

OOPS AND DATA STRUCTURES LABORATORY

Ex 1: Basic C++ Programming

Name: Pevin S

Roll No: 3122233002078

Section: ECE-B

Aim: To implement a variety of basic programs in C++.

1. Write a program to swap two numbers

Code:

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

int main() {
    int a = 5, b = 10;
    cout << "You entered: a = "<<a<<, b = "<<b<<endl;
    int temp = a;
    a = b;
    b = temp;
    cout << "Swapped values are: a = "<<a<< ", b = " <<b<<endl;
    return 0;
}
```

Output:

```
You entered: a = 5, b = 10
Swapped values are: a = 10, b = 5
```

2. Find Largest Among 3 Numbers

Code:

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

int main() {
    int a, b, c;
    cout<<"Enter three numbers: ";
    cin >> a >> b >> c;
    int x;
    if (a >= b && a >= c) {
        x = a;
    } else if (b >= a && b >= c) {
        x = b;
    } else {
        x = c;
    }
    cout<<"The largest number is: "<<x<< endl;
    return 0;
}
```

```
}
```

Output:

```
Enter three numbers: 100
200
78
The largest number is: 200
```

3. Check Whether a Character is a Vowel or Consonant

Code:

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

```
int main() {
    string a;
    cout << "Enter your alphabet: ";
    cin >> a;

    if (a == "a" || a == "e" || a == "i" || a == "o" || a == "u" || a ==
"A" || a == "E" || a == "I" || a == "O" || a == "U") {
        cout << "It's a vowel";
    } else {
        cout << "It's a consonant";
    }
    return 0;
}
```

Output:

```
Enter your alphabet: a  Enter your alphabet: s
It's a vowel           It's a consonant
```

4. Find Factorial of a Number

Code:

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

```
int main() {
    int num;
    int fact = 1;
    cout<<"Enter a number: ";
    cin>>num;
    for (int i=1; i<=num; i++) {
        fact *= i;
    }

    cout<<"Factorial of "<< num <<" is "<<fact<<endl;
    return 0;
}
```

Output:

```
Enter a number: 4
Factorial of 4 is 24
```

5. Compute the power a given number to a given power.

Code:

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

int main() {
    float base, result = 1;
    int power;
    cout << "Enter base: ";
    cin >> base;
    cout << "Enter power: ";
    cin >> power;

    for(int i = 0; i < power; i++) {
        result *= base;
    }
    cout << base << "^" << power << " = " << result << endl;
    return 0;
}
```

Output:

```
Enter base: 5
Enter power: 2
5^2 = 25
```

6. Print Multiplication Table of a Number

Code:

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

int main() {
    int num;
    cout << "Enter a number: ";
    cin >> num;

    for(int i=1; i<=10; i++) {
        cout << num << "\t * \t" << i << "\t = \t" << num * i << endl;
    }
    return 0;
}
```

Output:

```

Enter a number: 7
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70

```

7. Print Fibonacci Number

Code:

```

#include <iostream>
using namespace std;

int main() {
    int n, a=0, b=1, next;
    cout<< "Enter number of terms: ";
    cin>>n;

    for(int i = 1; i <= n; i++) {
        cout<<a<<" ";
        next=a+b;
        a=b;
        b=next;
    }
    return 0;
}

```

Output:

```

Enter number of terms: 10
0 1 1 2 3 5 8 13 21 34

```

8. Display Factors of a Natural Number

Code:

```

#include <iostream>
using namespace std;
int main() {
    int num;
    cout<<"Enter a natural number: ";
    cin>>num;
    cout << "Factors of " << num << " are: ";
    for(int i=1; i<=num; i++) {
        if(num%i==0) {
            cout<< i <<" ";
        }
    }
    return 0;
}

```

Output:

```

Enter a natural number: 72
Factors of 72 are: 1 2 3 4 6 8 9 12 18 24 36 72

```

9. Reverse a Number

Code:

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

int main() {
    int num, rev = 0;
    cout << "Enter a number: ";
    cin >> num;

    while (num != 0) {
        int r = num % 10;
        rev = rev * 10 + r;
        num /= 10;
    }

    cout << "Reversed number: " << rev << endl;
    return 0;
}
```

Output:

```
Enter a number: 456
Reversed number: 654
```

10. Find LCM of Two Numbers

Code:

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

int main() {
    int num1, num2;
    cout << "Enter two numbers: ";
    cin >> num1 >> num2;

    int lcm = (num1 > num2) ? num1 : num2;

    while (true) {
        if (lcm % num1 == 0 && lcm % num2 == 0) {
            cout << "LCM of " << num1 << " and " << num2 << " is " << lcm
            << endl;
            break;
        }
        ++lcm;
    }
    return 0;
}
```

Output:

```
Enter two numbers: 6
4
LCM of 6 and 4 is 12
```

11. Display Armstrong Numbers Between 1 to 1000

Code:

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

int main() {
    int num, originalNum, r, sum, digits;
    cout << "Armstrong numbers between 1 and 1000: ";

    for(num = 1; num <= 1000; num++) {
        originalNum = num;
        sum = 0;
        digits = 0;
        // Calc no of digits
        while (originalNum != 0) {
            originalNum /= 10;
            digits++;
        }
        originalNum = num;
        // Calc armstrong number
        while (originalNum != 0) {
            r = originalNum % 10;
            sum += r * r * r;
            originalNum /= 10;
        }
        if(sum == num) {
            cout << num << " ";
        }
    }
    cout << endl;
    return 0;
}
```

Output:

```
Armstrong numbers between 1 and 1000: 1 153 370 371 407
```

12. Check Whether a Number is Prime or Not

Code:

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

int main() {
    int num;
    cout << "Enter a number: ";
    cin >> num;
    if (num <= 1) {
        cout << num << " is not a prime number." << endl;
    }
}
```

```

    } else {
        bool isPrime=true;
        for (int i=2; i<=num / 2; i++) {
            if (num%i==0) {
                isPrime=false;
                break;
            }
        }
        if (isPrime) {
            cout << num << " is a prime number." << endl;
        } else {
            cout << num << " is not a prime number." << endl;
        }
    }
    return 0;
}

```

Output:

```

Enter a number: 2
2 is a prime number.

```

```

Enter a number: 6
6 is not a prime number.

```

13. Print Diamond Pattern

Code:

```

#include <iostream>
using namespace std;

int main() {
    int n = 5;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n - i; j++) {
            cout << " ";
        }
        for (int j = 1; j <= i; j++) {
            cout << "* ";
        }
        cout << endl;
    }

    for (int i = n - 1; i >= 1; i--) {
        for (int j = 1; j <= n - i; j++) {
            cout << " ";
        }
        for (int j = 1; j <= i; j++) {
            cout << "* ";
        }
        cout << endl;
    }
    return 0;
}

```

Output:

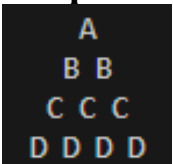


14. Print Character Pattern

Code:

```
#include <iostream>
using namespace std;
int main() {
    int n = 5;
    for (int i=1; i<=n; i++) {
        for (int j=1; j<=n-i; j++) {
            cout << " ";
        }
        for (int k=1; k<=i; k++) {
            cout << char('A' + i - 1) << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Output:



15. Pascal's Triangle

Code:

```
#include <iostream>
using namespace std;
int main() {
    int row;
    cout << "Enter the no of rows:";
    cin >> row;
    for (int i=0; i<row; i++) {
        for (int j=0; j<row-i-1; j++) {
            cout << " ";
        }

        int x = 1;
        for (int j = 0; j <= i; j++) {
            cout << x << " ";
            x = x * (i - j) / (j + 1);
        }
    }
}
```



```
        cout << endl;  
    }  
    return 0;  
}
```

Output:

```
  1  
 1 1  
1 2 1  
1 3 3 1
```

Result: Thus, various basic programs were successfully implemented in C++.

OOPS AND DATA STRUCTURES LABORATORY

Ex 2: Arrays and Strings

Name: Pevin S

Roll No: 3122233002078

Section: ECE-B

Aim: To implement programs using array and string in C++.

1. Find the Minimum and Maximum Element of an Array

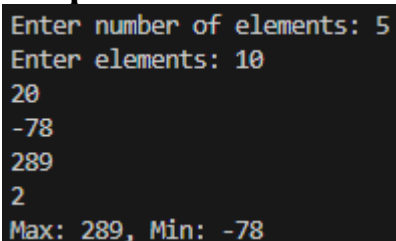
Code:

```
#include <iostream>
using namespace std;
int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;
    int arr[n];

    cout << "Enter elements: ";
    for (int i = 0; i < n; i++) cin >> arr[i];

    int max = arr[0], min = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > max) max = arr[i];
        if (arr[i] < min) min = arr[i];
    }
    cout << "Max: " << max << ", Min: " << min << endl;
    return 0;
}
```

Output:



```
Enter number of elements: 5
Enter elements: 10
20
-78
289
2
Max: 289, Min: -78
```

2. Calculate the average of all elements present in the array

Code:

```
#include <iostream>
using namespace std;
int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;
```

```

int arr[n];

cout << "Enter elements: ";
int sum = 0;
for (int i = 0; i < n; i++) {
    cin >> arr[i];
    sum += arr[i];
}
cout << "Average: " << (float)sum / n << endl;
return 0;
}

```

Output:

```

Enter three numbers: 100
200
78
The largest number is: 200

```

3. Merge two arrays

Code:

```

#include <iostream>
using namespace std;

int main() {
    int n1, n2;
    cout << "Enter the number of elements in the first array: ";
    cin >> n1;

    int arr1[n1];
    cout << "Enter the elements of the first array: ";
    for(int i = 0; i < n1; i++)
        cin >> arr1[i];

    cout << "Enter the number of elements in the second array: ";
    cin >> n2;

    int arr2[n2];
    cout << "Enter the elements of the second array: ";
    for(int i = 0; i < n2; i++)
        cin >> arr2[i];

    int merged[n1 + n2];
    for(int i = 0; i < n1; i++)
        merged[i] = arr1[i];
    for(int i = 0; i < n2; i++)
        merged[n1 + i] = arr2[i];

    cout << "Merged array: ";
    for(int i = 0; i < n1 + n2; i++)
        cout << merged[i] << " ";
    return 0;
}

```

```
}
```

Output:

```
Enter the number of elements in the first array: 4
Enter the elements of the first array: 1
2
3
4
Enter the number of elements in the second array: 3
Enter the elements of the second array: 5
6
7
Merged array: 1 2 3 4 5 6 7
```

4. Find transpose of a matrix

Code:

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

int main() {
    int rows, cols;
    cout << "Enter the number of rows and columns: ";
    cin >> rows >> cols;

    int matrix[rows][cols];
    cout << "Enter the elements of the matrix: ";
    for(int i = 0; i < rows; i++)
        for(int j = 0; j < cols; j++)
            cin >> matrix[i][j];

    cout << "Transpose of the matrix: " << endl;
    for(int i = 0; i < cols; i++) {
        for(int j = 0; j < rows; j++)
            cout << matrix[j][i] << " ";
        cout << endl;
    }
    return 0;
}
```

Output:

```
Enter the number of rows and columns: 2
3
Enter the elements of the matrix: 10
20
30
40
50
60
Transpose of the matrix:
10 40
20 50
30 60
```

5. Find the determinant of a matrix

Code:

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

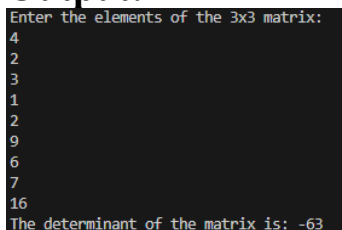
```

int main() {
    int matrix[3][3];
    int determinant = 0;
    cout << "Enter the elements of the 3x3 matrix:" << endl;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            cin >> matrix[i][j];
        }
    }
    determinant = matrix[0][0] * (matrix[1][1] * matrix[2][2] -
        matrix[1][2] * matrix[2][1]) -
        matrix[0][1] * (matrix[1][0] * matrix[2][2] -
        matrix[1][2] * matrix[2][0]) +
        matrix[0][2] * (matrix[1][0] * matrix[2][1] -
        matrix[1][1] * matrix[2][0]);

    cout << "The determinant of the matrix is: " << determinant << endl;
    return 0;
}

```

Output:



```

Enter the elements of the 3x3 matrix:
4
2
3
1
2
9
6
7
16
The determinant of the matrix is: -63

```

6. Multiply two matrices

Code:

```

#include <iostream>
using namespace std;

int main() {
    int r1, c1, r2, c2;
    cout << "Enter the number of rows and columns for the first matrix: ";
    cin >> r1 >> c1;
    cout << "Enter the number of rows and columns for the second matrix: ";
    cin >> r2 >> c2;

    if (c1 != r2) {
        cout << "Matrix multiplication not possible." << endl;
        return 0;
    }

    int matrix1[r1][c1], matrix2[r2][c2], result[r1][c2];

    cout << "Enter the elements of the first matrix: ";
    for (int i = 0; i < r1; i++)
        for (int j = 0; j < c1; j++)
            cin >> matrix1[i][j];

