ECE 462 Final Exam

08:00-10:00AM, December 15, 2009

I certify that I will not receive nor provide aid to any other student for this exam.

$\boxed{\textbf{Signature:}}$

You must sign here. Otherwise, you will receive **2-point** *penalty.*

This exam is printed **double sides**. There are 5 questions, 4 points for each question.

Write your answers next to the questions. If you need more space, you can use the blank pages.

This is an *open-book, open-note* exam. You can use any book or note or program printouts.

Please turn off your cellular phone **now**.

ECN will reset the password of your account and erase all files. Please save your files before **December 18***.*

May you have a great winter break.

Contents

1 C++ Multiple Inheritance, 4 points

- (1 point) 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).
- (1 point) 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.
- (1 point) 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).
- (1 point) 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, 4 points

Consider the following program.

- (1 point) 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.
- (2 points) In `main`, which call (or calls) of `test...Function` returns 1?
- (1 points) 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, 4 points

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.
```

- (1 point) Write `ClientHandler::ClientHandler`.
- (2 points) Write `ClientHandler::readFromClient`. Do **not** send anything to a client unless a client issues `load` or `save`.
- (1 point) 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 Version Control, 4 points

Consider *merge-based* version control programs, such as CVS.

Three students in ECE462 are doing a group project. Their accounts are `ece462a4`, `ece462a6`, and `ece462b2`. A group called `ece462g9` has been created. It contains the three accounts and no other account. Their repository will reside in `ece462a6`'s directory and the complete path is `/home/ece462a6/GPA2/`.

- (2 points) List the commands `ece462a6` must enter so that `ece462a4`, `ece462a6`, and `ece462b2` can read from and write to the repository and nobody else can.
- (1 point) On October 1, the latest version in the repository contains the following code and all three students have the same latest code:

```
#include<iostream>
using namespace std;
int main(int argc, char * argv[])
{
    return 0;
}
```

`ece462a4` changed the code by adding a comment

```
#include<iostream>
using namespace std;
// comment by ece462a4
int main(int argc, char * argv[])
{
    return 0;
}
```

`ece462b2` changed the code by defining an integer:

```
#include<iostream>
using namespace std;
int main(int argc, char * argv[])
{
    int i = 0;
    return 0;
}
```

Please pay attention to the **orders** of the commands. It is possible that some commands may **fail**.

On October 2, `ece462a4` executed `update` and `commit`.

On October 3, `ece462b2` executed `commit` and `update`.

On October 4, `ece462a6` executed `update` and `commit`.

What does `ece462a6` **see now**?

- (1 point) On October 5, `ece462a4` changed the code to

```
#include<iostream>
using namespace std;
// comment by ece462a4
int main(int argc, char * argv[])
{
    int j = 1;
    return 0;
}
```

  executed `update` and `commit`.

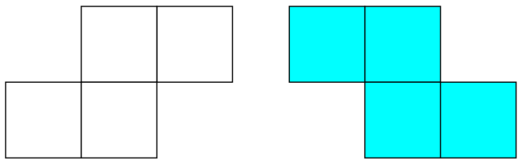
  On October 6, `ece462b2` executed `update` and `commit`. The program is unchanged between the commands.

  What does `ece462b2` **see now**?

5 Tetris Packing, 4 points

Tetris allows rotating pieces but mirroring is **not** allowed.

Burgiel* shows that a repeating **sequence** of



will force **any** player to lose in a board of width $2n$ when n is an odd number. This sequence appears (a) (b) (a) (b) (a) (b) ... It is a sequence and the order **cannot** be changed. The sequence repeats until the game is over.

* H. Burgiel. "How to lose at Tetris." Mathematical Gazette, July 1997.

- (2 points) For the above sequence, please prove whether it is possible to play indefinitely on a Tetris board whose width is 36 squares and height is 30 squares. Please notice that 36 = $2n$ and $n = 18$ is an even number.

  If it is possible, show one strategy to play indefinitely. If it is not possible, show that no strategy can possibly exist.

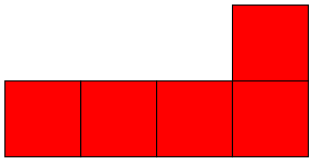
  Please clearly mark the **boundaries of pieces** in your explanation.

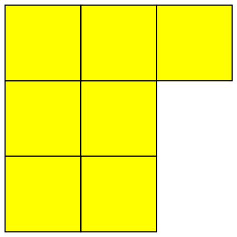
- (2 points) Consider the following (next page) **sequence** of Tetris pieces.

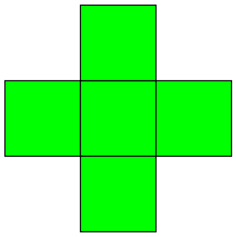
  Please prove whether it is possible to play indefinitely on a Tetris board whose width is 36 squares and height is 30 squares.

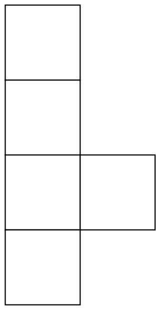
  If it is possible, show one strategy to play indefinitely. If it is not possible, show that no strategy can possibly exist.

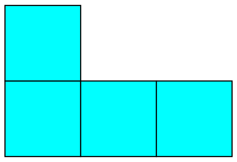
  Please clearly mark the **boundaries of pieces** in your explanation.

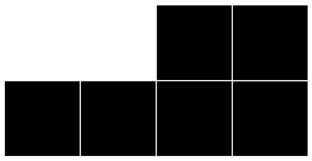

(a)

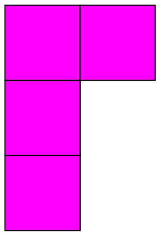

(b)


(c)


(d)


(e)


(f)


(g)