Setup

- Your first task is to “reverse engineer” the code to understand more fully what it does. Trace through the mknodes routine in order to understand what it does. Draw a diagram of the heap created by mknodes, this will be required for your lab report.
- The list consists of nodes of the form,

  [next-ptr | data]

  where the nodes are obtained from the singly-linked list built by mknodes.

Write *new*

- Before starting, review the ARM register conventions to understand how you can accomplish this task. In particular:
  - *r0-r3*: Use for arguments, returns, and scratch work (unpreserved).
  - *r4-r10*: General purpose, but must be saved by the callee.
    - *r7*: Used for syscalls in addition to general-purpose use.
  - *r11-r15*: Special registers that should not be modified in this lab.
- Design the *new* subroutine, write it down on paper, and diagram what it does. These will be required for your lab report. When called, it removes the first node from the free list and returns it in *r0*, or returns NIL in *r0* if the free list is empty. The caller should check whether it has returned a node.
- Implement the design from above.

Insert

- Design an *insert* routine that takes a number N, a pointer to the linked list, and the list of free nodes. Insert should take the number, make a new node if one exists, insert that node into the proper position of the list, and insert N into the data portion of the node. <u>The insertion should be done so the list is ordered</u>. The function should return a pointer to the first node in the list and the modified list of free nodes.
- *New* must be called to make this new node, and if the node is NIL, there were no more free nodes to add. An “out of nodes” error should be printed to the console and the program should terminate.
  - To terminate a program:

```
mov r7, #0xe0
swi 0
mov r1, #9
mov r7, #0xee
swi 0
```

Print

- In order to help with debugging write a print routine that traverses the list after the data are inserted and prints out the data values. It should only take the list pointer as a parameter

Putting it Together

- Complete the main program to call these two routines. Print out a screen dump of your console window to show that the traversal is correct and turn it in with your labeled list diagram and a carefully documented program. Your starter code is not documented.
- Hand In:
    - Your diagrams for the setup and for writing new.
    - Your final program, with all working parts.
    - Screenshots to verify success.

Extra Credit

(In the history of Rieffel and Zach's CSC270, **no student has ever done this**)

- Write a *dispose* routine that takes a pointer to a node, a pointer to the linked list, and a pointer to the free list. It should put the node back into the free list and return the updated linked list and free list.
- Write a *delete* routine that takes an integer N, a pointer to the free list, and a pointer to the linked list. It should *dispose* of every node containing N. It should return the updated linked list and free list.
- Write a *cleanup* routine that returns all nodes in the linked list to the free list. It should return the updated free list, returning the linked list is trivial.