A290/A590 – Meeting 6 Guide
C – Introduction to Strings, and 2D Arrays

Goal: We want start working with strings, which are arrays of chars and have some special commands associated with them, thanks to a dedicated "string" library. We also want to take a closer look at some important features of Unix.

I. Arrays.

Arrays are one of the data structures in C that some other programming languages have as basic data types.

Arrays are very easy to create, and really easy to use incorrectly, leading to headaches, segmentation faults, and even system crashes.

What is an array?

Hoover: An array is a construct used to store a set of values using only one variable name. (pg. 74)

Others: ?

We declare an array, set the data type we want the array to store in each cell or element, and we declare the number of cells or elements we want in the array. The simplest would be:

int a[2];

creating an array of ints with two cells.

WATCH OUT!!!!

We put data in a cell or retrieve data from a cell based on the **cell index** and this is the most common "gotcha" with arrays.

ARRAY INDICES ALWAYS START WITH ZERO (0)!! Forget this at your peril.

Array "a" (above) has 2 cells, with indices `a[0]` and `a[1]`. If you attempt to access `a[2]`, several bad things can happen, some of which are obvious and some are not. If you have declared a set of arrays, then `a[2]` may merely be another memory location your program has reserved, say `b[0]`. Or it could be a simple variable that you have declared. So you could actually read or write to "`a[2]`" and your program would compile and even run. HOWEVER, there would eventually be a case where you were really trying to access the value of that variable or the value of `b[0]` and everything would go wrong because you had over-written that value.

It can get worse. What if you try to access `a[9999]` and write a value there? Segmentation fault is the most common outcome. You have just tried to access memory outside the scope of that reserved by your program. At best, only your program will crash. Perhaps both programs will crash. In the 21st century, it is unlikely you will cause an Operating System crash, unless you make this error when writing an Operating System.

So, what can we do with arrays? How about the re-ordering of the contents of the cells? This could be handy if we need to retrieve them in "index order" later in our program.

Let's create a new function, called "firstswap" and eventually add it to our "callingfunc.c" example file. [This is Hoover's Exercise 1 from Chapter 3, pg. 94]

```
/*Creating an function that will re-arrange the content of the cells in an array.*/

void firstswap()
{

    /*Declare 4 variables*/
    int i,j,k,swap;

    /* Array "c" of chars with 8 cells*/
```

```
    char c[8];

    /*Initialize first 5 cells with values. Other 3 are empty*/
    c[0]='f'; c[1]='r'; c[2]='o'; c[3]='g'; c[4]=0;

    printf("Initial values for first 5 cells: %c,%c,%c,%c,%c\n",c[0],c[1],c[2],c[3],c[4]);

    /*Initialize i and start a for loop*/

    for (i=0; i<4; i++)
      {
      /*Initialize k as equal to incremented i*/

      k=i;

      /*Initialize j as one greater than i and k*/

      for (j=i+1; j<4; j++)

              if (c[j]-c[k] < 10) /*What does this means or do?*/
                k=j;
            swap=c[i];
            c[i]=c[k];
            c[k]=swap;
      }

printf("Final values: i: %d, j: %d, k: %d, swap: %d\n",i,j,k,swap);

printf("Final values for first 5 cells after we swap: %c,%c,%c,%c,%c\n",
c[0],c[1],c[2],c[3],c[4]);

/* [Optional for you to try]
printf("Final values for all 8 cells after we swap:
%c,%c,%c,%c,%c,%c,%c,%c\n",c[0],c[1],c[2],c[3],c[4],c[5],c[6],c[7]);
*/
}
```

Now, we need to be sure the entire thing will run, so we need to call these functions in a main. If we do it in our separate file, it is really simple. If we want to add it to our callingfunc.c, it would be something like this.

```
int main(void)
{
      printf("THIS IS MY CONDITIONALS FUNCTION.\n\n");
      conditionals();
      printf("THIS IS MY LOOPS FUNCTION.\n\n");
      loops();
      printf("THIS IS MY FIRST ARRAY FUNCTION.\n\n");
      firstswap();

return(0);
}
```

What will the output look like? Can we map it out? What will be happening to the "value" of `c[4]`? Here is the output of **just** the first array function.

```
SAMPLE OUTPUT:

THIS IS MY FIRST ARRAY FUNCTION.

[jwhitmer@silo.luddy.indiana.edu] ./swaptest
[OUTPUT of OTHER FUNCTIONS OMITTED HERE. REFER to Meeting 4 Guide for sample output.]
```

```
Initial values for first 5 cells: frog
Final values: i: 4, j: 4, k: 3, swap: 111
Final values for first 5 cells after we swap: gfro
```

We can create arrays of more than one-dimension. In fact, we can create arrays of n-dimensions, because the "actual" structure is just a continuous set of memory locations with the proper indices attached to each. (Hoover p. 77)

How do we add this so we can test it?

