The Swift Language Handbook

An Introduction and Guidebook for Pascal Programmers

Henry Lowe

The Swift Language Handbook
An Introduction and Guidebook for
Pascal Programmers

Published in the United States by
Ascriva Health Informatics LLC, Stanford, California.

www.ascriva.com

Preface

Origins of This Book

I first programed in Pascal in 1982, using Apple UCSD Pascal for the Apple II family of computers. I was fascinated by Pascal's support for structured programming, significantly enhancing the design, implementation, and maintenance of code. The ability to create rich user-defined data types in Pascal was also a powerful feature of the language. At the time, Pascal impressed me as an elegant, compact, intuitive, and extensible language. It was also a "safe" language because of strong type checking and sparse use of pointers.

In 1982 I discovered Randy Clarke and Stephen Koehler's wonderful book: The UCSD Pascal Handbook - A Reference and Guidebook for Programmers. Using this well-designed book, one could quickly review UCSD Pascal's features, language and syntax - plus code examples. As I began using the Swift programming language 40 years later, I yearned for a similar short reference guide.

Not finding one that met my needs, I decided to create this book, which is designed to help those familiar with Pascal begin exploring Swift. Where possible, I have tried to demonstrate how these two elegant languages intersect, hopefully creating a conceptual bridge between them.

This book is not intended to be a comprehensive reference guide to Swift. Nor does it cover all aspects of the Swift language. Instead, it attempts to map the core concepts of Standard Pascal to the equivalent language concepts in Swift to hopefully facilitate that “Aha!” moment when one gets a firm conceptual foothold on a new programming paradigm.

The Pascal Programming Language

The Pascal programming language was designed by Nicklaus Wirth at the Federal Institute of Technology in Zürich, Switzerland, in 1970. Pascal is descended from the Algol programming language, which was first described in 1958. By the 1980’s Pascal had become the standard programming language used to teach college-level computer science. It evolved to become an object-oriented language (Object Pascal) and was used in the development of the original Apple Macintosh. In time, Pascal was eclipsed by other languages, particularly C, C++, Objective-C and Java. Pascal lives on today as Delphi Pascal and the Free Pascal Compiler (FPC).

The Swift Programming Language

Swift is a compiled general-purpose object-oriented programming language developed by Apple Inc. and the open-source community. Development of Swift began at Apple in 2010 by Chris Lattner, with the eventual collaboration of many other programmers. Swift was designed to replace Apple's earlier programming language Objective-C, which lacked many modern language features. Swift was first released in 2014 and, at the time of writing (April 2023), is at version 5.8, supporting all Apple platforms and Linux.

Henry Lowe
Stanford 2023

Tools and References

The Swift Playgrounds App

To experiment with the Swift code examples in this book I recommend installing Apple’s Playgrounds app for Mac and iPad. Playgrounds provides an interactive Read Eval Print Loop (REPL) environment into which you type Swift code and get instant feedback.

Launch the Playgrounds app and select ‘New Blank Playground’ from the Mac File Menu or tap ‘+ Playground’ in the iPad version of the app. This gives you a new empty playground into which you can enter Swift code. Entering ‘import Foundation’ as the first line of the playground ensures that Swift’s Foundation framework is available when you use the examples in this book.

Code Examples

Throughout this book you will find illustrative code examples.

Pascal code examples are printed as plain text, e.g:

```
Program Hello;
        Var Greeting:String;
        Begin
                Greeting:=‘Hello world';
                Writeln(Greeting);
        End.
```

Swift code examples are displayed within a colored text box:

```
let greeting = "Hello world"
print(greeting)
```

Swift’s Print() Statement

Swift’s print() statement is used in the code examples throughout this book. The print() statement sends output to the console. If you are using the Playgrounds app to experiment with Swift, the print() statement is a useful way to display constants, variables and data in the console pane of a playground window. For example, enter ‘print(“Hello world!”)’ And the text ‘Hello world’ is displayed in the playground’s console pane.

When using Swift’s print() statement all non-string variables are converted to strings automatically.

The full definition of print() is:

```
print(items, separator, terminator)
```

The ‘items’ parameter contains what we wish to display either as a literal, constant, variable or expression.

The optional ‘separator’ parameter defines the character used to separate multiple items within print(). The default separator is a single space character.

