



DATA STRUCTURE AND ALGORITHM

LAB -1



AUGUST 24, 2025

Sharjeel Memon (24k-0555)

Lab Tasks

Q1. Bank Account Management System

Suppose you are developing a bank account management system, and you have defined the `BankAccount` class with the required constructors. You need to demonstrate the use of these constructors in various scenarios.

1. Default Constructor Usage: Create a default-initialized `BankAccount` object named `account1`. Print out the balance of `account1`.
2. Parameterized Constructor Usage: Create a `BankAccount` object named `account2` with an initial balance of \$1000. Print out the balance of `account2`.
3. Copy Constructor Usage: Using the `account2` you created earlier, create a new `BankAccount` object named `account3` using the copy constructor. Deduct \$200 from `account3` and print out its balance. Also, print out the balance of `account2` to ensure it hasn't been affected by the transaction involving `account3`.

Q2. Exam Class with Dynamic Memory Allocation (DMA)

Create a C++ class named "Exam" using DMA designed to manage student exam records, complete with a shallow copy implementation. Define attributes such as student name, exam date, and score within the class, and include methods to set these attributes and display exam details.

As part of this exercise, intentionally omit the implementation of the copy constructor and copy assignment operator. Afterward, create an instance of the "Exam" class, generate a shallow copy, and observe any resulting issues.

Q3. Box Class with Dynamic Memory Allocation and Rule of Three

Create a C++ class `Box` that uses dynamic memory allocation for an integer. Implement the Rule of Three by defining a destructor, copy constructor, and copy assignment operator. Demonstrate the behavior of both shallow and deep copy using test cases.

Solutions

Q1. Bank Account Management System

```
#include <iostream>
```

Output Operations

// Easy Access to Input and

```
using namespace std;
```

"std::" before every library features (like cout,cin,endl ,etc)

// So we dont have to write

```
class BankAccount {
```

Class

// Defining the BankAccount

```
    private :
```

Modifier , private, makes member accessible only inside the class

// Access

```
        double Balance;
```

// Data Member

```
    public :
```

Modifier , public, allows functions to be accessible from outside

// Access

```
};
```

//Constructors

```

    BankAccount () : Balance(00.00) {
                                                // Default Constructor - Initializes
Account with $0 if no paramater is passed!

        cout<<"Account Created!"<<endl;

    }

    BankAccount (double x) {
                                                // Parameterized Constructor - Initializes Account's
Balance with the paramater passed!

        if( x < 00.00 ) {

            Balance=00.00;

        }

        else {

            Balance=x;

        }

        cout<<"Account Created!"<<endl;

    }

    BankAccount(const BankAccount &other) {
                                                // Copy Constructor - Used to Create a copy
of another Object

        cout<<"Account Created!"<<endl;

        Balance = other.Balance;

    }

// Methods

```

```

void withdraw(double x) {
// Withdraw method -
Deducts the parameter from the Balance
    if( Balance >= x ) {
// Check if Balance is either
greater than or equal to parameter
        Balance -= x;
// Deducting paramter
from the Balance
        cout<< "Amount Deducted Successfully!"<<endl;
        return ;
// End of
Function
    }
    cout<< "Insufficient Balance!"<<endl;
// If not Sufficient Balance
    return ;
}

```

```

void deposit(double x) {
// Deposit Method - Adds
Paramter to the Balance
    if(x > 00.00) {
// Checks if the Parameter is
Positive
        Balance+=x;
        cout<< "Amount Deposited Successfully!"<<endl;
        return ;
// End of
Function
    }
    cout<< "Deposit can't be less than or equal to 0!"<<endl;
// Invalid Parameter
}

```

```

    }

    double getBalance() const {
        // Getter Method - Allows main to
        Access the Private Members
        return Balance;
    }

};

int main() {

    // Default Constructor Usage
    BankAccount account1;
    // Will Initialize with
    $0 Balance

    cout<<"Account1 Balance : $"<<account1.getBalance()<<endl<<endl;
    // Prints Balance for Account1

    // Parameterized Constructor Usage
    BankAccount account2(1000.00);
    // Will Initialize with $1000
    Balance

    cout<<"Account2 Balance : $"<<account2.getBalance()<<endl<<endl;
    // Prints Balance for Account2

    // Copy Constructor Usage
    BankAccount account3 = account2;
    // Will Initialize Balance of Account3
    with the Balance of Account2

