COMP 2012 Midterm Exam - Spring 2017 - HKUST

Date: March 25, 2017 (Saturday)
Time Allowed: 2 hours, 2–4 pm
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **7** questions on **24** pages (including this cover page and 3 blank pages at the end).
3. Write your answers in the space provided in black/blue ink. *NO pencil please, otherwise you are not allowed to appeal for any grading disagreements.*
4. All programming codes in your answers must be written in the ANSI C++ version as taught in the class.
5. For programming questions, unless otherwise stated, you are **NOT** allowed to define additional structures, use global variables nor any library functions not mentioned in the questions.

Student Name	Solution & Marking Scheme
Student ID	
Email Address	
Seat Number	

For T.A. Use Only

Problem	Score
1	/ 10
2	/ 12
3	/ 10
4	/ 13
5	/ 8
6	/ 24
7	/ 23
Total	/ 100

Problem 1 [10 points] True or false

Indicate whether the following statements are *true* or *false* by circling **T** or **F**. You get 1.0 point for each correct answer, −0.5 for each wrong answer, and 0.0 if you do not answer.

T **F** (a) The declaration "Thing* const p;" in a function means that no statement in that function can modify the Thing object that p points to via p.

T **F** (b) Both class member functions and global functions can be inline functions.

T **F** (c) If you have defined at least one constructor (which is neither the copy constructor nor the default constructor) in a class, then C++ will NOT provide the default constructor and the copy constructor to the class.

T **F** (d) There is a compilation error in the following code segment.

```
class A
{
  int x;
  public:
    A(const A a) { x = a.x; }
};
```

T **F** (e) If objects A and B both belong to the same class, then A and B can access each other's private data members through their member functions.

T **F** (f) If you use public inheritance or protected inheritance, you can reference a derived class object with a base class pointer.

T **F** (g) Slicing does not occur when you assign a base class pointer to refer to a derived class object.

T **F** (h) The following code is legal. That is, it is syntactically correct and can be compiled without errors.

```
#include <iostream>
using namespace std;

class B;  // Forward declaration of class B

class A: public B
{
  public:
    A() { cout << "A's constructor" << endl; }
    ~A() { cout << "A's destructor" << endl; }
};

class B : public A
{
  public:
    B() { cout << "B's constructor" << endl; }
    ~B() { cout << "B's destructor" << endl; }
};

int main() { A a; B b; return 0; }
```

T **F** (i) If class D is derived from class B which has a virtual member function g, then D must also implement its own version of the virtual member function g.

T **F** (j) The destructor of a derived class is made virtual if the base class destructor is virtual.

Problem 2 [12 points] Order of Constructions and Destructions

```
#include <iostream>
using namespace std;

class A
{
  public:
    A() { cout << "A's constructor" << endl; }
    ~A() { cout << "A's destructor" << endl; }
};

class B
{
  public:
    B() { cout << "B's constructor" << endl; }
    ~B() { cout << "B's destructor" << endl; }
  private:
    A a;
};

class C
{
  public:
    C() { cout << "C's constructor" << endl; }
    ~C() { cout << "C's destructor" << endl; }
};

class D : public C
{
  public:
    D() { cout << "D's constructor" << endl; }
    ~D() { cout << "D's destructor" << endl; }
  private:
    B* b[2];
};

class E: public D
{
  public:
    E() { cout << "E's constructor" << endl; }
    ~E() { cout << "E's destructor" << endl; }
  private:
    B b;
};

int main()
{
    E e;
    return 0;
}
```

(a) What are the outputs of the above program?

```
C's constructor
D's constructor
A's constructor
B's constructor
E's constructor
E's destructor
B's destructor
A's destructor
D's destructor
C's destructor
```

Answer: _____

(b) If we modify line #27 and line #36 of the above program as follows:

```
class D : private C
class E : protected D
```

what will be the new outputs?

Answer: **The same 10 lines of output as in part (a)**

Grading scheme:
(a) 1 point for each of the 10 output lines
(b) total 2 points: 1 point for constructors' output, 1 point for destructors' output. It is fine if students just simply write "same outputs as in (1)".
- must show correct order of constructors: C,D,A,B,E
- must show correct order of destructors: E,B,A,D,C
- for each sequence of the 5 output lines from constructors or destructors, scan from top to bottom and look for the longest correct sequence of outputs which must be in the correct order.
- 1 point for each correct match
- -1 for each insertion error

Problem 3 [10 points] Inheritance

Suppose we have the following sample program:

```
#include <iostream>
using namespace std;
class A
{
  public:
    A() { cout << "A's constructor 1" << endl; }
    A(const A& a) { cout << "A's constructor 2" << endl; }
    A(const char* msg) { cout << "A's constructor 3" << endl; }
    virtual ~A() { cout << "A's destructor" << endl; }
    virtual void funny() const { cout << "A's funny" << endl; }
};

class B : public A
{
  public:
    B() { cout << "B's constructor 1" << endl; }
    B(const B& b) { cout << "B's constructor 2" << endl; }
    B(const char* msg) : A(msg) { cout << "B's constructor 3" << endl; }
    virtual ~B() { cout << "B's destructor" << endl; }
    virtual void funny() const { cout << "B's funny" << endl; }

  protected:
    A obj;
};

A funny(A input)
{
  input.funny();
  cout << "::funny" << endl;
  return input;
}

int main() {
  cout << "---Part 1---:" << endl;
  A Obj0;
  cout << endl;

  cout << "---Part 2---:" << endl;
  B Obj1("Comp2012");
  cout << endl;

  cout << "---Part 3---:" << endl;
  B Obj2(Obj1);
  cout << endl;

  cout << "---Part 4---:" << endl;
  Obj0 = funny(Obj2);
  cout << endl;

  cout << "---Part 5---:" << endl;
}
```

What are the outputs of the above program? Write the outputs below. Assume the compiler does not do any optimization.

Answer:

—Part 1—:

```
A's constructor 1 // create A class Obj0
```

—Part 2—:

```
A's constructor 3 // create B class Obj1, which calls A class conversion constructor
A's constructor 1 // create the A class obj data member
B's constructor 3 // create the B class Obj1
```

—Part 3—:

```
A's constructor 1 // create A class base for the B class
A's constructor 1 // create A class data member obj
B's constructor 2 // create B class Obj2 using copy constructor
```

—Part 4—:

```
A's constructor 2 // Pass by value, copy B class Obj2 to A class input
A's funny         // run A::funny()
::funny           // run global funny()
A's constructor 2 // return by value, A class copy constructor
A's destructor    // function goes out of scope, object input destroyed
A's destructor    // destruct the returned A class object
```

—Part 5—:

```
B's destructor // main() exits, destroy Obj2
A's destructor // destroy obj data member of Obj2
A's destructor // destroy base class of Obj2
B's destructor // destroy Obj1
A's destructor // destroy obj data member of Obj1
A's destructor // destroy base class of Obj1
A's destructor // destroy Obj0
```

Grading scheme:

0.25 point for the presence of a correct term, 0.25 point for the correct relative position.

Total 10 points. Similar to question 2, -0.25 for each wrong insertion.

-0.25 for each extra incorrect statement.

Minimum score for each part is zero.

Problem 4 [13 points] Inheritance

The following program contains 5 ERRORS (syntax errors, logical errors, etc.). Study the program carefully, identify all the errors by writing down the line number where an error occurs, and explain why it is an error.

```
1  #include <iostream>
2  using namespace std;
3
4  class A { };
5
6  class Base {
7    protected:
8      int base;
9      A** p;
10   public:
11     Base(int base, A arr[]) {
12       base = base;
13       p = new A*[5];
14       for(int i=0; i<5; i++)
15         p[i] = &(arr[i]);
16     }
17
18     ~Base() {
19       for(int i=0; i<5; i++)
20         delete p[i];
21       delete [] p;
22     }
23 };
24
25 class Derived : public Base {
26   private:
27     int* q;
28   public:
29     Derived(int base, A arr[]) : Base(base, arr) {
30       q = new int[10];
31     }
32     explicit Derived(const Derived& derived) : Base(derived.base, *(derived.p)) {
33       q = derived.q;
34     }
35     ~Derived() {
36       delete [] q;
37     }
38 };
39
40 int main() {
41   A arr[5] = { A(), A(), A(), A(), A() };
42   Base* bPtr = new Derived(10, arr);
43   delete bPtr;
44   Derived d = *(static_cast<Derived*>(bPtr));
45   return 0;
46 }
```

Solution 1:

Error#	Line#	Explanation
1	12	Should not be self-assignment. Instead, we should do this->base = base;
2	18	Should be virtual destructor to facilitate proper de-allocation of dynamic memory, virtual ~Base()
3	19-20	Should not de-allocate static objects. Instead, we should do delete [ ] p;

4	33	Should do deep copying instead of memberwise copying
		q = new int[10];
		for(int i=0; i<10; i++)
		q[i] = derived.q[i];

5	44	As the derived class object has been de-allocated, dereference of casted pointer is illegal.

Grading scheme: 0.5 point for each correct line number (Error #3 is worth 1 point), 2 points per correct explanation

Solution 2:

Error#	Line#	Explanation
1	12	Should not be self-assignment. Instead, we should do this->base = base;
2	19-20	Should not de-allocate static objects. Instead, we should do delete [ ] p;

3	33	Should do deep copying instead of memberwise copying
		q = new int[10];
		for(int i=0; i<10; i++)
		q[i] = derived.q[i];

4	43	De-allocation of Derived class object should not be done here.
5	44 or 32	For line 44: Should use explicit call instead of implicit call, i.e. Derived d(*(static_cast<Derived*>(bPtr))); For line 32: Should not be explicit, i.e. Derived(const Derived& derived)

Grading scheme: 0.5 point for each correct line number (Error #2 is worth 1 point), 2 points per correct explanation

Problem 5 [8 points] Const-ness

In the following program, for the 8 statements ending with the following comment:

```
/* Error:   Yes   /   No   /   Don't know   */
```

decide whether the statement is syntatically INCORRECT - that is, it will produce compilation error(s). Circle "Yes" if it will give compilation error and "No" otherwise.
You get 1 point for each correct answer, -0.5 for each wrong answer, and 0.0 if you do not answer by circling "Don't know".

```
#include <iostream>
using namespace std;

class A {
  public:
    void nonConstFunc() {}
    void constFunc() const {}
};

void doSomething(A& a) { cout << "do something" << endl; }
void doMore(const A& a) { cout << "do more" << endl; }

int main() {
  A aObj;
  A* const cpa = &aObj;
  const A* pca = &aObj;

  cpa->nonConstFunc();   /* Error:   No   */
  cpa->constFunc();      /* Error:   No   */
  pca->nonConstFunc();   /* Error:   Yes  */
  pca->constFunc();      /* Error:   No   */
  doSomething(*cpa);     /* Error:   No   */
  doMore(*cpa);          /* Error:   No   */
  doSomething(*pca);     /* Error:   Yes  */
  doMore(*pca);          /* Error:   No   */

  return 0;
}
```

Grading scheme:

- 1 point for each correct answer
- -0.5 point for each wrong answer
- 0 point if students do not answer by circling "Don't know"
- The minimum score of this question is 0

