COMP 2012 Midterm Exam - Spring 2016 - HKUST

Date: April 2, 2016 (Saturday)
Time Allowed: 2 hours, 2–4pm
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **9** questions on **24** pages (including this cover page).
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, you are **NOT** allowed to define additional helper functions or structures, nor global variables unless otherwise stated. You also **cannot** use any library functions not mentioned in the questions.

Student Name	Model Answer
Student ID	00000000
Email Address	—
Lecture & Lab Section	—

For T.A. Use Only

Problem	Score
1	/ 10
2	/ 8
3	/ 5
4	/ 7
5	/ 9
6	/ 8
7	/ 9
8	/ 22
9	/ 22
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) Both class member functions and global functions can be inline functions.

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

T **F** (c) Slicing works for public, protected and private inheritance. That is, you may always assign a derived class object to a base class object regardless of whether the derived class is derived from the base class using public, protected and private inheritance.

T **F** (d) The prototype of the copy constructor for a class X generally is

```
X::X(const X& obj_to_copy);
```

Actually the following form is also fine:

```
X::X(X obj_to_copy);
```

T **F** (e) A class with virtual destructor is intended to be used as a base class.

T **F** (f) In the following class, author will be constructed before title:

```
#include "Title.h"
#include "Author.h"
class Book
{
  public:
    Book() : author("J.K.Rowling"), title("Harry Potter") { }
  private:
    Title title;
    Author author;
};
```

T **F** (g) There is no dynamic binding for virtual functions in constructors or destructors. (We are sorry that the question is not well written because while there is no dynamic binding

for virtual functions in constructors, there is in destructors. So the question is re-graded and everyone will get a point for this question regardless of your answer.)

T **F** (h) Passing a data member as a parameter in the member initialization list is dangerous because the data member will not have been properly initialized. For instance, the constructor `War_Game` in the following is potentially error-prone:

```
#include "Game.h"
#include "Player.h"
class War_Game : public Game
{
  public:
    War_Game( ) : Game(player) {}
  protected:
    Player player;
};
```

T **F** (i) A global function that takes an object of type A can be called on an object of type B as long as A has a conversion constructor that can convert a B object to an A object. One example is shown below and it is expected to run with no error.

```
/* Assume class B has been defined here */
class A
{
  public:
    A( ) { }
    A(const B& b) { }
};

void do_something(A a) { }

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

T **F** (j) Since one cannot create objects out of an abstract base class, it cannot have constructor(s) and destructor.

Problem 2 [8 points] Constness

In the following program, for the 16 statements ending with the following comment:

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

decide whether the statement is syntactically INCORRECT — that is, it will produce compilation error(s). Circle "Yes" if it will give compilation error and "No" otherwise.

You get 0.5 point for each correct answer, −0.25 for each wrong answer, and 0.0 if you do not answer by circling "Don't know".

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

class Anything {
  public:
    void set(int k) { data = k; }
    int get( ) const { return data; }
  private:
    int data;
};

void print1(Anything& a) { cout << a.get( ) << endl; }
void print2(const Anything& a) { cout << a.get( ) << endl; }

int main( ) {
    Anything x;
    Anything *const cpx = &x;                          /* Error: No */
    const Anything* pcx = &x;                          /* Error: No */
    const Anything *const cpcx = &x;                   /* Error: No */
    cpx->set(1);                                       /* Error: No */
    pcx->set(2);                                       /* Error: Yes */
    cpcx->set(3);                                      /* Error: Yes */
    print1(x);                                         /* Error: No */
    print2(x);                                         /* Error: No */

    const Anything y;   // Not tested; actually an error! const object requires explicit initializer
    Anything *const cpy = &y;                          /* Error: Yes */
    const Anything* pcy = &y;                          /* Error: No */
    const Anything *const cpcy = &y;                   /* Error: No */
    cpy->set(1);                                       /* Error: No */
    pcy->set(2);                                       /* Error: Yes */
    cpcy->set(3);                                      /* Error: Yes */
    print1(y);                                         /* Error: Yes */
    print2(y);                                         /* Error: No */
}
```

Problem 3 [5 points] Member Initialization

The following classes miss constructors. Write one constructor (which might not be unique) for each class so that the classes can be compiled without errors.

(a) [3 points]

```
class Base
{
  private:
    int b;
    int& ref;
  public:
    // Write your constructor in the space provided below
};
```

```
Base() : ref(b) { }
```

Answer: _____

Grading scheme: No partial credits for this question.
The constructor may or may not initialize member b.

