A290/A590 – Meeting 5 Guide
C – Functions, Calling Functions, and Arrays

Goal: We want start working with functions (sometimes called sub-routines) and the calling function(s) we need to actually use them. We also want to start working with arrays.

I. Functions.

We can create functions within a single C file that we can then call **in** our main(). This is usually considered the preferred style, as it simplifies main() and allows a wider variety of testing and makes it easy to disable a function without huge edits to main().

To create a function, you merely need to give it a name, and **declare it**, making sure the declaration appears prior to ("above") your main() function.

We can create functions for the separate loops and conditionals programs we made in Meeting 4.

```
/* creating a function to do the loops from Meeting 4 */

void loops()  [The declaration]

{
int i,x;
x=0;

/*for loop: runs for values of i = 0, 1, 2, and 3*/

printf ("This is a \"for\" loop\n");

for (i=0; i<4; i++)
  {
  x=x+i;
  printf("%d\n",x);
  }

/*while loop runs until i is less than 7, but what is the starting
  value of i for this loop?*/

printf ("This is a \"while\" loop\n");

while (i<7)
  {
  x=x+i;
  i++;
  printf("%d\n",x);
  }

/*do-while loop runs until i is less than 9, but runs at least
  once. What is the starting value of i for this loop?*/

printf ("This is a \"do-while\" loop\n");

do
  {
  x=x+i;
  i++;
  printf("%d\n",x);
  }
while (i<9);
}
```

You just need to be sure you: 1) make your function "void" (for now), 2) that you give it a unique name, and 3) that you enclose the entire contents of the function in its own set of "{}".

We can do the same things for our conditionals program:

```
/* Creating a function to show conditionals as in Meeting 4. */

void conditionals()  [The declaration]
{

int i,x;

x=0;
for (i=0; i<5; i++)
  {
  if (i%2 == 0 || i == 1)
    x=x+i;
  else
    x=x-i;
  printf("%d\n",x);
  }

/*An example of switch-case. Notice the cases are "grades" but they are looking for values
that are ints.*/

printf ("This is a \"switch-case\" conditional\n");

 int grade;

 printf ("Input Grade: [5=A, 4=B, 3=C, 2=D, 1=F] \n");
 scanf ("%d", &grade);

 switch (grade) {
              case 1:
                       printf ("Fail (F)\n");break;
              case 2:
                       printf ("Not Good (D)\n");break;
              case 3:
                       printf ("Good (C)\n"); break;
              case 4:
                       printf ("Very Good (B)\n"); break;
              case 5:
                       printf ("Excellent (A)\n"); break;
              default:
                       printf ("You have not inputted a valid grade value.\n");
                       break;
               }
}
```

All we need to do now is call these functions in our main().

II. Calling Functions.

You've probably already figured out how easy this part becomes, and that is one of the big points and/or payoffs. We'll also see how this allows very easy testing of a set of functions, either together or one-at-a-time.

To "run" our loops and conditionals functions, we merely need to call them in our main(), which will now be at the "end" of our *.c file. NOTE: When I speak or write about "calling function" I will be referring to main() unless I say otherwise. This is not the most common way to refer to things, so I want to be sure you know what I am referring to when I say "calling function."

Notice I have added some printf statements to help clearly separate the displayed output of my two functions.

```
int main(void)
{
  printf("THIS IS MY LOOPS FUNCTION.\n\n");
  loops(); [Calling the function]
  printf("THIS IS MY CONDITIONALS FUNCTION.\n\n");
  conditionals(); [Calling the function]

 return(0);
}
```

It is really as simple as that. Now, as the functions become more complex, so can the manner in which we call them. More about that in a later meeting.

III. Arrays.

Arrays are one of the data structures or **constructs** in C that some other programming languages have as a first-level or primary or native data type.

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: ?

It is important to understand none of the following will work if you do not `#include <stdio.h>` in your C file.

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. Again, all this is enabled by other methods included in **stdio.h**.

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()  [The declaration]
{

    /*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]);
}
```

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();  [Calling the function]
      printf("THIS IS MY LOOPS FUNCTION.\n\n");
      loops();  [Calling the function]
      printf("THIS IS MY FIRST ARRAY FUNCTION.\n\n");
      firstswap();  [Calling the function]

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.]
Initial values for first 5 cells: frog
Final values: i: 4, j: 4, k: 3, swap: 111
Final values for first 5 cells as 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?

IV. Coming Next: Strings.

Who has heard of these and/or used them?

NOTES: