



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

# **DATA STRUCTURES & ALGORITHMS PROGRAMMING LAB**



**Prepared by:**

Name of Student: Arpita Jadhav

Roll No: (150096723028)

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. Arpita Jadhav Roll  
No. 150096723028 Semester Two of B.Tech Computer Science & Engineering,  
ITM Skills University, Kharghar, Navi Mumbai , has completed the term work  
satisfactorily in subject Data Structures & Algorithms for the academic  
year 20 24 - 20 25 as prescribed in the curriculum.

Place: Mumbai

Date: 25-03-2024

**Subject I/C**

**HOD**

<b>Exp. No</b>	<b>List of Experiment</b>	<b>Date of Submission</b>	<b>Sign</b>
1	Implement Array and write a menu driven program to perform all the operation on array elements	25-03-2024	
2	Implement Stack ADT using array.	25-03-2024	
3	Convert an Infix expression to Postfix expression using stack ADT.	25-03-2024	
4	Evaluate Postfix Expression using Stack ADT.	25-03-2024	
5	Implement Linear Queue ADT using array.	25-03-2024	
6	Implement Circular Queue ADT using array.	25-03-2024	
7	Implement Singly Linked List ADT.	25-03-2024	
8	Implement Circular Linked List ADT.	25-03-2024	
9	Implement Stack ADT using Linked List	25-03-2024	
10	Implement Linear Queue ADT using Linked List	25-03-2024	
11	Implement Binary Search Tree ADT using Linked List.	25-03-2024	
12	Implement Graph Traversal techniques: a) Depth First Search b) Breadth First Search	25-03-2024	
13	Implement Binary Search algorithm to search an element in an array	25-03-2024	
14	Implement Bubble sort algorithm to sort elements of an array in ascending and descending order	25-03-2024	

Name of Student: Arpita Jadhav

Roll Number: 150096723028

Experiment No: 01

---

### Title:

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

### Theory:

The provided code introduces a robust class Array designed to streamline array operations, encompassing a wide array of functionalities including initialization, traversal, insertion, deletion, searching, sorting, and reversal. Within the main function, users are prompted to input the array's capacity and elements, thereafter gaining access to a user-friendly menu-driven interface for executing diverse operations seamlessly.

### Code:

```
#include <iostream>
using namespace std;
class Array {
private:
    int capacity, size;
    int *arr;
public:
    void init(int cap) {
        capacity = cap;
        arr = new int[capacity];
        size = 0;
    }
    void traverse() {
        cout << "Array elements: ";
        for (int i = 0; i < size; ++i) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
    void insertBeginning(int element) {
        if (size < capacity) {
            for (int i = size; i > 0; --i) {
                arr[i] = arr[i - 1];
            }
            arr[0] = element;
            size++;
            cout << "element inserted at beginning!" << endl;
        } else {
            cout << "array is full" << endl;
        }
    }
    void insertEnd(int element) {
        if (size < capacity) {
            arr[size++] = element;
            cout << "element inserted at end!" << endl;
        } else {
            cout << "array is full" << endl;
        }
    }
}
```

```

    }
}

void insertIndex(int element, int index) {
    if (index >= 0 && index <= size && size < capacity) {
        for (int i = size; i > index; --i) {
            arr[i] = arr[i - 1];
        }
        arr[index] = element;
        size++;
        cout << "element inserted at index " << index << endl;
    } else {
        cout << "invalid index or array is full" << endl;
    }
}

void deleteElement(int index) {
    if (index >= 0 && index < size) {
        for (int i = index; i < size - 1; ++i) {
            arr[i] = arr[i + 1];
        }
        size--;
        cout << "element deleted!" << endl;
    } else {
        cout << "invalid index" << endl;
    }
}

int search(int element) {
    for (int i = 0; i < size; ++i) {
        if (arr[i] == element) {
            return i;
        }
    }
    return -1;
}

void sort() {
    for (int i = 0; i < size - 1; ++i) {
        for (int j = 0; j < size - i - 1; ++j) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    cout << "array sorted!" << endl;
}

void reverse() {
    for (int i = 0; i < size / 2; ++i) {
        int temp = arr[i];
        arr[i] = arr[size - i - 1];
        arr[size - i - 1] = temp;
    }
    cout << "array reversed!" << endl;
}

};

int main() {
    Array arr;
    int capacity;
    cout << "enter capacity of array: ";
    cin >> capacity;
    arr.init(capacity);
    cout << "enter elements for array: " << endl;
    for (int i = 0; i < capacity; ++i) {
        int element;

```

```

        cout << "element " << i + 1 << ": ";
        cin >> element;
        arr.insertEnd(element);
    }

```

```

int choice, element, index;
do {
    cout << "1. traverse\n";
    cout << "2. insert at beginning\n";
    cout << "3. insert at end\n";
    cout << "4. insert at any index\n";
    cout << "5. delete\n";
    cout << "6. search\n";
    cout << "7. sort\n";
    cout << "8. reverse\n";
    cout << "9. exit\n";
    cout << "enter your choice: ";
    cin >> choice;
    switch (choice) {
        case 1:
            arr.traverse();
            break;
        case 2:
            cout << "enter element to insert at beginning: ";
            cin >> element;
            arr.insertBeginning(element);
            break;
        case 3:
            cout << "enter element to insert at end: ";
            cin >> element;
            arr.insertEnd(element);
            break;
        case 4:
            cout << "enter element to insert: ";
            cin >> element;
            cout << "enter index no. to insert at: ";
            cin >> index;
            arr.insertIndex(element, index);
            break;
        case 5:
            cout << "enter index of element to delete: ";
            cin >> index;
            arr.deleteElement(index);
            break;
        case 6:
            cout << "enter element to search: ";
            cin >> element;
            index = arr.search(element);
            if (index != -1) {
                cout << "element found at index " << index << endl;
            } else {
                cout << "element not found" << endl;
            }
            break;
        case 7:
            arr.sort();
            break;
        case 8:
            arr.reverse();
            break;
        case 9:
            cout << "exiting!\n";
            break;
    }
}

```

```

        default:
            cout << "invalid choice" << endl;
        }
    } while (choice != 9);
    return 0;
}

```

## Output: (screenshot)

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 1stcode.cpp -o 1stcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"1stcod
e
○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 1stcode.cpp -o 1stcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"1stcode
enter capacity of array: 4

```

## Test Case: Any two (screenshot)

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 1stcode.cpp -o 1stcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"1stcod
e
○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 1stcode.cpp -o 1stcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"1stcode
enter capacity of array: 4
enter elements for array:
element 1: 1
element inserted at end!
element 2: 2
element inserted at end!
element 3: 3
element inserted at end!
element 4: 4
element inserted at end!
1. traverse
2. insert at beginning
3. insert at end
4. insert at any index
5. delete
6. search
7. sort
8. reverse
9. exit
enter your choice: 2
enter element to insert at beginning: 1
array is full
1. traverse
2. insert at beginning
3. insert at end
4. insert at any index
5. delete
6. search
7. sort
8. reverse
9. exit
enter your choice: 1

