COMSC-165 Lecture Topic 14
Queues as Linked Lists

Reference
Tutorial

Queues
first in, first out (FIFO)
linked list based:
insert at END
remove at START

Queue Insertion Code
find insertion point:
(traverse to find last node)
```
for (p=start, prev=0; p;
prev=p, p=p->next);
```
perform insertion: *(using "all-purpose" code)*
```
t->next = p;
if (prev)
  prev->next = t;
else
  start = t;
```

The "End" Pointer
extra pointer to remember last-added node
because next new node will go *after* it
```
tod* start = 0, *end = 0;
// empty list
```

Queue Insertion Code, With End
insertion point: after `end`
`prev` is `end`
`p` is *zero*
apply to all-purpose insertion code:
```
t->next = 0;
if (end)
  end->next = t
else
  start = t;
end = t; // remember to move the end pointer!
```

Merging Unsorted Linked Lists

```
tod* start1 = 0; // one list...
tod* start2 = 0; // ...another list
... // add nodes to both lists
tod* t = ... // node to move

// STEP 3: find end of 1st list
tod* p, *prev;
for(p=start1, prev=0; p; prev=p, p=p->next);

// STEP 4: attach 2nd list to end of first
t->next = p;
if (prev)
  prev->next = start2;
else
  start1 = start2; // first list empty
```

Merging Sorted Linked Lists

```
tod* start1 = 0; // one list...
tod* start2 = 0; // ...another list
... // add nodes to both lists, and sort them
tod* start3 = 0; // combined list
tod* end3 = 0; // for temporary use
if (!start2) // SPECIAL CASE: 1st list initially empty
  start3 = start1; // finished!
else if (!start1) // SPECIAL CASE: 2nd list initially empty
  start3 = start2; // finished!
else while(1) // GENERAL CASE: combine node-by-node
{
  if (!start1) // 1st list exhausted first
  {
    end3->next = start2; //...attach rest of 1st list
    break; //...and we're finished
  }
  if (!start2) // 2nd list exhausted first
  {
    end3->next = start1; //...attach rest of 2nd list
    break; //...and we're finished
  }
  // both lists still have nodes
  tod* t = start1; // assume take from 1st list
  if (compare(start1, start2) < 0) // check this assumption
    start1 = start1->next; // got it right! remove from 1st list
  else // assumed wrong!
  {
    t = start2; // take from 2nd list instead
    start2 = start2->next; // remove from 2nd list
  }
  // insert at end of combined list (queue w/end pointer)
  t->next = 0;
```

Queue Removal Code

removal point: `start`

```
tod* p = start;
start = start->next;
delete p;
if (!start) end = 0; //
```

new!

```
  if (end3)
    end3->next = t;
  else
    start3 = t;
  end3 = t;
}
```

Lab 15 Preview

animal.cpp

node building, persistence, recursive deallocation function

see: Animal program, labs 15 and 16

Animal.zip w/Windows 32-bit and 64-bit, and Mac versions