The optional ‘terminator’ parameter defines the character used to terminate the display output. The default terminator is new line character “\n”. Alternatives include: the tab character “\t” and the space character.

We can use these optional print() parameters as:

```
let word1 = "hello"
let word2 = "world"
print(word1,word2,separator:"-",terminator:"\t")
// displays hello-world
```

References

The definitive Swift reference is Apple’s ‘The Swift Programming Language’, which is available for free from the Apple Bookstore using Apple’s Books app. I highly recommend using this book as your standard Swift reference.

Online Swift documentation is available on the Apple Developer Site (subscription required) - https://developer.apple.com/documentation/swift/ and on the (free) Swift Open Source site at https://www.swift.org.

1 Introduction

Pascal and Swift share many core conceptual features.

1.1 The Form of a Program

A Pascal program has the general structure shown below:

```
Program HelloWorld;
   Var (* Declare a variable *)
       Greeting: String;
Begin
   Greeting:='Hello World';
   Writeln(Greeting);
End.
```

A Swift program may be written as:

```
var greeting: String //Declare a variable
greeting = "Hello World"
print(greeting)
```

Let's start with some obvious differences in the example: Unlike Pascal, Swift has no construct to indicate the start of a program. A Swift program begins with the first line of code, in this case: var greetings: String.

Pascal requires that statements end with a semicolon. This is not a requirement in Swift. However, a semicolon can be used to separate multiple statements written in the same line of Swift source code. While Swift does not require a semicolon after each statement, you can use one if you wish.

A major difference between Pascal and Swift is Swift 's (and many other modern languages) use of 'curly braces' i.e. **{ }** to define blocks of code. A code block is a scope (i.e., the set of all variable-name bindings visible to the compiler within a part of a program) that determines the lifetime of variables declared within the block. Variables declared within a block are not valid when control exits the block. Curly braces are also used to define the beginning and end of logical programming structures such as loops. Pascal uses Begin and End statements to define blocks of code. More on the use of curly braces later.

A single line comment in Swift begins with // and multi-line comments begin with /* and end with */. In Pascal source code comments can be enclosed in 'curly brackets' or parentheses e.g.: {Comment} (* Comment *).

1.2 Declaration

All variables in a Pascal program have a specific type. Some types are built into the language, while others are user-defined. Swift follows a similar model.

1.2.1 Case Sensitivity

Pascal is a case-insensitive language. You can use all upper case, all lower case or a mix of cases when writing Pascal code. Identifiers, constants, and variables are also case-insensitive. One consequence of this is that in Pascal the variable ‘MyVariable’ and ‘myVariable’ reference the same data.

Swift is a case-sensitive language: ‘MyVariable’ and ‘myVariable’ are two completely different variables.

One common problem that Pascal programmers encounter when starting to use Swift is forgetting to use the correct case when entering identifiers. Most Swift commands are all lower case. For example, use ‘print’ not ‘Print’ or ‘PRINT’:

```
print("Hello World") // is fine
Print("Hello World") // fails – uppercase ‘P’
```

Swift’s built-in data types are written with the initial character in uppercase e.g.: String, Int, Double, Bool, Array etc.

1.2.2 Constants

Constants are declared once in a program and their value cannot be changed during program execution. Constants can be declared anywhere in a program, but they must be declared before they are referenced. In Pascal constants are declared using the Const keyword:

```
Const
Dozen = 12;
Pi = 3.14159;
ErrMessage = ‘An error occurred’;
```

Swift does not use the Const keyword but instead declares constants using the ‘let’ keyword e.g

```
let Dozen = 12
let Pi = 3.14159
let errMessage = "An error occurred"
```

As in Pascal, Swift does not require that you define a constant's type, but you can if you wish. This is referred to as 'Type Annotation':

```
let Dozen: Int = 12
let Pi: Double = 3.14159
let errMessage: String = "An error occured"
```

In Swift multiple constants can be declared on a single line separated by commas e.g:

```
let minValue = 1,  maxValue = 31
```

Or

```
let minValue: Int = 1,  maxValue: Int = 99
```

As in Pascal, Swift also allows the use of expressions in constants:

```
let square2 =  2 * 2
```

In Swift, for performance reasons, it is recommended that you use constants to hold data that will not change during program execution.