Problem 6 [24 points] Inheritance, Polymorphism and Dynamic Binding

This problem involves 3 classes called '`Balloon`', '`FoilBalloon`' (derived from '`Balloon`' by using public inheritance), and '`BalloonBouquet`'. Below are the header files of the 3 classes.

```
/* File: Balloon.h */

#ifndef BALLOON_H
#define BALLOON_H

class Balloon
{
  private:
    double maxRadius;      // The maximum radius of the balloon
    double inflationRatio; // Amount of gas required for unit radius increment
    double radius;         // The current radius of the balloon
    double volume;         // The volume of gas in the balloon
    bool popped;           // Flag that indicates whether the balloon is popped

  public:
    /* ----- Functions to implement -----  */

    // Construct a balloon with the given maximum radius, inflation ratio, current radius,
    // current volume and a boolean value representing whether the balloon is popped or not
    // Initially, the balloon is deflated (radius is 0, volume is 0) and is not popped
    Balloon(double maxRad = 0, double ratio = 0, double r = 0, double v = 0, bool p = 0);

    // Increase the radius of the balloon according to the input volume of gas
    // The balloon is popped if the radius is increased beyond the maximum radius
    // A popped balloon has radius 0 and volume 0
    // No effect if the balloon is already popped
    virtual void blow(int vol);

    // Print all the data of balloon
    virtual void print() const;

    /* ----- Accessor member functions ----- */

    // Return maxRadius of the balloon
    double getMaxRadius() const { return maxRadius; }

    // Return inflationRatio of the balloon
    double getInflationRatio() const { return inflationRatio; }

    // Return the current radius of the balloon
    double getRadius() const { return radius; }

    // Return the current volume in the balloon
    double getVolume() const { return volume; }

    // Check whether the ballooon is popped
    bool isPopped() const { return popped; }
};

#endif /* BALLOON_H */
```

```
/* File: FoilBalloon.h */

#ifndef FOILBALLOON_H
#define FOILBALLOON_H

#include "Balloon.h"

class FoilBalloon : public Balloon {
  private:
    double maxGasVolume; // The maximum gas volume of the foil balloon

  public:
    // Construct a foil balloon with the given maximum radius, maximum gas volume and
    // inflation ratio, current volume and a boolean value representing whether the
    // foil balloon is popped or not
    // As foil ballon is not elastic, its radius would always remain as max radius
    FoilBalloon(double rad = 0, double ratio = 0, double maxGasVol = 0, double v = 0, bool p = 0);

    // As foil balloon is not elastic, its radius would always remain as max radius and
    // will not be popped because of the gas injection
    // Increase the gas volume of the balloon according to the input gas volume,
    // subject to the maximum gas volume
    void blow(int vol);

    // Prints all the data of FoilBalloon
    void print() const;
};

#endif /* FOILBALLOON_H */

/* File: BalloonBouquet.h */

#ifndef BALLOONBOUQUET_H
#define BALLOONBOUQUET_H

#include "FoilBalloon.h"

class BalloonBouquet {
  private:
    Balloon** bouquet; // A pointer which points to an array of pointers in Balloon type
    int numBalloon;    // The number of balloons for the bouquet
                       // It refers to the size of the pointer array pointed by bouquet
  public:
    // Default constructor
    BalloonBouquet() : bouquet(NULL), numBalloon(0) { }

    // Copy constructor - Perform deep copying
    //
    // Note: Two different types of objects will be pointed by the array of pointers.
    //       Create Balloon object when the object to be duplicated is in Balloon type.
    //       Create FoilBalloon object when the object to be duplicated is in FoilBalloon type.
    //
    // Hint: Use typeid(<type>).name() and typeid(<expression>).name()
    BalloonBouquet(const BalloonBouquet& bb);

    // Destructor - De-allocate all dynamically-allocated memory to avoid any memory
    // leak as the program finishes
    ~BalloonBouquet();

    // Adds the balloon to the bouquet
    void addBalloon(Balloon& balloon);
};

#endif /* BALLOONBOUQUET_H */
```

Below is the testing program "`test-balloon.cpp`".

```
/* File: test-balloon.cpp */

#include <iostream>
#include "BalloonBouquet.h"
using namespace std;

void simulate_and_print(Balloon* b) {
  cout << "=== Initial ===" << endl;
  b->print();
  cout << "=== Blow 5 units ===" << endl;
  b->blow(5); b->print();
  cout << "=== Blow 10 units ===" << endl;
  b->blow(10); b->print();
}

int main() {
  Balloon* balloonArr[2] = { new Balloon(10,1), new FoilBalloon(30, 2, 30) };
  for(int i=0; i<2; i++) {
    cout << ( (i==0) ? "[ Balloon ]" : "[ Foil Bolloon ]" ) << endl;
    simulate_and_print(balloonArr[i]);
  }
  BalloonBouquet bouquet;
  bouquet.addBalloon(*(new Balloon(40,2)));
  bouquet.addBalloon(*(new FoilBalloon(50, 2, 40)));
  BalloonBouquet anotherBouquet(bouquet);
  return 0;
}
```

