Shaheed Benazir Bhutto University
Shaheed Benazirabad
KNOWLEDGE - COMMITMENT - LEADERSHIP

PROGRAMMING FUNDAMENTALS
LAB MANUAL/ASSIGNMENT

STUDENT NAME

ABDUL HASEEB MEMON

ROLL NO

24-BSCS-43

SUBJECT TEACHER

FAHEEM SHAH

DATE

10-MAY-2025

DEPARTMENT OF COMPUTER SCIENCE

SHAHEED BENAZIR BHUTTO UNIVERSITY, SHAHEED BENAZIR ABAD

Practical # 01

Objective:

Recall *C++ programming and demonstrate C++ classes and templates using Code Blocks.*

Theory:

In order to give a quick review of C++, Rectangle Class (class Declaration, Member function definition, calling) is demonstrated in the lab.

Rectangle Class Pseudo Code:

- ✓ Declare a class Rect, with data members: length, breadth, area, and perimeter.
- ✓ Declare member functions to accept input and compute area and perimeters: input (), Compute Area (), Compute Peri (), Display ()
- ✓ Write a main function:
    - instantiate the class by passing appropriate arguments to the constructor
    - invoke Compute rear (), Compute Peri () and Display () functions

Program in C++:

```
#include<iostream>
using namespace std;
class Rect{
public:
    int length, breadth, area, perimeter;

     void input() {
        cout << "Enter length of rectangle:";
        cin >> length;
        cout << "Enter breadth of rectangle:";
        cin>>breadth;
    }

void ComputeArea() {
        area = length * breadth;
    }
 void ComputePerimeter() {
     perimeter = 2 * (length + breadth) ;
 }
void display() {
    cout << "Area of rectangle is:" << area;
    cout << "\nPerimeter of rectangle is:" << perimeter;
}

};

int main() {

    Rect obj;

    obj.input();
    obj.ComputeArea();
    obj.ComputePerimeter();
    obj.display();

    return 0;
}
```

OUTPUT:

```
Enter length of rectangle:4
Enter breadth of rectangle:8
Area of rectangle is:32
Perimeter of rectangle is:24
```

Review Questions/ Exercise

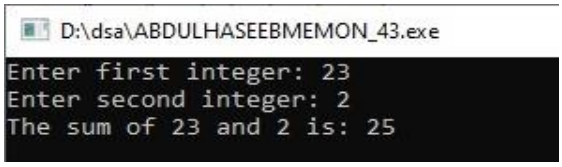
1. Write C++ program to add two integer values and print the output with descriptive message.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
    int num1, num2, sum;
    // Ask user for input
    cout << "Enter first integer: ";
    cin >> num1;
    cout << "Enter second integer: ";
    cin >> num2;
    // Calculate sum
    sum = num1 + num2;
    // Display result
    cout << "The sum of " << num1 << " and " << num2 << " is: " << sum << endl;
    return 0;
}
```

OUTPUT:



2. Write C++ program to convert temperature from centigrade to Fahrenheit.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
    float centigrade, fahrenheit;
    // Ask user for input
    cout << "Enter temperature in Centigrade: ";
    cin >> centigrade;
    // Convert temperature
    fahrenheit = (centigrade * 9.0 / 5.0) + 32.0;
    // Display result
    cout << centigrade << "°C is equal to " << fahrenheit << "°F" << endl;
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Enter temperature in Centigrade: 23
23°C is equal to 73.4°F
```

3. Write C++ program to build simple calculator using switch case statement

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
    float num1, num2;
    char operation;
    // Ask user for input
    cout << "Enter first number: ";
    cin >> num1;
    cout << "Enter operator (+, -, *, /): ";
    cin >> operation;
    cout << "Enter second number: ";
    cin >> num2;
    // Perform calculation using switch case
    switch (operation) {
        case '+':
            cout << num1 << " + " << num2 << " = " << num1 + num2 << endl;
            break;
        case '-':
            cout << num1 << " - " << num2 << " = " << num1 - num2 << endl;
            break;
        case '*':
            cout << num1 << " * " << num2 << " = " << num1 * num2 << endl;
            break;
        case '/':
            if (num2 != 0)
                cout << num1 << " / " << num2 << " = " << num1 / num2 << endl;
            else
                cout << "Error: Division by zero!" << endl;
            break;
        default:
            cout << "Error: Invalid operator!" << endl;
            break;
    }
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Enter first number: 3
Enter operator (+, -, *, /): +
Enter second number: 4
3 + 4 = 7
```

4. Write a program to display the pattern as shown below:

```
*
***
*****
*******
*********
*********
*******
*****
***
*
```

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
    int rows;
    // Ask user for input
    cout << "Enter number of rows: ";
    cin >> rows;
    // Display upper half of the diamond
    for (int i = 1; i <= rows; i++) {
        for (int j = 1; j <= i; j++) {
            cout << "* ";
        }
        cout << endl;
    }
    // Display lower half of the diamond
    for (int i = rows - 1; i >= 1; i--) {
        for (int j = 1; j <= i; j++) {
            cout << "* ";
        }
        cout << endl;
    }
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Enter number of rows: 5
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```

5. Write a program to display the average of three numbers entered by the user by creating a class named 'Average' having a function to calculate and display the average without creating any object of the Average class

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
class Average {
public:
    static void calculateAverage() {
        float num1, num2, num3;
        // Ask user for input
        cout << "Enter first number: ";
        cin >> num1;
        cout << "Enter second number: ";
        cin >> num2;
        cout << "Enter third number: ";
        cin >> num3;
        // Calculate and display average
        float average = (num1 + num2 + num3) / 3.0;
        cout << "The average is: " << average << endl;
    }
};
int main() {
    // Call static method without creating an object
    Average::calculateAverage();
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Enter first number: 34
Enter second number: 12
Enter third number: 2
The average is: 16
```

6. Write a program to display the sum, difference and product of two complex numbers by creating a class named 'Complex' with separate functions for each operation whose real and imaginary parts are entered by the user.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
class Complex {
private:
    double real, imag;
public:
    // Constructor to initialize complex number
    Complex(double r = 0, double i = 0) : real(r), imag(i) {}
    // Function to input complex number
    void input() {
        cout << "Enter real part: ";
        cin >> real;
        cout << "Enter imaginary part: ";
        cin >> imag;
    }
    // Function to display complex number
    void display() {
        cout << real;
        if (imag >= 0)
            cout << " + " << imag << "i" << endl;
        else
            cout << " - " << -imag << "i" << endl;
    }
    // Function to add two complex numbers
    Complex add(Complex c) {
        return Complex(real + c.real, imag + c.imag);
    }
    // Function to subtract two complex numbers
    Complex subtract(Complex c) {
        return Complex(real - c.real, imag - c.imag);
    }
    // Function to multiply two complex numbers
    Complex multiply(Complex c) {
        return Complex(real * c.real - imag * c.imag, real * c.imag + imag * c.real);
    }
};
```

```
37  int main() {
38      Complex c1, c2, sum, diff, prod;
39      cout << "Enter first complex number:" << endl;
40      c1.input();
41      cout << "Enter second complex number:" << endl;
42      c2.input();
43      sum = c1.add(c2);
44      diff = c1.subtract(c2);
45      prod = c1.multiply(c2);
46      cout << "First complex number: ";
47      c1.display();
48      cout << "Second complex number: ";
49      c2.display();
50      cout << "Sum: ";
51      sum.display();
52      cout << "Difference: ";
53      diff.display();
54      cout << "Product: ";
55      prod.display();
56      return 0;
57  }
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Enter first complex number:
Enter real part: 3
Enter imaginary part: 5
Enter second complex number:
Enter real part: 5
Enter imaginary part: 3
First complex number: 3 + 5i
Second complex number: 5 + 3i
Sum: 8 + 8i
Difference: -2 + 2i
Product: 0 + 34i
```

Name: Abdul Haseeb Memon

Roll #: 24-BS(CS)-43

Date: 10-MAY-2025

Remarks: **Subject Teacher: Faheem Shah**

Practical # 02

Objective:

Discuss *one dimensional Array. Write C++ to declare and print elements of one-dimensional Array.*

Theory:

In this Lab, we will introduce the array, a homogeneous data type also used to represent a group of data. This data type might be used to store many items of the same type as a conceptual unit. For example, all of the scores made on a test by students in a data structure and algorithm class might be stored in an array, or the names of students are stored in an array.

Lab Objectives:

- To be able to declare a one-dimensional array.
- To be able to perform fundamental operations on a one-dimensional array.
- To learn some common ways to search for an item in a one-dimensional array.

Introduction:

A **one-dimensional array** is a structured collection of components (often called array elements) that can be accessed individually by specifying the position of a component with a single index value. The syntax of a one-dimensional array declaration is:

Data_Type Array_Name [Int_Value];

Example program: D*eclare and print elements of one-dimensional Array.*

```cpp
#include<iostream>
using namespace std;
int main()
{    /*an array with 5 rows*/
    int a[5]= {10,20,30,40,50};
    int i;
    /* Output each array elements value*/
    for(i=0; i<5; i++)
    {
       cout<<"element"<<i<<"="<<a[i]<<"\n";
    }
}
```

OUTPUT

```
element0=10
element1=20
element2=30
element3=40
element4=50
```

Review Questions/ Exercise:

1. Write a C++ program to print largest value of array.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include<iostream>
using namespace std;
int main() {
    int size;
    // Ask user for array size
    cout << "Enter array size: ";
    cin >> size;
    int arr[size];
    // Input array elements
    cout << "Enter " << size << " elements:" << endl;
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }
    // Find largest element
    int largest = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] > largest)
            largest = arr[i];
    }
    // Display largest element
    cout << "Largest element: " << largest << endl;
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Enter array size: 5
Enter 5 elements:
2
3
4
5
6
Largest element: 6
```

2. Write a C++ program to add an element at given index.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
    int size, index, element;
    // Ask user for array size
    cout << "Enter array size: ";
    cin >> size;
    int arr[size + 1]; // Extra space for new element
    // Input array elements
    cout << "Enter " << size << " elements:" << endl;
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }
    // Ask user for index and element to insert
    cout << "Enter index (0-" << size << "): ";
    cin >> index;
    cout << "Enter element to insert: ";
    cin >> element;
    // Shift elements to make space for new element
    for (int i = size; i > index; i--) {
        arr[i] = arr[i - 1];
    }
    // Insert element at given index
    arr[index] = element;
    size++; // Increment size
    // Display updated array
    cout << "Updated array: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Enter array size: 4
Enter 4 elements:
2
3
5
6
Enter index (0-4): 4
Enter element to insert: 3
Updated array: 2 3 5 6 3
```

3. Write a C++ program to deletes an element at the given index.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
    int size, index;
    // Ask user for array size
    cout << "Enter array size: ";
    cin >> size;
    int arr[size];
    // Input array elements
    cout << "Enter " << size << " elements:" << endl;
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }
    // Ask user for index to delete
    cout << "Enter index (0-" << size - 1 << "): ";
    cin >> index;
    // Check if index is valid
    if (index < 0 || index >= size) {
        cout << "Invalid index!" << endl;
        return 1;
    }
    // Shift elements to fill the gap
    for (int i = index; i < size - 1; i++) {
        arr[i] = arr[i + 1];
    }
    size--; // Decrement size
    // Display updated array
    cout << "Updated array: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Enter array size: 4
Enter 4 elements:
2
3
5
7
Enter index (0-3): 2
Updated array: 2 3 7
```

4. Write a C++ program to searches an element using the given index or by the value.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Function to search by index
void searchByIndex(int arr[], int size) {
    int index;
    cout << "Enter index (0-" << size - 1 << "): ";
    cin >> index;
    if (index < 0 || index >= size) {
        cout << "Invalid index!" << endl;
    } else {
        cout << "Element at index " << index << ": " << arr[index] << endl;
    }
}
// Function to search by value
void searchByValue(int arr[], int size) {
    int value;
    cout << "Enter value to search: ";
    cin >> value;
    bool found = false;
    for (int i = 0; i < size; i++) {
        if (arr[i] == value) {
            cout << "Element " << value << " found at index " << i << endl;
            found = true;
        }
    }
    if (!found) {
        cout << "Element " << value << " not found!" << endl;
    }
}
int main() {
    int size;
    cout << "Enter array size: ";
    cin >> size;
    int arr[size];
    cout << "Enter " << size << " elements:" << endl;
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }
    int choice;
    cout << "Search by: 1. Index 2. Value" << endl;
    cin >> choice;
    switch (choice) {
        case 1:
            searchByIndex(arr, size);
            break;
        case 2:
            searchByValue(arr, size);
            break;
        default:
            cout << "Invalid choice!" << endl;
            break;
    }
    return 0;
}
```

OUTPUT:

```
Select D:\dsa\ABDULHASEEBMEMON_43.exe
Enter array size: 6
Enter 6 elements:
2
5
6
7
8
3
Search by: 1. Index 2. Value
1
Enter index (0-5): 2
Element at index 2: 6
```

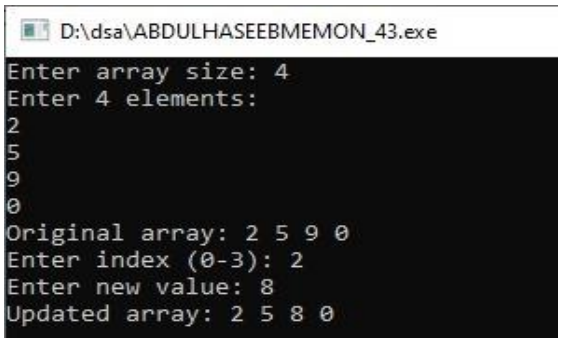
5. Write a C++ program to updates an element at the given index.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
    int size, index, newValue;
    // Ask user for array size
    cout << "Enter array size: ";
    cin >> size;
    int arr[size];
    // Input array elements
    cout << "Enter " << size << " elements:" << endl;
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }
    // Display original array
    cout << "Original array: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    // Ask user for index and new value
    cout << "Enter index (0-" << size - 1 << "): ";
    cin >> index;
    cout << "Enter new value: ";
    cin >> newValue;
    // Check if index is valid
    if (index < 0 || index >= size) {
        cout << "Invalid index!" << endl;
    } else {
        // Update element at given index
        arr[index] = newValue;
        // Display updated array
        cout << "Updated array: ";
        for (int i = 0; i < size; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

OUTPUT:



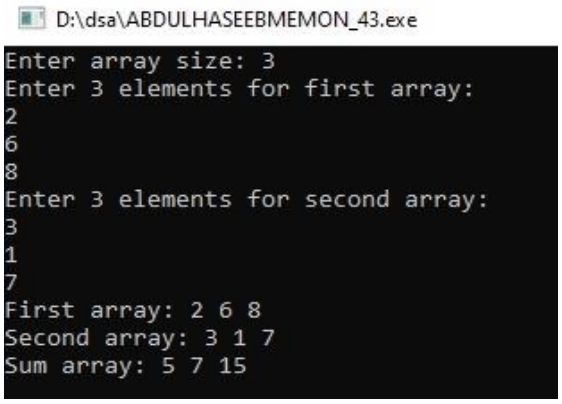
6. Write a C++ program to add two integer arrays and print the output array.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
    int size;
    // Ask user for array size
    cout << "Enter array size: ";
    cin >> size;
    int arr1[size], arr2[size], sum[size];
    // Input first array elements
    cout << "Enter " << size << " elements for first array:" << endl;
    for (int i = 0; i < size; i++) {
        cin >> arr1[i];
    }
    // Input second array elements
    cout << "Enter " << size << " elements for second array:" << endl;
    for (int i = 0; i < size; i++) {
        cin >> arr2[i];
    }
    // Add corresponding elements of two arrays
    for (int i = 0; i < size; i++) {
        sum[i] = arr1[i] + arr2[i];
    }
    // Display arrays and sum
    cout << "First array: ";
    for (int i = 0; i < size; i++) {
        cout << arr1[i] << " ";
    }
    cout << endl;
    cout << "Second array: ";
    for (int i = 0; i < size; i++) {
        cout << arr2[i] << " ";
    }
    cout << endl;
    cout << "Sum array: ";
    for (int i = 0; i < size; i++) {
        cout << sum[i] << " ";
    }
    cout << endl;
    return 0;
}
```

OUTPUT:



Name: Abdul Haseeb Memon

Roll #: 24-BS(CS)-43

Date: 10-MAY-2025

Remarks: **Subject Teacher: Faheem Shah**

Practical # 03

Objective:

Discuss *multi- dimensional Array. Write C++ to declare and print elements of multi dimensional Array.*

Theory:

In this Lab, we can define multidimensional arrays in simple words as array of arrays. Data in multidimensional arrays are stored in tabular form (in row major order).

Lab Objectives:

- To be able to declare a multi-dimensional array.
- To be able to perform fundamental operations on a multi-dimensional array.
- To learn some common ways to search for an item in a multi-dimensional array.

Introduction:

A **multi-dimensional array** is a structured collection of components (often called array elements) that can be accessed individually by specifying the position of a component with a double index value. General form of declaring N-dimensional arrays:

data_type array_name[size1][size2] ..... [sizeN];

Total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions.

Example program: D*eclare and print elements of multi-dimensional Array.*

```cpp
#include<iostream>
using namespace std;
int main()
{    // an array with 3 rows and 2 columns.
    int x[3][2] = {{0,1}, {2,3}, {4,5}};
    // output each array element's value
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            cout << "Element at x[" << i
                 << "][" << j << "]: ";
            cout << x[i][j]<<endl;
        }
    }
   return 0; }
```

OUTPUT

```
Element at x[0][0]: 0
Element at x[0][1]: 1
Element at x[1][0]: 2
Element at x[1][1]: 3
Element at x[2][0]: 4
Element at x[2][1]: 5
```

Review Questions/ Exercise:

1. Write a C++ program to add two 2-dimensional arrays.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
    int rows, cols;
    // Ask user for matrix dimensions
    cout << "Enter number of rows: ";
    cin >> rows;
    cout << "Enter number of columns: ";
    cin >> cols;
    int matrix1[rows][cols], matrix2[rows][cols], sum[rows][cols];
    // Input first matrix elements
    cout << "Enter elements for first matrix:" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> matrix1[i][j];
        }
    }
    // Input second matrix elements
    cout << "Enter elements for second matrix:" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> matrix2[i][j];
        }
    }
    // Add corresponding elements of two matrices
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            sum[i][j] = matrix1[i][j] + matrix2[i][j];
        }
    }
    // Display matrices and sum
    cout << "First matrix:" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << matrix1[i][j] << " ";
        }
        cout << endl;
    }
     cout << "Second matrix:" << endl;
     for (int i = 0; i < rows; i++) {
         for (int j = 0; j < cols; j++) {
             cout << matrix2[i][j] << " ";
         }
         cout << endl;
     }
     cout << "Sum matrix:" << endl;
     for (int i = 0; i < rows; i++) {
         for (int j = 0; j < cols; j++) {
             cout << sum[i][j] << " ";
         }
         cout << endl;
     }
     return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Enter number of rows: 2
Enter number of columns: 2
Enter elements for first matrix:
1 2
3 4
Enter elements for second matrix:
4 5
6 8
First matrix:
1 2
3 4
Second matrix:
4 5
6 8
Sum matrix:
5 7
9 12
```

2. Write C++ program to display the diagonal elements of a given matrix.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
    int rows, cols;
    // Ask user for matrix dimensions
    cout << "Enter number of rows: ";
    cin >> rows;
    cout << "Enter number of columns: ";
    cin >> cols;
    int matrix[rows][cols];
    // Input matrix elements
    cout << "Enter elements for matrix:" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> matrix[i][j];
        }
    }
    // Display matrix
    cout << "Matrix:" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
    // Display primary diagonal elements (if square matrix)
    if (rows == cols) {
        cout << "Primary diagonal elements: ";
        for (int i = 0; i < rows; i++) {
            cout << matrix[i][i] << " ";
        }
        cout << endl;
    } else {
        cout << "Matrix is not square, primary diagonal not applicable." << endl;
    }
    // Display secondary diagonal elements (if square matrix)
    if (rows == cols) {
        cout << "Secondary diagonal elements: ";
```

```
        for (int i = 0; i < rows; i++) {
            cout << matrix[i][rows - i - 1] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Enter number of rows: 2
Enter number of columns: 1
Enter elements for matrix:
2
4
Matrix:
2
4
Matrix is not square, primary diagonal not applicable.
```

3. Write a C++ program to Find Sum of Diagonal Elements of Matrix.

SOURCE CODE:

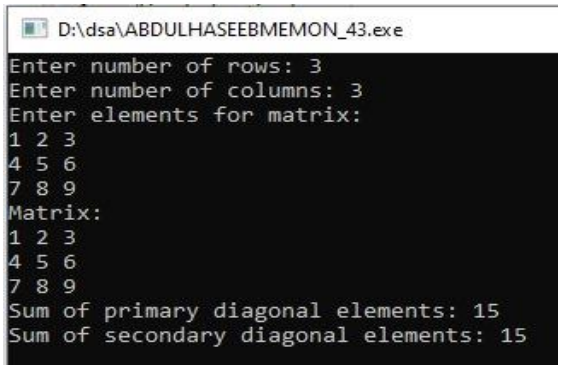
ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
    int rows, cols;
    // Ask user for matrix dimensions
    cout << "Enter number of rows: ";
    cin >> rows;
    cout << "Enter number of columns: ";
    cin >> cols;
    int matrix[rows][cols];
    // Input matrix elements
    cout << "Enter elements for matrix:" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> matrix[i][j];
        }
    }
    // Display matrix
    cout << "Matrix:" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
    // Calculate sum of primary diagonal elements (if square matrix)
    if (rows == cols) {
        int primarySum = 0;
        for (int i = 0; i < rows; i++) {
            primarySum += matrix[i][i];
        }
        cout << "Sum of primary diagonal elements: " << primarySum << endl;
    } else {
        cout << "Matrix is not square, primary diagonal not applicable." << endl;
    }
    // Calculate sum of secondary diagonal elements (if square matrix)
    if (rows == cols) {
        int secondarySum = 0;
```

```
        for (int i = 0; i < rows; i++) {
            secondarySum += matrix[i][rows - i - 1];
        }
        cout << "Sum of secondary diagonal elements: " << secondarySum << endl;
    }

    return 0;
}
```

OUTPUT:



4. Write a C++ program to multiply two integer 3-dimensional arrays and print the output array.

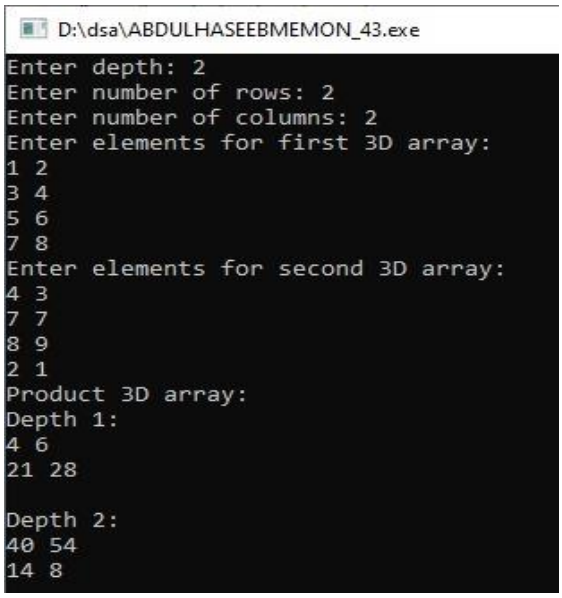
SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```
#include <iostream>
using namespace std;
int main() {
    int depth, rows, cols;
    // Ask user for 3D array dimensions
    cout << "Enter depth: ";
    cin >> depth;
    cout << "Enter number of rows: ";
    cin >> rows;
    cout << "Enter number of columns: ";
    cin >> cols;
    int arr1[depth][rows][cols], arr2[depth][rows][cols], product[depth][rows][cols];
    // Input first 3D array elements
    cout << "Enter elements for first 3D array:" << endl;
    for (int k = 0; k < depth; k++) {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                cin >> arr1[k][i][j];
            }
        }
    }
    // Input second 3D array elements
    cout << "Enter elements for second 3D array:" << endl;
    for (int k = 0; k < depth; k++) {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                cin >> arr2[k][i][j];
            }
        }
    }
    // Multiply corresponding elements of two 3D arrays
    for (int k = 0; k < depth; k++) {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                product[k][i][j] = arr1[k][i][j] * arr2[k][i][j];
            }
        }
    }
```

```
    // Display product 3D array
    cout << "Product 3D array:" << endl;
    for (int k = 0; k < depth; k++) {
        cout << "Depth " << k + 1 << ":" << endl;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                cout << product[k][i][j] << " ";
            }
            cout << endl;
        }
        cout << endl;
    }
    return 0;
}
```

OUTPUT:



5. Write a C++ program to Transpose Matrix.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
    int rows, cols;
    // Ask user for matrix dimensions
    cout << "Enter number of rows: ";
    cin >> rows;
    cout << "Enter number of columns: ";
    cin >> cols;
    int matrix[rows][cols], transpose[cols][rows];
    // Input matrix elements
    cout << "Enter elements for matrix:" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> matrix[i][j];
        }
    }
    // Transpose matrix
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            transpose[j][i] = matrix[i][j];
        }
    }
    // Display original matrix
    cout << "Original Matrix:" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
    // Display transposed matrix
    cout << "Transposed Matrix:" << endl;
    for (int i = 0; i < cols; i++) {
        for (int j = 0; j < rows; j++) {
            cout << transpose[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Enter number of rows: 2
Enter number of columns: 3
Enter elements for matrix:
3 4 5
2 6 7
Original Matrix:
3 4 5
2 6 7
Transposed Matrix:
3 2
4 6
5 7
```

Practical # 04

Objective:

Discuss *String Processing.*

Theory:

Give some printed text; the operations usually associated with word processing are the following:

a) **Insertion:** Inserting a string in the text.
b) **Deletion:** Deleting a string from text.
c) **Replacement:** Replacing one string in the text by another.

Lab Objectives:

- To be able to insert a string in the text.
- To be able to perform fundamental operations on text.

C++ program: Write C++ to insert the strings at the end of string1.

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char marr[100], sarr[10], narr[200];
    size_t index;
    cout << "Enter main array data: ";
    cin.getline( marr, sizeof( marr ) );
    size_t mlen = strlen( marr );
    cout << "Enter word to be inserted: ";
    cin.getline( sarr, sizeof( sarr) );

index = mlen;
index = mlen < index ? mlen : index;
```

```
    index = mlen;
    index = mlen < index ? mlen : index;
    size_t i = 0;
    for ( ; i < index; i++ ) narr[i] = marr[i];
    for ( size_t j = 0; sarr[j] != '\0'; j++, i++ ) narr[i] = sarr[j];
    for ( size_t j = index; j < mlen + 1; j++, i++ ) narr[i] = marr[j];
    cout<<"New Array: ";
    for ( size_t j = 0; j < i; j++ )
    cout << narr[j];
    cout << '\n';
    return 0;
}
```

OUTPUT

```
Enter main array data: CSE
Enter word to be inserted: QUEST
New Array: CSEQUEST
```

Review Questions/ Exercise:

1. Write C++ program to insert string "System" in the middle of the message "Computer Engineering" and print output as "Computer System Engineering".

