COMSC-165 Lecture Topic 11
Linked List Concepts

Reference
Wiki

Linked List Concepts
an object as a "node"
redistributing index array elements
  a "link" to the next node
    `tod* next;`
a "start pointer"
  points to first node
EOL marking
  link to zero (or NULL)
"node" insertion
  initializing the "start pointer"
"node" deletion
  checking the "start pointer"
vs an array: no capacity limit

The "start" Pointer
declaring an empty list: `tod* start = 0;`
adding a node: `start = &lunchtime;`

Node Design
add a "next pointer" at the end of a `struct`
struct syntax

```
struct tod
{
  int hour;
  int minute;
  int second;
  char descr[32]; // 31-characters plus null
  tod* next; // link
};
```

include link *last* (to permit brace initialization)
new `sizeof(tod)` value
  adds size of memory pointer (usu. 4 bytes)

The "next" Link
setting the link
  E.g.: `noon.next = &lunchtime;`
using the link

```
tod* p;
p = noon.next;
  or
p = p->next;
```

Building A List
the "classroom" analogy
workstations are "memory locations"
students are "objects"
nodes are students who can remember
  another student's workstation #
instructor has the "start pointer"
creating a linked list of student objects
zero "next" pointer marks list's end
traversing the list of student objects

List Traversing
visit each node in a list, using a loop
use: process or print each node in list
use: find a specific node in the list
start w/start pointer
end when `next` is zero
`while` loop to traverse
`for` loop to traverse: TWO versions:

```
1. tod* p; // the "p loop"
   for (p=start; p; p=p->next)
2. tod* p, *prev; // the "p and prev loop"
   for (p=start, prev=0; p; prev=p, p=p->next)
```

`const tod* p;` // a "read-only" pointer
a.k.a. "iteration" or "enumeration"

Variations And Applications
multi-indexed w/**`nextA`** and **`nextT`**
2-way lists w/**`prev`** w/ or wo/**`end`**
queue modeling -- FIFO and LIFO (stack)