```

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 1stcode.cpp -o 1stcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"1stcod
e
○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 1stcode.cpp -o 1stcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"1stcode
enter capacity of array: 3
enter elements for array:
element 1: 99
element inserted at end!
element 2: 89
element inserted at end!
element 3: 78
element inserted at end!
1. traverse
2. insert at beginning
3. insert at end
4. insert at any index
5. delete
6. search
7. sort
8. reverse
9. exit
enter your choice: 5
enter index of element to delete: 2
element deleted!
1. traverse

```

## Conclusion:

In conclusion, the provided code presents a robust implementation for array manipulation, encapsulating a diverse range of functionalities within the Array class. Through its intuitive menu-driven interface, users can effortlessly perform operations such as insertion, deletion, searching, sorting, and reversal, enhancing productivity and ease of use in array management tasks.

**Name of Student:** Arpita Jadhav

**Roll Number:** 150096723028

**Experiment No:** 02

---

## Title:

2. Implement Stack ADT using array in CPP.

## Theory:

The presented code introduces a foundational implementation of a stack data structure, facilitated by the Stack class. Equipped with functionalities for initialization, checking stack status, pushing and popping elements, and peeking at the top, the code empowers users with essential stack operations. Through a menu-driven interface within the main function, users gain convenient access to stack manipulation capabilities.

## Code:

```
#include <iostream>
using namespace std;
class Stack {
private:
    int capacity;
    int *arr;
    int top;
public:
    void init(int cap) {
        capacity = cap;
```



```

        arr = new int[capacity];
        top = -1;
    }
    bool isEmpty() {
        return top == -1;
    }
    bool isFull() {
        return top == capacity - 1;
    }
    void push(int element) {
        if (!isFull()) {
            arr[++top] = element;
            cout << element << " pushed in stack." << endl;
        } else {
            cout << "stack overflow" << endl;
        }
    }
    int pop() {
        if (!isEmpty()) {
            int element = arr[top--];
            cout << element << " popped from stack" << endl;
            return element;
        } else {
            cout << "stack underflow" << endl;
            return -1;
        }
    }
    int peek() {
        if (!isEmpty()) {
            return arr[top];
        } else {
            cout << "stack is empty" << endl;
            return -1;
        }
    }
};

int main() {
    Stack stack;
    int capacity;
    cout << "enter capacity of stack: ";
    cin >> capacity;
    stack.init(capacity);
    int num_elements;
    cout << "how many elements do you want to push in stack? ";
    cin >> num_elements;
    int element;
    for (int i = 0; i < num_elements; ++i) {
        cout << "enter element " << i + 1 << ": ";
        cin >> element;
        stack.push(element);
    }
    int choice;
    do {
        cout << "\n1. push\n";
        cout << "2. pop\n";
        cout << "3. peek\n";
        cout << "4. exit\n";
        cout << "enter your choice: ";
    } while (choice < 5);
}

```

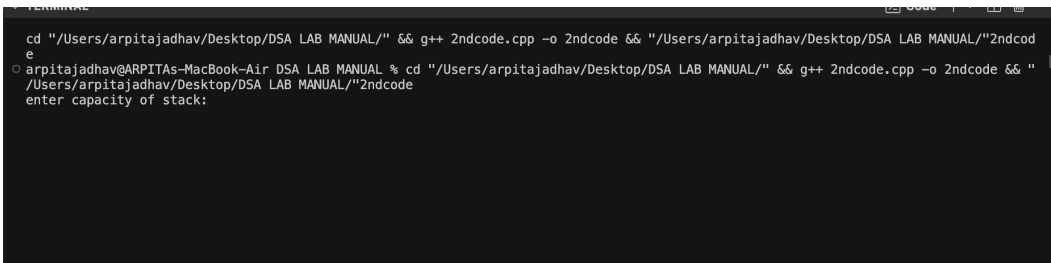
```

        cin >> choice;
        switch (choice) {
            case 1:
                cout << "enter element to push in stack: ";
                cin >> element;
                stack.push(element);
                break;
            case 2:
                stack.pop();
                break;
            case 3:
                element = stack.peek();
                if (element != -1) {
                    cout << "top element of stack: " << element << endl;
                }
                break;
            case 4:
                cout << "exited!\n";
                break;
            default:
                cout << "invalid choice" << endl;
        }
    }
    while (choice != 4);

    return 0;
}

```

**Output: (screenshot)**



```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 2ndcode.cpp -o 2ndcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"2ndcode
arpitajadhav@ARPITAS-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 2ndcode.cpp -o 2ndcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"2ndcode
enter capacity of stack:

```

**Test Case: Any two (screenshot)**



```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 2ndcode.cpp -o 2ndcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"2ndcode
arpitajadhav@ARPITAS-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 2ndcode.cpp -o 2ndcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"2ndcode
enter capacity of stack: 4
how many elements do you want to push in stack? 2
enter element 1: 1
1 pushed in stack.
enter element 2: 2
2 pushed in stack.

1. push
2. pop
3. peek
4. exit
enter your choice: 2
2 popped from stack

```

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 2ndcode.cpp -o 2ndcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"2ndcode
e
○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 2ndcode.cpp -o 2ndcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"2ndcode
enter capacity of stack: 3
how many elements do you want to push in stack? 1
enter element 1: 2
2 pushed in stack.

1. push
2. pop
3. peek
4. exit
enter your choice: 3
top element of stack: 2

```

## Conclusion:

In conclusion, the code lays a solid foundation for stack manipulation, offering essential functionalities encapsulated within the Stack class. Through its user-centric menu interface, users can seamlessly interact with the stack, performing operations such as pushing, popping, and peeking with ease. This implementation serves as a fundamental tool for stack-based applications, balancing simplicity with functionality.

**Name of Student:** Arpita Jadhav

**Roll Number:** 150096723028

**Experiment No:** 03

---

## Title:

3. Convert an Infix expression to Postfix expression using stack ADT.

## Theory:

The provided code employs a stack-based algorithm to convert infix expressions to postfix expressions. It solicits an infix expression from the user, then meticulously processes each character, accounting for operator precedence and parentheses to generate the postfix equivalent.