A sample run of the test program is given as follows:

```
[ Balloon ]
=== Initial ===
Max rad., Inflat. rat., rad, vol, popped:  10, 1, 0, 0, No
=== Blow 5 units ===
Max rad., Inflat. rat., rad, vol, popped:  10, 1, 5, 5, No
=== Blow 10 units ===
Max rad., Inflat. rat., rad, vol, popped:  10, 1, 0, 0, Yes
[ Foil Balloon ]
=== Initial ===
Max rad., Inflat. rat., rad, vol, popped:  30, 2, 30, 0, No
Maximum gas volume: 30
=== Blow 5 units ===
Max rad., Inflat. rat., rad, vol, popped:  30, 2, 30, 5, No
Maximum gas volume: 30
=== Blow 10 units ===
Max rad., Inflat. rat., rad, vol, popped:  30, 2, 30, 15, No
Maximum gas volume: 30
```

Based on the given information, complete the implementation of '`Balloon`' class and '`FoilBalloon`' class in their respective .cpp files, namely, "`Ballooon.cpp`" and "`FoilBalloon.cpp`" respectively.

(a) [7 points] Implement all member functions of the class '`Balloon`' in a separate file called "`Balloon.cpp`".

Answer: `/* File "Balloon.cpp" */`

```
/* File: Balloon.cpp */

#include <iostream>
#include "Balloon.h"
using namespace std;

Balloon::Balloon(double max, double ratio, double r, double v, bool p)
                                  // If default values are put, -1 point
{
  maxRadius = max;                                          // 0.5 point
  inflationRatio = ratio;                                   // 0.5 point
  radius = r;                                               // 0.5 point
  volume = v;                                               // 0.5 point
  popped = p;                                               // 0.5 point
}

void Balloon::blow(int vol)
{
  if(!popped)                                               // 0.5 point
  {
    volume += vol;                                          // 0.5 point
    radius += vol / inflationRatio;                         // 0.5 point
    if(radius > maxRadius)                                  // 0.5 point
    {
      popped = true;                                        // 0.5 point
      radius = 0;                                           // 0.5 point
      volume = 0;                                           // 0.5 point
    }
  }
}

void Balloon::print() const
{
  cout << "Max rad., Inflat. rat., rad, vol, popped:  "; // 0.5 point
  cout << maxRadius << ", " << inflationRatio << ", " << radius << ", " << volume << ", " <<
          ((popped) ? "Yes" : "No") << endl;             // 0.5 point
}
```

(b) [8 points] Implement all member functions of the class '`FoilBalloon`' in a separate file called "`FoilBalloon.cpp`".

Answer: `/* File "FoilBalloon.cpp" */`

```
/* File: FoilBalloon.cpp */

#include <iostream>
#include "FoilBalloon.h"
using namespace std;

FoilBalloon::FoilBalloon(double rad, double ratio, double maxGasVol, double v, bool p)
                        : Balloon(rad, ratio, rad, v, p), maxGasVolume(maxGasVol)
                                                  // Calling Balloon constructor: 1 point
                                                  // Initialization of maxGasVolume: 0.5 point
                                                  // If default values are put, -1 point
{
}

void FoilBalloon::blow(int vol)
{
  double newVol = getVolume() + vol;                          // 1.5 points
  newVol = (newVol > maxGasVolume) ? maxGasVolume : newVol;   // 1.5 points

                                                              // 2 points
  *this = FoilBalloon( getMaxRadius(), getInflationRatio(), maxGasVolume, newVol, false );
}

void FoilBalloon::print() const
{
  Balloon::print();                                           // 1 point
  cout << "Maximum gas volume: " << maxGasVolume << endl;     // 0.5 point
}
```

(c) [9 points] Implement all member functions of the class '`BalloonBouquet`' in a separate file called "`BalloonBouquet.cpp`".