II. Introduction to Strings.

You have already read Hoover and done additional research, so you know that strings are just special arrays of chars. This makes strings another form of a data structure or data construct, just like a "regular" array.

What is special about strings?

We create strings just like arrays. In fact, our swap function from last time was an array that was very similar to string, more or less. So,

```
char mystring[11];
```

creates a string of length 11, with cell indices of 0-10.

We can assign values to each cell, but since **this is supposed to be a string**, we need to be sure the final cell is assigned the value '\0'. This is the specific control character C expects to indicate the end of a string, **if** we want to use the `string.h` library and its functions.

So,

```
mystring[0]='G'; mystring[1]='r'; mystring[2]='e'; mystring[3]='e'; mystring[4]='t';
mystring[5]='i'; mystring[6]='n'; mystring[7]='g'; mystring[8]='s'; mystring[9]='!';
mystring[10]='\0';
```

Hoover points out (p. 80), that **printing strings** like an array, cell-by-cell, would be very tedious. The first "short-hand" we want to take advantage of is the fact that we can merely name the string, and printf will "know" what to do, as long as we identify the input data type. This means using `%s` to indicate the data should be interpreted as a string by printf.

So `printf("My first string says: %s\n",mystring);` should be sufficient to print out my string.

What about all these string functions? Let's look at them.

III. "String" Functions.

Hoover addresses the five most common string functions: strlen(), strcmp(); strcpy(); strcat(); and sprintf();. Let's briefly discuss each.

`strlen();` shows us the number of cells that contain data from index 0 to the cell containing '\0'. So the length of "mystring" should be 10. Is it? Wait, what happened? What does "warning: incompatible implicit declaration of built-in function strlen" mean? It means we have not included the library that includes all the string functions, so C and the gcc compiler reacts as if we are declaring a new function, strlen.

We need to `#include <string.h>` at the top of our .c file.

Once we do that, does it work?

If we want to compare strings, we need to create a second string. Let's make it "secondstring" and make it a different size. I made mine `secondstring[10]`.

`strcmp();` is potentially useful, but you need to be clear about what it does and what it returns. First, there are three possible returns values: -1, 0, and 1. 0 pretty clearly is the return value that tells us the strings are identical. This means that every cell has the same content in the same number of cells.

1 tells us that the second string is "smaller," means either we reach a cell in the second string that holds a value less than the same index in the first string, or the second string has fewer cells, even though all that hold values are the same as the first string.

-1 tells us that the first string is "smaller" according to the same criteria used in the previous paragraph for the return value of 1. Hoover offers a set of examples on p. 85.

Since we return a value, we will need a variable to hold it. We can then use that return value to make a decision. For now, let's confirm what is returned.

```
int compare;

compare = strcmp(mystring,secondstring);
```

`strcpy();` does what we would expect, taking an existing string and copying its contents into a second string. NOTE that the syntax asks for the new string name first and the string to be copied second.

```
strcpy(copystring, mystring);
```

Will this work? Are we forgetting something? We still have to allocate memory for the new string by declaring it.

We could also do something like

```
strcpy(copystring, "Happy Days");
```

`strcat();` can also be useful, but requires care and thought. It allows us to combine the contents of two strings by appending one to the other. What are the possible problems?

```
strcat(copystring,secondstring);
```

What will the output be?

Finally, there is sprint() which prints INTO a string. Notice Hoover's example on p. 89.

`sprintf(a, "Price %f, qty %d",f,i)` puts the four "values" Price %f, qty %d, f, and i into the string "a".

IV. 2D Arrays.

We can create arrays that have more than 1-dimension. The challenge is being sure we know how the data is stored so we are accessing the index we really want to access.

int a[3][2];

a[0][1] = 7;
a[1][0];

What are the 6 cell indices? a[0][0],a[0][1],a[1][0],a[1][1],a[2][0], a[2][1].

V. Unix.

We will spend most or all of our next meeting discussing Unix.

NOTES:
Code for Hoover's string functions example starts on next page.

```c
/*Updated from Hoover to match our requirements for properly "formatted" C code.*/
       /* Demonstrate using several string library functions (pgs 89-90) */

#include <stdio.h>      /* for printf(), scanf() */
#include <string.h>     /* for strlen(), strcmp() */

main(void) /*added by Jeff*/

{

char look[80], test[80];

printf("Look for: ");

scanf("%s",look);

while (1)
  {

  printf("Enter a string (0 to quit): ");

  scanf("%s",test);

  if (strcmp(test,"0") == 0)
    break;

  if (strlen(test) < strlen(look))
    printf("%s is too short for %s\n",test,look);

  else if (strcmp(test,look) == 0)
    printf("Found one!\n");

  else if (strncmp(test,look,3) == 0)
    printf("Started the same...\n");

  else
    printf("Not what we're looking for\n");

  }

return(0); /*added by Jeff*/

}
```