<table>
<thead>
<tr>
<th>Problem</th>
<th>Possible</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

CSC 210, Exam Two
Section 004
7 February 1999

Name

Unity/Eos ID

(a) The exam contains 5 pages and 6 problems. Make sure your exam is complete.

(b) Read each problem carefully before working on it.

(c) If you write your answer on a different sheet, please indicate that you have done so.

(d) In code answers, comments are not necessary.

(e) Please put your Eos/Unity username at the top of each page.

On my honor: I affirm that I have not and will not use any external sources of assistance (notes, another student’s examination form, etc.) during this examination. I am the student whose name appears on this exam.

Signed: _________________________________

Date: _________________________________

Failure to sign this form legibly will result in a zero score on this exam.
1. [12 points] Answer each of the following true or false:

(a) In C++, the types “pointer to integer” and “pointer to array of integers” are indistinguishable.
true

(b) Suppose I declare:

```c
int a[100];
int* apt = &a;
```

Then the identifiers ‘a’ and ‘apt’ are interchangeable; i.e. wherever ‘a’ occurs I can substitute ‘apt’, and vica-versa.
false, cannot write a++

(c) friend functions can refer to this.
false, only members can refer to this

(d) Suppose that class D is derived from (inherits from) class B, then every instance of B is also considered an instance of D.
false, other way around

(e) Suppose that class D is derived from class B. When constructing an instance of class D, the constructor for class B is called first, then the constructor for class D.
true

(f) Public and private inheritance are both examples of “inheritance of the interface,” protected inheritance is not.
false, only public inheritance is inheritance of the interface

2. [16 points] Short answer:

(a) Suppose I have a class C. Whenever a variable of type C goes out of scope, C++ automatically calls the destructor for class C on that variable. Constructors are also called automatically. In what circumstances? In each circumstance, what is the signature (type, or template) of the constructor called?

(1) When a new automatic variable is declared without an explicit call to a constructor, the default constructor (with no arguments) is used. (2) When temporary values are necessary, copy constructor may be called (for example, call by value parameter passing); in this case, argument is of type C. (3) When a value of class D is used as an argument of a function that requires an operand of class C, a conversion constructor is called for class C; the argument is of class D.

(b) Generally, member functions are preferred over friend functions. Give two reasons why one might wish to define a function as a friend of a class C.

(1) Symmetric binary operators, so that conversion work properly. (2) Input and output stream operators, where the left-hand argument is of a different class.
(c) What is the essential difference between a member function and a “normal” function?

Member functions all always called with respect to a particular instance of a class; this is defined only for member functions.

(d) How are public and private inheritance similar? How are they different? Conceptually, what kind of (blank-a) relationship does each kind of inheritance create between classes?

They are similar in that they are both forms of re-use: members from the base class are automatically defined for the derived class. The difference is in the usage. Public inheritance is inheritance-of-the-interface; IS-A relationship. Private inheritance is inheritance-of-the-implementation; HAS-A relationship.

3. [18 points] Suppose that I have the following declarations:

```c
int i = 42;
int a[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
int* ipt;
```

Which of the following statements are erroneous? If the statement would cause a compiler error or warning mark it as ILLEGAL. If the statement would cause a runtime error, mark it as RUNTIME-ERROR. Otherwise, mark it as OK.

(a) int* jpt = i;
   ILLEGAL; i is an integer, not a pointer

(b) int* kpt = &i;
   OK

(c) ipt = a; *ipt = i;
   OK

(d) ipt = a; *ipt = &i;
   ILLEGAL; &i is a pointer, not an integer

(e) ipt = 0; *(ipt+1) = 42;
   RUNTIME-ERROR; attempt to dereference the first address after NULL (*1)

(f) delete 0;
   OK

(g) ipt = a; delete ipt;
   RUNTIME-ERROR; should only delete dynamically allocated variables, not automatic ones; also should use delete[]

(h) ipt = new int(0); delete ipt;
   OK

(i) ipt = new int[10]; ipt++; delete ipt;
   RUNTIME-ERROR; can only delete when pointer points to the beginning of dynamically allocated memory, not the second element; also should use delete[]
class Door {
public:
    Door(): shut_(true) { }
    bool isOpen() const { return !shut_; }
    void open() { shut_ = false; }
    void close() { shut_ = true; }
protected:
    bool shut_;}

class LockableDoor: public Door {
public:
    LockableDoor(): Door(), locked_(true)
    {
    }
    bool isLocked() const { return locked_; }
    void open() { if (!locked_) Door::open(); }
    void lock() { locked_ = true; }
    void unlock() { locked_ = false; }
protected:
    bool locked_;}

Write the definition of a class DeadBoltDoor, which is a type of lockable door. It is possible to lock the deadbolt door when the door is open. If the deadbolt is locked, then it is impossible to open the door when closed or to close the door when open.

This is an exercise from chapter 10. Answer was online.

```cpp
class DeadBoltDoor : public LockableDoor {
public:
    DeadBoltDoor() { }
    void close() { if (!locked_) shut_ = true; }
};
```
class Node {
  public:
    int   info_;  
    Node* link_; 

    Node(int n, Node* npt=NULL) : info_(n), link_(npt) {  }
    ~Node() { delete link_;  }
};

The List class is defined using a Node* field named head_. You may assume any representation of the List class that you like, i.e. with or without a dummy node. In the following questions you are asked to write List member functions. You may do these iteratively or recursively. You may define other List or Node helper functions.

5. [20 points] Write a List member function remove that takes an integer $n$ and destructively removes all occurrences of $n$ from the list. Suppose $L$ is the list $[1,3,5,3,6]$. Then after the call $L->remove(3), L$ should be $[1,5,6]$. 

You must state whether or not you assume the list to have a dummy node.
6. [20 points] Write a List member function psum with the following prototype:

```
List* List::psum(int n = 0) const;
```

The function should return a pointer to a new List which is the prefix sum of the original list. For example, if \( L \) is the list \([1, 3, 5, 3, 6]\), then \( L->psum() \) should return the list \([1, 4, 9, 12, 18]\). The \( i \)th element of \( L->psum() \) is the sum of the first \( i \) elements of \( L \).

The optional parameter \( n \) is a dummy argument which is added to the first element before continuing. Thus if \( K \) is the list \([5, 3, 6]\), then \( K->psum(4) \) should return the list \([9, 12, 18]\).

You must state whether or not you assume the list to have a dummy node.