# Side Talk: Implementation of The Linked List

Yufei Tao

ITEE
University of Queensland

In this side talk, we discuss one possible way of implementing the linked list. Remember that there are many ways to do so; and what is described next is merely an approach that the instructor likes.

We will strive not to constrain our description to any particular programming languages. For this reason, the code shown below will be "pesudo" by hiding some language-specific statements.

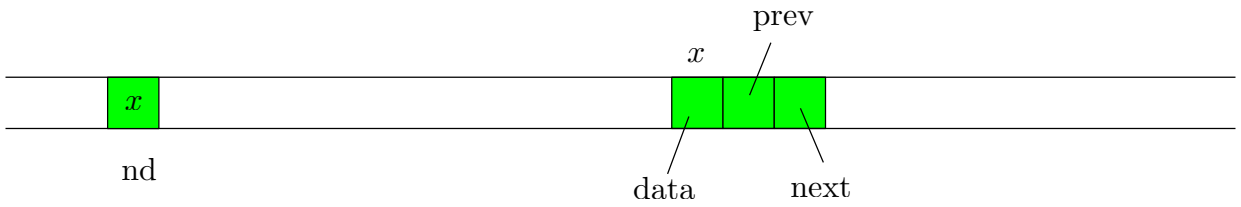
Defining a data type for each node in the linked list:

```
class lnode{
   int data;
   [Type of lnode Pointer] prev;
   [Type of lnode Pointer] next;
}
```

When you program executes

```
[Type of lnode Pointer] nd;
nd = ‘‘new’’ an lnode object
```

the operating system initiates consecutive memory cells to store the member variables of the `lnode` object:




The address of the first of those cells (in the above example, $x$) is stored in `nd`.

Defining a data type for the linked list itself:

```
class llist{
   [Type of lnode Pointer] head;
   [Type of lnode Pointer] tail;

   int insert(int e);
   int delete([Type of lnode Pointer] nd);

   ...
}
```

Note that at all times we remember the head and tail nodes.

Implementation of function `insert`:

```
int insert(int e){
   [assuming that the linked list is not empty]

   tailnd = the object referenced by tail;
   /* namely, tail stores the address of tailnd */

   ``new'' an lnode object; call it newnd;
   newnd.data = e;
   newnd.prev = tail;
   newnd.next = null;
   tailnd.next = address of newnd;
}
```

Implementation of function `delete`:

```
int delete([Type of lnode Pointer] nd){
   [assuming that nd is neither the first node nor the last]

   delnd = the node referenced by nd;
   prevnd = the object referenced by delnd.prev;
   nextnd = the object referenced by delnd.next;

   prevnd.next = delnd.next;
   nextnd.prev = delnd.prev;
   destroy node delnd
}
```

Think: How would you implement the stack? And the queue?