

Hands-on:-

Part-1: Classes and Objects

- Box
- Account
- Point
- Color
- Image
- IPAddress
- MyTime
- MyDate
- MyStack
- MyString

Part-2 : Operator Overloading

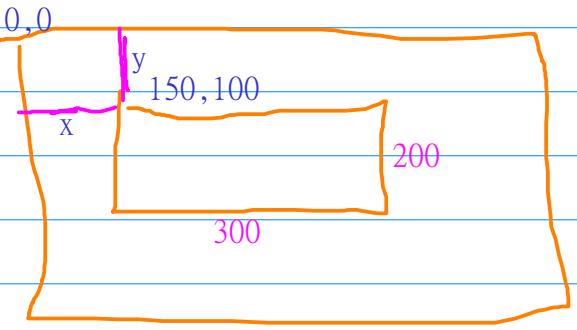
- Complex
- MyTime
- Fraction class
- Currency / Weight / Distance
- MyDate
- Matrix
- MyString

Part-3 : Inheritance & Virtual Functions

- Geometric Shapes
- Banking Accounts
- Mobile Billing Customers

Guidelines/Expectations

- * Class Diagram
 - * Multifile/modular coding
 - * member wise initializer list for simple members
 - * copy ctor, dtor only if needed (Non Trivial)
 - * const suffix for immutable operations
 - * Test cases using googletest
 - * Prefer use `std:: prefix (std::cout, std::cin)`
 avoid "using namespace std;"
 - * Getter/Accessor functions
 - * Avoid setter functions as much possible
- * meaningful names
* naming conventions
* code style
 (indentation, formatting)
-
- * static analysis
* heap analysis
 /memory leak detection

Point	IPAddress - ipval : uint32_t (or) - a : uint8_t - b : uint8_t	Color c2(0xFF00FF) Color c3("7F2352"); 192.168.72.25
+ quadrant() + distanceFromOrigin() + getter functions	- c : uint8_t - d : uint8_t	IPAddress ip1(192,168,72,25) IPAddress ip2("192.168.72.25") IPAddress ip3(0xC0A84819) IPAddress ip4; //127.0.0.1
Color	+ ipval() : uint32_t + ipstr() : string + display() + isLoopback() + class() //A,B,C,D	
- m_red - m_green - m_blue + Color(r:Int,g:Int,b:Int) + Color(hexcode:Int) + Color(hexstr:String) + hexval() + invert() + display() + getter functions		
Image	- m_x : Int // x pos - m_y : Int // y pos - m_width : Int - m_height : Int + rotate() + scale() // zoom() + shift() // move() + display()	

```
TEST(BoxTest, BoxGetter())
{
    Box b1(10,12,5);
    EXPECT_EQ(10,b1.length())
    EXPECT_EQ(12,b1.breadth())
    EXPECT_EQ(5,b1.height())
}

TEST(BoxTest, BoxVolume)
{
    Box b1(10,12,5);
    EXPECT_EQ(600, b1.findVolume())
}

TEST(BoxTest, BoxSurfaceArea())
{
    Box b1(10,12,5);
    EXPECT_EQ(460,b1.findSurfaceArea())
}

-----
IPAddress ip2("192.0.49.35");
EXPECT_TRUE(ip2.isLoopback())                                EXPECT_TRUE

EXPECT_FALSE(ip2.isLoopback())                               EXPECT_FALSE

IPAddress ip2("192.0.49.35");
EXPECT_STREQ(ip2.hexval(), "C0003123")                     EXPECT_STREQ

EXPECT_EQ(ip2.hexstr(), "C0003123")                         EXPECT_EQ

EXPECT_NE(ip2.hexstr(), "C0003124")                         EXPECT_NE

EXPECT_LT(ip2.ip(), "127.0.0.1")                            EXPECT_LT

EXPECT_GE(ip2.ip(), "127.0.0.0")                            EXPECT_GE

EXPECT_GT(ip2.ip(), "127.0.0.2")                            EXPECT_GT
```

```

class ICustomer           //abstract, like interface in Java
{
    public:
        void makeCall(int)=0; //duration in minutes
        void credit(double)=0; //recharge or bill amount
    };
    class Customer : public ICustomer           //also abstract
    {
        std::string m_id;
        std::string m_name;
        double m_balance;
        public:
            Customer(string id, string name, double balance);
            double balance() { return m_balance; }
            //TODO : display
    };
    class PrepaidCustomer : public Customer
    {
        //if any other data members needed
        public:
            Customer(string id, string name, double balance):Customer(id,name,balance) { }
            void credit(double);
            void makeCall(int);
    };
    class PostpaidCustomer : public Customer
    {
        //similar to Prepaid
    };
}

ICustomer *pcust;
if(cond)
    pcust = new PrepaidCustomer(/*...*/)
else
    pcust = new PostpaidCustomer(/*...*/)

pcust->makeCall(10);
pcust->credit(500);

customer.h          customer.h          #ifndef __CUSTOMER_H
customer.cpp         precustomer.h      #define __CUSTOMER_H
main.cpp / test.cpp  postcustomer.h
                      customer.cpp
                      precustomer.cpp
                      postcustomer.cpp
                      main.cpp / test.cpp
#endif

