# CMPT 135 Week 10 Lab Work

**1.** Investigate the following program and determine its output?

```
#include <iostream>
using namespace std;
class A
{
public:
    virtual void display() { cout << "A display" << endl; }
};
class B : public A
{
public:
    virtual void display() { cout << "B display" << endl; }
};
class C : public B
{
public:
    C () { cout << "C constructor" << endl; }
    virtual void display() {cout << "C display" << endl;}
};
void foo(A* p)
{
    p->display();
}
int main() // memory cleanup (deleting) is omitted in the following code
{
    A* ptA = new A;
    B* ptB = new B;
    C* ptC = new C;
    ptA->display();
    ptA = ptB;
    ptA->display();
    ptA = ptC;
    foo(ptA);
    ptA = new C;
    ptB = dynamic_cast<B*> (ptA) ; // Can you simply do ptB = ptA; ?
    ptB->display();
    // Any problem with the following code?
    //ptA = new A;
    //ptB = dynamic_cast<B*> (ptA);
    //ptB->display();
    system("Pause");
    return 0;
}
```

**2.** What happens if we remove the virtual modifier from all the member functions?
**Answer:-** The statement pointer->display(); would always execute the display function inside the class of the type of the pointer irrespective of to what kind of object the pointer is pointing to.

Moreover the dynamic_cast will give syntax error because dynamic _cast only works with classes that have at least one virtual member function.

**3.** Given the Shape, Rectangle, Triangle and Circle classes as discussed in the lecture, analyze the following program and determine its output. This example shows we can use Shape pointers and references in parameter passing but not Shape objects because the Shape is an abstract class.

```
typedef Shape* ShapePtr;
```

```
void foo(Shape* s1, Shape& s2)
{
    cout << "Entering the foo function" << endl;
    cout << *s1 << endl;
    cout << s2 << endl;
    cout << "Exiting the foo function" << endl;
}
int main()
{
    //Create an array of Shape pointers
    const int SIZE = 5;
    ShapePtr *A = new ShapePtr[SIZE];

    //Populate with shapes among rectangles, triangles and circles
    A[0] = new Rectangle(4, 5, "Red");
    A[1] = new Triangle(2, 4, "Yellow");
    A[2] = new Square(3, "Blue");
    A[3] = new Circle(2, "Green");
    A[4] = new Rectangle();

    //Print the elements of the array
    for (int i = 0; i < SIZE; i++)
       cout << *(A[i]) << endl;

    //Call foo function with A[0] and A[1]
    foo(A[0], *(A[1]));

    //Peform memory cleanup
    for (int i = 0; i < SIZE; i++)
       delete A[i];
    delete[] A;

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

**4.** Write a C++ function named **`insertGrouped`** that takes a vector of Shape pointers (whose elements are already grouped according to the type of the underlying objects) and a Shape pointer as arguments and that inserts the pointer in the vector such that the underlying objects will also be grouped afterwards.

Your algorithm must be linear. Below is a test main program with supporting functions that you can use to test your function. Please make sure to have typedef Shape* ShapePtr just below the Shape class code.

```
typedef Shape* ShapePtr;

ShapePtr getRandomShapePtr()
{
    string color[] = {"Red", "Blue", "Yellow", "Purple", "Green", "Cyan"};
    switch(rand() % 4)
    {
        case 0:
            cout << "Constructing a Rectangle object" << endl;
            return new Rectangle(rand()%11+5, rand()%11+5, color[rand()%6]);
        case 1:
            cout << "Constructing a Square object" << endl;
            return new Square(rand()%11+5, color[rand()%6]);
        case 2:
            cout << "Constructing a Triangle object" << endl;
            return new Triangle(rand()%11+5, rand()%11+5, color[rand()%6]);
        default:
            cout << "Constructing a Circle object" << endl;
            return new Circle(rand()%11+5, color[rand()%6]);
    }
}
```

```
int main()
{
    vector<ShapePtr> v;
    int size;
    cout << "How many objects would you like to insert? ";
    cin >> size;
    for (int i = 0; i < size; i++)
    {
       ShapePtr p = getRandomShapePtr();
       insertGrouped(v, p);
       //insertIncreasing(v, p);
       //insertGroupedSorted(v, p);
    }

    //Print the objects
    for (int i = 0; i < size; i++)
        cout << *(v[i]) << endl;

    //Destruct the objects
    for (int i = 0; i < size; i++)
        delete v[i];

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

5. Write a C++ function named **insertIncreasing** that takes a vector of Shape pointers (whose elements are already sorted in increasing order according to the area of the underlying objects) and a Shape pointer as arguments and that inserts the pointer in the vector such that the underlying objects will also be sorted in increasing order according to their areas afterwards. Please note that grouping together of the same type objects is not needed here.

   Your algorithm must be linear. In order to test your function, use the same main program as in Question 4 above with the **insertGrouped** function call in the main program modified to call the **insertIncreasing** function call.