```

```

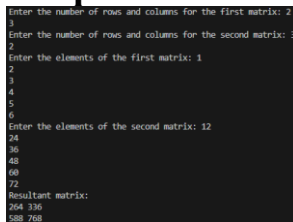
    cout << "Enter the elements of the second matrix: ";
    for (int i = 0; i < r2; i++)
        for (int j = 0; j < c2; j++)
            cin >> matrix2[i][j];

    for (int i = 0; i < r1; i++)
        for (int j = 0; j < c2; j++) {
            result[i][j] = 0;
            for (int k = 0; k < c1; k++)
                result[i][j] += matrix1[i][k] * matrix2[k][j];
        }

    cout << "Resultant matrix: " << endl;
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++)
            cout << result[i][j] << " ";
        cout << endl;
    }
    return 0;
}

```

Output:



```

Enter the number of rows and columns for the first matrix: 2
3
Enter the number of rows and columns for the second matrix: 3
2
Enter the elements of the first matrix: 1
2
3
4
5
6
Enter the elements of the second matrix: 12
24
36
48
60
72
Resultant matrix:
268 336
588 768

```

7. Rotate elements of matrix

Code:

```

#include <iostream>
using namespace std;

void rotatematrix(int m, int n, int mat[3][3]) {
    int row = 0, col = 0;
    int prev, curr;
    while (row < m && col < n) {
        if (row + 1 == m || col + 1 == n)
            break;
        prev = mat[row + 1][col];
        for (int i = col; i < n; i++) {
            curr = mat[row][i];
            mat[row][i] = prev;
            prev = curr;
        }
        row++;
        for (int i = row; i < m; i++) {
            curr = mat[i][n - 1];
            mat[i][n - 1] = prev;
            prev = curr;
        }
    }
}

```

```

    }
    n--;
    if (row < m) {
        for (int i = n - 1; i >= col; i--) {
            curr = mat[m - 1][i];
            mat[m - 1][i] = prev;
            prev = curr;
        }
    }
    m--;
    if (col < n) {
        for (int i = m - 1; i >= row; i--) {
            curr = mat[i][col];
            mat[i][col] = prev;
            prev = curr;
        }
    }
    col++;
}
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++)
        cout << mat[i][j] << " ";
    cout << endl;
}
}

int main() {
    int a[3][3] = {{1, 2, 3},
                   {4, 5, 6},
                   {7, 8, 9}};
    rotatematrix(3, 3, a);
    return 0;
}

```

Output:

```

4 1 2
7 5 3
8 9 6

```

8. Compare two strings lexicographically

Code:

```

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

int main() {
    char str1[100], str2[100];
    cout << "Enter the first string: ";
    cin >> str1;
    cout << "Enter the second string: ";

```

```

    cin >> str2;

    int result = strcmp(str1, str2);
    if(result == 0)
        cout << "Strings are equal." << endl;
    else if(result < 0)
        cout << "First string is less than the second string." << endl;
    else
        cout << "First string is greater than the second string." <<
endl;
    return 0;
}

```

Output:

```

Enter the first string: Hello
Enter the second string: World
First string is less than the second string.

```

```

Enter the first string: Hello
Enter the second string: Hello
Strings are equal.

```

```

Enter the first string: SSN
Enter the second string: NSS
First string is greater than the second string.

```

9. Reverse a string

Code:

```

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

int main() {
    char str[100];
    cout << "Enter a string: ";
    cin >> str;

    int n = strlen(str);
    for(int i = 0; i < n/2; i++)
        swap(str[i], str[n-i-1]);

    cout << "Reversed string: " << str << endl;
    return 0;
}

```

Output:

```

Enter a string: OOPS
Reversed string: SP00

```

10. Check whether string is pangram

Code:

```

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

int main() {
    char str[100];
    bool mark[26] = {false};
    int index;

```



```

cout << "Enter a string: ";
cin.getline(str, 100);

int n = strlen(str);
for(int i = 0; i < n; i++) {
    if(str[i] >= 'A' && str[i] <= 'Z')
        index = str[i] - 'A';
    else if(str[i] >= 'a' && str[i] <= 'z')
        index = str[i] - 'a';
    else
        continue;
    mark[index] = true;
}

for(int i = 0; i < 26; i++) {
    if(!mark[i]) {
        cout << "The string is not a pangram." << endl;
        return 0;
    }
}

cout << "The string is a pangram." << endl;
return 0;
}

```

Output:

Enter a string: SSN	Enter a string: The quick brown fox jumps over a lazy dog
The string is not a pangram.	The string is a pangram.

11. Search an element in an array

Code:

```

#include <iostream>
using namespace std;

int main() {
    int n, e, found = 0;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements: ";
    for(int i = 0; i < n; i++)
        cin >> arr[i];

    cout << "Enter element to search: ";
    cin >> e;

    for(int i = 0; i < n; i++) {
        if(arr[i] == e) {

```

```

        cout << "Element found at index " << i << endl;
        found = 1;
        break;
    }
}

if(!found)
    cout << "Element not found." << endl;

return 0;
}

```

Output:

```

Enter elements: 10
20
30
40
50
Enter element to search: 40
Element found at index 3

```

12. Search an element in an array (binary search)

Code:

```

#include <iostream>
using namespace std;
int main() {
    int num;
    cout << "Enter a number: ";
    cin >> num;
    if (num <= 1) {
        cout << num << " is not a prime number." << endl;
    } else {
        bool isPrime = true;
        for (int i = 2; i <= num / 2; i++) {
            if (num % i == 0) {
                isPrime = false;
                break;
            }
        }
        if (isPrime) {
            cout << num << " is a prime number." << endl;
        } else {
            cout << num << " is not a prime number." << endl;
        }
    }
    return 0;
}

```

Output:

```

Enter number of elements: 4
Enter sorted elements: 1
2
3
4
Enter element to search: 2
Element found at index 1

```

13. Sort an array (Selection Sort)

Code:

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

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

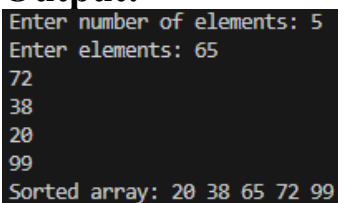
    int arr[n];
    cout << "Enter elements: ";
    for(int i = 0; i < n; i++)
        cin >> arr[i];

    for(int i = 0; i < n-1; i++) {
        int minIndex = i;
        for(int j = i+1; j < n; j++) {
            if(arr[j] < arr[minIndex])
                minIndex = j;
        }
        swap(arr[i], arr[minIndex]);
    }

    cout << "Sorted array: ";
    for(int i = 0; i < n; i++)
        cout << arr[i] << " ";

    return 0;
}
```

Output:

A screenshot of a terminal window showing the output of the Selection Sort program. The text is as follows:

```
Enter number of elements: 5
Enter elements: 65
72
38
20
99
Sorted array: 20 38 65 72 99
```

14. Sort an array (Bubble Sort)

Code:

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

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements: ";
```

```

for(int i = 0; i < n; i++)
    cin >> arr[i];

for(int i = 0; i < n-1; i++) {
    for(int j = 0; j < n-i-1; j++) {
        if(arr[j] > arr[j+1])
            swap(arr[j], arr[j+1]);
    }
}

cout << "Sorted array: ";
for(int i = 0; i < n; i++)
    cout << arr[i] << " ";

return 0;
}

```

Output:

```

Enter number of elements: 5
Enter elements: 78
90
128
49
2
Sorted array: 2 49 78 90 128

```

15. Decimal to Hexadecimal conversion

Code:

```

#include <iostream>
using namespace std;

int main() {
    int decimal;
    cout << "Enter decimal number: ";
    cin >> decimal;

    cout << "Hexadecimal: " << hex << decimal << endl;
    return 0;
}

```

Output:

```

Enter decimal number: 14
Hexadecimal: e

```

Result: Thus, various programs are implemented using arrays and strings in C++.

OOPS AND DATA STRUCTURES LABORATORY

Ex 3: Classes

Name: Pevin S

Roll No: 3122233002078

Section: ECE-B

Aim: To implement various C++ classes demonstrating object-oriented programming concepts, including constructors, member functions, and method overloading.

1. Write a C++ program to implement a class called Circle that has private member variables for radius. Include member functions to calculate the circle's area and circumference.

Code:

```
#include <iostream>
using namespace std;
class Circle {
    float radius;
public:
    Circle(float r) : radius(r) {}

    void setRadius(float r) {
        radius = r;
    }

    void calcArea () {
        float area = radius*radius;
        cout << "Area: "<<area<<endl;
    }

    void calcCircumference () {
        float c = 3.14 * 2*radius;
        cout << "Circumference: " <<c;
    }
};

int main () {
    float r;
    Circle circle1(r);
    cout << "Enter radius: ";
    cin >> r;
    circle1.setRadius(r);
    circle1.calcArea();
    circle1.calcCircumference();
}
```

Output:

```
Enter radius: 15
Area: 225
Circumference: 94.2
=====
```

2. Write a C++ program to create a class called Rectangle that has private member variables for length and width. Implement member functions to calculate the rectangle's area and perimeter. (Use Constructor)

Code:

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

class Rectangle {
    float length, width;

public:
    Rectangle(float l, float w) {
        length = l;
        width = w;
    }

    void calcArea() {
        float area = length * width;
        cout<<"Area: "<<area<<endl;
    }

    void calcPerimeter() {
        float p = 2 * (length + width);
        cout<<"Perimeter: "<<p;
    }
};

int main () {
    float l,w;
    cout<<"Enter length and width of the rectangle: ";
    cin>>l>>w;
    Rectangle rect1(l,w);
    rect1.calcArea();
    rect1.calcPerimeter();
}
```

Output:

```
Enter length and width of the rectangle: 20
15
Area: 300
Perimeter: 70
=====
```

3. Write a C++ program to implement a class called BankAccount that has private member variables for account number and balance. Include member functions to deposit and withdraw money from the account. (Use Constructor)

Code:

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

class BankAccount {
    int AccNo;
    float balance;

public:
    BankAccount(int a, float b) {
        AccNo = a;
        balance = b;
    }

    void display() {
        cout<<"Updated balance: "<<balance;
    }

    void deposit(float amt) {
        balance += amt;
        display();
    }

    void withdraw(float amt) {
        if (amt <= balance) {
            balance -= amt;
        } else {
            cout << "Balance Insufficient!" << endl;
        }
        display();
    }
};

int main() {
    int accountNumber;
    float balance, depositAmt, withdrawAmt;

    cout << "Enter account number: ";
    cin >> accountNumber;
    cout << "Enter initial balance: ";
    cin >> balance;

    BankAccount account(accountNumber, balance);

    cout << "Enter deposit amount: ";
    cin >> depositAmt;
    account.deposit(depositAmt);
```

```

        cout << "\nEnter withdrawal amount: ";
        cin >> withdrawAmt;
        account.withdraw(withdrawAmt);
        return 0;
}

```

Output:

```

Enter account number: 078
Enter initial balance: 10000
Enter deposit amount: 2000
Updated balance: 12000
Enter withdrawal amount: 8000
Updated balance: 4000
=====

```

4. Write a C++ program to create a class called Triangle that has private member variables for the lengths of its three sides. Implement member functions to determine if the triangle is equilateral, isosceles, or scalene. (Use Constructor)

Code:

```

#include <iostream>
using namespace std;
class Triangle {
    int l1,l2,l3;
public:
    Triangle () {
        cout<<"Enter length of sides of triangle: ";
        cin>>l1>>l2>>l3;
    }
    void typeCheck() {
        if (l1==l2 && l2==l3) {
            cout<<"Equilateral Triangle";
        }
        else if (l1==l2 || l2==l3 || l1==l3) {
            cout<<"Isosceles Triangle";
        }
        else {
            cout<<"Scalene Triangle";
        }
    }
};

int main() {
    Triangle tri1;
    tri1.typeCheck();
    return 0;
}

```

Output:

```

Enter length of sides of triangle: 12
11
10
Scalene Triangle

```


5. Write a C++ program to implement a class called Date that has private member variables for day, month, and year. Include member functions to set and get these variables, as well as to validate if the date is valid.