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
    string message = "Computer Engineering";
    string insertStr = " System ";
    // Find the middle position
    int midPos = message.find(' ');
    // Insert the string at the middle position
    message.insert(midPos + 1, insertStr);
    cout << "Output: " << message << endl;
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Output: Computer  System Engineering
```

2. Write C++ program to replacing one string in the text by another.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
    string text = "I love programming in Java. Java is fun.";
    string oldStr = "Java";
    string newStr = "C++";
    // Find and replace all occurrences
    size_t pos = text.find(oldStr);
    while (pos != string::npos) {
        text.replace(pos, oldStr.length(), newStr);
        pos = text.find(oldStr, pos + newStr.length());
    }
    cout << "Original text: I love programming in Java. Java is fun." << endl;
    cout << "Modified text: " << text << endl;
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Original text: I love programming in Java. Java is fun.
Modified text: I love programming in C++. C++ is fun.
```

3. Write C++ program to delete one sub string from the text.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
    string text = "I love programming in C++. C++ is fun.";
    string subStr = "C++";
    size_t pos = text.find(subStr);
    while (pos != string::npos) {
        text.erase(pos, subStr.length());
        // Remove extra spaces if any
        if (pos > 0 && text[pos - 1] == ' ' && pos < text.length() && text[pos] == ' ') {
            text.erase(pos, 1);
        }
        pos = text.find(subStr);
    }
    cout << "Original text: I love programming in C++. C++ is fun." << endl;
    cout << "Modified text: " << text << endl;
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Original text: I love programming in C++. C++ is fun.
Modified text: I love programming in . is fun.
```

Name: Abdul Haseeb Memon

Roll #: 24-BS(CS)-43

Date: 10-MAY-2025

Remarks: **Subject Teacher: Faheem Shah**

Practical # 05

Objective:

Discuss *and Analyze implementation of Singly Linked List Data Structure.* *Write a C++ program to create a singly linked list of integers.*

Theory:

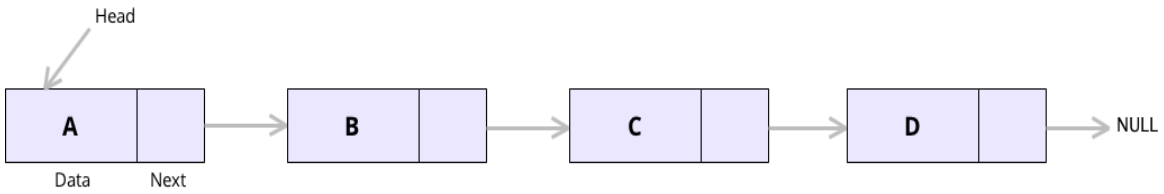
In this Lab, we will introduce the singly Linked list and possible operations performed on singly linked list.

Lab Objectives:

- To be able to declare a singly linked list.
- To be able to perform fundamental operations on singly linked list.

Introduction:

The **singly linked** list contains nodes that only point to the next node. A node has two parts: the data part and the next part. The **data** part contains the stored data, and the **next** part provides the address of the next node. The first node of a linked list is called the **head**, and the last node is called the **tail**. The list starts traversing from the head, while the tail ends the list by pointing at NULL.



Declaring a Node type

To set up a linked list, the first thing we'll need is a structure that represents a single node in the list. For simplicity, let's assume that a node contains nothing but an integer(the node's data) plus a pointer to the next node in the list. Here's what our node structure looks

like:

```
struct node{
        int value;                          /* data stored in the node */
        struct node * pointer;              /* pointer to the next node */
```

Notice that the ***next member*** has type ***struct node *,*** which means that it can store a pointer to a node structure. There's nothing about the name node, by the way; it's just an ordinary structure tag.
Now that we have the node structure declared, we'll need a way to keep track of where the list begins. In other words, we'll need a variable that always points to the first node in the list. Let's name the variable start:

```
struct node  *start=NULL;
```

Setting start to null indicates that the list is initially empty.

Creating a Node

As we construct a liked list, we want to create node one by one, adding each node to the list. Creating a node requires three steps.

1. Allocate memory for the node.
2. Store data in the node.
3. Insert node in to the list.

Example program: *Declare and print elements of singly linked list.*

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

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

void print_list(Node * n) {
    while (n != NULL) {
        cout << n->data << " ";
        n = n->next;
    }
}
int main() {

    Node * head = NULL;
    Node * second = NULL;
    Node * third = NULL;

    head = new Node();
    second = new Node();
    third = new Node();

    head->data = 1;
    head->next = second;

    second-> data = 2;
    second-> next = third;
     third-> data = 3;
     third-> next = NULL;

     print_list(head);
 }
```

OUTPUT

```
1 2 3
Process returned 0 (0x0)   execution time : 0.061 s
Press any key to continue.
```

Review Questions/ Exercise:

1. Write a program to find the number NUM of elements in a linked list.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
};
// Function to insert a new node at the end
void insertNode(Node** head, int newData) {
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = NULL;
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* last = *head;
    while (last->next) {
        last = last->next;
    }
    last->next = newNode;
}
// Function to count the number of nodes
int countNodes(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp) {
        count++;
        temp = temp->next;
    }
    return count;
}
// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    Node* head = NULL;
    // Insert nodes
    insertNode(&head, 1);
    insertNode(&head, 2);
    insertNode(&head, 3);
    insertNode(&head, 4);
    insertNode(&head, 5);
    cout << "Linked List: ";
    printList(head);
    // Count nodes
    int numNodes = countNodes(head);
    cout << "Number of nodes: " << numNodes << endl;
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Linked List: 1 2 3 4 5
Number of nodes: 5
```

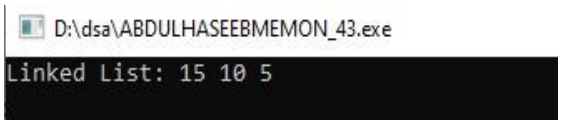
2. Write a program to insert a new node at the beginning of liked list data structure.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
};
// Function to insert a new node at the beginning
void insertAtBeginning(Node** head, int newData) {
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = *head;
    *head = newNode;
}
// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    Node* head = NULL;
    // Insert nodes
    insertAtBeginning(&head, 5);
    insertAtBeginning(&head, 10);
    insertAtBeginning(&head, 15);
    cout << "Linked List: ";
    printList(head);
    return 0;
}
```

OUTPUT:



3. Write a program to insert new node at given position of the linked list data structure.

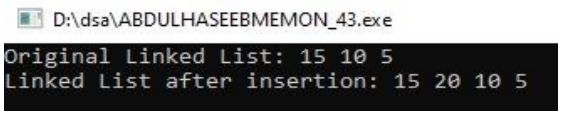
SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
};
// Function to insert a new node at the beginning
void insertAtBeginning(Node** head, int newData) {
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = *head;
    *head = newNode;
}
// Function to insert a new node at a given position
void insertAtPosition(Node** head, int newData, int pos) {
    Node* newNode = new Node();
    newNode->data = newData;
    if (pos == 0) {
        newNode->next = *head;
        *head = newNode;
        return;
    }
    Node* temp = *head;
    for (int i = 0; temp && i < pos - 1; i++) {
        temp = temp->next;
    }
    if (!temp) {
        cout << "Position out of bounds" << endl;
        return;
    }
    newNode->next = temp->next;
    temp->next = newNode;
}
// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
```

```
    cout << endl;
}
int main() {
    Node* head = NULL;
    // Insert nodes
    insertAtBeginning(&head, 5);
    insertAtBeginning(&head, 10);
    insertAtBeginning(&head, 15);
    cout << "Original Linked List: ";
    printList(head);
    // Insert node at position 1
    insertAtPosition(&head, 20, 1);
    cout << "Linked List after insertion: ";
    printList(head);
    return 0;
}
```

OUTPUT:



4. Write a program to find the location LOC of the last node in a sorted list.

SOURCE CODE:

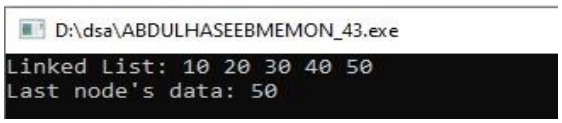
ABDULHASEEBMEMON_43.cpp

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
};
// Function to insert a new node at the end
void insertNode(Node** head, int newData) {
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = NULL;

    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* last = *head;
    while (last->next) {
        last = last->next;
    }
    last->next = newNode;
}
// Function to find the last node
Node* findLastNode(Node* head) {
    Node* temp = head;
    if (temp == NULL) {
        return NULL;
    }
    while (temp->next) {
        temp = temp->next;
    }
    return temp;
}
// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
```

```
    cout << endl;
}
int main() {
    Node* head = NULL;
    // Insert nodes
    insertNode(&head, 10);
    insertNode(&head, 20);
    insertNode(&head, 30);
    insertNode(&head, 40);
    insertNode(&head, 50);
    cout << "Linked List: ";
    printList(head);
    // Find the last node
    Node* lastNode = findLastNode(head);
    if (lastNode) {
        cout << "Last node's data: " << lastNode->data << endl;
    } else {
        cout << "List is empty" << endl;
    }
    return 0;
}
```

OUTPUT:



5. Write a program to delete node from linked list.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
};
// Function to insert a new node at the end
void insertNode(Node** head, int newData) {
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = NULL;
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* last = *head;
    while (last->next) {
        last = last->next;
    }
    last->next = newNode;
}
// Function to delete a node with given key
void deleteNode(Node** head, int key) {
    // If list is empty
    if (*head == NULL) return;
    // If node to be deleted is head node
    if ((*head)->data == key) {
        Node* temp = *head;
        *head = (*head)->next;
        delete temp;
        return;
    }
    Node* temp = *head;
    while (temp->next) {
        if (temp->next->data == key) {
            Node* nodeToDelete = temp->next;
            temp->next = temp->next->next;
            delete nodeToDelete;
            return;
        }
```

```
42          temp = temp->next;
43      }
44  }
45  // Function to print the linked list
46  void printList(Node* head) {
47      Node* temp = head;
48      while (temp) {
49          cout << temp->data << " ";
50          temp = temp->next;
51      }
52      cout << endl;
53  }
54  int main() {
55      Node* head = NULL;
56      // Insert nodes
57      insertNode(&head, 10);
58      insertNode(&head, 20);
59      insertNode(&head, 30);
60      insertNode(&head, 40);
61      insertNode(&head, 50);
62      cout << "Linked List: ";
63      printList(head);
64      // Delete node with key 30
65      deleteNode(&head, 30);
66      cout << "Linked List after deletion: ";
67      printList(head);
68      return 0;
69  }
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Linked List: 10 20 30 40 50
Linked List after deletion: 10 20 40 50
```

Practical # 06

Objective:

Discuss *and Analyze implementation of Doubly Linked List Data Structure. Write a C++ program to create a Doubly linked list of integers.*

Theory:

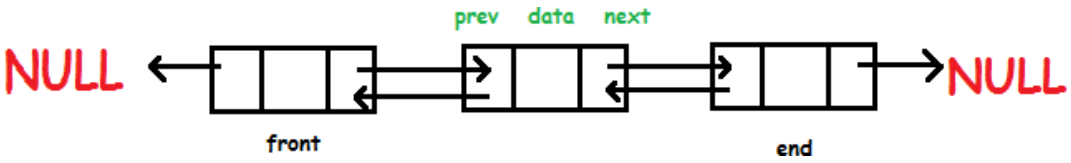
In this Lab, we will introduce the doubly Linked list and possible operations performed on doubly linked list.

Lab Objectives:

- To be able to declare a doubly linked list.
- To be able to perform fundamental operations on doubly linked list.

Introduction:

Doubly linked list is a type of linked list in which each node apart from storing its data has two links. The first link points to the previous node in the list and the second link points to the next node in the list. The first node of the list has its previous link pointing to NULL similarly the last node of the list has its next node pointing to NULL. The two links help us to traverse the list in both backward and forward direction. But storing an extra link requires some extra space.



Implementation a Node type

First we define the node.

