University of Waterloo

University of Waterloo
Final Examination
Term: Fall Year: 2009

CS246
Software Abstraction and Specification
Sections 01, 02
Instructor Peter Buhr

December 17, 2009
Start Time: 12:30 End Time: 15:00
Duration of Exam: 2.5 hours
Number of Exam Pages (including cover sheet): 6
Total number of questions: 7
Total marks available: 114
CLOSED BOOK, NO ADDITIONAL MATERIAL ALLOWED

Be brief, but you must answer the question! Do not rephrase the question in your answer; use point-form when possible. Programs do not require comments. **PACE YOURSELF PROPERLY, THE LAST PROGRAMMING QUESTION MAY TAKE 1–1.5 HOURS.**

1. (a) **3 marks** What is a constructor, who calls it and when?
   (b) **2 marks** What is the benefit from declaring **operator**s as routines rather than members?
   (c) **4 marks** Explain the difference between initialization and assignment. What mechanism is used to implement each in C++?
   (d) **2 marks** Explain the difference between a *shallow* and *deep* copy for an object.
   (e) **4 marks** Rewrite the following code fragment so it breaks the definition cycle.

```
struct T1 {
    T2 t2;
};
struct T2 {
    T1 t1;
};
T1 t1;
```

2. (a) **2 marks** How does type inheritance relax name equivalence?
   (b) **1 mark** What fundamental programming-language concept are virtual routines based on?
   (c) **5 marks** Given the following declarations:

```
struct Base {
    void f() {}
    void g() {}
    virtual void h() {}
};
struct Derived : public Base {
    void g() {};
    void h() {};
};
Base &bp = *new Derived();
```

   state which member is called for each of the following:

   i. `bp.f();`
   ii. `bp.g();`
   iii. `((Derived &)bp).g();`
   iv. `bp.Base::h();`
   v. `bp.h();`

   (d) **3 marks** Draw a diagram for the most common technique of implementing virtual routines.
   (e) **3 marks** What is a *down cast*?

3. (a) **2 marks** Name and explain the two kinds of templates in C++.
   (b) **2 marks** Explain the two main approaches a container library may use with respect to managing the elements in the container.
   (c) **2 marks** Why do container libraries provide *iterators*?
   (d) **2 marks** Where do the marker values `begin()` and `end()` point for a container in C++?
   (e) **3 marks** Explain 3 ways to access items in a C++ namespace.

4. (a) **2 marks** Explain the compiler flag -O2. Why is this flag not on all the time?
   (b) **2 marks** Define *debugging*.
   (c) **2 marks** Once a program starts running, name and explain the two basic categories of programming errors.
   (d) **1 mark** True or false : the make command expresses a relationship among sources.
   (e) **2 marks** What mechanism does the g++ compiler provide so that make dependencies can always be correct?

5. (a) **1 mark** In any software system, where is the *truth* about its behaviour?
   (b) **1 mark** True or false : adding more people to a late software project ensures the project meets its deadline.
   (c) **2 marks** Define *system modelling*.
   (d) **3 marks** Explain 3 levels that design tools can be used in modelling a software project.
   (e) **2 marks** Explain the notion of *association* in a UML class diagram.
   (f) **3 marks** What is a *managed* programming language? Give an advantage and disadvantage.

6. (a) **1 mark** In the *agile* development process, how do programmers often work?
   (b) **2 marks** What is a software *design pattern*?
   (c) **2 marks** Explain *equivalence partitioning* and *boundary value* testing.
   (d) **2 marks** Explain *regressing* and *performance* testing.
   (e) **1 mark** True or false : the programmer that writes the code should test the code.

7. **45 marks** Write the code for the following new component in the WATCola simulation. The bottling plant has a truck to make deliveries of soda to vending machines. However, during the holiday season, the truck cannot keep up with the demand to fill the machines, so the bottling plant has to out-source delivery to UTS shipping. UTS shipping has the following interface (you may add only a public destructor and private/protected members):

```
class UTS {
  public:
    virtual ~UTS() {};     // necessary to trigger destructors in inherited classes
    virtual unsigned int pickup() = 0;
    virtual bool status( unsigned int tracking ) = 0;
    virtual void action() = 0;
};

class UTSeast : public UTS, ... { // YOU MAY ADD ADDITIONAL INHERITANCE
  public:
    UTSeast( Printer &prt, NameServer &nameServer, BottlingPlant &plant,
        unsigned int numVendingMachines, unsigned int maxStockPerFlavour );
    ~UTSeast();
    virtual unsigned int pickup();
    virtual bool status( unsigned int tracking );
    virtual void action();
};
```

```
class UTSwest : public UTS, ... { // YOU MAY ADD ADDITIONAL INHERITANCE
  public:
    UTSwest( Printer &prt, NameServer &nameServer, BottlingPlant &plant,
        unsigned int numVendingMachines, unsigned int maxStockPerFlavour );
    ~UTSwest();
    virtual unsigned int pickup();
    virtual bool status( unsigned int tracking );
    virtual void action();
};
```

(Do not recopy any lines unless you need to make a change.) When a truck is created, it is passed a printer, name server, bottling plant, number of vending machines, and maximum number of bottles of each flavour in a vending machine. At creation, a UTS truck calls NameServer::getMachineList() to obtain an array of pointers (size numVendingMachines) to vending machines so it can visit each machine to deliver new soda. UTS trucks provide three capabilities: pickup, status, and action.