1.2.3 Variables

In Swift (as in Pascal) a variable's value can be changed during program execution. Variables must be declared before they are referenced. Swift variables are declared as:

```
var userName: String
var startValue: Int = 100
var mean, median, mode: Int
```

Note that Swift variable declarations do not require ending the declaration with a semicolon, as is the case in Pascal, though you can add a semicolon if you so desire.

In Swift the following variable declarations are legal:

```
var userName: String
var userName: String;
```

In Swift it is usually not necessary (or required) to declare the variable type (Type Annotation), as the Swift compiler can usually infer the type from the declaration (Type Inference). The following variable declarations are legal:

```
var greetingMessage = “Hello World”
var startValue = 10
```

Once a variable or constant has been declared to hold values of a specific type, you cannot redeclare that constant or variable later in the program to hold values of a different type.

1.2.4 Constant and Variable Names

Swift constant and variable names can contain almost any character, including Unicode characters e.g.:

```
let 🐸 = "Frog"
```

Variable names cannot start with a number. Names cannot contain whitespace characters, math symbols, arrows, private-use Unicode scalar characters or line and box-drawing characters.

It is recommended (for readability purposes) to use a ‘Camel case’ naming convention when naming constants and variables in Swift. Camel case uses a lowercase letter for the first character of the first ‘word’ in a variable name followed by a capital letter for each subsequent ‘word’ e.g:

```
var myVariableName: String
```

Pascal programmers often use a similar convention, called ‘Pascal Case’ - in which the first character of a constant or variable name made of compound words is upper case - as opposed to lower case in Swift.

Var: MyFileName: String; - Pascal Case

var myFileName: String - Camel Case in Swift

Constant and variable names in Swift can be of any length and all characters are significant. As Swift is a case-sensitive language the following are considered separate variables:

```
var MYVARIABLENAME: String
var myvariablename: String
var myVariableName: String
```

2 Operators

Pascal and Swift support a similar set of operators, though with different syntax.

2.1 Assignment

Pascal uses the ‘**:**=‘ operator for assignment. For example:

```
Var Int1: Integer;
Int1:=42;
```

Swift uses the ‘=’ operator for assignment. For example:

```
var int1,int2,int3:Int
int1 = 42
```

2.2 Equality

The equality operator in Pascal is ‘=’. For example:

```
If int1 = int2 then …
```

In Swift the equality operator is ‘==‘, e.g.:

```
var int1,int2:Int
int1 = 1; int2 = 1
if int1 == int2 {
    let equal = true }
```

2.3 Non-Equality

In Pascal we test for non-equality using the ‘<>’ operator:

```
If int1 <> int2 then …
```

In Swift we test for non-equality using the ‘!=’ operator:

```
var int1,int2:Int
int1 = 1
int2 = 1
if int1 != int2 {
    let equal = false
}
```

In both Pascal and Swift to test if one variable is less than another we use the ‘<‘ operator, and use the ‘>’ operator to test if one variable is greater than another:

```
var int1,int2:Int
int1 = 1
int2 = 1
if int1 > int2 {
    let greaterThan = false
}
```

The Greater Than Or Equals To operator ‘>=‘ and the Less Than Or Equals To operator ‘<=‘ are also the same in both Pascal and Swift.

2.4 Arithmetic Operators

Pascal and Swift use the same operators for addition ‘+’, subtraction ‘-‘, division ‘/‘ and multiplication ‘*’:

```
var int1,int2,int3:Int
int1 = 1
int2 = 1
int3 = (int1 + int2) * (int2 / int1) - int1
```

The Remainder operator returns the remainder when one

```
Int3:=int2 Mod Int1;
```

number is divided by another. In Pascal this is called ‘Mod’, which is short for Modulo:
Swift uses the Remainder operator ‘%’ in a similar fashion:

```
var int1,int2,int3:Int
int3 = int2 % int1
```

In both Pascal and Swift one can change the sign of a number by prefixing a numeric value with the ‘-’ operator (the Unary Minus Operator). This operator toggles the sign of a number:

```
var int1:Int
int1 = 1
int1 = -int1 // int1 = -1
int1 = -int1 // int1 = 1
```

The Unary Plus Operator ‘+’ does not change the sign of the number and its inclusion in both Pascal and Swift appears to be for symmetry.

2.4 Logical Operators

Both Pascal and Swift support the Logical Operators AND, OR and NOT:

The AND operator is ‘AND’ in Pascal and ‘&&’ in Swift.

