TASK 1

Q1. 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.

#include<iostream>

using namespace std;

class BankAccount{

    double balance;

    public:

    BankAccount(){

        balance = 0;

    }

    BankAccount(double amount): balance(amount){}

    BankAccount(BankAccount &obj){

        balance = obj.balance;

    }

    void printBalance(){

        cout<<"Current balance is: "<<balance<<endl;

    }

    void deductBalance(double amount){

        if (amount<=balance)

        {

            balance-=amount;

        }

    }

};

int main(){

    BankAccount account1;

    account1.printBalance();

    BankAccount account2(1000);

    account2.printBalance();

    BankAccount account3;

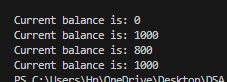
    account3 = account2;

    account3.deductBalance(200);

    account3.printBalance();

    account2.printBalance();

}



TASK 2:

Q2. 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?

#include<iostream>

using namespace std;

class Exam{

    string \*name;

    string \*date;

    double \*score;

    public:

        Exam(){

            name = new string;

            date = new string;

            score = new double;

        }

        void setName(string n){

            \*name = n;

        }

        void setDate(string d){

            \*date = d;

        }

        void setScore(double f){

            \*score = f;

        }

        void dispayExamDetails(){

            cout<<"Student Name: "<<\*name<<endl;

            cout<<"Exam Date: "<<\*date<<endl;

            cout<<"Score: "<<\*score<<endl<<endl;

        }

        ~Exam(){

            delete name;

            delete date;

            delete score;

        }

};

int main(){

    Exam e;

    e.setName("Umer");

    e.setDate("11-12-1999");

    e.setScore(54.32);

    e.dispayExamDetails();

    Exam e1 = e;

    e1.setName("Arif");

    e1.setScore(98);

    e.dispayExamDetails();

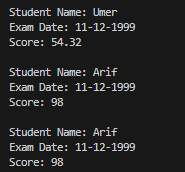
    e1.dispayExamDetails();

}

// omiting copy constructor and copy assignment but having destructor for shallow copy

                               // will destroy rule of three the destructor call for second object will try to delete

                               // the memory which is already deleted since they both are pointing towards the same memory // Also all the changes in any object will reflect on other also



TASK 3:

Q3. 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.

#include<iostream>

using namespace std;

class Box{

    int \*a;

    public:

        Box(int v = 0){

            a = new int (v);

        }

        Box(Box &obj){

            a = new int(\*obj.a);

            cout<<"Copy Constructor Called\n";

        }

        void operator=(Box &obj){

            a = new int(\*obj.a);

            cout<<"Assignment Operator Called\n";

        }

        void display()

        {

            cout<<"Value is: "<<\*a<<endl;

        }

        void changeValue(int ab){

            \*a = ab;

        }

        ~Box(){

            delete a;

        }

};

    int main(){ // for shallow copy comment out copy constructor

        Box b1(10);

        Box b2 = b1;

        Box b3;

        b3 = b2;

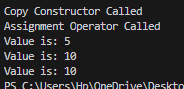
        b1.changeValue(5);

        b1.display();

        b2.display();

        b3.display();

}



TASK 4:

You are working as a software developer in a company that specializes in managing e-commerce platforms. The company is developing a Product Catalog System that helps in maintaining the details of products, including their prices, stock levels, and related offers.

Create the Product class with the following:

 Constructor to initialize the product attributes: name, price, and stock.

 display() method to print the product details.

Create the Bundle class:

 Constructor to initialize the list of products and the discount percentage.

 Methods to add products to the bundle.

 A method applyDiscount() to apply a discount to all products in the bundle.

 A method display() to print the details of the bundle, including the product details and the applied discount.

Implement Shallow and Deep Copy Create a type of copy of a bundle inn such a way that the new bundle should share the same

product list as the original bundle. Any changes made to the product list in one bundle should affect the other.

Create a such type of copy of a bundle that the new bundle should create its own independent

copy of the product list, so changes made to one bundle’s product list should not affect the other.

Test and Display:

 Create a main() function that:

 Creates a Bundle object and adds a few Product objects to it.

 Creates both shallow and deep copies of the bundle.

 Modifies the original bundle and shows how the shallow

#include <iostream>

using namespace std;

class Product

{

  string name;

  double price;

  int stock;

public:

  Product(string n = "", double p = 0, int s = 0)

  {

    name = n;

    price = p;

    stock = s;

  }

  void printDetails()

  {

    cout << "Product Name: " << name << endl;

    cout << "Product Price: " << price << endl;

    cout << "Product Stock: " << stock << endl;

  }

  void changePrice(float d)

  {

    price -= price \* d / 100;

  }

  void changeName(string n)

  {

    name = n;

  }

};

class Bundle

{

  float discount;

  Product \*products;

  int count;

public:

  Bundle(Bundle &obj)

  {

    this->discount = obj.discount;

    this->count = obj.count;

    products = new Product[count];

    for (int i = 0; i < count; i++)

    {

      products[i] = obj.products[i];

    }

  }

  Bundle(Product \*l, int num, float d) : products(l), discount(d), count(num) {}

  void addProduct(Product &p)

  {

    Product \*temp = new Product[count + 1];

    for (int i = 0; i < count; i++)

    {

      temp[i] = products[i];

    }

    temp[count] = p;

    delete[] products;

    products = temp;

    count++;

  }

  void applyDiscount()

  {

    for (int i = 0; i < count; i++)

    {

      products[i].changePrice(discount);

    }

  }

  void display()

  {

    cout << "\tBUNDLE:\n\nApplied Discount: " << discount << "%\n";

    for (int i = 0; i < count; i++)

    {

      cout << "Product " << i + 1 << endl;

      products[i].printDetails();

      cout << endl;

    }

    cout << "\n\n";

  }

  void changeNameOfProduct(string n, int productNumber)

  {

    products[productNumber].changeName(n);

  }

  ~Bundle()

  {

    delete[] products;

  }

};

int main()

{

  Product p1("Socks", 150, 3);

  Product p2("Socks", 150, 5);

  Product p3("Pen", 50, 3);

  Product \*list = new Product[3];

  list[0] = p1; // for shallow copy comment out the copy constructor and run again

                // the name of product index 0 will be different in deep copy

  list[1] = p2;

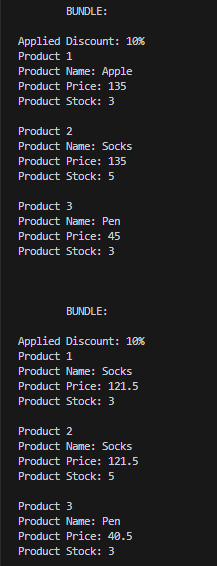
  list[2] = p3;

  Bundle b1(list, 3, 10);

  b1.applyDiscount();

  Bundle b2 = b1;

  b1.changeNameOfProduct("Apple", 0); // change product indexed 0 name from b1 list

  b2.applyDiscount();

  b1.display();

  b2.display();

}