Argument passing to functions

Pointer passed as arguments have some subtle behavior which is important to understand.

Changes within a function may or may not reflect in the calling code, it depends on what the change is.

There are 2 operations that you can do on pointers in a function:
1. Modification of the pointer value - reflects in the calling code (@navdeep this has to be like a comment)
2. Reassignment of a pointer - does not reflect in the calling code (@navdeep this has to be like a comment)

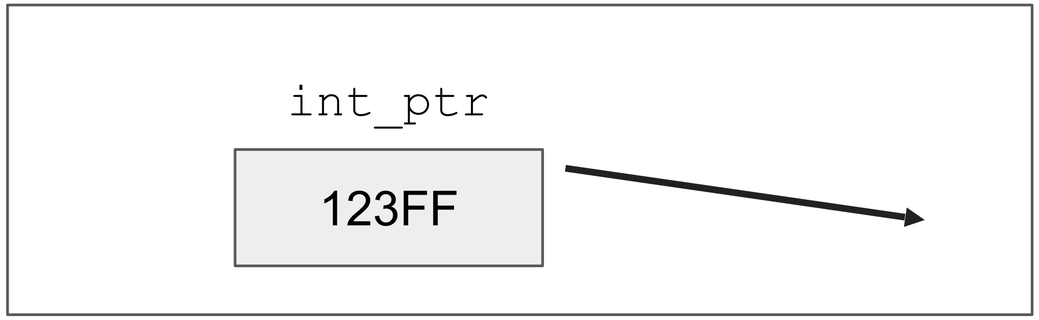
MODIFICATIONS TO POINTERS

```
int value = 10;
int *int_ptr = &value;
some_function(int_ptr);
printf("%d", value);
```

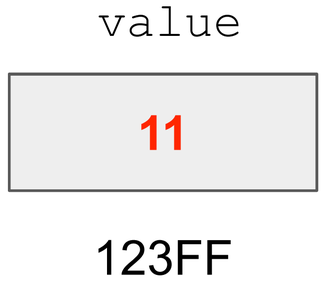
WHAT WILL THIS PRINT?

```
void some_function(int* some_ptr) {
  *some_ptr = 11;
}
```

HOW DOES POINTER VALUE MODIFICATION IN A FUNCTION WORK?



some_ptr REFERS TO THE SAME PORTION OF MEMORY THAT int_ptr POINTS TO




```
*some_ptr = 11;
```

THE PRINT STATEMENT WILL PRINT 11

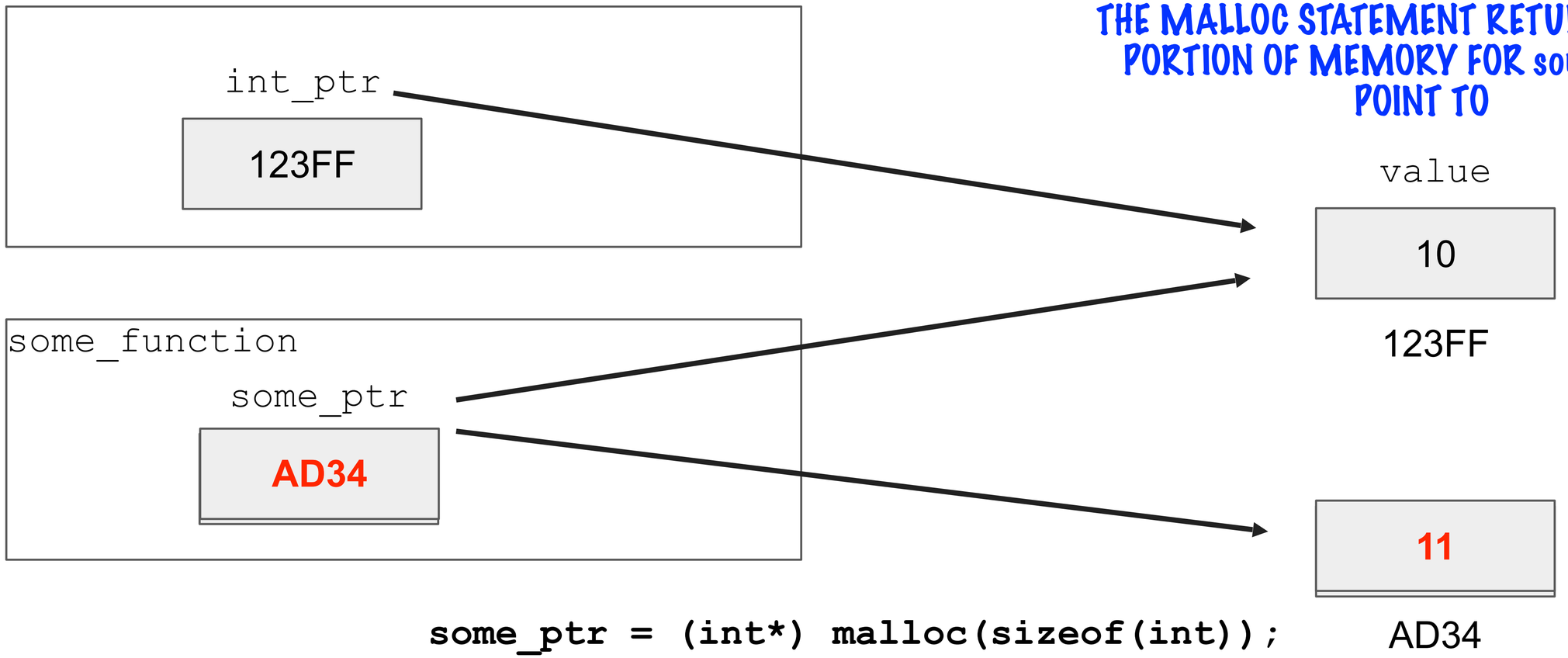
REASSIGNMENT OF POINTERS

```
int value = 10;
int *int_ptr = &value;
some_function(int_ptr);
printf("%d", value);
```

WHAT WILL THIS PRINT NOW?

```
void some_function(int* some_ptr) {
  some_ptr = (int*) malloc(sizeof(int));
  *some_ptr = 11;
}
```

HOW DOES POINTER REASSIGNMENT WORK?



THE MALLOC STATEMENT RETURNS A NEW PORTION OF MEMORY FOR some_ptr TO POINT TO

```
some_ptr = (int*) malloc(sizeof(int));
*some_ptr = 11;
```

THE PRINT STATEMENT WILL PRINT 10, THE ORIGINAL VALUE IS UNDISTURBED