6. Write a C++ function named **insertGroupedSorted** that takes a vector of Shape pointers (whose elements are already grouped according to the type of the underlying objects and such that they are sorted in increasing order of their areas) and a Shape pointer as arguments and that inserts the pointer in the vector such that the underlying objects will also be grouped and sorted afterwards.

   Your algorithm must be linear. In order to test your function, use the same main program as in Question 4 above with the **insertGrouped** function call in the main program modified to call the **insertGroupedSorted** function call.

7. Write a C++ function named **groupElements** that takes a vector of Shape pointers and groups the elements of the vector so that elements of same type will be grouped together. Can you do this in a linear algorithm?

   Hint:- Use your **insertGrouped** function together with the overloaded assignment operator of the vector class. In order to test your function use the **getRandomShapePtr** function to push_back several different kinds of objects onto a vector and then call the function **groupElements** to group the elements and then print your elements to see if they are grouped.

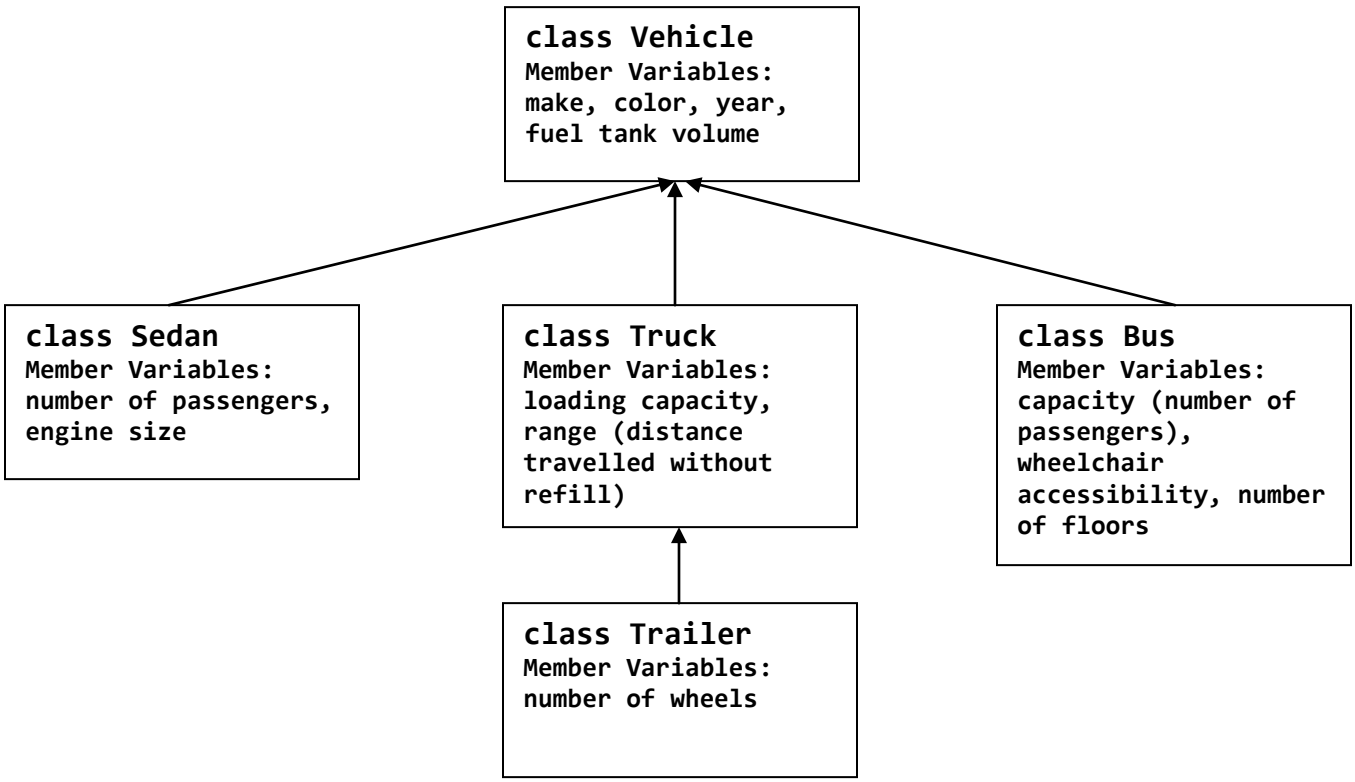
8. Define a function named **sortVector** that takes a vector of Shape pointers and sorts the elements of the vector according to the area of the underlying objects.

Hint:- Use your **insertIncreasing** function together with the overloaded assignment operator of the vector class. In order to test your function use the **getRandomShapePtr** function to push_back several different kinds of objects onto a vector and then call the function **sortVector** to group the elements and then print your elements to see if they are sorted.

**9.** Write a C++ function named **groupSortElements** that takes a vector of Shape pointers and groups the elements of the vector so that elements of same type will be grouped together and within the same group will be sorted in increasing order of their areas. Can you do this in a linear algorithm?

