Instructions:

- Print your name in the space provided below.
- Answer each question in the space provided. If you need to continue an answer onto the back of a page, clearly indicate that you have done so, and label the continuation with the question number.
- If you want partial credit, justify your answers briefly and concisely, even when justification is not explicitly required.
- There are 11 questions, priced as marked. The maximum score is 100.
- When you have completed the test, sign the pledge at the bottom of this page and turn in the test.
- This is a closed-book, closed-notes examination. No calculators or other electronic devices may be used during this examination. You may not discuss (in any form: written, verbal or electronic) the content of this examination with any student who has not taken it. You must return this test form when you complete the examination. Failure to adhere to any of these restrictions is an Honor Code violation.

Do not start the test until instructed to do so!

Name ________________________________

PID ________________________________

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

______________________________
signed
Assume that the CD class referred to in the following code is a simple data class, containing only primitive variables or string objects and the insertion stream operator << is overloaded in the CD class. For the next three questions, consider the partial class interface code that follows:

```cpp
// "Music.h"
#include "CD.h"

const int MAXCDs = 100;

class Music {
private:
    CD* discs[MAXCDs];
    int usage;

public:
    Music();
    Music(int num = 10, CD* cds[]);
    //possible missing Fn prototypes?
    ~Music();
    friend ostream& operator<<(ostream& out, const Music& m);
};
```

```cpp
// "Music.cpp"
Music::Music(int cap) : usage(0) {
    for (int i=0; i < MAXCDs; i++)
        discs[i] = NULL;
}

Music::~Music() {
}

Music::Music(int num, CD* cds[]) {
    if (num >= MAXCDs) usage = MAXCDs;
    else usage = num;
    for (int i=0; i < usage; i++)
        discs[i] = cds[i];
    for (int i=usage; i < MAXCDs; i++)
        discs[i] = NULL;
}

//possible missing Fn implementations?
ostream& operator<<(ostream& out, const Music& m) {
    for (int i=0; i<m.usage; i++)
        out << *(m.discs[i]) << endl;
    return out;
}
```

1. [6 points] The above code is an example of an association relationship between the CD and Music classes. Identify whether the above association relationship is **dynamic** or **static** association and very briefly explain/justify your response?
2. [10 points] For whichever form of association you selected for problem one, explain what change(s) or addition(s) would need to be made to transform the association relationship from dynamic to static or vice versa. If extra functions are necessary for the transformation, give the prototype and implementation of each.

3. [10 points] Assume that a Music object named mus exists and that the diagram below graphically depicts its current state in memory. Given the client code function prototype: `void manageCollection( Music myMusic);` If the function invocation: `manageCollection( mus );` occurs, then draw the resulting graphical diagram representing the state in memory of the formal parameter myMusic:

```
void manageCollection( Music myMusic);
manageCollection( mus );
```

```
usage: 3
size: 100
discs:
```
4. [10 points] You have been charged with developing a taxi simulation. Another programmer in the project is developing the Taxi class and you are developing the Fare class. Consider the partial Taxi class interface:

```cpp
class Taxi {
private:
    Fare** fares; //array of Fare ptrs
    unsigned int riders;
public:
    //omitted constructors
    bool pickupFare(Fare* f) {
        fares[riders] = f;
    }
}
```

Give the code to implement a `boardTaxi()` method of the Fare class by invoking the Taxi class `pickupFare` service.

5. [9 points] List the three C++ mechanisms necessary to achieve polymorphism:

