Linked Lists

In this figure, N items (of whatever type) are stored in a linked-list manner.

For our purposes now, imagine that each item is a Student's first name.

So, the items might be

Mary, John, Jose, Feifei, Roshan, Ryosuke, Nzinga

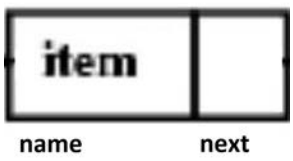
So, each item would be stored as a string, and it would be paired with another piece of information which would be the link (or pointer) to the next item.

The item and the link together would be called a NODE.

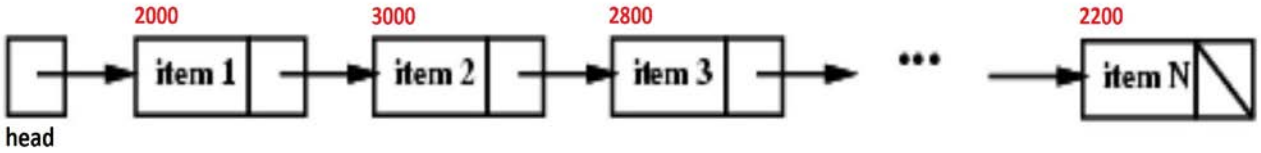
An appropriate way to declare a type for this data is:

```
struct node {
  char name[30];
  struct node *next;
};
```

Hence, a node can be imagined to look like:



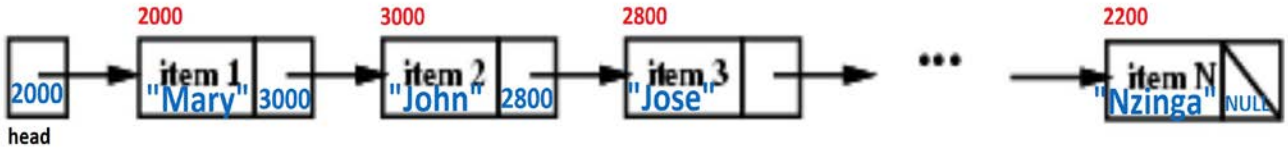
Now, suppose the red numbers in the figure are the actual addresses in memory of each such node in the list.



Note that the numbers are not in order, because each node was given whatever (wherever) memory was available when the node's request was made to the operating system. That is, the operating system (OS) gives each node the memory that it needs from the OS's list of available chunks based on what the OS has available at the time of the request. Because there are typically several processes simultaneously requesting memory, there is no way to tell where your memory will come from when you ask for it.

The point of Linked Lists is that they use memory only on an as-needed basis, and unlike arrays, they will not pre-reserve and waste any space. This is why linked lists are often said to employ DYNAMIC MEMORY ALLOCATION.

Now that the addresses of each node are visible to us, we can show what the contents of each part of the Linked List is (shown in blue). Note that the name part of each node is a string (which is an array of characters).



The Head is shown as holding the value of the address of the first node in the list. The final node in the list has its next field filled by the NULL address, which is a

special value that is employed to signify the end of the list. When the list is empty, the value of Head is NULL.

The malloc function

The request to the operating system for memory is made by using the malloc function (malloc stands for Memory Allocate). The malloc function is told the size of each node, and it returns the address of such a chunk of memory, where the chunk is big enough to hold one node. Note that once the node's type is declared, its size is known, but this will vary from program to program, because in some applications a node might be defined to contain much more information than merely the name and the next fields.

The call to malloc is specifically

```
struct  node  *p;
p  =  malloc (sizeof (struct  node));
```

and then **p** is the pointer to the new chunk of memory that has been provided for the node.

Some programmers write it as

```
p  =     ( struct node  *)     malloc (sizeof (struct  node));
```

where the **(struct node *)** is a way to CAST the obtained pointer into exactly the type that is needed. But, there is a debate whether it is necessary at all, with the main discussion in favor of putting the cast in being that it forces the writer to know and state clearly which type s/he meant to request.