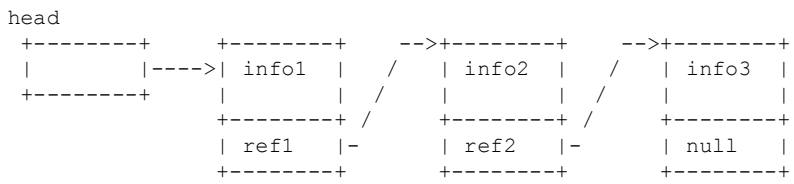
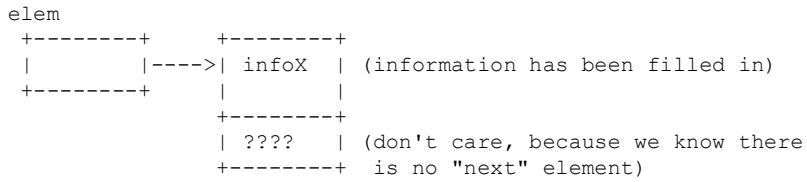
Recursive Algorithm to Insert at Tail of a Linked List...

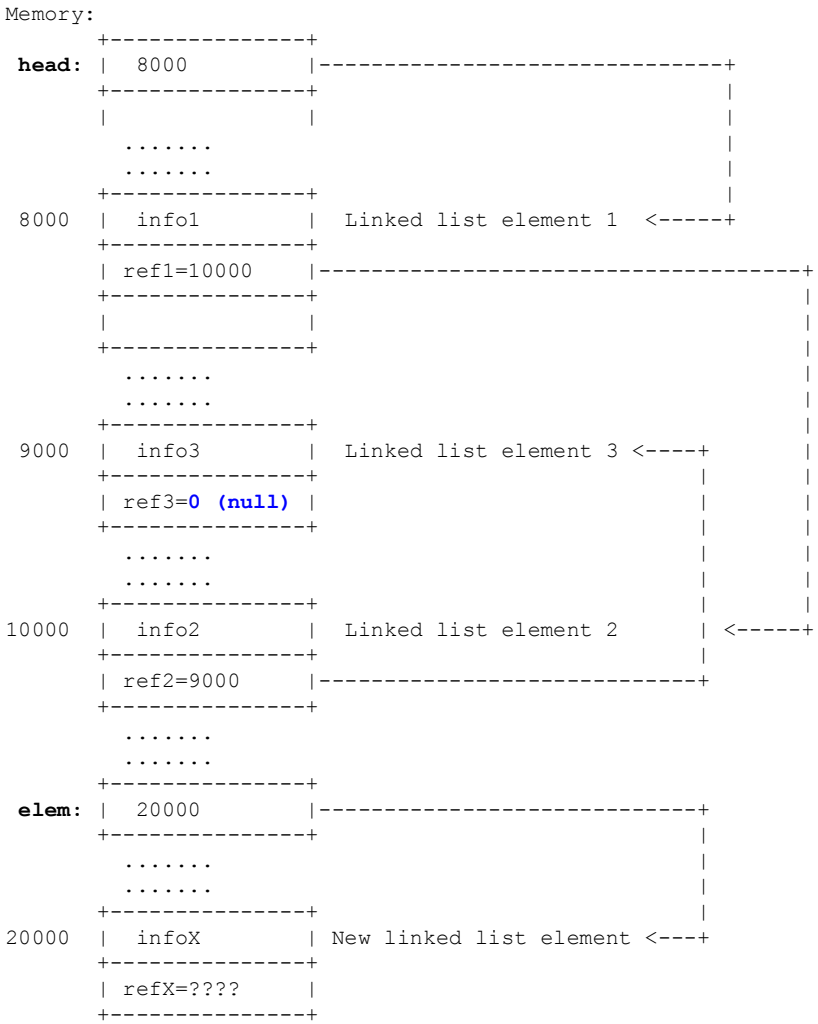
- Suppose you have a linked list at "head":



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



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



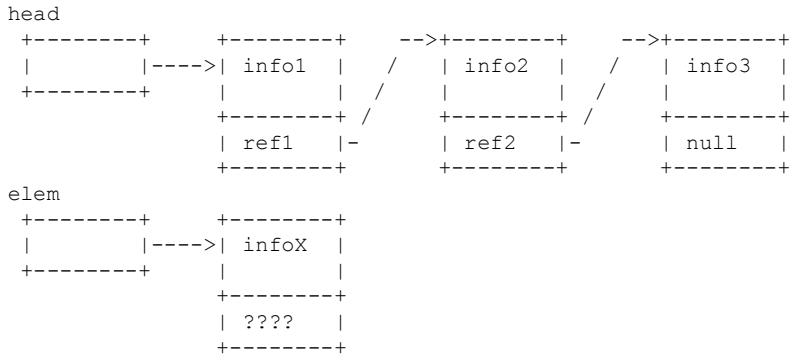
  - Notice that the "links" are memory addresses and the linked list can be found by tracing/following the addresses in the

"linkage" field

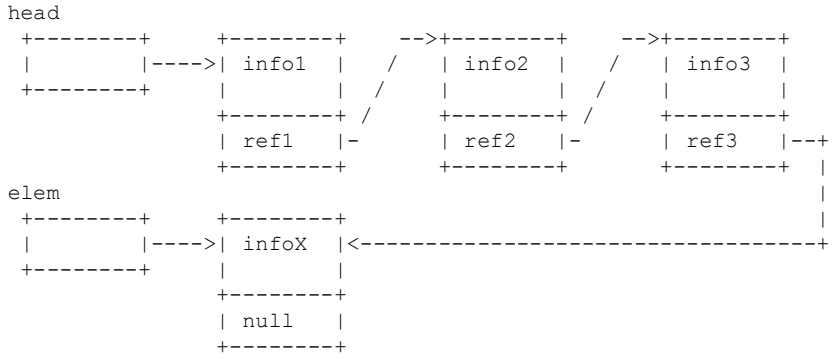
- Inserting the element **pointed** by "elem" at the **tail** of the linked is realised by making the last element in the linked list point to the element **pointed** by "elem".

