```
qestion * p;
for(vector< qestion *>:: const_iterator i = q.qe.Begin(); q != q.qe.end();
iT++){
   if(Typeid(**i)==Typeid(QC)){
         P = new QC~~<static_cast>~~(**i);
   }else{
      P= new QCV(static_cast <const QCV &>(**i));
   }
   qe.Push_Back(P);
 }
}
retun *this;
}
```

```
Qestionnaire :: Qestion operator - (const QCU & QCU)
{
for (vector <Qestion*> :: const iterator iT = QCU.Begin(); iT != Q.end() : iterat ){
 {
    if (iT -> score() != 1){
              iT = erase (iT);
    }else{
        iT ++;
    }
  }
}
```

9) #include <fstream>

```
 ofstream file ("save.txt");
  Try{
    if (!file){
           Throw exception():
    }else{
    for (vector <Qestion*> :: iterator iT = qe.Begin(); iT != qe.end(); iT ++)
      {
       file << iT -> get_Text() << endl;
       file << it -> get_cc() << endl;
       file << it -> get_num_rp() << endl;
    }
  }
catch (exception e){
  cout << e.what() << endl;
 }
}
```

10) Voir Page 3

11)
```
 Qestionnaire& Qestionnaire :: operator = (const Qestion &Q)
  { if (&Q != this){
      for ( vector < Question* > :: iterator j = ~~[illegible]~~ qe.Begin(); j != qe.end()
iT++){
      delet (j*):
   }
```

5

```
5) Bool Gestionnair :: checke (int id){          Question operator - (const QCU & QCU)
   int index = 0;
   for (vector < Question* > :: iterator iT = qe. Begin(); qiT != qe end() ){
         if (id == iT -> get_id() ){
               index = iT;
               return True,
         }else{
               {
         }   iT++ ;
   }
   cout << "index = " << index << endl;
}

6) void Gestionnair :: Ajoute_QU (const Question & Qed, int id)
   {
     if (checke (id) ) // True
     {
       Question *Ptr = new QC (Qed);
       Qe. Push_Back (Ptr);
     }else{
       cout << "id existe" << endl;
   }  }

7) int Gestionnair :: calc_Scor final ()
   {
     int Score = 0;
   for (vector < Question* > :: iterator iT = Qe Begin(), iT != Qe. End() ; iT++){
      Score += iT -> get_Score();
   }
   return Score ;
}

8) QCU : QCU operator _ ()
   {
   QCU nQCU (*this);
   nQCU. num_re = -1;
   return nQCU;
}
```

QCU.cpp

```
#include <iostream>
#include "QCU.cpp"
using namespace std;
QCU::QCU() {
  num_re = 0; num_rc = 0;
}
QCU::QCU(int num_re, int num_rc, int Ref, String Text): gestion(Ref, Text){
   this -> num_re = num_re;
   this -> num_rc = num_rc;
}

int QCU:: get_numRe() const { return num_re ;}
int QCU:: get_numRc() const { return num_Rc; }
int QCU:: Score () {
   if (num_Rc == num_Re){
       set_Score (1);
   }else{
       return 0;
   }
}
```

4) Questionnaire .h:

```
class Questionnaire
{
  Public:
   Gestionnaire () {};
   Virtual ~ Gestionnaire () {}; // destructor
   Bool cherche (int id);
   Void ajouter_QU (const Gestion & QC);
   int calc_Score final ();
   void display_all();
   Void Save ( );
   Gestinnaire (const Gestionnaire & Q); // copy constructor
   Gestionnaire operator = (const Gestionnaire & Q); // Surcharge operator
Protected:
  int get_id ();
Private:
  int id; Vector < gestion * > QR // vector dynamique
```

Regle de 3

```
             Qestion operator - (const QCU & QCU)
5) Bool Gestionnaire :: checke (int id){
    int index = 0;
    for (vector < Qestion* > :: iterator iT = qe. Begin(); qiT != qe end() ){
        if (id == iT -> get_id() ){
            index = iT;
            return True,
        }else{
        }   iT++;
    }
    cout << "index = " << index << endl;
}

6) void Gestionnaire :: Ajoute_QU ( const Qestion & Qed, int id)
    {
      if (checke (id) ) // True
      {
        Qestion *ptr = new QC (qed);
        Q. Push_Back (ptr);
      }else{
        cout << "id existe" << endl;
    }  }

7) int Gestionnaire :: calc_Scor final ()
    {
     int Score = 0;
   for (vector < Qestion* > :: iterator iT = Q. Begin (), iT != Qe. End() ; iT++){
       Score += iT -> get_Score();
   }
   return Score ;
  }

8) QCU : QCU operator - ()
   {
    QCU nqcu (*this);
    nQCU. num_re = -1;
    return nQCU;
  }
```

[illegible] CPP

2) QC.h

```
class QC : PuBlic qestion
 {
   PuBlic :
     QC ();
     virtual ~ QC();
     QC (int cc, int ce, int num, int Rf, String Text );
   int Score () override ;
 Protected :
 int  get_ classement erreurs () const;
 int  get_ classement correct () const;
 int  get_ num classement () const;
 Private :
 int ce, cc, num ;
 };
```

3) QCU.h :

```
class QCU : PuBlic qestion
 {
   PuBlic :
   QCU() {} ;
   QCU operation - (const QCU & QU);
   Virtual ~ QCU(){};
   QCU (int num_re, int num_rc, int Rf, String Text);
   int Score () override ;
 Protected :
 int get_ num Re () const ;
 int get_ num RC () const;
 Private :
 int num_re, num_rc ;

};
```

Examen 2025

1) Gestion.h

```
class Gestion {
 Public:
  Gestion();
  Virtual ~ Gestion();
  Gestion (int Ref, String Test, ~~int score~~);
  Virtual ~~float~~ int score() =0   // ABSTRACT class (Tout les class droit ajoute cette Methode)
 Protected:
  int get_Ref() const;
  int get_Score() const;      } Parmi Les Bonnes Pratique en utilise
  String get_Test() const;      const avec Les getters
  void Set_Score (int Score);
 Private:
 int Ref, Score;
  String Test;
};
```

Gestion.cpp:

```
 #include <iostream>
 #include "Gestion.h"
 Using namespace std;

Gestion::Gestion(){
  Ref =0; Score =0; Test =" "
 }
Gestion::Gestion(int Ref, String Test, ~~int score~~){
  this -> Ref = Ref;
  this -> Test = Test;
  ~~this -> Score = Score;~~
 }
int Gestion:: get_Ref(){ return Ref; }
int Gestion:: get_Score() const { return Score; }
Test Gestion:: get_Test() const { return Test; }
String
void Gestion:: Set_Score (int Score){
 this -> Score = Score
};  }
```