## Code:

```

#include <iostream>
#include <stack>
using namespace std;
int main() {
    stack<char> s;
    string infix, postfix;
    cout << "enter an infix expression: ";
    getline(cin, infix);
    for (char c : infix) {
        if (isalnum(c)) {

```

```

        postfix += c;
    } else if (c == '(') {
        s.push(c);
    } else if (c == ')') {
        while (s.top() != '(') {
            postfix += s.top(); s.pop();
        }
        s.pop();
    } else {
        while (!s.empty() && s.top() != '(' && ((c == '+' || c == '-' || c == '*' || c == '/') ? 1 : 2) <= ((s.top() == '+' || s.top() == '-') ? 1 : 2)) {
            postfix += s.top(); s.pop();
        }
        s.push(c);
    }
}

while (!s.empty()) {
    postfix += s.top(); s.pop();
}

cout << "postfix expression: " << postfix << endl;
return 0;
}

```

## Output: (screenshot)

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 3rdcode.cpp -o 3rdcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"3rdcode
e
arpitajadhav@ARPITAS-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 3rdcode.cpp -o 3rdcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"3rdcode
3rdcode.cpp:18:17: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
    for (char c : infix)
                ^
1 warning generated.
enter an infix expression:

```

## Test Case: Any two (screenshot)

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 3rdcode.cpp -o 3rdcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"3rdcode
e
arpitajadhav@ARPITAS-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 3rdcode.cpp -o 3rdcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"3rdcode
3rdcode.cpp:18:17: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
    for (char c : infix)
                ^
1 warning generated.
enter an infix expression: (1+2)*5-4
postfix expression: 12+5*4-
arpitajadhav@ARPITAS-MacBook-Air DSA LAB MANUAL %

```

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 3rdcode.cpp -o 3rdcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"3rdcode
e
arpitajadhav@ARPITAS-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 3rdcode.cpp -o 3rdcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"3rdcode
3rdcode.cpp:18:17: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
    for (char c : infix)
                ^
1 warning generated.
enter an infix expression: ((9/2)*4-1)
postfix expression: 92/4*-1-
arpitajadhav@ARPITAS-MacBook-Air DSA LAB MANUAL %

```

## Conclusion:

In conclusion, the code adeptly transforms infix expressions to postfix format through a stack-centric methodology. By meticulously managing operators and parentheses while considering precedence rules, it ensures accuracy in the conversion process. This implementation stands as a reliable tool for parsing and evaluating expressions across diverse applications.

**Name of Student:** Arpita Jadhav

**Roll Number:** 150096723028

**Experiment No:** 04

---

## Title:

4. Evaluate Postfix Expression using Stack ADT.

## Theory:

The presented code proficiently evaluates postfix expressions through an iterative process, meticulously managing operands and operators using a stack-based methodology. It adeptly pushes operands onto the stack and performs arithmetic operations when encountering operators, ensuring accurate expression evaluation.

## Code:

```
#include <iostream>
#include <stack>
using namespace std;
int evaluate(const string& postfix) {
    stack<int> s;
    for (char c : postfix) {
        if (isdigit(c)) {
            s.push(c - '0');
        } else {
            int operand2 = s.top(); s.pop();
            int operand1 = s.top(); s.pop();
            switch(c) {
```

```

        case '+': s.push(operand1 + operand2);
        break;
        case '-': s.push(operand1 - operand2);
        break;
        case '*': s.push(operand1 * operand2);
        break;
        case '/': s.push(operand1 / operand2);
        break;
    }
}
return s.top();
}
int main() {
    string postfixexp;
    cout << "enter postfix expression: ";
    getline(cin, postfixexp);
    cout << "result: " << evaluate(postfixexp) << endl;
    return 0;
}

```

**Output: (screenshot)**

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 4thcode.cpp -o 4thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"4thcode
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 4thcode.cpp -o 4thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"4thcode
4thcode.cpp:12:17: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
    for (char c : postfix)
        ^
1 warning generated.
enter postfix expression:

```

**Test Case: Any two (screenshot)**

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 4thcode.cpp -o 4thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"4thcode
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 4thcode.cpp -o 4thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"4thcode
4thcode.cpp:12:17: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
    for (char c : postfix)
        ^
1 warning generated.
enter postfix expression: 23/3-4
result: 4
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL %

```

```
cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 4thcode.cpp -o 4thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"4thcode
arpitajadhav@ARPITAS-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 4thcode.cpp -o 4thcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"4thcode
4thcode.cpp:12:17: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
    for (char c : postfix)
                  ^
1 warning generated.
enter postfix expression: 90*4+1
result: 1
arpitajadhav@ARPITAS-MacBook-Air DSA LAB MANUAL %
```

## Conclusion:

In conclusion, the code demonstrates a robust approach to postfix expression evaluation, leveraging stack operations for efficient computation. By meticulously handling arithmetic operations and operand manipulation, it provides a reliable mechanism for expression evaluation, applicable across diverse mathematical and computational domains.

**Name of Student:** Arpita Jadhav

**Roll Number:** 150096723028

**Experiment No:** 05

---

## Title:

5. Implement Linear Queue ADT using array.

## Theory:

The provided code illustrates the implementation of a queue data structure utilizing an array. It furnishes functionalities for enqueueing elements into the queue, dequeuing elements from it, and displaying the current queue elements. A menu-driven interface in the main function facilitates user interaction with the queue operations.

## Code:

```
#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;
    }
    bool isEmpty() {
        return (front == -1 && rear == -1);
    }
    bool isFull() {
        return (rear == MAX_SIZE - 1);
    }
    void enqueue(int data) {
        if (isFull()) {
            cout << "queue is full" << endl;
            return;
        } else if (isEmpty()) {
            front = rear = 0;
        } else {
            rear++;
        }
        arr[rear] = data;
        cout << data << " enqueued to queue" << endl;
    }
    void dequeue() {
        if (isEmpty()) {
            cout << "queue is empty" << endl;
            return;
        } else if (front == rear) {
            cout << arr[front] << " dequeued from queue" << endl;
            front = rear = -1;
        } else {
            cout << arr[front] << " dequeued from queue." << endl;
            front++;
        }
    }
    void display() {
        if (isEmpty()) {
            cout << "queue is empty" << endl;
            return;
        }
        cout << "elements in the queue: ";
        for (int i = front; i <= rear; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    Queue q;
    int choice, data;
    do {
        cout << "\n1. enqueue\n";
        cout << "2. dequeue\n";
        cout << "3. display\n";
        cout << "4. exit\n";
        cout << "choose an option : ";
    } while (choice != 4);
}

```



