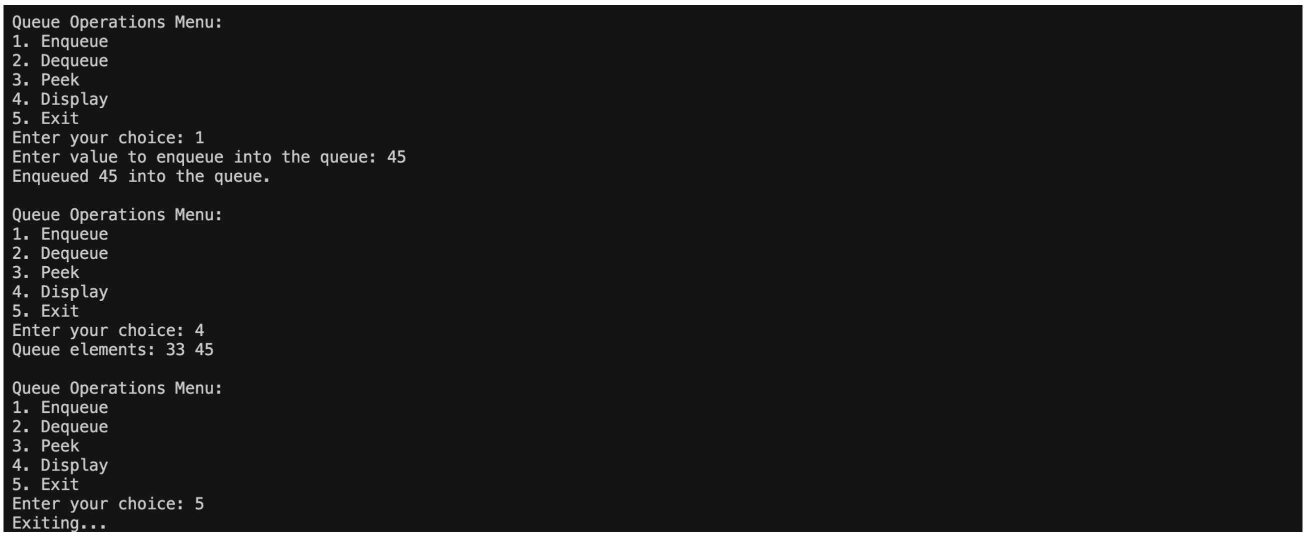


```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue into the queue: 32
Enqueued 32 into the queue.

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue into the queue: 33
Enqueued 33 into the queue.

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 3
Front element of the queue: 32
```

Test Case: Any two (screenshot)



```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Queue elements: 32 33

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 2
Dequeued 32 from the queue.
Dequeued value: 32
```



```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue into the queue: 45
Enqueued 45 into the queue.

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Queue elements: 33 45

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 5
Exiting...
```

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: Ashlin Lee George

Roll Number: 150096723011

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.

