CMPT 135 Week 9 Lab Work

1. Consider the following classes A, B and C

```
#include <iostream>
using namespace std;
class A
{
    public:
        A()
        {
            cout << "Constructing A object" << endl;
        }
        ~A()
        {
            cout << "Destructing A object" << endl;
        }
};
```

```
class B : public A
{
    public:
        B() : A()
        {
            cout << "Constructing B object" << endl;
        }
        ~B()
        {
            cout << "Destructing B object" << endl;
        }
};
```

```
class C : public B
{
    public:
        C() : B()
        {
            cout << "Constructing C object" << endl;
        }
        ~C()
        {
            cout << "Destructing C object" << endl;
        }
};
```

```
int main()
{
    C c;
    c.~C();

    system("Pause");
    return 0;
}
```

What will be the output of the main program? **Hint:-** During construction of an object c, remember that before we enter into the block of the constructor member function of class C, we will first jump to constructor of class B. Similarly before entering the block of the constructor of class B, we will first jump to the constructor of class A. However, the destructor does not follow similar path.

2. What would be the output if the main program was as follows?

```
int main()
{
    C* c;
    c = new C();
    delete c;

    system("Pause");
    return 0;
}
```

3. What would be the output if the main program was as follows?

```
int main()
{
    A* a;
    a = new C();
    delete a;

    system("Pause");
    return 0;
}
```

What do you conclude from this question? Is there a potentially memory leak in the program?

4. Consider a class named **BankAccount**. A **BankAccount** class may have two member variables: **accountholder** (string data type), **balance** (double data type). Implement the class **BankAccount** with all the design concepts you learned during the lectures.

Next consider a class named **ChequingAccount.** Clearly a **ChequingAccount** is a **BankAccount**. A **ChequingAccount** may have one additional member variable which is the maximum number of transactions (**maxNumTransactions** int data type) allowed per month. Moreover a ChequingAccount should two more member functions namely

- `void deposit(const double &amount);`
- `void withdraw(const double &amount);`

The deposit member function adds the value of its parameter to the account balance while the withdraw member function deducts the value of its parameter from the account balance. However these member functions must first check if the account holder has reached maximum number of transactions. If maximum number of transactions is reached then these member functions must inform the account holder that the maximum number of transactions is reached and deny the deposit/withdraw action. If on the other hand the maximum number of transactions is not reached; then these member functions must perform the deposit/withdraw action and count the action.

But then we don't have a member variable to keep track of the count. This means we need one more member variable to keep track of the count of transactions (**numTransactions** int data type). This count should be initialized to zero whenever an object is constructed and incremented by 1 in the deposit and withdraw member functions. This means whenever a non-default ChequingAccount is constructed, it should take only three arguments; namely account holder name, account balance, and maximum number of transactions; and it should set the count of transactions to zero.

Finally add one more member function named

- `void advanceMonth();`

This member function takes no argument and resets the count of transactions to zero. This is the same as saying we are starting a new month (i.e. next month) and thus the count of transactions is reset for the new month.

Implement a **ChequingAccount** class by extending it from **BankAccount** class.

Finally consider **SavingAccount** which is a **BankAccount** again. **SavingAccount** may have two more additional member variables which are daily interest rate (**interestRate** double data type) and the number of days since the account was opened (**numDays** int data type). Implement a **SavingAccount** class by extending it from **BankAccount** class. This class must override the **getBalance()** member function defined in the base class and must return the account balance plus the interest accumulated calculated with simple interest model. SavingAccount does not allow deposit or withdraw operations.

Write an appropriate test main program to test you classes and inheritance relationships.

5. We have seen in the lecture that a subclass inherits all the member variables of the base class and may add additional member variables of its own. Sometimes, a subclass may NOT have any additional member variable; instead it may have additional behavior. Behavior means, the requirements or restrictions that must be respected for objects in order to be valid objects from real world application point of view.