```

        cin >> choice;
        switch(choice) {
            case 1:
                cout << "enter elements to enqueue: ";
                cin >> data;
                q.enqueue(data);
                break;
            case 2:
                q.dequeue();
                break;
            case 3:
                q.display();
                break;
            case 4:
                cout << "exited!" << endl;
                break;
            default:
                cout << "invalid choice." << endl;
        }
    }
    while (choice != 4);
    return 0;
}

```

## Output: (screenshot)

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 5thcode.cpp -o 5thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"5thcode
○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 5thcode.cpp -o 5thcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"5thcode

1. enqueue
2. dequeue
3. display
4. exit
choose an option :

```

## Test Case: Any two (screenshot)

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 5thcode.cpp -o 5thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"5thcode
○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 5thcode.cpp -o 5thcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"5thcode

1. enqueue
2. dequeue
3. display
4. exit
choose an option : 1
enter elements to enqueue: 23
23 enqueued to queue

```

```
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 5thcode.cpp -o 5thcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"5thcode
1. enqueue
2. dequeue
3. display
4. exit
choose an option : 1
enter elements to enqueue: 23
23 enqueued to queue
1. enqueue
2. dequeue
3. display
4. exit
choose an option : 2
23 dequeued from queue
1. enqueue
2. dequeue
3. display
4. exit
choose an option : █
```

## Conclusion:

In conclusion, the code delivers a fundamental yet effective implementation of a queue using an array-based approach. It encompasses essential operations like enqueue, dequeue, and display, along with a user-friendly menu interface for seamless interaction. This code serves as a valuable foundation for understanding and implementing queue data structures.

**Name of Student:** Arpita Jadhav

**Roll Number:** 150096723028

**Experiment No:** 06

---

**Title:**

6. Implement Circular Queue ADT using array.

## Theory:

The provided code showcases an efficient implementation of a queue data structure utilizing an array with circular buffering. It offers functionalities for enqueueing and dequeuing elements from the queue while ensuring optimized space usage through circular buffering. Additionally, the code enables users to display the elements currently in the queue via a user-friendly menu interface.

## Code:

```
#include <iostream>
using namespace std;
const int MAX_SIZE = 100;
class Queue {
private:
    int front, rear;
    int arr[MAX_SIZE];
public:
    Queue() {
        front = rear = -1;
    }
    bool isEmpty() {
        return (front == -1);
    }
    bool isFull() {
        return ((front == 0 && rear == MAX_SIZE - 1) || (front == rear + 1));
    }
    void enqueue(int data) {
        if (isFull()) {
            cout << "queue is full" << endl;
            return;
        } else if (isEmpty()) {
            front = rear = 0;
        } else if (rear == MAX_SIZE - 1) {
            rear = 0;
        } else {
            rear++;
        }
        arr[rear] = data;
        cout << data << " enqueued to queue" << endl;
    }
    void dequeue() {
        if (isEmpty()) {
            cout << "queue is empty" << endl;
            return;
        } else if (front == rear) {
            cout << arr[front] << " dequeued from queue" << endl;
            front = rear = -1;
        } else if (front == MAX_SIZE - 1) {
            cout << arr[front] << " dequeued from queue" << endl;
            front = 0;
        } else {
            cout << arr[front] << " dequeued from queue" << endl;
            front++;
        }
    }
    void display() {
        if (isEmpty()) {
            cout << "queue is empty" << endl;
            return;
        }
    }
};
```

```

    }
    cout << "elements in queue: ";
    if (rear >= front) {
        for (int i = front; i <= rear; i++) {
            cout << arr[i] << " ";
        }
    } else {
        for (int i = front; i < MAX_SIZE; i++) {
            cout << arr[i] << " ";
        }
        for (int i = 0; i <= rear; i++) {
            cout << arr[i] << " ";
        }
    }
    cout << endl;
}
};

int main() {
    Queue q;
    int choice, data;
    do {
        cout << "\n1. enqueue\n";
        cout << "2. dequeue\n";
        cout << "3. display\n";
        cout << "4. exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch(choice) {
            case 1:
                cout << "enter elements to enqueue: ";
                cin >> data;
                q.enqueue(data);
                break;
            case 2:
                q.dequeue();
                break;
            case 3:
                q.display();
                break;
            case 4:
                cout << "exited!" << endl;
                break;
            default:
                cout << "invalid choice" << endl;
        }
    } while (choice != 4);
    return 0;
}

```

## Output: (screenshot)

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 6thcode.cpp -o 6thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"6t
e
○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 6thcode.cpp -o 6thcode
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"6thcode

1. enqueue
2. dequeue
3. display
4. exit
Enter your choice:

```

## Test Case: Any two (screenshot)

```
cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 6thcode.cpp -o 6thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"6thcode
○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 6thcode.cpp -o 6thcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"6thcode

1. enqueue
2. dequeue
3. display
4. exit
Enter your choice: 3
queue is empty

1. enqueue
2. dequeue
3. display
4. exit
Enter your choice: 1
enter elements to enqueue: 24
24 enqueued to queue
```

```
1. enqueue
2. dequeue
3. display
4. exit
Enter your choice: 3
elements in queue: 24

1. enqueue
2. dequeue
3. display
4. exit
Enter your choice: $
```

## Conclusion:

In conclusion, the code presents an optimized approach to implementing a queue using an array with circular buffering. By incorporating essential operations like enqueue, dequeue, and display, along with a user-friendly menu interface, it provides an effective solution for managing elements in a queue. This code demonstrates efficient space utilization and ease of interaction with queue functionalities.

Name of Student: Arpita Jadhav

Roll Number: 150096723028

## Experiment No: 07

---

### Title:

7. Implement Singly Linked List ADT.

### Theory:

This code presents a fundamental implementation of a singly linked list data structure, featuring operations to append elements to the list, display its contents, and clear it. Utilizing a Node class for individual elements and a SinglyList class for managing list operations, the code offers a structured approach to list management. Moreover, the main function includes a menu-driven interface, enhancing user interaction with the list functionalities.