```
struct node
{           int data;           // Data
            node *prev;         // A reference to the previous node
            node *next;         // A reference to the next node
};
```

Basic Operations on doubly linked list

Following are some of the operations that we can perform on a doubly linked list.

a) Insertion

Insertion operation of the doubly linked list inserts a new node in the linked list. Depending on the position where the new node is to be inserted, we can have the following insert operations.

- **Insertion at front –** Inserts a new node as the first node.
- **Insertion at the end –** Inserts a new node at the end as the last node.
- **Insertion before a node –** Given a node, inserts a new node before this node.
- **Insertion after a node –** Given a node, inserts a new node after this node.

b) Deletion

Deletion operation deletes a node from a given position in the doubly linked list.

- **Deletion of the first node –** Deletes the first node in the list
- **Deletion of the last node –** Deletes the last node in the list.
- **Deletion of a node given the data –** Given the data, the operation matches the data with the node data in the linked list and deletes that node.

c) Traversal

Traversal is a technique of visiting each node in the linked list. In a doubly linked list, we have two types of traversals as we have two pointers with different directions in the doubly linked list.

- **Forward traversal –** Traversal is done using the next pointer which is in the forward direction.
- **Backward traversal –** Traversal is done using the previous pointer which is the backward direction.

d) Reverse

This operation reverses the nodes in the doubly linked list so that the first node becomes the last node while the last node becomes the first node.

e) Search

Search operation in the doubly linked list is used to search for a particular node in the linked list. For this purpose, we need to traverse the list until a matching data is found.

Example program: *Create and Traverse doubly linked list.*

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

//node structure
struct Node {
    int data;
    Node* next;
    Node* prev;
};

class LinkedList {
  public:
    Node* head;
  public:
    //constructor to create an empty LinkedList
    LinkedList(){
      head = NULL;
    }

    //display the content of the list
    void PrintList() {
      Node* temp = head;
      if(temp != NULL) {
        cout<<"The list contains: ";
        while(temp != NULL) {
          cout<<temp->data<<" ";
          temp = temp->next;
        }
        cout<<endl;
      } else {
        cout<<"The list is empty.\n";
      }
    }
};

// test the code
int main() {
  //create an empty LinkedList
  LinkedList MyList;

  //Add first node.
  Node* first = new Node();
  first->data = 10;
  first->next = NULL;
  first->prev = NULL;
  //linking with head node
  MyList.head = first;

  //Add second node.
  Node* second = new Node();
  second->data = 20;
  second->next = NULL;
  //linking with first node
  second->prev = first;
  first->next = second;

  //Add third node.
  Node* third = new Node();
  third->data = 30;
  third->next = NULL;
  //linking with second node
  third->prev = second;
  second->next = third;

  //print the content of list
  MyList.PrintList();
  return 0;
}
```

```
The list contains: 10 20 30
```

Review Questions/ Exercise:

1. Write a program to insert a new node at the beginning of doubly linked list.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
    Node* prev;
};
// Function to insert a new node at the beginning
void insertAtBeginning(Node** head, int newData) {
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = *head;
    newNode->prev = NULL;
    if (*head != NULL) {
        (*head)->prev = newNode;
    }
    *head = newNode;
}
// Function to print the doubly linked list in forward direction
void printForward(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
// Function to print the doubly linked list in backward direction
void printBackward(Node* head) {
    if (head == NULL) return;
    Node* temp = head;
    while (temp->next) {
        temp = temp->next;
    }
    while (temp) {
        cout << temp->data << " ";
        temp = temp->prev;
    }
    cout << endl;
}
int main() {
    Node* head = NULL;
    // Insert nodes
    insertAtBeginning(&head, 10);
    insertAtBeginning(&head, 20);
    insertAtBeginning(&head, 30);
    cout << "Doubly Linked List (Forward): ";
    printForward(head);
    cout << "Doubly Linked List (Backward): ";
    printBackward(head);
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Doubly Linked List (Forward): 30 20 10
Doubly Linked List (Backward): 10 20 30
```

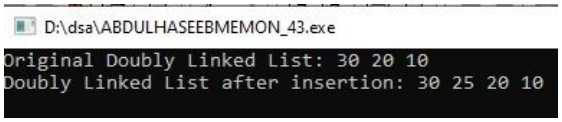
2. Write a program to insert a new node before given node in doubly linked list.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
    Node* prev;
};
// Function to insert a new node at the beginning
void insertAtBeginning(Node** head, int newData) {
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = *head;
    newNode->prev = NULL;
    if (*head != NULL) {
        (*head)->prev = newNode;
    }
    *head = newNode;
}
// Function to insert a new node before a given node
void insertBeforeNode(Node** head, int givenData, int newData) {
    Node* temp = *head;
    // Find the node with given data
    while (temp && temp->data != givenData) {
        temp = temp->next;
    }
    if (temp == NULL) {
        cout << "Node not found" << endl;
        return;
    }
    Node* newNode = new Node();
    newNode->data = newData;
    // If given node is the head node
    if (temp->prev == NULL) {
        newNode->next = temp;
        newNode->prev = NULL;
        temp->prev = newNode;
        *head = newNode;
    } else {
        newNode->next = temp;
        newNode->prev = temp->prev;
        temp->prev->next = newNode;
        temp->prev = newNode;
    }
}
// Function to print the doubly linked list in forward direction
void printForward(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    Node* head = NULL;
    // Insert nodes
    insertAtBeginning(&head, 10);
    insertAtBeginning(&head, 20);
    insertAtBeginning(&head, 30);
    cout << "Original Doubly Linked List: ";
    printForward(head);
    // Insert node before node with data 20
    insertBeforeNode(&head, 20, 25);
    cout << "Doubly Linked List after insertion: ";
    printForward(head);
    return 0;
}
```

OUTPUT:



3. Write a program to insert new node at the end of doubly linked list.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
    Node* prev;
};
// Function to insert a new node at the end
void insertAtEnd(Node** head, int newData) {
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = NULL;
    if (*head == NULL) {
        newNode->prev = NULL;
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
// Function to print the doubly linked list in forward direction
void printForward(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
// Function to print the doubly linked list in backward direction
void printBackward(Node* head) {
    if (head == NULL) return;
    Node* temp = head;
    while (temp->next) {
        temp = temp->next;
    }
    while (temp) {
        cout << temp->data << " ";
        temp = temp->prev;
    }
    cout << endl;
}
int main() {
    Node* head = NULL;
    // Insert nodes
    insertAtEnd(&head, 10);
    insertAtEnd(&head, 20);
    insertAtEnd(&head, 30);
    cout << "Doubly Linked List (Forward): ";
    printForward(head);
    cout << "Doubly Linked List (Backward): ";
    printBackward(head);
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Doubly Linked List (Forward): 10 20 30
Doubly Linked List (Backward): 30 20 10
```

4. Write a program to delete node from doubly linked list.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
    Node* prev;
};
// Function to insert a new node at the end
void insertAtEnd(Node** head, int newData) {
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = NULL;
    if (*head == NULL) {
        newNode->prev = NULL;
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
// Function to delete a node with given key
void deleteNode(Node** head, int key) {
    Node* temp = *head;
    // Find the node with given key
    while (temp && temp->data != key) {
        temp = temp->next;
    }
    if (temp == NULL) {
        cout << "Node not found" << endl;
        return;
    }
    // If node to be deleted is head node
    if (temp == *head) {
        *head = temp->next;
        if (*head != NULL) {
            (*head)->prev = NULL;
        }
    } else {
        temp->prev->next = temp->next;
        if (temp->next != NULL) {
            temp->next->prev = temp->prev;
        }
    }
    delete temp;
}
// Function to print the doubly linked list in forward direction
void printForward(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    Node* head = NULL;
    // Insert nodes
    insertAtEnd(&head, 10);
```

```
    insertAtEnd(&head, 20);
    insertAtEnd(&head, 30);
    insertAtEnd(&head, 40);
    insertAtEnd(&head, 50);
    cout << "Original Doubly Linked List: ";
    printForward(head);
    // Delete node with key 30
    deleteNode(&head, 30);
    cout << "Doubly Linked List after deletion: ";
    printForward(head);
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Original Doubly Linked List: 10 20 30 40 50
Doubly Linked List after deletion: 10 20 40 50
```

5. Write a program to show the Reverse Doubly Linked List.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
    Node* prev;
};
// Function to insert a new node at the end
void insertAtEnd(Node** head, int newData) {
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = NULL;
    if (*head == NULL) {
        newNode->prev = NULL;
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
// Function to reverse the doubly linked list
void reverseList(Node** head) {
    Node* temp = NULL;
    Node* current = *head;
    while (current != NULL) {
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;
        current = current->prev;
    }
    if (temp != NULL) {
        *head = temp->prev;
    }
```

```
}
// Function to print the doubly linked list in forward direction
void printForward(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    Node* head = NULL;
    // Insert nodes
    insertAtEnd(&head, 10);
    insertAtEnd(&head, 20);
    insertAtEnd(&head, 30);
    insertAtEnd(&head, 40);
    insertAtEnd(&head, 50);
    cout << "Original Doubly Linked List: ";
    printForward(head);
    // Reverse the doubly linked list
    reverseList(&head);
    cout << "Reversed Doubly Linked List: ";
    printForward(head);
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Original Doubly Linked List: 10 20 30 40 50
Reversed Doubly Linked List: 50 40 30 20 10
```

Name: Abdul Haseeb Memon

Roll #: 24-BS(CS)-43

Date: 10-MAY-2025

Remarks: **Subject Teacher: Faheem Shah**

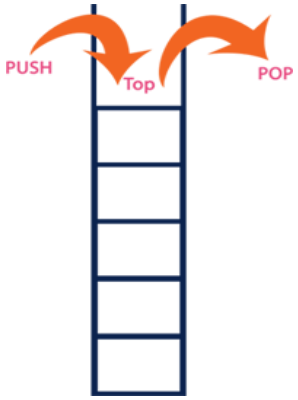
Practical # 07

Objective:

Define *Stack and* ***discuss*** *all the operations performed on Stack.*

Theory:

In this Lab, we discuss the stack data type. This data type supports Last In, First Out (LIFO) data access. Contiguous (array) and linked structures are used for implementations.



Basic Operations

Here are the minimal operations we'd need for an abstract stack (and their typical names):

- Push( ): Places a value/object on the Top of the stack.
- Pop( ): Removes a value/object from the Top of the stack and produces that value/object.
- IsEmpty( ): Reports whether the stack is empty or not.
- IsFull( ): Reports whether the stack is Full or not (for array implementation).
- Peak( ) : produces Top value/object of the stack without removing it.
- Traverse( ) : visit all elements from Top to Bottom without removing them.

Stack data structure can be implemented in two ways. They are as follows:

1. **Using Array**
2. **Using LinkedList**

Lab Objectives:

- To be able to perform fundamental operations on stack

Stack Operations using Array

A stack can be implemented using array as follows:
Before implementing actual operations: first follow the below steps to create an empty stack.

- **Step 1 -** Include all the **header files** which are used in the program and define a constant **'SIZE'** with specific value.
- **Step 2 -** Declare all the **functions** used in stack implementation.
- **Step 3 -** Create a one-dimensional array with fixed size (**int stack[SIZE]**)
- **Step 4 -** Define a integer variable **'top'** and initialize with **'-1'**. (**int top = -1**)

Step 5 - In main method, display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.

push(value) - Inserting value into the stack

In a stack, push() is a function used to insert an element into the stack. In a stack, the new element is always inserted at **top** position. Push function takes one integer value as parameter and inserts that value into the stack. We can use the following steps to push an element on to the stack.

- **Step 1 -** Check whether **stack** is **FULL**. (**top == SIZE-1**)
- **Step 2 -** If it is **FULL**, then display **"Stack is FULL!!! Insertion is not possible!!!"** and terminate the function.
- **Step 3 -** If it is **NOT FULL**, then increment **top** value by one (**top++**) and set stack[top] to value (**stack[top] = value**).

pop() - Delete a value from the Stack

In a stack, pop() is a function used to delete an element from the stack. In a stack, the element is always deleted from **top** position. Pop function does not take any value as parameter. We can use the following steps to pop an

element from the stack...

- **Step 1 -** Check whether **stack** is **EMPTY**. (**top == -1**)
- **Step 2 -** If it is **EMPTY**, then display **"Stack is EMPTY!!! Deletion is not possible!!!"** and terminate the function.
- **Step 3 -** If it is **NOT EMPTY**, then delete **stack[top]** and decrement **top** value by one (**top--**
).

display() - Displays the elements of a Stack

We can use the following steps to display the elements of a stack...

- **Step 1 -** Check whether **stack** is **EMPTY**. (**top == -1**)
- **Step 2 -** If it is **EMPTY**, then display **"Stack is EMPTY!!!"** and terminate the function.
- **Step 3 -** If it is **NOT EMPTY**, then define a variable '**i**' and initialize with top. Display **stack[i]** value and decrement **i** value by one (**i--**).
- **Step 3 -** Repeat above step until **i** value becomes '0'.

C++ program: Write C++ program to implement stack using Array.