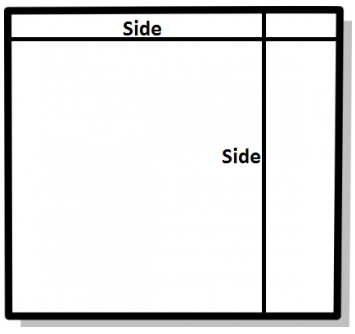
Example, consider a class named **MyRectangle** to represent rectangle objects and that has two member variables named length and width. Then we may have the following class declaration for MyRectangle class


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

class MyRectangle
{
private:
    double length, width;
public:
    //Constructors
    MyRectangle();
    MyRectangle(const double &len, const double &wid);
    MyRectangle(const MyRectangle &r);

    //Getters
    double getLength() const;
    double getWidth() const;

    //Setters
    void setLength(const double &len);
    void setWidth(const double &wid);

    //Other member functions
    double getArea() const;
    double getPerimeter() const;
    friend ostream& operator << (ostream &out, const MyRectangle &r);
    friend istream& operator >> (istream &in, MyRectangle &r);
};
```

Implement of the **MyRectangle** class.

Now consider a class named **MySquare** to represent square objects. Mathematically we know that every square is a rectangle. Hence there is the relationship square IS-A rectangle. This means we may use inheritance to build the **MySquare** class.

Now, the question is does the **MySquare** class have any additional member variable? The answer is NO because any square object has only length and width attributes with the additional requirement (behavior or restriction or constraint) that the length is always equal to the width.



So how should we design the **MySquare** class? It will make sense to implement it as a subclass of the **MyRectangle** class. That way, we will safely use the getArea and getPerimiter member functions of the **MyRectangle** class with square objects. But then we must be careful when we call setWidth or setLength member functions with square objects in order to keep the constraints of square objects valid. This means calling the setLength member function for a square object must perform not only assigning the parameter to length member variable (which is what setLength member function in the **MyRectangle** class does) BUT ALSO ASSIGN THIS LENGTH TO THE WIDTH MEMBER VARIABLE TOO. This means we must override the setLength and setWidth member functions in the base class with new implementation in the subclass. Therefore the declaration of the **MySquare** class must look like as follows:

```
class MySquare : public MyRectangle
{
public:
    //Constructors
    MySquare();
    MySquare(const double &side);
    MySquare(const MySquare &s);

    //Getters
    double getSide() const;

    //Setters
    void setLength(const double &len);
    void setWidth(const double &wid);
    void setSide(const double &side);

    //Other member functions
    friend ostream& operator << (ostream &out, const MySquare &s);
    friend istream& operator >> (istream &in, MySquare &s);
};
```

Give the implementation of the **MySquare** class making sure the constraints of square objects are always preserved.

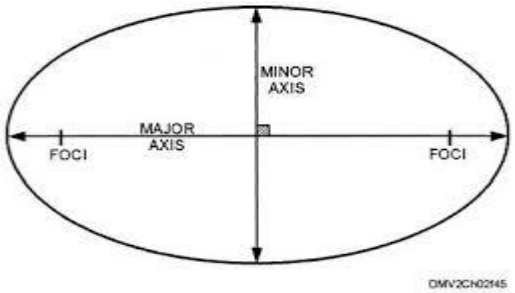
Use the following main program and its output to test your classes.

```
int main()
{
    //Test Constructors, getters and output streaming operator
    MyRectangle r1;
    MyRectangle r2(2, 3);
    MyRectangle r3 = r2;
    cout << "r1: " << r1 << endl;
    cout << "r2: " << r2 << endl;
    cout << "r3: " << r3 << endl;

    MySquare s1;
    MySquare s2(3);
    MySquare s3 = s2;
    cout << "s1: " << s1 << endl;
    cout << "s2: " << s2 << endl;
    cout << "s3: " << s3 << endl;

    //Test setters
    r2.setLength(5);
    r3.setWidth(6);
    cout << "r2: " << r2 << endl;
    cout << "r3: " << r3 << endl;

    s1.setLength(8);
    s2.setWidth(4);
    s3.setSide(6);
    cout << "s1: " << s1 << endl;
    cout << "s2: " << s2 << endl;
    cout << "s3: " << s3 << endl;

    //Test input stream operators
    cin >> r1;
    cin >> s1;
    cout << "r1: " << r1 << endl;
```

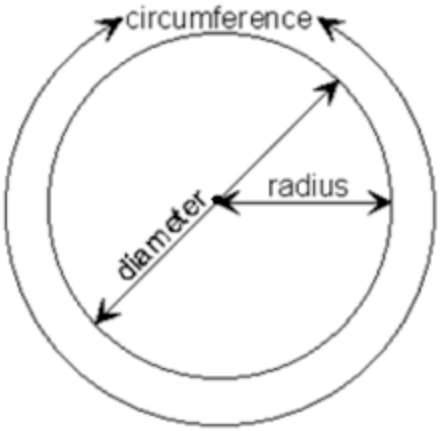
```
    cout << "s1: " << s1 << endl;

    //Test type casting
    MyRectangle r4 = s1;
    cout << "r4: " << r4 << endl;
    cout << "s2 casted to rectangle: " << static_cast<MyRectangle>(s2) << endl;

    system("Pause");
    return 0;
}
```

```
r1: Rectangle object: Length = 0, Width = 0, Area = 0, Perimeter = 0
r2: Rectangle object: Length = 2, Width = 3, Area = 6, Perimeter = 10
r3: Rectangle object: Length = 2, Width = 3, Area = 6, Perimeter = 10
s1: Square object: Side = 0, Area = 0, Perimeter = 0
s2: Square object: Side = 3, Area = 9, Perimeter = 12
s3: Square object: Side = 3, Area = 9, Perimeter = 12
r2: Rectangle object: Length = 5, Width = 3, Area = 15, Perimeter = 16
r3: Rectangle object: Length = 2, Width = 6, Area = 12, Perimeter = 16
s1: Square object: Side = 8, Area = 64, Perimeter = 32
s2: Square object: Side = 4, Area = 16, Perimeter = 16
s3: Square object: Side = 6, Area = 36, Perimeter = 24
Reading rectangle object...
Enter length 3.5
Enter width 2.7
Reading square object...
Enter side 1.6
r1: Rectangle object: Length = 3.5, Width = 2.7, Area = 9.45, Perimeter = 12.4
s1: Square object: Side = 1.6, Area = 2.56, Perimeter = 6.4
r4: Rectangle object: Length = 1.6, Width = 1.6, Area = 2.56, Perimeter = 6.4
s2 casted to rectangle: Rectangle object: Length = 4, Width = 4, Area = 16, Perimeter = 16
Press any key to continue . . .
```

6. Design a class named **Ellipse** that has two double member variables: **minorAxis**, **majorAxis**. Provide a default constructor (sets both member variables to 0.0), non-default constructor that takes two double arguments and sets the member variables to the parameter values, copy constructor, assignment operator, two getter member functions that return the values of the member variables, two setter member functions that set the values of the two member variables, and the functions getArea that returns $\boldsymbol{\pi ab}$ and getCircumference that returns $2\pi\sqrt{\frac{a^2+b^2}{2}}$ where **a** and **b** are the values of the minor and major axis. Provide also overloaded input/output streaming operators.

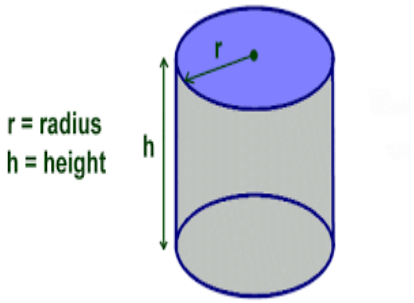


7. Design a class named **Circle** that inherits from the class **Ellipse** you created above. The Circle class should not add any new member variable; instead it must add additional behavior which is the values of the member variables **minorAxis** and **majorAxis** must always be the same. Provide default constructor that sets the values of both member variables to zero, non-default constructor that takes one double argument and sets both member variables to the value of the parameter, and a copy constructor. Provide a member function named **getRadius** that returns the radius of the circle and **setRadius** that takes one double argument and sets both member variables to the value of the parameter. **Override (re-define) the setter functions defined in the Ellipse class so that the Circle class will always have valid minor and major axis values.** Provide also the overloaded input/output streaming operators.



8. Consider a **Cylinder** object. A cylinder has a radius and a height. A **Cylinder** IS-A **Circle**. Create a **Cylinder** class by inheriting the **Circle** class we created above.

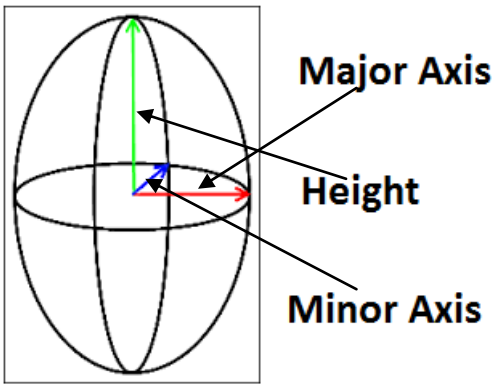
A cylinder class adds one more member variable named **height**. Provide a default constructor that sets both the radius and height of the cylinder to 0.0, non-default constructor that takes two double arguments and sets the radius and the height of the cylinder, a copy constructor and an assignment operator. Provide getter and setter member functions to get and set the height of the cylinder. **Override (redefine) the getArea member function so that it returns the surface area of the Cylinder**. The surface area of a cylinder is given by $\boldsymbol{\pi r^2 + 2\pi rh}$ where **r** is the radius of the cylinder and **h** is the height of the cylinder. Provide additional member function named getVolume that takes no argument and returns the volume of the cylinder. The volume of a cylinder is given by $\boldsymbol{\pi r^2 h}$ where **r** is the radius of the cylinder and **h** is its height. Provide also the overloaded input/output streaming operators.



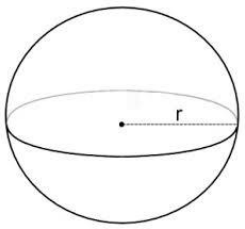
Write a test main program to test your inheritance lineage.