```

```

        account3.withdraw(200.00);
                                                    // Deducting $200.00 from
Account3

        cout<<"Account3 Balance : $"<<account3.getBalance()<<endl<<endl;
                                                    // Prints Balance for Account3

        cout<<"Verifying to Show that the Balance of Account2 is Unchanged..."<<endl;

        cout<<"Account2 Balance : $"<<account2.getBalance()<<endl<<endl;
                                                    // Prints Balance for Account2

        return 0;
                                                    //End of
Program

}

```

Q1. Screenshots

```

C:\Users\opstegra\Desktop\U x + v - □ x
Account Created!
Account1 Balance : $0

Account Created!
Account2 Balance : $1000

Account Created!
Amount Deducted Successfully!
Account3 Balance : $800

Verifying to Show that the Balance of Account2 is Unchanged...
Account2 Balance : $1000

-----
Process exited after 0.449 seconds with return value 0
Press any key to continue . . . |

```

```

1 #include <iostream> // Easy Access to Input and Output Operations
2
3 using namespace std; // So we dont have to write "std:" before every library features (like cout,cin,endl ,etc)
4
5 class BankAccount { // Defining the BankAccount Class
6
7     private : // Access Modifier , private, makes member accessible only inside the class
8
9         double Balance; // Data Member
10
11     public : // Access Modifier , public, allows functions to be accessible from outside
12
13         //Constructors
14
15         BankAccount () : Balance(00.00) { // Default Constructor - Initializes Account with $0 if no parameter is passed!
16             cout<<"Account Created!"<<endl;
17         }
18
19         BankAccount (double x) : Balance(x) { // Parameterized Constructor - Initializes Account's Balance with the parameter passed!
20             cout<<"Account Created!"<<endl;
21         }
22
23         BankAccount(const BankAccount &other) { // Copy Constructor - Used to Create a copy of another Object
24             cout<<"Account Created!"<<endl;
25             Balance = other.Balance;
26         }
27
28         // Methods
29
30         void withdraw(double x) { // Withdraw method - Deducts the parameter from the Balance
31             if( Balance >= x ) { // Check if Balance is either greater than or equal to parameter
32                 Balance -= x; // Deducting parameter from the Balance
33                 cout<< "Amount Deducted Successfully!"<<endl;
34                 return ; // End of Function
35             }
36             cout<< "Insufficient Balance!"<<endl;
37             return ; // If not Sufficient Balance
38         }
39
40         void deposit(double x) { // Deposit Method - Adds Parameter to the Balance
41             if(x > 00.00) { // Checks if the Parameter is Positive
42                 Balance+=x;
43                 cout<< "Amount Deposited Successfully!"<<endl;
44                 return ; // End of Function
45             }
46             cout<< "Deposit can't be less than or equal to 0!"<<endl; // Invalid Parameter
47         }
48
49         double getBalance() const { // Getter Method - Allows main to Access the Private Members
50             return Balance;
51         }
52     };
53
54 int main() {
55
56     // Default Constructor Usage
57     BankAccount account1; // Will Initialize with $0 Balance
58     cout<<"Account1 Balance : $"<<account1.getBalance()<<endl<<endl; // Prints Balance for Account1
59
60     // Parameterized Constructor Usage
61     BankAccount account2(1000.00); // Will Initialize with $1000 Balance
62     cout<<"Account2 Balance : $"<<account2.getBalance()<<endl<<endl; // Prints Balance for Account2
63
64     // Copy Constructor Usage
65     BankAccount account3 = account2; // Will Initialize Balance of Account3 with the Balance of Account2
66     account3.withdraw(200.00); // Deducting $200.00 from Account3
67     cout<<"Account3 Balance : $"<<account3.getBalance()<<endl<<endl; // Prints Balance for Account3
68
69     cout<<"Verifying to Show that the Balance of Account2 is Unchanged..."<<endl;
70     cout<<"Account2 Balance : $"<<account2.getBalance()<<endl<<endl; // Prints Balance for Account2
71
72     return 0; //End of Program
73 }

