Part I: Short Answer (6 questions, 18 points total)

Answer all of the following questions. READ EACH QUESTION CAREFULLY. Answer each question in the space provided on these pages. Budget your time so you spend enough on the programming questions at the end.

Keep your answers short and to the point. Good luck.

The first 4 questions all refer to the following related class declarations. Implementations of most, but not all, of the functions are given in the declarations.

```
    class Person {
    public:
        Person(string inName, bool inIsFemale) { name = inName; isFemale = inIsFemale;  }
        virtual ~Person( )                  { cout << “aaaarghhhhh…” << endl; }
        virtual string getName( )           { return name; }
        virtual bool   getIsFemale( )       { return isFemale; }
/*1*/   virtual void   eat(int amount) = 0;
    private:
        string name;              // person’s name
        bool isFemale;            // true if person is female
    };

    class Man : public Person {
    public:
        Man(string inName) : name (inName), isFemale (false) { }
/*2*/   virtual void eat(int amount)        { if (amount > 5) burp(); }
/*3*/   virtual void eat(string what)       { if (what == “steak” ) burp(); }
        virtual void burp( )                { cout << "buuurrrrp" << endl; }
    };


    class King : public Man {
    public:
        King(string inName);      // not implemented here
        virtual ~King( )          { cout << “you too, Brutus?!” << endl; }
        virtual void burp( )      { burp(); cout << "It's good to be the king!" << endl; }
    };
```

1. (3 points) The constructor for Man contains an error. What is the problem, and how could you fix it?

The constructor is attempting to access private data in class Person (name, isFemale). The fix is to use a proper initializer using the constructor for Person:

```
Man(string inName) : Person(inName, false) { }
```

2. (3 points) The constructor for King should create a male person, with the specified name, preceded with "His Highness, King " (e.g., if the argument to the constructor were "Louis XV", then name should be initialized to "His Highness, King Louis XV"). Implement it below, as it would appear in a separate .cpp implementation file.

```
King::King(string inName)  : Man("His Highness King " + inName) { }
```

(Some answers had Person("His Highness King " + inName, false) as the initializer. This isn't legal – the only constructors that can be used are those of the immediate base class. However, we decided to give credit for that answer, since it is a technical point that we haven't covered carefully.)

3. (3 points) The function King::burp( ) is syntactically correct, but contains a bug. What is the bug, and how could you fix it so that the king burps first (by calling an appropriate function to write the string "buuurrrrp"), and then says "It's good to be the king!"? Give an implementation below, as it would appear in a separate .cpp implementation file.

The function call burp() inside King::burp() is a recursive call to itself, which creates an infinite recursion.

```
void King::burp( ) {
        Man::burp( );
        cout << "It's good to be the king!" << endl;
}
```

4. (3 points) What is the relationship between the eat functions marked "/*1*/", "/*2*/" and "/*3*/"? Circle the correct answer. (Only one of these choices is correct, although there may be other ways, not given here, to describe the relationships between the functions.)

a) /*1*/ overrides /*2*/ and overloads /*3*/

b) /*2*/ overrides /*1*/ and overloads /*3*/

c) /*2*/ overloads /*1*/ and overrides /*3*/

d) /*3*/ overloads /*1*/ and overrides /*2*/

5. (3 points) What happens when the following program is executed? If there are errors in the code, indicate where the errors occur and explain them. If the program is ok, write the output in the space provided.

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

class plane {
  public:
    virtual void takeoff()        { cout << "whoosh!" << endl; }
    virtual void landing () { cout << "Plane going down" << endl; }
};

class transport:public plane {
  public:
    void landing() { cout << "Transport going down" << endl; }
    void refuel()   { cout << "Transport is refueling" << endl; }
};

int main() {
   plane *planePtr = new transport;
   planePtr->takeoff();
   planePtr->landing();
   delete planePtr;
   return 0;
}
```

Output:

whoosh!
Transport going down

6. (3 points) The following program reads a number from `cin` and calls function test with that number as an argument:

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

void test(int n) {
        if (n > 1) {
                test(n-1);
        }
        cout << n << " ";
}

int main() {
        int i;
        cin >> i;
        test(i);
        return 0;
}
```

Which of the following possible outputs could be produced when the program is executed, assuming the user enters a suitable input number? (If there are two or more correct answers, circle all of the correct ones.)

a) 1 2 3 4 5

b) 5 4 3 2 1

c) 0 1 2 3 4

d) 4 3 2 1 0

Part II. Programming Problem (1 question, 20 points total)

7. You've been hired by UW to modify their student records database. Assume that the following classes are defined to store information about student grades.

```
class StudentRecord {          // single grade entry for a student:
public:
   string  name;               // student name
   string  course;             // course name, e.g., "CSE143"
   double grade;               // student's grade in the course
};

class StudentDatabase {                // List of student grade entries
public:
     StudentDatabase();                // construct empty list
     ~StudentDatabase();               // destructor

     // Look for a record containing the given student name and course name.
     // If found, replace the previous grade with newGrade; otherwise, create
     // a new student record with the given information and add it to the database,
     // expanding the list if needed.

     void addGrade(string studentName, string courseName, double newGrade);

private:
     int                capacity;     // current allocated size of array records
     StudentRecord *records;          // student records are stored in records[0..size-1]
     int                size;         // current size of the list
};
```

The grades are stored in a dynamically allocated array named records. The list is not sorted. The array should be automatically expanded as needed to hold additional data.

(a) (3 points) Complete the implementation of an appropriate destructor for this class.

```
StudentDatabase::~StudentDatabase( ) {


        delete [ ] records;




}
```

(b) (17 points) Give an implementation of function addGrade so it changes an existing grade or adds a new record as specified. If the list is already full when a new grade record is added, its capacity should be doubled to make additional room first.

If you want to use additional helper functions in your solution, give their implementations here. You do not need to worry about adding their declarations to the class declaration.

```
// Look for a record containing the given student name and course name.
// If found, replace the previous grade with newGrade; otherwise, create
// a new student record with the given information and add it to the database,
// expanding the list if needed.

void StudentDatabase::addGrade(string studentName, string courseName, double newGrade) {

    // Search for matching record; if found, update grade and return
    int k = 0;
    while (k < size) {
        if (records[k].name == studentName && records[k].course == courseName) {
            records[k].grade = newGrade;
            return;
        }
        k++;
    }

    // Not found – add new record to database

    // If array is full, double its size
    if (size == capacity) {
        capacity = 2*capacity;
        StudentRecord * newRecords = new StudentRecord[capacity];
        for (k = 0; k < size; k++) {
            newRecords[k] = records[k];
        }
        delete [ ] records;
        records = newRecords;
    }

    // add new entry to the list
    records[size].name   = studentName;
    records[size].course = courseName;
    records[size].grade  = newGrade;
    size++;
}
```

(see note on next page for another version of the search)

This version of the search that combines the test for a match with the check for whether all records have been searched. Note that it is crucial that the k<size check appears first in the condition. (Why?)

```
// Search for matching record; if found, update grade and return
k = 0;
while (k < size &&
        (records[k].name != studentName || records[k].course != courseName)  {
    k++
}

if (k < size) {
    records[k].grade = newGrade;
    return;
}

// Not found – add new record to database
…
```