9. A **Disk** is a solid **Cylinder**. Disk object has the same member variables and member functions as a Cylinder object except that it has both its top and bottom sides closed and therefore its surface area is $\boldsymbol{2\pi r^2 + 2\pi rh}$. Create a class named **Disk** that inherits the Cylinder class and overrides the getArea member function. Provide default, non-default constructor, copy constructors, and assignment operator. Provide also the overloaded input/output streaming operators.

10. Consider an **Ellipsoid** object. An Ellipsoid has a minor axis, a major axis, and a height. An **Ellipsoid** IS-A **Ellipse**. Create an **Ellipsoid** class by inheriting the **Ellipse** class we created above. An Ellipsoid class adds one more member variable named **height**. Provide a default constructor that sets all the minor axis, major axis and height of the Ellipsoid to 0.0, non-default constructor that takes three double arguments and sets the minor axis, major axis and the height of the Ellipsoid, a copy constructor and assignment operator. Provide getter and setter member functions to get and set the height of the Ellipsoid. **Override (redefine) the getArea member function so that it returns the surface area of the Ellipsoid**. The surface area of an Ellipsoid is give by $4\pi\left(\frac{a^p b^p + a^p c^p + b^p c^p}{3}\right)^{1/p}$ where **a,b,c** are the minor axis, major axis and height of the Ellipsoid and **p** has the value **1.6075**. Provide additional member function named getVolume that takes no argument and returns the volume of the Ellipsoid. The volume of an Ellipsoid is given by $\frac{4}{3}\pi abh$ where **a** is the minor axis of the Ellipsoid, b is its major axis, and **h** is its height. Provide also the overloaded input/output streaming operators.



11. Consider a **Sphere** object. A sphere has only radius. A **Sphere** IS-A **Ellipsoid**. Create a **Sphere** class by inheriting the **Ellipsoid** class we created above. The **Sphere** class should not add any member variable; instead it should add the behavior that all its minor axis, major axis and height have equal values. Provide default constructor that sets the radius of the sphere to 0.0, non-default constructor that takes one double argument and sets the radius of the sphere, a copy constructor and an assignment operator. Provide getter and setter member functions to get and set the radius of the sphere. **Override (redefine) the setMinorAxis, setMajorAxis and setHeight member functions so that the sphere will always have equal values for its minor axis, major axis and height**. Provide also the overloaded input/output streaming operators. Write a test program to test your classes and inheritance lineage.



12. Consider the following class declaration that represents a hard disk of a computer. The class declaration contains a protected static member variable named `memAllocated` in order to track the amount of memory allocated in an application. Whenever memory is allocated on the heap, then it should be added to member `memAllocated` variable as shown in the default constructor provided below. Similarly, whenever memory is released, then it should be deducted from the `memAllocated` variable.

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

class HardDisk
{
private:
    char* buffer; //Dynamic array to represent hard disk
    int capacity; //The capacity of the hard disk
    int used; //The actual amount of data in the hard disk
```

```
protected:
    static int memAllocated;
public:
    HardDisk(); //Default constructor: capacity = 50, used = 0
    HardDisk(int capacityValue); //Non-default constructor: capacity = capacityValue, used = 0
    HardDisk(const HardDisk& hd); //Copy constructor. Deep copy.
    ~HardDisk(); //Destructor. Deletes the buffer and sets both capacity and used to 0.
    HardDisk& operator = (const HardDisk& hd); //Assignment operator. Deep copy.
    void write(char e); //writes e to the hard disk. That is it appends e to the buffer.
                        //If buffer is full then print message that the hard disk is full
                        //and then return without doing anything
    bool isFull() const; //Returns true if the hard disk is full (capacity == used), false otherwise.
    int getCapacity() const;
    int getUsed() const;
    void cleanup(); //Resets the used member variable to zero (used = 0)
    char& operator[](int index) const; //Asserts index is valid and then returns the element at index
    friend ostream& operator<<(ostream& out, const HardDisk& hd);
    static int getMemAllocated()
    {
           return memAllocated;
    }
};
int HardDisk::memAllocated = 0;

HardDisk::HardDisk()
{
    capacity = 50;
    buffer = new char[capacity];
    used = 0;
    memAllocated += capacity;
}
```

