Sections

Introduction

Getting started

Chapter 1.) Data types (including Date)

1.1 Int

1.2 Float

1.3 String

1.4 Bool

\*\* Examples should be included on each subsection

Chapter 2 Arrays (including unbounded)

2.1 Declarationand Unbounded arrays and more

2.2 Iteration

Chapter 3 Operations (including infix vs prefix, IN, NOTIN, additional numeric assignment)

3.1 what constitutes an expression?

3.2 Operators

3.3 precedence

3.4 assignment??

Chapter 4 Flow Control (if, while, for, select, break, continue, for tokenizer)

4.1 if

4.2 while

4.3 for

Chapter 5 Builtin Functions

Flexible Points Completed

You must provide your own test data for

* unbounded arrays
* additional numeric assignment operators
* IN, NOTIN
* slices
* for tokenizer
* variable number of arguments to a function
* recursive use of a function
* integration between different capabilities (e.g., slices with IN, slices as arguments, slices as function result, unbound arrays as arguments, break/continue in select)

**Introduction**

Let’s start with a brief introduction to HavaBol. It is a strongly typed interpreted language. The interpreter parses and compiles all HavaBol source code into Java bytecode. The scope of this language is global. (more details in each section) This is a simple language with basic control flow mechanisms : for, foreach, if, while and just a handful of builtin functions. The syntax is very Python like but what this language can do as far as any complicated concepts is very limited. This would be a good beginner language. The aim of this document is to serve as a reference to the programmer on how to use HavaBol. Hope you enjoy and make sure you have a ball .

**Getting Started**

First Program:

As with any programing language the first program will be ‘hello world’. Heres that program in Havabol:

print(“Hello World!\n); // I’m a comment and the interpreter ignores me!

HavaBol supports one line comments. They are started will // as shown above.

‘print’ is a builtin function that takes in string literals and other intialized variables as arguments. Its prints to standard out (aka the screen by default)

A string literal is indicated by the double quotes: “String literal” is a string literals

The backslash n is a special character which will be discussed in chapter one when going over strings. (it just means add a newline)

The print builtin function by default will ALWAYS add a newline, even if not specified.

The above program prints out: ‘Hello World!’ followed by 2 newline characters

**Chapter One : Data Types**

**Section 1.1 type Int**

This section describes the type ‘Int’. ‘Int’ represents the type integer. ‘Int’ is just the keyword used upon declaration of a variable to indicate that the type is an integer. An integer in this language is 4 bytes. An integer’s value can range is from -2,147,483,648 to 2,147,483,647.

Example declaration of a variable called num of type Int being initialized to value 100:

Int num = 100;

Example of declaration without initialization:

Int num;

Format for declaration: type varName; or type varName ‘=’ expression;

\*\*Note\*\* A concise definition of expressions will be outlined in Chapter 3.

The type is required and obviously the variable name is required and a semicolon is required as well.

Initialization does not have to be done upon declaration. Regardless of where a variable is declared in HavaBol, its scope is global. This language has no other scope.

\*\*Note: ‘Int’ can be used to declare a simple variable or to declare an array of type int.

Example: Int inums[3] = 2, 4, 8;

\*\* Arrays will be covered in detail in Chapter 2.

\*\*NOTE: For ALL types, before any operations can be performed a variable must be initialized first. More on this in Chapter 3.

A note about type casting for type Int.

The right hand side will always be cast to the type of the left handside (of an equals for example) when possible.

i.e. Int num = 3.14; will result in the value 3 being assigned to num as it was cast to an integer before assignement.

rules for Int casting:

Int → Float YES , will cast float to int, rounding the value down to the nearest integer

Int → String NO, will cause an error, even if the string is an integer. I.e Int num = “10”;

Int → Bool NO, will cause an error

**Section 1.2 type Float**

This section describes the type ‘Float’. ‘Float’ represents a double-precision 64 bit IEEE 754 floating point. ‘Float’ is just the keyword used upon declaration of a variable to indicate that the type is a double value. Type ‘Float’ in this language is 8 bytes.

