Linked Lists

Copy Constructor/Assignment Review
Question: which of the following use the copy constructor and which use the assignment operator?

```
int main()
{
    PiNerd a(4), b(3);
    b = a;
}
```

Answer: assignment operator

```
int main()
{
    PiNerd c(5), d(c);
    PiNerd e = d;
}
```

Answer: copy constructor; defining e, which didn't exist before, and initializing it to d's values

```
PiNerd func(void)
{
    PiNerd g(15);
    return(g);
}
int main()
{
    PiNerd f = func();
}
```

Answer: copy constructor in the main() method

Classes and the "this" pointer
Every time you call a member function of an object, e.g.: a.addBill(5);

C++ invisibly rewrites your function call and passes in the variable's address!
`addBill(&a, 5);`

```
void Wallet::Init()
{
    num1s = num5s = 0;
}
void Init(Wallet *this)
{
    this->num1s = this->num5s = 0;
}
int main()
{
    Wallet a;
    a.Init(); //a.Init(&a);
    a.AddBill(5); //a.AddBill(&a, 5);
}
```

This is how it actually works under the hood....
C++ hides the "this pointer" from you to simplify things

Linked Lists

Any time you don't know how many items you'll need to store ahead of time, you use linked lists.

- Allows you to store an arbitrary number of items
- Makes it fast to insert a new item in the middle
- Makes it fast to delete an item from the middle

```
struct Chest
{
    string treasure;
    Chest* nextChest;
};
```

```
int main(void)
{
    //*first is linked list
    Chest *first; //gives us the location to our first clue
    Chest chest1, chest2, chest3; //nodes
    first = &chest1;
    chest1.treasure = "books";
    chest1.nextChest = &chest2;
    chest2.treasure = "shells";
    chest2.nextChest = &chest3;
    chest3.treasure = "cash";

}
```

This data structure is called a linked list.
We call each item in the linked list a "node".

Normally, we don't use local variables to create our linked list. Instead, we use dynamically allocated variables (and pointers)!

```
struct Chest
{
    string treasure;
    Chest* nextChest;
}
int main()
{
    Chest *first, *second, *third;
    first = new Chest;
    second = new Chest;
    third = new Chest;
    first->treasure = "books";
    first->nextChest = second;
```

```
    second->treasure = "shells";

    second->nextChest = third;

    third->treasure = "cash";

    third->nextChest = nullptr; //last node has a nullptr

    delete first;

    delete second;

    delete third;

}
```

Pointer to the first item is traditionally called the head pointer.
You can have your last node point up to the first node.

To allocate new nodes:
`Node *p = new Node;`
`Node *q = new Node;`
To change/access a node p's value:
`p->value = "blah";`
To make node p link to another node that at's address q:
`p->next = q;`
To get the address of the node after p:
`Node *r = p->next;`
To make node q a "terminal" node:
`q->next = nullptr;`
To free your nodes:
`delete p;`
`delete q;`

A Linked List Class!
Draw a diagram of a linked list with all the pointers so you can visualize what it looks like.

```
struct Node

{

    string value;

    Node *next;

}

class LinkedList

{

    public:

        LinkedList()

        {
```

```
    head = nullptr; //creates an empty linked list
}
void addToFront(string v)
{
    Node *p;
    p = new Node;
    p->value = v;
    p->next = head;
    head = p;
    //Link the head pointer to our new top node
}
void addToRear(string v);
{
    if(head == nullptr)
        addToFront(v);
    else
    {
        //Use a temp variable to traerse to the current last node of the list
        //Allocate a new node
        //Put value v in the node
        //Link the current last node to our new node
        //Link the last node to nullptr
        Node *p;
        p = head;
        while(p->next != nullptr)
            p = p->next;
        Node *n = new Node;
        n->value = v;
        p->next = n;
        n->next = nullptr;
    }
}
```

```
        void deleteItems(string v);
        bool findItem(string v);
        void printItems()
        {
            Node *p;
            p = head; //p points to 1st node
            while(p != nullptr) //while p points to a valid node
            {
                cout << p->value << endl;//print the value in the node
                p = p->next;//p = address of the next node
                //you can't use p++!!!!!!
            }
        }
        ~LinkedList();
    private:
        Node *head;
}
```