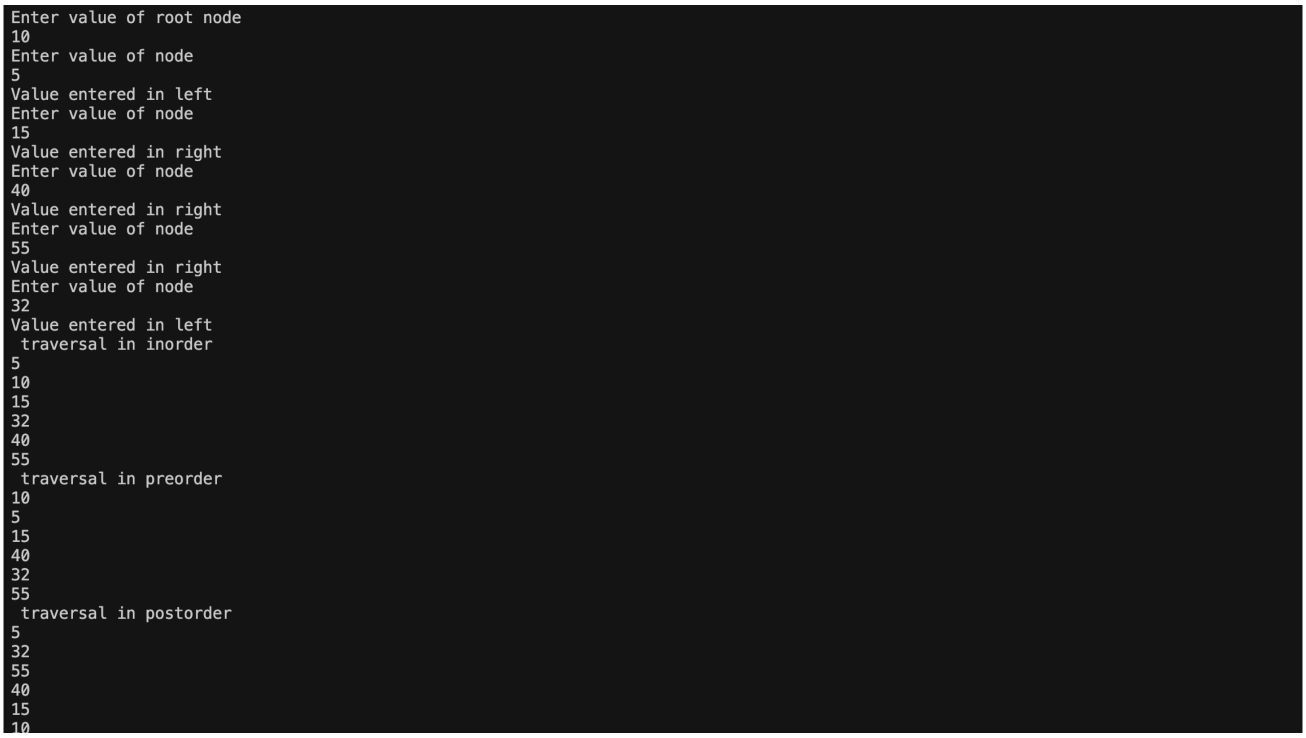
Code:

```
#include <iostream>

using namespace std;


struct nod

{

   nod *l, *r;

   int d;

} *r = NULL, *p = NULL, *np = NULL, *q;


void create()

{

   int v, c = 0;

   while (c < 6)

   {

      if (r == NULL)

      {

         r = new nod;
```

```
    cout << "enter value of root node\n";
    cin >> r->d;
    r->r = NULL;
    r->l = NULL;
}
else
{
    p = r;
    cout << "enter value of node\n";
    cin >> v;
    while (true)
    {
        if (v < p->d)
        {
            if (p->l == NULL)
            {
                p->l = new nod;
                p = p->l;
                p->d = v;
                p->l = NULL;
                p->r = NULL;
                cout << "value entered in left\n";
                break;
            }
            else if (p->l != NULL)
            {
                p = p->l;
            }
        }
        else if (v > p->d)
        {
            if (p->r == NULL)
```

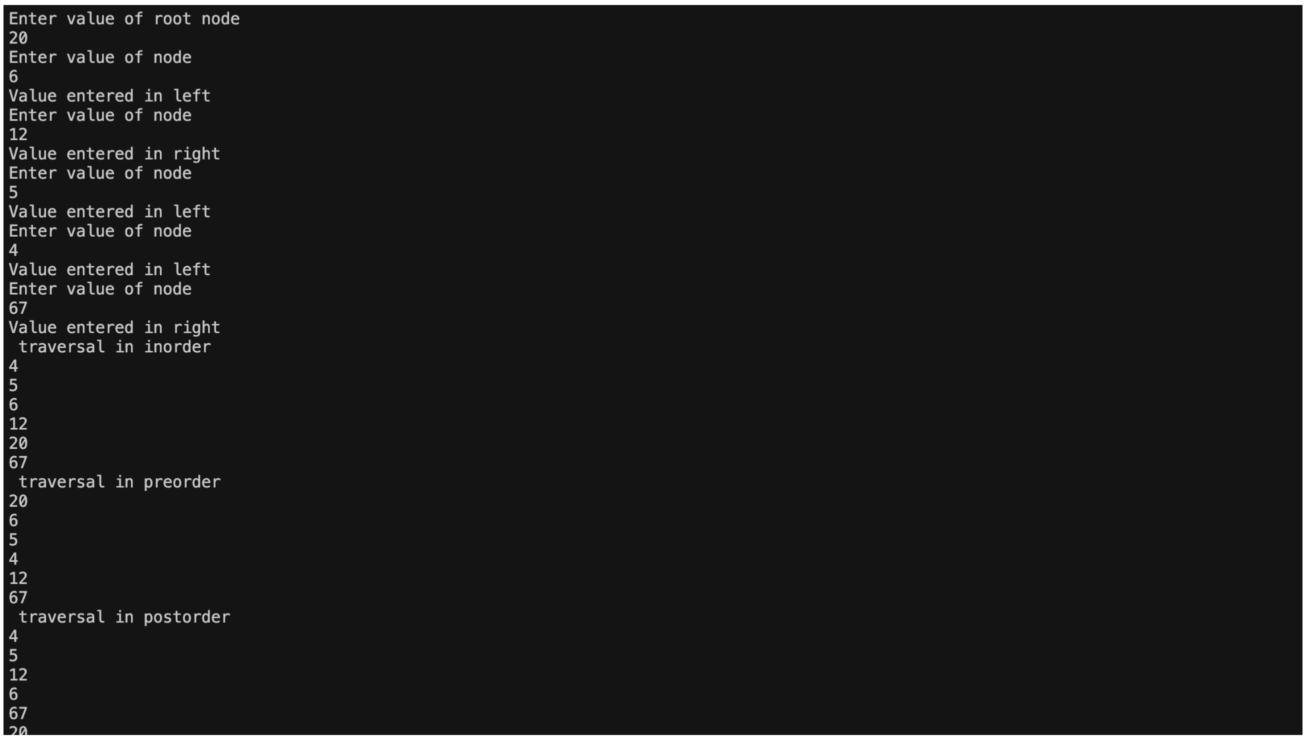
```
                    {
                        p->r = new nod;
                        p = p->r;
                        p->d = v;
                        p->l = NULL;
                        p->r = NULL;
                        cout << "value entered in right\n";
                        break;
                    }
                    else if (p->r != NULL)
                    {
                        p = p->r;
                    }
                }
            }
        }
        c++;
    }
}

void inorder(nod *p)
{
    if (p != NULL)
    {
        inorder(p->l);
        cout << p->d << endl;
        inorder(p->r);
    }
}

void preorder(nod *p)
{
    if (p != NULL)
    {
        cout << p->d << endl;
        preorder(p->l);
        preorder(p->r);
    }
}

void postorder(nod *p)
{
    if (p != NULL)
    {
        postorder(p->l);
        postorder(p->r);
        cout << p->d << endl;
    }
}

int main()
{
    create();
    cout << " traversal in inorder\n";
    inorder(r);
    cout << " traversal in preorder\n";
    preorder(r);
    cout << " traversal in postorder\n";
    postorder(r);
}
```

Output: (screenshot)



```
Enter value of root node
10
Enter value of node
5
Value entered in left
Enter value of node
15
Value entered in right
Enter value of node
40
Value entered in right
Enter value of node
55
Value entered in right
Enter value of node
32
Value entered in left
 traversal in inorder
5
10
15
32
40
55
 traversal in preorder
10
5
15
40
32
55
 traversal in postorder
5
32
55
40
15
10
```

Test Case: (screenshot)



```
Enter value of root node
20
Enter value of node
6
Value entered in left
Enter value of node
12
Value entered in right
Enter value of node
5
Value entered in left
Enter value of node
4
Value entered in left
Enter value of node
67
Value entered in right
 traversal in inorder
4
5
6
12
20
67
 traversal in preorder
20
6
5
4
12
67
 traversal in postorder
4
5
12
6
67
20
```

Conclusion:

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

Name of Student: Ashlin Lee George

Roll Number: 150096723011

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:

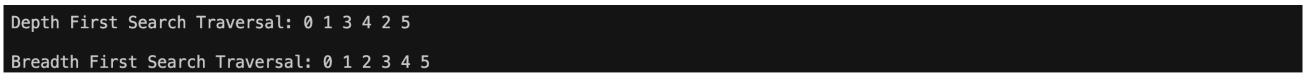
```
#include <iostream>

using namespace std;

#define MAX_VERTICES 100



class Graph

{

private:

   int vertices;

   int adjacency[MAX_VERTICES][MAX_VERTICES];


public:


   Graph(int V)

   {

      vertices = V;
```

```
    for (int i = 0; i < vertices; ++i)
    {
        for (int j = 0; j < vertices; ++j)
        {
            adjacency[i][j] = 0;
        }
    }
}

void addEdge(int u, int v)
{
    adjacency[u][v] = 1;
    adjacency[v][u] = 1;
}


void DFSUtil(int vertex, bool visited[])
{
    cout << vertex << " ";
    visited[vertex] = true;
    for (int i = 0; i < vertices; ++i)
    {
        if (adjacency[vertex][i] && !visited[i])
        {
            DFSUtil(i, visited);
        }
    }
}

void DFS(int start)
{
    bool visited[MAX_VERTICES] = {false};
```

```
        cout << "Depth First Search Traversal: ";
        DFSUtil(start, visited);
        cout << endl;
    }

    // Breadth First Search traversal starting from a given vertex
    void BFS(int start)
    {
        bool visited[MAX_VERTICES] = {false};
        int queue[MAX_VERTICES];
        int front = 0, rear = 0;

        cout << "Breadth First Search Traversal: ";
        visited[start] = true;
        queue[rear++] = start;

        while (front != rear)
        {
            int current = queue[front++];
            cout << current << " ";
            for (int i = 0; i < vertices; ++i)
            {
                if (adjacency[current][i] && !visited[i])
                {
                    visited[i] = true;
                    queue[rear++] = i;
                }
            }
        }
        cout << endl;
    }
};
```

```
int main()
{
    Graph g(6);

    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 4);
    g.addEdge(3, 4);
    g.addEdge(3, 5);
    g.addEdge(4, 5);

    g.DFS(0);
    g.BFS(0);

    return 0;
}
```