Answer: `/* File "BalloonBouquet.cpp" */`

Solution 1:

```
/* File: BalloonBouquet.cpp */

#include <iostream>
#include "BalloonBouquet.h"
using namespace std;

BalloonBouquet::BalloonBouquet(const BalloonBouquet& bb)
{
  bouquet = new Balloon*[bb.numBalloon];                                  // 0.5 point
  numBalloon = bb.numBalloon;                                             // 0.5 point
  for(int i=0; i<numBalloon; i++)                                         // 0.5 point
  {
    if( !strcmp(typeid(*bb.bouquet[i]).name(), typeid(Balloon).name()) ) // 1 point
    // Alternative: if(typeid(*bb.bouquet[i]) == typeid(Balloon))
      bouquet[i] = new Balloon;                                           // 0.5 point
    else
      bouquet[i] = new FoilBalloon;                                       // 0.5 point
    *bouquet[i] = *bb.bouquet[i];                                         // 0.5 point
  }
}

BalloonBouquet::~BalloonBouquet()
{
  for(int i=0; i<numBalloon; i++)                                         // 0.5 point
    delete bouquet[i];                                                    // 0.5 point
  delete [] bouquet;                                                      // 0.5 point
}

void BalloonBouquet::addBalloon(Balloon& balloon)
{
  Balloon** temp = new Balloon*[numBalloon + 1];                          // 0.5 point
  for(int i=0; i<numBalloon; i++)                                         // 0.5 point
    temp[i] = bouquet[i];                                                 // 0.5 point
  temp[numBalloon] = &balloon;                                            // 0.5 point
  delete [] bouquet;                                                      // 0.5 point
  bouquet = temp;                                                         // 0.5 point
  numBalloon++;                                                           // 0.5 point
}
```

Solution 2:

```
/* File: BalloonBouquet.cpp */

#include <iostream>
#include "BalloonBouquet.h"
using namespace std;

BalloonBouquet::BalloonBouquet(const BalloonBouquet& bb)
{
  numBalloon = 0;                                                          // 0.5 point
  bouquet = new Balloon*[0];                                               // 0.5 point
  for(int i=0; i<bb.numBalloon; i++)                                       // 0.5 point
    addBalloon(*(bb.bouquet[i]));                                          // 0.5 point
}

BalloonBouquet::~BalloonBouquet()
{
  for(int i=0; i<numBalloon; i++)                                          // 0.5 point
    delete bouquet[i];                                                     // 0.5 point
  delete [] bouquet;                                                       // 0.5 point
}

void BalloonBouquet::addBalloon(Balloon& balloon)
{
  Balloon** temp = new Balloon*[numBalloon + 1];                           // 0.5 point
  for(int i=0; i<numBalloon; i++)                                          // 0.5 point
    temp[i] = bouquet[i];                                                  // 0.5 point
  if(typeid(balloon).name() == typeid(Balloon).name())                     // 1 point
    temp[numBalloon] = new Balloon(balloon);                               // 0.5 point
  else
    temp[numBalloon] = new FoilBalloon(dynamic_cast<FoilBalloon&>(balloon)); // 1 point
  delete [] bouquet;                                                       // 0.5 point
  bouquet = temp;                                                          // 0.5 point
  numBalloon++;                                                            // 0.5 point
}
```

Problem 7 [23 points] Abstract Base Class

Given the definition of the class 'Date' below:

```
/* File: Date.h */

#ifndef DATE_H
#define DATE_H
using namespace std;

class Date
{
  public:
    Date(int d, int m) : dd(d), mm(m) {}; // d is the day, m is the month
    int getdd() const { return dd; }
    int getmm() const { return mm; }

  private:
    int dd; // day information
    int mm; // month information
};

#endif /* DATE_H */
```