```

NovaProbs:-

Q6:-

```
@startuml
class ElectricVehicle
{
    - m_vehicleId : int
    - m_make : string
    - m_model : string
    - m_batteryCapacity : double
    - m_charging : double
    + ElectricVehicle(id : int, make : string, model : string,
                      capacity : double, charging : double)
    + charge(duration : int) // hours
    + drive(distance : int) // miles or km
    + displayDetails()
}
@enduml
```

RateOfCharging - increase in capacity per hour , e.g. 10kwh

RateOfDischarge - decrease in capacity per mile

```
const static double RATE_OF_CHARGING = 10; //10kwn
```

```

@startuml
class Vehicle
{
    - m_vehicleId : int
    - m_make : string
    - m_model : string
    - m_mileage : int
    - m_price : double
    + virtual displayDetails() : void
    + mileage() : int
    + price() : double
}
class Car
{
    - engineType : string
    + Car(/*args*/)
    + displayDetails() : void
}
class Truck
{
    - payloadCapacity : double
    + Truck(/*args*/)
    + displayDetails() : void
}
Car *-up-> Vehicle
Truck *-up-> Vehicle

void displayVehicleDetails(Vehicle* pVehicle)
{
}

void calculateTotalMileage()
{

}

void sortVehiclesByPrice()
{

}

void searchVehiclesByvehicleId()
{
}

```

@enduml

```

const int NUM_VEHICLES = 10
Vehicle** vehicles = new Vehicle*[NUM_VEHICLES];

int count=0;
car1 = new Car( /*...*/ );
vehicles[count++] = car1;

car2 = new Car( /*...*/ );
vehicles[count++] = car1;

truck1 = new Truck( /*...*/ );
vehicles[count++] = car1;

//similarly add truck2, truck3 to vehicles[3], vehicles[4]

//delete all dynamic memory
for(i=0;i<count;i++)
    delete vehicles[i];
delete[] vehicles;

```

```
void displayAllVehicleDetails(Vehicle** vehicles, int nVehicles)
{
    for(int i=0;i<nVehicles;i++)
        vehicles[i]->displayDetails();
}
void displayVehicleDetails(Vehicle** vehicles, int nVehicles)
{
    int totalMileage = 0;
    for(int i=0;i<nVehicles;i++)
        totalMileage += vehicles[i]->mileage();
}
void sortVehiclesByPrice(Vehicle** vehicles, int nVehicles)
{
    //TODO: sorting logic
}
Vehicle* searchVehicleById(Vehicle** vehicles, int nVehicles,int keyId)
{
    //return vehicles[i] if found, else nullptr
}
```

```
class MyIterator
{
    int* m_ptr;
public:
    MyIterator(int *ptr) : m_ptr(ptr) { }
    int& operator*() {
        return *m_ptr;
    }
    int* getptr() const { return m_ptr; }
};

int main()
{
    int xval = 100;
    MyIterator iter(&xval);

    int *tptr = iter.getptr();
    int res = *tptr;

    res = *iter; //iter.operator*()

    return 0;
}
```

```

class MyArray
{
    int *m_arr; //base addr
    int m_len; //capacity
public:
    MyArray(int len):m_len(len) {
        m_arr = new int[m_len];
    }
    MyArray():m_len(10) {
        m_arr = new int[m_len];
    }
    ~MyArray() {
        if(m_arr!= nullptr)
            delete[] m_arr;
    }
    void fillArrayWithRandomValues()
    {
        std::srand(time(0));
        for(i=0;i<m_len;i++)
            m_arr[i] = std::rand() % 100;
    }
    void fillArrayWithValue(int val)
    {
        for(i=0;i<m_len;i++)
            m_arr[i] = val;
    }
    //displayValues()

    int length() const { return m_len; }
    int valueAt(int index) {
        return m_arr[index];
    }
    int operator[](int index)
    {
        return m_arr[index];
    }
}

```

```

int main()
{
    MyArray a1(15);
    MyArray a2;
    a1.fillArrayWithRandomValues();
    a1.fillArrayWithValue(20);
    int val = a1.valueAt(4);
    val = a1[4]; //a1.operator[](4)
}

```