Data Structures (Spring 2020)
Lab #4: C++memory management and inheritance

Task-1: Memory management

- Download and execute the C++ program called **lab4.1.cpp** from NYU Classes. You will observe after running the program, that modifying salary values of one object is also modifying salary values of the other objects. This problem is due to the **shallow copying** performed in the copy constructor.

- Modify the copy constructor such that instead of shallow copy, it should perform **deep copy**.

- Add overloaded assignment operator (=) in the Employee class. The method should also perform **deep copy** while copying values from one object to another object. **Note:** While making deep copy make sure that you delete/free the existing memory and allocate enough memory to hold the values from the new object. Special attention should be paid to avoid self-assignment e.g. ob1 = obj1

Task-2 inheritance vs composition

- Define a simple class **Point** with two attributes x and y representing the geometric coordinates of the point. Include a 2-argument constructor with default values (e.g. 0), and a function **display()** to printout the coordinates of the point. Make sure to declare the attributes x and y in protected mode so they become available to the subclasses of Point.

- In the **main(),** create few instances of **Point** and test all the functions of the **Point** class.

- Define a class **ColorPoint** as a *subclass* of the class **Point**. The class **ColorPoint** has one single attribute *color*. Define a constructor, a copy constructor and an assignment operator for this class, and override the function **display()** so that it can display all the information of **ColorPoint** objects.

- In the **main(),** create few instances of **ColorPoint** and test all the functions of **ColorPoint** (including the *assignment operator (=)* and the copy constructor) as well as the functions inherited from **Point** class.

- Define a class **Circle** with 2 attributes: center (of type **Point**) and radius. Define a constructor, a copy constructor, an assignment operator for this class, and a function **display()** so that it can display all the information of **Circle** objects.

- In the **main(),** create few instances of **Circle** and test all the functions of **Circle**.

- Note: to define the class **ColorPoint** we used the *inheritance* approach, but we used the *composition* approach to define the class **Circle**. Try to analyze the two approaches and find when to use which approach.

Task-3 inheritance and polymorphism

- Define a class **Vehicle** with two attributes *make* and model of type *string*. Define a constructor and a method called display() to printout the values of the attributes.
- In the main(), create instances of type **Vehicle** and test the functions of this class.

- Define another class **Car** as a subclass of **Vehicle** with one more attribute *passenger_capacity* of type int. Provide a constructor for this class.
- In the **main(),** create instances of type **Car** and test the functions of this class. Verify the output of the function display(), you will find out that this function doesn't print the passenger_capacity. Override the function for the subclass so that it prints inherited attributes as well as the passenger load. Test your program.
- Define another class **Truck** as a subclass of vehicle with one more attribute *payload* of type int. Provide a constructor, and a **display()** function for this class.
- In the **main(),** create instances of type *Truck* and test the functions of this class.

- In the **main(),** declare a dynamic array of pointers to *Vehicle*. Prompt the user to enter the size of the array as well type of the different objects the array is pointing to. (e.g. 2 pointers to Vehicle, 2 pointers to Car, and 2 pointers to Truck). Traverse the array and use the pointers to call the display function. Check the output of your program, you will notice that even if the array contains pointers to subclasses, the program keeps calling the function **Vehicle::display(),** this is called static binding.
- Add the keyword "*virtual*" to the header of the function **display()** in the class Vehicle, test you program again and verify that the functions called match with the addresses contained in the array (dynamic binding).