```


Q2. Exam Class with Dynamic Memory Allocation (DMA)

```
#include <iostream>
```

```
Operations // Easy Access to Input and Output
```

```
using namespace std;
```

```
every library feature (like cout, cin, endl, etc) // So we don't have to write "std::" before
```

```
class Exam {
```

```
// Defining the Exam Class
```

```
private:
```

```
makes member accessible only inside the class // Access Modifier: private,
```

```
char* name;
```

```
memory for student name // Data Member: dynamically allocated
```

```
char* date;
```

```
memory for exam date // Data Member: dynamically allocated
```

```
int score;
```

```
// Data Member: storing exam score
```

```
public:
```

```

// Access Modifier: public,
allows functions to be accessible from outside

// Setters
void setScore(int x ) {

    // Method to set exam score

    score = x ;

    // Assign new score

}

void setDate(const char* x ) {

    //
Method to set exam date

    strcpy(date, x );

    // Copy new date into allocated memory

}

void setName(const char* x) {

    //
Method to set student's name

    if (name != nullptr) {

        // If memory already allocated, free it

        delete[] name;

        // Free old memory before setting new name

    }

    name = new char[strlen( x ) + 1];

    // Allocate memory
for new name

```

```

        strcpy(name, x );

                                                    // Copy new name into allocated memory
    }

// Constructors
Exam(const char* x, const char* y, int z) {
                                                    // Parameterized Constructor

    name = new char[strlen( x ) + 1];
                                                    // Allocating memory
for the student name

    strcpy(name, x );

                                                    // Copying student name into the allocated
memory

    date = new char[strlen( y ) + 1];
                                                    // Allocating memory
for the exam date

    strcpy(date, y );

                                                    // Copying exam date into the allocated memory

    score = z;

                                                    // Assigning the exam score

    cout << "Exam Initialized!" << endl<<endl;
                                                    // Message when
exam is initialized

}

// Display Method
void displayDetails() const {
                                                    //
Method to display student's exam details

```

```
    cout << "Student: " << name << endl << "Exam Date: " << date << endl << "Score: " <<
score << endl; // Print details of the exam
```

```
}
```

```
// Destructor
```

```
~Exam() {
```

```
    // Destructor to free
```

```
dynamically allocated memory
```

```
    string cleanupMessage = (name) ? name : "Unknown OR Nullptr";
```

```
    // Prepare the cleanup message
```

```
    cout << "Cleaning up memory for: " << cleanupMessage << endl;
```

```
    // Display cleanup message
```

```
    delete[] name;
```

```
    // Free memory for name
```

```
    delete[] date;
```

```
    // Free memory for exam date
```

```
}
```

```
};
```

```
int main() {
```

```
    // Creating Exam Object Using Parameterized Constructor
```

```
    cout << "-----" << endl;
```

```
    cout << "Creating Exam Object student1" << endl;
```

```
    Exam student1("Sharjeel Memon", "2025-05-30", 85);
```

```
    // Creates Exam
```

```
object with initial values
```

```

student1.displayDetails();

// Display the details of student1

cout << endl;

// Creating a Shallow Copy of student1
cout << "-----" << endl;
cout << "Creating a Shallow Copy of student1 into student2" << endl;
Exam student2 = student1;

// Creates student2 as a shallow copy of student1

student2.displayDetails();

// Display the details of student2

cout << endl;

// Case Number 1: Shallow Copy Issue - Dangling Pointer
cout << "-----" << endl;
cout << "Case Number 1: Dangling Pointer Issue when name is changed" << endl;
student2.setName("Muhammad Haneef");
// Changing
the name of student2

student2.displayDetails();

// Display student2's details after name change

cout << "Student1 Details After student2 Name Change" << endl << endl;

student1.displayDetails();

// student1 shows garbage value for the name
(dangling pointer)

cout << endl;

```

```

// Case Number 2: Shared Memory Issue - Exam Date is shared

cout << "-----" << endl;

cout << "Case Number 2: Both objects share the same memory for exam date" << endl;

student2.setDate("2025-06-15");

// Changing
the exam date of student2

student2.displayDetails();

// Display student2's details after changing the
date

cout << endl;

cout << "Student1 Details After student2 Date Change" << endl;

student1.displayDetails();

// student1's exam date also changes as they share
the same memory

cout << endl << endl;

return 0;

// End of Program

}