Hint:- Use your **insertGroupedSorted** function together with the overloaded assignment operator of the vector class. In order to test your function use the **getRandomShapePtr** function to push_back several different kinds of objects onto a vector and then call the function **insertGroupedSorted** to group and sort the elements and then print your elements to see if they are grouped and sorted.

**10.** Consider the following class hierarchies:




Implement the classes taking into account the given class hierarchies; that is class Vehicle is a base class for all the other classes, the classes Sedan, Truck and Bus are derived from class Vehicle, and the class Trailer is derived from class Truck. You may add as many member variables and member, friend or non-member functions as you see fit. Finally write a test program to test the design and implementation of your classes. This is an open question and you are free to change the class hierarchies design and get a better design. The main idea is for you to play with some class designs that exhibit polymorphic behavior.

**11.** 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.
    virtual ~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
    virtual void printInfo(ostream&) const; //The << operator must call the prinInfo member function
    friend ostream& operator << (ostream& cout, const HardDisk& hd);
    static int getMemAllocated()
    {
           return memAllocated;
    }
};
int HardDisk::memAllocated = 0;

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

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

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

       cout << "Populating Hard Disk 1" << endl;
       for (int i = 0; i < 20; i++)
              hd1.write(rand()%26+97);
       cout << "Now, Hard Disk 1 " << hd1 << endl;
```

```
        cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl << endl;

        HardDisk hd3(hd1);
        cout << "Hard Disk 3 copied from Hard Disk 1: " << hd3 << endl;
        cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl << endl;

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

        cout << "Assigning Hard Disk 2 the value of Hard Disk 3" << endl;
        hd2 = hd3;
        cout << "Now, Hard Disk 2 " << hd2 << endl;
        cout << "Memory used so far = " << HardDisk::getMemAllocated() << endl << endl;

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

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

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

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

## Sample Run Output

```
Default Hard Disk 1: , CAPACITY = 50, USED = 0
Memory used so far = 50

Non-Default Hard Disk 2: , CAPACITY = 10, USED = 0
Memory used so far = 60

Populating Hard Disk 1
Now, Hard Disk 1 vhpnzmdnvpwmoacawyme, CAPACITY = 50, USED = 20
Memory used so far = 60

Hard Disk 3 copied from Hard Disk 1: vhpnzmdnvpwmoacawyme, CAPACITY = 50, USED = 20
Memory used so far = 110

Destructing Hard Disk 1
Now, Hard Disk 1 , CAPACITY = 0, USED = 0
Memory used so far = 60

Assigning Hard Disk 2 the value of Hard Disk 3
Now, Hard Disk 2 vhpnzmdnvpwmoacawyme, CAPACITY = 50, USED = 20
Memory used so far = 100

Destructing Hard Disk 1
Now, Hard Disk 1 , CAPACITY = 0, USED = 0
Memory used so far = 100

Destructing Hard Disk 2
Now, Hard Disk 2 , CAPACITY = 0, USED = 0
Memory used so far = 50
```

```
Destructing Hard Disk 3
Now, Hard Disk 3 , CAPACITY = 0, USED = 0
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
	void printInfo(ostream&) const; //The << operator must use the prinInfo member function
};
```

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;
}
```

## Sample Run Output

```
Hard Disk with backup 1 , CAPACITY = 50, USED = 0, backup buffer = , BACKUP USED = 0
Memory used so far = 100
Hard Disk with backup 1 , CAPACITY = 0, USED = 0
Memory used so far = 0
Hard Disk with backup 2 NUXFPNDMMI, CAPACITY = 40, USED = 10, backup buffer = , BACKUP USED = 0
Memory used so far = 80
Hard Disk with backup 3 NUXFPNDMMI, CAPACITY = 40, USED = 10, backup buffer = , BACKUP USED = 0
Memory used so far = 160
Hard Disk with backup 2 , CAPACITY = 0, USED = 0
Memory used so far = 80
Hard Disk with backup 1 NUXFPNDMMI, CAPACITY = 40, USED = 10
Memory used so far = 160
Hard Disk with backup 1 NUXFPNDMMI, CAPACITY = 40, USED = 10
Memory used so far = 160
Hard Disk with backup 1 NUXFPNDMMILBYRX, CAPACITY = 40, USED = 15
Memory used so far = 160
Hard Disk with backup 2 NUXFPNDMMILBYRX, CAPACITY = 40, USED = 15
Memory used so far = 240
Hard Disk with backup 2 , CAPACITY = 40, USED = 0
Memory used so far = 240
Hard Disk with backup 2 NUXFPNDMMI, CAPACITY = 40, USED = 10
Memory used so far = 240
Hard Disk with backup 1 , CAPACITY = 0, USED = 0
Memory used so far = 160
Hard Disk with backup 2 , CAPACITY = 0, USED = 0
Memory used so far = 80
Hard Disk with backup 3 , CAPACITY = 0, USED = 0
Memory used so far = 0
```

```
Press any key to continue . . .
```