1. _________________________________________________
2. _________________________________________________
3. _________________________________________________
For the next two questions, consider the following simple class hierarchy:

```cpp
#include <iostream>
#include <sstream>
#include <iomanip>
#include <string>
using namespace std;

//////////////////////////////////////////////////////////// Counter
class Counter {
    private:
        unsigned int Sec;
    public:
        Counter(unsigned int Seconds = 0);
        unsigned int Increment();
};

Counter::Counter(unsigned int Seconds) {
    Sec = Seconds;
}

unsigned int Counter::Increment() {
    Sec = (Sec + 1) % 60;
    if (Sec == 0)
        return 1;
    else
        return 0;
}

//////////////////////////////////////////////////////////// Timer
class Timer : public Counter {
    private:
        unsigned int Min;
    public:
        Timer(unsigned int Seconds = 0);
        unsigned int Increment();
    }; 

Timer::Timer(unsigned int Seconds) {
    Min = Seconds / 60;
    Sec = Seconds % 60;
}

unsigned int Timer::Increment() {
    unsigned int Carry = // invoke base Increment();
    Min = Min + Carry;
    return 0;
}
```
6. [10 points] One of the lines in the above class code contains a common error dealing with inheritance. Quote the statement with the error, briefly explain the error and then list two simple fixes for the problem.

7. [9 points] Is it possible to invoke the over-ridden Increment base class function from within the derived Timer class over-riding function? If your response is yes, then give the syntax for doing so. If your response is no, then briefly explain why not.
For the next three questions, consider the following poorly designed class hierarchy:

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

class Red {
private:
    int r;
public:
    Red(int i = 11): r(i) {} //{r = i;}
    int getR() const {return r;}
    virtual int sum() const = 0;
};

class Green : public Red {
private:
    int g;
public:
    Green(int i = 22): g(i) {} //{g = i;}
    int getG() const {return g;}
    int sum() const {return getR() + g; }
    int avg() const {return sum()/2;}
};

class Blue : public Green {
private:
    int b;
public:
    Blue(int i = 33): b(i) {} //{b = i;}
    int getB() const {return b;}
    int sum() const {return getR() + getG() + b; }
    virtual int RGB() const {return getR()*10000 + getG()*100 + b;}
};
```

For the next two questions, consider the following declaration:

```cpp
Green  g(44);
Green* gr = &g;
Blue*  bl = new Blue(66);
Green* gb = bl;
Red*   rd = gr;
```

8. [7 points] Can one make the following member invocation, (very briefly explain)? (If yes, give the output.)

```cpp
int i9 = gb->RGB();
cout << “i9 = “ << i9 << endl;
```
9. [7 points] Can one make the following member invocation, (very briefly explain)? (If yes, give the output.)

```cpp
int i10 = bl->avg();
cout << "i10 = " << i10 << endl;
```

10. [7 points] Can one make the following member invocation, (very briefly explain)? (If yes, give the output.)

```cpp
int i11 = rd->sum();
cout << "i11 = " << i11 << endl;
```
With the growth in specialized footwear in recent years software is needed for categorization. You have been selected to develop an inheritance hierarchy for the various types of footwear in order to minimize the code size for the system. The following various types of footwear and data must be modeled:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sandal</td>
<td>Slip-on shoe with a flat sole, open at toe and heel, held on with straps at toe and heel</td>
</tr>
<tr>
<td>Sneaker</td>
<td>Lace-up shoe with low synthetic upper and flat semi-rigid sole</td>
</tr>
<tr>
<td>Wingtip</td>
<td>Lace-up shoe with flat semi-rigid sole and decorative, pointed cap over toe</td>
</tr>
<tr>
<td>Duck boot</td>
<td>Waterproof lace-up shoe with high leather upper and flat lug sole</td>
</tr>
<tr>
<td>High heel</td>
<td>Slip-on shoe with open, elevated heel</td>
</tr>
<tr>
<td>Flip-flop</td>
<td>A flat sole that is held to the foot by straps over the toes</td>
</tr>
<tr>
<td>Hiker</td>
<td>Lace-up shoe with high leather upper and flat lug sole</td>
</tr>
<tr>
<td>Mule</td>
<td>Slip-on shoe with a closed toe and open heel, flat sole</td>
</tr>
</tbody>
</table>

11. [15 points] Considering only the possible types listed above, draw a sensible class relationship / inheritance hierarchy. You do not have to show any class members. (Hint: consider the information above and much less about any program being developed from the hierarchy. Be aware that base/intermediate types not explicitly listed above may need to be modeled for organizational purposes.)