Iterative Algorithm to Insert at Tail of a Linked List...

- Suppose you have a linked list at "head":

```
head
 +--------+     +--------+    -->+--------+    -->+--------+
 |        |---->| info1  |   /   | info2  |   /   | info3  |
 +--------+     |        |  /    |        |  /    |        |
                +--------+ /     +--------+ /     +--------+
                | ref1   |-      | ref2   |-      | null   |
                +--------+       +--------+       +--------+
```

  And you have a linked list **element** at "elem":

```
elem
 +--------+     +--------+
 |        |---->| infoX  |  (information has been filled in)
 +--------+     |        |
                +--------+
                | ????   |  (don't care, because we know there
                +--------+   is no "next" element)
```

- Here is an example of how the linked list is stored in memory:

```
Memory:
        +----------------+
 head:  |   8000         |--------------------------------+
        +----------------+                                |
        |                |                                |
          .......                                         |
          .......                                         |
        +----------------+                                |
 8000   |   info1        |   Linked list element 1   <-----+
        +----------------+
        | ref1=10000     |--------------------------------------+
        +----------------+                                      |
        |                |                                      |
        +----------------+                                      |
          .......                                               |
          .......                                               |
        +----------------+                                      |
 9000   |   info3        |   Linked list element 3 <----+       |
        +----------------+                              |       |
        | ref3=0  (null) |                              |       |
        +----------------+                              |       |
          .......                                       |       |
          .......                                       |       |
        +----------------+                              |       |
10000   |   info2        |   Linked list element 2      | <-----+
        +----------------+                              |
        | ref2=9000      |------------------------------+
        +----------------+
          .......
          .......
        +----------------+
 elem:  |   20000        |------------------------------+
        +----------------+                              |
          .......                                       |
          .......                                       |
        +----------------+                              |
```

```
20000  |  infoX         | New linked list element <---+
       +----------------+
       | refX=????      |
       +----------------+
```

- Notice that the "links" are memory addresses and the linked list can be found by tracing/following the addresses in the "linkage" field

- Inserting the element **pointed** by "elem" at the **tail** of the linked is realised by making the last element in the linked list point to the element **pointed** by "elem".

  This process is illustrated by the following diagram:

  Before insertion:

```
 head
  +---------+     +---------+    -->+---------+    -->+---------+
  |         |---->| info1   |   /   | info2   |   /   | info3   |
  +---------+     |         |  /    |         |  /    |         |
                  +---------+ /     +---------+ /     +---------+
                  | ref1    |-      | ref2    |-      | null    |
                  +---------+       +---------+       +---------+
 elem
  +---------+     +---------+
  |         |---->| infoX   |
  +---------+     |         |
                  +---------+
                  | ????    |
                  +---------+
```

  After insertion:

```
 head
  +---------+     +---------+    -->+---------+    -->+---------+
  |         |---->| info1   |   /   | info2   |   /   | info3   |
  +---------+     |         |  /    |         |  /    |         |
                  +---------+ /     +---------+ /     +---------+
                  | ref1    |-      | ref2    |-      | ref3    |--+
                  +---------+       +---------+       +---------+  |
 elem                                                              |
  +---------+     +---------+                                      |
  |         |---->| infoX   |<-------------------------------------+
  +---------+     |         |
                  +---------+
                  | null    |
                  +---------+
```