In Pascal:

```
Var Bool1,Bool2:Boolean;
Bool1:=True;
Bool2:=False;
If Bool1 AND Bool2 then
      Writeln(‘True’)
      Else
      Writeln(‘False’);
```

In Swift:

```
var bool1:Bool = true
var bool2:Bool = false

if bool1 && bool2 {
    print ("True")
}    else  {
    print ("False")
}
```

The OR operator is ‘OR’ in Pascal and ‘||’ in Swift.

In Pascal:

```
Var Bool1,Bool2:Boolean;
Bool1:=True;
Bool2:=False;
If Bool1 OR Bool2 then
      Writeln(‘True’)
      Else
      Writeln(‘False’);
```

In Swift:

```
if bool1 || bool2 {
    print ("True")
}    else  {
    print ("False")
}
```

The NOT operator is ‘NOT’ in Pascal and ‘!’ in Swift.

In Swift Logical NOT is a prefix operator (i.e., appears just before the value it operates on, without whitespace) that toggles a Boolean value.

```
let bool1 = false
let bool2 = !bool1 // bool2 = true
```

In Pascal **NOT** is used in a similar fashion:

```
Bool1:=False;
Bool2:=Not Bool1; {Bool2 is True}
```

2.5 Range Operators

Swift includes several range operators that define a range of values. The closest equivalent construct in Pascal are subranges. In Pascal we can define a type, constant or variable as a subrange of any scalar type. For example:

```
digits = 0..9; (* Subrange of Integer *)
Grade = “A,B,C,D,E,F” (* User-defined type *)
pass = ‘A’..’C’; (* Subrange of Grade *)
```

Pascal may also use subranges when defining arrays:

```
myArray[1..99] of String;
```

Swift’s Closed Range Operator (‘x...y’) defines a range extending from the value of x to the value of y. For example, 2...7 is a range of integers from 2 to 7. The value of x must not be greater that the value of y. Note the three periods used by Swift as opposed to the two periods used in Pascal.

Swift’s Half-Open Range Operator (x..<y) defines a range starting with the value of x but not including the value of y. For example, 2..<7 is a range of integers starting with 2 and ending with 6. This type of range operator is useful when working with zero-based lists, such as Swift arrays. The value of x must not be greater that the value of y. Note the use of two periods in this type of range.

Swift’s One-Sided Range operator define a range that starts with a value and extend as far as possible beyond that start value. For example, if an array contains 100 members, e.g. Array[0..99] then the One-Sided Range [9…] will include the array members Array[9] to Array[99]. We can also use a form of the One-Sided Range in which the left side of the range is undefined, e.g. […9] would reference array members 0 to 9.

Ranges are often used in Swift for-in loops. In Pascal we might define a for-loop as:

```
for index:= 1 to 9
      writeln(index);
```

In Swift we can iterate in a similar fashion using ranges:

```
var index1,index2,index3: Int
var myArray: [Int] = [1,2,3,4,5]
for index1 in 1...9 {
    print(index1) }
for index2 in 1..<10 {
    print(index2) }
for index3 in myArray[1...] {
    print(myArray[index3]) }
```

Because Swift ranges are a type, there are range properties and methods that we can use:

```swift
let myRange = 0...9
print(myRange.contains(5)) //true
print(myRange.lowerBound) // 0
print(myRange.upperBound) // 9
print(myRange.isEmpty) //false
```

Swift also supports comparing ranges:

```swift
let myRange = 0...9
let anotherRange = 1...8
print(myRange == anotherRange) //false
print(myRange !=  anotherRange) //true
```

We can also check if ranges overlap:

```swift
print(myRange.overlaps(anotherRange)) //true
```

Index

About the Author

Henry Lowe MD, FACMI is Emeritus Professor at Stanford University and a Board-Certified Internal Medicine physician. From 2002 until 2013 Dr. Lowe was Chief Information Officer and Senior Associate Dean for Information Resources and Technology at Stanford University School of Medicine. He was also the founding Director of Stanford's Center for Clinical Informatics. He has extensive experience in clinical informatics research and development. Dr. Lowe is an elected Fellow of the American College of Medical Informatics (ACMI) and the Founder of Ascriva Health Informatics, a company based in Silicon Valley, California.