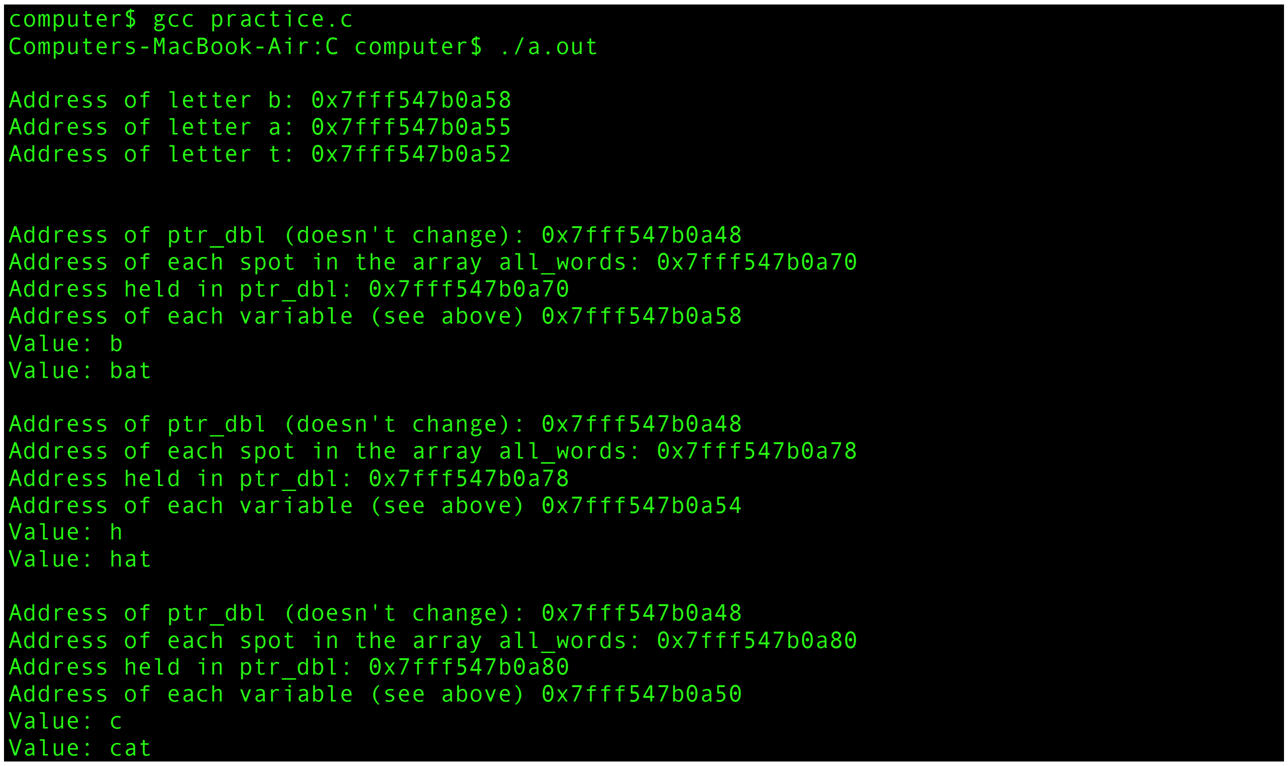
I STRONGLY ADVISE YOU TO DRAW THESE OUT LIKE I DO ON THE BOARD TO REALLY UNDERSTAND THE TOPIC.

Example 1-Array of pointers and pointer arithmetic

```
computer$ gcc practice.c
computer$ ./a.out

Address of price1: 0x7fff4fcada58
Address of price2: 0x7fff4fcada4c
Address of price3: 0x7fff4fcada48


Address of ptr_dbl (doesn't change): 0x7fff4fcada38
Address of each spot in the array: 0x7fff4fcada70
Address held in ptr_dbl: 0x7fff4fcada70
Address of each variable (see above) 0x7fff4fcada58
Value: 3.990000

Address of ptr_dbl (doesn't change): 0x7fff4fcada38
Address of each spot in the array: 0x7fff4fcada78
Address held in ptr_dbl: 0x7fff4fcada78
Address of each variable (see above) 0x7fff4fcada4c
Value: 5.990000

Address of ptr_dbl (doesn't change): 0x7fff4fcada38
Address of each spot in the array: 0x7fff4fcada80
Address held in ptr_dbl: 0x7fff4fcada80
Address of each variable (see above) 0x7fff4fcada48
Value: 5.500000
```

