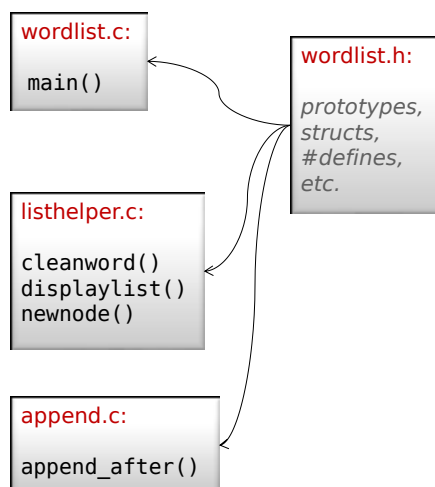


Lab: linked list of words

Overview: what the word-list program does

Program Structure

- **wordlist.c**
 - Function `main()` performs input, output, calls other functions
- **listhelper.c**
 - Miscellaneous functions used by `main()` to do I/O
- **append.c**
 - Function `append_after()` does the work of list-building
- **wordlist.h**
 - Contains shared information that is needed to "tie together" the other files



Overview: main()

- Open textfile for input
- "Clean up" the word
 - Remove non-alphabetic characters
 - » What if none left???
 - lowercase everything
- Close file handle, report results
 - Time the program
- "To do" at this point:
 - Add word to a linked list

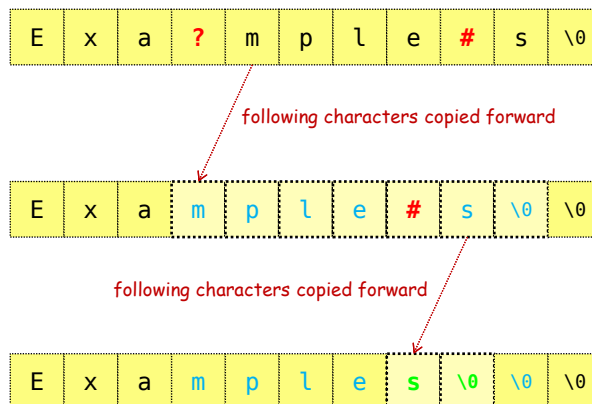
```

1 // Read words from a file and write any non-adjacent characters (puncts) found into
2 // 2019-10-21, but I changed to "word" in the output[]
3
4 #include "wordlist.h" // include system header files
5
6 #define BUFFER_SIZE 1024
7 #define BUFSIZE "512"
8
9 unsigned wordcount = 0; // GLOBAL variable is initialized to 0
10
11 int main(int argc, char **argv)
12 {
13     if (argc < 2) {
14         fprintf(stderr, "usage: %s <filename>\n", argv[0]);
15         return 1;
16     }
17
18     long unsigned clockstart = clock(); // begin timing the program
19
20     char wordbuffer[BUFFER_SIZE]; // hold words as they are read in
21     unsigned totalwords = 0;
22
23     FILE *fp = fopen(argv[1], "r");
24     while (!feof(fp) && !ferror(fp)) {
25         fread(wordbuffer, 1, BUFSIZE, fp);
26         if (strlen(wordbuffer) > 0) { // if there are any "word" left
27             totalwords++;
28             printf("%s\n", totalwords, wordbuffer);
29         }
30     }
31     fclose(fp);
32     printf("total words: %d", totalwords);
33     printf("\n\n");
34
35     // How long did it take to build the list?
36     long unsigned clockend = clock();
37     fprintf(stderr, "Read words/Build List: %.01f seconds\n",
38             (double)(clockend - clockstart) / (double)CLOCKS_PER_SEC);
39
40     return 0;
41 }

```

Overview: cleanword()

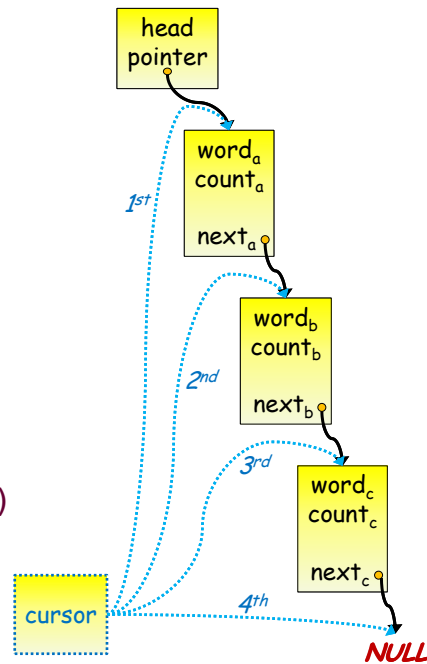
- For each character:
 - if (character is non-word), copy the rest of the word on top of it



Overview: displaylist()

Assuming the list has been created:

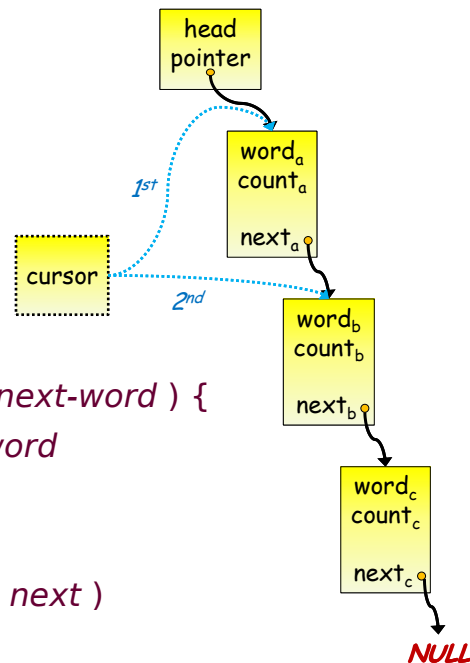
- "Walk the list" -
 - Cursor starts at head
 - if (cursor != NULL) {
 - print node contents*
 - displaylist(node's next)*
 - }



