Pass by Value vs Pass by Reference

If you have used Java then you will probably have heard of this before. It is an idea that is related to the scope of a function. This is the classical example of pass by value semantics written in C.

Pass by Value

```
void boring(int i){
  i += 5;
  printf("Val of i in function: %d\n", i);
}


int main(){

  int i = 6;
  printf("Val of i before function call: %d\n", i);
  boring(i);
  printf("Val of i after function call: %d\n", i);
}
```

Output

```
Val of i before function call: 6
Val of i in function: 11
Val of i after function call: 6
```

So you can imagine that the value of `i` is copied to local variable in the scope of `boring()` so inside the function we reference a copy, instead of the variable in the scope of `main()`.

Pass by Reference

In C if we want a function to modify its argument instead of a copy of the argument we can pass pointer to the argument instead. We can obtain a pointer to anything by simply using the address of (&) operator.

```
void boring2(int* i){//now we take a pointer
  *i += 5; //dereference and assign
  printf("Val of i in function: %d\n", *i);
}

void main(){
  int i = 6;
  printf("Val of i before function call: %d\n", i);
  boring2(&i); //using (&) to pass a pointer
  printf("Val of i after function call: %d\n", i);
}
```

Output

```
Val of i before function call: 6
Val of i in function: 11
Val of i after function call: 11
```

Passing a pointer to variable is what is referred to as "pass by reference" in C.

Lets see a more interesting example of modifying our arguments, by making a function that takes an array of ints and squares everything in the list.

```
void squareArr(int* arr, int n){//takes array and its size

  while(n--){//recall anything nonzero is true
    //value at pointer times equals value at pointer
    *arr *= *arr;
    //increment pointer by 1
    arr++;
  }
  printf("Val of pointer (arr) in function after loop: 0x%x\n",arr);
}
void main(){

  //we can omit the size when statically assigning, it will be inferred
  int arr[] = {1,2,4,6,7};

  printf("Val of pointer (arr) before function call: 0x%x\n",arr);

  printf("array before function call: ");
  for(int i=0; i < 5; i++) printf("%d ",arr[i]);
  printf("\n");

  squareArr(arr, 5);

  printf("Val of pointer (arr) after function call: 0x%x\n",arr);
  printf("array after function call: ");
  for(int i=0; i < 5; i++) printf("%d ",arr[i]);
  printf("\n");

}
```

Output

```
Val of pointer (arr) before function call: 0x9aa28580
array before function call: 1 2 4 6 7
Val of pointer (arr) in function after loop: 0x9aa28594
Val of pointer (arr) after function call: 0x9aa28580
array after function call: 1 4 16 36 49
```

You can see that everything in our array was squared as intended, and I hope you will also notice that the pointers value had changed by the end of the function, but that change was not reflected in the scope of main. This shows that there really are not different argument semantics in C. When you are passing a pointer, you are still just passing an integer. That integer value is copied into a variable in the local scope, and modifying it does not effect anything in an outside scope. So really pass by reference is kind of fake news. Everything is pass by value, but changing the contents of memory (by dereferencing pointers) is absolute.