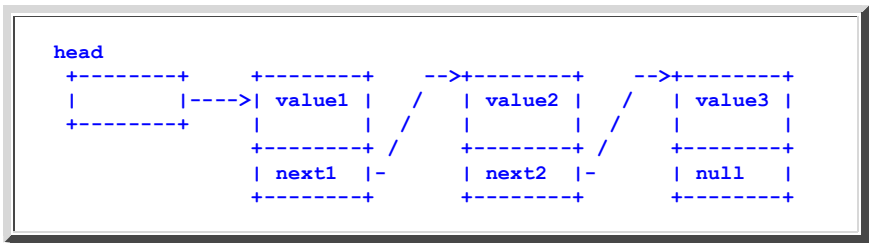
Insert at Start of a Linked List

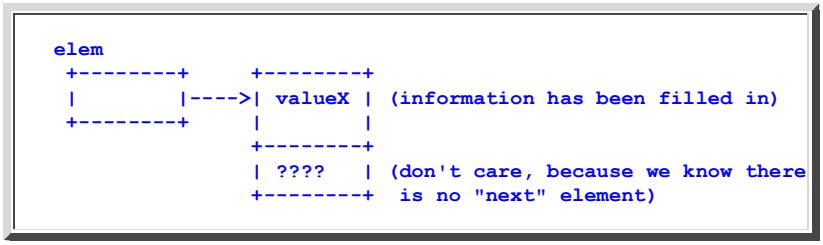
- **The List class used in example**
  - We will use the following **List** class:

```
public class List
{
   int  value;
   List next;
}
```

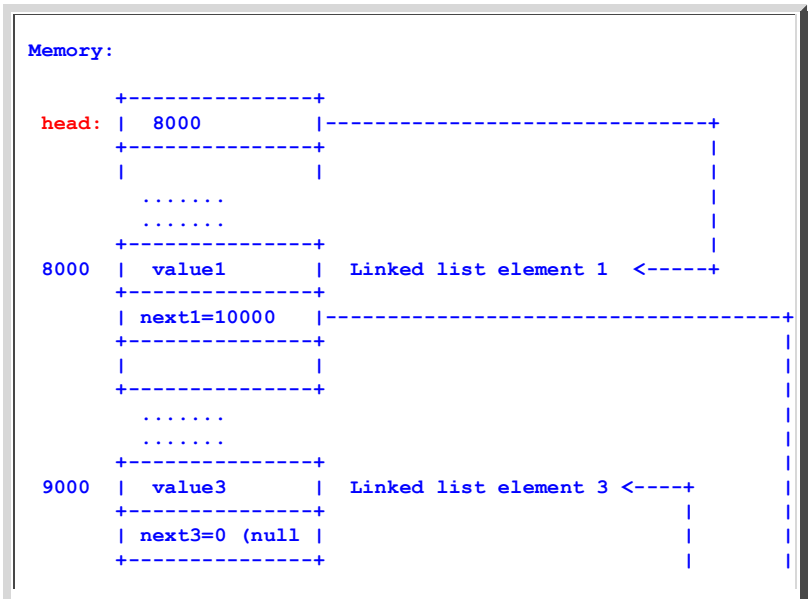
- **Inserting at the start (head) of a linked list**
  - Suppose you have a **linked list** at the List variable `head`:

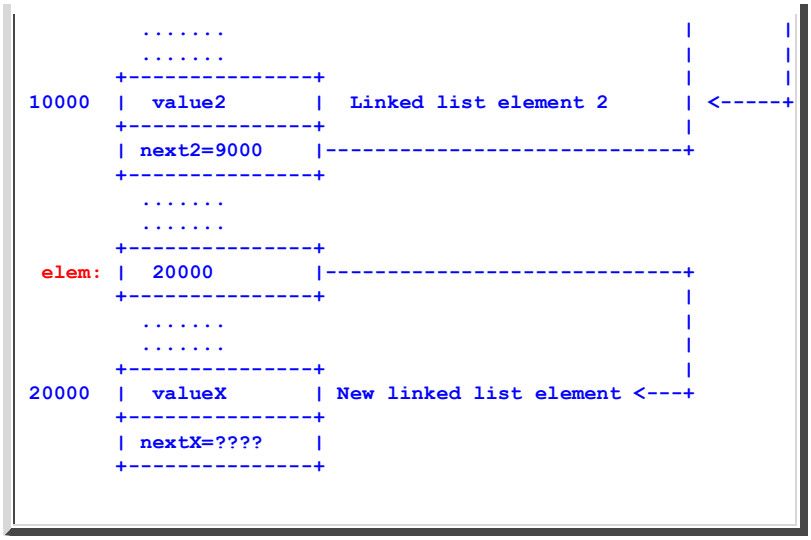


And you have a linked list **element** at "elem":



