Paper I

COMP 2012 Midterm Exam - Fall 2018 - HKUST

Date: October 27, 2018 (Saturday)
Time Allowed: 2 hours, 2pm–4pm
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **7** questions on **28** pages (including this cover page and 3 blank pages at the end) printed on **2** sets of papers:
   - Paper I: Description of problem 1 - 4 and space for ALL your answers.
   - Paper II: Description of problem 5 - 7 (Problem description only).
3. Write your answers in the space provided in paper 1.
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, classes, helper functions and use global variables, nor any library functions not mentioned in the questions.

Student Name	
Student ID	
Email Address	
Venue	
Class Number	

For T.A. Use Only

Problem	Topic	**Score**
1	True or False	/ 10
2	Const-ness	/ 11
3	Order of Construction and Destruction	/ 9
4	Debugging	/ 6
5	Classes and Objects	/ 21
6	Class Template	/ 20
7	Operator Overloading	/ 23
Total		/ 100

Problem 1 [10 points] True or false

Indicate whether the following statements are *true* or *false* by circling **T** or **F**.

T **F** (a) There is NO compilation error in the following program.

```
class A {
    int _a;
  public:
    explicit A(int a) {
      _a = a;
    }
};

int main() {
  A obj = 1;
}
```

T **F** (b) The statement `A obj = A(1);` in the following program constructs an object `obj` assigning 1 to its data member `a`.

```
class A {
    int a;
  public:
    A(int a) {
      this->a = a;
    }
};

int main() {
  A obj = A(1);
}
```

T **F** (c) The conversion constructor, `A(int a)`, correctly initializes the data member `a` of object `obj` with the specified parameter `a`.

```
class A {
    int a;
  public:
    A(int a) : a(a) { }
};

int main() {
  A obj(10);
}
```

T **F** (d) The default copy constructor provided by the compiler does NOT perform correct copying of integer values in `arr` for the following class.

```
class A {
  int arr[10];
};
```

T **F** (e) If a class requires a destructor to de-allocate some resources, then it almost certainly needs a user-defined copy constructor and assignment operator function as well.

T **F** (f) The following program has an invalid operation that causes runtime error.

```
class A {
    int* p;
  public:
    A() { p = new int; }
    ~A() { delete p; }
};

int main() {
  A obj1, obj2;
  obj1 = obj2;
}
```

T **F** (g) There is NO compilation error in the following program.

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

template <typename T, int a>
int func(const T& b) {
  cout << "Enter number of " << b << "in the list: ";
  cin >> a;
  return a;
}

int main() {
  int c = 10;
  cout << "Number of 10 in the list is: " << func<int, 10>(c) << endl;
}
```

T **F** (h) If a binary operator is overloaded using a non-member function, then two parameters are required.

T **F** (i) The post-increment operator function (`operator++`) of class A can be defined as follows:

```
class A {
    int a;
  public:
    A() {
      a = 10;
    }
    A operator++(double) {
      int temp = a;
      a++;
      return a;
    }
};
```

T **F** (j) There is NO compilation error in the following program.

```
class Vector {
    int x, y;
  public:
    Vector(int x, int y) {
      this->x = x;
      this->y = y;
    }
    void operator=(const Vector& v) {
      x = v.x;
      y = v.y;
    }
};

int main() {
  Vector a(1,2), b(3,4), c(5,6);
  a = b = c;
}
```

Problem 2 [11 points] Const-ness

In the following program, for the 11 statements ending with the following comment:

```
/* Error:   Yes   /   No   */
```

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.

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

class A
{
  public:
    void nonConstMemFunc() { }
    void constMemFunc() const { }
    const A getConstObj() { return *this; }
    const A& getConstRefObj() { return *this; }
};

int main()
{
  const A constObj;

  constObj.nonConstMemFunc();                          /* Error:   Yes   /   No   */

  constObj.constMemFunc();                             /* Error:   Yes   /   No   */

  A nonConstObj;

  A nonConstObj2 = nonConstObj.getConstObj();          /* Error:   Yes   /   No   */

  A nonConstObj3 = nonConstObj.getConstRefObj();       /* Error:   Yes   /   No   */

  A& nonConstRefObj1 = nonConstObj.getConstObj();      /* Error:   Yes   /   No   */

  A& nonConstRefObj2 = nonConstObj.getConstRefObj();   /* Error:   Yes   /   No   */

  const A constObj2 = nonConstObj.getConstObj();       /* Error:   Yes   /   No   */

  const A constObj3 = nonConstObj.getConstRefObj();    /* Error:   Yes   /   No   */

  const A& constRefObj1 = nonConstObj.getConstObj();   /* Error:   Yes   /   No   */

  const A& constRefObj2 = nonConstObj.getConstRefObj(); /* Error:   Yes   /   No   */

  A& const mystery = nonConstObj.getConstObj();        /* Error:   Yes   /   No   */

  return 0;
}
```

Problem 3 [9 points] Order of Construction and Destruction

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

class Table {
  public:
    Table() { cout << "T" << endl; }
    ~Table() { cout << "~T" << endl; }
};

class Chair {
  public:
    Chair() { cout << "C" << endl; }
    ~Chair() { cout << "~C" << endl; }
};

class Whiteboard {
  public:
    Whiteboard() { cout << "W" << endl; }
    ~Whiteboard() { cout << "~W" << endl; }
};

class Room {
    Table* tables;
    Chair* chairs;
    Whiteboard* whiteboards;
    int numT, numC, numW;
  public:
    Room(int numT, int numC, int numW) : whiteboards(new Whiteboard[numW]),
                          chairs(new Chair[numC]), tables(new Table[numT]) {
      this->numT = numT;  this->numC = numC;  this->numW = numW;
      cout << "R Other" << endl;
    }
    Room(const Room& r) {
      tables = new Table[r.numT];   this->numT = r.numT;
      chairs = new Chair[r.numC];   this->numC = r.numC;
      whiteboards = new Whiteboard[r.numW]; this->numW = r.numW;
      cout << "R Copy" << endl;
    }
    ~Room() {
      delete [] tables;
      delete [] chairs;
      delete [] whiteboards;
      cout << "~R" << endl;
    }
};

int main() { Room miniRoom = *(new Room(1, 2, 1)); }
```

(a) [7.5 points] Write down the output of the above program when it is run.

Answer:

(b) [1.5 points] Is there memory leak problem in the above program? If so, identify the problem by writing down the line number where it occurs.

Answer:

Problem 4 [6 points] Debugging

The following program contains 4 errors. Identify each error by writing down the line number where it occurs, and explain why it is an error. In identifying the errors, please consider them independently, assume that the other errors have been fixed and do not exist.

```
1  #include <iostream>
2  using namespace std;
3
4  class Person {
5    private:
6      string name;
7      int age;
8    public:
9      Person(string n, int a) : name(n), age(a) {}
10 };
11
12 template <typename T> class Container {
13   private:
14     T* data;
15     int size;
16   public:
17     Container(int size = 10) : data(new T[size]) { }
18     Container(const Container& c) : size(c.size), data(new T[c.size]) { }
19     ~Container() { delete [] data; }
20     T& operator[](int i) { return data[i]; }
21     bool operator==(const Container& c);
22
23     friend ostream& operator<<(ostream& os, const Container<T>& c);
24 };
25
26 bool Container::operator==(const Container& c) {
27   if(size != c.size)
28     return false;
29   else {
30     for(int i=0; i<size; i++) {
31       if(data[i] != c.data[i])
32         return false;
33     }
34     return true;
35   }
36 }
37
38 template <typename T>
39 ostream& operator<<(ostream& os, const Container<T>& c) {
40   for(int i=0; i<c.size; i++)
41     cout << c.data[i] << " ";
42 }
```

```
43
44 int main() {
45   Container<Person> pContainer1(2);
46   pContainer1[0] = Person("Peter", 18);
47   pContainer1[1] = Person("John", 20);
48
49   Container<Person> pContainer2(2);
50   pContainer2[0] = Person("Jason", 21);
51   pContainer2[1] = Person("Jackson", 22);
52
53   cout << "Two person containers are ";
54   cout << (pContainer1 == pContainer2 ? "are the same" : "are different") << endl;
55
56   return 0;
57 }
```

Answer:

Error#	Line#	Explanation
1		
2		
3		
4		

Problem 5 [21 points] Classes and Objects

Please refer to Paper II for the problem description.

(a) [1 point] In "`Dog.h`", add a line at the indicated position to allow `Human` access any private data members of `Dog` freely.

Answer:

(b) [6 points] Write "`Dog.cpp`" below. It should contain all implementations of `Dog` according to the header file.

Answer:

(c) [14 points] Write "`Human.cpp`" below. It should contain all implementations of `Human` according to the header file.

Answer:

Problem 6 [20 points] Class Template

Please refer to Paper II for the problem description.

Answer:

Answer (Cont'd)

Problem 7 [23 points] Operator Overloading

Please refer to Paper II for the problem description.

(a) [3 points] Complete the missing parts in the space provided under Part(a)(i)-(a)(iii) "`ADD YOUR CODE HERE`" by implementing (1) the constructor to initialize the data member `staff` according to the parameter, (2) the accessor member function `getStaff()`, and (3) mutator member function `setStaff(const Staff& s)`.