```cpp
#include <iostream>
using namespace std;
int stack[100], n=100, top=-1;
void push(int val) {
    if(top>=n-1)
    cout<<"Stack Overflow"<<endl;
    else {
       top++;
       stack[top]=val;
    }
}
void pop() {
    if(top<=-1)
    cout<<"Stack Underflow"<<endl;
    else {
       cout<<"The popped element is "<< stack[top] <<endl;
       top--;
    }
}
void display() {
    if(top>=0) {
       cout<<"Stack elements are:";
       for(int i=top; i>=0; i--)
       cout<<stack[i]<<" ";
       cout<<endl;
    } else
    cout<<"Stack is empty";
}
int main() {
    int ch, val;
    cout<<"1) Push in stack"<<endl;
```

```
    cout<<"2) Pop from stack"<<endl;
    cout<<"3) Display stack"<<endl;
    cout<<"4) Exit"<<endl;
    do {
       cout<<"Enter choice: "<<endl;
       cin>>ch;
       switch(ch) {
          case 1: {
             cout<<"Enter value to be pushed:"<<endl;
             cin>>val;
             push(val);
             break;
          }
          case 2: {
             pop();
             break;
          }
          case 3: {
             display();
             break;
          }
          case 4: {
             cout<<"Exit"<<endl;
             break;
          }
          default: {
             cout<<"Invalid Choice"<<endl;
          }
       }
    }while(ch!=4);
     return 0;
}
```

OUTPUT:

```
1) Push in stack
2) Pop from stack
3) Display stack
4) Exit
Enter choice:
1
Enter value to be pushed:
10
Enter choice:
1
Enter value to be pushed:
20
Enter choice:
3
Stack elements are:20 10
Enter choice:
2
The popped element is 20
Enter choice:
3
Stack elements are:10
Enter choice:
4
Exit
```

Review Questions/ Exercise:

1. **Explain the procedure of stack implementation using linked list.**

Stack Implementation Using Linked List:

A stack can be implemented using a linked list by following these steps:

Node Structure:

- Define a Node structure with two members: data to store the value and next to point to the next node in the stack.

Stack Operations

1. Push: To add an element to the stack, create a new node with the given data and insert it at the beginning of the linked list. Update the top pointer to point to the new node.
2. Pop: To remove an element from the stack, check if the stack is empty. If not, store the data of the top node, update the top pointer to point to the next node, and free the memory allocated to the popped node.
3. Peek: To retrieve the top element without removing it, return the data of the top node if the stack is not empty.
4. isEmpty: To check if the stack is empty, verify if the top pointer is NULL.

Procedure
1. Initialize the top pointer to NULL.
2. Perform stack operations (push, pop, peek, isEmpty) using the linked list.

2. Write C++ program to implement the stack using linked list?

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
};
// Stack class
class Stack {
private:
    Node* top;
public:
    Stack() {
        top = NULL;
    }
    // Push element onto the stack
    void push(int data) {
        Node* newNode = new Node();
        newNode->data = data;
        newNode->next = top;
        top = newNode;
    }
    // Pop element from the stack
    int pop() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
            return -1;
        }
        int data = top->data;
        Node* temp = top;
        top = top->next;
        delete temp;
        return data;
    }
    // Peek the top element
    int peek() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
            return -1;
        }
        return top->data;
    }
    // Check if the stack is empty
    bool isEmpty() {
        return top == NULL;
    }
    // Print the stack elements
    void printStack() {
        Node* temp = top;
```

```
        while (temp) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};
int main() {
    Stack stack;
    // Push elements onto the stack
    stack.push(10);
    stack.push(20);
    stack.push(30);
    stack.push(40);
    stack.push(50);
    cout << "Stack elements: ";
    stack.printStack();
    // Peek the top element
    cout << "Top element: " << stack.peek() << endl;
    // Pop elements from the stack
    cout << "Popped element: " << stack.pop() << endl;
    cout << "Popped element: " << stack.pop() << endl;
    cout << "Stack elements after popping: ";
    stack.printStack();
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Stack elements: 50 40 30 20 10
Top element: 50
Popped element: 50
Popped element: 40
Stack elements after popping: 30 20 10
```

3. Write a C++ program that uses stack operations to convert a given infix expression into its postfix equivalent, Implement the stack using an array.

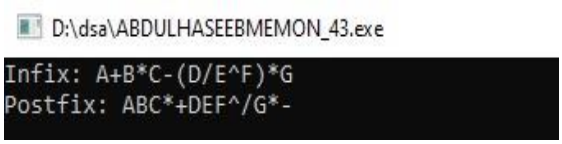
SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;
class Stack {
private:
    int top;
    char* arr;
    int capacity;
public:
    Stack(int size) {
        arr = new char[size];
        capacity = size;
        top = -1;
    }
    void push(char data) {
        if (isFull()) {
            cout << "Stack is full" << endl;
            return;
        }
        arr[++top] = data;
    }
    char pop() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
            return '\0';
        }
        return arr[top--];
    }
    char peek() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
            return '\0';
        }
        return arr[top];
    }
    bool isEmpty() {
        return top == -1;
    }
```

```
    bool isFull() {
        return top == capacity - 1;
    }
};
int getPrecedence(char operator_) {
    switch (operator_) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        default:
            return 0;
    }
}
string infixToPostfix(string infix) {
    Stack stack(infix.length());
    string postfix = "";
    for (int i = 0; i < infix.length(); i++) {
        char c = infix[i];
        if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z') || (c >= '0' && c <= '9')) {
            postfix += c;
        } else if (c == '(') {
            stack.push(c);
        } else if (c == ')') {
            while (!stack.isEmpty() && stack.peek() != '(') {
                postfix += stack.pop();
            }
            if (!stack.isEmpty() && stack.peek() == '(') {
                stack.pop();
            }
        } else {
            while (!stack.isEmpty() && getPrecedence(c) <= getPrecedence(stack.peek())) {
                postfix += stack.pop();
            }
            stack.push(c);
        }
    }
    while (!stack.isEmpty()) {
        postfix += stack.pop();
    }
    return postfix;
}
int main() {
    string infix = "A+B*C-(D/E^F)*G";
    string postfix = infixToPostfix(infix);
    cout << "Infix: " << infix << endl;
    cout << "Postfix: " << postfix << endl;
    return 0;
}
```

OUTPUT:



Name: Abdul Haseeb Memon

Roll #: 24-BS(CS)-43

Date: 10-MAY-2025

Remarks: **Subject Teacher: Faheem Shah**

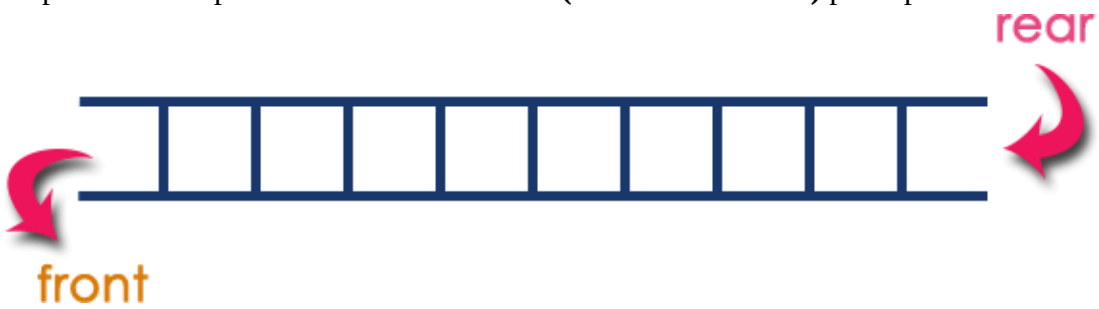
Practical # 08

Objective:

Demonstrate *Queue and discuss all the operations performed on Queue.*

Theory:

In this Lab, we discuss the Queue data type. Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends. In a queue data structure, adding and removing elements are performed at two different positions. The insertion is performed at one end and deletion is performed at another end. In a queue data structure, the insertion operation is performed at a position which is known as '**rear**' and the deletion operation is performed at a position which is known as '**front**'. In queue data structure, the insertion and deletion operations are performed based on **FIFO (First In First Out)** principle.



Basic Operations

The following operations are performed on a queue data structure:

1. **enQueue(value) - (To insert an element into the queue)**
2. **deQueue() - (To delete an element from the queue)**
3. **display() - (To display the elements of the queue)**

Queue data structure can be implemented in two ways. They are as follows

1. **Using Array**
2. **Using Linked List**

When a queue is implemented using an array, that queue can organize an only limited number of elements. When a queue is implemented using a linked list, that queue can organize an unlimited number of elements.

Queue Operations using Array

Queue data structure using array can be implemented as follows: Before we implement actual operations, first follow the below steps to create an empty queue.

Step 1 - Include all the **header files** which are used in the program and define a constant **'SIZE'** with specific value.

Step 2 - Declare all the **user defined functions** which are used in queue implementation.

Step 3 - Create a one-dimensional array with above defined SIZE (**int queue[SIZE]**)

Step 4 - Define two integer variables **'front'** and '**rear**' and initialize both with **'-1'**. (**int front = -1, rear = -1**)

Step 5 - Then implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on queue.

enQueue(value) - Inserting value into the queue

In a queue data structure, enQueue() is a function used to insert a new element into the queue. In a queue, the new element is always inserted at **rear** position. The enQueue() function takes one integer value as a parameter and inserts that value into the queue. We can use the following steps to insert an element into the queue...

Step 1 - Check whether **queue** is **FULL**. (**rear == SIZE-1**)

Step 2 - If it is **FULL**, then display **"Queue is FULL!!! Insertion is not possible!!!"** and terminate the function.

Step 3 - If it is **NOT FULL**, then increment **rear** value by one (**rear++**) and set **queue[rear]** = **value**.

deQueue() - Deleting a value from the Queue:

In a queue data structure, deQueue() is a function used to delete an element from the queue. In a queue, the element is always deleted from **front** position. The deQueue() function does not take any value as parameter. We can use the following steps to delete an element from the queue...
Step 1 - Check whether **queue** is **EMPTY**. (**front == rear**)

Step 2 - If it is **EMPTY**, then display **"Queue is EMPTY!!! Deletion is not possible!!!"** and terminate the function.

Step 3 - If it is **NOT EMPTY**, then increment the **front** value by one (**front ++**). Then display **queue[front]** as deleted element. Then check whether both **front** and **rear** are equal (**front** == **rear**), if it **TRUE**, then set both **front** and **rear** to '**-1**' (**front** = **rear** = **-1**).

display() - Displays the elements of a Queue

We can use the following steps to display the elements of a queue...
Step 1 - Check whether **queue** is **EMPTY**. (**front == rear**)

Step 2 - If it is **EMPTY**, then display **"Queue is EMPTY!!!"** and terminate the function.

Step 3 - If it is **NOT EMPTY**, then define an integer variable '**i**' and set '**i** = **front+1**'.

Step 4 - Display '**queue[i]**' value and increment '**i**' value by one (**i++**). Repeat the same until '**i**' value reaches to **rear** (**i** <= **rear**)

Lab Objectives:

- To be able to perform fundamental operations on Queue.

C++ program: Write C++ program to implement QUEUE using Array.

```
#include <iostream>
using namespace std;
int queue[100], n = 100, front = - 1, rear = - 1;
void Insert() {
   int val;
   if (rear == n - 1)
   cout<<"Queue Overflow"<<endl;
   else {
      if (front == - 1)
      front = 0;
      cout<<"Insert the element in queue : "<<endl;
      cin>>val;
      rear++;
      queue[rear] = val;
   }
}
void Delete() {
   if (front == - 1 || front > rear) {
      cout<<"Queue Underflow ";
      return ;
   } else {
      cout<<"Element deleted from queue is : "<< queue[front] <<endl;
      front++;;
   }
}
void Display() {
   if (front == - 1)
   cout<<"Queue is empty"<<endl;
   else {
      cout<<"Queue elements are : ";
      for (int i = front; i <= rear; i++)
```

```
      cout<<queue[i]<<" ";
         cout<<endl;
   }
}
int main() {
   int ch;
   cout<<"1) Insert element to queue"<<endl;
   cout<<"2) Delete element from queue"<<endl;
   cout<<"3) Display all the elements of queue"<<endl;
   cout<<"4) Exit"<<endl;
   do {
      cout<<"Enter your choice : "<<endl;
      cin>>ch;
      switch (ch) {
         case 1: Insert();
         break;
         case 2: Delete();
         break;
         case 3: Display();
         break;
         case 4: cout<<"Exit"<<endl;
         break;
         default: cout<<"Invalid choice"<<endl;
      }
   } while(ch!=4);
   return 0;
}
```

OUTPUT:

```
1) Insert element to queue
2) Delete element from queue
3) Display all the elements of queue
4) Exit
Enter your choice :
1
Insert the element in queue :
50
Enter your choice :
1
Insert the element in queue :
60
Enter your choice :
3
Queue elements are : 50 60
Enter your choice :
2
Element deleted from queue is : 50
Enter your choice :
3
Queue elements are : 60
Enter your choice :
4
Exit
```

Review Questions/ Exercise:

1. **Explain the procedure of Queue implementation using linked list?**

Queue Implementation Using Linked List:

A queue can be implemented using a linked list by following these steps:

Node Structure:

- Define a Node structure with two members: data to store the value and next to point to the next node in the queue.

Queue Operations:

1. Enqueue: To add an element to the queue, create a new node with the given data and insert it at the end of the linked list. Update the rear pointer to point to the new node.

2. Dequeue: To remove an element from the queue, check if the queue is empty. If not, store the data of the front node, update the front pointer to point to the next node, and free the memory allocated to the dequeued node.

3. Peek: To retrieve the front element without removing it, return the data of the front node if the queue is not empty.

4. isEmpty: To check if the queue is empty, verify if the front pointer is NULL.

Procedure:

1. Initialize the front and rear pointers to NULL.
2. Perform queue operations (enqueue, dequeue, peek, isEmpty) using the linked list.

Key Points:

- The front pointer points to the node that will be dequeued next.
- The rear pointer points to the last node in the queue, which is where new nodes are added.

2. Write C++ program to implement the Queue using linked list

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
};
// Queue class
class Queue {
private:
    Node* front;
    Node* rear;
public:
    Queue() {
        front = NULL;
        rear = NULL;
    }
    // Enqueue element into the queue
    void enqueue(int data) {
        Node* newNode = new Node();
        newNode->data = data;
        newNode->next = NULL;
        if (rear == NULL) {
            front = newNode;
            rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
    }
    // Dequeue element from the queue
    int dequeue() {
        if (isEmpty()) {
            cout << "Queue is empty" << endl;
            return -1;
        }
        int data = front->data;
        Node* temp = front;
        front = front->next;
        if (front == NULL) {
            rear = NULL;
        }
        delete temp;
        return data;
```

```
45      }
46      // Peek the front element
47      int peek() {
48          if (isEmpty()) {
49              cout << "Queue is empty" << endl;
50              return -1;
51          }
52          return front->data;
53      }
54      // Check if the queue is empty
55      bool isEmpty() {
56          return front == NULL;
57      }
58      // Print the queue elements
59      void printQueue() {
60          Node* temp = front;
61          while (temp) {
62              cout << temp->data << " ";
63              temp = temp->next;
64          }
65          cout << endl;
66      }
67  };
68  int main() {
69      Queue queue;
70      // Enqueue elements into the queue
71      queue.enqueue(10);
72      queue.enqueue(20);
73      queue.enqueue(30);
74      queue.enqueue(40);
75      queue.enqueue(50);
76      cout << "Queue elements: ";
77      queue.printQueue();
78      // Dequeue elements from the queue
79      cout << "Dequeued element: " << queue.dequeue() << endl;
80      cout << "Dequeued element: " << queue.dequeue() << endl;
81      cout << "Queue elements after dequeue: ";
82      queue.printQueue();
83      // Peek the front element
84      cout << "Front element: " << queue.peek() << endl;
85      return 0;
86  }
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Queue elements: 10 20 30 40 50
Dequeued element: 10
Dequeued element: 20
Queue elements after dequeue: 30 40 50
Front element: 30
```

Practical # 09

Objective:

*Discuss linear search algorithm in data structure. **Design** a C++ program for linear search algorithm.*

Theory:

In this Lab, we discuss the linear search technique. Linear search algorithm finds a given element in a list of elements with **O(n)** time complexity where **n** is total number of elements in the list. This search process starts comparing search element with the first element in the list. If both are matched then result is element found otherwise search element is compared with the next element in the list. Repeat the same until search element is compared with the last element in the list, if that last element also doesn't match, then the result is "Element not found in the list". That means, the search element is compared with element by element in the list.

Linear search is implemented using following steps...

Step 1 - Read the search element from the user.
Step 2 - Compare the search element with the first element in the list.
Step 3 - If both are matched, then display "Given element is found!!!" and terminate the function
Step 4 - If both are not matched, then compare search element with the next element in the list.
Step 5 - Repeat steps 3 and 4 until search element is compared with last element in the list.
Step 6 - If last element in the list also doesn't match, then display "Element is not found!!!" and terminate the function.

Lab Objectives:

- To be able to write C++ program for linear search algorithm.

Write a C++ program for linear Search Algorithm

```
// C++ Program to Implement Linear Search
#include<iostream>
using namespace std;
int main()
{
     int arr[10], i, num, index;
     cout<<"Enter 10 Numbers: ";
     for(i=0; i<10; i++)
          cin>>arr[i];
     cout<<"\nEnter a Number to Search: ";
     cin>>num;
     for(i=0; i<10; i++)
     {
          if(arr[i]==num)
          {
               index = i;
               break;
          }
     }
     cout<<"\nFound at Index No."<<index;
     cout<<endl;
     return 0;
}
```

```
Enter 10 Numbers:
10
11
12
1
3

6
5
22
7
8

Enter a Number to Search: 1

Found at Index No.3
```

Review Questions/ Exercise:

1. **Implement linear search algorithm using function in C++.**

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Function to perform linear search
int linearSearch(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            return i; // Return the index of the target element
        }
    }
    return -1; // Return -1 if the target element is not found
}
int main() {
    int arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 23;
    cout << "Array: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    int result = linearSearch(arr, size, target);
    if (result != -1) {
        cout << "Element " << target << " found at index " << result << endl;
    } else {
        cout << "Element " << target << " not found in the array" << endl;
    }
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Array: 2 5 8 12 16 23 38 56 72 91
Element 23 found at index 5
```

2. Implement linear search with duplicate element in C++.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Function to perform linear search with duplicate elements
void linearSearch(int arr[], int size, int target) {
    bool found = false;
    cout << "Target element " << target << " found at indices: ";
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            cout << i << " ";
            found = true;
        }
    }
    if (!found) {
        cout << "\nTarget element " << target << " not found in the array";
    }
    cout << endl;
}
int main() {
    int arr[] = {2, 5, 8, 2, 16, 2, 38, 5, 72, 2};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 2;
    cout << "Array: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    linearSearch(arr, size, target);
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Array: 2 5 8 2 16 2 38 5 72 2
Target element 2 found at indices: 0 3 5 9
```

Practical # 10

Objective:

*Discuss binary search algorithm in data structure. **Design** a C++ program for binary search algorithm.*

Theory:

In this Lab, we discuss the binary search technique. Binary search algorithm finds a given element in a list of elements with **O(log n)** time complexity where **n** is total number of elements in the list. The binary search algorithm can be used with only a sorted list of elements. That means the binary search is used only with a list of elements that are already arranged in an order. This search process starts comparing the search element with the middle element in the list. If both are matched, then the result is "element found". Otherwise, we check whether the search element is smaller or larger than the middle element in the list. If the search element is smaller, then we repeat the same process for the left sublist of the middle element. If the search element is larger, then we repeat the same process for the right sublist of the middle element. We repeat this process until we find the search element in the list or until we left with a sublist of only one element. And if that element also doesn't match with the search element, then the result is "Element not found in the list".

Binary search is implemented using following steps:

Step 1 - Read the search element from the user.
Step 2 - Find the middle element in the sorted list.
Step 3 - Compare the search element with the middle element in the sorted list.
Step 4 - If both are matched, then display "Given element is found!!!" and terminate the function.
Step 5 - If both are not matched, then check whether the search element is smaller or larger than the middle element.
Step 6 - If the search element is smaller than middle element, repeat steps 2, 3, 4 and 5 for the left sublist of the middle element.
Step 7 - If the search element is larger than middle element, repeat steps 2, 3, 4 and 5 for the right sublist of the middle element.
Step 8 - Repeat the same process until we find the search element in the list or until sublist contains only one element.
Step 9 - If that element also doesn't match with the search element, then display "Element is not found in the list!!!" and terminate the function.

Lab Objectives:

- To be able to write C++ program for binary search algorithm

C++ program: Write C++ program for binary search algorithm.

```
#include<iostream>
using namespace std;
int main()
{
    int i, arr[10], num, first, last, middle;
    cout<<"Enter 10 Elements (in ascending order): ";
    for(i=0; i<10; i++)
        cin>>arr[i];
    cout<<"\nEnter Element to be Search: ";
    cin>>num;
    first = 0;
    last = 9;
    middle = (first+last)/2;
    while(first <= last)
    {
        if(arr[middle]<num)
            first = middle+1;
        else if(arr[middle]==num)
        {
            cout<<"\nThe number, "<<num<<" found at Position "<<middle+1;
            break;
        }
        else
            last = middle-1;
        middle = (first+last)/2;
    }
    if(first>last)
        cout<<"\nThe number, "<<num<<" is not found in given Array";
    cout<<endl;
    return 0;
}
```

OUTPUT:

```
Enter 10 Elements (in ascending order): 12
13
14
15
16
17
18
19
20
21

Enter Element to be Search: 17

The number, 17 found at Position 6
```

Review Questions/ Exercise:

1. **Implement binary search algorithm using function in C++.**

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Function to perform binary search
int binarySearch(int arr[], int size, int target) {
    int left = 0;
    int right = size - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            return mid; // Return the index of the target element
        } else if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1; // Return -1 if the target element is not found
}
int main() {
    int arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 23;
    cout << "Array: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    int result = binarySearch(arr, size, target);
    if (result != -1) {
        cout << "Element " << target << " found at index " << result << endl;
    } else {
        cout << "Element " << target << " not found in the array" << endl;
    }
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Array: 2 5 8 12 16 23 38 56 72 91
Element 23 found at index 5
```

2. Implement binary search algorithm that allow user to define size of array in C++.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Function to perform binary search
int binarySearch(int arr[], int size, int target) {
    int left = 0;
    int right = size - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            return mid; // Return the index of the target element
        } else if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1; // Return -1 if the target element is not found
}
int main() {
    int size;
    cout << "Enter the size of the array: ";
    cin >> size;
    int* arr = new int[size];
    cout << "Enter " << size << " elements in ascending order:" << endl;
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }
    int target;
    cout << "Enter the target element to search: ";
    cin >> target;
    int result = binarySearch(arr, size, target);
    if (result != -1) {
        cout << "Element " << target << " found at index " << result << endl;
    } else {
        cout << "Element " << target << " not found in the array" << endl;
    }
    delete[] arr; // Deallocate memory
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Array: 2 5 8 12 16 23 38 56 72 91
Element 23 found at index 5
```

Practical # 11

Objective:

Understand three Sorting techniques (i.e., Selection Sort, Bubble Sort, and Insertion Sort) and

***Design** C++program to sort data in some specific order.*

Theory:

In this Lab, we discuss the three Sorting techniques (i.e., Selection Sort, Bubble Sort, and Insertion Sort). Sorting refers to the operation or technique of arranging and rearranging sets of data in some specific order.

Selection Sort works:

The selection sort algorithm is performed using the following steps:
Step 1 - Select the first element of the list (i.e., Element at first position in the list).
Step 2: Compare the selected element with all the other elements in the list.
Step 3: In every comparison, if any element is found smaller than the selected element (for Ascending order), then both are swapped.
Step 4: Repeat the same procedure with element in the next position in the list till the entire list is sorted.

Bubble Sort work:

The Bubble sort algorithm is performed using the following steps:
Step 1 - Bubble sort starts with very first two elements, comparing them to check which one is greater. Put larger one at higher index.
Step 2 - It takes next two values compare these values and place larger one at higher index. **Step 3 -** This process does iteratively until the largest value is not reached at the last index. Then start again from 0 (zero) index up to n-1 index.
Step 4 - The algorithm follows the same steps iteratively until elements are not sorted

Insertion Sort work:

The insertion sort algorithm is performed using the following steps:

Step 1 - Assume that first element in the list is in sorted portion and all the remaining elements are in unsorted portion.
Step 2: Take first element from the unsorted portion and insert that element into the sorted portion in the order specified.
Step 3: Repeat the above process until all the elements from the unsorted portion are moved into the sorted portion.

Lab Objectives:

- To be able to write C++ program for sorting algorithm

C++ program: Write C++ program to arrange elements in ascending order using selection sorting algorithm.

```
#include<iostream>
using namespace std;
int main()
{
    int tot, arr[50], i, j, temp, small, chk, index;
    cout<<"Enter the Size of Array: ";
    cin>>tot;
    cout<<"Enter "<<tot<<" Array Elements: ";
    for(i=0; i<tot; i++)
        cin>>arr[i];
    for(i=0; i<(tot-1); i++)
    {
        chk=0;
        small = arr[i];
        for(j=(i+1); j<tot; j++)
        {
            if(small>arr[j])
            {
                small = arr[j];
                chk++;
                index = j;
            }
        }
        if(chk!=0)
        {
            temp = arr[i];
            arr[i] = small;
            arr[index] = temp;
        }
    }
    cout<<"\nSorted Array is:\n";
      for(i=0; i<tot; i++)
          cout<<arr[i]<<" ";
      cout<<endl;
      return 0;
}
```

OUTPUT

```
Enter the Size of Array: 4
Enter 4 Array Elements: 45
25
87
31

Sorted Array is:
25 31 45 87
```

Review Questions/ Exercise:

1. **Write a C++ program that implements Selection sort algorithm to arrange a list of integers in descending order.**

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Function to perform selection sort in descending order
void selectionSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        int maxIndex = i;
        for (int j = i + 1; j < size; j++) {
            if (arr[j] > arr[maxIndex]) {
                maxIndex = j;
            }
        }
        // Swap the maximum element with the current element
        int temp = arr[i];
        arr[i] = arr[maxIndex];
        arr[maxIndex] = temp;
    }
}
// Function to print the array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << "Original array: ";
    printArray(arr, size);
    selectionSort(arr, size);
    cout << "Sorted array in descending order: ";
    printArray(arr, size);
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Original array: 64 25 12 22 11
Sorted array in descending order: 64 25 22 12 11
```

2. Write a C++ program that implements Bubble sort algorithm to arrange a list of integers in ascending order.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Function to perform bubble sort in ascending order
void bubbleSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap the elements
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
// Function to print the array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << "Original array: ";
    printArray(arr, size);
    bubbleSort(arr, size);
    cout << "Sorted array in ascending order: ";
    printArray(arr, size);
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Original array: 64 34 25 12 22 11 90
Sorted array in ascending order: 11 12 22 25 34 64 90
```

3. Write a C++ program that implements Bubble sort algorithm to arrange a list of integers in descending order.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Function to perform bubble sort in descending order
void bubbleSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (arr[j] < arr[j + 1]) {
                // Swap the elements
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
// Function to print the array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << "Original array: ";
    printArray(arr, size);
    bubbleSort(arr, size);
    cout << "Sorted array in descending order: ";
    printArray(arr, size);
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Original array: 64 34 25 12 22 11 90
Sorted array in descending order: 90 64 34 25 22 12 11
```

4. Write a C++ program that implements Insertion sort algorithm to arrange a list of integers in ascending order.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Function to perform insertion sort in ascending order
void insertionSort(int arr[], int size) {
    for (int i = 1; i < size; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
// Function to print the array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << "Original array: ";
    printArray(arr, size);
    insertionSort(arr, size);
    cout << "Sorted array in ascending order: ";
    printArray(arr, size);
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Original array: 64 34 25 12 22 11 90
Sorted array in ascending order: 11 12 22 25 34 64 90
```

5. Write a C++ program that implements Insertion sort algorithm to arrange a list of integers in descending order

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Function to perform insertion sort in descending order
void insertionSort(int arr[], int size) {
    for (int i = 1; i < size; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] < key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
// Function to print the array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << "Original array: ";
    printArray(arr, size);
    insertionSort(arr, size);
    cout << "Sorted array in descending order: ";
    printArray(arr, size);
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Original array: 64 34 25 12 22 11 90
Sorted array in descending order: 90 64 34 25 22 12 11
```

Practical # 12

Objective:

Understand three Sorting techniques (i.e., Merge Sort, Quick Sort, Radix Sort) and ***Design***

C++program to sort data in some specific order.

Theory:

In this Lab, we discuss the four Sorting techniques (i.e., Merge Sort, Quick Sort, and Radix Sort). Sorting refers to the operation or technique of arranging and rearranging sets of data in some specific order.

Merge Sort works:

Merge Sort Algorithm works in the following steps:
Step 1 - It divides the given unsorted array into two halves- left and right sub arrays.
Step 2 - The sub arrays are divided recursively.
Step 3 - This division continues until the size of each sub array becomes 1.
Step 4 - After each sub array contains only a single element, each sub array is sorted trivially.
Step 5 - Then, the merge procedure combines these trivially sorted arrays to produce a final sorted array.

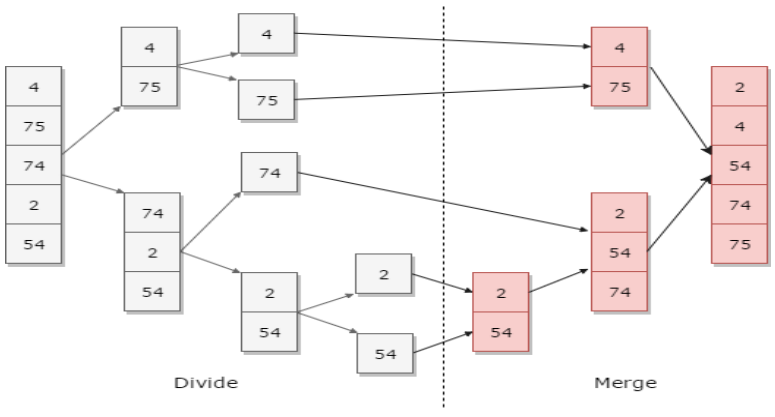


Fig: Merge Sort Technique

Quick Sort work:

In Quick sort algorithm, partitioning of the list is performed using following steps:

Step 1 - Consider the first element of the list as **pivot** (i.e., Element at first position in the list).
Step 2 - Define two variables i and j. Set i and j to first and last elements of the list respectively.
Step 3 - Increment i until list[i] > pivot then stop. **Step 4 -** Decrement j until list[j] < pivot then stop. **Step 5 -** If i < j then exchange list[i] and list[j].
Step 6 - Repeat steps 3,4 & 5 until i > j.

Step 7 - Exchange the pivot element with list[j] element.

Radix Sort work:

The Radix sort algorithm is performed using the following steps...

Step 1 - Define 10 queues each representing a bucket for each digit from 0 to 9.

Step 2 - Consider the least significant digit of each number in the list which is to be sorted.

Step 3 - Insert each number into their respective queue based on the least significant digit.

Step 4 - Group all the numbers from queue 0 to queue 9 in the order they have inserted into their respective queues.

Step 5 - Repeat from step 3 based on the next least significant digit

Step 6 - Repeat from step 2 until all the numbers are grouped based on the most significant digit.

Lab Objectives:

- To be able to write C++ program for sorting algorithm.

C++ program: Write C++ program to arrange elements in ascending order using Merge sorting algorithm.

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

void merge(int *,int, int , int );
void mergesort(int *a, int low, int high)
{
     int mid;
     if (low < high)
     {
mid=(low+high)/2;
mergesort(a,low,mid);
mergesort(a,mid+1,high);
merge(a,low,high,mid);
     }
     return;
}
// Merge sort concepts starts here
void merge(int *a, int low, int high, int mid)
{
     int i, j, k, c[50];
     i = low;
     k = low;
     j = mid + 1;
     while (i <= mid && j <= high)
     {
if (a[i] < a[j])
{
     c[k] = a[i];
     k++;
     i++;
}
else
{
     c[k] = a[j];
     k++;
     j++;
}
     }
     while (i <= mid)
     {
c[k] = a[i];
k++;
i++;
     }
     while (j <= high)
     {
c[k] = a[j];
k++;
j++;
     }
     for (i = low; i < k; i++)
     {
a[i] = c[i];
     }
}
// from main mergesort function gets called
int main()
{
     int a[30], i, b[30];
     cout<<"enter  five elements (unsorted array):\n";
     for (i = 0; i < 5; i++) { cin>>a[i];
```

```
    }
    mergesort(a, 0, 4);
    cout<<"sorted array\n";
    for (i = 0; i < 5; i++)
    {
cout<<a[i]<<"\t";
    }
}
```

OUTPUT

```
enter  five elements (unsorted array):
45
12
32
2
5
sorted array
2       5       12      32      45
```

Review Questions/ Exercise:

1. **Write a C++ program that implements Merge sort algorithm to arrange a list of integers in descending order.**

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```
#include <iostream>
using namespace std;
// Function to merge two subarrays in descending order
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int leftArr[n1], rightArr[n2];
    // Copy data to temporary arrays
    for (int i = 0; i < n1; i++) {
        leftArr[i] = arr[left + i];
    }
    for (int j = 0; j < n2; j++) {
        rightArr[j] = arr[mid + 1 + j];
    }
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (leftArr[i] > rightArr[j]) {
            arr[k] = leftArr[i];
            i++;
        } else {
            arr[k] = rightArr[j];
            j++;
        }
        k++;
    }
    // Copy remaining elements, if any
    while (i < n1) {
        arr[k] = leftArr[i];
        i++;
        k++;
    }
```

```
    while (j < n2) {
        arr[k] = rightArr[j];
        j++;
        k++;
    }
}
// Function to perform merge sort in descending order
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}
// Function to print the array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << "Original array: ";
    printArray(arr, size);
    mergeSort(arr, 0, size - 1);
    cout << "Sorted array in descending order: ";
    printArray(arr, size);
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Original array: 64 34 25 12 22 11 90
Sorted array in descending order: 90 64 34 25 22 12 11
```

2. Write a C++ program that implements Quick sort algorithm to arrange a list of integers in ascending order.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```
#include <iostream>
using namespace std;
// Function to swap two elements
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}// Function to partition the array
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // Choosing the last element as the pivot
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}
```

```cpp
// Function to perform quick sort
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivotIndex = partition(arr, low, high);
        quickSort(arr, low, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, high);
    }
}// Function to print the array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << "Original array: ";
    printArray(arr, size);
    quickSort(arr, 0, size - 1);
    cout << "Sorted array in ascending order: ";
    printArray(arr, size);
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Original array: 64 34 25 12 22 11 90
Sorted array in ascending order: 11 12 22 25 34 64 90
```

3. **Write a C++ program that implements Quick sort algorithm to arrange a list of integers in descending order**.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Function to swap two elements
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}// Function to partition the array
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // Choosing the last element as the pivot
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}
// Function to perform quick sort
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivotIndex = partition(arr, low, high);
        quickSort(arr, low, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, high);
    }
}// Function to print the array
```

```cpp
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << "Original array: ";
    printArray(arr, size);
    quickSort(arr, 0, size - 1);
    cout << "Sorted array in ascending order: ";
    printArray(arr, size);
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Original array: 64 34 25 12 22 11 90
Sorted array in ascending order: 11 12 22 25 34 64 90
```

4. Write a C++ program that implements Radix sort algorithm to arrange a list of integers in ascending order.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Function to get the maximum value in the array
int getMax(int arr[], int size) {
    int max = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}// Function to perform counting sort based on a digit
void countingSort(int arr[], int size, int exp) {
    int output[size];
    int count[10] = {0};
    // Count occurrences of each digit
    for (int i = 0; i < size; i++) {
        count[(arr[i] / exp) % 10]++;
    }
    // Calculate cumulative count
    for (int i = 1; i < 10; i++) {
        count[i] += count[i - 1];
    }
```

```
24      // Build the output array
25      for (int i = size - 1; i >= 0; i--) {
26          output[count[(arr[i] / exp) % 10] - 1] = arr[i];
27          count[(arr[i] / exp) % 10]--;
28      }
29      // Copy the output array to the original array
30      for (int i = 0; i < size; i++) {
31          arr[i] = output[i];
32      }
33  }// Function to perform radix sort
34  void radixSort(int arr[], int size) {
35      int max = getMax(arr, size);
36
37      for (int exp = 1; max / exp > 0; exp *= 10) {
38          countingSort(arr, size, exp);
39      }
40  }// Function to print the array
41  void printArray(int arr[], int size) {
42      for (int i = 0; i < size; i++) {
43          cout << arr[i] << " ";
44      }
45      cout << endl;
46  }
47  int main() {
48      int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
49      int size = sizeof(arr) / sizeof(arr[0]);
50      cout << "Original array: ";
51      printArray(arr, size);
52      radixSort(arr, size);
53      cout << "Sorted array in ascending order: ";
54      printArray(arr, size);
55      return 0;
56  }
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Original array: 170 45 75 90 802 24 2 66
Sorted array in ascending order: 2 24 45 66 75 90 170 802
```

5. Write a C++ program that implements Radix sort algorithm to arrange a list of integers in descending order.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
// Function to get the maximum value in the array
int getMax(int arr[], int size) {
    int max = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}
// Function to perform counting sort based on a digit
void countingSort(int arr[], int size, int exp) {
    int output[size];
    int count[10] = {0};
    // Count occurrences of each digit
    for (int i = 0; i < size; i++) {
        count[(arr[i] / exp) % 10]++;
    }
    // Calculate cumulative count
    for (int i = 1; i < 10; i++) {
        count[i] += count[i - 1];
    }
    // Build the output array
    for (int i = size - 1; i >= 0; i--) {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }
    // Copy the output array to the original array
    for (int i = 0; i < size; i++) {
        arr[i] = output[i];
    }
}
// Function to perform radix sort
void radixSort(int arr[], int size) {
    int max = getMax(arr, size);
    for (int exp = 1; max / exp > 0; exp *= 10) {
        countingSort(arr, size, exp);
    }
}
// Function to reverse the array
void reverseArray(int arr[], int size) {
    int start = 0;
```

```
45      int end = size - 1;
46      while (start < end) {
47          int temp = arr[start];
48          arr[start] = arr[end];
49          arr[end] = temp;
50          start++;
51          end--;
52      }
53  }
54  // Function to print the array
55  void printArray(int arr[], int size) {
56      for (int i = 0; i < size; i++) {
57          cout << arr[i] << " ";
58      }
59      cout << endl;
60  }
61  int main() {
62      int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
63      int size = sizeof(arr) / sizeof(arr[0]);
64      cout << "Original array: ";
65      printArray(arr, size);
66      radixSort(arr, size);
67      reverseArray(arr, size);
68      cout << "Sorted array in descending order: ";
69      printArray(arr, size);
70      return 0;
71  }
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Original array: 170 45 75 90 802 24 2 66
Sorted array in descending order: 802 170 90 75 66 45 24 2
```

Practical # 13

Objective:

Discuss tree data structure. ***Develop*** *a C++ program that traverse the binary tree.*

Theory:

In this Lab, we discuss the Tree data structure. A tree is a very popular non-linear data structure used in a wide range of applications. Tree is a non-linear data structure which organizes data in hierarchical structure and this is a recursive definition. A tree data structure can also be defined a collection of data (Node) which is organized in hierarchical structure recursively. A tree in which every **node can have a maximum of two children** is called **Binary Tree**. A binary tree data structure is represented using two methods:

- **Array Representation**
- **Linked List Representation**

Displaying (or) visiting order of nodes in a binary tree is called as Binary Tree Traversal. There are three types of binary tree traversals.

1. **In - Order Traversal (leftChild - root - rightChild)**
2. **Pre - Order Traversal (root - leftChild - rightChild)**
3. **Post - Order Traversal (leftChild - rightChild - root)**

1. **In - Order Traversal (leftChild - root - rightChild)**

   Following operations are carried out to traverse a binary tree using In-Order way:

   a) Traverse the left most sub-tree starting at the left external node
   b) Visit the root
   c) Traverse the right sub-tree starting at the left external node

1. Create an empty Stack S
2. Initialize current node as root
3. Push the current node to S and set Current= Current Left **until** Current = Null
4. If Current is Null and Stack is not empty then:
   a. POP the item from S
   b. Print the Popped item, Set Current Right
   c. Go to Step 3
5. If Current is Null and Stack is empty then Exit

2. Pre - Order Traversal (root - leftChild - rightChild)

Following operations are carried out to traverse a binary tree using Pre-Order way:

d) Visit the root
e) Traverse the left most sub-tree starting at the left external node
f) Traverse the right sub-tree starting at the left external node

1. Create an empty Stack S
2. Push the root on Stack
3. While the Stack is not empty
   a. POP the item from Stack and print
   b. PUSH its children in the Stack

2. Post - Order Traversal (leftChild - rightChild - root)

Following operations are carried out to traverse a binary tree using Post-Order way:

a. Traverse the left most sub-tree starting at the left external node
b. Traverse the right sub-tree starting at the left external node
c. Visit the root

1. Create an empty Stack S
2. Do following While root is not NULL
   a. PUSH root's right child and then root in stack
   b. Set root as root's left child
3. POP an item from stack and set it as root
   a. If the POPPED item has a right child and the right child is at top of stack, then remove the right child from the stack, PUSH the root back and set root as root's right child
   b. Else print root's data and set root as NULL
4. Repeat step 2 and 3 while stack is not empty

Lab Objectives:

- To be able to write C++ program for performing operations on Tree data structure.

C++ program: Write C++ program to traverse binary tree using In-order traversal method.