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
{
  int i;

  float price1=3.99;
  float price2=5.99;
  float price3=5.50;

  printf("\nAddress of price1: %p\n", &price1);
  printf("Address of price2: %p\n", &price2);
  printf("Address of price3: %p\n\n", &price3);

  float *ptr_price1=&price1;
  float *all_prices[3]; /*an array of float pointers*/

  all_prices[0]=ptr_price1; /*remember only addresses go in the array (pointer holds an address)*/
  all_prices[1]=&price2; /*you can use the address operator to get the address*/
  all_prices[2]=&price3;

  float **ptr_dbl=all_prices; /*this is a double pointer-pointing at an array of pointers (ptr->ptr)*/

  for(i=0;i<3;i++)
  {
          printf("\nAddress of ptr_dbl (doesn't change): %p\n",&ptr_dbl);
          printf("Address of each spot in the array: %p\n",&all_prices[i]);
          printf("Address held in ptr_dbl: %p\n", ptr_dbl); /*notice this is a pointer-increasing by 8 each time
because we are doing ptr_dbl++. it is the address of the pointer holding the addresses of each (shown below)*/
          printf("Address of each variable (see above) %p\n", *ptr_dbl);
          printf("Value: %f\n", **ptr_dbl); /*deref twice to get to value*/
          ptr_dbl++; /*incrementing by 8 bytes each time (size of a pointer in my comp)*/
  }

}
```

Example 2-Array of pointers and pointer arithmetic (char array)



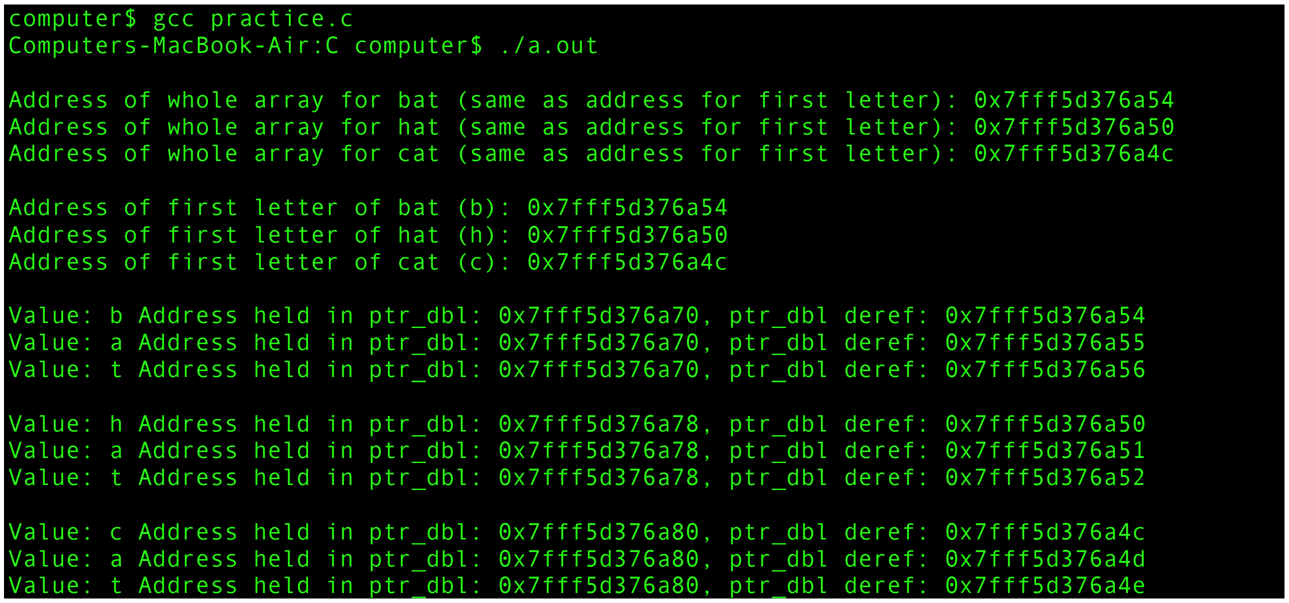
```
computer$ gcc practice.c
Computers-MacBook-Air:C computer$ ./a.out

Address of letter b: 0x7fff547b0a58
Address of letter a: 0x7fff547b0a55
Address of letter t: 0x7fff547b0a52


Address of ptr_dbl (doesn't change): 0x7fff547b0a48
Address of each spot in the array all_words: 0x7fff547b0a70
Address held in ptr_dbl: 0x7fff547b0a70
Address of each variable (see above) 0x7fff547b0a58
Value: b
Value: bat

Address of ptr_dbl (doesn't change): 0x7fff547b0a48
Address of each spot in the array all_words: 0x7fff547b0a78
Address held in ptr_dbl: 0x7fff547b0a78
Address of each variable (see above) 0x7fff547b0a54
Value: h
Value: hat