  This process is illustrated by the following diagram:

  Before insertion:



  After insertion:



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

```
        +---------------+                                |
          .......                                        |
          .......                                        |
        +---------------+                                |
 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.

- ***Recursive* function to insert a list element at *tail***

  - Insertion at the tail of a list can be done **very easily** by using a **recursive algorithm**.
  - The following shows what the recursive algorithm must do to insert at the tail of a list: click here
  - In summary, to insert "elem" at the tail of the list starting at head, the recursive algorithm will:
    - If head == null, then we have the "base case" of the recursion. (A base case is the situation where we know the answer immediately.)

      In this case, the list is empty and we make "elem" the only element of the list.

      In this case, the head of the new list is "elem"

    - If head != null, then we recurse. (Not a base case)

      In this case, we insert "elem" at the tail of a "shorter" list that is formed by leaving the first element off the original list.

      The new list that is formed by inserting "elem" at the tail of the "shorter" list is linked AFTER the first element in the original list.

      In this case, the head of the list is unchanged.

  - The steps above can be expressed in the following function:

```
ListElement Insert(ListElement head, ListElement newelem)
{
   ListElement help;

   if head == null)
     { // Base case
       newelem.next = null;
       return(newelem);    // New elem is the head
     }
    else
     { // Recursion
```

```
        help = Insert(head.next, newelem);
                            // Insert elem in shorter list

        head.next = help;   // Link return list AFTER
                            // the first elem in orig. list
        return(head);       // head is unchanged
      }
  }
```

- **The Main function**
  - **In Java:**

```
   ptr = new ListElement();          // Make a new list element
                                     // Effect:
                                     //   1. reserve 8 bytes of memory
                                     //      to store a List oject
                                     //   2. return the address of the
                                     //      location of the reserved memory

   ptr.value = 1234;                 // Assignment some value

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

  - **In assembler:**

```
**** ptr = new ListElement();

   move.l #8, d0                   ; 8 byte used to store a List object
   jsr    malloc                   ; allocate (8 bytes) of memory
   move.l a0, ptr                  ; ptr = address of the allocated memory

**** ptr.value = 1234;

   move.l #1234, (a0)

**** head = Insert(head, ptr);

   move.l head, -(17)              ; pass head in a0
   move.l ptr, -(a7)               ; pass ptr in a1
   bsr    InsertList               ; call InsertList
   adda.l #8, a7                   ; pop parameters
   move.l d0, head                 ; head = return value
```

- ***Recursive* InsertList function in M68000**
  - **Recall** the ***recursive* insert algorithm** at **tail** written in **Java:**

```
  static ListElement Insert(ListElement head, ListElement newelem)
  {
      ListElement help;

      if head == null)
```

```
      {  // Base case
         newelem.next = null;
         return(newelem);     // New elem is the head
      }
      else
      {  // Recursion

         help = Insert(head.next, newelem);
                           // Insert elem in shorter list

         head.next = help;  // Link return list AFTER
                           // the first elem in orig. list
         return(head);      // head is unchanged
      }
   }
```

- **In assembler:**

```
InsertList:
*-------------------------------- Prelude
    move.l a6, -(a7)
    move.l a7, a6
    suba.l #4, a7
*--------------------------------
    move.l 12(a6), d0      d0 = head
    cmp.l  #0, d0          Test: head == null
    bne    ElsePart        Go to "ElsePart" if head != null

* ========================= Then part
    move.l 8(a6), a0       a0 = newelem
    move.l #0, 4(a0)       newelem.next = null;
    move.l 8(a6), d0       return(newelem) [in agreed location d0]

*-------------------------------- Postlude
    move.l a6, a7
    move.l (a7)+, a6
    rts
*--------------------------------


 ElsePart:
 * ======================== help = InsertList(head.next, newelem);
    move.l 12(a6), a0      (a0 = head)
    move.l 4(a0), -(a7)    (pass head.next)
    move.l 8(a6), -(a7)    (pass newelem)
    bsr  InsertList        Recursion: InsertList(head.next, newelem);
    adda.l #8, a7          Pop the 2 parameters from the stack

    move.l d0, -4(a6)      help = result of InsertList(head.next, newelem);

 * ======================== head.next = help;
    move.l 12(a6), a0      (a0 = head;)
    move.l -4(a6), 4(a0)   help is in -4(16), so this does: head.next = help;

 * ======================== return(head);
    move.l 12(a6), d0      return(head) [in agreed location d0]

*-------------------------------- Postlude
    move.l a6, a7
    move.l (a7)+, a6
    rts
```

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

NOTE: when you read the assembler program:

- the assembler program uses the function **malloc** (memory allocate) which is used to **reserve** a certain amount of memory.
- The function **malloc** takes one input parameter in D0
- The function **malloc** returns the **location** (address) of the start of reserved memory in register A0.
- The function **malloc** would be equivalent to the **new** operation in Java

---