Now, use the following test main program and its output in order to help you implement the member and friend functions of the class. Your output must be identical to the output shown except for the actual characters stored in the Hard Disk which will be different because we are writing randomly generated characters.

```
int main()
{
    srand(time(0));
    ////////////////////////BASE CLASS TEST//////////////////////////////
    HardDisk hd1;
    cout << "Hard Disk 1 "<< hd1 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    HardDisk hd2(10);
    cout << "Hard Disk 2 "<< hd2 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    for (int i = 0; i < 20; i++)
           hd1.write(rand()%26+97);
    cout << "Hard Disk 1 "<< hd1 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    HardDisk hd3(hd1);
    cout << "Hard Disk 3 "<< hd3 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    hd1.~HardDisk();
    cout << "Hard Disk 1 "<< hd1 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    hd2 = hd3;
    cout << "Hard Disk 2 "<< hd2 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;
```

```
    hd1.~HardDisk();
    cout << "Hard Disk 1 "<< hd1 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    hd2.~HardDisk();
    cout << "Hard Disk 2 "<< hd2 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    hd3.~HardDisk();
    cout << "Hard Disk 3 "<< hd3 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    system("Pause");
    return 0;
}
```

A sample run output is given below

```
Hard Disk 1
        Capacity = 50
        Used = 0
        Data = []

Memory used so far = 50
Hard Disk 2
        Capacity = 10
        Used = 0
        Data = []

Memory used so far = 60
Hard Disk 1
        Capacity = 50
        Used = 20
        Data = [hwpbmmxksjuoxxstayoy]

Memory used so far = 60
Hard Disk 3
        Capacity = 50
        Used = 20
        Data = [hwpbmmxksjuoxxstayoy]

Memory used so far = 110
Hard Disk 1
        Capacity = 0
        Used = 0
        Data = []

Memory used so far = 60
Hard Disk 2
        Capacity = 50
        Used = 20
        Data = [hwpbmmxksjuoxxstayoy]

Memory used so far = 100
Hard Disk 1
        Capacity = 0
        Used = 0
        Data = []

Memory used so far = 100
Hard Disk 2
        Capacity = 0
        Used = 0
        Data = []

Memory used so far = 50
Hard Disk 3
        Capacity = 0
        Used = 0
        Data = []
```

```
Memory used so far = 0
Press any key to continue . . .
```

Next, consider a Hard Disk with a backup. A Hard Disk with a backup is made up of two equal sized Hard Disks where one will be used for day to day uses while the second one will be used for a backup. The following class is designed to represent Hard Disk with backup.

```
class HardDiskWithBackup : public HardDisk
{
private:
    char* backupBuffer;   //backup hard disk for the hard disk in the base class
    int backupUsed;       //the actual amount of data in the backup hard disk
public:
    HardDiskWithBackup(); //Default constructor: Default base class. Derived class (used = 0)
    HardDiskWithBackup(int capacityValue); //Non-default constructor: Non-default base class. Derived
class used = 0
    HardDiskWithBackup(const HardDiskWithBackup& hdb); //Copy constructor. Deep copy both base and
derived classes
    ~HardDiskWithBackup(); //Destructor. Destruct both base and derived classes.
    HardDiskWithBackup& operator = (const HardDiskWithBackup& hdb);   //Assignment operator. Deep copy
both base and derived classes
    void backup(); //Makes a backup copy of the base class hard disk
    void restore(); //Restores base class hard disk from the derived class backup hard disk
    friend ostream& operator<<(ostream& out, const HardDiskWithBackup& hdb);
};
```