Output: (screenshot)



```
Depth First Search Traversal: 0 1 3 4 2 5
Breadth First Search Traversal: 0 1 2 3 4 5
```

Conclusion:

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

Name of Student: Ashlin Lee George

Roll Number: 150096723011

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).

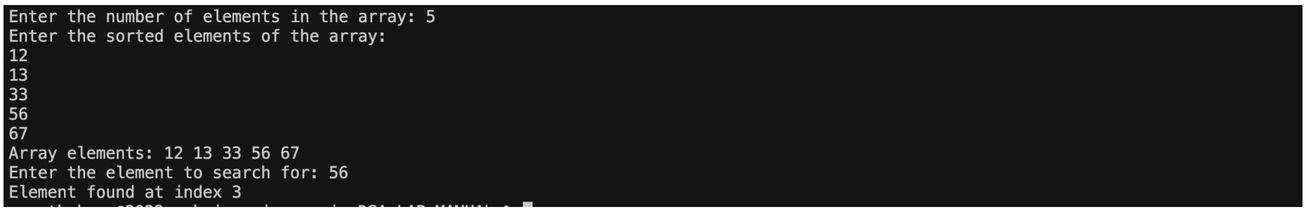
Code:

```
#include <iostream>

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;

  }
```

```
    return -1;
}

void displayArray(int arr[], int n)
{
    cout << "Array elements: ";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    int n;
    cout << "Enter the number of elements in the array: ";
    cin >> n;
    int arr[n];
    cout << "Enter the sorted elements of the array: ";
    for (int i = 0; i < n; ++i)
        cin >> arr[i];

    displayArray(arr, n);

    int target;
    cout << "Enter the element to search for: ";
    cin >> target;

    int result = binarySearch(arr, 0, n - 1, target);
    if (result == -1)
        cout << "Element not present in the array";
    else
```

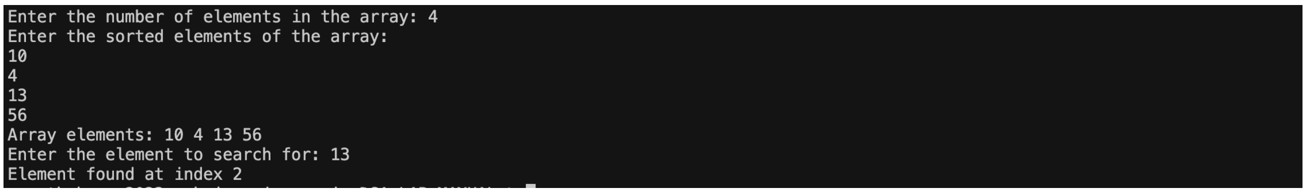
```
        cout << "Element found at index " << result;
    return 0;
}
```

Output: (screenshot)

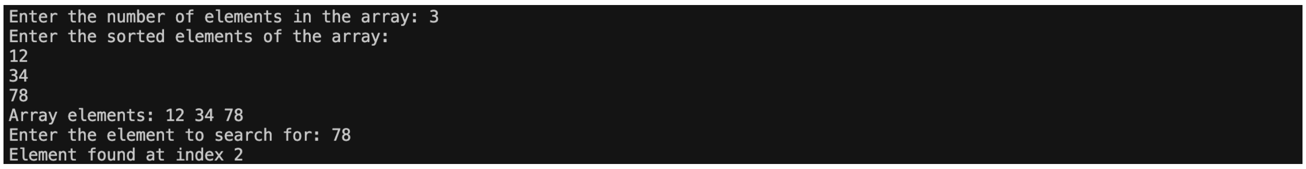


```
Enter the number of elements in the array: 5
Enter the sorted elements of the array:
12
13
33
56
67
Array elements: 12 13 33 56 67
Enter the element to search for: 56
Element found at index 3
```

