Pop Quiz:

Given the struct Card as defined in ch. 12, and this definition:

```
    vector<Card> D;
```

Write code to print out all the Cards in D in reverse order.

(Assume D has been initialized to contain some Cards.)

One Possible Correct Answer:

```
for (int x = D.size()-1; x >= 0; x--) {
    D[x].print();
}
```

Exam II: Review Questions (with Key) available tonight or tomorrow.

Enumerated Types:

We've seen that many things can be encoded (represented) as integers, but this is not very readable.

What card is Card (2, 12)? This is hard to remember. Humans are not good at remembering this kind of thing; it's error prone.

Labeling Principle:

Avoid requiring people to remember a list more than a few items long. Instead, give the items meaningful names.

In C++ we have enumerated types, a convenient way of giving labels to small sets of things encoded as integers.

```
enum Suit { CLUBS, DIAMONDS, HEARTS, SPADES };

enum Rank { ACE=1, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE,
    TEN, JACK, QUEEN, KING };
```

Syntax:

```
enum <type name> { <value 1> [= <int>], <value 2> [= <int>], …, <value N>
[= <int>] };
```

(The square brackets surround optional parts.)

To give a name to a double:

```
const double PI = 3.14159265358979;
```

The type bool is a built-in enumerated type:

```
enum bool {false, true};
```

Arithmetic is not directly defined on enumerated types, but you can do it this way:

```
Rank R;
…
R = Rank( R+1 );
```

Switch Statements:

```
switch ( <expression> ) {
        <case 1>
        <case 2>
        ….
        <case N>
[ default: <statements> ]
}
```

A <case> has the following form:

```
        case <case constant>: <statements>
```

Normally the last statement is break;

Example:

```
Suit S;
…
switch (S) {
        case CLUBS: cout << "It's a club\n";
                    break;
```

```
    case DIAMONDS: cout << "a diamond\n";
                break;
    case HEARTS: cout << "heart\n";
                break;
    case SPADES: cout << "spade\n";
                break;
    default: cout << "illegal suit!\n";
}

switch (S) {
    case SPADES:
    case CLUBS:
    case DIAMONDS: ... // do the same for these suits
        break;
    case HEARTS: ... // handle hearts
        break;
    default: ... // shouldn't happen
}
```

Searching:

Sequential Search: You start somewhere (usually the beginning), and you look at each item until you find what you are looking for.

```
int find (const Card& card, const vector<Card>& deck) {
    for (int i = 0; i < deck.size(); i++) {
        if (equals (deck[i], card)) return i;
    }
    return -1;
}
```

A sequential search is often the best you can do without preprocessing the data or making assumptions about it.

What is the average time to find something in a vector by sequential search? Intuitively it takes N/2.

More accurately, if it's equally likely to be in any of the N positions, then the average time is:

$(1/N)\ [\ 1 + 2 + 3 + \ldots + N\ ] = (1/N)\ [N(N+1)/2] = (N+1)/2$,

which is approximately N/2 for large N. We call this an $\mathcal{O}$(N) algorithm, because the time is proportional to N.

Bisection Search:

If we have sorted the data, then we can divide what we're searching in half on each iteration, so what we have to search goes down 1/2, 1/4, 1/8, ….

How long does it take to search N sorted items? It takes time proportional $\log_2 N$.

We call this an $\mathcal{O}(\log N)$ algorithm.