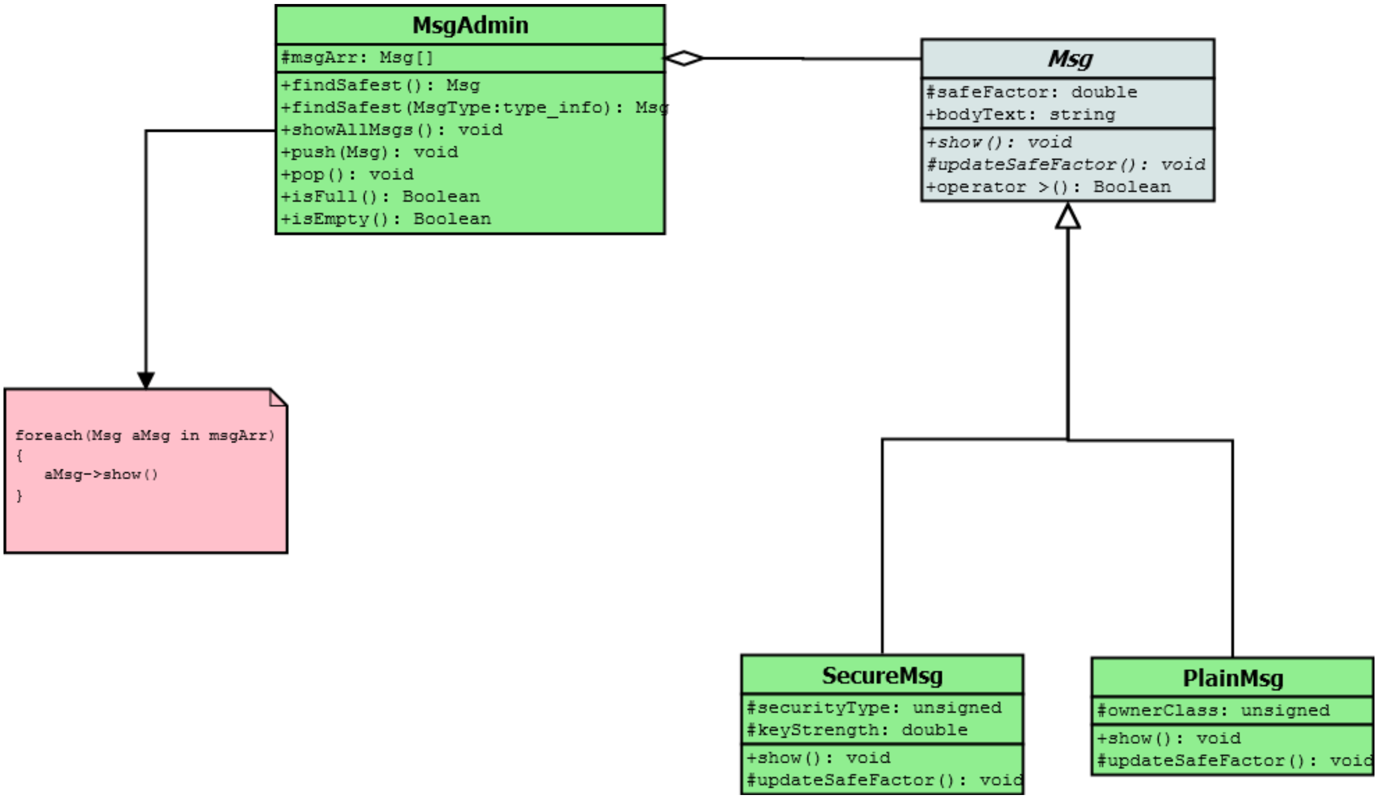
C++ Summary Assignment

Part A (60 %) Design task

Overview
A text message which is delivered in the net can have various degrees of risk, potentially carrying malicious contents which can harm a target machine. In order to evaluate the safety level of a message, it is examined upon arrival in the target machine, and is attached with various parameters, used for the assessment of its final security degree.

The following diagram is a partial UML for a design intended to manage a collection of messages:



```
foreach(Msg aMsg in msgArr)
{
    aMsg->show()
}
```

Class *Msg*

Msg defines the base structure of a simple text massage. It has the following main properties:

- safeFactor – indicates the final security degree of the message. This is later calculated differently by each concrete sub class: SecureMsg/ PlainMsg.
- updateSafeFactor() – a virtual method to compute the final safe factor: This varies between a secure or an unsecured – plain message. See ahead.

- Comparison operator> which tells whether a Msg is safer than the other by comparing their respective safe Factors.
- show() function to display the Msg's fields.
- bodyText - a buffer containing the "raw" text

Class PlainMsg

PlainMsg defines a structure of a simple un-encrypted plain message. The extent to which such message can have any malicious intent can be assessed by the class category to which its originator belongs.

- This is indicated by the ownerClass parameter , and can span incrementally in range [A,B,C,D,E] where E is most trustable(safest).

- The safe factor of a plain message is given by:
  safeFactor = 70%(ownerClass) +30%(1 /(message length))*

Class SecureMsg

SecureMsg defines a structure of a secure message. It is encrypted using a mechanism type (securityType) combined with an encryption key with a strength level (keyStrength).

- The securityType incrementally spans in range (*PWD, AES, PKI, SSL*), where *PWD* is the least powerful mechanism.