### Code:

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

class singlylist {
private:
    Node* head;
    Node* tail;
public:
    singlylist() {
        head = nullptr;
        tail = nullptr;
    }
    void append(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
        cout << value << " appended to the list." << endl;
    }
    void display() {
        if (head == nullptr) {
            cout << "list is empty" << endl;
        }
    }
};
```

```

        return;
    }
    cout << "elements in list: ";
    Node* current = head;
    while (current != nullptr) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

void clear() {
    while (head != nullptr) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
    tail = nullptr;
    cout << "list cleared." << endl;
}

};

int main() {
    singlylist list;
    int choice, data;
    do {
        cout << "\n1. append\n";
        cout << "2. display\n";
        cout << "3. clear\n";
        cout << "4. exit\n";
        cout << "enter your choice: ";
        cin >> choice;
        switch(choice) {
            case 1:
                cout << "enter element to append: ";
                cin >> data;
                list.append(data);
                break;
            case 2:
                list.display();
                break;
            case 3:
                list.clear();
                break;
            case 4:
                cout << "exited!" << endl;
                break;
            default:
                cout << "invalid choice" << endl;
        }
    } while (choice != 4);
    return 0;
}

```

## Output: (screenshot)

```
cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 7thcode.cpp -o 7thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"7thcode
○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 7thcode.cpp -o 7thcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"7thcode

1. append
2. display
3. clear
4. exit
enter your choice:
```

## Test Case: Any two (screenshot)

```
cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 7thcode.cpp -o 7thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"7thcode
○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 7thcode.cpp -o 7thcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"7thcode

1. append
2. display
3. clear
4. exit
enter your choice: 1
enter element to append: 12
12 appended to the list.

1. append
2. display
3. clear
4. exit
enter your choice: █
```

```
1. append
2. display
3. clear
4. exit
enter your choice: 2
elements in list: 12

1. append
2. display
3. clear
4. exit
enter your choice: █
```

## Conclusion:

In conclusion, the code provides a straightforward solution for managing a singly linked list, offering essential operations such as appending elements, displaying the list contents, and clearing the list. With its user-friendly menu interface, users can easily



navigate and utilize the list functionalities. This implementation serves as a foundational example for understanding and working with singly linked lists in programming.

**Name of Student:** Arpita Jadhav

**Roll Number:** 150096723028

**Experiment No:** 08

---

**Title:**

8. Implement Circular Linked List ADT.

**Theory:**

This code exemplifies the implementation of a circular singly linked list, offering functionalities to append elements, display the list contents, and clear the list. Leveraging a Node class to represent individual elements and a CircularList class to manage list operations, the code ensures efficient circular linking for seamless traversal. Additionally, the main function features a menu-driven interface, enhancing user interaction with the list functionalities.

**Code:**

```
#include <iostream>
using namespace std;
class Node {
public:
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }
};
class circularlist {
private:
    Node* head;
public:
    circularlist() {
        head = nullptr;
    }
    void append(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = newNode;
```

```

        head->next = head;
    } else {
        Node* current = head;
        while (current->next != head) {
            current = current->next;
        }
        current->next = newNode;
        newNode->next = head;
    }
    cout << value << " appended to the list." << endl;
}

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

void clear() {
    if (head == nullptr) {
        cout << "list is empty." << endl;
        return;
    }
    Node* current = head;
    while (current->next != head) {
        Node* temp = current;
        current = current->next;
        delete temp;
    }
    delete current;
    head = nullptr;
    cout << "list cleared." << endl;
}

};

int main() {
    circularlist list;
    int choice, data;
    do {
        cout << "\n1. append\n";
        cout << "2. display\n";
        cout << "3. clear\n";
        cout << "4. exit\n";
        cout << "enter your choice: ";
        cin >> choice;
        switch(choice) {
            case 1:
                cout << "enter element to append: ";
                cin >> data;
                list.append(data);
                break;
            case 2:

```

```

        list.display();
        break;
    case 3:
        list.clear();
        break;
    case 4:
        cout << "exited!" << endl;
        break;
    default:
        cout << "invalid choice" << endl;
    }
} while (choice != 4);
return 0;
}

```

## Output: (screenshot)

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 8thcode.cpp -o 8thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"8thcode
e
○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 8thcode.cpp -o 8thcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"8thcode

1. append
2. display
3. clear
4. exit
enter your choice:

```

## Test Case: Any two (screenshot)

```

○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 8thcode.cpp -o 8thcode
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"8thcode

1. append
2. display
3. clear
4. exit
enter your choice: 1
enter element to append: 13
13 appended to the list.

1. append
2. display
3. clear
4. exit
enter your choice: 1
enter element to append: 34
34 appended to the list.

```

```

1. append
2. display
3. clear
4. exit
enter your choice: 2
elements in list: 13 34

1. append
2. display
3. clear
4. exit
enter your choice:

```

## Conclusion:

In conclusion, the code provides a robust implementation of a circular singly linked list, enabling users to append elements, display the list contents, and clear the list efficiently. By employing circular linking, the code ensures that the last node points back to the head, creating a circular structure. The user-friendly menu interface simplifies list operations, making it convenient for users to interact with the list.

Name of Student: Arpita Jadhav

Roll Number: 150096723028

Experiment No: 09

---

## Title:

9. Implement Stack ADT using Linked List

## Theory:

This code showcases the implementation of a stack data structure using a singly linked list, offering functionalities like pushing elements onto the stack, popping elements from it, displaying the stack contents, and checking if the stack is empty. By leveraging dynamic memory allocation to manage nodes, the code ensures efficient memory usage. Additionally, the main function features a menu-driven interface, enabling users to interact seamlessly with the stack operations.

## Code:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
};
class Stack {
private:
    Node* top;
public:
    Stack() {
        top = nullptr;
    }
    void push(int value) {
        Node* newNode = new Node;
```

```

        newNode->data = value;
        newNode->next = top;
        top = newNode;
    }

    int pop() {
        if (isEmpty()) {
            cout << "stack is empty\n";
            return -1;
        }
        int poppedvalue = top->data;
        Node* temp = top;
        top = top->next;
        delete temp;
        return poppedvalue;
    }

    bool isEmpty() {
        return top == nullptr;
    }

    void display() {
        if (isEmpty()) {
            cout << "stack is empty\n";
            return;
        }
        cout << "stack: ";
        Node* current = top;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};

int main() {
    Stack stack;
    char choice;
    int value;
    do {
        cout << "1. push\n";
        cout << "2. pop\n";
        cout << "3. display\n";
        cout << "4. exit\n";
        cout << "enter your choice: ";
        cin >> choice;

        switch(choice) {
            case '1':
                cout << "enter element to push: ";
                cin >> value;
                stack.push(value);

```

```

                break;
            case '2':
                if (!stack.isEmpty())
                    cout << "popped element: " << stack.pop() <<
endl;
                break;
            case '3':
                stack.display();
                break;
            case '4':
                cout << "exited!\n";
                break;
            default:
                cout << "invalid choice\n";
        }
    } while(choice != '4');
    return 0;
}

```

**Output: (screenshot)**

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 9thcode.cpp -o 9thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"9thcod
e
○ arpitajadhav@ARPITAS-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 9thcode.cpp -o 9thcode && "
/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"9thcode
1. push
2. pop
3. display
4. exit
enter your choice:

```

**Test Case: Any two (screenshot)**

```

1. push
2. pop
3. display
4. exit
enter your choice: 1
enter element to push: 99
1. push
2. pop
3. display
4. exit
enter your choice: 1
enter element to push: 34
1. push
2. pop
3. display
4. exit
enter your choice: 3
stack: 34 99
1. push
2. pop
3. display
4. exit
enter your choice: █

```

```
4. exit
enter your choice: 3
stack: 34 99
1. push
2. pop
3. display
4. exit
enter your choice: 2
popped element: 34
1. push
2. pop
3. display
4. exit
enter your choice: █
```

## Conclusion:

In conclusion, the code provides a versatile implementation of a stack using a singly linked list, encompassing essential operations such as push, pop, display, and stack empty check. By employing dynamic memory allocation, the code optimizes memory usage, enhancing performance. The intuitive menu interface enhances user experience, making it effortless to execute stack operations.