Address of ptr_dbl (doesn't change): 0x7fff547b0a48
Address of each spot in the array all_words: 0x7fff547b0a80
Address held in ptr_dbl: 0x7fff547b0a80
Address of each variable (see above) 0x7fff547b0a50
Value: c
Value: cat
```

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
{
  int i;

  char word1[]="bat"; /*a string is really just an array of chars*/
  char word2[]="hat";
  char word3[]="cat";

  printf("\nAddress of letter b: %p\n", &word1[0]);
  printf("Address of letter a: %p\n", &word2[1]);
  printf("Address of letter t: %p\n\n", &word3[2]);

  char *all_words[3]; /*an array of char pointers-remember that each word is kept in a char array and an
array is really just known by its address.  so we will be holding each char array word using the address of
the first letter*/
```

```
  all_words[0]=word1;
  all_words[1]=word2;
  all_words[2]=word3;

  char **ptr_dbl=all_words; /*this is a double pointer-pointing at an array of pointers (ptr->ptr). */

  for(i=0;i<3;i++)
  {
          printf("\nAddress of ptr_dbl (doesn't change): %p\n",&ptr_dbl);
          printf("Address of each spot in the array all_words: %p\n",&all_words[i]);
          printf("Address held in ptr_dbl: %p\n", ptr_dbl); /*notice this is a pointer-increasing by 8 each
time because we are doing ptr_dbl++. it is the address of the pointer holding the addresses of each (shown
below)*/
          printf("Address of each variable (see above) %p\n", *ptr_dbl);
          printf("Value: %c\n", **ptr_dbl); /*deref twice to get to value (letter)*/
          printf("Value: %s\n", *ptr_dbl); /*deref once to get the string itself (whole word)*/
          ptr_dbl++; /*incrementing by 8 bytes each time (size of a pointer in my comp)*/
  }

}
```

Note we can modify the above code to print out the whole word by letters:

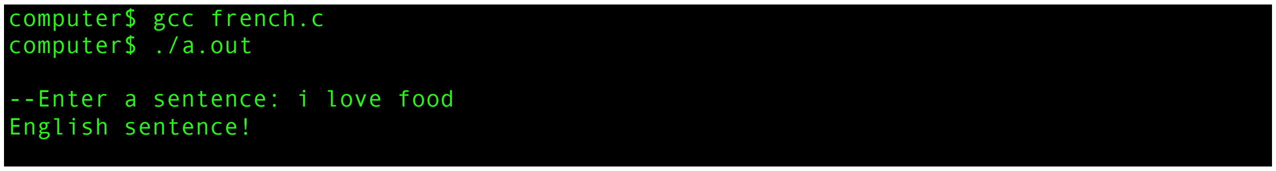
Example 3-Array of pointers and pointer arithmetic (each letter)



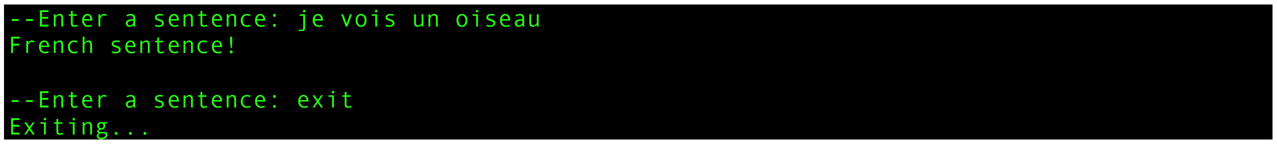
```
computer$ gcc practice.c
Computers-MacBook-Air:C computer$ ./a.out

Address of whole array for bat (same as address for first letter): 0x7fff5d376a54
Address of whole array for hat (same as address for first letter): 0x7fff5d376a50
Address of whole array for cat (same as address for first letter): 0x7fff5d376a4c

Address of first letter of bat (b): 0x7fff5d376a54
Address of first letter of hat (h): 0x7fff5d376a50
Address of first letter of cat (c): 0x7fff5d376a4c

Value: b Address held in ptr_dbl: 0x7fff5d376a70, ptr_dbl deref: 0x7fff5d376a54
Value: a Address held in ptr_dbl: 0x7fff5d376a70, ptr_dbl deref: 0x7fff5d376a55
Value: t Address held in ptr_dbl: 0x7fff5d376a70, ptr_dbl deref: 0x7fff5d376a56

Value: h Address held in ptr_dbl: 0x7fff5d376a78, ptr_dbl deref: 0x7fff5d376a50
Value: a Address held in ptr_dbl: 0x7fff5d376a78, ptr_dbl deref: 0x7fff5d376a51
Value: t Address held in ptr_dbl: 0x7fff5d376a78, ptr_dbl deref: 0x7fff5d376a52

Value: c Address held in ptr_dbl: 0x7fff5d376a80, ptr_dbl deref: 0x7fff5d376a4c
Value: a Address held in ptr_dbl: 0x7fff5d376a80, ptr_dbl deref: 0x7fff5d376a4d
Value: t Address held in ptr_dbl: 0x7fff5d376a80, ptr_dbl deref: 0x7fff5d376a4e
```

