ECE 462 Written Assignment 4

Due: 12:20PM, December 06, 2010

Name:

PUID:

1 C++ Multiple Inheritance

- The program intends to express the concept "*A teaching assistant is a graduate student and teacher.*" using multiple inheritance in C++. Draw the UML class hierarchy of this program, **including** all attributes. You do not have to draw the methods (namely, the member functions).
- Unfortunately, the program has some problems at compilation. Please mark the locations of the problems and provide brief explanations. Please be aware that there may be multiple problems.
- Fix the problem by **adding or removing** any of the following words:

`final`	`catch`	`this`	`operator`
`public`	`protected`	`private`	`const`
`class`	`try`	`virtual`	`self`
`super`	`throw`	`::`	`new`

  Please mark the locations and the changes. You may need to remove and add a different word at the same location (i.e. replacing); you may need to remove and add the same word in different locations (i.e. moving).
- What is the output of this program?

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

class Person
{
public:
  Person(string n, int a):
```

```
    name(n),
    age(a)
  {
  }
  void print()
  {
    cout << "Person::print" << endl;
    cout << "    name: " << name << endl;
    cout << "    age: " << age << endl;
  }
  virtual ~Person()
  {
  }
private:
  const string name;
  int age;
};

class Teacher: public Person
{
public:
  Teacher(string n, int a, string s):
    Person(n, a),
    subject(s)
  {
  }
private:
  void print()
  {
    cout << "Teacher::print" << endl;
    Person::print();
    cout << "    subject: " << subject << endl;
  }
  string subject;
};

class Student: public Person
{
public:
  Student(string n, int a, string d):
    Person(n, a), department(d)
  {
  }
  void print()
  {
```

```
    cout << "Student::print" << endl;
    Person::print();
    cout << "    department: " << department << endl;
  }
private:
  string department;
};

class GradStudent: public Student
{
public:
  GradStudent(string n, int a, string d, string v):
    Person(n, a),
    Student(n, a, d),
    adviser(v)
  {
  }
  void print()
  {
    cout << "GradStudent::print" << endl;
    Student::print();
    cout << "    adviser: " << adviser << endl;
  }
private:
  string adviser;
};

class TeachingAssistant: public GradStudent, public Teacher
{
public:
  TeachingAssistant(string n, int a, string s,
                    string d, string v, int y):
    Person(n, a),
    GradStudent(n, a, d, v),
    Teacher(n, a, s),
    salary(y)
  {
  }
  void print()
  {
    cout << "TeachingAssistant::print" << endl;
    GradStudent::print();
    Teacher::print();
    cout << "    salary: " << salary << endl;
  }
```

```
private:
  int salary;
};

int main(int argc, char * argv[])
{
  Person * ta =
    new TeachingAssistant("Smith", 25, "Java", "ECE", "Johnson", 2000);
  ta -> print();
  /* ATTENTION: should print ALL attributes */
  delete ta;
  return 0;
}
```

2 C++ Argument Passing

Consider the following program.

- Does the program have any syntax error? Assume that all `names`, `ages`, `sizes`, and `data` are correctly defined. Hint: Neither copy constructor nor operator = is provided.
- In `main`, which call (or calls) of `test..Function` returns 1?
- If `Vector`'s destructor contains (i.e. taking it out from the comment)

  ```
  delete [] data;
  ```

  which call (or calls) will make the program crash at run-time? **Why**?

```
#include <iostream>
#include <string>
using namespace std;
class User
{
public:
  User (string n, int a):
    name(n),
    age(a)
  {
  }
  virtual ~User()
  {
  }
  virtual void print()
  {
    cout << "name: " << name << endl;
    cout << "age: " << age << endl;
  }
  bool operator == (const User & u2) const
  {
    if ((name != u2.name) || (age != u2.age))
      {
        return false;
      }
    return true;
  }
private:
  string name;
  int age;
```