Now, use the following test main program and its output in order to help you implement the member and friend functions of the class. Your output must be identical to the output shown except for the actual characters stored in the Hard Disk which will be different because we are writing randomly generated characters.

```
int main()
{
    srand(time(0));
    HardDiskWithBackup hdwb1;
    cout << "Hard Disk with backup 1 " << hdwb1 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    hdwb1.~HardDiskWithBackup();
    cout << "Hard Disk with backup 1 " << hdwb1 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    HardDiskWithBackup hdwb2(40);
    for (int i = 0; i < 10; i++)
           hdwb2.write(rand()%26+65);
    cout << "Hard Disk with backup 2 " << hdwb2 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    HardDiskWithBackup hdwb3(hdwb2);
    cout << "Hard Disk with backup 3 " << hdwb3 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    hdwb2.~HardDiskWithBackup();
    cout << "Hard Disk with backup 2 " << hdwb2 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    hdwb1 = hdwb3;
    cout << "Hard Disk with backup 1 " << hdwb1 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;
```

```
    hdwb1.backup();
    cout << "Hard Disk with backup 1 " << hdwb1 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    for (int i = 0; i < 5; i++)
           hdwb1.write(rand()%26+65);
    cout << "Hard Disk with backup 1 " << hdwb1 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    hdwb2 = hdwb1;
    cout << "Hard Disk with backup 2 " << hdwb2 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    hdwb2.cleanup();
    cout << "Hard Disk with backup 2 " << hdwb2 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    hdwb2.restore();
    cout << "Hard Disk with backup 2 " << hdwb2 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    hdwb1.~HardDiskWithBackup();
    cout << "Hard Disk with backup 1 " << hdwb1 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    hdwb2.~HardDiskWithBackup();
    cout << "Hard Disk with backup 2 " << hdwb2 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    hdwb3.~HardDiskWithBackup();
    cout << "Hard Disk with backup 3 " << hdwb3 << endl;
    cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl;

    system("Pause");
    return 0;
}
```

A sample run output is given below

```
Hard Disk with backup 1
        Capacity = 50
        Used = 0
        Data = []
        Backup Capacity = 50
        Backup Used = 0
        Backup Data = []

Memory used so far = 100
Hard Disk with backup 1
        Capacity = 0
        Used = 0
        Data = []
        Backup Capacity = 0
        Backup Used = 0
        Backup Data = []

Memory used so far = 0
Hard Disk with backup 2
        Capacity = 40
        Used = 10
        Data = [GJPJWTMBWT]
        Backup Capacity = 40
        Backup Used = 0
        Backup Data = []

Memory used so far = 80
Hard Disk with backup 3
        Capacity = 40
```

```
        Used = 10
        Data = [GJPJWTMBWT]
        Backup Capacity = 40
        Backup Used = 0
        Backup Data = []

Memory used so far = 160
Hard Disk with backup 2
        Capacity = 0
        Used = 0
        Data = []
        Backup Capacity = 0
        Backup Used = 0
        Backup Data = []

Memory used so far = 80
Hard Disk with backup 1
        Capacity = 40
        Used = 10
        Data = [GJPJWTMBWT]
        Backup Capacity = 40
        Backup Used = 0
        Backup Data = []

Memory used so far = 160
Hard Disk with backup 1
        Capacity = 40
        Used = 10
        Data = [GJPJWTMBWT]
        Backup Capacity = 40
        Backup Used = 10
        Backup Data = [GJPJWTMBWT]

Memory used so far = 160
Hard Disk with backup 1
        Capacity = 40
        Used = 15
        Data = [GJPJWTMBWTQFCUZ]
        Backup Capacity = 40
        Backup Used = 10
        Backup Data = [GJPJWTMBWT]

Memory used so far = 160
Hard Disk with backup 2
        Capacity = 40
        Used = 15
        Data = [GJPJWTMBWTQFCUZ]
        Backup Capacity = 40
        Backup Used = 10
        Backup Data = [GJPJWTMBWT]

Memory used so far = 240
Hard Disk with backup 2
        Capacity = 40
        Used = 0
        Data = []
        Backup Capacity = 40
        Backup Used = 10
        Backup Data = [GJPJWTMBWT]

Memory used so far = 240
Hard Disk with backup 2
        Capacity = 40
        Used = 10
        Data = [GJPJWTMBWT]
        Backup Capacity = 40
        Backup Used = 10
        Backup Data = [GJPJWTMBWT]

Memory used so far = 240
Hard Disk with backup 1
        Capacity = 0
        Used = 0
```

```
        Data = []
        Backup Capacity = 0
        Backup Used = 0
        Backup Data = []

Memory used so far = 160
Hard Disk with backup 2
        Capacity = 0
        Used = 0
        Data = []
        Backup Capacity = 0
        Backup Used = 0
        Backup Data = []

Memory used so far = 80
Hard Disk with backup 3
        Capacity = 0
        Used = 0
        Data = []
        Backup Capacity = 0
        Backup Used = 0
        Backup Data = []

Memory used so far = 0
Press any key to continue . . .
```