COMSC-165 Topic 12 Lectures
Stacks as Linked Lists

Reference
Tutorial
Video

Arrays Of Unspecified Size
problem: cannot use pre-allocated elements
solution: use a linked list
problem: cannot use pre-allocated nodes
solution: use "dynamically-allocated" nodes
  allocate nodes *as needed*

When Memory Is Allocated
named nodes
  requires prior allocation
unnamed nodes
  allocate as needed

C++ Keywords
`new` keyword *allocates memory*
```
tod* t = new tod; // an unnamed node
```
`delete` keyword *deallocates memory*
```
delete t;
```
error checking -- usually ignored
```
if (!t)
if (t == NULL)
if (t == 0)
```

All-Purpose Insertion Code
```
tod* t = new tod;
```
after finding insertion point
  between `prev` and `p`
  ...*anywhere in the list*
```
t->next = p;
if (prev)
  prev->next = t;
else
  start = t;
```

All-Purpose Removal Code
`p` points to node to be deleted
`prev` points to "upstream" node
  ...*anywhere in the list*
```
if (prev)
  prev->next = p->next;
else
  start = p->next;
delete p;
```

Stacks
to make the coding easy...
  insert nodes at the start (p=start, prev=0)
  remove nodes at the start (p=start, prev=0)
  "last in, first out (LIFO)"
useful for playing-card games, etc.

Stack Insertion Code
insertion point: before `start` node
  `prev` is *zero*
  `p` is `start`
apply to all-purpose insertion code:
```
t->next = start;
start = t;
```

Stack Removal Code
```
tod* p = start; // node to delete
start = start->next; // reset start
delete p; // now delete it
```

How To Deallocate A Linked List
the "new" keyword *allocates* memory
  and returns its memory address
it has to be *deallocated*
  to avoid "memory leaks"
the "delete" keyword *deallocates* memory
  at a specified memory address

```
while (start) // zero when the list is empty
{
  tod* p = start; // node to delete
  start = start->next; // reset start
  delete p; // now delete it
}
```

Linked List Code Blocks
Click here for sample code
  to manage linked lists...
deleting nodes

C Syntax
`malloc()` in **alloc.h** *there is no calloc -- it's in cstdlib!*
`tod* t = (tod*)malloc(sizeof(tod));` *allocates memory*
  casting required from `void*`
error checking
```
if (!t)
if (t == 0)
```
`free(t);` *deallocates memory* (*casting not required*)

How to check for memory leaks:
Windows XP, in command-line mode: `systeminfo | find "Available Physical Memory"`
Mac "leaks":
https://developer.apple.com/library/mac/#documentation/Performance/Conceptual/ManagingMemory/Articles/FindingLeaks.html