CSI 310: Lecture 12

Doubly Linked List Programming, needed for Proj 5.
Summary.....

What is a *variable* (synonyms: object, memory location, “cell”, “box for data”)?

What is a *pointer value* (synonyms: address, locator. “Reference” in Java)?
class Thing { public void memberFun(){...};
           ....  }

Thing mything;
mything = new Thing(..);  //copy a Java reference(pointer!)
mything.memberFun();     //call memberFun() ON the new Thing.
• C/C++/assembly/machine languages EXPOSE numerical pointer values/addresses.

• Java HIDES numerical pointer values/addresses and provides variables called Java reference variables, to hold and copy Java reference values.

• Java has only 2 types:
  (1) Primative type (roughly like C/C++’s but implementation independent)
  (2) Java “Reference type” which are pointers (to class objects only). (You cannot convert them to/from integers or any other type.)
Back to Proj 3-5 data structures...
A linked data structure consists of some `structure` type objects (variables) that contain some pointer type fields that hold addresses of `structure` type objects.

Such data structures can be virtually unlimited in size if the objects are **dynamically allocated**. In Java, they are ALWAYS dynamically allocated (by `new`).

The dynamic memory used shrinks as well as grows as needed, dynamically.
Dynamically Allocated Objects
(You need pointers also known as Java “references”.)

1. Dynamic objects are not declared. They are not named by identifiers. But they, like all variables, have memory addresses.

2. Dynamic objects are created during process execution. Unpredictable input data can determine if and how many dynamic variables are created.

3. **CSI201** Local lifetime variables are created when a function is called and are destroyed (storage recycled) when it returns.

But, a dynamic variable is created or destroyed only by an **explicitly coded operation** (**new** and **delete**) executed when the program runs. (**malloc** and **free** in C.)
Advice: Try to design and draft code for an algorithm idea in the general case.

Check that it works in the general case. Re-do design and draft code if not.

Then, check it for any boundary case(s). (E.G: Empty list and 1-item list)

If it fails for boundary cases, try to fix it and repeat these steps.

Have patience: Speed and elegance come from precise understanding of variables, data and (C/C++) language meaning; plus practice solving problems!
Given this doubly linked list,

how can you insert some Dogs between Two and Th3 ??
Given the original data structure, how can you change it to this one???
Remember, we want to insert **Dogs** between **Two** and **Th3**.

We NEED the addresses of the 2 circled nodes so we can modify their values.
class Dnode {
    Dnode pfore;
    // pfore == addr of the next Dnode if any.
    // pfore == NULL if this is the last dnode.

    Dnode pback;
    // pback == addr of the previous dnode if any. pback
    // pback == NULL if this is the first dnode.

    Picture data;
    // data == addr of a G&E Picture object.
    // data! = NULL. 
};
If \texttt{tpleft} contains the address of the \textcolor{red}{left} circled node, the following code calculates the address of the \textcolor{red}{right} circled node into \texttt{tpright}: \texttt{tpright} = \texttt{tpleft.pfore};

Alternatively, if we know the address of the \textcolor{red}{right} node, we can program: \texttt{tpleft} = \texttt{tpright.pback};
dnode  tpleft;  (Java declarations for
dnode  tpright;  these variables)

But, how can we obtain ANY one of these addrs?
(1) Starting from value of HEAD or TAIL traverse
     or search the linked list forward or backward..
(OR)
(2) Use the value in a "cursor" variable.
Now a new Dnode containing the addr. of the Dogs Picture is accessible via $\text{tpn}$ AND 2 variables hold the addresses of dnodes we will put it between.

**HOW CAN WE CONNECT IT?**

Relevant data str. is circled:
Here is a sample of Java code that sets up the data structure described by the diagram below.

In Project 5, an Dnode class inner to the LinearCollage class is a good idea.

class Dnode
{
    Dnode pfore;  //temporary node
    Dnode pback;  //pointer, is local
    Picture data;
}

Dnode tpn;  //tpn, for temporary node

tpn = new Dnode();
tpn.data = new Picture("Dogs.jpg")
Only 2 more steps remain to connect the new Dnode into the linked list:

(1) Set 2 link fields in the new Dnode:

```
tpn.fore = tpright;
tpn.pback = tpleft;
```

(2) Set 1 link field in each of the Dnodes it’s between:

```
tpleft.pfore = tpn;
tpright.pback = tpn;
```
The result of step 1,
\[ \text{tpn.} \text{.fore}=\text{tpn.} \text{.pback} \]
\[ = \text{tpn.} \text{.fore} \rightarrow \text{tpn.} \text{.pback} \]
The result of step 2,
\[\text{tpleft.pfore} = \text{tpn};\]
\[\text{tpright.pback} = \text{tpn};\]
The temporary pointer variables `tpleft`, `tpright`, `tpn` are not needed any more. (out of the picture)
A bigger picture after the changes:

Danger: pfore and pback might retain GARBAGE!!

pointer to (addr of) a node
pointer to (addr of) a char