**Name of Student:** Arpita Jadhav

**Roll Number:** 150096723028

**Experiment No: 10**

---

## Title:

10. Implement Linear Queue ADT using Linked List

## Theory:

This code illustrates the implementation of a queue data structure using a singly linked list. It offers functionalities like enqueueing elements into the queue, dequeuing elements from it, displaying the queue contents, and checking if the queue is empty. By employing dynamic memory allocation for node management, the code ensures efficient memory usage. Additionally, the main function incorporates a user-friendly menu-driven interface, enabling users to interact effortlessly with the queue operations.

## Code:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
};
class Queue {
private:
```

```

    Node* front;
    Node* rear;
public:
    Queue() {
        front = nullptr;
        rear = nullptr;
    }
    void enqueue(int value) {
        Node* newNode = new Node;
        newNode->data = value;
        newNode->next = nullptr;
        if (isEmpty()) {
            front = rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
    }
    int dequeue() {
        if (isEmpty()) {
            cout << "queue is empty\n";
            return -1;
        }
        int dequeuedvalue = front->data;
        Node* temp = front;
        front = front->next;
        if (front == nullptr) {
            rear = nullptr;
        }
        delete temp;
        return dequeuedvalue;
    }
    bool isEmpty() {
        return front == nullptr;
    }
    void display() {
        if (isEmpty()) {
            cout << "queue is empty\n";
            return;
        }
        cout << "queue: ";
        Node* current = front;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};

int main() {
    Queue queue;
    char choice;
    int value;
    do {
        cout << "1. enqueue\n";
        cout << "2. dequeue\n";
        cout << "3. display\n";
        cout << "4. exit\n";
    }

```



```

    cout << "enter your choice: ";
    cin >> choice;
    switch(choice) {
        case '1':
            cout << "enter element to enqueue: ";
            cin >> value;
            queue.enqueue(value);
            break;
        case '2':
            if (!queue.isEmpty())
                cout << "dequeued value: " << queue.dequeue() << endl;
            break;
        case '3':
            queue.display();
            break;
        case '4':
            cout << "exited!\n";
            break;
        default:
            cout << "invalid choice\n";
    }
}
while(choice != '4');
return 0;
}

```

## Output: (screenshot)

```

cd /Users/arpitajadhav/Desktop/DSA LAB MANUAL/ && g++ 10thcode.cpp -o 10thcode && /Users/arpitajadhav/Desktop/DSA LAB MANUAL/10thcode
○ arpitajadhav@ARPITAS-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 10thcode.cpp -o 10thcode &&
"/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"10thcode
1. enqueue
2. dequeue
3. display
4. exit
enter your choice:

```

## Test Case: Any two (screenshot)

```

1. enqueue
2. dequeue
3. display
4. exit
enter your choice: 1
enter element to enqueue: 33
1. enqueue
2. dequeue
3. display
4. exit
enter your choice: 1
enter element to enqueue: 44
1. enqueue
2. dequeue
3. display
4. exit

```

```
enter element to enqueue: 44
1. enqueue
2. dequeue
3. display
4. exit
enter your choice: 2
dequeued value: 33
1. enqueue
2. dequeue
3. display
4. exit
enter your choice: █
```

## Conclusion:

In conclusion, the code provides a versatile implementation of a queue using a singly linked list, encompassing essential operations such as enqueue, dequeue, display, and queue empty check. The dynamic memory allocation optimizes memory usage, enhancing performance. The intuitive menu interface enhances user experience, making it straightforward to execute queue operations.

**Name of Student:** Arpita Jadhav

**Roll Number:** 150096723028

**Experiment No:** 11

---

## Title:

11. Implement Binary Search Tree ADT using Linked List.

## Theory:

This code demonstrates a binary search tree (BST) data structure implementation, featuring functionalities like element insertion, element search, and displaying elements in sorted order. Leveraging a recursive approach for insertion and traversal, the code ensures efficient tree operations. Additionally, the main function incorporates a user-friendly menu-driven interface, enabling seamless interaction with the BST functionalities.

## Code:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
```

```

    Node* left;
    Node* right;
};

class bst {
private:
    Node* root;
    Node* createnode(int value) {
        Node* newNode = new Node;
        newNode->data = value;
        newNode->left = nullptr;
        newNode->right = nullptr;
        return newNode;
    }
    Node* insert(Node* root, int value) {
        if (root == nullptr) {
            return createnode(value);
        }
        if (value < root->data) {
            root->left = insert(root->left, value);
        } else if (value > root->data) {
            root->right = insert(root->right, value);
        }
        return root;
    }
    void traversal(Node* root) {
        if (root != nullptr) {
            traversal(root->left);
            cout << root->data << " ";
            traversal(root->right);
        }
    }
public:
    bst() {
        root = nullptr;
    }
    void insert(int value) {
        root = insert(root, value);
    }
    bool search(int value) {
        Node* current = root;
        while (current != nullptr) {
            if (value == current->data) {
                return true;
            } else if (value < current->data) {
                current = current->left;
            } else {
                current = current->right;
            }
        }
        return false;
    }
    void display() {
        if (root == nullptr) {
            cout << "binary search tree is empty\n";
        } else {
            cout << "binary search tree: ";
            traversal(root);
            cout << endl;
        }
    }
};

int main() {
    bst bst;

```

```

char choice;
int value;
do {
    cout << "1. insert\n";
    cout << "2. search\n";
    cout << "3. display\n";
    cout << "4. exit\n";
    cout << "enter your choice: ";
    cin >> choice;
    switch(choice) {
        case '1':
            cout << "enter element to insert: ";
            cin >> value;
            bst.insert(value);
            break;
        case '2':
            cout << "enter element to search: ";
            cin >> value;
            if (bst.search(value)) {
                cout << "found in binary search tree.\n";
            } else {
                cout << "not found in binary search tree.\n";
            }
            break;
        case '3':
            bst.display();
            break;
        case '4':
            cout << "exited!\n";
            break;
        default:
            cout << "invalid choice\n";
    }
} while(choice != '4');
return 0;
}