Code:

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

class Date {
private:
    int day;
    int month;
    int year;

    bool isLeapYear(int year) {
        return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
    }

    bool isValidDate(int d, int m, int y) {
        if (y < 1 || m < 1 || m > 12 || d < 1) return false;

        int daysInMonth[] = { 31, isLeapYear(y) ? 29 : 28, 31, 30, 31,
30, 31, 31, 30, 31, 30, 31 };

        return d <= daysInMonth[m - 1];
    }

public:
    Date() : day(0), month(0), year(0) {}

    bool setDate(int d, int m, int y) {
        if (isValidDate(d, m, y)) {
            day = d;
            month = m;
            year = y;
            return true;
        } else {
            return false;
        }
    }

    int getDay() const {
        return day;
    }

    int getMonth() const {
        return month;
    }

    int getYear() const {
        return year;
    }
}
```

```

    }

    void displayDate() const {
        if (day != 0 && month != 0 && year != 0) {
            cout << day << "/" << month << "/" << year << endl;
        } else {
            cout << "No valid date set." << endl;
        }
    }
};

int main() {
    Date date;
    int day, month, year;

    cout << "Enter day: ";
    cin >> day;
    cout << "Enter month: ";
    cin >> month;
    cout << "Enter year: ";
    cin >> year;

    if (date.setDate(day, month, year)) {
        cout << "The date is: ";
        date.displayDate();
    } else {
        cout << "Invalid date entered." << endl;
    }

    return 0;
}

```

Output:

```

Enter day: 30
Enter month: 02
Enter year: 2023
Invalid date entered.

```

6. Write a class with two overloaded constructors. The first constructor should initialize a Person object with a name and age, while the second constructor should initialize it with only a name, assuming a default age of 0.

Code:

```

#include <iostream>
using namespace std;

class Person {
private:
    string name;
    int age;

```

```

public:
    Person(string name, int age) : name(name), age(age) {
        cout << "Name: " << name << ", Age: " << age << endl;
    }

    Person(string name) : name(name), age(0) {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    Person p1("Pevin", 19);
    Person p2("Pevin");

    return 0;
}

```

Output:

```

Name: Pevin, Age: 19
Name: Pevin, Age: 0

```

7. Create a Rectangle class with overloaded constructors for different ways to initialize the rectangle: width and height, only width (with height as 1), and no parameters (with default width and height as 1).

Code:

```

#include <iostream>
using namespace std;

class Rectangle {
private:
    int width, height;

public:
    Rectangle(int w, int h) : width(w), height(h) {}

    Rectangle(int w) : width(w), height(1) {}

    Rectangle() : width(1), height(1) {}

    friend istream& operator>>(istream& is, Rectangle& r) {
        is >> r.width >> r.height;
        return is;
    }

    void display() const {
        cout << "Width: " << width << ", Height: " << height << endl;
    }
};

```

```

int main() {
    Rectangle rect1(5, 10);
    Rectangle rect2(5);
    Rectangle rect3;

    cout << "Rectangle 1 dimensions: ";
    rect1.display();
    cout << "Rectangle 2 dimensions: ";
    rect2.display();
    cout << "Rectangle 3 dimensions: ";
    rect3.display();
    Rectangle rect4;
    cout << "Enter width and height: ";
    cin >> rect4;
    cout << "Rectangle 4 dimensions: ";
    rect4.display();

    return 0;
}

```

Output:

```

Rectangle 1 dimensions: Width: 5, Height: 10
Rectangle 2 dimensions: Width: 5, Height: 1
Rectangle 3 dimensions: Width: 1, Height: 1
Enter width and height: 20
10
Rectangle 4 dimensions: Width: 20, Height: 10

```

8. Implement a class Matrix with overloaded constructors to handle different matrix dimensions: 2x2, 3x3, and n x m dimensions. Ensure that the matrix is initialized with zeroes.

Code:

```

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

class Matrix {
private:
    vector<vector<int> > mat;
    int rows, cols;

public:
    Matrix() : rows(2), cols(2) {
        initializeMatrix();
    }

    Matrix(int size) : rows(size), cols(size) {
        initializeMatrix();
    }
}

```

```

Matrix(int n, int m) : rows(n), cols(m) {
    initializeMatrix();
}

void initializeMatrix() {
    int value = 1;
    mat.resize(rows, vector<int>(cols));
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            mat[i][j] = value++;
        }
    }
}

void display() const {
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            cout << mat[i][j] << " ";
        }
        cout << endl;
    }
}

};

int main() {
    Matrix mat2x2;
    Matrix mat3x3(3);
    Matrix matNxM(4, 5);

    cout << "2x2 Matrix:" << endl;
    mat2x2.display();
    cout << "3x3 Matrix:" << endl;
    mat3x3.display();
    cout << "4x5 Matrix:" << endl;
    matNxM.display();
    return 0;
}

```

Output:

```

2x2 Matrix:
1 2
3 4
3x3 Matrix:
1 2 3
4 5 6
7 8 9
4x5 Matrix:
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20

```

9. Modify the following Book class to include an overloaded constructor that initializes with title, author, and publication year. Additionally, add a constructor that initializes only the title, setting default values for the author and publication year.

Code:

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

class Book {
private:
    string title;
    string author;
    int publicationYear;

public:
    Book(string t, string a, int pY) : title(t), author(a),
    publicationYear(pY) {}

    Book(string t) : title(t), author("Anonymous"), publicationYear(2000)
    {}

    void display() const {
        cout << "Title: " << title << ", Author: " << author << ",
    Publication Year: " << publicationYear << endl;
    }
};

int main() {
    Book book1("Atomic Habits", "James Clear", 2018);
    Book book2("Rich Dad Poor Dad");

    cout << "Book 1 details: ";
    book1.display();
    cout << "Book 2 details: ";
    book2.display();

    return 0;
}
```

Output:

```
Enter the name of the first person: Pevin
Enter the age of the first person: 19
Enter the name of the second person: Colpitts
Enter the age of the second person: 84
Person 1: Name: Pevin, Age: 19
Person 2: Name: Colpitts, Age: 84
```

10. Write a C++ class Math Operations that has overloaded add functions for two integers, two doubles, and an integer and a double. Implement these functions to return the sum of the arguments.

Code:

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

class MathOperations {
public:
    //overloaded add functions
    //two int
    int add(int a, int b) {
        return a + b;
    }

    //two doubles
    double add(double a, double b) {
        return a + b;
    }

    //integer and double
    double add(int a, double b) {
        return a + b;
    }

    //double and integer
    double add(double a, int b) {
        return a + b;
    }
};

int main() {
    MathOperations mathOps;

    int intResult = mathOps.add(3, 4);
    double doubleResult = mathOps.add(2.5, 3.7);
    double mixedResult1 = mathOps.add(5, 6.2);
    double mixedResult2 = mathOps.add(4.1, 7);

    cout << "Sum of 3 and 4 (int): " << intResult << endl;
    cout << "Sum of 2.5 and 3.7 (double): " << doubleResult << endl;
    cout << "Sum of 5 (int) and 6.2 (double): " << mixedResult1 << endl;
    cout << "Sum of 4.1 (double) and 7 (int): " << mixedResult2 << endl;

    return 0;
}
```

Output:

```
Sum of 3 and 4 (int): 7
Sum of 2.5 and 3.7 (double): 6.2
Sum of 5 (int) and 6.2 (double): 11.2
Sum of 4.1 (double) and 7 (int): 11.1
```

11. Create a Logger class with overloaded log functions that handle different numbers of parameters. Implement one log function for a single integer, another for an integer and a string, and another for three integers.

Code:

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

class Logger {
public:
    void log(int a) {
        cout << "Log: " << a << endl;
    }

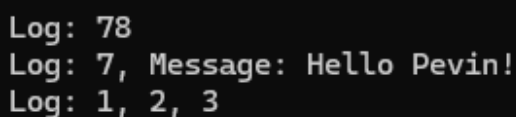
    void log(int a, const string& message) {
        cout << "Log: " << a << ", Message: " << message << endl;
    }

    void log(int a, int b, int c) {
        cout << "Log: " << a << ", " << b << ", " << c << endl;
    }
};

int main() {
    Logger logger;
    logger.log(78);
    logger.log(7, "Hello Pevin! ");
    logger.log(1, 2, 3);

    return 0;
}
```

Output:

A screenshot of a terminal window with a black background and white text. It displays the output of the C++ program: 'Log: 78', 'Log: 7, Message: Hello Pevin!', and 'Log: 1, 2, 3' on three separate lines.

```
Log: 78
Log: 7, Message: Hello Pevin!
Log: 1, 2, 3
```

Result:

Thus, a range of basic classes were successfully implemented in C++, showcasing fundamental OOP principles.

OOPS AND DATA STRUCTURES LABORATORY

Ex 4: Operator Overloading

Name: Pevin S

Roll No: 3122233002078

Section: ECE-B

Aim: To demonstrate operator overloading in C++ by creating various classes that implement unary and binary operators for different functionalities.

1. Create a class Temperature that represents a temperature in Celsius. Overload the unary prefix increment (++) operator to increase the temperature by 1 degree Celsius. Extend the Temperature class to support the postfix increment (++) operator. Implement this operator using a dummy int parameter to distinguish it from the prefix version. Write a main function to test both prefix and postfix increment operators and display function to show the corresponding temperature.

Code:

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

class Temperature {
private:
    int celsius; //temp

public:
    Temperature(int temp = 0) : celsius(temp) {} //constructor
    initialised with default value

    //prefix increment operator
    void operator++() {
        ++celsius;
    }

    // postfix increment operator
    void operator++(int) {
        int oldTemp = celsius;
        ++celsius;
        cout << "Postfix increment (former value): " << oldTemp << "°C"
<< endl;
    }

    // Display temp
    void display() const {
        cout << "Temperature: " << celsius << "°C" << endl;
    }
};

int main() {
```

```

Temperature temp(25);

cout << "Initial temperature: ";
temp.display();

//prefix increment
++temp;
cout << "After prefix increment: ";
temp.display();

//postfix increment
temp++;
cout << "After postfix increment (latter value): ";
temp.display();

return 0;
}

```

Output:

```

Initial temperature: Temperature: 25C
After prefix increment: Temperature: 26C
Postfix increment (former value): 26C
After postfix increment (latter value): Temperature: 27C

```

2. Create a class BankAccount that has a private integer data member balance. Overload the unary prefix decrement (--) operator to decrease the balance by 10 units. Implement a display function to show the current balance.

Code:

```

#include <iostream>
using namespace std;

class BankAccount {
private:
    int balance;

public:
    BankAccount(int bal) : balance(bal) {}

    void operator--() {
        balance -= 10;
        cout << "Your current balance: " << balance << endl;
    }
};

int main() {
    int bal;

    while (true) {

```

```

        cout << "Enter your account balance: ";
        cin >> bal;

        if (bal >= 10) {
            break;
        } else {
            cout << "Enter valid balance (>= 10)\n";
        }
    }

    BankAccount acc(bal);
    --acc; //unary decrement operator

    return 0;
}

```

Output:

```

Enter your account balance: 100
Your current balance: 90

```

3. Create a class Vector3D that represents a 3D vector with x, y, and z components. Overload the unary negation (-) operator to negate all three components of the vector. Implement a display function to show the vector components.

Code:

```

#include <iostream>
using namespace std;

class Vector3D {
private:
    int x, y, z;

public:
    Vector3D(int a, int b, int c) : x(a), y(b), z(c) {
        cout << "You entered: ";
        cout << "(" << x << ", " << y << ", " << z << ")" << endl;
    }

    // Unary negation operator
    void operator-() {
        x = -x;
        y = -y;
        z = -z;
        cout << "Applying negation..."<<endl;
        cout << "The negated coordinates are (" << x << ", " << y << ", "
<< z << ")" << endl;
    }
};

int main() {

```

```

int x, y, z;

cout << "Enter x coordinate: ";
cin >> x;
cout << "Enter y coordinate: ";
cin >> y;
cout << "Enter z coordinate: ";
cin >> z;

Vector3D vector1(x, y, z);
-vector1; // unary negation operator

return 0;
}

```

Output:

```

Enter x coordinate: 10
Enter y coordinate: 5
Enter z coordinate: 2
You entered: (10, 5, 2)
Applying negation...
The negated coordinates are (-10, -5, -2)

```

4. Define a class NUM with a private integer data member. Implement a function void getNum(int x) to set the value of this data member and a function void dispNum() to display its value. Overload the unary logical NOT (!) operator for the NUM class. This operator should invert the boolean value of the integer data member (i.e., set it to 0 if it was non-zero, and set it to 1 if it was 0).

Code:

```

#include <iostream>
using namespace std;

class NUM {
private:
    int num;

public:
    NUM() : num(0) {}

    void getNum(int x) {
        num = x;
    }

    void dispNum() const {
        cout << "Value entered: " << num << endl;
    }

    void operator!() {

```

```

        if (num != 0) {
            num = 0;
        } else {
            num = 1;
        }
    }

    ~NUM() {
        if (num == 0) {
            cout << "Non-zero number entered. \nInverted boolean value: "
<< num<< endl;
        } else {
            cout << "Inverted boolean value of the integer value you have
entered is " << num <<endl;
        }
    }
};

int main() {
    int x;

    cout << "Enter any value: ";
    cin >> x;

    NUM num1;
    num1.getNum(x);
    num1.dispNum();
    !num1;

    return 0;
}

```

Output:

```

Enter any value: 10
Value entered: 10
Non-zero number entered.
Inverted boolean value: 0

```

5. Create a class Complex that represents a complex number with real and imaginary parts. Overload the binary addition (+) operator to add two Complex numbers. Implement a display function to show the complex number in the form $a + bi$. Write a main function to test the addition of two complex numbers.

Code:

```

#include <iostream>
using namespace std;

class Complex {
private:
    double real;

```

```

    double imaginary;

public:
    Complex(double r, double i) : real(r), imaginary(i) {}

    // Overloading addition operator (+)
    Complex operator+(const Complex& other) const {
        return Complex(real + other.real, imaginary + other.imaginary);
    }

    void display() const {
        cout << real << " + " << imaginary << "i" << endl;
    }
};

int main() {
    double real1, imag1, real2, imag2;

    cout << "Enter real part of complex number 1: ";
    cin >> real1;
    cout << "Enter imaginary part of complex number 1: ";
    cin >> imag1;

    cout << "Enter real part of the complex number 2: ";
    cin >> real2;
    cout << "Enter imaginary part of the complex number 2: ";
    cin >> imag2;

    Complex c1(real1, imag1);
    Complex c2(real2, imag2);

    Complex c3 = c1 + c2;

    cout << "Complex number 1: ";
    c1.display();
    cout << "Complex number 2: ";
    c2.display();
    cout << "Sum of the complex numbers: ";
    c3.display();

    return 0;
}

```

Output:

