Alex Myers

Prof. Marty Mansfield

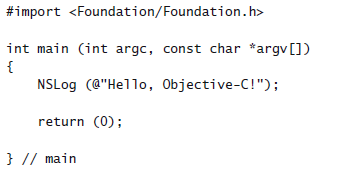
CS4600

2 December 2013

Objective-C

Introduction

Objective-C has had a long lineage and development through its existence. Objective-C is based on a programming language called C. The C language is almost 40 years old and created by Dennis Ritchie and Ken Thompson while working at AT&T Bell Labs. Objective-C uses syntax from the C language. C gives Objective-C many of the built in data types, functions, structs, and enumerated data types. Objective-C gives the ability for object orientation by implementing classes, objects, protocols, and properties. Objective-C was created by Brad Cox and Tom Love in 1983. They originally sought to add Smalltalk-80 functionality to C. Approximately two years later Steve Jobs' company NeXT to be used in APIs for their operating system called NeXTSTEP. In December 1996, Apple acquired NeXT into their OS called Rhapsody, which later evolved into Mac OS X. Objective-C eventually become a native programming language for Apples' mobile devices running IOS apps. The following figure is an example of a simple Hello World Program written in Objective-C.  
  
Figure 1.



Objective-C has a very similar structural organization to that of C. Xcode is the primary development tool for Objective-C. Xcode uses the .m extension to indicate a file that holds Objective-C code and will be processed by the Objective-C compiler. Just like C, Objective-C uses header files to hold declarations of elements such as structs, symbolic constants, and function prototypes. These libraries can be used by using #import instead of #include like in C. Objective-C classes is divided into two parts. One Part is the interface, which provides the public view of the class. The interface contains all the information necessary for someone to use the class. The other part of class's source is the implementation. The implementation section tells the Objective-C compiler how to make the class actually work.

There are many ways how the flow of control in Objective-C is managed. The if Statement, if...else statement, if...else if statements, switch statement, Variable Loop Scope, Infinite for Loop, Nested for Loops, while loops, and, do...while loops.

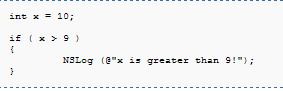
The if statement is the most basic of flow control options. The basic syntax of the Objective-C if statement is as follows:

**if** (Boolean expression) {

// Code to be executed

}  
If the Boolean expression evaluates to true then the code in the body of the statement is executed, otherwise if it is false the code in the body is false.

The syntax for the if...else statement is as follows:



**If** (Boolean expression) {

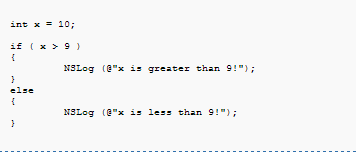
// code to be executed if true

} else {

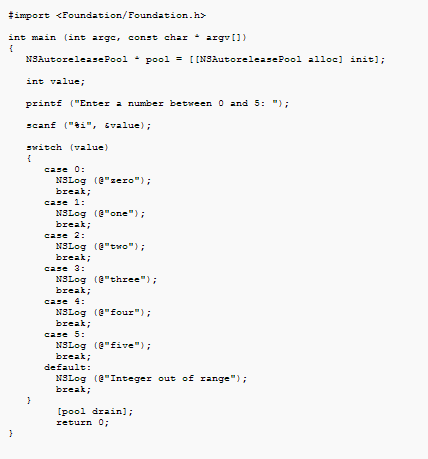
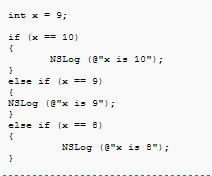
// code to be executed if false

}  
The if...else statement allows for alternative code to executed in the case of the Boolean expression to be found as false.

In the case there is the need to make decisions based on multiple different criteria. This works well for a moderate number of comparisons but becomes inefficient as the volume of expression evaluations increase. An example is as follows:

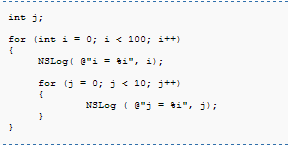


The switch statement is the more efficient way to test larger numbers of possible conditions. This is best used when there is more than three possible scenarios.

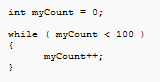


The next control structure is a for loop. These are good for performing repetitive tasks, quickly. Whereas these loops are available to be nested, and with this they need to be able to escape or restart. These loops are able to be escaped from using the keyword **break**, and will skip remaining code in the body of the loop and redo execution from the top when it hits the keyword **continue**.

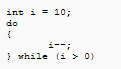
The while loop repeats a set of tasks until the condition is met. In the following example the expression will evaluate whether the variable is less than 100, if it is less than 100 the code will execute until the variable is found to be greater than 100.



The do...while loop evaluates an expression before executing the code in the body of the loop. Even if the expression is evaluated to false the code will always executed once.



These are all of the structures that can be used to control the flow of programs in Objective-C.



