Notes: 2009-04-21.

lvalues and rvalues:

lvalues are the sorts of things that can go on the left-hand side of an assignment operator, namely addresses of locations in memory.

rvalues are the sorts of things that can go on the right-hand side of an assignment oeprator, namely data values such as ints, doubles, chars, strings, etc., but also pointers.

Pointers:

Pointers allow you to manipulate the addresses of memory locations.

Two important uses:

(1) Pointers represent relationships between objects. For example, an organization chart shows who a person's supervisor is. In object-oriented programming, we might have a pointer from an employee object of a subordinate to the employee object for his or her supervisor.

(2) Pointers are efficient because they allow direct access from one object to another. For example, if we didn't have a pointer from an employee to their supervisor, but only had, for example, the supervisor's ID, we would have to search through the employee database to find someone with that ID. With a pointer, we can follow the link directly to the supervisor.

Reference "variables" (they are not variable!):
Reference variables basically declare aliases for variables.

```
SOMETYPE x;
....
SOMETYPE& ax = x; // declares ax to be an alias for x
```

Restrictions on reference variables:
(1) a ref var has to be initialized.
(2) a ref var cannot be reassigned.

So what are they good for? Not much!
Aliases can lead to confusing programs, because you have more than one name for the same location in memory.

```
x = 105;
...
ax = 30;
...
x++;
cout << x << endl;
```

If you are trying to figure out where x got clobbered, you might not think to look for places where ax is getting assigned.

Just because something is in C++ doesn't mean it's a good idea to use it!

Dynamic Memory Allocation:

Normally when you declare a variable in a function, it's allocated memory space when you enter the function, and that space is deallocated when you return from it. So you shouldn't return an address of a variable in that local storage, because its memory space will be reused. This is stack (LIFO) memory.

There is another area of memory, called the heap, and you can allocate pieces of this and keep using them until you're done with them. Can get allocated/ deallocated in any order.

To get a pointer to an unused area in memory sufficient to hold something of type T, possible initialized, use:

```
new T
new T (... constructor arguments ...)
```

This returns a pointer to a memory block containing a (possibly initialized) value or object of type T. E.g.:

```
T* pT = new T;
```

Later when you're done with it, you can recycle the memory by doing

```
delete pT;
```

This means: release the memory pointed to by pT and allow it to be used for other purposes.

A common error in C++ programming is releasing some storage and then going ahead and using it like you still have it. Some languages prevent this from happening, but in C++ you have to use disciplined programming techniques to make sure you don't make this mistake.

Pointers are "first class values" in C++. You can do the sorts of things with them that you can do with other first class values (numbers, characters, structs, etc.): you can have ptr variables, you can pass ptrs to functions, you can return ptrs from functions, there are some built in ptr operators, and so forth.

Ptr operators: *, &, ==, !=
(Also ++, --, but using them is not necessarily a good idea!
Ptr arithmetic is a common cause of hard-to-find errors, and even security breaches.)

Here is the example we discussed in class:

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

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

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

struct Card { … omitted – same as before … };

Card::Card (Suit s, Rank r) {
    suit = s;  rank = r;
}

void Card::print () const { … omitted – same as before … }
```

```
class employee { // employee record
public:
    string name;
    string title;
    double salary;
    employee* supv;  // pointer to supervisor
    vector<employee*> subordinate;

    // constructors
    employee () {
        name = title = "";
        supv = NULL;
    }

    employee (string N, string T, employee* S) {
        name = N;
        title = T;
        supv = S;
    }
};

int x;
int& rx = x; // this makes rx an alias for x

int main () {
    x = 43;
    rx = 17;
    cout << rx << ", " << x << endl;
    int someNumber = 12345;
    int* ptrSomeNumber = &someNumber;

    cout << "someNumber = " << someNumber << endl;
    cout << "ptrSomeNumber = " << ptrSomeNumber << endl;
    cout << "ptrSomeNumber points to " << *ptrSomeNumber << endl;

    int* px = &x;

    while (px != NULL) {
        cout << "Pointer px points to something\n";
        px = NULL;
    }
```

```
    cout << "Pointer px points to null, nothing, nada!\n";

    Card* cardPtr = new Card (DIAMONDS, ACE);

    // using parens and * operator:
    (*cardPtr).rank = Rank( (*cardPtr).rank + 1 );
    cout << "rank = " << (*cardPtr).rank << endl;
    (*cardPtr).print();

    // using -> abbreviation:
    cardPtr->rank = Rank( cardPtr->rank + 1 );
    cout << "rank = " << cardPtr->rank << endl;
    cardPtr->print();

    // make a little employee database
    employee* pAlice = new employee ("Alice", "pres", NULL); /* no
       supervisor */
    pAlice->salary = 150000;
    employee* pBob = new employee ("Bob", "VP mfg", pAlice);
    pBob->salary = 103040;
    employee* pCarol = new employee ("Carol", "VP sales", pAlice);
    pCarol->salary = 125000;
    employee* pJoe = new employee ("Joe", "welder", pBob);
    pJoe->salary = 50000;
    cout << "Joe the " << pJoe->title
        << " makes $" << pJoe->salary
        << " and his boss " << pJoe->supv->name
        << " makes $" << pJoe->supv->salary << endl;
    // Joe gets a new job and a raise
    pJoe->supv = pCarol; // new boss
    pJoe->title = "salesman"; // new job
    pJoe->salary *= 1.1; // 10% raise
    cout << "Joe the " << pJoe->title
        << " makes $" << pJoe->salary
        << " and his boss " << pJoe->supv->name
        << " makes $" << pJoe->supv->salary << endl;

    return 0;
}
```