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	SOLUTIONS AND MARKING SCHEME
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 user-defined 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;
}
```

Marking scheme:

- 1 point for each correct answer.

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;
}
```

Answer:

```
#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

  constObj.constMemFunc();                                 // Error: No

  A nonConstObj;

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

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

  A& nonConstRefObj1 = nonConstObj.getConstObj();          // Error: Yes

  A& nonConstRefObj2 = nonConstObj.getConstRefObj();       // Error: Yes

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

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

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

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

  A& const mystery = nonConstObj.getConstObj();            // Error: Yes

  return 0;
}
```

Marking scheme:

- 1 point for each correct answer.

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:

```
T
C
C
W
R Other
T
C
C
W
R Copy
~T
~C
~C
~W
~R
```

Marking scheme:

- 2.5 points for `T, C, C, W, R Other`
- 2.5 points for `T, C, C, W, R Copy`
- 2.5 points for `T, C, C, W, R`

(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:

Yes, there is memory leak problem in the above program. The problem occurs on line 47.

Marking scheme:

- 0.5 point for stating there is memory leak problem in the program.
- 1 point for giving correct line number.

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 ? "the same" : "different") << endl;
55
56   return 0;
57 }
```

Answer:

Error#	Line#	Explanation
1	17	No default constructor in Person class.

2	23	friend declaration problem of non-member function template operator<<.
		Fix:
		`template <typename S>`
		`friend ostream& operator<<(ostream& os, const Container<S>& c);`

3	26	As the member function operator== is defined outside the class, the function header should be
		`template <typename T>`
		`bool Container<T>::operator==(const Container& c) {`

4	31	Cannot compare two Person type objects. Since operator!= is not overloaded for Person class.

Marking scheme:

- 0.5 point for each correct line number.
- 1 point per correct explanation.

Problem 5 [21 points] Classes and Objects

This problem involves 2 classes called '`Dog`' and '`Human`'. Below are the header files of the 2 classes.

```
#ifndef DOG_H  /* Filename: Dog.h  */
#define DOG_H

#include <string>
using namespace std;

class Dog {
  // A line shall be added here to allow Human access any private data members
  // of Dog freely.

  private:
    string name;   // Name of the dog
    int weight;    // Weight of the dog
    int happiness; // Measures how happy the dog is, should be 100 initially

  public:
    // Constructor.
    // Initialize the Dog.
    Dog(string name, int weight);

    // Eat.
    // Increase the weight by 1.
    void eat();

    // Run.
    // Decrease the weight by half, round down to the nearest integer if needed.
    void run();

    // Lick another dog.
    // Increase the happiness of itself (this object) by 10.
    // Increase the happiness of anotherDog by 20.
    void lick(Dog& anotherDog);
};

#endif /* DOG_H */
```

```
#ifndef HUMAN_H  /* Filename: Human.h */
#define HUMAN_H

#include <iostream>
#include "Dog.h"

class Human {
  private:
    Dog** dogs; // It is a dynamic array of Dog pointers which point to Dog objects.
                // It should always be just big enough to hold all the dogs owned
                // by the human. e.g. if there are 5 dogs owned by the human,
                // the size of the dogs array should be exactly 5

    int dogCount; // The number of dogs the human owns,
                  // i.e., the size of the dogs array

  public:
    // Constructor.
    // Set dogs to NULL.
    // Set dogCount to 0.
    Human();

    // Destructor.
    // Perform memmory deallocation.
    // Make sure there is no memory leak.
    ~Human();

    // Adopt a dog.
    // Deep copy the given dog and add it to the end of the dogs array.
    // Make sure the array is just big enough to hold all dogs, as described.
    // Therefore, you need to increase the size of the array by 1.
    // If an existing dog already has the same name as the given dog, do nothing.
    void adoptDog(const Dog& dog);

    // Give a dog to someone.
    // Remove the dog of the given name from the dogs array.
    // Make sure the array is just big enough to hold all dogs, as described.
    // Therefore, you need to reduce the size of the array by 1.
    // Do nothing if no dog has the given name.
    void giveDog(string name);

    // Find the dog with the given name, and return it.
    // Return NULL if no dog has the given name.
    Dog* findDog(string name) const;

    // Pet the dog with the given name.
    // That means happiness of that dog will be increased by 30.
    // Do nothing if no dog has the given name.
    void petDog(string name);
```

```
    void showDogs() const {
      cout << "I love my dogs: ";
      for(int i=0;i<dogCount;i++)
        cout << dogs[i]->name << (i<dogCount-1?",":"\n");
    }
};

#endif /* HUMAN_H */
```

Your task is to implement the missing member functions in the two given classes '`Dog`' and '`Human`'. The following is the test program 'test-human-dog.cpp' for the 2 classes.

