Swapping pointers in C (char, int)

I have been struggling to understand the different behaviour when swapping pointers in C. If I want to swap two int pointers, then I can do

```
void intSwap (int *pa, int *pb){
    int temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```

However, if I want to swap two char pointers I need to do something like

```
void charSwap(char** a, char** b){
    char *temp = *a;
    *a = *b;
    *b = temp;
}
```

because if I do

```
void charSwap(char* a, char* b){
    char temp = *a;
    *a = *b;
    *b = temp;
}
```

the compiler complains about the expression *a = *b as it cannot change the values. If I want to swap two strings (i.e. `char* s1= "Hello"; char* s2="Bye";`)how would one do it?

Could you please give me a little bit of help? I would like to really learn how it works so I will not need to experience trial and error all the time until I get the right answer. I hope it's useful for many other people.

c pointers

edited Dec 6 '11 at 16:53

asked Dec 6 '11 at 16:36

Manolete
943 2 18 43

10 The first example doesn't swap two int pointers. I think that's the source of your confusion. – R. Martinho Fernandes Dec 6 '11 at 16:39

@R.MartinhoFernandes I think you might be right – Manolete Dec 6 '11 at 16:48

5 Answers

The first thing you need to understand is that when you pass something to a function, that something is copied to the function's arguments.

Suppose you have the following:

```
void swap1(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
    assert(a == 17);
    assert(b == 42);
    // they're swapped!
}

int x = 42;
int y = 17;
swap1(x, y);
assert(x == 42);
assert(y == 17);
// no, they're not swapped!
```

The original variables will not be swapped, because their values are copied into the function's

arguments. The function then proceeds to swap the values of those arguments, and then returns. The original values are not changed, because the function only swaps its own private copies.

Now how do we work around this? The function needs a way to refer to the original variables, not copies of their values. How can we refer to other variables in C? Using pointers.

If we pass pointers to our variables into the function, the function can swap the values in *our* variables, instead of its own argument copies.

```
void swap2(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
    assert(*a == 17);
    assert(*b == 42);
    // they're swapped!
}

int x = 42;
int y = 17;
swap2(&x, &y); // give the function pointers to our variables
assert(x == 17);
assert(y == 42);
// yes, they're swapped!
```

Notice how inside the function we're not assigning to the pointers, but assigning to what they point to. And the pointers point to our variables `x` and `y`. The function is changing directly the values stored in *our* variables through the pointers we give it. And that's exactly what we needed.

Now what happens if we have two pointer variables and want to swap the *pointers* themselves (as opposed to the values they point to)? If we pass pointers, the pointers will simply be copied (not the values they point to) to the arguments.

```
void swap3(int* a, int* b) {
    int* temp = a;
    a = b;
    b = temp;
    assert(*a == 17);
    assert(*b == 42);
    // they're swapped!
}
void swap4(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
    assert(*a == 17);
    assert(*b == 42);
    // they're swapped!
}

int x = 42;
int y = 17;
int* xp = &x;
int* yp = &y;
swap3(xp, yp);
assert(xp == &x);
assert(yp == &y);
assert(x == 42);
assert(y == 17);
// Didn't swap anything!
swap4(xp, yp);
assert(xp == &x);
assert(yp == &y);
assert(x == 17);
assert(y == 42);
// Swapped the stored values instead!
```

The function `swap3` only swaps its own private copies of our pointers that it gets in its arguments. It's the same issue we had with `swap1`. And `swap4` is changing the values our variables point to, not the pointers! We're giving the function a means to refer to the variables `x` and `y` but we want them to refer to `xp` and `yp`.

How do we do that? We pass it their addresses!

```
void swap5(int** a, int** b) {
    int* temp = *a;
    *a = *b;
    *b = temp;
    assert(**a == 17);
    assert(**b == 42);
    // they're swapped!
}
```

```
int x = 42;
int y = 17;
int* xp = &x;
int* yp = &y;
swap5(&xp, &yp);
assert(xp == &y);
assert(yp == &x);
assert(x == 42);
assert(y == 17);
// swapped only the pointers variables
```

This way it swaps our pointer variables (notice how `xp` now points to `y`) but not the values they point to. We gave it a way to refer to our pointer variables, so it can change them!

By now it should be easy to understand how to swap two strings in the form of `char*` variables. The swap function needs to receive pointers to `char*`.