(b) [2 points]

```
class Derived : public Base
{
  private:
    const int MAX;
  public:
    // Write your constructor in the space provided below
};
```

```
Derived() : MAX(10) { }
```

Answer: _____

Grading scheme: No partial credits for this question.

Problem 4 [7 points] Order of Construction & Destruction

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

class Wrapper
{
  private:
    int value;
  public:
    Wrapper(int v) : value(v) { cout << "Wrapper:   " << value << endl; }
    ~Wrapper() { cout << "~Wrapper:   " << value << endl; }
    int get( ) const { return value; }
};

class Info
{
  private:
    Wrapper data;
  public:
    Info(int a) : data(a) { cout << "New Info" << endl; }
    Info(const Info& x) : data(x.data.get( ) + 10) { cout << "Info:   Copy" << endl; }
    ~Info( ) { cout << "~Info" << endl; }
};

void tricky(Info oops)
{
    Info q = oops;
    cout << "After tricky" << endl;
}

int main( )
{
    Info x(1);
    cout << "Call tricky" << endl;
    tricky(x);
    return 0;
}
```

What is the output of the above program?

Wrapper: 1	construction of x(1)
New Info	
Call tricky	
Wrapper: 11	copy construction of oops during CBV
Info: Copy	
Wrapper: 21	copy construction of q from oops
Info: Copy	
After tricky	
~Info	destruction of q
~Wrapper: 21	
~Info	destruction of oops
~Wrapper: 11	
~Info	destruction of x
~Wrapper: 1	

Answer: _____

Grading scheme:

- 1 point for the outputs from the construction or destruction of an object ⇒ total 6 points.
- Add another point — i.e., giving full mark — if outputs are completely correct.
- No points for the output of the 2 lines:"Call trickly" and "After tricky".
- If the outputs are not totally correct, partial credits are given but the order of construction/destruction of various objects must be kept.

Problem 5 [9 points] Type and Order of Construction & Destruction

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

class A
{
  private:
    int x;
  public:
    A( ) : x(0) { cout << "Default:   " << x << endl; }

    A(int n) : x(n) { cout << "Conversion:   " << x << endl; }

    A(const A& a) : x(a.x) { cout << "Copy:   " << x << endl; }

    ~A( ) { cout << "~A:  " << x << endl; }
};

A fun(A& a)
{
    cout << "Fun!" << endl;

    {
        A obj;
    }

    return a;
}

int main( )
{
    A a(10);
    A b = a;
    A c[3];

    c[2] = fun(c[1]);
    return 0;
}
```

What is the output of the above program?

Conversion: 10	construction of a
Copy: 10	construction of b
Default: 0	construction of c[0]
Default: 0	construction of c[1]
Default: 0	construction of c[2]
Fun!	
Default: 0	construction of obj
~A: 0	destruction of obj
Copy: 0	construction of a temp object during RBV
~A: 0	destruction of the temp object created by RBV
~A: 0	destruction of c[2]
~A: 0	destruction of c[0]
~A: 0	destruction of c[1]
~A: 10	destruction of b
~A: 10	destruction of a

Answer: _____

Grading scheme:

- There are 5 objects to construct/destruct: a, b, c, obj, RBV.
- Points are given as: [2, 2, 2, 2, 1] for [a, b, c, obj, RBV].
- No points for the output of the line:"Fun!"'.
- If the outputs are not totally correct, partial credits are given but the order of construction/destruction of various objects must be kept.

Problem 6 [8 points] Type of Inheritance

```
#include <iostream>                                    /* File: inheritance-type.cpp */
using namespace std;

class A
{
  public:
    A(int k = 5) : data(5) { }
    int set(int k) { return (data = k); }
    int get( ) const { return data; }
    void print( ) const { cout << data << endl; }
  private:
    int data;
};

class B: public A                                                    // Line #15
{
  public:
    int scale(int s) { return set(s * get( )); }
};

class C: public B
{
  public:
    int add(int j) { return set(j + get( )); }
};

void output(const A& a) { a.print( ); }

int main( )
{
    C x;
    x.scale(2);
    x.add(3);
    output(x);
    return 0;
}
```

Given the above program, answer the following questions.

(a) [2 points] What is the output of the program?

13

Answer: _____

(b) [3 points] If line #15 is changed to

```
class B: protected A
```

will the program still compile with no error(s)? If yes, what is the output then? If not, which statement(s) will not compile and explain briefly why?

No, output(x) cannot compile because class C object cannot be cast to its protected base class A object.

Answer: _____

Grading scheme: no partial credits.

(c) [3 points] If line #15 is changed to

```
class B: private A
```

will the program still compile with no error(s)? If yes, what is the output then? If not, which statement(s) will not compile and explain briefly why?

No, the add( ) function of class C can't compile because now the set( ) and get( ) function of class A are private to C. Also the output(x) cannot compile again because class C object cannot be cast to its private base class A object.

Answer: _____

Grading scheme:
1.5 points for the reason that add( ) can't compile;
1.5 points for the reason that output( ) can't compile.

Problem 7 [9 points] Polymorphism and Dynamic Binding

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

class Shape
{
  public:
    Shape( ) { cout << "New Shape" << endl; }
    virtual ~Shape( ) { cout << "~Shape" << endl; }

    virtual Shape* duplicate( ) const { return new Shape; }
    virtual void print( ) const { cout << "Simple Shape" << endl; }
 };

class Box : public Shape
{
  public:
    Box( ) { cout << "New Box" << endl; }
    virtual ~Box( ) { cout << "~Box" << endl; }

    virtual Shape* duplicate( ) const { return new Box; }
    virtual void print( ) const { cout << "Simple Box" << endl; }
};


int main( )
{
    Shape* shape = new Box;
    Box* box = dynamic_cast<Box*>(shape->duplicate( ));

    shape->print( );
    delete shape;
    delete box;
    return 0;
}
```

What is the output of the above program?

```
New Shape
New Box
New Shape
New Box
Simple Box
~Box
~Shape
~Box
~Shape
```

Answer: _____

Grading scheme: 1 point for each output line.

Problem 8 [22 points] Polymorphism and Dynamic Binding

This question deals with a simple course management system for HKUST students and you are asked to design C++ classes for various types of courses that satisfy the following description and requirements. Firstly, all HKUST courses have lecture session. However, some courses also have lab session, while some other courses have tutorial session.

In your design, you only need to store and print the schedule of each course in addition to its code. All data members should be of the type `string` (from the C++ string class). Moreover, all member functions should be implemented inside the class definitions as inline member functions.

That is, you are to design the following three classes:

- `Course`: The base class for all courses, and all courses have lectures.
  - It stores and prints the lecture schedule.
- `CourseL`: The class for courses with only lecture and lab.
  - It should store and print both the lecture and lab schedules.
- `CourseT`: The class for courses with only lecture and tutorial.
  - It should store and print both the lecture and tutorial schedules.

You should implement a C++ class for each of the three types of courses by applying the concepts of class inheritance, polymorphism, and dynamic binding so that the following driver program will compile, run, and produce the output below.

```
#include "Course.h"                          /* File: course-management-sys.cpp */
#include "CourseL.h"
#include "CourseT.h"

int main( )
{
    CourseL comp3041("COMP3041", "TuTh", "Mon");
    CourseT comp3711("COMP3711", "TuTh", "Wed");

    Course* courses[2];
    courses[0] = &comp3041;
    courses[1] = &comp3711;

    for(int i = 0 ; i < 2 ; ++i)
        courses[i]->print( );
    return 0;
}
```

And the output of the program should be:

```
The schedule of COMP3041:
Lecture time is TuTh
Lab time is Mon
The schedule of COMP3711:
Lecture time is TuTh
Tutorial time is Wed
```

(a) [8 points] Implement the class `Course` which will be saved in the "Course.h" file.

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

```
// In general, 0.5 per each type of syntax error,
// and max 2 points for each question
#ifndef COURSE_H                    // 0.5 point for #ifndef pre-processor directive
#define COURSE_H

#include <iostream>                                          // 0.5 point
#include <string>                                  // Optional; no points
using namespace std;                                         // 0.5 point

class Course
{
  public:
    Course(string c, string lecture_s)     // 2 points, 1 for each member initialization
        : code(c), lect_schedule(lecture_s) { }

    virtual void print( )                                  // 0.5 point for virtual
    {
        cout << "The schedule of " << code << ":" << endl;          // 1 point
        cout << "Lecture time is " << lect_schedule << endl;        // 1 point
    }

  private:
    string code;                                                   // 1 point
    string lect_schedule;                                          // 1 point
};

#endif                                                       // COURSE_H
```

(b) [6 points] Implement the class `CourseL` which will be saved in the "CourseL.h" file.

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

```
#ifndef COURSEL_H                                                    // Optional
#define COURSEL_H

#include "Course.h"                                                  // 0.5 point

class CourseL : public Course                                        // 0.5 point
{
  public:
    CourseL(string c, string lecture_s, string lab_s)                // 2 points
        : Course(c, lecture_s), lab_schedule(lab_s) { }

    virtual void print( )                                // virtual is optional
    {
        Course::print( );                                            // 1 point
        cout << "Lab time is " << lab_schedule << endl;              // 1 point
    }

  private:
    string lab_schedule;                                             // 1 point
};

#endif                                                               // COURSEL_H
```

(c) [6 points] Implement the class `CourseT` which will be saved in the "CourseT.h" file.

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

```
#ifndef COURSET_H                                              // Optional
#define COURSET_H

#include "Course.h"                                            // 0.5 point

class CourseT : public Course                                  // 0.5 point
{
  public:
    CourseT(string c, string lecture_s, string tutorial_s)     // 2 points
        : Course(c, lecture_s), t_schedule(tutorial_s) { }

    virtual void print( )                                      // virtual is optional
    {
        Course::print( );                                      // 1 point
        cout << "Tutorial time is " << t_schedule << endl;     // 1 point
    }

  private:
    string t_schedule;                                         // 1 point
};

#endif                                                         // CourseT.h
```

(d) [2 points] Now there are some new courses that have both lab and tutorial sessions. How would you design a new C++ class, say, `CourseLT` for such new courses, and add it to the management system so that you may best re-use the available code (you have written above)? You only need to sketch your solution in not more than 60 words. YOU DO NOT NEED TO WRITE ANY C++ CODE NOR PSEUDO-CODE.

Answer:

The new class `CourseLT` may be defined using multiple inheritance so that it inherits the lab and tutorial information from the two existing classes: CourseL and CourseT. This will best re-use the codes of the 2 classes which are already written above.

Grading scheme: full 2 points for multiple inheritance; 1 point for using has relationship.

Problem 9 [22 points] Abstract Base Class and Inheritance

This problem involves an abstract base class called '`Question`' and two classes '`TF_Question`' and '`MC_Question`' which are derived from '`Question`' by public inheritance. Below are the header files of all the 3 classes.

```
/* File: question.h */
#ifndef QUESTION_H
#define QUESTION_H

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

class Question
{
  public:
    Question(string s) : text(s) { }
    virtual ~Question( ) { }
    virtual void display( ) const { cout << text << endl; }

    // The function cout a prompt that depends on the actual question type,
    // and the user has to input an answer via cin
    virtual bool check_answer( ) const = 0;

  private:
    string text;                                    // Question text
};

#endif                                              // QUESTION_H
```

```
/* File: tf-question.h */
#ifndef TF_QUESTION_H
#define TF_QUESTION_H
#include "question.h"

class TF_Question: public Question
{
  public:
    TF_Question(string s, bool a);                    // s is the text, a is the answer
    ~TF_Question( );

    virtual void display( ) const;
    virtual bool check_answer( ) const;

  private:
    bool answer;
};
#endif                                                // TF_QUESTION_H

/* File: mc-question.h */
#ifndef MC_QUESTION_H
#define MC_QUESTION_H
#include "tf-question.h"

class MC_Question: public Question
{
  public:
    MC_Question(string s, char a);                    // s is the text, a is the answer
    ~MC_Question( );

    virtual void display( ) const;
    virtual bool check_answer( ) const;

  private:
    char answer;                                      // 'A', 'B', 'C' or 'D'
    TF_Question* choice[4];                           // Always assume 4 choices only
    TF_Question* add_choice(int j);                   // j=0 => choice 'A', j=1 => 'B', etc.
};
#endif                                                // MC_QUESTION_H
```

Notice that

- for all parts of this problem, you may assume that the user will always enter correct information in the correct format, and you don't need to check for invalid inputs.
- the implementation of class `Question` is complete in its header file, and thus, there is no need for a "`question.cpp`" file.
- during the call of `check_answer( )`, users will be prompted for an answer to the question and the prompt differs for different types of questions.
- each `MC_Question` object always *owns* 4 choices which are represented by `TF_Question` objects. The choices are added during the construction of an `MC_Question` object by calling its private `add_choice( )` member function. You may assume that the choices are always added with answers matching the correct answer given to the constructor of the `MC_Question` object.

Below is the testing program "question-test.cpp" for the 3 classes, and it is compiled by the following command to produce the executable program called "`question-test`".