```
};

class UserFunction
{
public:
  static void uf1(User u1, User u2)
  {
    u1 = u2;
  }
  static void uf2(User & u1, User & u2)
  {
    u1 = u2;
  }
  static void uf3(User * u1, User * u2)
  {
    u1 = u2;
  }
  static void uf4(User * u1, User * u2)
  {
    * u1 = * u2;
  }
};

bool testUserFunction(int f,
                      string n1, int a1,
                      string n2, int a2)
{
  User u1(n1, a1);
  User u2(n2, a2);
  switch (f)
    {
    case 1:
      UserFunction::uf1(u1, u2);
      break;
    case 2:
      UserFunction::uf2(u1, u2);
      break;
    case 3:
      UserFunction::uf3(& u1, & u2);
      break;
    case 4:
      UserFunction::uf4(& u1, & u2);
      break;
    default:
      cout << "unknown option" << endl;
```

```
    }
  return (u1 == u2);
}

class Vector
{
public:
  Vector(int s, int * d)
  {
    size = s;
    data = new int[s];
    for (int i = 0; i < s; i ++)
      {
        data[i] = d[i];
      }
  }
  virtual ~Vector()
  {
    // *************
    // ATTENTION
    // delete [] data;
    // *************
  }
  virtual void print()
  {
    cout << "There are " << size << " elements:" << endl;
    for (int i = 0; i < size; i ++)
      {
        cout << data[i] << " ";
      }
    cout << endl;
  }
  bool operator == (const Vector & u2) const
  {
    if (size != u2.size)
      {
        return false;
      }
    for (int i = 0; i < size; i ++)
      {
        if (data[i] != u2.data[i])
          {
            return false;
          }
      }
```

```
    return true;
  }
private:
  int size;
  int * data;
};

class VectorFunction
{
public:
  static void vf1(Vector v1, Vector v2)
  {
    v1 = v2;
  }
  static void vf2(Vector & v1, Vector & v2)
  {
    v1 = v2;
  }
  static void vf3(Vector * v1, Vector * v2)
  {
    v1 = v2;
  }
  static void vf4(Vector * v1, Vector * v2)
  {
    * v1 = * v2;
  }
};

bool testVectorFunction(int f,
                        int s1, int * d1,
                        int s2, int * d2)
{
  Vector v1(s1, d1);
  Vector v2(s2, d2);
  switch (f)
    {
    case 1:
      VectorFunction::vf1(v1, v2);
      break;
    case 2:
      VectorFunction::vf2(v1, v2);
      break;
    case 3:
      VectorFunction::vf3(& v1, & v2);
      break;
```

```
    case 4:
      VectorFunction::vf4(& v1, & v2);
      break;
    default:
      cout << "unknown option" << endl;
    }
  return (v1 == v2);
}

int main(int argc, char * argv[])
{
  /* Assume
     name11 != name12, age11 != age12
     name21 != name22, age21 != age22
     name31 != name32, age31 != age32
     name41 != name42, age41 != age42

     size11 != size12, data11 != data12
     size21 != size22, data21 != data22
     size31 != size32, data31 != data32
     size41 != size42, data41 != data42
  */

  /* Which prints 1? */
  cout << testUserFunction(1, name11, age11, name12, age12) << endl;
  cout << testUserFunction(2, name21, age21, name22, age22) << endl;
  cout << testUserFunction(3, name31, age31, name32, age32) << endl;
  cout << testUserFunction(4, name41, age41, name42, age42) << endl;

  /* Which prints 1? */

  /* Which would cause the program to crash if Vector's
     destructor has

     delete [] data;

  */
  cout << testVectorFunction(1, size11, data11, size12, data12) << endl;
  cout << testVectorFunction(2, size21, data21, size22, data22) << endl;
  cout << testVectorFunction(3, size31, data31, size32, data32) << endl;
  cout << testVectorFunction(4, size41, data41, size42, data42) << endl;

  return 0;
}
```

3 C++ Client

Consider the following "storage server" written in C++ and Qt. The server stores an integer. A client can have three commands:

- *bye*: close the connection with the server.
- *load*: read the latest value from the server.
- *save number*: save the number at the server. This number is **shared** by all clients.

The following is a snapshot of a client's commands and responses:

```
> telnet ip_addr port_number
Trying ...
Connected to ...
Escape character is '^]'.
Welcome to a storage server
>>  Enter 'bye' to exit                    <<
>>  Enter 'load' to retrieve stored number <<
>>  Enter 'save num' to save number        <<
save 5
load
5
save 7
load
7
          // another client saved 9
load
9
save 8
bye
Connection closed by foreign host.
```

- Write `ClientHandler::ClientHandler`.
- Write `ClientHandler::readFromClient`. Do **not** send anything to a client unless a client issues `load` or `save`.
- Suppose the shared value is 5 right now. Three clients want to add 1, 2, and 3 respectively. Each client performs three steps:
  1. `load`: read the current value from the server.
  2. `add`: add 1, 2, or 3 to the value just retrieved from the server.

3. `save n`: save the new value back to the server; `n` is the value obtained in step 2.

What are the possible values of the number stored at the server? Consider all possible interleavings of the clients' three steps.

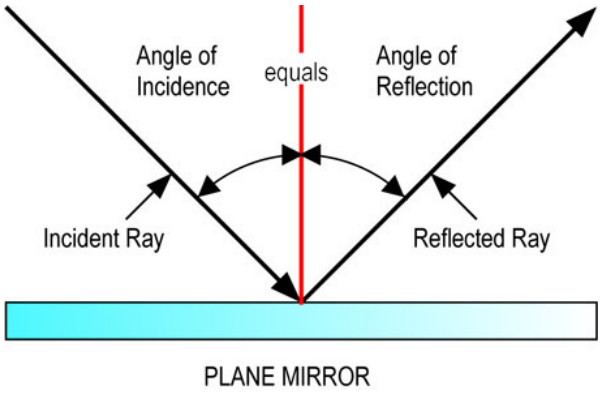
```
//
// server.h
//
#ifndef CHATSERVER_H
#define CHATSERVER_H
#include <QtNetwork>
#include <QString>
#include <QThread>
#include <QApplication>
#include <vector>
using namespace std;
class ChatServer;
class ClientHandler : public QObject
{
  Q_OBJECT
private:
  QTcpSocket* ch_socket;
  QTextStream* ch_os;
public:
  ClientHandler(QTcpSocket* sock, ChatServer * serv);
  virtual ~ClientHandler();
  ChatServer * ch_server;
private slots:
  void readFromClient();
  friend class ChatServer;
};

class ChatServer: public QObject
{
  Q_OBJECT
private:
  QTcpServer * cs_server;
  QList<ClientHandler *> cs_clientList;
public:
  ChatServer();
  virtual ~ChatServer();
  // *****************
  // ATTENTION: static
  // *****************
  static int cs_value; // <----
public slots:
  void connectNewClient();
};
#endif

//
```

```
// server.cpp
//
#include "server.h"
#include <iostream>
using namespace std;

int ChatServer::cs_value = 0;
ChatServer::ChatServer()
{
  cs_server = new QTcpServer();
  if (! cs_server->listen())
    {
      qWarning("Failed to register the server port");
      exit(1);
    }
  cout << "Server port " << cs_server->serverPort() << endl;
  connect(cs_server, SIGNAL(newConnection()),
          this, SLOT(connectNewClient()));
}

void ChatServer::connectNewClient()
{
  QTcpSocket* socket = cs_server->nextPendingConnection();
  ClientHandler* clh = new ClientHandler(socket, this);
  cs_clientList.push_back(clh);
  cout << "A new client connected " << endl;
}

ChatServer::~ChatServer()
{
  if (cs_server)
    { delete cs_server; }
  ClientHandler* clh = cs_clientList.takeFirst();
  while (clh != 0)
    {
      delete clh;
      clh = cs_clientList.takeFirst();
    }
}

/*
   Write the constructor of

   ClientHandler::ClientHandler
```

```
   remember to connect appropriate signal(s) with slot(s)