Overview: append_after()

Creating the list:

- "Walk the list" -
 - Cursor starts at head
 - if (*new-word precedes next-word*) {
 - insert between this-word and next-word*
 - } else {
 - append_after(node's next)*
 - }



"Walking" a Linked List Iteratively

- This for() loop starts at the head of a linked list, and follows the chain of "next" pointers

```
for (cursor = head; cursor != NULL; cursor = cursor->next)
    printf( "( %6.2lf, %6.2lf, %6.2lf) %5.1lf\t%6.2lf\n",
           cursor->x, cursor->y, cursor->z,
           cursor->weight, length(cursor) );
```

- Each element is processed in turn - here, it is simply displayed

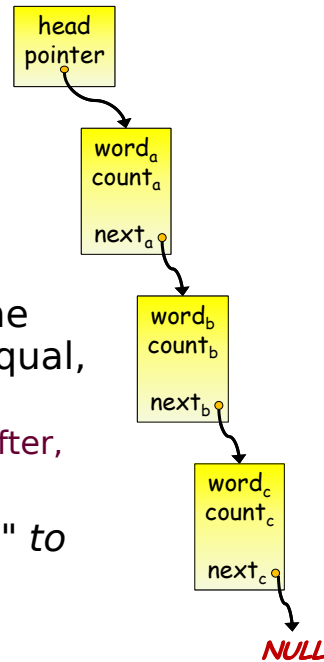
Binary Tree

Overview:

How the binary-tree assignment
processes files of words

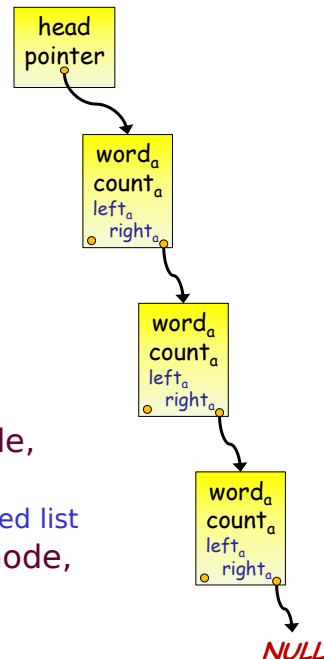
Consider an Ordered, Linked List...

- At each node, "previous" values are in previous nodes, while "later" values are in following nodes.
- Add a new node by walking the list until the current node is equal, or the next node is "later".
 - Add to current node or append after, as appropriate
- What if new node is "previous" to current node?
 - Only happens at the head???



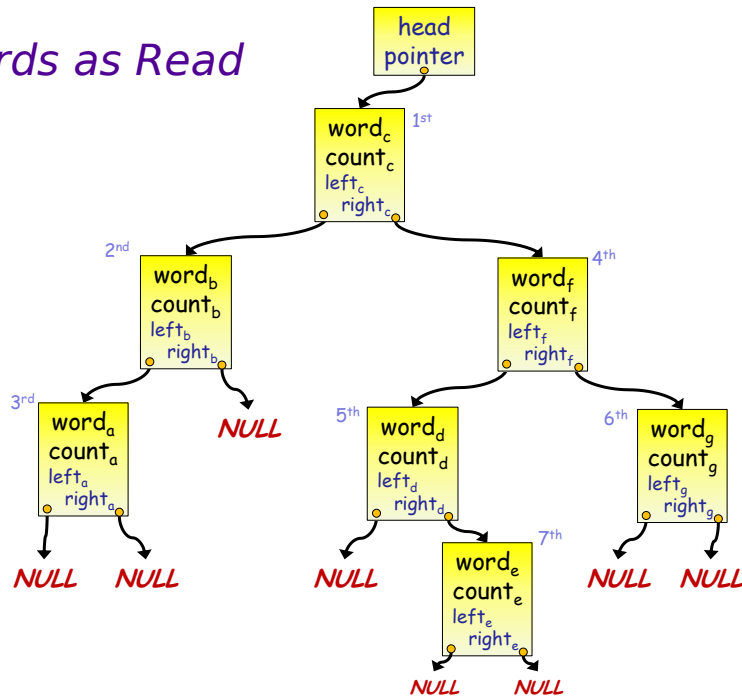
A new approach

- Each node has two pointers, "left" and "right" (or "before" and "after")
 - Instead of single "next" pointer
- To add a new node:
 - If new node equals current node, add to count
 - if new node "follows" current node, proceed to right/after pointer
 - As for the "next" pointer in the linked list
 - If new node "precedes" current node, proceed to left/before pointer



Add Words as Read

- 1) word_c
- first word
- 2) word_b
- add left
- 3) word_a
- add left,
add left
- 4) word_f
- add right
- 5) word_d
- add right,
add left
- 6) word_g
- add right,
add right
- 7) word_e
- add right,
add left,
add right



Display the Tree in Order

- Start at root node
- Pointed to by head pointer
- 1) Recursively descend left subtree
 - 2) Display current node
 - 3) Recursively descend right subtree

