Announcements

1. Homework 4 is out, as is the Gradescope.
2. Friday’s lecture will be recorded and posted to Canvas
   • It will also be posted to AggieVideo
3. I will be out of town on Thursday and Friday, so office hours on Friday are cancelled.
4. Would extending the due date for homework 3 be helpful?
Breaking a Line into Alphanumeric Words

• You can use `strtok()`
  • You have to exclude everything except alphanumerics
  • So the end of token characters has to be *everything* but alphanumerics!

• You can use `fscanf()` and a pattern
  • If you do this, the pattern should be only alphanumerics
  • You also have to go between alphanumeric patterns as `fscanf()` will not

• There’s a better way!
Breaking a Line into Alphanumeric Words

• Read one line at a time; for each line . . .
• If the first character is an alphanumeric:
  1. Advance until you do not see an alphanumeric, copying each character into an array; add a ‘\0’ after it
  2. Insert the word into your list
  3. Advance until you see an alphanumeric or ‘\0’
  4. If ‘\0’, go back up and read the next line
  5. If alphanumeric, go to 1
  6. If not, you will get to a ‘\0’; when you do, read the next line and go to 1
Rough Outline in Pseudocode

while (get there’s another line)
  p points to beginning
  while (p does not point to ‘\0’)
    while (p is not an alphanumeric and not ‘\0’) skip character, advance p
    if (p points to ‘\0’) break
while (p is an alphanumeric)
  copy *p into temp array, advance p
  put ‘\0’ at end of temparray
insert word in temparray into the list
In Pictures

• Input line “ def ght “ (not including quotes)

![Diagram showing buffer and temporary array](image_url)
In Pictures

• Input line “def ght” (not including quotes)

```
buf: [def \tght \0]
p: temparray
```
In Pictures

• Input line “def ght” (not including quotes)

```
 buf:  d e f \t g h t \0
```

```
 temparray:  d
```
• Input line "def ght" (not including quotes)
In Pictures

• Input line “ def ght “ (not including quotes)
In Pictures

• Input line “def ght” (not including quotes)
In Pictures

• Input line “def ght” (not including quotes)

```
buf: def \t ght \0

p
```

```
temparray: def \0
```

insert temparray (“def”)
In Pictures

- Input line "def ght" (not including quotes)
In Pictures

• Input line “def ght” (not including quotes)

buf

def \t ght \0

p

temparray

g h f \0
• Input line “def ght” (not including quotes)
In Pictures

• Input line “def ght” (not including quotes)

```
buf: def \t ght \0
```

```
temparray: g h t \0
```

May 31, 2023
In Pictures

- Input line "def ght" (not including quotes)

```
buf:  def  \t ght  \0
```

```
temparray:  ght  \0
```

p

insert temparray ("ght")
In Pictures

• Input line “def ght” (not including quotes)

```
buf   d   e   f   \t   g   h   t   \0
```

```
temparray  g   h   t   \0
```
**In Pictures**

- **Input line** "def ght" (not including quotes)

```
buf: def \t ght \0
temparray: ght \0
```

Go to next line
Linked List

• A list composed of instantiations of structures
  • One element is whatever is to be sorted (int, for us)
  • Another element is a pointer to the next element; NULL if none
Structure for This List

```
struct node {
    int num;
    struct node *next;
};
struct node *list;
```

This holds the integer that you read in

This holds the pointer to the next element in the linked list; it’s NULL if it’s at the end

This points to the first element of the list
Changing How Memory Is Allocated

- Now you can allocate memory one element (“node”) at a time
- Insertion at beginning is like this (see “linked.c”, ll. 72–76):
  - new->next = first;
  - list = new;
- Insertion in the middle between prev and succ is (see “linked.c”, ll. 78–97):
  - new->next = succ;
  - prev->next = new;
- Insertion at the end nomore of the list (same as above):
  - nomore->next = new;
Insertion

headList

1

5 -> 12 -> 23 -> 49
Insertion: At the Beginning of the List

```
1
```

First, change the pointer in the new node to point to the head of the list (where headList points; just copy the pointer)
Insertion: At the Beginning of the List

Next, change the pointer to the head of the list to point to the new node
Code for This

• new is a pointer to the new node, headList points to the head of the list
• First, make new point to the old head. of the list
  new->next = headList;
• Next, make the pointer to the head of the list point to new
  headList = new;
Insertion: In the Middle of the List

First, scan down the list until you reach the node before which the new node goes.

new node goes after this one
Insertion: In the Middle of the List

15

Change the pointer in the new node to point to the first node after where the new node is to go

headList

5  12  23  49

new node goes after this one
Insertion: In the Middle of the List

Next, have the pointer in the node *before* where the new node is to go point to the new node.

new node goes after this one
Code for This

• new is a pointer to the new node, headList points to the head of the list, and p is a pointer to node
• First, find the node that new goes after
  for (p = headList;
     p != NULL && p->next < new->next;
     p = p->next)
   /* do nothing */
• Next, change the pointer in new to point to the node after where this one goes
  new->next = p->next;
• Finally, make the node p points to point to new
  p->next = new;
Insertion: At the End of the List

First, scan down the list until you reach the end node.

new node goes after this one
Insertion: At the End of the List

Next, change the pointer in the end node to point to the new node

new node goes after this one
Code for This

- new is a pointer to the new node, headList points to the head of the list, and p is a pointer to node
- First, find the node at the end
  ```c
  for(p = headList;
      p != NULL && p->next != NULL;
      p = p->next)
  /* do nothing */;
  ```
- Next, change the pointer in what p points to to point to new
  ```c
  p->next = new;
  ```
- This may be an excess, but make sure new’s pointer field is NULL
  ```c
  new->next = NULL;
  ```
Sorting

- Function is:

  ```c
  void qsort(void *base, size_t nmemb, size_t size,
              int (*compar)(const void *, const void *));
  ```

- Here `compar` is a function that takes 2 pointers to elements of the array `base`, with `nmemb` members of size `size`.

- `compar` returns negative if first is less than second; 0 if the two are equal; and positive if the first is greater than the second.

- You supply `compar`.
Example *compar*

```c
int cmp(const void *x, const void *y)
{
    int *px, *py;
    px = (int *) x;
    py = (int *) y;

    return (*px - *py);
}
```
Calling `qsort`

```c
int arr[100]; /* array of integers to be sorted */
int narr;  /* number of integers in arr */
/* ... put random numbers into arr */
/* now sort them */
qusort(arr, narr, sizeof(int), (int (*)(const void *, const void *)) cmp);
```