1) **[10 Marks]** Write in the space on the right the output of the following program.

```
#include <iostream>
using namespace std;
enum PassengerType { Child, Youth, Adult, Senior, None };

struct Passenger {

        std::string  name;
        PassengerType type = PassengerType::None;
public:

        bool isEmpty() const { return type == PassengerType::None; }
        Passenger() {  }
        Passenger(const std::string& n, PassengerType t) {
                name = n;
                type = t;
        }
};
std::ostream& operator<<(std::ostream& os, const Passenger& client)
{
        os << "\n Passenger Name = " <<  client.name ;
        os << "\n Passenger Type = " << client.type;
        os << "\n";

        return os;
}

template <typename T>
void print(T& val)
{
        std::cout << "l-value: " << val << std::endl;
}

template <typename T>
void print(T&& val)
{
        std::cout << "r-value: " << val << std::endl;
}



int main()
{
        static float xyz = 55;

        Passenger henry("Henry", Senior);

        Passenger mary("Mary", Child);


        double a{ 1000 };

        print(a);

        print(a + double(60));

        print(xyz);

        print(std::move(henry));

        print(mary);

        return 0;
}
```

Output:

2) **[10 Marks]** Upgrade the following code to make it a template for classes of any specified type and any specified size

```
class QueueItem {

     std::string complain_type;
     std::string customer_id;

public:
     QueueItem() { }
     QueueItem(const std::string& aa, const std::string& bb)
     {
        complain_type = aa;
        customer_id = bb;
     }

     const std::string& key() const { return complain_type; }
     const std::string& value() const { return customer_id; }
     void display(std::ostream& os) const
     {
           os << complain_type << ": "<< customer_id << std::endl;
     }
}
```

3) **[5 Marks]** A class can be derived directly from a templated family of classes. All the usual rules of inheritance and polymorphism apply. Upgrade the following header file to a templated family of classes.

```
#include <string>
class GeometricObject
{
public:
   GeometricObject();
   GeometricObject(const string & color, bool filled);
   string getColor() const;
   void setColor(const std::string & color);
   bool isFilled() const;
   void setFilled(bool filled);
   string toString() const;

private:
    std::string color;
    bool filled;
};

class Rectangle : public GeometricObject
{
public:
   Rectangle();
   Rectangle(double width, double height);
   Rectangle(double width, double height,
         const std::string & color, bool filled);
   double getWidth() const;
   void   setWidth(double);
   double getHeight() const;
   void   setHeight(double);
   double getArea() const;
   double getPerimeter() const;
   std::string toString() const;

private:
   double width;
   double height;
};
```