```

Q2. Screenshots

```
C:\Users\opstegra\Desktop\Q  X + v

Creating Exam Object student1
Exam Initialized!

Student: Sharjeel Memon
Exam Date: 2025-05-30
Score: 85

Creating a Shallow Copy of student1 into student2
Student: Sharjeel Memon
Exam Date: 2025-05-30
Score: 85

Case Number 1: Dangling Pointer Issue when name is changed
Student: Muhammad Haneef
Exam Date: 2025-05-30
Score: 85
Student1 Details After student2 Name Change

Student: Muhammad Haneef
Exam Date: 2025-05-30
Score: 85

Case Number 2: Both objects share the same memory for exam date
Student: Muhammad Haneef
Exam Date: 2025-06-15
Score: 85

Student1 Details After student2 Date Change
Student: Muhammad Haneef
Exam Date: 2025-06-15
Score: 85

Cleaning up memory for: Muhammad Haneef
Cleaning up memory for: =

Process exited after 1.678 seconds with return value 3221226356
Press any key to continue . . . |
```

```
1 #include <iostream> // Easy Access to Input and Output Operations
2
3 using namespace std; // So we don't have to write "std::" before every library feature (like cout, cin, endl, etc)
4
5 class Exam { // Defining the Exam Class
6
7     private: // Access Modifier: private, makes member accessible only inside the class
8         char* name; // Data Member: dynamically allocated memory for student name
9         char* date; // Data Member: dynamically allocated memory for exam date
10        int score; // Data Member: storing exam score
11
12    public: // Access Modifier: public, allows functions to be accessible from outside
13
14        // Setters
15        void setScore(int x) { // Method to set exam score
16            score = x; // Assign new score
17        }
18
19        void setDate(const char* x) { // Method to set exam date
20            strcpy(date, x); // Copy new date into allocated memory
21        }
22
23        void setName(const char* x) { // Method to set student's name
24            if (name != nullptr) { // If memory already allocated, free it
25                delete[] name; // Free old memory before setting new name
26            }
27            name = new char[strlen( x ) + 1]; // Allocate memory for new name
28            strcpy(name, x); // Copy new name into allocated memory
29        }
30
31    // Constructors
32    Exam(const char* x, const char* y, int z) { // Parameterized Constructor
33        name = new char[strlen( x ) + 1]; // Allocating memory for the student name
34        strcpy(name, x); // Copying student name into the allocated memory
35        date = new char[strlen( y ) + 1]; // Allocating memory for the exam date
36        strcpy(date, y); // Copying exam date into the allocated memory
37        score = z; // Assigning the exam score
38        cout << "Exam Initialized!" << endl<<endl; // Message when exam is initialized
39    }
40
41    // Display Method
42    void displayDetails() const { // Method to display student's exam details
43        cout << "Student: " << name << endl << "Exam Date: " << date << endl << "Score: " << score << endl; // Print details of the exam
44    }
45
46    // Destructor
47    ~Exam() { // Destructor to free dynamically allocated memory
48        string cleanupMessage = (name) ? name : "Unknown OR Nullptr"; // Prepare the cleanup message
49        cout << "Cleaning up memory for: " << cleanupMessage << endl; // Display cleanup message
50        delete[] name; // Free memory for name
51        delete[] date; // Free memory for exam date
52    }
53
54 };
55
```

```

56 int main() {
57
58     // Creating Exam Object Using Parameterized Constructor
59     cout << "-----" << endl;
60     cout << "Creating Exam student1" << endl;
61     Exam student1("Sharjeel Memon", "2025-05-30", 85); // Creates Exam object with initial values
62     student1.displayDetails(); // Display the details of student1
63     cout << endl;
64
65     // Creating a Shallow Copy of student1
66     cout << "-----" << endl;
67     cout << "Creating a Shallow Copy of student1 into student2" << endl;
68     Exam student2 = student1; // Creates student2 as a shallow copy of student1
69     student2.displayDetails(); // Display the details of student2
70     cout << endl;
71
72     // Case Number 1: Shallow Copy Issue - Dangling Pointer
73     cout << "-----" << endl;
74     cout << "Case Number 1: Dangling Pointer Issue when name is changed" << endl;
75     student2.setName("Muhammad Haneef"); // Changing the name of student2
76     student2.displayDetails(); // Display student2's details after name change
77     cout << "Student1 Details After student2 Name Change" << endl << endl;
78     student1.displayDetails(); // student1 shows garbage value for the name (dangling pointer)
79     cout << endl;
80
81     // Case Number 2: Shared Memory Issue - Exam Date is shared
82     cout << "-----" << endl;
83     cout << "Case Number 2: Both objects share the same memory for exam date" << endl;
84     student2.setDate("2025-06-15"); // Changing the exam date of student2
85     student2.displayDetails(); // Display student2's details after changing the date
86     cout << endl;
87
88     cout << "Student1 Details After student2 Date Change" << endl;
89     student1.displayDetails(); // student1's exam date also changes as they share the same memory
90     cout << endl << endl;
91
92     return 0; // End of Program
93 }
94