Example declaration of a variable called fnum of type Float being initialized to value 3.14:

Float fnum = 3.14;

Example of declaration without initialization:

Float fnum;

Format for declaration: type varName; or type varName ‘=’ expression;

\*\*Note\*\* A concise definition of expressions will be outlined in Chapter 3.

The type is required and obviously the variable name is required and a semicolon is required as well.

Initialization does not have to be done upon declaration. Regardless of where a variable is declared in HavaBol, its scope is global.

\*\*Note: ‘Float’ can be used to declare a simple variable or to declare an array of type int.

Example: Float nums[3] = 3.14, 2.22, 1.1;

\*\* Arrays will be covered in detail in Chapter 2.

\*\*NOTE: For ALL types, before any operations can be performed a variable must be initialized first. More on this in Chapter 3.

A note about type casting for type Float.

The right hand side will always be cast to the type of the left handside (of an equals for example) when possible.

i.e. Float myFloat = 3; will result in the value 3.00 being assigned to num as it was cast to a Float before assignment. (2 decimal points added)

rules for Float casting:

Float → Int YES , will cast int to a float, adding a decimal point and 2 zeros

Float → String NO, will cause an error, even if the string is an Float. I.e Float num = “10.1”;

Float → Bool NO, will cause an error

**Section 1.3 type String**

This section describes the type ‘String’. ‘String’ represents the type ‘String’. It is not abbreviated at all. The string is actually the Java String class behind the scenes. ‘String’ is just the keyword used upon declaration of a variable to indicate that the type is a String . The size of type string is not set in stone. As it is Java string class its size starts of at 36 bytes (due to necessary references and other values) plus 2 bytes \* length of the string. A String can be almost any ASCII character plus a few non-printable ones. The non-printables include:

\n - newline character

\t - the tab character

\a - the alarm bell character

Example declaration of a variable called str of type String being initialized:

String str = “Hello\tWorld!\n”;

Example of declaration without initialization:

String str;

Format for declaration: type varName; or type varName ‘=’ string literal ;

The type is required and obviously the variable name is required and a semicolon is required as well.

Initialization does not have to be done upon declaration. Regardless of where a variable is declared in HavaBol, its scope is global. This language has no other scope.

Type String also can be indexed like an array. For example:

String name = “Chris\n”;

name[5] = “t”;

print(name); // this will print ‘Christ’

Also String is the only scalar type that is iterable. That means you can use a for counter loop or a foreach loop to iterate thru the individual characters in the String type.

\*\*\*Note: more on control flow (for loops and such) in chapter 4.

\*\*Note: ‘String’ can be used to declare a simple variable or to declare an array of type String.

Example: String names[3] = “Chris”, “Matt”, “Miguel”;

\*\* Arrays will be covered in detail in Chapter 2.

\*\*NOTE: For ALL types, before any operations can be performed a variable must be initialized first. More on this in Chapter 3.

A note about type casting for type String.

The right hand side will always be cast to the type of the left handside (of an equals for example) when possible.

i.e. String str = 10; will result in the value “10” (a string) being assigned to str as it was cast to a String before assignment.

rules for String casting:

String → Int YES , will cast Int to a String

String → Float YES, will cast Float to a String

String → Bool YES cast Bool to String

**Section 1.4 type Bool**

This section describes the type ‘Bool’. ‘Bool’ represents the type boolean. Aka true or false. ‘Bool’ is just the keyword used upon declaration of a variable to indicate that the type is a boolean. ‘Bool’ s values are either true or false. True is represented is HavaBol by ‘T’ and false is represented by ‘F’ (no quotes needed at all).

Example declaration of a variable called ‘b’ of type Bool being initialized to value true:

Bool b = T;

Other Bool examples:

Bool b2 = 10 > 2; //Bool b2 = ‘expression that can be evaluated to True or False’

\*Note: b2 was assigned the value T because the expression evaluated to true.

Example of declaration without initialization:

Bool b;

Format for declaration: type varName; or type varName ‘=’ expression;

\*\*Note\*\* A concise definition of expressions will be outlined in Chapter 3.