```
#include <iostream>
using namespace std;
/* A binary tree node has data, pointer to left child
and a pointer to right child */
struct Node {
    int data;
    struct Node *left, *right;
    Node(int data)
    {
        this->data = data;
        left = right = NULL;
    }
};
/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct Node* node)
{
    if (node == NULL)
        return;

    /* first recur on left child */
    printInorder(node->left);

    /* then print the data of node */
    cout << node->data << " ";

    /* now recur on right child */
    printInorder(node->right);
}
/* Main program */
int main()
{
    struct Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    cout << "\nInorder traversal of binary tree is \n";
    printInorder(root);
    return 0;
}
```

OUTPUT

```
Inorder traversal of binary tree is
4 2 5 1 3
```

Review Questions/ Exercise:

1. **Write a C++ program traverse the tree using pre-order traversal method.**

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
};
Node* createNode(int data) {
    Node* newNode = new Node();
    (*newNode).data = data;
    (*newNode).left = NULL;
    (*newNode).right = NULL;
    return newNode;
}
void preOrderTraversal(Node* root) {
    if (root == NULL) {
        return;
    }
    cout << (*root).data << " ";
    preOrderTraversal((*root).left);
    preOrderTraversal((*root).right);
}
int main() {
    Node* root = createNode(1);
    (*root).left = createNode(2);
    (*root).right = createNode(3);
    (*(*root).left).left = createNode(4);
    (*(*root).left).right = createNode(5);
    (*(*root).right).left = createNode(6);
    (*(*root).right).right = createNode(7);
    cout << "Pre-order traversal: ";
    preOrderTraversal(root);
    cout << endl;
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Pre-order traversal: 1 2 4 5 3 6 7
```

2. Write a C++ program traverse the tree using post-order traversal method.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
};
Node* createNode(int data) {
    Node* newNode = new Node();
    (*newNode).data = data;
    (*newNode).left = NULL;
    (*newNode).right = NULL;
    return newNode;
}
void postOrderTraversal(Node* root) {
    if (root == NULL) {
        return;
    }
    postOrderTraversal((*root).left);
    postOrderTraversal((*root).right);
    cout << (*root).data << " ";
}
int main() {
    Node* root = createNode(1);
    (*root).left = createNode(2);
    (*root).right = createNode(3);
    (*(*root).left).left = createNode(4);
    (*(*root).left).right = createNode(5);
    (*(*root).right).left = createNode(6);
    (*(*root).right).right = createNode(7);
    cout << "Post-order traversal: ";
    postOrderTraversal(root);
    cout << endl;
    return 0;
}
```

OUTPUT:

```
D:\dsa\ABDULHASEEBMEMON_43.exe
Post-order traversal: 4 5 2 6 7 3 1
```

3. Design a C++ program that uses functions to perform the following:

a. Create a binary search tree of integers

b. Search for an integer key in the above binary search tree non recursively.

c. Search for an integer key in the above binary search tree recursively.

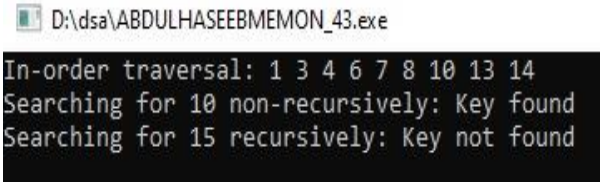
SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
};
Node* createNode(int data) {
    Node* newNode = new Node();
    (*newNode).data = data;
    (*newNode).left = NULL;
    (*newNode).right = NULL;
    return newNode;
}
Node* insertNode(Node* root, int data) {
    if (root == NULL) {
        root = createNode(data);
    } else if (data < (*root).data) {
        (*root).left = insertNode((*root).left, data);
    } else if (data > (*root).data) {
        (*root).right = insertNode((*root).right, data);
    }
    return root;
}
bool searchNonRecursive(Node* root, int key) {
    while (root != NULL) {
        if ((*root).data == key) {
            return true;
        } else if (key < (*root).data) {
            root = (*root).left;
        } else {
            root = (*root).right;
        }
    }
    return false;
}
bool searchRecursive(Node* root, int key) {
    if (root == NULL) {
        return false;
    }
    if ((*root).data == key) {
        return true;
    } else if (key < (*root).data) {
        return searchRecursive((*root).left, key);
```

```
    } else {
        return searchRecursive((*root).right, key);
    }
}
void inOrderTraversal(Node* root) {
    if (root != NULL) {
        inOrderTraversal((*root).left);
        cout << (*root).data << " ";
        inOrderTraversal((*root).right);
    }
}
int main() {
    Node* root = NULL;
    root = insertNode(root, 8);
    root = insertNode(root, 3);
    root = insertNode(root, 10);
    root = insertNode(root, 1);
    root = insertNode(root, 6);
    root = insertNode(root, 14);
    root = insertNode(root, 4);
    root = insertNode(root, 7);
    root = insertNode(root, 13);
    cout << "In-order traversal: ";
    inOrderTraversal(root);
    cout << endl;
    int key1 = 10;
    cout << "Searching for " << key1 << " non-recursively: ";
    if (searchNonRecursive(root, key1)) {
        cout << "Key found" << endl;
    } else {
        cout << "Key not found" << endl;
    }
    int key2 = 15;
    cout << "Searching for " << key2 << " recursively: ";
    if (searchRecursive(root, key2)) {
        cout << "Key found" << endl;
    } else {
        cout << "Key not found" << endl;
    }
    return 0;
}
```

OUTPUT:



Practical # 14

Objective:

*Discuss Graph data structure. **Design** a C++ program that implement traversal technique for a searching vertex in a graph.*

Theory:

In this Lab, we discuss the Graph data structure. Graph is a non-linear data structure. It contains a set of points known as nodes (or vertices) and a set of links known as edges (or Arcs). Here edges are used to connect the vertices. A graph is defined as a collection of vertices and arcs in which vertices are connected with arcs. Graph traversal is a technique used for a searching vertex in a graph. There are two graph traversal techniques and they are as follows:

1. **DFS (Depth First Search)**
2. **BFS (Breadth First Search)**

DFS (Depth First Search)

DFS traversal of a graph produces a **spanning tree** as final result. **Spanning Tree** is a graph without loops. We use **Stack data structure** with maximum size of total number of vertices in the graph to implement DFS traversal. Following steps are used to implement DFS traversal.
Step 1 - Define a Stack of size total number of vertices in the graph.
Step 2 - Select any vertex as **starting point** for traversal. Visit that vertex and push it on to the Stack.

Step 3 - Visit any one of the non-visited **adjacent** vertices of a vertex which is at the top of stack and push it on to the stack.
Step 4 - Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.
Step 5 - When there is no new vertex to visit then use **back tracking** and pop one vertex from the stack.

Step 6 - Repeat steps 3, 4 and 5 until stack becomes Empty.
Step 7 - When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

Lab Objectives:

- To be able to write C++ program for performing traversal technique on Graph data structure.
- ***C++ program:*** *Write C++ program to traverse Graph data structure using DFS traversal technique.*

```
#include <iostream>
#include <list>
using namespace std;
//graph class for DFS travesal
class DFSGraph
{
int V;    // No. of vertices
list<int> *adjList;    // adjacency list
void DFS_util(int v, bool visited[]);  // A function used by DFS
public:
    // class Constructor
DFSGraph(int V)
    {
 this->V = V;
 adjList = new list<int>[V];
    }
    // function to add an edge to graph
void addEdge(int v, int w){
adjList[v].push_back(w); // Add w to v's list.
    }

void DFS();    // DFS traversal function
};
void DFSGraph::DFS_util(int v, bool visited[])
{
    // current node v is visited
visited[v] = true;
cout << v << " ";

    // recursively process all the adjacent vertices of the node
list<int>::iterator i;
```

```
for(i = adjList[v].begin(); i != adjList[v].end(); ++i)
if(!visited[*i])
DFS_util(*i, visited);
}
// DFS traversal
void DFSGraph::DFS()
{
    // initially none of the vertices are visited
bool *visited = new bool[V];
for (int i = 0; i < V; i++)
visited[i] = false;
    // explore the vertices one by one by recursively calling  DFS_util
for (int i = 0; i < V; i++)
if (visited[i] == false)
DFS_util(i, visited);
}
int main()
{
    // Create a graph
DFSGraph gdfs(5);
gdfs.addEdge(0, 1);
gdfs.addEdge(0, 2);
gdfs.addEdge(0, 3);
gdfs.addEdge(1, 2);
gdfs.addEdge(2, 4);
gdfs.addEdge(3, 3);
gdfs.addEdge(4, 4);
cout << "Depth-first traversal for the given graph:"<<endl;
gdfs.DFS();
return 0;
}
```

OUTPUT

```
Depth-first traversal for the given graph:
0 1 2 4 3
```

Review Questions/ Exercise:

1. **Explain Breadth first search traversal technique.**

Breadth-First Search (BFS) Traversal Technique:

Breadth-First Search (BFS) is a graph traversal algorithm that explores all the nodes at a given depth level before moving on to the next level. It uses a queue data structure to keep track of the nodes to be visited.

How BFS Works:

1. Choose a starting node (also called the source node) in the graph.
2. Create a queue and enqueue the starting node.
3. While the queue is not empty, repeat the following steps:
   - Dequeue a node from the front of the queue.
   - Visit the node (i.e., process its data).
   - Enqueue all unvisited neighbors of the node.

Example:

Suppose we have a graph with the following nodes and edges:

A -> B
A -> C
B -> D
B -> E
C -> F

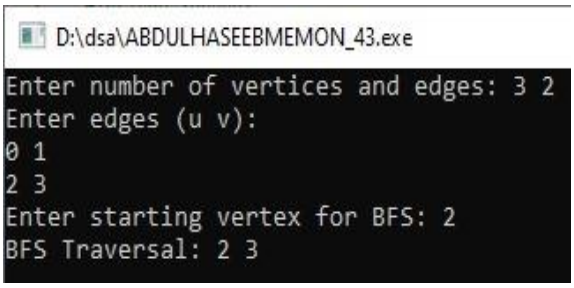
2. Write a C++ program traverse the Graph using Breadth first search traversal technique.

SOURCE CODE:

ABDULHASEEBMEMON_43.cpp

```cpp
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
const int MAX = 100;
vector<int> adj[MAX];
bool visited[MAX];
void bfs(int start) {
queue<int> q;
visited[start] = true;
q.push(start);
cout << "BFS Traversal: ";
while (!q.empty()) {
int current = q.front();
q.pop();
cout << current << " ";
for (int i = 0; i < adj[current].size(); i++) {
int neighbor = adj[current][i];
if (!visited[neighbor]) {
visited[neighbor] = true;
q.push(neighbor);   }   }   }
cout << endl;   }
int main() {
int vertices, edges;
cout << "Enter number of vertices and edges: ";
cin >> vertices >> edges;
cout << "Enter edges (u v):\n";
for (int i = 0; i < edges; i++) {
int u, v;
cin >> u >> v;
adj[u].push_back(v);
adj[v].push_back(u);    }
int start;
cout << "Enter starting vertex for BFS: ";
cin >> start;
bfs(start);
return 0;
}
```

OUTPUT:



Name: Abdul Haseeb Memon

Roll #: 24-BS(CS)-43

Date: 10-MAY-2025

Remarks: **Subject Teacher: Faheem Shah**