Objective-C has many data structures, as well as the ability to create new ones. The following table shows the built-in types.

|  |  |  |
| --- | --- | --- |
| Type | Description | Size |
| Char | A Character | 1 byte |
| Int | An integer, a whole number | 4 bytes |
| Float | Single precision floating point number | 4 bytes |
| Double | Double precision floating point number | 8 bytes |
| Short | A short integer | 2 bytes |
| Long | A double short | 4 bytes |
| Long Long | A double long | 8 bytes |
| BOOL | Boolean (signed char) | 1 byte |

Objective-C also has enumerated types, typedef, and constants. They can be defined as follows:   
**Enumeration types  
  
Typedef  
  
Constants**



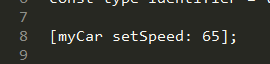
Objective-C can use both dynamic typing and static typing. The Dynamic typing is when an object can be sent a message that isn't specified in its interface. This will allow for flexibility, and the ability to redirect the message to a different object that may be able to respond better. This is known as message forwarding or delegation. Static typing information may also be added to variables. This information will be checked at compile time.

Forwarding is the sending of a message to an object that may not respond. Instead of responding or dropping the message an object can forward the message to an object that can actually respond to it.

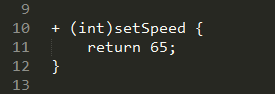
In Objective-C an object is defined using two different sections, @interface, and @implementation. The @interface defines what messages the object can respond to as well as any instance variables to be used. These are written in a .h or better known as header file. The @implementation sections contains the code that pertains to each of the various messages. These sections are written in an Objective-C class or a .m file.

Objective-C revolves around object orientation and the idea of sending messages. The following example is that of an object sending a message with parameters in the Objective-C Syntax.

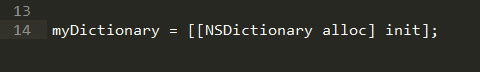
The above statement will send a message to an instance of the Car class named myCar. myCar is known as the receiver because it is receiving the message. The message is used to select which method to be used within the object, setSpeed is known as a selector. So in Objective-C the names message and method are synonymous because a message simply selects a method based on name. However a class doesn't need to be instantiated to be used. All class methods in Objective-C must start with a plus sign as seen below.



One of the biggest disadvantages to class methods is that no instance variables can be used, and if it is used Xcode will still compile it with a warning. This will result in broken program due to the class method won't be able to find this none instantiated variable. However Class methods can still have its own local variables within itself.



The above code snippet, shows what is called a compound call. This method returns a new NSDictionary object, which is then sent to the init instance method. This will be used inside the class to initialize itself and then return the newly instantiated object back to the caller.



Objective-C started out to be an object oriented version of Smalltalk and a layer on C. Overall Objective-C has accomplished its goal, and surpassed it. Objective-C not only provide this function but has become the base for one of the most popular mobile operating systems in the world. Objective-C is set up to be easy enough for all to learn. By being based on C it is relatively easy to pick up and learn. Which is why there has been many successful apps on the app store by young self taught individuals. Objective-C is still widely used, and currently maintained by Apple Inc. I think this language has been relatively easy to pick up and fun to write in. I previously had experience in C and this made the transition into Objective-C very easy. The way that Objective-C handles messaging between objects makes integrating with IOS's touch screen objects very easy. With the ever increasing popularity of Ipods, Iphones, Ipads and assorted other future I devices, I don't speculate that Objective-C will be going anywhere unless of course Apple as a whole moves away from it and creates a native language. There have been several competitors against Cocoa and Objective-C. Titanium is a cross-platform that will allow apps that would normally have to be written in Objective-C in JavaScript which is then compiled down to C code. However Objective-C still stands as a giant amongst mobile app development languages. My plan on future use of Objective-C will be more mobile app development. I have a few apps I still want to design and put on the app store before I jump into the freelance game. The need for mobile developers is quite high in the market right now, and they make good money. I plan to continue my study of this language and hope to put it to good use.

Works Cited

Bennett, Gary, Mitch Fisher, and Brad Lees. *Objective-C for Absolute Beginners: IPhone, IPad, and Mac Programming Made Easy*. [S.l.]: Apress, 2011. Print.

Goldstien, Neil. *For Dummies*. Wiley Brand, n.d. Web.

Harrington, Brianna. "Introduction & Brief History of Objective-C | Intertech Blog." *Intertech Blog*. Intertech, 20 Feb. 2013. Web. 02 Dec. 2013.

Knaster, Scott, Waqar Malik, Mark Dalrymple, and Mark Dalrymple. *Learn Objective-C on the Mac: For IOS and OS X*. [Berkeley, Calif.]: Apress, 2012. Print.

"Objective-C 2.0 Essentials." *- Techotopia*. N.p., n.d. Web. 03 Dec. 2013.

"Outline." *Objective-C Beginner's Guide*. N.p., n.d. Web. 03 Dec. 2013.