```
class CarPickup {  /* Filename: Carpickup.h */
  private:
    Staff& staff;  // Staff reference variable

  public:
    CarPickup(Staff& s);            // Constructor
    Staff& getStaff();              // Accessor function
    void setStaff(const Staff& s);  // Mutator function
};

// Implement the constructor CarPickup(Staff& s) here
// Part (a)(i) - ADD YOUR CODE HERE









// Implemnet the accessor member function "Staff& getStaff()" here
// Part (a)(ii) - ADD YOUR CODE HERE









// Implement the mutator member function "void setStaff(const Staff& s)" here
// Part (a)(iii) - ADD YOUR CODE HERE
```

(b) [20 points] Complete the missing parts in the space provided under Part(b)(i)-(b)(vi) "`ADD YOUR CODE HERE`" by implementing

(1) the constructor of `Shop`

(2) the member function "`Shop& operator<<(CarPickup& c)`"

(3) the member function "`Shop& operator<<(Customer& c)`"

(4) the member function "`Shop& operator>>(CarPickup& c)`"

(5) the member function "`Shop& operator>>(Customer& c)`"

(6) the non-member insertion operator function so that `Shop` object can be printed using `operator<<`.

```
#include <iostream>  /* Filename: Shop.h */
#include "Queue.h"   /**** Template class defined in Question 6 ****/
#include "Staff.h"
#include "Customer.h"
#include "Carpickup.h"
using namespace std;

class Shop {
  private:
    string name;
    const int size;               // Size of shop in square feet
    Queue<Staff> staffList;       // A queue of staff (first-come-first-served)
    Queue<Customer> customerList; // A queue of customers (first-come-first-served)

  public:
    // Constructor of Shop
    // Initialize all the data members and construct both queues of maxSize 10.
    Shop(string n, int s);

    // Make the staff in the Carpickup object on-duty by
    // enqueing the staff to the back of staffList.
    Shop& operator<<(CarPickup& c);

    // Add a customer to the queue of the shop by
    // enqueing the customer to the back of customerList.
    Shop& operator<<(Customer& c);

    // Make the staff member off-duty by doing the following:
    // - Check if staffList is empty. If not,
    //    1. Get the staff object at the front of the staffList.
    //    2. Assign the staff object to the staff object in the CarPickup object c
    //       AND remove the staff object from the front of staffList.
    //   Otherwise,
    //    1. Create a Staff object with name data member "Dummy", and
    //    2. Assign it to the staff object in the CarPickup object c
    Shop& operator>>(CarPickup& c);
```

```
    // Serve a customer, remove the customer from the front of customerList
    // by doing the following:
    // - Check if customerList is empty. If not,
    //   1. Get the customer object at the front of the customerList.
    //   2. Assign the customer object to the customer object c
    //      AND remove the customer object from the front of customerList.
    //   Otherwise,
    //   1. Create a Customer object with name data member "Dummy", and
    //   2. Assign it to customer object c.
    Shop& operator>>(Customer& c);

    // Friend declaration
    // Implementation details:
    // - The output format of a Shop object is as follows:
    //    Name: HKUST Bakery
    //    Size (in square feet): 50
    //    Staff: Wallace Luke
    //    Customers: John Peter
    // - Hint:
    //    For staffList and customerList, as they queue containers, only the
    //    element at the front of the list can be peeked, so you need to make
    //    clever use of queue operations to make the printing of Staff and
    //    Customer objects possible.
    friend ostream& operator<<(ostream& os, const Shop& s);
};

// Implement the constructor of Shop here
// Part (d)(i) - ADD YOUR CODE HERE









// Implement the member function "Shop& operator<<(CarPickup& c)" here
// Part (d)(ii) - ADD YOUR CODE HERE
```

```
// Implement the member function "Shop& operator<<(Customer& c)" here
// Part (d)(iii) - ADD YOUR CODE HERE
```

```
// Implement the member function "Shop& operator>>(CarPickup& c)" here
// Part (d)(iv) - ADD YOUR CODE HERE
```

```
// Implement the member function "Shop& operator>>(Customer& c)" here
// Part (d)(v) - ADD YOUR CODE HERE
```

```
// Implement the non-member function operator<< for shop here.
// Part (d)(vi) - ADD YOUR CODE HERE
```

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