  - To make thing **more concrete**, here is **how** the **linked list** (at **head**) and the **list element** ***may*** be **stored**:





Note:

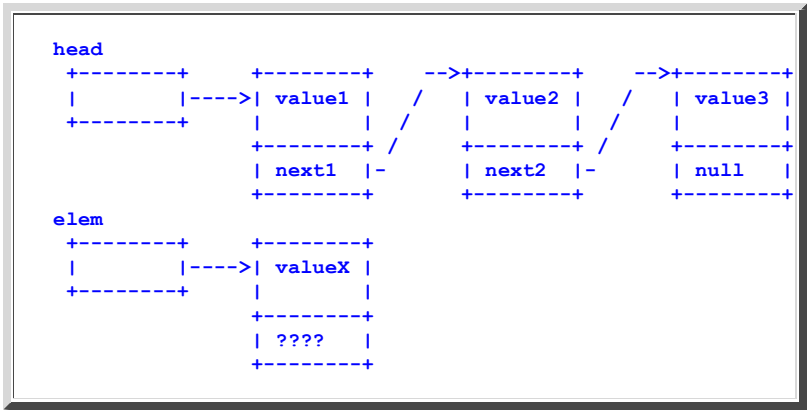
- You ***must*** use `head` to **access** the **list elements** !!!!

- **Inserting** the element **pointed** by the **List variable** `elem` at the ***start*** of the linked **means**:

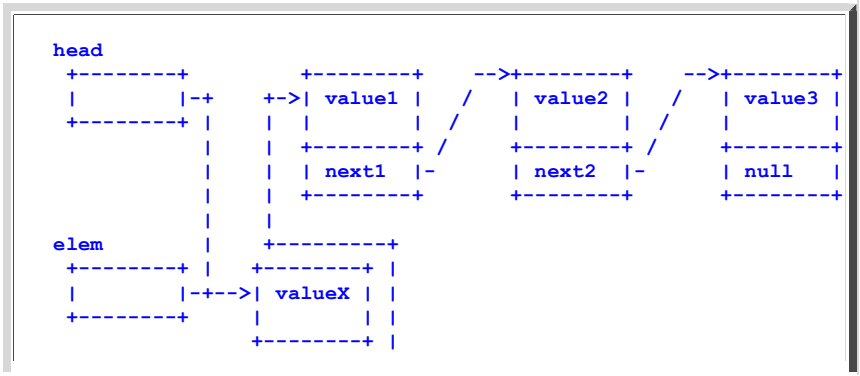
  - To make the element **pointed** by the **List variable** `elem` as the ***first*** **element** in the **linked list** at `head`

  Graphically:

  - **Before** the **insertion:**

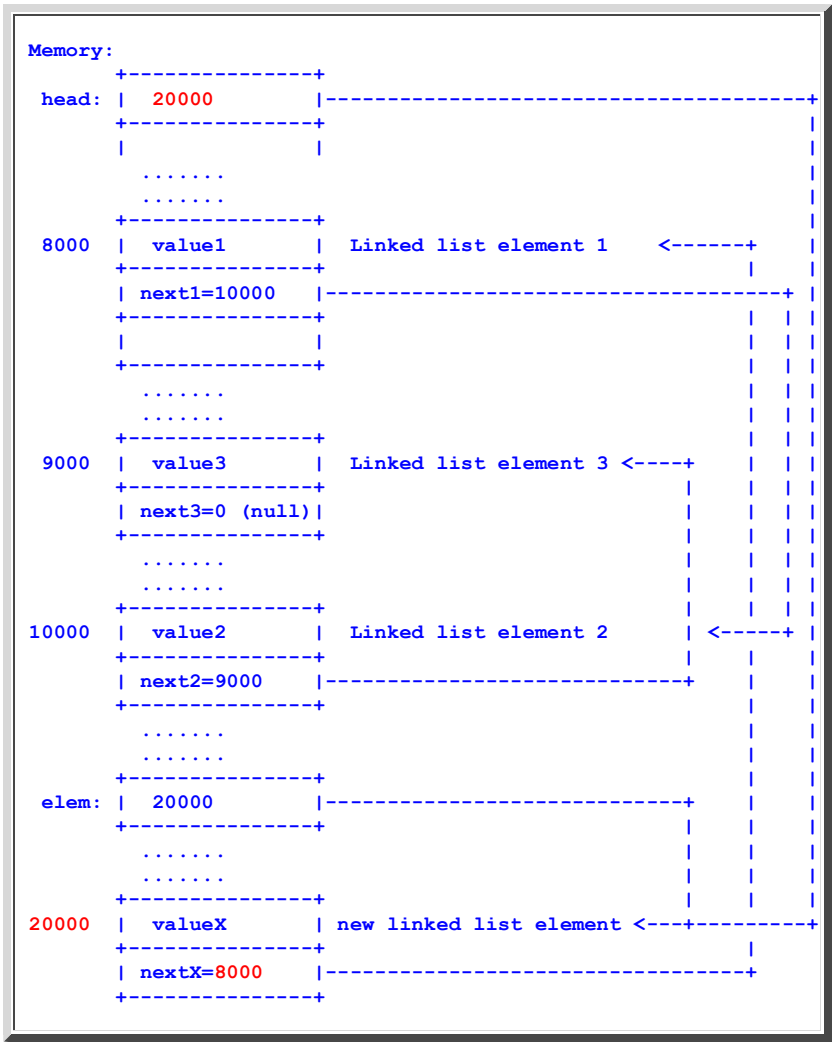


  - **After** the **insertion:**



```
                    | nextX  |-+
                    +--------+
```

- This is what will **actually** happen **inside** the **memory** using the **concrete example** **above**:



- **Review: insert at start of list in Java**
  - **Main method:**

```
List head;  // Assume head has a list already
List elem;

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

    ptr.value = 1234;                 // Assignment some value

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

- The **insert method**:

```
static List Insert(List head, List newelem)
{
   newelem.next = head;

   return(newelem);
}
```

- **Example Program:** (Demo above code) *Example*
  - Prog file: click here

How to run the program:

- **Right click** on link and **save** in a scratch directory
- To compile: `javac Linklist1.java`
- To run: `java Linklist1`

- **Allocating memory for a List element**
  - The **effect** of the `new` operator:
    - `new List( )` will:
      - **Reserve** (8 bytes of) **memory space** (in the ***heap*** to store one `List` object
      - **Return** the **address** of the ***first*** **reserved byte**
  - **Simulating** the `new` **operator**:
    - I have **provided** a **function** (called **malloc**) in the `cs255` **assembler library** to **simulate** the `new` **operator**
    - The `malloc( )` function:

```
Input parameter:  D0 = number of bytes that need to be allocated (i.e., reserved)

Output parameter: A0 = location (address) of the start of reserved memory
```

- **The main function in M68000 code**
  - **Main( )** in **Java**

```
ptr = new List();          // 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
```

Main( ) in **M68000 code**:

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

   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, -(a7)             ; pass head on stack
   move.l ptr, -(a7)              ; pass ptr on stack
   bsr    InsertList              ; call InsertList
   adda.l #8, a7                  ; clean up parameters
   move.l d0, head                ; head = return value
```

- **The insert at head method in M68000**
  - The **Insert()** function in **Java**:

```
  static List Insert(List head, List newelem)
  {
     newelem.next = head;

     return(newelem);
  }
```

  - The **Insert()** function in **M68000 code**:

```
   Stack Frame: (for your reference)

        +--------------+ <------ a6
        |  saved a6    | 0(a6)
        +--------------+
        | return addr  | 4(a6)
        +--------------+
        |    newelem   | 8(a6)
        +--------------+
        |    head      | 12(a6)
        +--------------+

-------------------------------------------------------------

InsertList:

* Prelude
```

```
        move.l  a6, -(a7)       * Save a6
        move.l  a7, a6          * Set up my own a6
        suba.l  #0, a7          * No local variable needed....

*       nexelem.next = head;
        move.l  12(a6),d0       * d0 = head
        movea.l 8(a6), a0       * a0 = newelem

        move.l  d0, 4(a0)       * newelem.next = head

*       return(newelem);
        move.l  8(a6), d0       * Put newelem in return location

* Postlude
        movea.l a6, a7          * De-allocate the local variables
        movea.l (a7)+, a6       * Restore a6 for caller

        rts
```

- **Example Program:** (Demo above code) *Example*
  - Prog file: **list1-stack.s** in `/home/cs255000/demo/asm/list1-stack.s`
