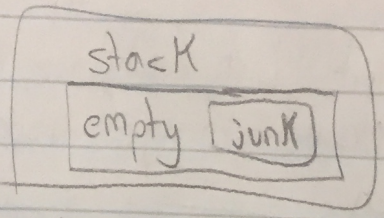
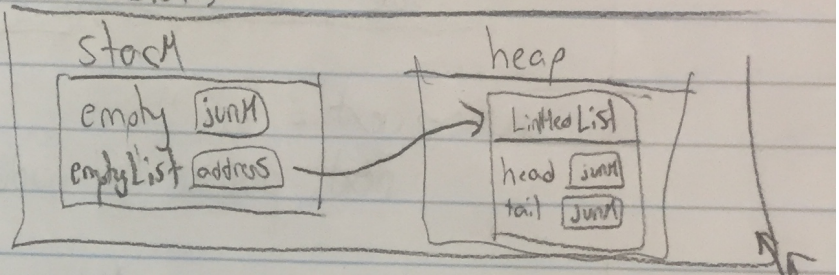
- `int empty[0] = { };`

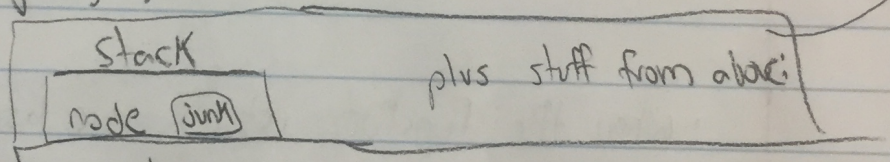


- `LinkedList * emptyList = arrayToLinkedList(empty, 0);`
  - `LinkedList *list = new LinkedList;`

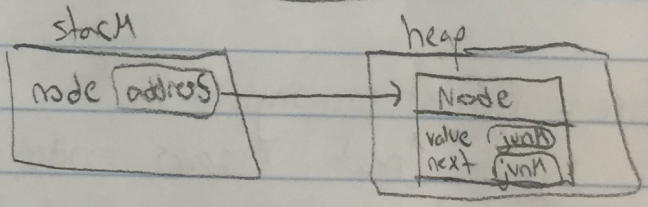


  - `list->head = NULL, list->tail = NULL;` same as above, but head, tail are NULL instead of junk.

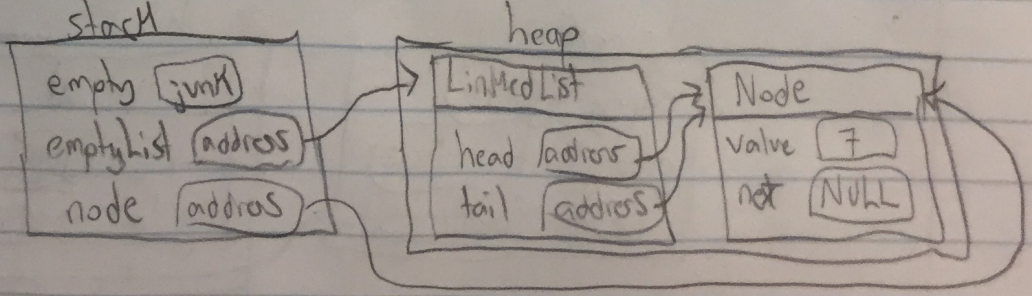
- `addIntToStartOfList(emptyList, 7)`
  - `Node node;`



  - `node = new Node;`

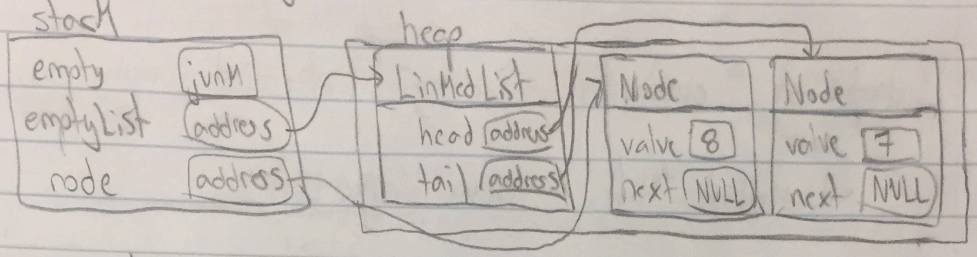


  - `node->data = value, node->next = NULL;` same as above, but junk values replaced w/ 7 and NULL.
  - `list->head = node, list->tail = node;`

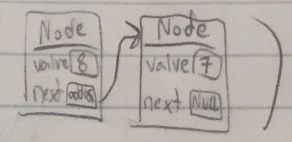


- after the function returns, we lose the pointer "node" on the stack.

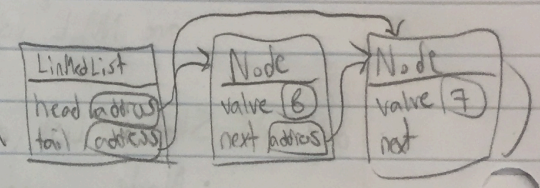
- addIntToStartOfList(emptyList, 8):
  - Node *node through node->next = NULL;



  - node->next = list->head
    - next pointer in node 8 points to node 7

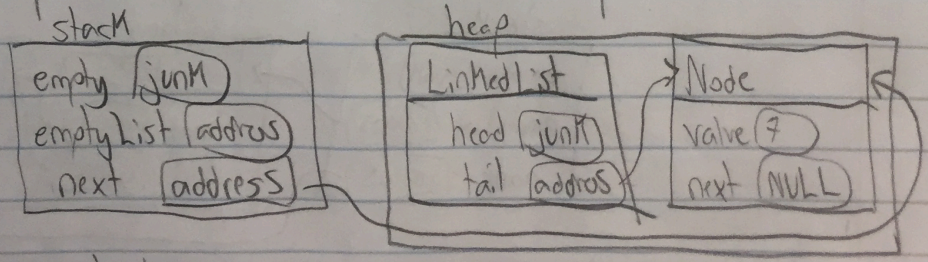


  - list->head = node
    - head points to node 8 now

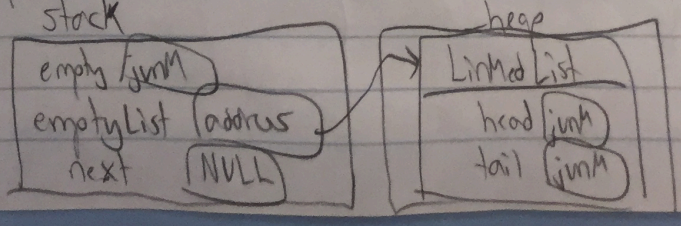


  - when the function returns, we lose "node" on the stack.

- freeLinkedList(emptyList)
  - Node *next creates node on stack (junk value)
  - for loop creates pointer p on the stack, pointing to node 8
  - next = p->next makes "next" point to node 7
  - delete p deletes node 8 from the heap.



  - next loop iteration removes node 7:



- delete list gets rid of empty list, clearing the heap: