[How come an array's address is equal to its value in C?](http://hello.k7mm.com/5i5.php?u=T2k4dmMzUmhZMnR2ZG1WeVpteHZkeTVqYjIwdmNYVmxjM1JwYjI1ekx6STFNamd6TVRndmFHOTNMV052YldVdFlXNHRZWEp5WVhsekxXRmtaSEpsYzNNdGFYTXRaWEYxWVd3dGRHOHRhWFJ6TFhaaGJIVmxMV2x1TFdNPQ%3D%3D&b=13)

|  |  |
| --- | --- |
| up vote75down vote[favorite](http://hello.k7mm.com/5i5.php?u=T2k4dmMzUmhZMnR2ZG1WeVpteHZkeTVqYjIwdmNYVmxjM1JwYjI1ekx6STFNamd6TVRndmFHOTNMV052YldVdFlXNHRZWEp5WVhsekxXRmtaSEpsYzNNdGFYTXRaWEYxWVd3dGRHOHRhWFJ6TFhaaGJIVmxMV2x1TFdNPQ%3D%3D&b=13)  49 | In the following bit of code, pointer values and pointer addresses differ as expected.  But array values and addresses don't!  How can this be?  Output  my\_array = 0022FF00  &my\_array = 0022FF00  pointer\_to\_array = 0022FF00  &pointer\_to\_array = 0022FEFC  #include <stdio.h>  int main()  {  char my\_array[100] = "some cool string";  printf("my\_array = %p\n", my\_array);  printf("&my\_array = %p\n", &my\_array);  char \*pointer\_to\_array = my\_array;  printf("pointer\_to\_array = %p\n", pointer\_to\_array);  printf("&pointer\_to\_array = %p\n", &pointer\_to\_array);  printf("Press ENTER to continue...\n");  getchar();  return 0;  } |

|  |  |
| --- | --- |
| 91down voteaccepted | The name of an array usually evaluates to the address of the first element of the array, so array and &array have the same value (but different types, so array+1 and &array+1 will *not* be equal if the array is more than 1 element long).  There are two exceptions to this: when the array name is an operand of sizeof or unary &(address-of), the name refers to the array object itself. Thus sizeof array gives you the size in bytes of the entire array, not the size of a pointer.  For an array defined as T array[size], it will have type T \*. When/if you increment it, you get to the next element in the array.  &array evaluates to the same address, but given the same definition, it creates a pointer of the type T(\*)[size] -- i.e., it's a pointer to an array, not to a single element. If you increment this pointer, it'll add the size of the entire array, not the size of a single element. For example, with code like this:  char array[16];  printf("%p\t%p", (void\*)&array, (void\*)(&array+1));  We can expect the second pointer to be 16 greater than the first (because it's an array of 16 char's). Since %p typically converts pointers in hexadecimal, it might look something like:  0x12341000 0x12341010 |