```
#include <iostream>  /* test-human-dog.cpp */
#include "Dog.h"
#include "Human.h"

using namespace std;

int main()
{
  Dog shiba("kuro", 20);
  Dog husky("terminator", 40);
  Dog hkOriginal("dimsum", 30);

  Human tina;

  tina.adoptDog(shiba);
  tina.adoptDog(husky);
  tina.adoptDog(hkOriginal);

  tina.showDogs();

  cout << "Gift terminator to my son!" << endl;
  tina.giveDog("terminator");

  tina.showDogs();

  return 0;
}
```

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

```
I love my dogs: kuro,terminator,dimsum
Gift terminator to my son!
I love my dogs: kuro,dimsum
```

(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:

```
friend class Human;                    // 1 point
```

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

Answer:

```
#include "Dog.h"

Dog::Dog(string name, int weight) {  // 1 point
  this->name = name;
  this->weight = weight;
  this->happiness = 100;
}

void Dog::eat() {                    // 1 point
  this->weight++;
}

void Dog::run() {                    // 1 point
  this->weight /= 2;
}

void Dog::lick(Dog& anotherDog) {    // 2 points
  this->happiness += 10;
  anotherDog.happiness += 20;
}

// 1 point for all corrrect headers
// -1 point for each logical error / mistake
// -0.5 for each syntax error
```

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

Answer:

```
#include "Human.h"

Human::Human() {                              // 1 point
  dogs = NULL;
  dogCount = 0;
}

Human::~Human() {                             // 2 points
  for(int i=0; i<dogCount; i++)
    delete dogs[i];
  delete [] dogs;
}

void Human::adoptDog(const Dog& dog) {        // 3 points
  Dog** n = new Dog*[dogCount+1];
  for(int i=0;i<dogCount;i++)
    n[i] = dogs[i];
  n[dogCount] = new Dog(dog);
  dogCount++;
  delete [] dogs;
  dogs = n;
}

void Human::giveDog(string name) {            // 3 points
  if(findDog(name) == NULL)
    return;

  Dog** n = new Dog*[dogCount-1];
  int j = 0;
  for(int i=0;i<dogCount;i++) {
    if(dogs[i]->name == name) {
      delete dogs[i];
    }
    else {
      n[j++] = dogs[i];
    }
  }
  dogCount--;
  delete [] dogs;
  dogs = n;
}
```

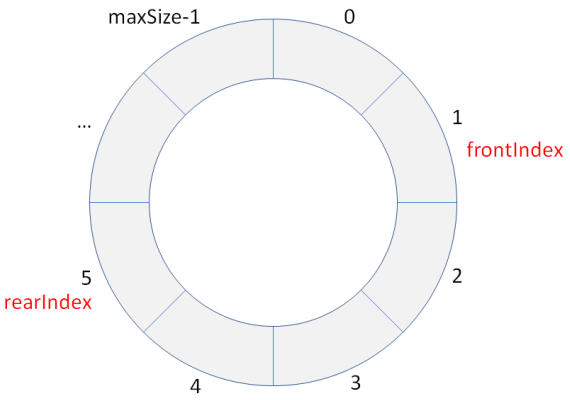
```
Dog* Human::findDog(string name) const { // 2 points
  for(int i=0;i<dogCount;i++) {
    if(dogs[i]->name == name) {
      return dogs[i];
    }
  }
  return NULL;
}

void Human::petDog(string name) {        // 2 points
  Dog* d = findDog(name);
  if(d == NULL)
    return;
  d->happiness += 30;
}

// 1 point for all correct headers
// -1 point for each logical error / mistake
// -0.5 for each syntax error
```

Problem 6 [20 points] Class Template

This problem involves the implementation of a template class `Queue` using a dynamic circular array. A queue is a data structure in which objects are added to the back (specified by `rearIndex`) of the queue, and removed from the front (specified by `frontIndex`) of the queue, and it enforces First-In-First-Out (FIFO) behaviour, while a circular array is an array with the end of the array wraps around to the start of the array. The following shows a queue implemented using a circular array.



Write a template class `Queue` that stores a list of objects using a dynamic circular array and provides the following data members and member functions:

- Data members
  - `maxSize`: a constant variable that represents the maximum size of the dynamic array.
  - `data`: a pointer that points at a dynamic array of type `T`, where `T` is a type name.
  - `frontIndex`: an `int` that represents the index of the front object in the queue.
  - `rearIndex`: an `int` that represents the index of the last object in the queue.
- Member functions
  - a constructor with an `int` parameter, `maxSize`, and it dynamically allocates an array of objects in type `T` according to the parameter `maxSize`. As the queue is initially empty, both `frontIndex` and `rearIndex` should be initialized to -1.
  - a copy constructor that performs deep copy.
  - a destructor that de-allocates the dynamic array.
  - a constant member function `isEmpty` that returns a `bool` value indicating whether the queue is empty.
  - a constant member function `isFull` that returns a `bool` value indicating whether the queue is full.
  - `front()` that returns a reference to the front object in the queue, without taking the front object out of the queue.

- `enqueue()` that accepts an object of type `T` using a constant reference parameter, and it inserts the object to the back of queue. It returns true if the object is successfully inserted to the queue. Otherwise, it returns false.
- `dequeue()` that removes the object at the front of the queue. It returns true if the object is successfully removed from the queue. Otherwise, it returns false.

For simplicity, implement all the member functions **INSIDE** the class template definition in a single file called "`Queue.h`". Note that your solution should work with the testing program "`test-queue.cpp`" and produce the following outputs:

```
Enqueue 20 and 30
Front element: 20
Dequeue: Success
Front element: 30
Dequeue: Success
The queue is empty
```

```cpp
#include <iostream>   /* Filename: test-queue.cpp */
#include "Queue.h"
using namespace std;

int main() {
  Queue<int> intQueue(10);  // A queue of size 10

  cout << "Enqueue 20 and 30" << endl;
  // Insert data to the back of the queue
  intQueue.enqueue(20);
  intQueue.enqueue(30);

  // Print data at the front of the queue
  if(!intQueue.isEmpty())
    cout << "Front element: " << intQueue.front() << endl;

  // Remove data from the front of the queue
  cout << "Dequeue: " << ((intQueue.dequeue()) ? "Success" : "Fail") << endl;
  // Print data at the front of the queue
  if(!intQueue.isEmpty())
    cout << "Front element: " << intQueue.front() << endl;

  // Remove data from the front of the queue
  cout << "Dequeue: " << ((intQueue.dequeue()) ? "Success" : "Fail") << endl;
  if(!intQueue.isEmpty())
    cout << "Front element: " << intQueue.front() << endl;
  else
    cout << "The queue is empty" << endl;
}
```

Answer:

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

template <typename T> class Queue {                                  // 0.5 point
  private:
    const int maxSize;                                               // 2 points
    T* data;
    int frontIndex;
    int rearIndex;

  public:
    Queue(int maxSize) : maxSize(maxSize) {                          // 1.5 points
      data = new T[maxSize];
      frontIndex = rearIndex = -1;
    }

    Queue(const Queue& q) : maxSize(q.maxSize) {                     // 3 points
      frontIndex = q.frontIndex;
      rearIndex = q.rearIndex;
      data = new T[maxSize];
      for(int i=0; i<maxSize; ++i)
        data[i] = q.data[i];
    }

    ~Queue() { delete [] data; }                                     // 0.5 point

    bool isEmpty() const {                                           // 1 point
      return (frontIndex == -1 && rearIndex == -1);
    }

    bool isFull() const {                                            // 2 points
      return (rearIndex + 1) % maxSize == frontIndex;
    }

    T& front() {                                                     // 0.5 point
      return data[frontIndex];
    }
```

```
    bool enqueue(const T& obj) {                                    // 5.5 points
      if(isFull())
        return false;
      else if(frontIndex == -1) {
        frontIndex = rearIndex = 0;
        data[rearIndex] = obj;
      }
      else if(rearIndex == maxSize - 1 &&
              frontIndex != 0) {
        rearIndex = 0;
        data[rearIndex] = obj;
      }
      else
        data[++rearIndex] = obj;
      return true;
    }

    bool dequeue() {
      if(isEmpty())                                                 // 3.5 points
        return false;
      if(frontIndex == rearIndex) {
        frontIndex = -1;
        rearIndex = -1;
      }
      else if(frontIndex == maxSize - 1)
        frontIndex = 0;
      else
        ++frontIndex;
      return true;
    }
};
```

Problem 7 [23 points] Operator Overloading

The following shows a mini application that involves 4 classes called '`Staff`', '`Customer`', '`CarPickup`', and '`Shop`'. The class '`Shop`' consists of two queue containers that are instantiated using the template class you have just defined in Question 6 to store `Staff` and `Customer` objects. Given the following complete definition and implementation of the first 2 classes, '`Staff`' and '`Customer`',

```
#include <iostream>  /* Filename: Staff.h */
#include <string>
using namespace std;

class Staff {
  private:
    string name; // Name of staff

  public:
    Staff(string n = "") { name = n; } // Default / Conversion constructor

    // Friend declaration
    friend class Shop;
    friend ostream& operator<<(ostream& os, const Staff& s) {
      os << s.name;
      return os;
    }
};
```

```
#include <iostream>  /* File: Customer.h */
#include <string>
using namespace std;

class Customer {
  private:
    string name; // Name of customer

  public:
    Customer(string n = "") { name = n; } // Default / Conversion constructor

    // Friend declaration
    friend class Shop;
    friend ostream& operator<<(ostream& os, const Customer& c) {
      os << c.name;
      return os;
    }
};
```