Test Case: Any two (screenshot)



```
Enter the number of elements in the array: 4
Enter the sorted elements of the array:
10
4
13
56
Array elements: 10 4 13 56
Enter the element to search for: 13
Element found at index 2
```



```
Enter the number of elements in the array: 3
Enter the sorted elements of the array:
12
34
78
Array elements: 12 34 78
Enter the element to search for: 78
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: Ashlin Lee George

Roll Number: 150096723011

Experiment No: 14

Title:

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

Theory:

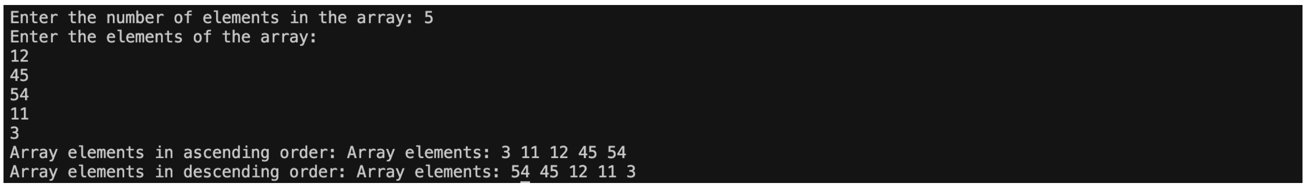
In Bubble Sort algorithm, we traverse from left and compare adjacent elements and the higher one is placed at right side. In this way, the largest element is moved to the rightmost end at first. This process is then continued to find the second largest and place it and so on until the data is sorted.

Code:

```
#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])

         {

            int temp = arr[j];

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

            arr[j + 1] = temp;

         }

      }

   }
```

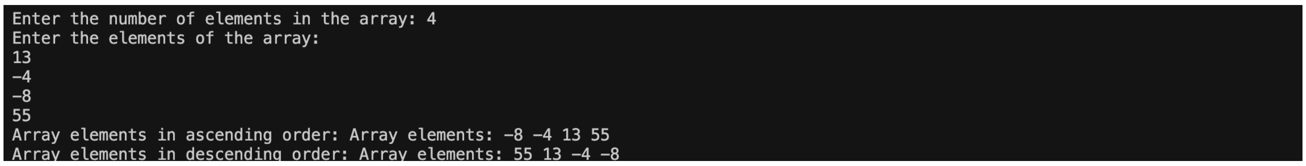
```cpp
}

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])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

void displayArray(int arr[], int n)
{
    cout << "Array elements: ";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    int n;
    cout << "Enter the number of elements in the array: ";
    cin >> n;
    int arr[n];
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; ++i)
        cin >> arr[i];

    bubbleSortAscending(arr, n);
    cout << "Array elements in ascending order: ";
    displayArray(arr, n);

    bubbleSortDescending(arr, n);
    cout << "Array elements in descending order: ";
    displayArray(arr, n);

    return 0;
}
```

Output: (screenshot)

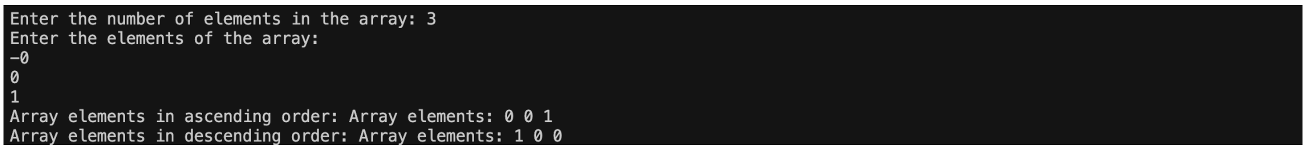


```
Enter the number of elements in the array: 5
Enter the elements of the array:
12
45
54
11
3
Array elements in ascending order: Array elements: 3 11 12 45 54
Array elements in descending order: Array elements: 54 45 12 11 3
```

Test Case: Any two (screenshot)



```
Enter the number of elements in the array: 4
Enter the elements of the array:
13
-4
-8
55
Array elements in ascending order: Array elements: -8 -4 13 55
Array elements in descending order: Array elements: 55 13 -4 -8
```



```
Enter the number of elements in the array: 3
Enter the elements of the array:
-0
0
1
Array elements in ascending order: Array elements: 0 0 1
Array elements in descending order: Array elements: 1 0 0
```

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.