- Similarily, keyStrength incrementally spans in range *(LOW, NORM, MID, HIGH.*

- The safe factor of a secure message is given by:

safeFactor = 40%(securityType) + 30%(KeyStrength)+ 30%(1 /(message length))*

Class `MsgAdmin`

`MsgAdmin` is a module to manage a collection of messages. Typically it will have the following features:

- `msgArr` – a collection of `Msg`s. the number of items is determined by the user and should not exceed MAX_MSG_NUM
- `findSafest()` –finds the safest message among the collection
- `findSafest (msgType)` : given a concrete type as parameter, this method will find the safest message of that type in the collection
- `show()` method to display all the messages properties in the collection
- a group of `push/pop/empty/full` methods to manage the insertion/extraction of messages in/out of the collection

Testing things in *`main()`* :

- define the following samples vector and assess results:

	Message	length	ownerClass	securityType	keyStrength	safeFactor
1	secure	6		*PWD*	*HIGH*	0.95
2	Plain	11	B			*0.727273*
3	secure	28		PKI	MID	*1.41071*
4	Plain	22	D			*2.11364*
5	Secure	6		*PWD*	*HIGH*	*0.95*
6	Secure	11		AES	NORM	*0.727273*
7	Secure	28		PKI	MID	*1.41071*
8	Plain	22	D			*2.11364*
9	Secure	6		*PWD*	*HIGH*	0.95
10	Plain	11	B			*0.727273*

- A possible output could be:

```
-------------Administrating the following msgs: ---------

--------------------------------

SecureMsg:
safeFactor: 0.95
Msg Len: 6
securityType: PWD  keyStrength: HIGH

--------------------------------

PlainMsg:
safeFactor: 0.727273
Msg Len: 11
ownerClass: CLS_B
--------------------------------
```

```
--------------------------------

SecureMsg:
safeFactor: 1.41071
Msg Len: 28
securityType: PKI  keyStrength: MID
--------------------------------

--------------------------------

PlainMsg:
safeFactor: 2.11364
Msg Len: 22
ownerClass: CLS_D
--------------------------------

--------------------------------

SecureMsg:
safeFactor: 0.95
Msg Len: 6
securityType: PWD  keyStrength: HIGH
--------------------------------

--------------------------------

SecureMsg:
safeFactor: 0.727273
Msg Len: 11
securityType: AES  keyStrength: NORM
--------------------------------

--------------------------------

SecureMsg:
safeFactor: 1.41071
Msg Len: 28
securityType: PKI  keyStrength: MID
--------------------------------

--------------------------------

PlainMsg:
safeFactor: 2.11364
Msg Len: 22
ownerClass: CLS_D
--------------------------------

--------------------------------

SecureMsg:
safeFactor: 0.95
Msg Len: 6
```

```
securityType: PWD  keyStrength: HIGH
--------------------------------

--------------------------------

PlainMsg:
safeFactor: 0.727273
Msg Len: 11
ownerClass: CLS_B
--------------------------------


----------total: 10 Msgs ------------ --------------

**************Safest of all msgs :*********************
--------------------------------

PlainMsg:
safeFactor: 2.11364
Msg Len: 22
ownerClass: CLS_D
--------------------------------

******************************************************
safest of all Plain msgs:
--------------------------------

PlainMsg:
safeFactor: 2.11364
Msg Len: 22
ownerClass: CLS_D
--------------------------------

safest of all secure msgs:
--------------------------------

SecureMsg:
safeFactor: 1.41071
Msg Len: 28
securityType: PKI  keyStrength: MID
--------------------------------
```

//--

Notes:

- The above diagram is only partial and symbolic:
  - You should determine the access level of all data/methods: `public/private/protected` as well as their abstraction: virtual or not.
  - You may add any necessary data/methods you see fit: Ctors/Dtors, helper methods etc.
  - You may add more arguments to the methods shown in the UML and modify their return types

- Memory management:
  - `msgAdmin is singelton`
  - The “real” massages are created outside the admin module. But `msgAdmin` must create and work only on its own copy of `Msg`S and not on any reference to external `Msg`S.
  - All objects (such as `msg`S, `msgAdmin` etc.`)` must be released before the program terminates.

- Error management: Make sure you handle invalid arguments for the objects: overflows, invalid parameters etc.

Part B (40% , 5% per question)

1. what is the output of the following Code ?!

```
class test{
        public:
                static int n;
                test() {n++;};
                ~test() {n--;};
};
int test::n=0;
int main()
{
        test a;
        test b[5];
        test *c = new test;
        cout << a.n << endl;
        delete c;
        cout << test::n << endl;
        return 0;
}
```

   - **a.** 7 6
   - **b.** 6 7
   - **c.** 5 6
   - **d.** 6 5

2. by default, members of the class are _____________.
   - **a.** protected
   - **b.** private
   - **c.** public
   - **d.** static

3. What is the minimal number of data-members possible in a class?
   - **a.** Minimum one: defined by the programmer
   - **b.** Minimum one: the virtual table pointer, defined by the compiler
   - **c.** Minimum two: the first defined by the programmer, the 2nd is the VT pointer defined by the compiler
   - **d.** Zero

4. Can a `static` method be declared `const`?
   - **a.** Yes, if it doesn't intend to modify any of the class members
   - **b.** No, a static method is not object related but rather class related
   - **c.** Yes, if it doesn't intend to modify any of the `static` class members
   - **d.** Yes, if it's used within another non-static method
   - **e.** Answers b + d are correct

5. Which value we cannot assing to reference ?
   - **a.** int
   - **b.** float
   - **c.** unsigned
   - **d.** null
   - **e.** none of the above
   - **f.** all answers are correct

6. what is the correct sentence about reference and pointer ?
   - **a.** we cannot create an array of reference
   - **b.** we can create array of reference
   - **c.** we can use reference to reference
   - **d.** none of the above

7. RunTime Polymorphism is achieved by ___________
   - **a.** friend function
   - **b.** virtual function
   - **c.** operator overloading
   - **d.** function overloading

8. What is “polymorphism”? How is it implemented in C++? Explain with an example.

Good Luck !