your task in this question is to complete all the missing parts in `CarPickup` and `Shop` classes so that the given testing program "`test-shop.cpp`" will compile, run, and produce the expected output.

```
#include <iostream>  /* Filename: test-shop.cpp */
#include "Shop.h"
using namespace std;

int main() {
  // Instantiate a shop object
  Shop shop("HKUST Bakery", 50);

  // Instantiate two staff objects
  Staff wallace("Wallace");
  Staff luke("Luke");

  // Instantiate two carpickup objects
  CarPickup car1(wallace);
  CarPickup car2(luke);
  shop << car1 << car2;   // Add two staff to the staffList of the shop

  // Instantiate two customer objects
  Customer john("John");
  Customer peter("Peter");
  shop << john << peter;  // Add two customers to the customerList of the shop

  cout << shop;           // Print the shop

  Staff staff;            // Instantiate a staff object

  // Instantiate a carpickup object
  CarPickup car3(staff);  // Instantiate a carpickup object
  shop >> car3;           // Pickup a staff who will be off-duty
  cout << "Off duty: " << staff << endl;  // Print the staff

  CarPickup car4(staff);  // Instantiate a carpickup object
  shop >> car4;           // Pickup a staff who will be off-duty
  cout << "Off duty: " << staff << endl;  // Print the staff

  Customer customer;      // Instantiate a customer object
  shop >> customer;       // Serve a customer, remove the customer from the queue
  cout << "Served: " << customer << endl; // Print the details of the served customer

  return 0;
}
```

Expected output of the testing program:

```
Name: HKUST Bakery
Size (in square feet): 50
Staff: Wallace Luke
Customers: John Peter
Off duty: Wallace
Off duty: Luke
Served: John
```

(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
```

Answer:

```
#include <string>  /* Filename: Carpickup.h */
using namespace std;

class CarPickup {
  private:
    Staff& staff;

  public:
    CarPickup(Staff& s);
    Staff& getStaff();
    void setStaff(const Staff& s);
};

CarPickup::CarPickup(Staff& s) : staff(s) { }             // 1 point
Staff& CarPickup::getStaff() { return staff; }            // 1 point
void CarPickup::setStaff(const Staff& s) { staff = s; }   // 1 point
```

(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 (b)(i) - ADD YOUR CODE HERE









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

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

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

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

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

Answer:

```
#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:
    Shop(string n, int s);

    // Get car pickup service for a staff of the shop
    Shop& operator<<(CarPickup& c);

    // Add a customer to the queue of the shop
    Shop& operator<<(Customer& c);

    // Staff member takes a car, removed the staff from front of the staffList
    Shop& operator>>(CarPickup& c);

    // Served a customer, removed the customer from the front of the customerList
    Shop& operator>>(Customer& c);

    // Friend declaration
    friend ostream& operator<<(ostream& os, const Shop& s);
};

Shop::Shop(string n, int s)                          // 2.5 points
     : staffList(10),
       customerList(10),
       size(s) {
  name = n;
}

Shop& Shop::operator<<(CarPickup& c) {               // 2 points
  staffList.enqueue(c.getStaff());
  return *this;
}
```

```
Shop& Shop::operator<<(Customer& c) {                    // 1.5 points
  customerList.enqueue(c);
  return *this;
}

Shop& Shop::operator>>(CarPickup& c) {                   // 4 points
  if(staffList.isEmpty())
    c.getStaff() = Staff("Dummy");
  else {
    c.getStaff() = staffList.front();
    staffList.dequeue();
  }
  return *this;
}

Shop& Shop::operator>>(Customer& c) {                    // 3 points
  if(customerList.isEmpty())
    c = Customer("Dummy");
  else {
    c = customerList.front();
    customerList.dequeue();
  }
  return *this;
}

ostream& operator<<(ostream& os, const Shop& s) {        // 7 points
  os << "Name: " << s.name << endl;
  os << "Size (in square feet): " << s.size << endl;

  os << "Staff: ";
  Queue<Staff> tempStaffList = s.staffList;
  while(!tempStaffList.isEmpty()) {
    os << tempStaffList.front() << " ";
    tempStaffList.dequeue();
  }
  os << endl;
  os << "Customers: ";
  Queue<Customer> tempCustomerList = s.customerList;
  while(!tempCustomerList.isEmpty()) {
    os << tempCustomerList.front() << " ";
    tempCustomerList.dequeue();
  }
  os << endl;
  return os;
}
```

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

/* Rough work */

/* Rough work */

/* Rough work */

/* Rough work */