Swift Weekly - Issue 01
Vandad Nahavandipoor
http://www.oreilly.com/pub/au/4596

Email: vandad.np@gmail.com
Blog: http://vandadnp.wordpress.com
Skype: vandad.np

Pointers

Table of Contents:

Email: vandad.np@gmail.com
Blog: http://vandadnp.wordpress.com
Skype: vandad.np

Introduction

Apple's intention with Swift was to start making pointers very abstract, as though pointers were not abstract enough already! To me they are fine, since I grew up with C and a lot of Assembly (x86, NASM and MASM). But to those of us who aren't familiar with pointers, working with them in Swift is a very daunting challenge. Or is it?

I assume that you already know what a pointer is and you want to learn about pointers in Swift. Now let's say that you have an integer of type *Int* and you want to change its value via a pointer:

```
var myInt = 10
var ptr = UnsafeMutablePointer<Int>.initialize(&myInt)
ptr(20)
myInt // this is now 20
```

swift-weekly-issue01-example01/MyPlayground.playground

Allocating Memory for Pointers

The *UnsafeMutablePointer* type is a mutable pointer where you are in charge of the memory management of the object to which it points. In this example, we pointed it to an object that is automatically managed under ARC. We don't have to manage the memory. But what if we want to actually allocate a pointer, put a value into it, then destroy it? Well, we do that using the alloc method of the pointer and the value will be the number of blocks of the specific object type that we want to allocate:

```
// allocate one integer block
var myPointer = UnsafeMutablePointer<Int>.alloc(1)
// change its value
myPointer.memory = 1_234
let myInteger = myPointer.memory
// myInteger = 1234
myPointer.destroy()
// now get rid of the pointer's memory
myPointer.dealloc(1)
```

swift-weekly-issue01-example02/MyPlayground.playground

Pointing to Arrays

Now what if we want to work with arrays and pointers? Let's say that you have an array of *Int* instances and you want to get the pointer to it and change the array values using the pointer. That's when we want to use the *UnsafeMutableBufferPointer* type. Use the initializer of this type and pass the memory address of your array to the *start* initializer parameter and for the *count* parameter pass the number of items in the original array. Once you've done that, you can access the memory address of the first item in the array using the *baseAddress* method of the pointer which will return an *UnsafeMutableArray*<T> where T is the type of the array. Here is an example:

Email: vandad.np@gmail.com
Blog: http://vandadnp.wordpress.com
Skype: vandad.np

```
var myArray = [10, 20, 30, 40]
var arrayPtr = UnsafeMutableBufferPointer(start: &myArray, count:
myArray.count)
var base = arrayPtr.baseAddress as UnsafeMutablePointer<Int>
base.memory // 10
base.memory = 100 // change the value of the first element in the array
myArray // 100, 20, 30, 40

// move the array forward by one element
base = base.successor()
base.memory // 20 (second item in the array)
// add the value of myArray[1] to myArray[0] and put it in myArray[1]
base.memory = base.memory + base.predecessor().memory
myArray // 100, 120, 30, 40
```
swift-weekly-issue01-example03/MyPlayground.playground

Pointing to Custom Structures

As we have seen, we can simply create a pointer to a system structure, like an *Int* type, which actually is a structure, not a class. But how do we create a pointer to a custom structure? Let's create the structure first:

```
struct Person{
  var firstName: String
  var lastName: String
}
```
swift-weekly-issue01-example04/MyPlayground.playground

The way to now create a pointer that points to an instance of this structure is to use the UnsafeMutablePointer type again but this time allocate, and then initialize it with an instance of the structure like so:

```
var fooPtr = UnsafeMutablePointer<Person>.alloc(1)
fooPtr.initialize(Person(firstName: "Vandad", lastName: "Nahavandipoor"))
fooPtr.memory.firstName // will print "Vandad"
// change the last name
fooPtr.memory.lastName = “Lastname"
```
swift-weekly-issue01-example04/MyPlayground.playground

Working with Pointers in Cocoa Touch

Every now and then you will get a pointer from Cocoa Touch APIs that you have to know how to use. One of those examples is with the NSArray class when you try to enumerate through the items. In that case, the third parameter to the enumeration closure is a pointer to an Objective-C boolean value. If you want to stop the enumeration, you have to put the value of *true* into the memory of that pointer like so:

Email: vandad.np@gmail.com
Blog: http://vandadnp.wordpress.com
Skype: vandad.np

```
let array:NSArray = ["Name", "Last name", "Age", "Sex"]
array.enumerateObjectsUsingBlock {
  (obj: AnyObject!, index: Int,
  stop: UnsafeMutablePointer<ObjCBool>) -> Void in
  if let myString = obj as? String{
    if myString == "Last name"{
      println("Found the last name")
      stop.memory = true
    }
  }
}
```
swift-weekly-issue01-example05/MyPlayground.playground

Using Pointers in Your Own APIs

If you are designing some APIs, I highly discourage you in using pointers. Simply following Swift's runtime rules (passing const instead of by reference) is the best idea. However, sometimes you might need to export your Swift classes to Objective-C code and in that case, you might want to use pointers to Objective-C types.

```
func invertObjCBoolean(var bool: UnsafeMutablePointer<ObjCBool>){
  bool.memory = ObjCBool(!bool.memory.boolValue)
}
var myBool = true as ObjCBool
invertObjCBoolean(&myBool)
myBool // myBool is now false is false
```
swift-weekly-issue01-example06/MyPlayground.playground

Conclusion

Try to avoid using pointers as much as possible. Just use the intrinsic and internal mechanism of pointer handling in Swift which applies only to classes by the way. Use explicit pointers only if you __really__ need them.