*/

ClientHandler::~ClientHandler()
{
  if (ch_os)
    {
      delete ch_os;
    }
}

void ClientHandler::readFromClient()
{
  /*

    handle three commands: bye, load, and save

   */
}

int main(int argc, char* argv[])
{
  QApplication app(argc, argv);
  ChatServer server;
  return app.exec();
}
```

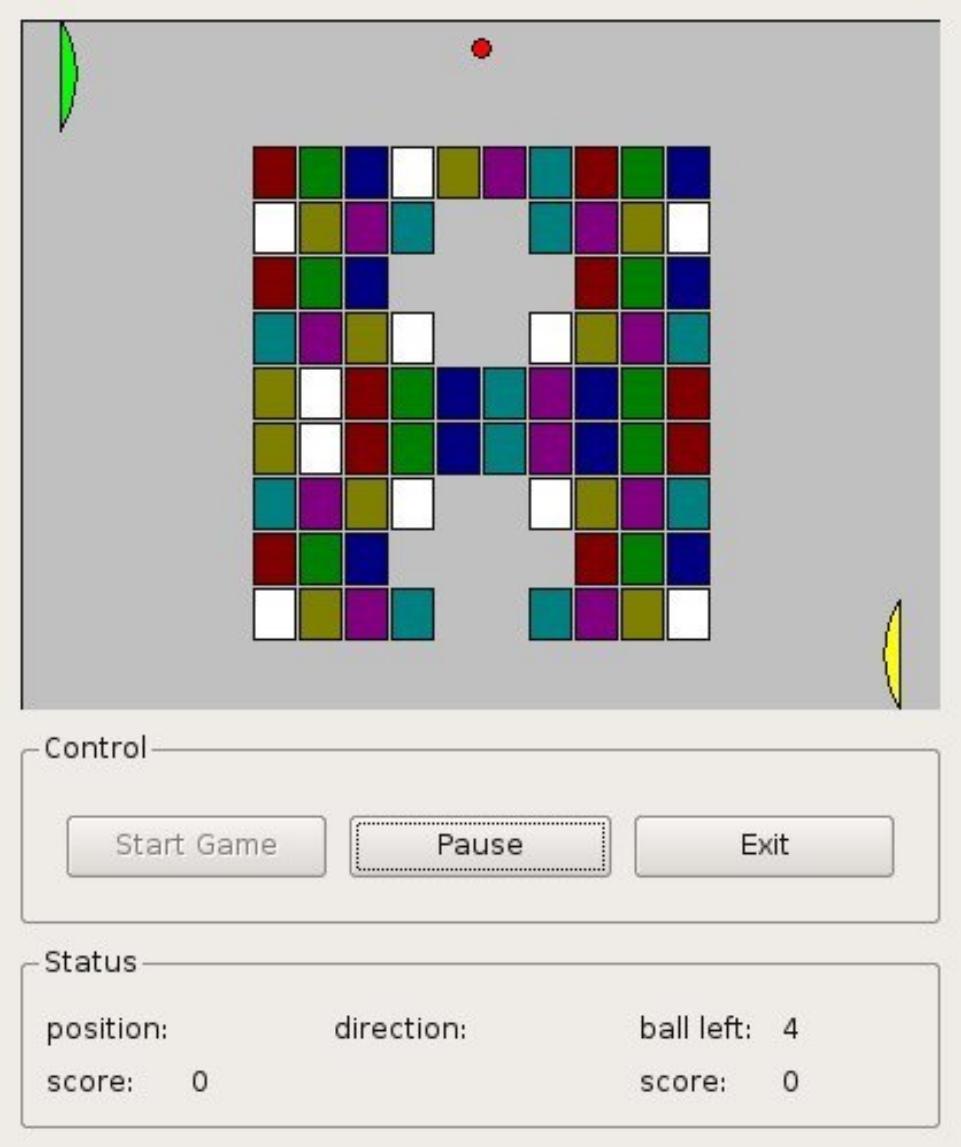
4 Game Strategy

In a Breakout Game, the ball follows the law of reflection: the angle of incidence θ_i is the same as the angle of reflection θ_r, illustrated below. The angles are determined relative to the *normal vector* of the surface.



Consider a game in which the paddle is a semicircle and the normal vector varies, determined on the location of the collision. Suppose the ball's current position is (13.7274, 2.9366) and the direction is (-0.9659 -0.2588). The ball's center is at (6, 0) and the radius is 2.

- Where is the location of the collision?
- What is the new direction of the ball?



Consider a two-player breakout game, as shown in the figure. A player gets a point if the red ball hits a brick after bouncing off this player's paddle (the green and yellow curves). The player also receives a point if the ball hits the top or the bottom wall. A player would like the ball to hit a brick and then bounce back to this player, not to the other player.

Is it possible to create an unbeatable player that controls the ball's direction so that the opponent never has a chance to hit the ball? The trick is using the curve surface of the paddle to control the ball. If it is possible, explain the strategy, write down the mathematical formulas, and solve the problem. If it is not possible, prove it.

Is it still possible if the paddles are not semicircles, but only 90° (45 ° above and below the horizontal line)? Why?