```

Enter real part of complex number 1: 10
Enter imaginary part of complex number 1: 4
Enter real part of the complex number 2: 20
Enter imaginary part of the complex number 2: 8
Complex number 1: 10 + 4i
Complex number 2: 20 + 8i
Sum of the complex numbers: 30 + 12i

```

6. Create a class Matrix that represents a 2x2 matrix. Overload the binary multiplication (*) operator to multiply two Matrix objects. Implement a display function to show the matrix. Write a main function to test the multiplication of two matrices.

Code:

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

class Matrix {
private:
    int mat[2][2];

public:
    Matrix(int a, int b, int c, int d) {
        mat[0][0] = a;
        mat[0][1] = b;
        mat[1][0] = c;
        mat[1][1] = d;
    }

    // Overload binary multiplication operator
    Matrix operator*(const Matrix& other) const {
        int result[2][2] = {0};

        for (int i = 0; i < 2; ++i) {
            for (int j = 0; j < 2; ++j) {
                for (int k = 0; k < 2; ++k) {
                    result[i][j] += mat[i][k] * other.mat[k][j];
                }
            }
        }

        return Matrix(result[0][0], result[0][1], result[1][0],
result[1][1]);
    }

    void display() const {
        for (int i = 0; i < 2; ++i) {
            for (int j = 0; j < 2; ++j) {
                cout << mat[i][j] << "\t";
            }
            cout << endl;
        }
    }
};

int main() {
    int a1, b1, c1, d1;
    int a2, b2, c2, d2;
```

```

    cout << "Enter elements of the first matrix (a b c d): ";
    cin >> a1 >> b1 >> c1 >> d1;

    cout << "Enter elements of the second matrix (a b c d): ";
    cin >> a2 >> b2 >> c2 >> d2;

    Matrix m1(a1, b1, c1, d1);
    Matrix m2(a2, b2, c2, d2);

    Matrix m3 = m1 * m2;

    cout << "Product: " << endl;
    m3.display();

    return 0;
}

```

Output:

```

Enter elements of the first matrix (a b c d): 1
2
3
4
Enter elements of the second matrix (a b c d): 5
6
7
8
Product:
19      22
43      50

```

7. Create a class Vector3D that represents a 3D vector with components x, y, and z. Overload the binary equality (==) operator to compare two Vector3D objects. Implement a display function to show the vector components. Write a main function to test the comparison of two vectors.

Code:

```

#include <iostream>
using namespace std;

class Vector3D {
private:
    double x, y, z;

public:
    Vector3D(double x, double y, double z) : x(x), y(y), z(z) {}

    //Overloading binary equality operator
    bool operator==(const Vector3D& other) const {
        return x == other.x && y == other.y && z == other.z;
    }

    void display() const {
        cout << "(" << x << ", " << y << ", " << z << ")" << endl;
    }
}

```



```

    }
};

int main() {
    double x1, y1, z1;
    double x2, y2, z2;

    cout << "Enter Vector 1 components (x,y,z): ";
    cin >> x1 >> y1 >> z1;

    cout << "Enter Vector 2 components (x,y,z): ";
    cin >> x2 >> y2 >> z2;

    Vector3D v1(x1, y1, z1);
    Vector3D v2(x2, y2, z2);

    cout << "First vector: ";
    v1.display();
    cout << "Second vector: ";
    v2.display();

    if (v1 == v2) {
        cout << "The vectors are equal." << endl;
    } else {
        cout << "The vectors are not equal." << endl;
    }

    return 0;
}

```

Output:

```

Enter Vector 1 components (x,y,z): 1
2
3
Enter Vector 2 components: (x,y,z)2
3
4
First vector: (1, 2, 3)
Second vector: (2, 3, 4)
The vectors are not equal.

```

8. Create a class BitwiseData with an integer data member. Overload the binary bitwise AND (&) operator to perform a bitwise AND operation between two BitwiseData objects. Implement a display function to show the result in binary format. Write a main function to test the bitwise AND operation.

Code:

```

#include <iostream>
using namespace std;

class BitwiseData {

```

```

private:
    int value;

public:
    BitwiseData(int value = 0) : value(value) {}

    BitwiseData operator&(const BitwiseData& other) const {
        return BitwiseData(this->value & other.value);
    }

    void display() const {
        int num = value;
        bool hasOutput = false;

        for (int i = 31; i >= 0; --i) {
            int bit = (num >> i) & 1;
            if (bit || hasOutput) {
                cout << bit;
                hasOutput = true;
            }
        }

        if (!hasOutput) {
            cout << '0';
        }

        cout << endl;
    }
};

int main() {
    int bd1, bd2;

    cout << "Enter the first integer value: ";
    cin >> bd1;

    cout << "Enter the second integer value: ";
    cin >> bd2;

    BitwiseData num1(bd1);
    BitwiseData num2(bd2);

    cout << "BitwiseData 1: ";
    num1.display();
    cout << "BitwiseData 2: ";
    num2.display();

    BitwiseData result = num1 & num2;
    cout << "Result of BitwiseData 1 & BitwiseData 2: ";
    result.display();
}

```

```
    return 0;
}
```

Output:

```
Enter the first integer value: 5
Enter the second integer value: 4
BitwiseData 1: 101
BitwiseData 2: 100
Result of BitwiseData 1 & BitwiseData 2: 100
```

9. Define a class Person that has private members for name (a std::string) and age (an int). Overload the binary stream insertion (<<) operator to print the Person object in the format Name: [name], Age: [age]. Write a main function to create a Person object, set its values, and print it using the overloaded << operator.

Code:

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

class Person {
private:
    string name;
    int age;

public:
    Person(const string& name = "", int age = 0) : name(name), age(age)
    {}

    void display() const {
        cout << "Name: " << name << ", Age: " << age << endl;
    }

    void setName(const string& name) {
        this->name = name;
    }

    //setter
    void setAge(int age) {
        this->age = age;
    }

    //getter
    string getName() const {
        return name;
    }

    int getAge() const {
        return age;
    }
}
```

```

};

int main() {
    string name;
    int age;

    cout << "Enter the name of the first person: ";
    getline(cin, name);
    cout << "Enter the age of the first person: ";
    cin >> age;
    cin.ignore();
    Person person1(name, age);

    cout << "Enter the name of the second person: ";
    getline(cin, name);
    cout << "Enter the age of the second person: ";
    cin >> age;
    Person person2(name, age);

    cout << "Person 1: ";
    person1.display();
    cout << "Person 2: ";
    person2.display();

    return 0;
}

```

Output:

```

Enter the name of the first person: Pevin
Enter the age of the first person: 19
Enter the name of the second person: Colpitts
Enter the age of the second person: 84
Person 1: Name: Pevin, Age: 19
Person 2: Name: Colpitts, Age: 84

```

8. Define a class Rectangle with private members width and height (both integers). Overload the binary stream extraction (>>) operator to read the width and height from an input stream in the format width height. Write a main function to create a Rectangle object, read its dimensions using the overloaded >> operator, and display its dimensions.

Code:

```

#include <iostream>
using namespace std;

```

```

class Rectangle {
private:
    int width;
    int height;

```

```

public:

```

```

    Rectangle(int width = 0, int height = 0) : width(width),
height(height) {}

    friend istream& operator>>(istream& is, Rectangle& rect);

    void display() const {
        cout << "Width: " << width << ", Height: " << height << endl;
    }
};
istream& operator>>(istream& is, Rectangle& rect) {
    cout << "Enter the width and height of the rectangle : ";
    is >> rect.width >> rect.height;
    return is;
}

int main() {
    Rectangle rect;
    cin >> rect;

    rect.display();
    return 0;
}

```

Output:

```

Enter the width and height of the rectangle : 20
5
Width: 20, Height: 5

```

Result: Thus, operator overloading concepts were successfully implemented in C++.

OOPS AND DATA STRUCTURES LABORATORY

Ex 5: Exception Handling

Name: Pevin S

Roll No: 3122233002078

Section: ECE-B

Aim: To implement C++ programs that demonstrate exception handling, including standard and user-defined exceptions.

1. Implement a C++ program that performs division of two numbers and stores the result in an array. The program should handle two specific exceptions:

a. Array Index Out of Bounds:

An exception should be thrown if an attempt is made to access or modify an array element using an index that is out of the array's bounds.

b. Division by Zero:

An exception should be thrown if there is an attempt to divide a number by zero.

Code:

```
#include <iostream>
#include <stdexcept>

using namespace std;
int main() {
    const int arraySize = 10;
    int result[arraySize] = {0};

    int num, den, i;

    cout << "Enter numerator: ";
    cin >> num;

    cout << "Enter denominator: ";
    cin >> den;

    try {
        if (den == 0) {
            throw runtime_error ("Error! Division by zero");
        }

        cout << "Enter index (0 to " << arraySize - 1 << "): ";
        cin >> i;

        if (i < 0 || i >= arraySize) {
            throw runtime_error ("Error: Index out of Array's bound");
        }

        result[i] = num / den;
```

```

        cout << "Result stored in index " << i << ": " << result[i] <<
endl;
    }

    catch (const exception& e) {
        cout << "Exception " << e.what() << endl;
    }
    return 0;
}

```

Output:

```

Enter numerator: 18
Enter denominator: 3
Enter index (0 to 9): 12
Exception Error: Index out of Array's bound

```

```

Enter numerator: 5
Enter denominator: 0
Exception Error! Division by zero

```

```

Enter numerator: 18
Enter denominator: 2
Enter index (0 to 9): 5
Result stored in index 5: 9

```

2. Write a C++ program that attempts to open and read the contents of a file. The program should handle the scenario where the file does not exist or cannot be opened. Use standard C++ exception handling to manage such errors

Code:

```

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main() {
    string filename;
    cout << "Enter the filename: ";
    cin >> filename;

    ifstream file(filename);

    if (!file.is_open()) {
        cerr << "Error: Unable to open file." << endl;
        return 1;
    }
    string line;
    while (getline(file, line)) {
        cout << line << endl;
    }
    file.close();
    return 0;
}

```

```
}
```

Output:

```
Enter the filename: pevin.txt
This is Pevin from ECE-B.
3122233002078
```

3. Implement a C++ program that validates user input based on specific criteria. The program should include user-defined exceptions to handle validation errors for age, income, and city. Below is a description of the requirements and the exceptions you need to handle:

a. Age Validation:

The age must be between 18 and 55 inclusive. If the age is outside this range, throw an exception.

b. Income Validation:

The income must not exceed 50 lakh (5,000,000). If the income is greater than this amount, throw an exception.

c. City Validation:

The city must be either Chennai or Pune. If the city entered is neither of these, throw an exception.

Code:

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

class AgeException {
public:
    string message;
    AgeException(const string& msg) : message(msg) {}
    string getMessage() const {
        return message;
    }
};

class IncomeException {
public:
    string message;
    IncomeException(const string& msg) : message(msg) {}
    string getMessage() const {
        return message;
    }
};

class CityException {
public:
    string message;
    CityException(const string& msg) : message(msg) {}
    string getMessage() const {
```



```

        return message;
    }
};

void validateAge(int age) {
    if (age < 18 || age > 55) {
        throw AgeException("Age must be between 18 and 55 inclusive");
    }
}

void validateIncome(double income) {
    if (income > 5000000) {
        throw IncomeException("Income must not exceed 50 lakh.");
    }
}

void validateCity(const string& city) {
    if (city != "Chennai" && city != "Pune") {
        throw CityException("City must be either Chennai or Pune.");
    }
}

int main() {
    int age;
    double income;
    string city;

    try {
        cout << "Enter age: ";
        cin >> age;
        validateAge(age);

        cout << "Enter income: ";
        cin >> income;
        validateIncome(income);

        cout << "Enter city: ";
        cin >> city;
        validateCity(city);

        cout << "All inputs are valid." << endl;
    }
    catch (AgeException& e) {
        cerr << e.getMessage() << endl;
    }
    catch (IncomeException& e) {
        cerr << e.getMessage() << endl;
    }
    catch (CityException& e) {
        cerr << e.getMessage() << endl;
    }
}

```

```
    return 0;
}
```

Output:

```
Enter age: 16
Age must be between 18 and 55 inclusive
```

```
Enter age: 19
Enter income: 25000000
Income must not exceed 50 lakh.
```

```
Enter age: 19
Enter income: 25000
Enter city: Delhi
City must be either Chennai or Pune.
```

```
Enter age: 19
Enter income: 30000
Enter city: Chennai
All inputs are valid.
```

4. Write a C++ program that reads an input string from the user and checks its length. If the length of the input string exceeds 5 characters, the program should throw and catch an exception.

Define a user-defined exception class `StringTooLongException` to handle this specific error.

