Linked Lists

Journal this tutorial
First, get the procedures I defined. Just `git pull` in the `ASTR2600_materials` directory and copy the new files into your tutorials directory.

You should have the following 2 procedures in your `Tutorial19_LinkedLists` directory:

```
node__DEFINE.pro
print_ll.pro
```

Then make sure you add the new directory to your repository and push it: `git add Tutoria19_LinkedLists`
`git commit`
`git push`
(obviously these commands won't work if you're in the wrong directory)

Examine these files - open them all in `gvim`. One nice way to do that is:
`gvim -p *pro`
Keep them open; we'll use them later.

First step, make a linked list by hand. You'll store each node in its own variable, which is not the normal way to work with linked lists but makes it easier to learn at first.

```
node1 = ptr_new( {Node,1,ptr_new()} )
node2 = ptr_new( {Node,5,ptr_new()} )
node3 = ptr_new( {Node,12,ptr_new()} )
node4 = ptr_new( {Node,19,ptr_new()} )
head = node1
(*node1).next = node2
(*node2).next = node3
(*node3).next = node4
```

Why are all of these pointers? Well, unfortunately, if you tried this approach:

```
node1 = {Node,1,ptr_new()}
node2 = {Node,5,ptr_new()}
head = ptr_new(node1)
node1.next = ptr_new(node2)
```

it would fail! This line:
`node1.next = ptr_new(node2)`
actually makes a copy of `node2` when it makes the new pointer!

Once you've completed making your linked list, run `print_ll,head`. You should see the data in the list printed out:

```
IDL> print_ll,head
       1
       5
      12
      19
```

In order to run this code, you'll either need to add `Tutorial19_LinkedLists` to your `!PATH` or work in the `Tutorial19` directory.
`IDL> !PATH=!PATH+":/full/path/to/Tutorial19_LinkedLists"`
(you have to replace `/full/path/to` with the correct full path)

New code

Create new *procedures*:
`add_tail.pro` and `add_head.pro`
`insert_node.pro` and `remove_node.pro`
and the new *function*:
`n_elements_ll.pro`

These procedures should all accept a *pointer to a node*, just like `print_ll`, and an integer argument. The pointer to a node should point to the first node in the linked list, i.e. it should be a `head`. For example, the `add_tail` definition should look like:
`pro add_tail,number,head`
and `head` is both an *input* and and *output* parameter

`n_elements_ll.pro`
will return the number of elements in the linked list.

`add_tail.pro`
will add a new element to the *end* of the linked list

`add_head.pro`
will add a new element to the *beginning* of the linked list

Here's an example node structure:
`[1|next] -> [2|next] -> [3|next] -> !null`

If you run `add_tail,4,head`
you should get
`[1|next] -> [2|next] -> [3|next] -> [4|next] -> !null`

If you then run `add_head,0,head`
you should get
`[0|next] -> [1|next] -> [2|next] -> [3|next] -> [4|next] -> !null`

If you run the command `remove_node,2,head`
the resulting structure should be
`[0|next] -> [1|next] -> [3|next] -> [4|next] -> !null`

And finally,
`print,n_elements_ll(head)`
should print `4`

Remember that the "standard" loop through a list should include something like:

```
while (node ne !null) do begin
    ; something (possibly an 'if X then break' statement)
    node = (*node).next
endwhile
```

Use `print_ll` as an example / template for looping.

Write `insert_node` using the code shown in Lecture 20. `remove_node` is a bit tricky. Refer to Lecture 20 and `insert_node` for help.

Some special things to consider when writing `remove_node`:
What will happen if there's only one node in the list? Two nodes?
What do you need to do differently for removing nodes from the front of the LL? The back?

Turn in your journal along with the code you've written. Make sure you test each piece of code you have written so that we can see it working in the journal.