```

## Output: (screenshot)

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 11thcode.cpp -o 11thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"11thcode
○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 11thcode.cpp -o 11thcode &&
"/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"11thcode
1. insert
2. search
3. display
4. exit
enter your choice:

```

## Test Case: Any two (screenshot)

```

○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 11thcode.cpp -o 11thcode &&
"/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"11thcode
1. insert
2. search
3. display
4. exit
enter your choice: 1
enter element to insert: 22
1. insert
2. search
3. display
4. exit
enter your choice: 1
enter element to insert: 55
1. insert
2. search
3. display

```

```
enter your choice: 1
enter element to insert: 55
1. insert
2. search
3. display
4. exit
enter your choice: 2
enter element to search: 55
found in binary search tree.
1. insert
2. search
3. display
4. exit
enter your choice: █
```

## Conclusion:

In conclusion, the code offers a versatile implementation of a binary search tree, facilitating essential operations with recursive insertion and traversal. The provided menu interface enhances user experience, making it intuitive to interact with the BST functionalities. Overall, the code provides an efficient solution for managing binary search trees.

**Name of Student:** Arpita Jadhav

**Roll Number:** 150096723028

**Experiment No:** 12(a)

---

## Title:



12. Implement Graph Traversal techniques: a) Depth First Search

## Theory:

// The provided code demonstrates the implementation of depth-first search (DFS) traversal on an undirected graph, utilizing an adjacency matrix representation. It first constructs the graph by establishing edges between vertices and then proceeds with DFS traversal from a specified starting vertex. DFS explores each branch as deeply as possible before backtracking.

## Code:

```
#include <iostream>
#include <stack>
using namespace std;
const int MAX_VERTICES = 100;
class Graph {
    int V;
    int** adj;
public:
    Graph(int V) {
        this->V = V;
        adj = new int*[V];
        for (int i = 0; i < V; ++i) {
            adj[i] = new int[V];
            for (int j = 0; j < V; ++j) {
                adj[i][j] = 0;
            }
        }
    }
    void addEdge(int v, int w) {
        adj[v][w] = 1;
        adj[w][v] = 1;
    }
    void DFS(int v) {
        bool* visited = new bool[V];
        for (int i = 0; i < V; ++i)
            visited[i] = false;
        stack<int> stack;
        visited[v] = true;
        stack.push(v);
        while (!stack.empty()) {
            v = stack.top();
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL ...  Code    ... 

```
cd "/Users/sakshikore/Desktop/Semester 2/Data Structure Algorithm 1/DSA Lab Manual/" &&
g++ tnjkn.cpp -o tnjkn && "/Users/sakshikore/Desktop/Semester 2/Data Structure Algorithm 1/DSA Lab Manual/"tnjkn
```

```
○ sakshikore@Sakshis-MacBook-Air DSA Lab Manual % cd "/Users/sakshikore/Desktop/Semester 2/Data Structure Algorithm 1/DSA Lab Manual/" && g++ tnjkn.cpp -o tnjkn && "/Users/sakshikore/Desktop/Semester 2/Data Structure Algorithm 1/DSA Lab Manual/"tnjkn
enter number of vertices & edges: █
```

```
};
int main() {
```

```

int V, E;
cout << "enter number of vertices & edges: ";
cin >> V >> E;
Graph g(V);
cout << "enter edges:" << endl;
for (int i = 0; i < E; ++i) {
    int v, w;
    cin >> v >> w;
    g.addEdge(v, w);
}
int start_vertex;
cout << "enter starting vertex: ";
cin >> start_vertex;
cout << "Depth First Traversal: ";
g.DFS(start_vertex);
return 0;
}

```

## Output: (screenshot)

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 12thcode-1st.cpp -o 12thcode-1st && "/Users/arpitajadhav/Desktop/DSA LAB MANU
AL/"12thcode-1st
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 12thcode-1st.cpp -o 12thcod
e-1st && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"12thcode-1st
enter number of vertices & edges:

```

## Test Case: Any two (screenshot)

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 12thcode-1st.cpp -o 12thcode-1st && "/Users/arpitajadhav/Desktop/DSA LAB MANU
AL/"12thcode-1st
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 12thcode-1st.cpp -o 12thcod
e-1st && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"12thcode-1st
enter number of vertices & edges: 2 3
enter edges:
0 1
1 1
1 0
enter starting vertex: 1
Depth First Traversal: 1 0
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL %

```

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 12thcode-1st.cpp -o 12thcode-1st && "/Users/arpitajadhav/Desktop/DSA LAB MANU
AL/"12thcode-1st
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 12thcode-1st.cpp -o 12thcod
e-1st && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"12thcode-1st
enter number of vertices & edges: 3 4
enter edges:
1 2
1 0
0 1
1 2
enter starting vertex: 2
Depth First Traversal: 2 1 0
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL %

```

**Conclusion:**

In summary, the code effectively executes DFS traversal on an undirected graph represented using an adjacency matrix. It accurately constructs the graph and performs DFS exploration from the specified starting vertex. By traversing each vertex and its adjacent vertices in a depth-first manner, the code ensures comprehensive coverage of reachable vertices within the graph.

**Name of Student:** Arpita Jadhav

**Roll Number:** 150096723028

**Experiment No:** 12(b)

---

**Title:**

12. Implement Graph Traversal techniques: b) Breadth First Search

**Theory:**

The provided code exemplifies the implementation of breadth-first search (BFS) traversal on an undirected graph, employing an adjacency matrix representation. Beginning from a designated starting vertex, BFS systematically explores adjacent vertices iteratively while marking visited nodes and maintaining a queue for traversal.

**Code:**

```
#include <iostream>
```



```

#include <queue>

using namespace std;
const int MAX_VERTICES = 100;
class Graph
{
    int V;
    int** adj;

public:
    Graph(int V) {
        this->V = V;
        adj = new int*[V];
        for (int i = 0; i < V; ++i) {
            adj[i] = new int[V];
            for (int j = 0; j < V; ++j) {
                adj[i][j] = 0;
            }
        }
    }

    void addEdge(int v, int w) {
        adj[v][w] = 1;
        adj[w][v] = 1;
    }

    void BFS(int start) {
        bool* visited = new bool[V];
        for (int i = 0; i < V; ++i)
            visited[i] = false;
        queue<int> q;
        visited[start] = true;
        q.push(start);
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            cout << v << " ";
            for (int i = 0; i < V; ++i) {
                if (adj[v][i] && !visited[i]) {
                    visited[i] = true;
                    q.push(i);
                }
            }
        }
        delete[] visited;
    }
};

int main()
{
    int V, E;
    cout << "enter number of vertices & edges: ";

```

```

cin >> V >> E;
Graph g(V);
cout << "enter edges: " << endl;
for (int i = 0; i < E; ++i) {
    int v, w;
    cin >> v >> w;
    g.addEdge(v, w);
}
int start_vertex;
cout << "enter starting vertex: ";
cin >> start_vertex;
cout << "Breadth First Traversal: ";
g.BFS(start_vertex);
return 0;
}

```

## Output: (screenshot)

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 12thcode-2nd.cpp -o 12thcode-2nd && "/Users/arpitajadhav/Desktop/DSA LAB MANU
AL/"12thcode-2nd
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 12thcode-2nd.cpp -o 12thcod
e-2nd && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"12thcode-2nd
enter number of vertices & edges:

```

## Test Case: Any two (screenshot)

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 12thcode-2nd.cpp -o 12thcode-2nd && "/Users/arpitajadhav/Desktop/DSA LAB MANU
AL/"12thcode-2nd
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 12thcode-2nd.cpp -o 12thcod
e-2nd && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"12thcode-2nd
enter number of vertices & edges: 4 5
enter edges:
1 2
2 1
3 1
1 0
0 1
enter starting vertex: 3
Breadth First Traversal: 3 1 0 2
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL %

```

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 12thcode-2nd.cpp -o 12thcode-2nd && "/Users/arpitajadhav/Desktop/DSA LAB MA
AL/"12thcode-2nd
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 12thcode-2nd.cpp -o 12thc
e-2nd && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"12thcode-2nd
enter number of vertices & edges: 3 3
enter edges:
1 0
0 1
1 2
enter starting vertex: 2
Breadth First Traversal: 2 1 0
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL %

```

## Conclusion:

In summary, the code adeptly executes BFS traversal on an undirected graph represented by an adjacency matrix. By traversing vertices in a breadth-first manner and utilizing a queue for exploration, the code efficiently explores connected components within the graph. Overall, it provides a fundamental algorithmic approach for graph traversal tasks.

**Name of Student:** Arpita Jadhav

**Roll Number:** 150096723028

**Experiment No:** 13

---

## Title:

13. Implement Binary Search algorithm to search an element in an array

## Theory:

The provided code exemplifies the binary search algorithm, a highly efficient means of locating a target element within a sorted array. By repeatedly halving the search interval based on a comparison with the middle element, the algorithm significantly reduces the search space with each iteration, leading to a logarithmic time complexity.

## Code:

```
#include <iostream>
```

```

using namespace std;
int bst(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;
}

int main() {
    int n;
    cout << "enter number of elements in array: ";
    cin >> n;
    int arr[n];
    cout << "enter elements of 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 = bst(arr, 0, n - 1, target);
    if (index != -1)
        cout << "element found at index " << index << endl;
    else
        cout << "element not found" << endl;
    return 0;
}

```

## Output: (screenshot)

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 13thcode.cpp -o 13thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"13thcode
○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 13thcode.cpp -o 13thcode &&
"/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"13thcode
enter number of elements in array:

```

## Test Case: Any two (screenshot)

```

cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 13thcode.cpp -o 13thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"13thcode
● arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 13thcode.cpp -o 13thcode &&
"/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"13thcode
enter number of elements in array: 4
enter elements of array in sorted order: 2
12
11
10
enter the element to search: 12
element found at index 1
○ arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL %

```

```
arpitajadhav@ARPITAS-MacBook-Air: DSA LAB MANUAL % cd /Users/arpitajadhav/Desktop/DSA LAB MANUAL/ && g++ 13thcode.cpp -o 13thcode &&
"/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"13thcode
enter number of elements in array: 5
enter elements of array in sorted order: 2
99
100
23
45
enter the element to search: 23
element not found
arpitajadhav@ARPITAS-MacBook-Air: DSA LAB MANUAL %
```

## Conclusion:

In conclusion, the binary search algorithm demonstrated in the code offers a robust and efficient solution for finding a target element within a sorted array. With its logarithmic time complexity, it outperforms linear search methods, particularly when dealing with large datasets, making it a valuable tool in algorithmic problem-solving.

**Name of Student:** Arpita Jadhav

**Roll Number:** 150096723028

**Experiment No:** 14

---

## Title:

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

## Theory:

This code demonstrates two sorting functions: ascendingsort and descendingsort. Both functions implement the bubble sort algorithm to sort an array of integers in ascending and descending order, respectively. Bubble sort repeatedly steps through the list,

compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the array is sorted.

### Code:

```
#include <iostream>
using namespace std;
void ascendingsort(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 descendingsort(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;
            }
        }
    }
}
int main() {
    int n;
    cout << "enter number of elements in array: ";
    cin >> n;
    int arr[n];
    cout << "enter elements of array: ";
    for (int i = 0; i < n; ++i)
        cin >> arr[i];
    ascendingsort(arr, n);
    cout << "array in ascending order: ";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << endl;
    descendingsort(arr, n);
    cout << "array in descending order: ";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << endl;
    return 0;
}
```

### Output: (screenshot)

```
cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 14thcode.cpp -o 14thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"14th
code
○ arpitajadhav@ARPITAs-MacBook-Air:~/Desktop/DSA LAB MANUAL$ g++ 14thcode.cpp -o 14thcode &&
"/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"14thcode
enter number of elements in array:
```

## Test Case: Any two (screenshot)

```
cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 14thcode.cpp -o 14thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"14thcode
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 14thcode.cpp -o 14thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"14thcode
enter number of elements in array: 4
enter elements of array: 1
2
3
4
array in ascending order: 1 2 3 4
array in descending order: 4 3 2 1
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL %
```

```
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL % cd "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/" && g++ 14thcode.cpp -o 14thcode && "/Users/arpitajadhav/Desktop/DSA LAB MANUAL/"14thcode
enter number of elements in array: 5
enter elements of array: 111
112
113
99
1
array in ascending order: 1 99 111 112 113
array in descending order: 113 112 111 99 1
arpitajadhav@ARPITAs-MacBook-Air DSA LAB MANUAL %
```

## Conclusion:

The bubble sort algorithm presented in the code provides a simple approach to sorting elements in an array. However, it has a time complexity of  $O(n^2)$ , making it inefficient for large datasets compared to more efficient sorting algorithms like merge sort or quicksort.