Ensure that the exception is caught, and an appropriate error message is printed

Code:

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

class StringTooLongException {
public:
    string message;
    StringTooLongException(const string& msg) : message(msg) {}
    string getMessage() const {
        return message;
    }
};

void checkStringLength(const string& str) {
    if (str.length() > 5) {
        throw StringTooLongException("Input string exceeds 5
characters.");
    }
}

int main() {
    string input;
```

```
cout << "Enter a string: ";  
cin >> input;  
  
try {  
    checkStringLength(input);  
    cout << "Input string is valid." << endl;  
}  
catch (StringTooLongException& e) {  
    cerr << "Error: " << e.getMessage() << endl;  
}  
  
return 0;  
}
```

Output:

```
Enter a string: Program  
Error: Input string exceeds 5 characters.
```

Result: Thus, errors are handled using exception handling techniques.

OOPS AND DATA STRUCTURES LABORATORY

Ex 6: Lists

Name: Pevin S

Roll No: 3122233002078

Section: ECE-B

Aim: To implement data structures in C++, including an array-based list and a singly linked list.

1. Implement a list using an array in C++. Provide functions for:

1. Inserting an element at a specific position.
2. Deleting an element from a specific position.
3. Displaying all elements in the list.
4. Searching for an element in the list.

Code:

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

class ArrayList {
private:
    int* arr;
    int capacity;
    int size;
public:
    ArrayList(int cap) : capacity(cap), size(0) {
        arr = new int[capacity];
    }

    ~ArrayList() {
        delete[] arr;
    }

    void initialize(const int* values, int numValues) {
        if (numValues <= capacity) {
            for (int i = 0; i < numValues; i++) {
                arr[i] = values[i];
            }
            size = numValues;
        }
    }

    void insert(int position, int value) {
        if (position >= 0 && position <= size && size < capacity) {
            for (int i = size; i > position; i--) {
                arr[i] = arr[i-1];
            }
            arr[position] = value;
            size++;
        }
    }
};
```

```

        } else {
            cout<<"Insertion Error: Invalid position"<<endl;
        }
    }

void remove(int position) {
    if (position>=0 && position<size) {
        for (int i=position; i<size - 1; i++) {
            arr[i] = arr[i+1];
        }
        size--;
    } else {
        cout<<"Deletion Error: Invalid position"<<endl;
    }
}

void display() const {
    if (size==0) {
        cout<<"List is empty!"<<endl;
        return;
    }

    cout<<"Elements in the list: ";
    for (int i=0; i<size; i++) {
        cout<<arr[i]<<" ";
    }
    cout<<endl;
}

bool search(int value) const {
    for (int i=0; i<size; i++) {
        if (arr[i]==value) {
            return true;
        }
    }
    return false;
}

int getSize() const {
    return size;
}
};

int main() {
    ArrayList list(10);

    int initialValues[] = {10, 20, 30, 40, 50};
    list.initialize(initialValues, 5);

    cout << "List: ";
    list.display();
}

```

```

int choice, position, value;

cout << "Menu" << endl;
cout << "1. Insert an element" << endl;
cout << "2. Delete an element" << endl;
cout << "3. Display all elements" << endl;
cout << "4. Search for an element" << endl;
cout << "5. Exit" << endl;

cout << "Enter your choice: ";
cin >> choice;

while (choice != 5) {
    switch (choice) {
        case 1:
            cout << "Enter position to insert (0 to " << list.getSize()
<< "): ";
            cin >> position;
            cout << "Enter value: ";
            cin >> value;
            list.insert(position, value);
            break;

            case 2:
                cout << "Enter position to delete (0 to " << list.getSize()
- 1 << "): ";
                cin >> position;
                list.remove(position);
                break;

            case 3:
                list.display();
                break;

            case 4:
                cout << "Enter element to search: ";
                cin >> value;
                cout << "Element " << value << ": " << (list.search(value) ?
"Found" : "Not Found") << endl;
                break;

            default:
                cout << "Invalid input" << endl;
    }

    cout << "Enter your choice: ";
    cin >> choice;
}

cout << "Exiting." << endl;

```

```

    return 0;
}

```

Output:

```

List: Elements in the list: 10 20 30 40 50
Menu
1. Insert an element
2. Delete an element
3. Display all elements
4. Search for an element
5. Exit
Enter your choice: 1
Enter position to insert (0 to 5): 4
Enter value: 78
Enter your choice: 2
Enter position to delete (0 to 5): 2
Enter your choice: 3
Elements in the list: 10 20 40 78 50
Enter your choice: 4
Enter element to search: 50
Element 50: Found
Enter your choice: 5

```

2. Write a function to reverse the elements of an array-based list.

Code:

```

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

void reverseList(const vector<int>& arr) {
    cout<<"Original List: ";
    for (int num : arr) cout<<num<< " ";
    cout << endl;

    cout<<"Reversed List: ";
    for (auto i = arr.rbegin(); i != arr.rend(); i++) cout << *i << " ";
    cout << endl;
}

int main() {
    vector<int> myList = {10,20,30,40,50,60,70,80,90,100};
    reverseList(myList);
    return 0;
}

```

Output:

```

Original List: 10 20 30 40 50 60 70 80 90 100
Reversed List: 100 90 80 70 60 50 40 30 20 10

```

3. Implement a singly linked list with the following operations:

1. Inserting a node at the beginning.
2. Inserting a node at the end.
3. Deleting a node by value.
4. Displaying all nodes in the list.
5. Searching for a node by value

Code:

```

#include <iostream>
using namespace std;

```

```

class Node {
public:
    int value;
    Node* next;
    Node(int val) : value(val), next(nullptr) {}
};

```

```

class SinglyLinkedList {
private:
    Node* head;
public:
    SinglyLinkedList() {
        head=new Node(10);
        head->next = new Node(20);
        head->next->next = new Node(30);
    }

    void insertBegin(int value) {
        Node* newNode = new Node(value);
        newNode->next = head;
        head=newNode;
    }

    void insertEnd(int value) {
        Node* newNode = new Node(value);
        if (!head) {
            head=newNode;
            return;
        }
        Node* current = head;
        while (current->next) {
            current = current->next;
        }
        current->next = newNode;
    }

    void dlt(int value) {
        if (!head) return;

        if (head->value == value) {
            Node* temp = head;
            head = head->next;
            delete temp;
            return;
        }

        Node* current = head;
        while (current->next) {
            if (current->next->value == value) {
                Node* temp = current->next;

```



```

        current->next = current->next->next;
        delete temp;
        return;
    }
    current=current->next;
}
}

bool search(int value) {
    Node* current = head;
    while (current) {
        if (current->value == value) {
            return true;
        }
        current = current->next;
    }
    return false;
}

void display() {
    Node* current = head;
    while (current) {
        cout << current->value << " -> ";
        current = current->next;
    }
    cout << "null" << endl;
}

~SinglyLinkedList() {
    while (head) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
}

};

```

```

int main() {
    SinglyLinkedList list;
    cout << "List: ";
    list.display();
    int sel;

    cout<<"Menu:"<<endl;
    cout<<"1. Insert at Beginning"<<endl;
    cout<<"2. Insert at End"<<endl;
    cout<<"3. Delete by Value\n";
    cout<<"4. Display List"<<endl;
    cout<<"5. Search for Value"<<endl;
    cout<<"6. Exit"<<endl;
}

```

```

while (true) {
    cout<<"Select your operation ";
    cin>>sel;

    switch (sel) {
        case 1: {
            int val;
            cout<<"Enter value to insert at beginning: ";
            cin>>val;
            list.insertBegin(val);
            break;
        }
        case 2: {
            int val;
            cout<<"Enter value to insert at end: ";
            cin>>val;
            list.insertEnd(val);
            break;
        }
        case 3: {
            int val;
            cout<<"Enter value to delete: ";
            cin>>val;
            list.dlt(val);
            break;
        }
        case 4:
            cout << "Updated List: ";
            list.display();
            break;
        case 5: {
            int val;
            cout<<"Enter value to search: ";
            cin>>val;
            if (list.search(val)) {
                cout<<val<<" FOUND!"<<endl;
            } else {
                cout<<val<<" NOT FOUND"<<endl;
            }
            break;
        }
        case 6:
            cout << "Exiting" << endl;
            return 0;
        default:
            cout <<"Invalid input!"<< endl;
    }
}
return 0;
}

```

Output:

```

List: 10 -> 20 -> 30 -> null
Menu:
1. Insert at Beginning
2. Insert at End
3. Delete by Value
4. Display List
5. Search for Value
6. Exit
Select your operation 1
Enter value to insert at beginning: 0
Select your operation 2
Enter value to insert at end: 78
Select your operation 3
Enter value to delete: 20
Select your operation 4
Updated List: 0 -> 10 -> 30 -> 78 -> null
Select your operation 5
Enter value to search: 40
40 NOT FOUND

```

4. Write a function to reverse a singly linked list. Ensure that you correctly update the links between nodes and handle the case of an empty list.

Code:

```

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

#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
    Node(int value) : data(value), next(nullptr) {}
};

Node* reverseLinkedList(Node* head) {
    if (head == nullptr) return nullptr;

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

    while (current != nullptr) {
        Node* nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode;
    }
    return prev;
}

void printList(Node* head) {
    Node* current = head;
    while (current) {
        cout << current->data << " -> ";
        current = current->next;
    }
    cout << "nullptr" << endl;
}

void deleteList(Node* head) {
    Node* current = head;
    while (current != nullptr) {

```

```

        Node* nextNode = current->next;
        delete current;
        current = nextNode;
    }
}

Node* createListFromInput() {
    int value;
    Node* head = nullptr;
    Node* tail = nullptr;

    cout << "Enter elements: (Enter -1 to stop): " << endl;
    while (true) {
        cin >> value;
        if (value == -1) break;

        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }
    return head;
}

int main() {
    Node* head = createListFromInput();
    cout << "Original List:" << endl;
    printList(head);

    Node* reversedHead = reverseLinkedList(head);
    cout << "Reversed List:" << endl;
    printList(reversedHead);
    deleteList(reversedHead);
    return 0;
}

```

Output:

```

Enter elements: (Enter -1 to stop):
20
30
40
78
100
-1
Original List:
20 -> 30 -> 40 -> 78 -> 100 -> nullptr
Reversed List:
100 -> 78 -> 40 -> 30 -> 20 -> nullptr

```

Result: Thus, various operations on array and linked list structures were successfully implemented in C++.

OOPS AND DATA STRUCTURES LABORATORY

Ex 7: Inheritance

Name: Pevin S

Roll No: 3122233002078

Section: ECE-B

Aim: To demonstrate inheritance and polymorphism in C++.

1. Create a base class Employee with attributes like name and id. Derive classes FullTimeEmployee and PartTimeEmployee that include a method to calculate salary and display employee details and salary details.

Code:

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

class Employee {
protected:
    string name;
    int id;

public:
    Employee(string empName, int empId) : name(empName), id(empId) {}

    virtual void displayDetails() {
        cout << "Employee Name: " << name << endl;
        cout << "Employee ID: " << id << endl;
    }

    virtual double calculateSalary() = 0;
    int getId() const { return id; }
};

class FullTimeEmployee : public Employee {
private:
    double monthlySalary;

public:
    FullTimeEmployee(string empName, int empId, double salary)
        : Employee(empName, empId), monthlySalary(salary) {}

    void displayDetails() override {
        Employee::displayDetails();
        cout << "Employee Type: Full-Time" << endl;
        cout << "Monthly Salary: " << monthlySalary << endl;
    }
}
```

```

        double calculateSalary() override {
            return monthlySalary;
        }
};

class PartTimeEmployee : public Employee {
private:
    double hourlyRate;
    int hoursWorked;

public:
    PartTimeEmployee(string empName, int empId, double rate, int hours)
        : Employee(empName, empId), hourlyRate(rate), hoursWorked(hours)
    {}

    void displayDetails() override {
        Employee::displayDetails();
        cout << "Employee Type: Part-Time" << endl;
        cout << "Salary per Hour: " << hourlyRate << endl;
        cout << "Hours Worked: " << hoursWorked << endl;
    }

    double calculateSalary() override {
        return hourlyRate * hoursWorked;
    }
};

//find employee by ID
Employee* findEmployeeById(const vector<Employee*>& employees, int id) {
    for (Employee* emp : employees) {
        if (emp->getId() == id) {
            return emp;
        }
    }
    return nullptr;
}

int main() {
    vector<Employee*> employees;
    //employee database
    employees.push_back(new FullTimeEmployee("Chris", 1, 3000));
    employees.push_back(new PartTimeEmployee("Pevin", 2, 20, 80));
    employees.push_back(new FullTimeEmployee("Rose", 3, 4000));
    employees.push_back(new PartTimeEmployee("Sooraj", 4, 15, 60));

    int searchId;
    cout << "Enter Employee ID: ";
    cin >> searchId;

    Employee* emp = findEmployeeById(employees, searchId);

```

```

    if (emp) {
        emp->displayDetails();
        cout << "Total Salary: " << emp->calculateSalary() << endl;
    } else {
        cout << "ID Not found!" << endl;
    }

    return 0;
}

```

Output:

```

Enter Employee ID: 2
Employee Name: Pevin
Employee ID: 2
Employee Type: Part-Time
Salary per Hour: 20
Hours Worked: 80
Total Salary: 1600

```

```

Enter Employee ID: 3
Employee Name: Rose
Employee ID: 3
Employee Type: Full-Time
Monthly Salary: 4000
Total Salary: 4000

```

Q2. Create a base class named Shape with functions to calculate the area and perimeter, as well as to display the shape's details. Derive classes Circle and Rectangle from Shape, implementing constructors and methods to calculate and display area and perimeter.

Code:

```

#include <iostream>
using namespace std;

class Shape {
public:
    virtual double calcArea() = 0;
    virtual double calcPerimeter() = 0;
    virtual void displayDetails() = 0;
};

class Circle : public Shape {
private:
    double r;

public:
    Circle(double radius) : r(radius) {}

    double calcArea() override {
        return 3.14 * r * r;
    }

    double calcPerimeter() override {
        return 2 * 3.14 * r;
    }

    void displayDetails() override {
        cout << "Shape: Circle" << endl;
    }
}

```