The type is required and obviously the variable name is required and a semicolon is required as well.

Initialization does not have to be done upon declaration. Regardless of where a variable is declared in HavaBol, its scope is global. This language has no other scope.

\*\*Note: ‘Bool’ can be used to declare a simple variable or to declare an array of type Bool.

Example: Bool boolVals[3] = T, 10 > 2, F;

\*\* Arrays will be covered in detail in Chapter 2.

\*\*NOTE: For ALL types, before any operations can be performed a variable must be initialized first. More on this in Chapter 3.

**Chapter 2 Arrays**

**Section 2.1 Declaration, Unbounded arrays and more**

This section describes the declaration for all types of arrays. Arrays can be of any type: Int, Float, String, Bool. Unlike a simple int, bool, string or float, an array is not a scalar variable. It is the only builtin data structure that HavaBol has.

\*\* A Note on arrays before diving in:

- Arrays can be declared of a fixed size or can be unbounded (dynamic).

- Array to array assignment of all elements in array is also supported

- An array is iterable and can be iterated over using a for loop.

- ‘unbound’ is the keyword to declare an unbounded array

- a fixed size array has out of bounds checking and will throw an error if the user tries to index something that larger than the size of the array

Array declaration examples:

Int array[10]; - declared to be a fixed size array, type Int of 10 elements, no elements initialized

Float array[unbound]; - declared to be an unbounded array of type Float, array has one element initialized to null

Int nums[] = 1, 2, 3, 4; - declared to be a fixed size array of 4 elements each of which initialized

Int nums[4] = 1, 2, 3, 4; – the same as above but being more explicit about size. If you dont specify a size in the brackets upon declaration HavaBol will count the number of elements being initialized and set the size to that.

Int nums[10] = 1, 3.14, 2.22; - declared to be a fixed array of size 10. the first 4 elements are initialized but both floats (3.14 and 2.22) are cast to Ints before initialization. (elements are 1,3,2)

String elems[10] = “cat”, 10, 3.14; - String type array, fixed size of size 10 declared and three elements initialized: a string literal (“cat”), an integer and a float. All values were cast to type String.

\*Note: although a Bool type can be cast to a String type with scalars. Arrays are different. You cannot try to initialize a Bool type and try to cast to a String upon array declaration.

Example:

String elems[10] = 1, “hello”, T, “bye”;

This will error out when trying to initialize the boolean value ‘T’ as it cannot be cast to a string during array declarations.

Also , you cannot declare a fixed array of size 0 as it will also throw an error.

Example:

Int array[0]; // this will error out!

Arrays also support array to array copying in one line.

\*\*Note: both arrays must be of same type \*\*

Example:

Int nums[10] = 1,2,3,4,5; (size 10)

Int moreNums[] = 6,7,8,9,10,11,12,13,14,15; (also size 10)

nums = moreNums;

Since nums is of size 10 , all 10 values of moreNums will be copied over to nums so if:

print(nums);

was executed.

Output→[6, 7, 8, 9, 10, 11, 12, 13, 14 ,15]

If arrays of of different sizes:

Example:

Int array[3] = 1,2,3;

Int array2[5] = 5,4,3,2,1;

array = array2;

Only the first 3 values of array2 would be copied to array. If we were to print(array);

the output would be:

[5, 4, 3]

We can have ‘holes’ in arrays on certain conditions. When I say ‘holes’ I mean that the value in that index will be null.

Examples:

Int array[unbound]; - declare unbounded

array[10] = 1000; - set index 10 to 1000

The first 9 values of this unbounded array would be null’s while only one index would have value 1000(the last index)

\*Note: this is a way to grow an unbounded array.

Int array[10] = 1,2,3;

array[5] = 1000;

If we print out array we get:

[1, 2, 3, null, null, 1000, null, null, null, null]

We set the first 3 values to numbers and then set the sixth value (index 5) to 1000. The array is of size 10 so all uninitialized values are set to null.

**Section 2.2 Iteration**