Member pickup is called by the bottling plant to inform the UTS truck to come pickup a delivery for the vending machines. The pickup member returns a tracking number that can subsequently be used to check on the status of a delivery by the UTS truck. Tracking numbers start at 0 and increase by one for each pickup (assume no overflow).

Member status is called by the bottling plant with a tracking number to check on the status of a delivery. The status member returns **true** if the delivery has occurred and **false** otherwise. Since the status member can be called anytime for any tracking number, a UTS truck has to retain the tracking-number information indefinitely for every delivery.

Member action defines the behaviour of a UTS truck. It return immediately if no call has occurred to the pickup member by the bottling plant. Otherwise, it calls the bottling-plant's BottlingPlant::getShipment(shipment) member, which fills the shipment array (size NUM_OF_FLAVOURS) with the number of bottles of each soda flavour in the shipment. Compute the amount in the shipment returned by the bottling plant. Then make a delivery to each vending machine, so long as there are bottles remaining in the shipment. Obtain the amount of remaining stock left from the vending machine by calling VendingMachine::inventory(). The UTS truck uses this information to transfer into each machine as much of its stock of new soda as fits; for each kind of soda, no more than MaxStockPerFlavour per flavour can be added to a machine. If the truck cannot top-up a particular flavour, it transfers as many bottles as it has (which could be 0). Finally, the UTS truck informs the vending machine that restocking is complete by calling VendingMachine::restocked().

The UTS truck prints the following information:

State	Meaning	Additional Information
S	starting	
P	picked up shipment	total amount of all sodas in the shipment
s	status check	result of status check
d	begin delivery to vending machine	vending machine id, amount remaining in the shipment
D	end delivery to vending machine	vending machine id, amount remaining in the shipment
F	finished	

States d and D are printed for each vending machine visited during restocking.

There are two kinds of UTS trucks the bottling plant may hire: east and west. The difference between an east and west UTS truck is that the east truck deliveries to the vending machine in order 0 to numVendingMachines-1, while the west truck deliveries to the vending machine in order numVendingMachines-1 to 0.

Share common code across implementations. No marks will be given for duplicate lines of code. Write only the code associated with the UTS trucks. Assume any other modules are written.

The name-server interface is:

```
class NameServer {
  public:
    NameServer( Printer &prt, unsigned int numVendingMachines, unsigned int numStudents );
    void VMregister( VendingMachine *vendingmachine );
    VendingMachine *getMachine( unsigned int id );
    VendingMachine **getMachineList();
};
```

The name server's function is to manage the vending-machine names. Vending machine's call VMregister to register themselves so students can subsequently locate them. A student calls getMachine to find a vending machine, and the name server must cycle through the vending machines *separately* for each student starting from the initial position via modulo incrementing to ensure a student has a chance to visit every machine. A truck calls getMachineList to obtain an array of pointers to vending machines so it can visit each machine to deliver new soda.

The bottling-plant interface is:

```
class BottlingPlant {
  public:
    BottlingPlant( Printer &prt, NameServer &nameServer, unsigned int numVendingMachines,
                   unsigned int maxShippedPerFlavour, unsigned int maxStockPerFlavour,
                   unsigned int timeBetweenShipments );
    void getShipment( unsigned int cargo[] );
    void action();
};
```

The bottling plant produces random new quantities of each flavour of soda, [0, MaxShippedPerFlavour] per flavour. The truck calls getShipment to obtain a shipment from the plant (i.e., the production run), and the shipment is copied into the cargo array passed by the truck.

The vending-machine interface is:

```
class VendingMachine {                  // general vending machine
  public:
    virtual ~VendingMachine() {}        // necessary to trigger destructors in inherited classes
    virtual bool buy( Flavours flavour, WATCard *&card ) = 0; // you define Flavours
    virtual unsigned int *inventory() = 0;
    virtual void restocked() = 0;
    virtual unsigned int cost() = 0;
    virtual unsigned int getId() = 0;
};
```

A student calls buy to obtain one of its favourite sodas. If the specified soda is unavailable or the student has insufficient funds to purchase the soda, buy returns **false**; otherwise, the student's WATCard is debited by the cost of a soda and buy returns **true**. The truck calls inventory to return a pointer to an array containing the amount of each kind of soda currently in the vending machine. After transferring new soda into the machine by directly modifying the array passed from inventory, the truck calls restocked to indicate the operation is complete. The cost returns the cost of purchasing a soda for this machine. The getId returns the identification number of the vending machine.

The printer interface is:

```
class Printer {
  public:
    enum Kind { WATCardOffice, NameServer, Truck, UTS, BottlingPlant, Student, Vending };
    Printer( unsigned int numStudents, unsigned int numVendingMachines );
    ~Printer();
    void change( Kind kind, char state );
    void change( Kind kind, char state, int value1 );
    void change( Kind kind, char state, int value1, int value2 );
    void change( Kind kind, unsigned int LId, char state );
    void change( Kind kind, unsigned int LId, char state, int value1 );
    void change( Kind kind, unsigned int LId, char state, int value1, int value2 );
};
```