class 6: Complex Data Structures +
Dynamically Allocated Memory

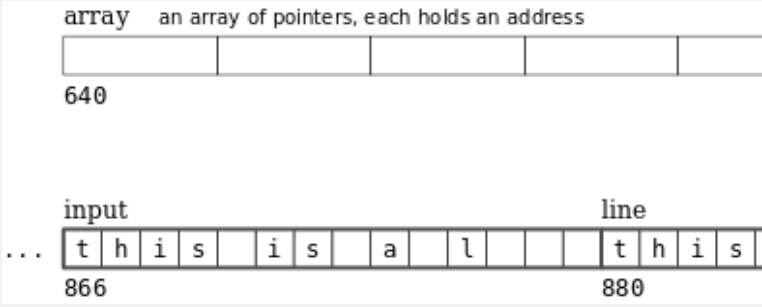
News:

hw3 due Sun at 1159pm
Section - Brandon
-- helps with homework
-- please attend and bring questions
Off Hrs Sat(3-4:30) and Sun(2-5) Bruce

hw4 out on Sun or Mon with 2 weeks to do it.

Warmup Question: What is wrong?

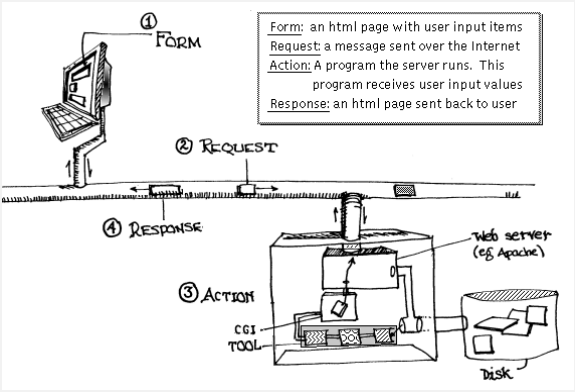
```
int main(void)
{
   char *array[SIZE];
   char input[SIZE];          /* a string for input */
   int counter = 0;
   char line[SIZE];           /* a string from input line */

   while ( fgets(input, SIZE, stdin) != NULL ) {
      sscanf(input, "%s", line);
      array[counter] = line;
      counter++;
   }
   for (int i = 0; i < counter; i++){
      printf("%s\n", array[i]);
   }
   return 0;
}
```



Class 6: Dynamic Memory Allocation and Management

Today's Project: Frequency Tables
Today's Data Structure: a Dictionary
Today's Ideas: dynamic memory allocation
Today's Functions: malloc, realloc, free

recall the big picture:



More Scripts and Web Interfaces: train-freq

Google shows how busy a store is by hour. Can we do the same thing for number of trains stopping at a station?

Question: How many trains stop at a station for each hour of the day? (Similar to *busiest-time* problem on hw0.)

Each entry in the sched file has a time as HH:MM, so we can extract the HH part and count how may times each hour appears.

a) Select the entries for a given station, dir, day
b) Cut out the hour portion of the time
c) Count the number of instances of each hour
(frequency)

Three parts: form, connector, tool:

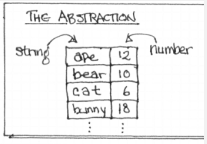
train-freq.html: get user request
train-freq1.cgi: extract request, call train-freq
train-freq: select rows, count hours

Version 2: use tt2ht for table output

A Common Problem: Frequency Tables

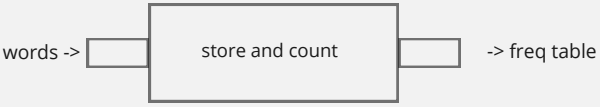
Computers are often used to compute frequency tables:

- trains by hour
- baby names by decade
- census: people by state
- site hits per day
- words/letters/trigrams in a document
  author identification, cryptography, language ident.
- melodic/harmonic patterns: composer ident

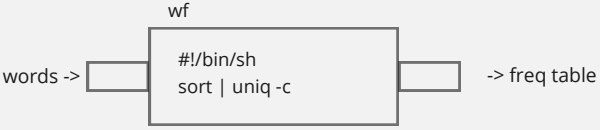


The Abstract Model:

input: a list of items: words, trigrams. names, chord changes
output: a list of items with counts



One Solution: sort | uniq -c



But: we only use sort because uniq -c only requires it.
Q: Can we count without sorting? It might be faster.

Another Solution: A dictionary

Storage System: Dictionary: string, count pairs

We need to store items and corresponding counts.
We need a storage system and operations for that system.