```

        cout << "Radius: " << r << endl;
        cout << "Area: " << calcArea() << endl;
        cout << "Perimeter: " << calcPerimeter() << endl;
    }
};

class Rectangle : public Shape {
private:
    double l;
    double b;
public:
    Rectangle(double length, double breadth) : l(length), b(breadth) {}

    double calcArea() override {
        return l * b;
    }

    double calcPerimeter() override {
        return 2 * (l + b);
    }

    void displayDetails() override {
        cout << "Shape: Rectangle" << endl;
        cout << "Length: " << l << endl;
        cout << "Breadth: " << b << endl;
        cout << "Area: " << calcArea() << endl;
        cout << "Perimeter: " << calcPerimeter() << endl;
    }
};

int main() {
    Shape* shapes[2];
    shapes[0] = new Circle(5);
    shapes[1] = new Rectangle(4, 6);

    for (int i = 0; i < 2; i++) {
        shapes[i]->displayDetails();
        cout << endl;
    }
    return 0;
}

```

Output:

```

Shape: Circle
Radius: 5
Area: 78.5
Perimeter: 31.4

```

```

Shape: Rectangle
Length: 4
Breadth: 6
Area: 24
Perimeter: 20

```


Q3. Design a base class Account with attributes for accountNumber and balance. Derive a class SavingsAccount that includes methods for depositing, withdrawing, and calculating interest.

- Implement methods in Account to display account information.
- In SavingsAccount, add functionality to calculate interest based on balance.

Code:

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

class Account {
protected:
    string accNum;
    double balance;

public:
    Account(string number, double initBal)
        : accNum(number), balance(initBal) {}

    void displayAccInfo() {
        cout << "Account Number: " << accNum << endl;
        cout << "Balance: Rs. " << balance << endl;
    }
};

class SavingsAcc : public Account {
private:
    double int_rate;

public:
    SavingsAcc(string number, double initBal, double rate)
        : Account(number, initBal), int_rate(rate) {}

    void deposit(double depositAmt) {
        balance += depositAmt;
        cout << "Deposited: Rs. " << depositAmt << endl;
    }

    void withdraw(double withdrawAmt) {
        if (withdrawAmt > 0 && withdrawAmt <= balance) {
            balance -= withdrawAmt;
            cout << "Withdrawn: Rs. " << withdrawAmt << endl;
        } else {
            cout << "Balance insufficient!" << endl;
        }
    }

    double calcInt() {
        return (balance * int_rate) / 100;
    }
}
```

```

void displayAccInfo() {
    Account::displayAccInfo();
    cout << "Interest Rate: " << int_rate << "%" << endl;
    cout << "Interest: Rs. " << calcInt() << endl;
}
};

int main() {
    string accNum;
    double initBal;
    cout << "Welcome to Banking System. \nEnter your account details to continue.\n";
    cout << "Enter Account Number: ";
    cin >> accNum;
    cout << "Enter Initial Balance: Rs. ";
    cin >> initBal;

    SavingsAcc myAccount(accNum, initBal, 3.6);
    cout << "1. Deposit\n";
    cout << "2. Withdraw\n";
    cout << "3. Calculate Interest\n";
    cout << "4. Display Account Info\n";
    cout << "5. Exit\n";

    int x;
    do {
        cout << "Choose an option: "<<endl;
        cin >> x;
        switch (x) {
            case 1: {
                double depositAmt;
                cout << "Enter amount to deposit: Rs. ";
                cin >> depositAmt;
                myAccount.deposit(depositAmt);
                break;
            }
            case 2: {
                double withdrawAmt;
                cout << "Enter amount to withdraw: Rs. ";
                cin >> withdrawAmt;
                myAccount.withdraw(withdrawAmt);
                break;
            }
            case 3: {
                cout << "Interest: Rs. " << myAccount.calcInt() << endl;
                break;
            }
            case 4: {
                myAccount.displayAccInfo();
                break;
            }
        }
    } while (x != 5);
}

```

```

    }
    case 5:
        cout << "Thank you for using our services" << endl;
        break;
    default:
        cout << "Invalid input!" << endl;
    }
} while (x != 5);
return 0;
}

```

Output:

<pre> Welcome to Banking System. Enter your account details to continue. Enter Account Number: 3122233002078 Enter Initial Balance: Rs. 1000 1. Deposit 2. Withdraw 3. Calculate Interest 4. Display Account Info 5. Exit Choose an option: 1 Enter amount to deposit: Rs. 100 Deposited: Rs. 100 </pre>	<pre> Choose an option: 2 Enter amount to withdraw: Rs. 500 Withdrawn: Rs. 500 Choose an option: 3 Interest: Rs. 21.6 Choose an option: 4 Account Number: 3122233002078 Balance: Rs. 600 Interest Rate: 3.6% Interest: Rs. 21.6 Choose an option: 5 Thank you for using our services </pre>
--	---

4. Create a class hierarchy for an online course platform. Start with a base class Course with attributes for courseName and duration. Derive a class OnlineCourse, and then create a class SpecializedCourse that derives from OnlineCourse and adds an attribute for specialization.

- Implement methods to display course details in each class.
- In SpecializedCourse, override the method to include specialization information.

Code:

```

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

class Course {
protected:
    string courseName;
    int duration;

public:
    Course(string name, int dur) : courseName(name), duration(dur) {}

    virtual void displayDetails() {
        cout << "Course Name: " << courseName << endl;
        cout << "Duration: " << duration << " hours" << endl;
    }
};

class OnlineCourse : public Course {
public:
    OnlineCourse(string name, int dur) : Course(name, dur) {}

    void displayDetails() override {

```

```

        cout << "Online Course Details:" << endl;
        Course::displayDetails();
    }
};

class SpecializedCourse : public OnlineCourse {
private:
    string specialization;

public:
    SpecializedCourse(string name, int dur, string spec)
        : OnlineCourse(name, dur), specialization(spec) {}

    void displayDetails() override {
        cout << "Specialized Course Details:" << endl;
        OnlineCourse::displayDetails();
        cout << "Specialization: " << specialization << endl;
    }
};

int main() {
    Course cppCourse("Programming in C++ for Beginners", 40);
    OnlineCourse onlineCppCourse("Advanced C++", 60);
    SpecializedCourse specializedCppCourse("Introduction to OOPS in C++",
30, "Advanced OOPS Concepts");
    cout << "Basic Course:" << endl;
    cppCourse.displayDetails();
    cout << endl;
    onlineCppCourse.displayDetails();
    cout << endl;
    specializedCppCourse.displayDetails();
    return 0;
}

```

Output:

```

Basic Course:
Course Name: Programming in C++ for Beginners
Duration: 40 hours

Online Course Details:
Course Name: Advanced C++
Duration: 60 hours

Specialized Course Details:
Online Course Details:
Course Name: Introduction to OOPS in C++
Duration: 30 hours
Specialization: Advanced OOPS Concepts

```

5. Implement a base class Game with attributes for title and genre. Derive a class VideoGame that adds attributes like platform and releaseYear. Implement methods to display game details.

- Include a method in Game to display basic information.
- In VideoGame, override this method to include platform and release year.

Code:

```
#include <iostream>
#include <string>
using namespace std;
class Game {
protected:
    string title;
    string genre;
public:
    Game(string t, string g) : title(t), genre(g) {}
    virtual void displayInfo() {
        cout << "Title: " << title << endl;
        cout << "Genre: " << genre << endl;
    }
};

class VideoGame : public Game {
private:
    string platform;
    int releaseYear;
public:
    VideoGame(string t, string g, string p, int year)
        : Game(t, g), platform(p), releaseYear(year) {}

    void displayInfo() override {
        Game::displayInfo();
        cout << "Platform: " << platform << endl;
        cout << "Release Year: " << releaseYear << endl;
    }
};

int main() {
    Game pubg("PUBG", "Battle Royale");
    VideoGame cod("Call of Duty", "First-Person Shooter", "PC/Console",
2021);
    VideoGame gtaV("Grand Theft Auto V", "Action-Adventure",
"PC/Console", 2013);
    cout << "Basic Game Information:" << endl;
    pubg.displayInfo();
    cout << endl;
    cout << "Detailed Video Game Information:\n" << endl;
    cod.displayInfo();
    cout << endl;
    gtaV.displayInfo();

    return 0;
}
```

Output:

```

Basic Game Information:
Title: PUBG
Genre: Battle Royale

Detailed Video Game Information:

Title: Call of Duty
Genre: First-Person Shooter
Platform: PC/Console
Release Year: 2021

Title: Grand Theft Auto V
Genre: Action-Adventure
Platform: PC/Console
Release Year: 2013

```

6. Implement a class hierarchy for an e-commerce platform. Create a base class Product with attributes for name, price, and stock. Derive a class DigitalProduct, and then create a class Subscription that derives from DigitalProduct with an attribute for subscriptionLength.

- Implement methods to display product details in each class.
- In Subscription, override the method to include the subscription length.

Code:

```

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

class Product {
protected:
    string name;
    double price;
    int stock;
public:
    Product(string n, double p, int s) : name(n), price(p), stock(s) {}

    virtual void display() {
        cout << "Product Name: " << name << endl;
        cout << "Price: Rs. " << price << endl;
        cout << "Stock: " << stock << endl;
    }
};

class DigitalProduct : public Product {
public:
    DigitalProduct(string n, double p, int s) : Product(n, p, s) {}

    void display() override {
        cout << "Digital Product Details:" << endl;
        Product::display();
    }
};

class Subscription : public DigitalProduct {
private:
    int subscriptionLength;
public:

```

```

Subscription(string n, double p, int s, int length)
    : DigitalProduct(n, p, s), subscriptionLength(length) {}

void display() override {
    cout << "Subscription Product Details:" << endl;
    DigitalProduct::display();
    cout << "Subscription Length: " << subscriptionLength << "
months" << endl;
}
};

int main() {
    Product physicalProduct("Logitech Wireless Mouse", 1500, 100);
    DigitalProduct ebook("Nokia 3310", 2500, 200);
    Subscription softwareSubscription("Kaspersky Total Security Plus",
1299, 50, 12);
    cout << "Physical Product Information:" << endl;
    physicalProduct.display();
    cout << endl;
    ebook.display();
    cout << endl;
    softwareSubscription.display();
    cout << endl;
    return 0;
}

```

Output:

```

Physical Product Information:
Product Name: Logitech Wireless Mouse
Price: Rs. 1500
Stock: 100

Digital Product Details:
Product Name: Nokia 3310
Price: Rs. 2500
Stock: 200

Subscription Product Details:
Digital Product Details:
Product Name: Kaspersky Total Security Plus
Price: Rs. 1299
Stock: 50
Subscription Length: 12 months

```

Result: Thus, inheritance and polymorphism concepts were successfully implemented in C++.

OOPS AND DATA STRUCTURES LABORATORY

Ex 8: Data Structures

Name: Pevin S

Roll No: 3122233002078

Section: ECE-B

Aim: To implement advanced data structures in C++, including doubly linked lists, circular linked lists, stacks using arrays and linked lists, and to perform expression conversion and evaluation.

1. Implement a doubly linked list and circular linked list with the following operations:

1. Inserting a node at the beginning.
2. Inserting a node at the end.
3. Deleting a node by value.
4. Displaying all nodes in the list.
5. Searching for a node by value.

Doubly Linked List:

Code:

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

struct Node {
    int data;
    Node* prev;
    Node* next;
};

class DoublyLinkedList {
private:
    Node* head;

public:
    DoublyLinkedList() : head(nullptr) {}

    void insertAtBeginning(int value) {
        Node* newNode = new Node{value, nullptr, head};
        if (head != nullptr) {
            head->prev = newNode;
        }
        head = newNode;
    }

    void insertAtEnd(int value) {
```



```

Node* newNode = new Node{value, nullptr, nullptr};
if (head == nullptr) {
    head = newNode;
    return;
}
Node* temp = head;
while (temp->next != nullptr) {
    temp = temp->next;
}
temp->next = newNode;
newNode->prev = temp;
}

void deleteByValue(int value) {
    Node* temp = head;
    while (temp != nullptr && temp->data != value) {
        temp = temp->next;
    }
    if (temp == nullptr) return;
    if (temp->prev != nullptr) {
        temp->prev->next = temp->next;
    } else {
        head = temp->next;
    }
    if (temp->next != nullptr) {
        temp->next->prev = temp->prev;
    }
    delete temp;
}

void display() {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data;
        if (temp->next != nullptr) {
            cout << " <-> ";
        }
        temp = temp->next;
    }
    cout << endl;
}

bool search(int value) {
    Node* temp = head;
    while (temp != nullptr) {
        if (temp->data == value) return true;
        temp = temp->next;
    }
    return false;
}

};

```

```

int main() {
    DoublyLinkedList dll;
    int choice, value;

    cout << "\nMenu:\n";
    cout << "1. Insert at beginning\n";
    cout << "2. Insert at end\n";
    cout << "3. Delete by value\n";
    cout << "4. Display\n";
    cout << "5. Search\n";
    cout << "6. Exit\n";
    while (true) {
        cout << "\nEnter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "Enter value: ";
                cin >> value;
                dll.insertAtBeginning(value);
                break;
            case 2:
                cout << "Enter value: ";
                cin >> value;
                dll.insertAtEnd(value);
                break;
            case 3:
                cout << "Enter value to delete: ";
                cin >> value;
                dll.deleteByValue(value);
                break;
            case 4:
                cout << "List: ";
                dll.display();
                break;
            case 5:
                cout << "Enter value to search: ";
                cin >> value;
                if (dll.search(value)) {
                    cout << "Value found in the list.\n";
                } else {
                    cout << "Value not found in the list.\n";
                }
                break;
            case 6:
                cout << "Exiting...\n";
                return 0;
            default:
                cout << "Invalid choice! Please try again.\n";
        }
    }
}

```