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
{
  int i,j;

  char word1[]="bat"; /*a string is really just an array of chars*/
  char word2[]="hat";
  char word3[]="cat";

  printf("\nAddress of whole array for bat (same as address for first letter): %p\n", word1);
 printf("Address of whole array for hat (same as address for first letter): %p\n", word2);
  printf("Address of whole array for cat (same as address for first letter): %p\n", word3);

  printf("\nAddress of first letter of bat (b): %p\n", &word1[0]);
  printf("Address of first letter of hat (h): %p\n", &word2[0]);
  printf("Address of first letter of cat (c): %p\n\n", &word3[0]);

  char *all_words[3]; /*an array of char pointers-remember that each word is kept in a char array and an
array is really just known by its address.  so we will be holding each char array word using the address of
the first letter*/

  all_words[0]=word1;
  all_words[1]=word2;
  all_words[2]=word3;

  char **ptr_dbl=all_words; /*this is a double pointer-pointing at an array of pointers (ptr->ptr). */

  for(i=0;i<3;i++)
  {
          for(j=0;j<3;j++)
          {
                  printf("Value: %c ", **ptr_dbl);
                  printf("Address held in ptr_dbl: %p, ptr_dbl deref: %p\n",ptr_dbl, *ptr_dbl); /*deref twice
to get to value (letter)*/
                  (*ptr_dbl)++; /*Here i am incrementing the pointer being pointed at (*ptr_dbl), so I am
printing out each letter.  remember *ptr_dbl originally got me the whole string when I had it with %s
(previous example)*/
          }
          printf("\n");
          ptr_dbl++; /*now im incrementing the whole pointer-moving to the next word*/

  }


}
```

Example 4-Sample Program 1



```
computer$ gcc french.c
computer$ ./a.out

--Enter a sentence: i love food
English sentence!
```



```
--Enter a sentence: je vois un oiseau
French sentence!

--Enter a sentence: exit
Exiting...
```

```
#include <stdio.h>
#include <string.h>
#define SIZE_WORDS 4
#define SIZE_BUFF 40

int which_lang(char *word, char**ptr);
void get_rid_newline(char* input); /*i dont actually use this function in the code, but its another way to get rid of the \n put on user input by fgets (you can actually see each step)*/
int check_lang(char *input, char **lang);


int main(int argc, char ** argv)
{

  char * french_words[]={"amour", "soleil", "oiseau", "coeur"};
  char **ptr_fr=french_words; /*double pointer holding an array of char pointers (each word as a char array)*/

  int loop=1;

  char input[SIZE_BUFF];

  while(loop)
  {
          printf("\n--Enter a sentence: ");
          fgets(input,SIZE_BUFF,stdin);
          strtok(input,"\n"); /*chop off the new line char that fgets puts on the input-look up strtok function*/
          /*get_rid_newline(input);  (using the function instead-same outcome)*/
          loop=check_lang(input, ptr_fr);
  }



}


int which_lang(char *word, char**ptr)
{
  int i;
  int check=0;

  for(i=0;i<SIZE_WORDS && check==0;i++)
  {
          if(strcmp(*ptr, word)==0)
          {
                  check=1;
          }
```

```
            ptr++;
    }

    return check;

}

/*I dont actually use this function in this program, but its a good way to see whats going on with fgets
adding a \n and strings*/
void get_rid_newline(char* input)
{
    int i;
    for(i = 0;; i++) {
            if(input[i] == '\n') {
                    input[i] = '\0'; /*puts a null terminator in the place of \n put in by fgets*/
                    break; /*break out when done (since we didnt give a stopping condition)*/
            }
    }
}

int check_lang(char *input, char ** lang)
{
    int check=0;
    char *token=strtok(input, " ");

    if(strcmp(input, "exit")==0)
    {
            printf("Exiting...\n");
            return 0; /*this will break us out of our loop in main*/
    }

    while (token != NULL && check==0)
    {
            check=which_lang(token, lang);
            token = strtok(NULL, " "); /*remember like we discussed in class that strtok does not actually
“split” the string- it modifies the string given to it and token is a pointer pointing at each segment.  It will
equal NULL when we reach the end of the string*/
    }

    if(check==0)
    {
            printf("English sentence!\n");
    }

    else
    {
            printf("French sentence!\n");
    }

    return 1; /*we don't actually care about this value, we just need to a return an int to account for a
possible exit answer*/

}
```