Name: **ID:**

Concepts:

Problem 1:
If you do not explicitly define a copy constructor or assignment operator for your objects, what happens when you attempt to make copies or assign one object to another? In what cases must you define a copy constructor and assignment operator?

Problem 2:
What are the primary advantages of inheritance, provide a small programing example with two classes illustrating each point.

Name: **ID:**

Problem 3:
Suppose Child is a class derived from the class Parent, and the class Grandchild is a class derived from the class Child. This question is concerned with the constructors and destructors for the three classes Parent, Child, and Grandchild. When a constructor for the class Grandchild is invoked, what constructors are invoked and in what order? When the destructor for the class Grandchild is invoked, what destructors are invoked and in what order?

Problem 4:
Suppose you were tasked with developing a program that incorporated the following three objects: Car, Prius, and Engine. How would you organize these objects' relationship with one another? Why?

Problem 5:
Why can't we assign a base class object to a derived class variable?

Problem 6:
Suppose the base class and the derived class each have a member function with the same signature. When you have a pointer to a base class object and call a function member through the pointer, discuss what determines which function is actually called—the base class member function or the derived-class function.

Name: **ID:**

Program Problems

For the following programming problems make an attempt **without** using your computer this can be done on scratch paper if you prefer. After the initial attempt you may program the problems to check you work. Your final answer should be written on the assignment.

Problem 7:

This program is supposed to write 1 4 9 16 25 36 49 64 81 100 , but it probably does not. What is the problem with this program? (We're not asking you to propose a fix to the problem.)

```
int* computeSquares(int& n) {
      int arr[10];
      n = 10;
      for (int k = 0; k < n; k++)
            arr[k] = (k + 1) * (k + 1);
      return arr;
}

void f() {
      int junk[100];
      for (int k = 0; k < 100; k++)
            junk[k] = 123400000 + k;
}

int main() {
      int m;
      int* ptr = computeSquares(m);
      f();
      for (int i = 0; i < m; i++)
            cout << ptr[i] << ' ';
}
```

Name: **ID:**

Problem 8:
Write delete statements that correctly delete the following dynamically allocated entities.

```
int *p1 = new int[10];

int *p2[15];
for (int i = 0; i < 15; i++)
      p2[i] = new int[5];
int **p3 = new int*[5];
for (int i = 0; i < 5; i++)
      p3[i] = new int;
int *p4 = new int;
int *temp = p4;
p4 = p1;
p1 = temp;
```

Problem 9:
Consider the code fragment below. It is supposed to construct a 3x4 (3 rows 4 columns) 2d array of integers and set each value to zero. However, as given it does not. Add the proper dereferences (*) or references (&) to make this code work properly:

```
int   rows,   col1,   col2,   col3;

rows   = new    int[3];       // Create 3 pointers to columns
col1   = new    int[4];       // Create first row with 4 elements
col2   = new    int[4];       // Create second row with 4 elements
col3   = new    int[4];       // Create third row with 4 elements

(  rows + 0 ) =   col1[0];    // Point to first row
(  rows + 1 ) =   col2[0];    // Point to second row
(  rows + 2 ) =   col3[0];    // Point to third row

for (int i = 0; i<3; i++)
      for (int j = 0; j<3; j++)
            (   (   rows + i) + j) = 0 // rows[i][j] = 0;
```

Name: **ID:**

Problem 10:

For this problem, you will be asked to write some code to accomplish a particular task given the code fragment below. Each task may depend on the tasks that came before it. Your code must be syntactically correct.

```
class S {
public:
      S(int init) :m_num(init) {}
      S() :m_num(0) {}
      void set(int num) {m_num = num);}
      int get() { return m_num; }
private:
      int num;
};

S d1, d2(4), d3(-15);
S *sp1, **sp2, ***sp3;
```

Set sp1 to point to d1.

Using sp2 change the value of *num* in d1 to the value of *num* in d2 (you may not use d1).

Using sp3 make sp1 point to d3 (you may not use sp1).

What does the following code output? `cout<< **&*sp3;` If it is a value, state the value, if it is an address state the name of the variable to which the address belongs.

Name: **ID:**

Problem 11:
Consider the following program:

```
class A {
public:
      A() :m_msg("Apple") {}
      A(string msg) : m_msg(msg) {}
      virtual ~A() { message(); }
      void message() const {
            cout << m_msg << endl;
      }
private:
      string m_msg;
};
```

```
class B :public A {
public:
      B() :A("Orange") {}
      B(string msg) : A(msg), m_a(msg) {}
      void message() const {
            m_a.message();
      }
private:
      A m_a;
};
```

```
int main() {
      A *b1 = new B;
      B *b2 = new B;
      A *b3 = new B("Apple");
      b1->message();
      b2->message();
      (*b3).message();
      delete b1;
      delete b2;
      delete b3;
}
```

How many times will you see the word Apple in the output? ____

How about Orange? ____

Now make A's message() virtual, i.e.,

```
virtual void message() const;
```

How many times will you see the word Apple in the output? ____

How about Orange? ____

Name: **ID:**

Problem 12:
Consider the following program and generated output:

```
class Security {
public:
   Security(int id)
       :m_id(id), m_badge(id % 10) {}

   ~Security() {
       cout << "Security::~Security: "
              << m_id << endl;
   }
   // Get badge reference
   Badge & badge() {
       cout << "Security::badge: Ret ref "
              << endl;
       return m_badge;
   }
   // Get badge value
   Badge  badgeV() {
       cout << "Security::badge: Ret val "
              << endl;
       return m_badge;
   }
private:
   int m_id;
   Badge m_badge;
};
```

```
class Badge {
public:
    // Constructor
    Badge(int num) :m_num(num) {
       m_stuff = new int[6];
       for (int i = 0; i < 6; i++) {
              m_stuff[i] = num;
       }
    }
    // Destructor
    ~Badge() {
       cout << "Badge::~Badge: " << m_num
              << endl;
       delete[] m_stuff;
    }

    void setNum(int num) {
       m_num = num;
       for (int i = 0; i < 6; i++) {
              m_stuff[i] = num;
       }
    }

    void print() {
       cout << "Badge Num: ";
       for (int i = 0; i < 6; i++) {
              cout <<m_stuff[i];
       }
       cout << endl;
    }
private:
    int m_num;
    int *m_stuff;
};
```

```
int main() {
   Security s(11);
   s.badge().print();
   if (true) {
       Badge b = s.badge();
       b.setNum(2);
       b.print();

       s.badge().print();

       cout << "Main::Leaving if:"
              << endl;
   }
   cout << "Main::Leaving main:" << endl;
}
```

```
Output:
 Security::badge: Ret ref
 Badge Num: 111111
 Security::badge: Ret ref
 Badge Num: 222222
 Security::badge: Ret ref
 Badge Num: 222222
 Main::Leaving if:
 Badge::~Badge: 2
 Main::Leaving main:
 Security::~Security: 11
 Badge::~Badge: 1
```

Name: **ID:**

Question on the next page:

This program runs fine until the very end where we experience a runtime crash. Using a debugger we discover that there is an exception when calling the destructor for Badges at the line `delete[] m_stuff;`. What is causing this crash how would you improve the Badge class to prevent this from occurring?