Part I: Short Answer (4 questions, 20 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. Each question is weighted differently. Keep your code as short as possible. Good luck.

1. (5 points) Write the `swap` function for the code below. Your `swap` function should correctly swap values so that the program below will output the line "Swap function works correctly!"

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

int main () {
   int x = 5;
   int y = 2;
   swap(x, &y);

   if ((x == 2) && (y == 5))
      cout << "Swap function works correctly!" << endl;

   return 0;
}
```

```
void swap(int &num1, int * num2) {
     int temp = num1;
     num1 = *num2;
     *num2 = temp;
}
```

2. (4 points) Consider the following class declaration:

```
class Books {
public:
   Books();
   Books(string bName, int bPrice);
   int getPrice(int bPrice);
private:
   string name;
   int price
};
```

In order to allow the user to change the `price` member variable, the designer has decided to add a new member function `setPrice` to the class `Books` above. Explain in words, the differences in the use of the **`const`** keyword in the following two possible function declarations. What restrictions does **`const`** impose in each case?

(a) `void setPrice(int bPrice) const;`
(b) `void setPrice(const int bPrice);`

In case (a), the const method means that `setPrice` CANNOT change any data member of the `Books` class. In this example, this doesn't make any sense since the point of the `setPrice` method is to change one of the data members (`price`).

In case (b), const protects the implementer of `setprice` by specifying that the input parameter `bPrice` CANNOT be changed with in the `setprice` method. Since the argument to `setprice` is copied into the parameter `bPrice`, const does not affect the client.

3. (4 points) Answer the following question based on the code below.

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

void main()  {
     int* intPtr1 = new int[20];
     for(int i = 1; i <= 20; i++)
          intPtr1[i-1] = i;
     int* intPtr2 = &intPtr1[2];
     cout << intPtr2[3];
     // YOUR CODE BEGINS

     delete [] intPtr1;

}
```

(a) If there are errors in the above program, explain them below. If no errors exist, write the program's output below.

6

(b) In the space provided above, write code necessary to deallocate all dynamic memory in the program.

4. (7 points) Read the following class definition which exists in the file "point.h"

```
// a point with integer coordinates on an <x,y> plane
class Point {
public:
   Point(int x, int y);   // construct point with given location
   void setX(int newX);  // set x coordinate to newX
   void setY(int newY);  // set y coordinate to newY
   int getX();  // returns the x coordinate of this Point
   int getY();  // returns the y coordinate of this Point
private:
   …    // details hidden
};
```

(a) What occurs when the following program is executed? Describe the output produced, or, if errors exist, explain the problem(s).

```
#include <iostream>
#include "point.h"
using namespace std;

int main() {
   Point p;
   Point q;
   p.setX(17);
   p.setY(42);
   q = p;
   cout << q.getX() << endl;
   return 0;
}
```

No default constructor exists for the `Point` class. This will cause an error. We cannot create two `Point` objects in the first two lines of `main` without a default constructor.

(b) Write a `point` member function which overloads the "+" operator so that a client can add two points together. If `p1` is a point object with coordinates `x1` and `x2`, and `p2` is a point object with coordinates `y1` and `y2`, then the sum of `p1` and `p2` should have coordinates `x1+y1` and `x2+y2`.

```
Point Point::operator+(const Point & secondPoint) {
   int sumX = getX() + secondPoint.getX();
   int sumY = getY() + secondPoint.getY();
   Point p(sumX, sumY);
   return p;
}
```

Part II. Programming Problem
(1 question with 2 parts, 15 points total)

5. (15 points) You are the owner of a comic book store and need to update your computer software. You are currently using the following data structure to represent the comics.

```
class ComicData {       //  represents a single comic book
public:
    string getName();
    double getPrice();
    int getQuantity();
private:
    string  mName;      //  Comic Name
    double mPrice;      //  Comic book price.
    int   mQuantity;    //  number of this comic currently in the store
};

//  ComicDatabase stores information about all comics in the store in a
//  dynamic array of ComicData objects.  The array is always full, i.e., the
//  number of comics in the array equals the capacity of the array.
class ComicDatabase {
public:
    ComicDatabase();
    ComicData* restock(int quantity);
    //Your Declaration Here

    ComicDatabase(ComicDatabase & other);

private:
    int size;                 //  current number of comics in the database
    ComicData *comics;        //  pointer to a dynamic array of ComicData
};
```

(a) The first part of your job is to implement a copy constructor for the `ComicDatabase` class. Place your declaration for the copy constructor in the space provided above, and your code in the space below.

```
ComicDatabase::ComicDatabase(ComicDatabase & other) {
   size = other.size;
   comics = new ComicData[size];
   for (int i=0; i<size; ++i)
      comics[i] = other.comics[i];
}
```

(b) The second part of your assignment is to implement the `restock` function defined above. This function should return a pointer to an array containing copies of all elements in the `comics` array whose member variable `mQuantity` is **less than or equal to** the `quantity` input parameter. Inside the `restock` function, you should dynamically allocate sufficient memory for this array.

```
ComicData * ComicDatabase::restock(int quantity) {
   // First we determine how many comics need restocking
   int numberToRestock = 0;
   for (int i=0; i<size; ++i) {
      if (comics[i].getQuantity() <= quantity)
         numberToRestock++;
   }

   // Then we create a dynamic array for these comics
   ComicData * restockList = new ComicData[numberToRestock];
   int restockCount = 0;
   for (int j=0; j<size; ++j) {
      if (comics[i].getQuantity() <= quantity) {
         restockList[restockCount] = comics[i];
         restockCount++;
      }
   }

   return restockList;
}
```