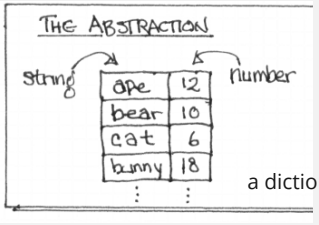
init_table()
in_table()
insert()
lookup()
update()
firstword()
nextword()

Another Solution: We Write Our Own wordfreq Program

Main Logic:

```
// count freqs
while( get_next_word )
 if ( word is in table )
   add 1 to count for word
 else
   add to table, count = 1
```

words ->



-> freq table

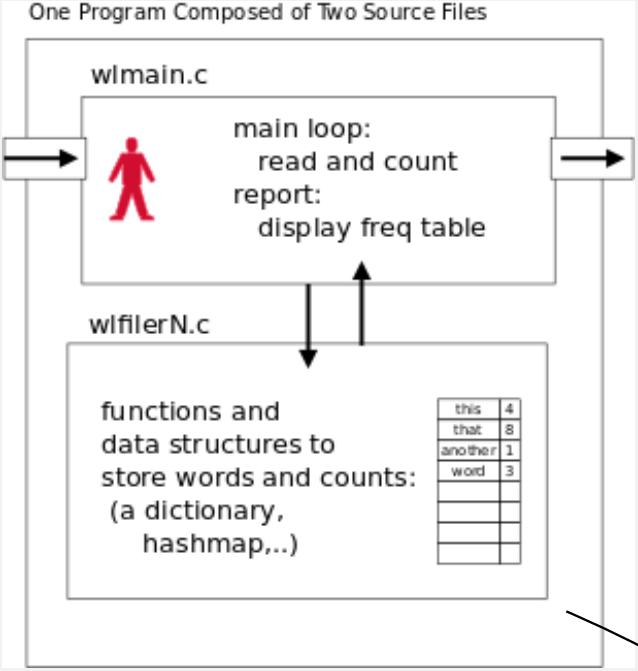
```
// report freqs
go to start of table
while( is another word)
  print word, its count
```

Storage System: Dictionary: string, count pairs

We need to store items and corresponding counts.
We need a storage system and operations for that system.

init_table()
in_table()
insert()
lookup()
update()
firstword()
nextword()

Program Structure: main file and helper file

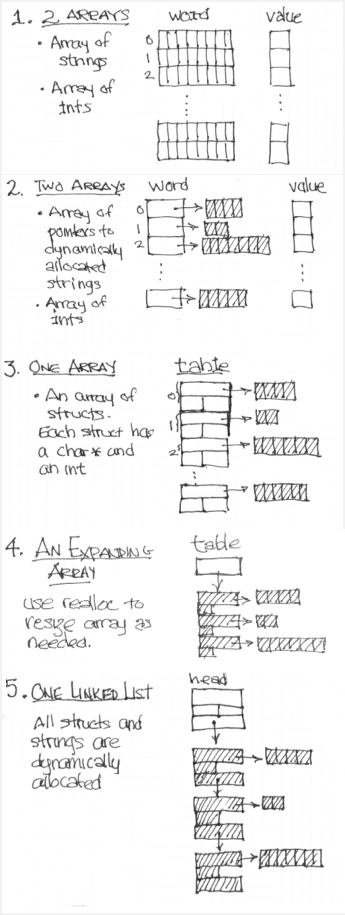


To Compile a Multi-file Program:

```
cc -Wall -Wextra wlmain.c wlfiler1.c -o wf1
```

Simply list all the source files on the command line.

Five Implementations of a table of words and counts:



For each of these models:

a. Define the storage
b. Write functions to:

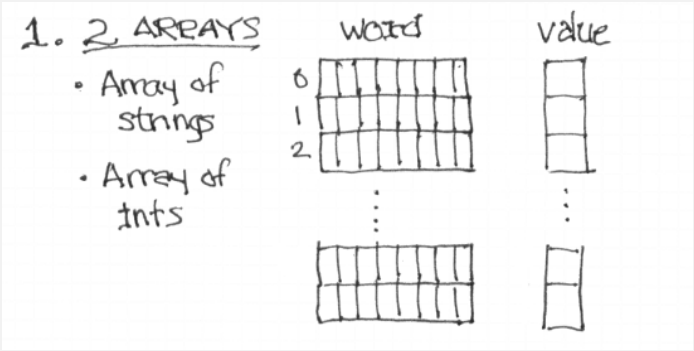
init_table()
in_table()
insert()
lookup()
update()

firstword()
nextword()

In doing this, we:

a. Learn about malloc(), realloc(), and free()
b. Use pointers some more

1. An array of char arrays (2D array) and an array of int



Defining the data structures:

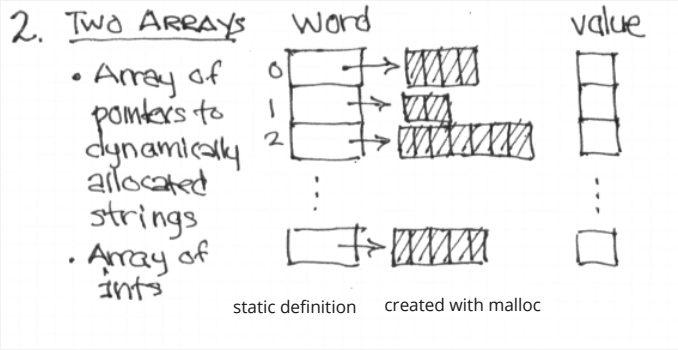
```
char word[MAXROWS][WLEN+1];
int value[MAXROWS];
int n_rows;                 // how many rows are occupied
int currrent_row;           // used for iterating through table
```

Note: static variables are local to file shared by all methods
Note: #include "wl.h" share declarations and #defines

Examine Code: Discuss Algorithms for functions

Pros and Cons of this solution
make a **todo list** for improvements

2 . Allocate Custom-Length Strings using malloc()



Q: How can we allocate memory when we know how long the string is? At runtime?

A: Ask the operating system by using **malloc**()
see next page for details --->
Note: malloc returns NULL on out-of-memory
Note: need to call **free**(addr) to reycle memory

look at mallocdemo.c

Defining the data structures:

```
char *word[MAXROWS];    // array of pointers to chars
                        // storage allocated at runtime
int value[MAXROWS];     // array of values
int n_rows;             // how many rows are occupied
int currrent_row;       // used for iterating through table
```

Examine Code: Discuss Algorithms for functions

Pros and Cons of this solution

malloc: Asking for memory when the program is running

Local Storage vs Dynamic Storage

Local Variables: created when a function starts and deallocated when function returns. These are stored on 'the stack' - a region of memory

Dynamic Variables: created when requested with malloc(storage_in_chars) and deallocated when program calls free(). These are stored in 'the heap' a different region of memory

malloc(amt) I need a new variable *now* !
args: size of requested block: in chars
rets: address of a block that size
or NULL for no more memory available
ex: p = malloc(100); // create block of 100 chars
p[0] = 'a'; p[1] = 'b'; // p points to that block
note: need to call **free(addr)** to recycle memory

see diagram below this page.

code: malloc1.c, mallocdemo.c

malloc() to store strings

```
p = malloc( strlen(s) + 1 );    // allocate space
strcpy(p, s);                   // copy str to new mem
```

<-- back to version 2

Allocating Memory when Program Runs

Allocate an array of chars:

```
char *newarr = malloc( 100 );
strcpy(newarr, "a string in dyn. mem.");
```

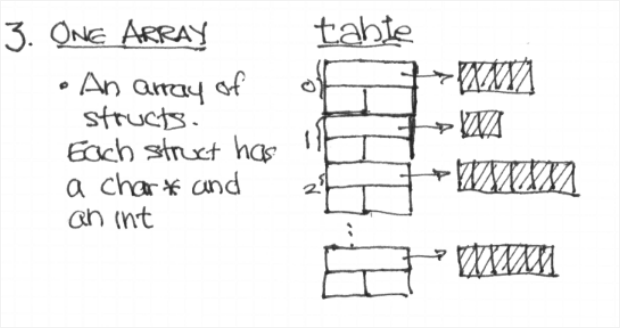
Allocate an array of ints:

```
int *listp;
listp = malloc( 20 * sizeof(int) );
lisp[0] = 10; listp[1] = 20; ...
```

Allocate a struct:

```
struct tstop *p;
p = malloc( sizeof(struct tstop) );
p->dir = 'i';
```

3 . One fixed-size array of structs



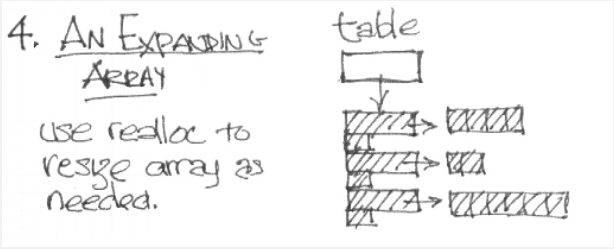
Defining the data structures:

```
struct entry table[MAXROWS];
int n_rows;                 // how many rows are occupied
int currrent_row;           // used for iterating through table
```

Examine Code: Discuss Algorithms for functions

Pros and Cons of this solution

4 . **realloc()** Growing an Array when you need more space



Problem: A fixed size array can fill up
Solution: an expanding array:

1 . use malloc to create an array
2 . if array fills up, use realloc to expand array

Defining the data structures:

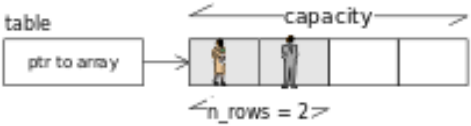
```
struct entry *table;
int capacity;          // capacity of array currently
int n_rows;            // how many rows are occupied
int currrent_row;      // used for iterating through table
```

Examine Code: Discuss Algorithms for functions

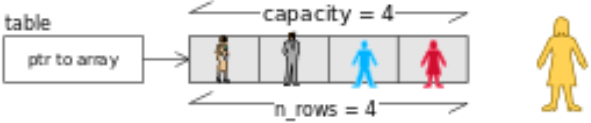
Pros and Cons of this solution

Algorithm for Growable Array

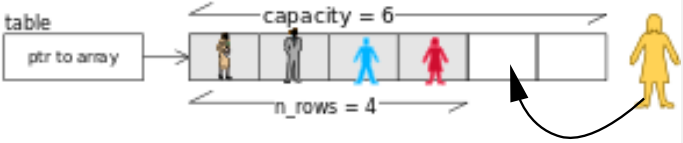
Add entries to array



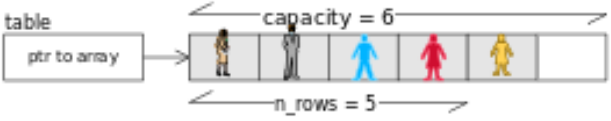
When occupied == capacity:



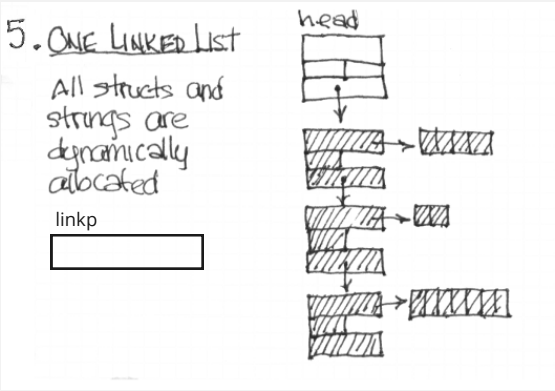
Then realloc memory and update capacity



Then continue adding to expanded array.



5 . A Linked List



Problem: realloc() may have to move the array
Solution: Create each entry as needed, insert in list

1 . use malloc to create each entry
2 . move pointers to insert entry in list

Defining the data structures:
struct link { char *word; int value; struct link *next; };
struct link head;
struct link *current_link;

Examine Code: Discuss Algorithms for functions

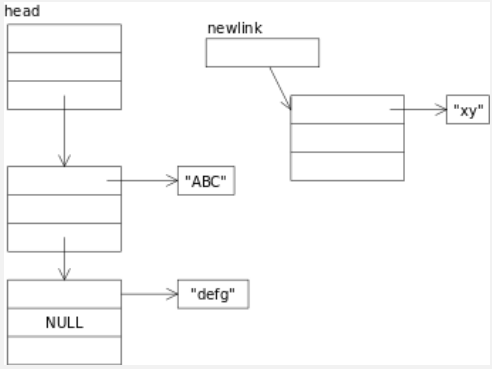
Pros and Cons of this solution

Inserting a link at front of a linked list

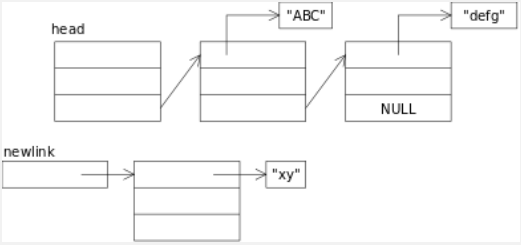
To add a link to the front of a iinked list"

1 . make new link point to current first link

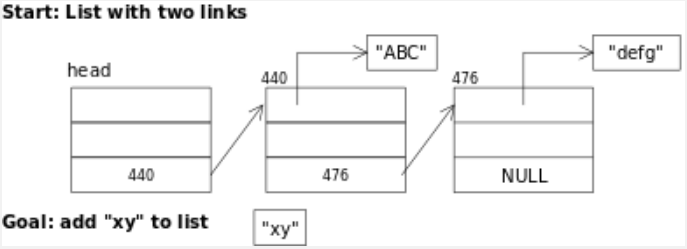
2 . make head link point to the new link



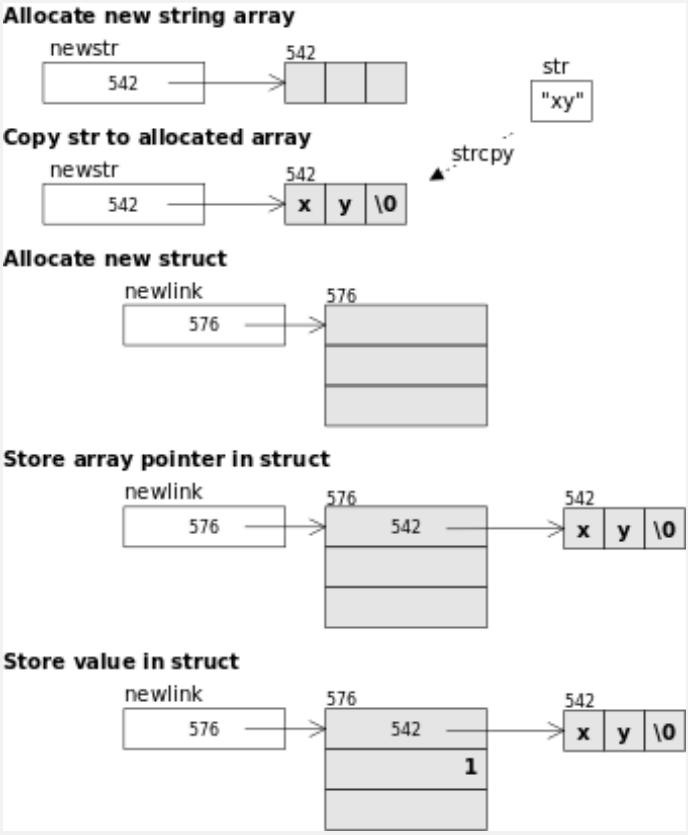
Linked lists are often drawn horizontally:



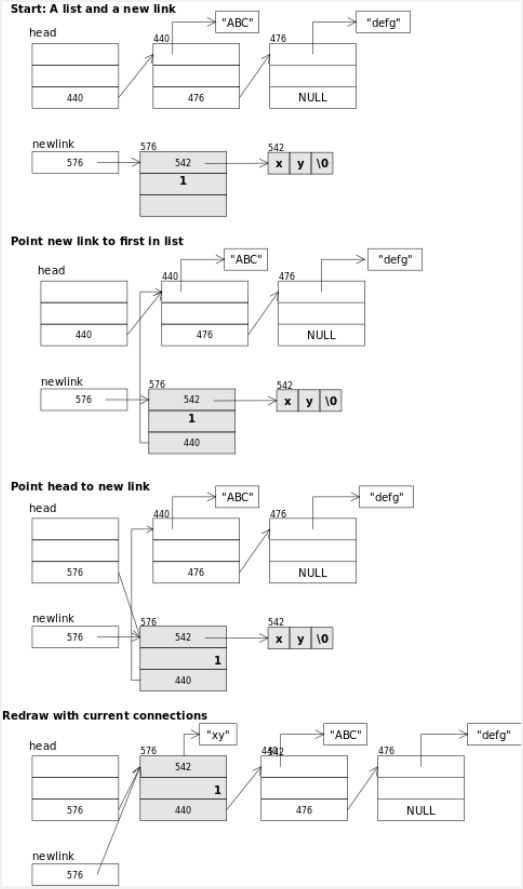
Procedure for Inserting a New Item



Make and populate new Link

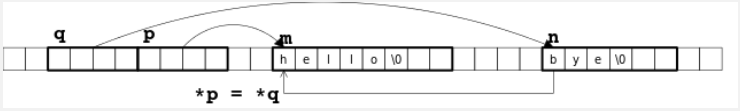


Insert new Link at Front



Common Use of Pointers: Processing Strings

```
char m[8] = "hello";
char n[6]  = "bye";
strcpy(m, n);
```



You have to copy the string, char by char.
Use a loop with indexing m[i] = n[i] until n[i] == \0
or
Use a loop with pointers *p = *q until *q == \0