```
void swapStrings(char** a, char** b){
    char *temp = *a;
    *a = *b;
    *b = temp;
    assert(strcmp(*a, "world") == 0);
    assert(strcmp(*b, "Hello") == 0);
}

char* x = "Hello";
char* y = "world";
swapStrings(&x, &y);
assert(strcmp(x, "world") == 0);
assert(strcmp(y, "Hello") == 0);
```

edited Jan 8 '12 at 20:12

answered Dec 6 '11 at 16:55

R. Martinho Fernandes
111k 36 279 385

2 wow, impressive answer – Mooing Duck Dec 6 '11 at 17:51

2 Good stuff man, that is one of the best C lessons I have ever had! – Manolete Dec 6 '11 at 23:10

might just be good to let the answer focus on just one thing, but to be correct for this example we need `swapStrings(const char** a, const char** b)` – u0b34a0f6ae Dec 7 '11 at 10:15

1 @kaizer.se thanks for noticing that! I removed the `const`s from the answer to avoid confusion. – R. Martinho Fernandes Dec 7 '11 at 10:26

In C, a string, as you know, is a character pointer (char *). If you want to swap two strings, you're swapping two char pointers, i.e. just two addresses. In order to do any swap in a function, you need to give it the addresses of the two things you're swapping. So in the case of swapping two pointers, you need a pointer to a pointer. Much like to swap an int, you just need a pointer to an int.

The reason your last code snippet doesn't work is because you're expecting it to swap two char pointers -- it's actually written to swap two characters!

Edit: In your example above, you're trying to swap two int pointers incorrectly, as R. Martinho Fernandes points out. That will swap the two ints, if you had:

```
int a, b;
intSwap(&a, &b);
```

edited Dec 6 '11 at 16:51

answered Dec 6 '11 at 16:39

Dan Fego
7,091 1 20 40

```
void intSwap (int *pa, int *pb){
    int temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```

You need to know the following -

```
int a = 5; // an integer, contains value
int *p; // an integer pointer, contains address
```

```
p = &a; // &a means address of a
a = *p; // *p means value stored in that address, here 5
```

```
void charSwap(char* a, char* b){
    char temp = *a;
    *a = *b;
    *b = temp;
}
```

So, when you swap like this. Only the value will be swapped. So, for a `char*` only their first `char` will swap.

Now, if you understand char* (string) clearly, then you should know that, you only need to exchange the pointer. It'll be easier to understand if you think it as an `array` instead of string.

```
void stringSwap(char** a, char** b){
    char *temp = *a;
    *a = *b;
    *b = temp;
}
```

So, here you are passing double pointer because starting of an `array` itself is a pointer.

edited Dec 6 '11 at 20:24

answered Dec 6 '11 at 16:51

Rifat
3,970 3 18 37

@R.MartinhoFernandes Thanks, I've fixed it :) – Rifat Dec 6 '11 at 17:47

You need to understand the different between pass-by-reference and pass-by-value.

Basically, C only support pass-by-value. So you can't reference a variable directly when pass it to a function. If you want to change the variable out a function, which the swap do, you need to use pass-by-reference. To implement pass-by-reference in C, need to use pointer, which can dereference to the value.

The function:

```
void intSwap(int* a, int* b)
```

It pass two pointers value to intSwap, and in the function, you swap the values which a/b pointed to, but not the pointer itself. That's why R. Martinho & Dan Fego said it swap two integers, not pointers.

For chars, I think you mean string, are more complicate. String in C is implement as a chars array, which referenced by a char*, a pointer, as the string value. And if you want to pass a char* by pass-by-reference, you need to use the ponter of char*, so you get char**.

Maybe the code below more clearly:

```
typedef char* str;
void strSwap(str* a, str* b);
```

The syntax swap(int& a, int& b) is C++, which mean pass-by-reference directly. Maybe some C compiler implement too.

Hope I make it more clearly, not comfuse.

answered Dec 6 '11 at 16:58

Googol
136 3

If you have the luxury of working in C++, use this:

```
template<typename T>
void swapPrimitives(T& a, T& b)
{
    T c = a;
    a = b;
    b = c;
}
```

Granted, in the case of `char*` , it would only swap the pointers themselves, not the data they point to, but in most cases, that is OK, right?

answered Jan 8 '12 at 20:24

TheBuzzSaw
5,144 15 35