Announcements

Today: Mentor evaluations, status reports
Thursday Extras: None next week
#define
CSC 161 next week
  Note: In-class quiz on schedule for next Wednesday
  If you have a scheduling issue, see me ASAP!
  On Monday, we'll discuss logistics for lab partners for next week

Clear progression of linked list activities
  1. Gain experience with access of lists
     Program given to create lists with cons
     Consider processing for car, second, etc
  2. Gain experience with traversing lists
     Program given to create lists
     Processing starts at front and progresses node-by-node
     Examples: count item, print last, change data in node
  3. Gain experience modifying list
     Example: insert node
       two cases: new first node, new node later
     Example: delete node—must update previous pointer
       two cases: remove first node, remove node later
     Example: move node to front
Still more practice with Linked Lists

Questions

Clicker questions
A procedure is supposed to initialize an int variable (in main) to 7. Two approaches are proposed:

```c
void initA (int x) { x = 7; }

int main () { int a; initA (a); }

void initB (int * p) { *p = 7; }

int main () { int b; initB (&b); }
```

Which of these approaches works?
A. Both initA and initB
B. initA only
C. initB only
D. Neither initA nor initB
A procedure is supposed to initialize a struct node variable (in main) to NULL. Two approaches are proposed:

```c
void initA (struct node * x)
{ x = NULL;
}

void initB (struct node * * p)
{ *p = NULL;
}

int main ()
{ struct node * a;
    initA (a);
}

int main ()
{ struct node * b;
    initB (&b);
}
```

Which of these approaches works?

A. Both initA and initB  
B. initA only  
C. initB only  
D. Neither initA nor initB
Assume B is initialized in main as follows

```
struct node {
    int data;
    struct node * next;
}
```

Consider the following procedure:
```
proc (struct node * p)
{
    p->data = 17;
}
```

After initialization, this code appears in main
```
proc (B);
struct node * ptr = B;
while (ptr != NULL)
{
    printf (" %d", (*ptr).data);
    ptr = ptr->next;
}
```

What is printed?

A. 2 7 1 8 2 8  
B. 17 7 1 8 2 8  
C. nothing printed  
D. compile or run-time error  
E. something else
Assume this declaration:
```c
struct node {
    int data;
    struct node * next;
};
```

Consider the following procedure:
```c
proc (struct node * p)
{
    p->data = 17;
    p = p->next;
}
```

Assume B is initialized in main as follows:
```
B → 2 7 1 8 2 8
```

After initialization, this code appears in main:
```c
proc (B);
struct node * ptr = B;
while (ptr != NULL)
{
    printf (" %d", (*ptr).data);
    ptr = ptr->next;
}
```

What is printed?

A. 2 7 1 8 2 8  
B. 17 7 1 8 2 8  
C. 7 1 8 2 8  
D. compile error or run-time  
E. something else
Consider the following procedure:

```c
proc (struct node ** p)
{
  (*p)->data = 17;
  *p = (*p)->next;
}
```

Assume B is initialized in main as follows

![Diagram showing the initialization of B with values 2, 7, 1, 8, 2, 8]

After initialization, this code appears in main

```c
proc (&B);
struct node * ptr = B;
while (ptr != NULL)
{
  printf (" %d", (*ptr).data);
  ptr = ptr->next;
}
```

What is printed?

A. 2 7 1 8 2 8  
B. 17 7 1 8 2 8  
C. 7 1 8 2 8  
D. compile error or run-time  
E. something else
Assume B is initialized in main as follows

```
B → 2 → 7 → 1 → 8 → 2 → 8
```

After initialization, this code appears in main

```
proc (B);
struct node * ptr = B;
while (ptr != NULL)
{ printf (" %d", (*ptr).data);
  ptr = ptr->next;
}
```

What is printed?

A. 17 7 1 8 2 8
B. 17 1 8 2 8
C. 7 1 8 2 8
D. compile error or run-time
E. something else
Conclusions:
• In analyzing and developing code for linked lists, draw box-
and-pointer diagrams early and often!
• To change a pointer variable within main using a
  procedure, use an address parameter
• To change values within nodes or pointers within nodes,
  either a value or address parameter can work