Notes on Parameter Passage in C Functions

All parameter passage in C is by value. This means that the value of a parameter is passed to a function – not the parameter itself. If the function changed the passed value it does not change the original variable in the calling program.

Example 1

The following function has three parameters and it returns an int.

```c
int x, y = 7, z = 2;
float r = 3.2;
x = ParameterList(y, z, r);
```

```c
int ParameterList(int a, int b, float r) {
    int x = 5; // Local variable
    r = (float)a*b; // changing a, b, and r does not change any variables in the main program
    a = x + (int)r;
    return a + b; // program
}
```

When the function is called the values of y, z, and r in the main program are copied into the new variables a, b, and r in the function. Since the values are copied to new variables in the function, changing those new variables does not change the original variables in the main program. The function, in this case returns an int and that is the only thing that comes back from the function.

In order to get more than one thing back from a function we must pass the address of a variable – not the value of the variable. Another name for address is reference so that when we pass an address of a variable we say we are passing the parameter by reference.

In C you can create a variable which holds the address of another variable. Such a variable is called a pointer. To create a pointer you do something like this:

```c
int *pi; // pointer to an int
float *pf; // pointer to a float
```

Creating a pointer variable does not make it point to a variable. To do that you must create the variable and assign its address to the pointer. The ampersand (&) operator provides the address of a variable. In C we could create an int and a float and assign our pointers to point to them like this:

```c
int i = 5;
float f = 3.14;
pi = &i;
pf = &f;
```

To pass a variable to a function by reference we need to pass its address and catch that address with a pointer. The following example illustrates how this is done.

Example 2

```c
// Prototype
void ParameterPointer(int *x);

// Function call
int y = 7;
ParameterPointer(&y);
```

```c
void ParameterPointer(int *x) {
    int a = 2;
    *x = 2*a; // changing x changes y in the main program.
}
```
In this example *x is a pointer to the main program variable y. Changing *x in the function changes the value of y in the main program.

In some cases, it is necessary to pass an array to a function. In C an array name is already a pointer so it's not necessary to create a pointer to pass an array by reference. All arrays are already passed by reference as a default. Here is an example that illustrates this concept.

**Example 3**

```c
//Prototype
void ParameterArray(int *x);

//Function call
int a[] = {1, 2, 3};
ParameterArray(a);
```

In this case function ParameterArray has a pointer argument. Since an array name (in this case it's just a) is already a pointer so we just include the name as the argument. Changes to the array values in the function cause change to the array in the main program since they both have the same address.