Your task is to implement an abstract base class 'Bill' and two classes 'CreditCardBill' and 'OnlineBill' directly derived from 'Bill' using public inheritance. The two classes are supposed to work with the "main.cpp" program shown below:

```
/* main.cpp */

#include <iostream>
#include "Date.h"
#include "OnlineBill.h"
#include "CreditCardBill.h"
using namespace std;

int main()
{
  Bill* bills[2];

  bills[0] = new CreditCardBill(2000.00, 200.00, 25, 3);
  bills[1] = new OnlineBill(2017.00, 25, 3);

  cout << endl << "-------The Bills are --------" << endl;
  bills[0]->print();
  bills[1]->print();

  cout << endl << "-------After paying back the bills are-------" << endl;
  bills[0]->pay(2000);
  bills[1]->pay(2000);
  bills[0]->print();
  bills[1]->print();

  cout << endl << "-------After paying the exact bill for the OnlineBill-------" <<endl;
  bills[1]->pay(2017);
  bills[1]->print();
}
```

Your two classes together with "`main.cpp`" are supposed to produce the following output:

```
-------The Bills are -------
Credit card bill
Date: 25/3
You owe $2000
Service charge $200
You have totally paid $0 and you owe the bank $2200 net.

Online bill
Date: 25/3
You owe $2017


-------After paying back the bills are-------
Credit card bill
Date: 25/3
You owe $2000
Service charge $200
You have totally paid $2000 and you owe the bank $200 net.

Online bill
Date: 25/3
You owe $2017


-------After paying the exact bill for the OnlineBill-------
Online bill
Date: 25/3
You owe $0
```

(a) [7 points] Based on the given information, define the abstract base class '`Bill`' in the "`Bill.h`" file. This class has a private data member '`date`' which is a '`Date`' class object and it indicates the date of a bill. The '`Bill`' class has a constructor that initializes the '`date`' data member, and other functions as listed below:

- **Bill**: Constructor, accepts two integer arguments and initializes the '`date`' data member of the class with the arguments.
- **printBillDate**: This function is used in the "`CreditCardBill::print()`" function and the "`onlineBill::print()`"functions to output the bill date.
- **getOwedAmount**: Pure virtual function. Takes no input, returns a float number indicating the total bill amount owed.
- **pay**: Pure virtual function. Takes a float '`p`' as the amount the user pays back. It does not return anything.
- **print**: Pure virtual function. Prints the bill information, see "`main.cpp`" and the outputs to get more information. This function does not return anything.

Answer: `/* File "Bill.h" */`

```
/* File: Bill.h */

#ifndef BILL_H
#define BILL_H

#include <iostream>
#include "Date.h"                                              // 0.5 point

class Bill
{
  public:
    Bill(int dd, int mm): date(Date(dd, mm)) {};               // 1.0 point
    virtual float getOwedAmount() const = 0;                   // 1.0 point
    virtual void pay(float p) = 0;                             // 1.0 point
    virtual void print() const = 0;                            // 1.0 point
    void printBillDate() const                                 // 1.5 point
    {
      cout << "Date: " << date.getdd() << "/" << date.getmm() << endl;
    }

  private:
    Date date;                                                 // 1.0 point
};

#endif /* BILL_H */
```

Grading scheme:

(a) Give points according to the comment on each line of the program.

0.25 for the keywords const.

0.5 or 0 for correct prototype in the printBillDate() function.

0.25 for The keyword virtual

0.5 for =0 in pure virtual function

(b) [8 points] Define and implement the class '`CreditCardBill`' in the file "`CreditCardBill.h`". The '`CreditCardBill`' class is derived from '`Bill`' using public inheritance. It has three private data members: '`billA`' (type float, indicating the bill amount), '`serviceC`' (type float, indicating the service charge owed), and '`paymentR`' (type float, indicating the amount the user has paid). For the '`paymentR`' data member, it is initially 0. It accumulates all the payments the user has made. For example if the user pays $1000 through the '`pay`' member function and then he/she pays another $200 through the same member function, '`paymentR`' should become 1000+200=1200.

Note:

1. Implement an appropriate constructor, and all the pure virtual member functions in the same "`CreditCardBill.h`" file.
2. Assume all user inputs are valid and you do not need to perform any check on them.
3. The net amount owed by the user = '`billA`' + '`serviceC`' – '`paymentR`' (you do not need to check whether this expression is positive).

Answer: `/* CreditCardBill.h */`

```
/* File: CreditCardBill.h */

#ifndef CREDITCARDBILL_H
#define CREDITCARDBILL_H

#include "Date.h"
#include "Bill.h"                                                    // 1.0 point

class CreditCardBill: public Bill                                    // 0.5 points
{
  public:
    CreditCardBill(float arg_bill, float arg_service, int dd, int mm): // 1.0 point
                   Bill(dd,mm), billA(arg_bill), serviceC(arg_service), paymentR(0) {}
    virtual float getOwedAmount() const {                            // 1.0 point
      return billA + serviceC - paymentR;
    }
    virtual void pay(float p) {                                      // 1.0 point
      paymentR = paymentR + p;
    }
    virtual void print() const {                                     // 2.0 point
      cout << "Credit card bill" << endl;
      printBillDate();
      cout << "You owe $" << billA << endl;
      cout << "Service charge $" << serviceC << endl;
      cout << "You have totally paid $" << paymentR <<
            " and you owe the bank $" << getOwedAmount() << " net." << endl << endl;
    }

  private:
    float billA;                                                     // 0.5 point
    float serviceC;                                                  // 0.5 point
    float paymentR;                                                  // 0.5 point
};

#endif /* CREDITCARDBILL_H */
```

Grading scheme:

(a) Give points according to the comment on each line of the program.

(b) Correct inheritance 0.5 points as being shown on the comment line.

(c) For const functions, 0.25 points for the keyword 'const'.

(d) 0.25 points in total are deducted if the data members are not under the 'private' section

(c) [8 points] Define and implement the class 'OnlineBill' in the "OnlineBill.h" file. The 'OnlineBill' class is derived from 'Bill' using public inheritance. It has one private data member: 'amount' (type float, indicating the net amount of bill owed by the user). Unlike the 'CreditCardBill' where we allow partial payments, 'OnlineBill' needs the user to pay the full owed 'amount' through the 'pay' function. If the user wants to pay his/her bill using 'pay' member function, and if the payment is not exactly equal to 'amount', the 'pay' function should reject the payment. When the 'pay' function rejects a payment, there is no need to display any warning message (see the output for more information). Implement all the inherited member functions and a constructor to initialize all the data member, so that it works with the "main.cpp" file.

Answer: /* OnlineBill.h */

```
/* File: OnlineBill.h */

#ifndef ONLINEBILL_H
#define ONLINEBILL_H

#include <iostream>
#include "Date.h"
#include "Bill.h"                                                    // 1.0 point

class OnlineBill: public Bill                                        // 0.5 points
{
  public:                                                            // 1.0 point onlineBill()
    OnlineBill(float arg_amount,int dd, int mm) : Bill(dd,mm), amount(arg_amount) {}
    virtual float getOwedAmount() const { return amount; }           // 1.0 point
    virtual void pay(float p) {                                      // 2.0 points
      if(p == amount)
        amount = 0.0;
    }
    virtual void print() const {                                     // 2.0 points
      cout << "Online bill" << endl;
      printBillDate();
      cout << "You owe $" << amount << endl << endl;
    }

  private:
    float amount;                                                    // 0.5 points

};

#endif /* ONLINEBILL_H */
```

Grading scheme:

(a) Give points according to the comment on each line of the program.

(b) Correct inheritance 0.5 points as being shown on the comment line.

(c) For const functions, 0.25 points for the keyword 'const'.

(d) 0.25 points are deducted if the data members are not under the 'private' section

---------- END OF PAPER ----------