```
gcc -o question-test question-test.cpp tf-question.cpp mc-question.cpp
```

```
/* File: question-test.cpp */
#include "question.h"
#include "tf-question.h"
#include "mc-question.h"

int main( )
{
    Question* q[2];

    q[0] = new TF_Question("Are you happy with COMP2012?", true);
    q[1] = new MC_Question("Among the following, who is a Chinese?", 'C');
    cout << endl;

    for (int j = 0; j < 2; ++j)
    {
        cout << "QUESTION " << j+1 << endl << "----------" << endl;
        q[j]->display();
        cout << (q[j]->check_answer( ) ? "Correct\n\n" : "Incorrect\n\n");
    }

    return 0;
}
```

A sample run is given as follows:

```
Enter choices for the MC question:
Among the following, who is a Chinese?
Enter choice A followed by the answer (1 for yes, 0 for no): Mozart 0
Enter choice B followed by the answer (1 for yes, 0 for no): Newton 0
Enter choice C followed by the answer (1 for yes, 0 for no): Confucius 1
Enter choice D followed by the answer (1 for yes, 0 for no): Picasso 0

QUESTION 1
----------
This is a True/False Question: Are you happy with COMP2012?
Answer "yes" or "no": yes
Correct

QUESTION 2
----------
This is a Multiple-Choice Question:
(A) Mozart
(B) Newton
(C) Confucius
(D) Picasso
Answer 'A', 'B', 'C' or 'D': B
Incorrect
```

Based on the given information, complete the implementation of `TF_Question` class and `MC_Question` class in their respective .cpp files, namely, '`tf-question.cpp`' and '`mc-question.cpp`' respectively.

(a) [6 points] Implement all member functions of the class `TF_Question` in a separate file called "tf-question.cpp".

Answer: `/* File "tf-question.cpp" */`

```
/* No points for member function prototypes as they are all given */

#include "tf-question.h"                                          // 0.5 point

TF_Question::TF_Question(string s, bool a)
        : Question(s), answer(a) { }   // 1 + 0.5 points

TF_Question::~TF_Question( ) { }   // 0.5 point

void TF_Question::display( ) const
{
    cout << "This is a True/False Question:  ";                   // 0.5 point
    Question::display();                                          // 1 point
}

bool TF_Question::check_answer( ) const
{
    string input;
    cout << "Answer \"yes\" or \"no\":  ";                        // 0.5 point
    cin >> input;                                                 // 0.5 point
    return (((input == "yes") ? true : false) == answer);         // 1 point
}
```

(b) [16 points] Implement all member functions of the class `MC_Question` in a separate file called "mc-question.cpp".

Answer: `/* File "mc-question.cpp" */`

```
#include "mc-question.h"                                          // 0.5 point

MC_Question::MC_Question(string s, char a)                        // (total 4 points)
        : Question(s), answer(a)                                  // 1.5 points
{
    // 0.5 point
    cout << "Enter choices for the MC question:  " << endl << s << endl;

    for (int j = 0; j < 4; ++j)                                   // 2 points
        choice[j] = add_choice(j);
}

MC_Question::~MC_Question( )                                      // (total 2 points)
{
    for (int j = 0; j < 4; ++j)                                   // 2 points
        delete choice[j];
}

void MC_Question::display( ) const                                // (total 3.5 points)
{
    cout << "This is a Multiple-Choice Question:  " << endl;      // 0.5 point

    for (int j = 0; j < 4; ++j)
    {
        cout << "(" << static_cast<char>('A' + j) << ") ";        // 1 point
        choice[j]->Question::display();                           // 2 points
    }
}

bool MC_Question::check_answer( ) const                           // (total 2 points)
{
    char input;
    cout << "Answer 'A', 'B', 'C' or 'D': ";                      // 0.5 point
    cin >> input;                                                 // 0.5 point
    return (input == answer);                                     // 1 point
}
```

```
TF_Question* MC_Question::add_choice(int j)                    // (total 4 points)
{
    string statement;
    int ans;

    cout << "Enter choice " << static_cast<char>('A' + j)          // 1 point
         << " followed by the answer (1 for yes, 0 for no):  ";
    cin >> statement >> ans;                                      // 0.5 point

    return new TF_Question(statement, ans);                       // 2.5 points
}
```