```


Q3. Box Class with Dynamic Memory Allocation and Rule of Three

```
#include <iostream>

Operations // Easy Access to Input and Output

using namespace std;

// So we don't have to write "std::" before
every library feature (like cout, cin, endl, etc)

class Box {

// Defining the Box Class

private:

// Access Modifier,
private, makes member accessible only inside the class

    int* BoxSize;

// Data Member: dynamically
allocated memory for BoxSize

    bool deepCopy;

// A flag for deep copy vs shallow
copy (runtime controlled)

public:

// Access
Modifier, public, allows functions to be accessible from outside
```

// Constructors

Box() : BoxSize(new int(0)), deepCopy(true) {

// Default Constructor

- Initializes BoxSize with 0 if no parameter is passed

cout << "Box Created with default size!" << endl;

// Message when the box is created

}

Box(int size) : BoxSize(new int(size)), deepCopy(true) {

// Parameterized Constructor -

Initializes BoxSize with the given size

cout << "Box Created with size " << *BoxSize << endl;

// Message when the box is created

with a specified size

}

// Copy Constructor

Box(const Box &other) {

// Copy Constructor - Used to Create a copy of

another Object

if (other.deepCopy) {

BoxSize = new int(*(other.BoxSize));

// Deep copy: allocating new

memory and copying the value of BoxSize

cout << "Box Created by copying with size " << *BoxSize << endl;

// Message when a box is created by copying

another box

} else {

BoxSize = other.BoxSize;

// Shallow

copy: both objects point to the same memory

```

        cout << "Box Created by shallow copying with size " << *BoxSize << endl;
        // Message when a box is created by shallow
copying another box
    }
}

// Methods

void setBoxSize(int size) {
    //
Method to set the size of the Box
    *BoxSize = size;

    // Assigning the given size to BoxSize
}

int getBoxSize() const {
    // Getter Method - Allows main to Access the
Private Member
    return *BoxSize;

    // Returning the size of the Box
}

// Setter for deepCopy flag (to change at runtime)
void setDeepCopy(bool value) {
    deepCopy = value;

    // Allow modification of deepCopy flag at runtime
}

```

```

// Copy Assignment Operator

Box& operator=(const Box& other) {
                                                                    // Assignment
operator to handle assignment between two Box objects

    if (this != &other) {

                                                                    // Check for self-assignment

        delete BoxSize;

                                                                    // Free the existing memory

        if (other.deepCopy) {

            BoxSize = new int(*(other.BoxSize));
                                                                    // Perform deep copy

            cout << "Box Assigned with size " << *BoxSize << endl;
                                                                    // Message after deep copy assignment

        } else {

            BoxSize = other.BoxSize;
                                                                    // Perform shallow
copy
            cout << "Box Assigned by shallow copy with size " << *BoxSize << endl;
                                                                    // Message after shallow copy assignment

        }

    }

    return *this;

                                                                    // Returning the current object

}

// Destructor

~Box() {

```

```

// Destructor to free
dynamically allocated memory

delete BoxSize;

// Free memory for BoxSize

cout << "Box Destroyed" << endl;
// Message when the
box is destroyed
}

};

int main() {

    cout << "---- Deep Copy Example ----" << endl;

    Box box1(10);

    // Will initialize with BoxSize 10

    cout << "Box1 (Initial) Size: " << box1.getBoxSize() << endl;
    // Prints Box1's size

    Box box2(box1);

    // Deep copy using the copy constructor
(deepCopy is true by default)

    cout << "Box2 (After Deep Copy) Size: " << box2.getBoxSize() << endl;
    // Should be 10, independent from Box1

    box2.setBoxSize(20);

    // Modify box2's size

    cout << "Box1 Size (After Box2 Change): " << box1.getBoxSize() << endl;
    // Should remain 10, no effect due to deep
copy