- Here is an example of actually happens within the computer memory when you "link" a new element to the list:

```
Memory:
       +----------------+
 head: |  8000          |-------------------------------+
       +----------------+                               |
       |                |                               |
         .......                                        |
         .......                                        |
       +----------------+                               |
 8000  |  info1         |  Linked list element 1  <-----+
       +----------------+
       | ref1=10000     |-------------------------------------+
       +----------------+                                     |
```

```
            |               |                                     |
            +---------------+                                     |
              .......                                             |
              .......                                             |
            +---------------+                                     |
      9000  |  info3        |  Linked list element 3 <----+       |
            +---------------+                             |       |
            | ref3=20000    |-----------------------------|---+   |
            +---------------+                             |   |   |
              .......                                     |   |   |
              .......                                     |   |   |
            +---------------+                             |   |   |
     10000  |  info2        |  Linked list element 2      | <-----+
            +---------------+                             |   |
            | ref2=9000     |-----------------------------+   |
            +---------------+                                 |
              .......                                         |
              .......                                         |
            +---------------+                                 |
     elem:  |  20000        |-----------------------------+   |
            +---------------+                             |   |
              .......                                     |   |
              .......                                     |   |
            +---------------+                             |   |
     20000  |  infoX        | New linked list element <---+---+
            +---------------+
            | refX=null     |
            +---------------+
```

- Notice the change of the address value in the last element of the linked list.

  This address (20000) is also the address of the list element pointed to by "elem"

- The other change is the linkage value of the new last element. It must contain ZERO (null) to indicate the end of the list.

- ***Iterative* algorithm to insert list element at *tail***

  - Linking a new list element to the end of a list can be achieve by an **iterative algorithm** as follows:

    We must make a distinction between inserting into an **empty** list and a **non-empty** list....

    - If the list at head is **empty**:

      ```
      elem.next = null;
      head = elem;  (new head)
      ```

    - If the list at head is **not empty**:

      ```
      ptr = "Find the last element in the list at head";
      ptr.next = elem;
      elem.next = null;  (head remains unchanged)
      ```

  - The following is a Java program containing an **Insert** function that inserts the "newelem" list element at the tail of a linked list that begins at "head":

```
static ListElement Insert(ListElement head, ListElement newelem)
{
   ListElement ptr;   // local variable to run down the list

   if (head == null)
   {  // Empty list, make "newelem" the first element
      newelem.next = null;       // Mark last element
      return(newelem);
   }
   else
   {  // find the last element
      ptr = head;
      while (ptr.next != null)
         ptr = ptr.next;
      ptr.next = newelem;        // Link new to the last element
      newelem.next = null;       // Mark last element
      return(head);
   }
}
```

This function will return the **head** of the new list (with the "newelem" inserted.

- Since we are **discussing *recursion***, I will **not** spend time discussing the ***iterative* algorithm**

  But I will **show you** the **code** -- you can study it on your own.

  The following program fragment shows how to use this function:

```
ptr = new ListElement();        // Create a new list element

.... (initialize the list element)

head = Insert(head, ptr);       // Insert new element into list
```

- Program in Java with **iterative algorithm** to link at tail of list: click here
- Program in M68000 assembler with **iterative algorithm** to link at tail of list: click here

---

- **A note on allocating memory space for a new list element**
  - In **Java:**

```
ListELement ptr;

ptr = new ListElement();        // Create a new list element
```

Recall that the `new` operator will:

- ***Reserve* (= allocate)** some (enough) **memory space** to store a `ListElement` typed **object**

- And **return** the ***location* (= address)** of the **reserved memory space**

- To **similate** the **effect** of the `new` operator, I have **written** the `malloc()` **function** that does the **following**:
  - The function **malloc** takes one **input parameter** in **D0**
    - **D0** = the **number of bytes** of **memory** you want to **reserve**

  - The **malloc** (memory allocate) function will **reserve** the amount of **bytes** of memory given in **D0**.
  - The function **malloc** then returns the **location** (address) of the start of reserved memory in register **A0**.

- **Example:**

```
Java:

 ListELement ptr;

 ptr = new ListElement();
 ptr.value = 1234;
```

```
M68000 equivalent:

 // ptr = new ListElement();
      move.l #8, d0                 reserve 8 byte for a List object
      jsr    malloc
      move.l a0, ptr                put address in ptr

 // ptr.value = 1234;
      movea.l  ptr, a0
      move.l  #1234, (a0)
```