```

    return 0;
}

```

Output:

```

Menu:
1. Insert at beginning
2. Insert at end
3. Delete by value
4. Display
5. Search
6. Exit

Enter your choice: 1
Enter value: 10

Enter your choice: 2
Enter value: 20

Enter your choice: 2
Enter value: 30

Enter your choice: 1
Enter value: 30

Enter your choice: 4
List: 30 <-> 10 <-> 20 <-> 30

Enter your choice: 3
Enter value to delete: 10

Enter your choice: 5
Enter value to search: 20
Value found in the list.

```

Circular Linked List:

Code:

```

#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
};

class CircularLinkedList {
private:
    Node* head;

public:
    CircularLinkedList() : head(nullptr) {}

    void insertAtBeginning(int value) {
        Node* newNode = new Node{value, nullptr};
        if (head == nullptr) {
            head = newNode;
            newNode->next = head;
        } else {
            Node* temp = head;
            while (temp->next != head) {
                temp = temp->next;
            }
            newNode->next = head;
            temp->next = newNode;
            head = newNode;
        }
    }
}

```

```

void insertAtEnd(int value) {
    Node* newNode = new Node{value, nullptr};
    if (head == nullptr) {
        head = newNode;
        newNode->next = head;
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
    }
}

void deleteByValue(int value) {
    if (head == nullptr) return;
    if (head->data == value && head->next == head) {
        delete head;
        head = nullptr;
        return;
    }
    Node* curr = head;
    Node* prev = nullptr;
    do {
        if (curr->data == value) {
            if (prev != nullptr) {
                prev->next = curr->next;
            } else {
                Node* temp = head;
                while (temp->next != head) {
                    temp = temp->next;
                }
                head = curr->next;
                temp->next = head;
            }
            delete curr;
            return;
        }
        prev = curr;
        curr = curr->next;
    } while (curr != head);
}

void display() {
    if (head == nullptr) return;
    Node* temp = head;
    do {
        cout << temp->data;
        temp = temp->next;
    }

```

```

        if (temp != head) {
            cout << " -> ";
        }
    } while (temp != head);
    cout << endl;
}

bool search(int value) {
    if (head == nullptr) return false;
    Node* temp = head;
    do {
        if (temp->data == value) return true;
        temp = temp->next;
    } while (temp != head);
    return false;
}

};

int main() {
    CircularLinkedList cll;
    int choice, value;
    cout << "\nMenu:\n";
    cout << "1. Insert at beginning\n";
    cout << "2. Insert at end\n";
    cout << "3. Delete by value\n";
    cout << "4. Display\n";
    cout << "5. Search\n";
    cout << "6. Exit\n";

    while (true) {
        cout << "\nEnter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value: ";
                cin >> value;
                cll.insertAtBeginning(value);
                break;
            case 2:
                cout << "Enter value: ";
                cin >> value;
                cll.insertAtEnd(value);
                break;
            case 3:
                cout << "Enter value to delete: ";
                cin >> value;
                cll.deleteByValue(value);
                break;
            case 4:
                cout << "List: ";

```

```

        cll.display();
        break;
    case 5:
        cout << "Enter value to search: ";
        cin >> value;
        if (c11.search(value)) {
            cout << "Value found in the list.\n";
        } else {
            cout << "Value not found in the list.\n";
        }
        break;
    case 6:
        cout << "Exiting...\n";
        return 0;
    default:
        cout << "Invalid choice! Please try again.\n";
    }
}
return 0;
}

```

Output:

```

Menu:
1. Insert at beginning
2. Insert at end
3. Delete by value
4. Display
5. Search
6. Exit

Enter your choice: 1
Enter value: 10

Enter your choice: 2
Enter value: 20

Enter your choice: 2
Enter value: 30

Enter your choice: 3
Enter value to delete: 20

Enter your choice: 4
List: 10 -> 30

Enter your choice: 5
Enter value to search: 30
Value found in the list.

```

2. Write a function to reverse a list. Ensure that you correctly update the links between nodes and handle the case of an empty list.

Code:

```

#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

```

```

Node* reverseList(Node* head) {
    if (head == nullptr) return nullptr; // Handle the empty list case

    Node* prev = nullptr;
    Node* curr = head;
    Node* next = nullptr;

    while (curr != nullptr) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }

    return prev;
}

void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

Node* createList(int arr[], int size) {
    if (size == 0) return nullptr;
    Node* head = new Node(arr[0]);
    Node* temp = head;
    for (int i = 1; i < size; ++i) {
        temp->next = new Node(arr[i]);
        temp = temp->next;
    }
    return head;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);

    Node* head = createList(arr, size);

    cout << "Original list: ";
    printList(head);

    head = reverseList(head);

    cout << "Reversed list: ";
    printList(head);
}

```

```
    return 0;
}
```

Output:

```
Original list: 1 2 3 4 5
Reversed list: 5 4 3 2 1
```

3. Implement stack operation using

a. Arrays

Code:

```
#include <iostream>
using namespace std;
const int MAX = 1000;

class Stack {
    int top;
    int arr[MAX];
public:
    Stack() { top = -1; }
    bool push(int x);
    int pop();
    int peek();
    bool isEmpty();
};

bool Stack::push(int x) {
    if (top >= (MAX - 1)) {
        cout << "Stack Overflow\n";
        return false;
    } else {
        arr[++top] = x;
        cout << x << " pushed into stack\n";
        return true;
    }
}

int Stack::pop() {
    if (top < 0) {
        cout << "Stack Underflow\n";
        return 0;
    } else {
        int x = arr[top--];
        return x;
    }
}

int Stack::peek() {
    if (top < 0) {
        cout << "Stack is Empty\n";
        return 0;
    }
}
```



```

        } else {
            int x = arr[top];
            return x;
        }
    }

bool Stack::isEmpty() {
    return (top < 0);
}

int main() {
    Stack stack;
    stack.push(10);
    stack.push(20);
    stack.push(30);
    cout << stack.pop() << " popped from stack\n";
    cout << "Top element is " << stack.peek() << endl;
    cout << "Stack is " << (stack.isEmpty() ? "empty" : "not empty") <<
endl;
    return 0;
}

```

Output:

```

10 pushed into stack
20 pushed into stack
30 pushed into stack
30 popped from stack
Top element is 20
Stack is not empty

```

b. Linked List

Code:

```

#include <iostream>
using namespace std;

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

    Node(int data) {
        this->data = data;
        this->next = nullptr;
    }
};

class Stack {
private:
    Node* top;
public:
    Stack() {

```

```

        top = nullptr;
    }

    void push(int data) {
        Node* newNode = new Node(data);
        newNode->next = top;
        top = newNode;
    }

    void pop() {
        if (top == nullptr) {
            cout << "Stack underflow" << endl;
            return;
        }
        Node* temp = top;
        top = top->next;
        delete temp;
    }

    int peek() {
        if (top == nullptr) {
            cout << "Stack is empty" << endl;
            return -1; // Return sentinel value indicating stack is empty
        }
        return top->data;
    }

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

int main() {
    Stack stack;

    stack.push(10);
    stack.push(20);
    stack.push(30);

    cout << "Top element is: " << stack.peek() << endl;

    stack.pop();
    cout << "Top element after pop is: " << stack.peek() << endl;

    stack.pop();
    stack.pop();
    stack.pop(); // shows "Stack underflow"
    return 0;
}

```

Output:

```
Top element is: 30
Top element after pop is: 20
Stack underflow
```

4. Write a program to convert an Infix expression to Postfix form and evaluate Postfix expression

Code:

```
#include <iostream>
#include <stack>
#include <string>
#include <cctype>
using namespace std;

int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    return 0;
}

int applyOp(int a, int b, char op) {
    switch (op) {
        case '+': return a + b;
        case '-': return a - b;
        case '*': return a * b;
        case '/': return a / b;
    }
    return 0;
}

string infixToPostfix(string infix) {
    stack<char> operators;
    string postfix;
    for (char &ch : infix) {
        if (isspace(ch)) continue;
        if (isdigit(ch)) {
            postfix += ch;
        } else if (ch == '(') {
            operators.push(ch);
        } else if (ch == ')') {
            while (!operators.empty() && operators.top() != '(') {
                postfix += operators.top();
                operators.pop();
            }
            operators.pop();
        } else {
            while (!operators.empty() && precedence(operators.top()) >=
precedence(ch)) {
                postfix += operators.top();
                operators.pop();
            }
            operators.push(ch);
        }
    }
    while (!operators.empty()) {
        postfix += operators.top();
        operators.pop();
    }
    return postfix;
}
```

```

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

int evaluatePostfix(string postfix) {
    stack<int> values;
    for (char &ch : postfix) {
        if (isdigit(ch)) {
            values.push(ch - '0');
        } else {
            int val2 = values.top(); values.pop();
            int val1 = values.top(); values.pop();
            values.push(applyOp(val1, val2, ch));
        }
    }
    return values.top();
}

int main() {
    string infix;
    cout << "Enter an infix expression: ";
    getline(cin, infix);
    string postfix = infixToPostfix(infix);
    cout << "Postfix expression: " << postfix << endl;

    int result = evaluatePostfix(postfix);
    cout << "Result of evaluation: " << result << endl;
    return 0;

    string exampleInfix = "3+(2*4)-5";
    cout << "Infix expression: " << exampleInfix << endl;

    string examplePostfix = infixToPostfix(exampleInfix);
    cout << "Converted Postfix expression: " << examplePostfix << endl;

    int exampleResult = evaluatePostfix(examplePostfix);
    cout << "Result of evaluation: " << exampleResult << endl;
}

```

Output:

```

Enter an infix expression: (5 + 3) * (8 / (4 - 2)) - 7
Postfix expression: 53+842-/ *7-
Result of evaluation: 25

```

Result: Thus, linked lists and stack functionalities were successfully implemented in C++

OOPS AND DATA STRUCTURES LABORATORY

Ex 9: Data Structures

Name: Pevin S

Roll No: 3122233002078

Section: ECE-B

1. Implement a queue operations using both array and linked list.

- **Write a function to insert an element into the queue.**
- **Write a function to search for an element in the queue.**
- **Test the insertion and search functions with various test cases.**

Array:

Code:

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

class Queue {
private:
    int front, rear, size;
    int* queue;
public:
    Queue(int s) {
        front = rear = -1;
        size = s;
        queue = new int[s];
    }

    ~Queue() {
        delete[] queue;
    }

    void enqueue(int value) {
        if ((rear + 1) % size == front) {
            cout << "Queue is full\n";
            return;
        }
        if (front == -1) front = 0;
        rear = (rear + 1) % size;
        queue[rear] = value;
        cout << "Inserted " << value << endl;
    }

    bool search(int value) {
        if (front == -1) {
            cout << "Queue is empty\n";
            return false;
        }
    }
```

```

    }
    for (int i = front; i != rear; i = (i + 1) % size) {
        if (queue[i] == value) return true;
    }
    if (queue[rear] == value) return true;
    return false;
}
};

int main() {
    int size, choice, value;
    cout << "Enter the size of the queue: ";
    cin >> size;
    Queue q(size);

    cout << "\nMenu:\n";
    cout << "1. Insert\n";
    cout << "2. Search\n";
    cout << "3. Exit\n";

    do {
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value to insert: ";
                cin >> value;
                q.enqueue(value);
                break;
            case 2:
                cout << "Enter value to search: ";
                cin >> value;
                if (q.search(value)) {
                    cout << "Value found! \n";
                } else {
                    cout << "Value not found! \n";
                }
                break;
            case 3:
                cout << "Exiting\n";
                break;
            default:
                cout << "Invalid choice!\n";
        }
    } while (choice != 3);
    return 0;
}

```

Output:

```
Enter the size of the queue: 5
Menu:
1. Insert
2. Search
3. Exit
Enter your choice: 1
Enter value to insert: 10
Inserted 10
Enter your choice: 1
Enter value to insert: 20
Inserted 20
Enter your choice: 1
Enter value to insert: 30
Inserted 30
Enter your choice: 2
Enter value to search: 30
Value found!
Enter your choice: 2
Enter value to search: 40
Value not found!
```

Linked List:

Code:

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

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

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

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

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

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

    bool search(int value) {
```

```

        Node* temp = front;
        while (temp != nullptr) {
            if (temp->data == value) {
                return true;
            }
            temp = temp->next;
        }
        return false;
    }

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

};

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

    cout << "\nMenu:\n";
    cout << "1. Insert an element into the queue\n";
    cout << "2. Search for an element in the queue\n";
    cout << "3. Display the queue\n";
    cout << "4. Exit\n";

    while (true) {
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter the value to insert: ";
                cin >> value;
                q.enqueue(value);
                break;
            case 2:
                cout << "Enter the value to search: ";
                cin >> value;
                if (q.search(value)) {
                    cout << "Element found in the queue." << endl;
                } else {
                    cout << "Element not found in the queue." << endl;
                }
            case 3:
                q.display();
                break;
            case 4:
                return 0;
            default:
                cout << "Invalid choice. Please enter a valid choice." << endl;
        }
    }
}

```