```

```

cout << "Box2 Size (After Change): " << box2.getBoxSize() << endl;
// Should be 20 (modified)

cout << endl;

// New Line for clarity

cout << "---- Shallow Copy Example ----" << endl;

Box box3(30);

// Will Initialize with BoxSize 30

cout << "Box3 (Initial) Size: " << box3.getBoxSize() << endl;
// Prints Box3's size

// Set deepCopy to false at runtime to demonstrate shallow copy
box3.setDeepCopy(false);

// Change to shallow copy mode at runtime

Box box4 = box3;

// Shallow copy using the copy constructor

cout << "Box4 (After Shallow Copy) Size: " << box4.getBoxSize() << endl;
// Should be 30, both Box3 and Box4 point to the
same memory

box4.setBoxSize(40);

// Modify box4's size

cout << "Box3 Size (After Box4 Change): " << box3.getBoxSize() << endl;
// Should also be 40, since they share
memory

cout << "Box4 Size (After Change): " << box4.getBoxSize() << endl << endl;
// Should be 40 (modified)

```

```

// Copy Assignment Example (Deep Copy)

cout << "---- Copy Assignment Example (Deep Copy) ----" << endl;

Box box5(50);

box5 = box1;

// Deep copy using the assignment
operator

cout << "Box5 Size (After Deep Copy Assignment): " << box5.getBoxSize() << endl;
// Should be 10

box1.setBoxSize(40);

// Changing Size to 40

cout << "Box1 Size (After Change): " << box1.getBoxSize() << endl;
// Should be 40

cout << "Box5 Size (After Changing Box1 Size): " <<
box5.getBoxSize() << endl << endl; // Should still be 10 (deep copy)


// Copy Assignment Example (Shallow Copy)

cout << "---- Copy Assignment Example (Shallow Copy) ----" << endl;

Box box6(60);

box6.setDeepCopy(false);

// Set shallow copy mode at runtime

box6 = box3;

// Shallow copy using the assignment
operator (box3 and box6 will share memory)

box6.setBoxSize(70);

// Modify box6

cout << "Box3 Size (After Shallow Copy Assignment): " << box3.getBoxSize() << endl;
// Should be 70 (shared memory with box6)

```

```
cout << "Box6 Size (After Change): " << box6.getBoxSize() << endl << endl;  
    // Should be 70 (modified)
```

```
return 0;
```

```
    // End of Program
```

```
}
```


Q3 . Screenshots

```
1 #include <iostream> // Easy Access to Input and Output Operations
2
3 using namespace std; // So we don't have to write "std::" before every library feature (like cout, cin, endl, etc)
4
5 class Box { // Defining the Box Class
6
7     private : // Access Modifier, private, makes member accessible only inside the class
8         int* BoxSize; // Data Member: dynamically allocated memory for BoxSize
9
10    public : // Access Modifier, public, allows functions to be accessible from outside
11
12        // Constructors
13
14        Box() : BoxSize(new int(0)) { // Default Constructor - Initializes BoxSize with 0 if no parameter is passed
15            cout << "Box Created with default size!" << endl; // Message when the box is created
16        }
17
18        Box(int size) : BoxSize(new int(size)) { // Parameterized Constructor - Initializes BoxSize with the given size
19            cout << "Box Created with size " << *BoxSize << endl; // Message when the box is created with a specified size
20        }
21
22        Box(const Box &other, bool deepCopy = true) { // Copy Constructor - Used to Create a copy of another Object
23            if (deepCopy) { // Deep copy: allocating new memory and copying the value of BoxSize
24                BoxSize = new int(*other.BoxSize); // Message when a box is created by copying another box
25                cout << "Box Created by copying with size " << *BoxSize << endl;
26            } else { // Shallow copy: both objects point to the same memory
27                BoxSize = other.BoxSize; // Message when a box is created by shallow copying another box
28                cout << "Box Created by shallow copying with size " << *BoxSize << endl;
29            }
30        }
31
32        // Methods
33
34        void setBoxSize(int size) { // Method to set the size of the Box
35            *BoxSize = size; // Assigning the given size to BoxSize
36        }
37
38        int getBoxSize() const { // Getter Method - Allows main to Access the Private Member
39            return *BoxSize; // Returning the size of the Box
40        }
41
42        // Copy Assignment Operator - Used to assign values from one object to another
43        Box& operator=(const Box& other) { // Assignment operator to handle assignment between two Box objects
44            bool deepCopy = true; // Set to true for deep copy, false for shallow copy
45            if (this != &other) { // Check for self-assignment
46                delete BoxSize; // Free the existing memory
47                if (deepCopy) { // Perform deep copy
48                    BoxSize = new int(*other.BoxSize); // Message after deep copy assignment
49                    cout << "Box Assigned with size " << *BoxSize << endl;
50                } else { // Perform shallow copy
51                    BoxSize = other.BoxSize; // Message after shallow copy assignment
52                    cout << "Box Assigned by shallow copy with size " << *BoxSize << endl;
53                }
54            }
55            return *this; // Returning the current object
56        }
57
58        // Destructor
59        ~Box() { // Destructor to free dynamically allocated memory
60            delete BoxSize; // Free memory for BoxSize
61            cout << "Box Destroyed" << endl; // Message when the box is destroyed
62        }
63}
```

```

64 };
65
66 int main() {
67     // Deep Copy Example (Copy Constructor with deepCopy = true)
68     cout << "---- Deep Copy Example ----" << endl;
69     Box box1(10); // Will initialize with BoxSize 10
70     Box box2(box1, true); // Deep copy using the flag (true for deep copy)
71     box2.setBoxSize(20); // Modify box2
72     cout << "Box1 Size: " << box1.getBoxSize() << endl; // Should remain 10
73     cout << "Box2 Size: " << box2.getBoxSize() << endl << endl; // Should be 20
74
75     // Shallow Copy Example (Copy Constructor with deepCopy = false)
76     cout << "---- Shallow Copy Example ----" << endl;
77     Box box3(30); // Will initialize with BoxSize 30
78     Box box4(box3, false); // Shallow copy using the flag (false for shallow copy)
79     box4.setBoxSize(40); // Modify box4
80     cout << "Box3 Size: " << box3.getBoxSize() << endl; // Should change to 40 (shared memory)
81     cout << "Box4 Size: " << box4.getBoxSize() << endl << endl; // Should also be 40 (shared memory)
82
83     // Copy Assignment Example (Deep Copy)
84     cout << "---- Copy Assignment Example (Deep Copy) ----" << endl;
85     Box box5(50);
86     box5 = box1; // Deep copy using the assignment operator
87     cout << "Box5 Size: " << box5.getBoxSize() << endl << endl; // Should be 10
88
89     // Copy Assignment Example (Shallow Copy)
90     cout << "---- Copy Assignment Example (Shallow Copy) ----" << endl;
91     Box box6(60);
92     box6 = box3; // Shallow copy using the assignment operator (box3 and box6 will share memory)
93     box6.setBoxSize(70); // Modify box6
94     cout << "Box3 Size: " << box3.getBoxSize() << endl; // Should be 70 (shared memory with box6)
95     cout << "Box6 Size: " << box6.getBoxSize() << endl << endl; // Should be 70
96
97     return 0; // End of Program
98 }
99
100

```

```

C:\Users\opstegra\Desktop\D  X + v
---- Deep Copy Example ----
Box Created with size 10
Box1 (Initial) Size: 10
Box Created by copying with size 10
Box2 (After Deep Copy) Size: 10
Box1 Size (After Box2 Change): 10
Box2 Size (After Change): 20

---- Shallow Copy Example ----
Box Created with size 30
Box3 (Initial) Size: 30
Box Created by shallow copying with size 30
Box4 (After Shallow Copy) Size: 30
Box3 Size (After Box4 Change): 40
Box4 Size (After Change): 40

---- Copy Assignment Example (Deep Copy) ----
Box Created with size 50
Box Assigned with size 10
Box5 Size (After Deep Copy Assignment): 10
Box1 Size (After Change): 40
Box5 Size (After Changing Box1 Size): 10

---- Copy Assignment Example (Shallow Copy) ----
Box Created with size 60
Box Assigned by shallow copy with size 40
Box3 Size (After Shallow Copy Assignment): 70
Box6 Size (After Change): 70

Box Destroyed
Box Destroyed

-----
Process exited after 4.084 seconds with return value 3221226356
Press any key to continue . . . |

```