```

        }
        break;
    case 3:
        q.display();
        break;
    case 4:
        exit(0);
    default:
        cout << "Invalid choice. Please try again." << endl;
    }
}

return 0;
}

```

Output:

```

Menu:
1. Insert an element into the queue
2. Search for an element in the queue
3. Display the queue
4. Exit
Enter your choice: 1
Enter the value to insert: 10
Inserted 10 into the queue.
Enter your choice: 1
Enter the value to insert: 20
Inserted 20 into the queue.

```

```

Enter your choice: 2
Enter the value to search: 20
Element found in the queue.
Enter your choice: 2
Enter the value to search: 78
Element not found in the queue.
Enter your choice: 3
10 20

```

Q2. Implement a Binary Search Tree

- Write a function to insert a node in the BST.
- Write a function to search for a node in the BST.
- Test the insertion and search functions with different sets of input

Code:

```

#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;

    Node(int value) : data(value), left(nullptr), right(nullptr) {}
};

class BinarySearchTree {
public:
    BinarySearchTree() : root(nullptr) {}

    void insert(int value) {
        root = insertRec(root, value);
    }
}

```

```

    bool search(int value) {
        return searchRec(root, value);
    }

    void inorder() {
        inorderRec(root);
        cout << endl;
    }

private:
    Node* root;

    Node* insertRec(Node* node, int value) {
        if (node == nullptr) {
            return new Node(value);
        }
        if (value < node->data) {
            node->left = insertRec(node->left, value);
        } else if (value > node->data) {
            node->right = insertRec(node->right, value);
        }
        return node;
    }

    bool searchRec(Node* node, int value) {
        if (node == nullptr) {
            return false;
        }
        if (value == node->data) {
            return true;
        } else if (value < node->data) {
            return searchRec(node->left, value);
        } else {
            return searchRec(node->right, value);
        }
    }

    void inorderRec(Node* node) {
        if (node != nullptr) {
            inorderRec(node->left);
            cout << node->data << " ";
            inorderRec(node->right);
        }
    }
};

int main() {
    BinarySearchTree bst;
    int choice, value;
    cout << "Menu:\n";
    cout << "1. Insert\n";

```

```

cout << "2. Search\n";
cout << "3. Inorder Traversal\n";
cout << "4. Exit\n";

while (true) {
    cout << "Enter your choice: ";
    cin >> choice;
    switch (choice) {
        case 1:
            cout << "Enter value to insert: ";
            cin >> value;
            bst.insert(value);
            break;
        case 2:
            cout << "Enter value to search: ";
            cin >> value;
            cout<<(bst.search(value) ? "Found" : "Not Found")<<endl;
            break;
        case 3:
            cout << "Inorder traversal: ";
            bst.inorder();
            break;
        case 4:
            return 0;
        default:
            cout << "Invalid choice! \n";
    }
}
return 0;
}

```

Output:

Menu:	Enter your choice: 2
1. Insert	Enter value to search: 20
2. Search	Found
3. Inorder Traversal	Enter your choice: 3
4. Exit	Inorder traversal: 10 20 30
Enter your choice: 1	Enter your choice: 2
Enter value to insert: 10	Enter value to search: 25
Enter your choice: 1	Not Found
Enter value to insert: 20	Enter your choice: 2
Enter your choice: 1	Enter value to search: 30
Enter value to insert: 30	Found

OOPS AND DATA STRUCTURES LABORATORY

Ex 10: Binary Search Tree

Name: Pevin S

Roll No: 3122233002078

Section: ECE-B

Aim: To implement deletion of a specified node in a binary search tree (BST) and performing in-order traversal.

1. Implement the deletion of a specified node in a binary search tree (BST). The program should handle all three deletion scenarios: deleting a leaf node, deleting a node with one child, and deleting a node with two children. Perform in-order traversal after each deletion of node

Code:

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

struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int value) : data(value), left(nullptr), right(nullptr) {}
};

class BST {
public:
    Node* root;
    BST() : root(nullptr) {}
    void insert(int value) {
        root = insertRec(root, value);
    }

    void deleteNode(int value) {
        root = deleteRec(root, value);
    }

    void inOrderTraversal() {
        inOrderRec(root);
        cout << endl;
    }

private:
    Node* insertRec(Node* node, int value) {
        if (node == nullptr) {
            return new Node(value);
        }
    }
```

```

        if (value < node->data) {
            node->left = insertRec(node->left, value);
        } else if (value > node->data) {
            node->right = insertRec(node->right, value);
        }
        return node;
    }

Node* deleteRec(Node* root, int key) {
    if (root == nullptr) return root;

    if (key < root->data) {
        root->left = deleteRec(root->left, key);
    } else if (key > root->data) {
        root->right = deleteRec(root->right, key);
    } else {
        if (root->left == nullptr) {
            Node* temp = root->right;
            delete root;
            return temp;
        } else if (root->right == nullptr) {
            Node* temp = root->left;
            delete root;
            return temp;
        }

        Node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteRec(root->right, temp->data);
    }
    return root;
}

Node* minValueNode(Node* node) {
    Node* current = node;
    while (current && current->left != nullptr) {
        current = current->left;
    }
    return current;
}

void inOrderRec(Node* root) {
    if (root != nullptr) {
        inOrderRec(root->left);
        cout << root->data << " ";
        inOrderRec(root->right);
    }
}

};

int main() {

```

```

BST tree;
tree.insert(50);
tree.insert(30);
tree.insert(20);
tree.insert(40);
tree.insert(70);
tree.insert(60);
tree.insert(80);

cout << "In-order traversal: ";
tree.inOrderTraversal();
cout << "\nDelete 20 (leaf node)\nInorder Traversal:";
tree.deleteNode(20);
tree.inOrderTraversal();
cout << "\nDelete 30 (node with one child)\nInorder Traversal:";
tree.deleteNode(30);
tree.inOrderTraversal();
cout << "\nDelete 50 (node with two children)\nInorder Traversal: ";
tree.deleteNode(50);
tree.inOrderTraversal();
return 0;
}

```

Output:

```

In-order traversal: 20 30 40 50 60 70 80

Delete 20 (leaf node)
Inorder Traversal:30 40 50 60 70 80

Delete 30 (node with one child)
Inorder Traversal:40 50 60 70 80

Delete 50 (node with two children)
Inorder Traversal: 40 60 70 80

```

Result:

Thus, specific nodes are removed from the binary search tree, and the tree is subsequently traversed in in-order.

OOPS AND DATA STRUCTURES LABORATORY

Ex 11: Sorting

Name: Pevin S

Roll No: 3122233002078

Section: ECE-B

Aim: To implement different sorting algorithm using C++ programs.

1. Write a C++ program to implement the Bubble Sort algorithm. Include functionality to print the array before and after sorting.

Code:

```
#include <iostream>
using namespace std;
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << "Array before sorting: ";
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap arr[j] and arr[j + 1]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    cout << "Array after sorting: ";
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
    return 0;
}
```

Output:

```
Array before sorting: 64 34 25 12 22 11 90
Array after sorting: 11 12 22 25 34 64 90
```

2. Implement Selection Sort in C++. Create a function that takes an array and its size as parameters and sorts the array in ascending order.

Code:

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

```

void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        // Swap the found minimum element with the first element
        int temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}

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 n = sizeof(arr) / sizeof(arr[0]);
    selectionSort(arr, n);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}

```

Output:

```

Sorted array:
11 12 22 25 64

```

3. Write a C++ program that implements Insertion Sort. Allow the user to sort either in ascending or descending order. Display the elements before and after sorting

Code:

```

#include <iostream>
#include <vector>
using namespace std;
int main() {
    int n, order;
    cout << "Enter the number of elements: ";
    cin >> n;
    vector<int> arr(n);
    cout << "Enter the elements: ";
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }
    cout << "Enter 1 for ascending order or 2 for descending order: ";
    cin >> order;
    bool ascending = (order == 1);

```



```

    cout << "Array before sorting: ";
    for (int i : arr) {
        cout << i << " ";
    }
    cout << endl;
    for (int i = 1; i < n; ++i) {
        int key = arr[i];
        int j = i - 1;

        if (ascending) {
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
        } else {
            while (j >= 0 && arr[j] < key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
        }
        arr[j + 1] = key;
    }
    cout << "Array after sorting: ";
    for (int i : arr) {
        cout << i << " ";
    }
    cout << endl;
    return 0;
}

```

Output:

```

Enter 1 for ascending order or 2 for descending order: 2
Array before sorting: 24 265 -76 45 2
Array after sorting: 265 45 24 2 -76

```

```

Enter 1 for ascending order or 2 for descending order: 1
Array before sorting: 23 78 9 1001
Array after sorting: 9 23 78 1001

```

4. Write a C++ program that implements Merge Sort. Allow the user to input an array, and display the sorted output.

Code:

```

#include <iostream>
#include <vector>
using namespace std;
int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    vector<int> arr(n);
    cout << "Enter the elements: ";
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }
    for (int curr_size = 1; curr_size <= n - 1; curr_size = 2 *
curr_size) {

```

```

        for (int left_start = 0; left_start < n - 1; left_start += 2 *
curr_size) {
            int mid = min(left_start + curr_size - 1, n - 1);
            int right_end = min(left_start + 2 * curr_size - 1, n - 1);
            int n1 = mid - left_start + 1;
            int n2 = right_end - mid;
            vector<int> L(n1), R(n2);
            for (int i = 0; i < n1; ++i)
                L[i] = arr[left_start + i];
            for (int j = 0; j < n2; ++j)
                R[j] = arr[mid + 1 + j];
            int i = 0, j = 0, k = left_start;
            while (i < n1 && j < n2) {
                if (L[i] <= R[j]) {
                    arr[k] = L[i];
                    ++i;
                } else {
                    arr[k] = R[j];
                    ++j;
                }
                ++k;
            }
            while (i < n1) {
                arr[k] = L[i];
                ++i;
                ++k;
            }
            while (j < n2) {
                arr[k] = R[j];
                ++j;
                ++k;
            }
        }
    }
    cout << "Sorted array: ";
    for (const int& num : arr) {
        cout << num << " ";
    }
    cout << endl;
    return 0;
}

```

Output:

```

Enter the number of elements: 4
Enter the elements: 20
200
78
96
Sorted array: 20 78 96 200

```

Result:

Thus, different sorting algorithms are implemented using C++ program.

OOPS AND DATA STRUCTURES LABORATORY

Ex 12: Binary Heaps

Name: Pevin S

Roll No: 3122233002078

Section: ECE-B

Aim: To implement binary heaps using C++ program.

1. Write a C++ program to implement heap sort

Code:

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    for (int i = n / 2 - 1; i >= 0; i--) {
        int largest = i;
        int left = 2 * i + 1;
        int right = 2 * i + 2;
        if (left < n && arr[left] > arr[largest])
            largest = left;
        if (right < n && arr[right] > arr[largest])
            largest = right;
        if (largest != i) {
            swap(arr[i], arr[largest]);
            int j = largest;
            while (true) {
                int largest = j;
                int left = 2 * j + 1;
                int right = 2 * j + 2;
                if (left < n && arr[left] > arr[largest])
                    largest = left;
                if (right < n && arr[right] > arr[largest])
                    largest = right;
                if (largest != j) {
                    swap(arr[j], arr[largest]);
                    j = largest;
                } else {
                    break;
                }
            }
        }
    }

    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
```

```

    int j = 0;
    while (true) {
        int largest = j;
        int left = 2 * j + 1;
        int right = 2 * j + 2;
        if (left < i && arr[left] > arr[largest])
            largest = left;
        if (right < i && arr[right] > arr[largest])
            largest = right;
        if (largest != j) {
            swap(arr[j], arr[largest]);
            j = largest;
        } else {
            break;
        }
    }
    cout << "Sorted array is \n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
    return 0;
}

```

Output:

```

Sorted array is
5 6 7 11 12 13

```

2. Implement Selection Sort in C++. Create a function that takes an array and its size as parameters and sorts the array in ascending order.

Code:

```

#include <iostream>
#include <vector>
#include <list>

using namespace std;

class Graph {
    int V;
    vector<list<int>> adj;

public:
    Graph(int V); // Constructor
    void addEdge(int v, int w);
    void printGraph();
};

Graph::Graph(int V) {
    this->V = V;
    adj.resize(V);
}

```

```

}

void Graph::addEdge(int v, int w) {
    adj[v].push_back(w);
    adj[w].push_back(v);
}

void Graph::printGraph() {
    for (int v = 0; v < V; ++v) {
        cout << "\n Adjacency list of vertex " << v << "\n head ";
        for (auto x : adj[v])
            cout << "-> " << x;
        cout << endl;
    }
}

int main() {
    Graph g(5);
    g.addEdge(0, 1);
    g.addEdge(0, 4);
    g.addEdge(1, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 3);
    g.addEdge(3, 4);
    g.printGraph();

    return 0;
}

```

Output:

```

Adjacency list of vertex 0
head -> 1-> 4

Adjacency list of vertex 1
head -> 0-> 2-> 3-> 4

Adjacency list of vertex 2
head -> 1-> 3

Adjacency list of vertex 3
head -> 1-> 2-> 4

Adjacency list of vertex 4
head -> 0-> 1-> 3

```

Result:

Thus, binary heaps are implemented using C++ programs.