Scribble Language

Types

There are several predefined types within Scribble. A developer can also declare his own types to be one of the existing types or a data structure.

The currently supported types are int, bool and string.

Note: Currently floats, chars and longs are not included within the language however I plan that they will be (Especially floats)

Functions

Scribble requires that every instruction be part of a function.

Functions come in the form

```
func Name ( Arguments ) : Type {
}
```

If type is not stated explicitly it defaults to Void. So writing

```
func DoAThing() {
}
```

is valid.

Arguments are written like name : Type separated with a comma.

```
func Name ( id : int, age : int ) : string {
	return “TestName”;
}
```

You are allowed to specify multiple functions of the same name which take different arguments as long as they have the same return type. The compiler will work out which one to use at compile time and each one will be given a unique name for the VM. For example the code

```
func IsTrue( v : bool ) : bool {
	if v then return true;
	else return false;
}

func IsTrue( v : int ) : bool {
	if v = 0 then return false;
	else return true;
}
```

Expressions

Expressions are in a C style. They are a combination of possible operators and operands to effect the program. Expressions can be placed into brackets so (3 + 9) + 6 would be an expression.

Variables

Variables are defined using the var keyword in the form var Name : Type;

A new variables type can be inferred by using the assign operator so var i : int := 0; can be written as var i := 0; This type inference is done during compilation by setting the variables type to be the type returned by the execution of whatever statement it is being assigned to.

Assignment

To assign a value to a variable you use the ':=' assignment operator. For instance you type var HelloWorld := “Hello world”; a new variable will be created and its value will be set to the string “Hello World”.

Arithmetic

The standard +, -, * and / operators are used to handle arithmetic. Any more advanced functions such as modulo can be implemented through functions (The default sys namespace contains a Modulo function)

The operators Name++, ++Name, Name-- and –Name can be used to increment and decrement a variable. This functionality works in the same way as C.

Comparison

There are several possible comparisons in Scribble.

< - Test less than
<= - Test less than or equal to
> - Test greater than
>= - Test greater than or equal to
= - Test equality.
& - Local and. If left hand side is false then right hand side will not be tested.

Comparisons can only be done on statements of the same type. The comparison is done using the syntax LHS Operator RHS. So to test equality you would use 0 = 1 or to test less than you could use 5 < 0.

Flow control

While loops are written like

```
while BooleanValue do {
        Statements
}
```

While loops will continue executing their body until the condition expression returns false when executed.

For example

```
var temp := true;
while temp do {
        temp := false;
}
```

For loops are written like

```
for Initialize; Condition; Step do {
}
```

where Initialize is a statement that sets up the for loop variables (E.g. var I := 0;) Condition is a boolean expression that will return true while the for loop should continue and step is a statement that will be executed each time the body of the for loop is executed (E.g. i++). This functionality and syntax is similar to the standard C for loop.

For example

```
for var i := 0; i < 100; i++ do {
}
```

If statements come in the form

```
if X then Statement;
if X then { Statements; }
```

where X is a boolean expression. They can either be provided with a single statements or a set of statements in brackets.

either of the previous two can be combined with else
if X then Statements else Statements;

For example

```
var testValue := true;

if testValue then sys.Write("hello");
else if testValue = false then sys.Write("world");
```

is valid. As is

```
var testValue := true;

if testValue = true then {
	sys.Write(“Hello”);
} else if testValue = false then {
	sys.Write(“World”);
}
```

Arrays

The type of an array is declared by 'array (subtype)' for example an integer array is declared like 'array(int)'. This is used when declaring variables as arrays before they are initialized such as in a function argument.

For example declare that argument n in function fn is an array.

```
func fn(n : array(int)) {
	//Do array things
}
```

To create an array the syntax [number]type is used. So an array of 100 integers would be created using [100]int. So to call the fn function above you could use fn([1000]int);

An arrays length will also be stored with the array. Calling len(Array) will return the length of that array as a integer. So len([100]int) will equal 100.

Namespaces

Every file will be regarded as a namespace. What this will mean in practice is that functions and structures placed in one file will need to be explicit about the namespace when calling functions in another file. The other namespace will also need to be 'imported' which makes sure that if the other namespace hasn't been built already then it will be.

In practice this looks like

```
import(“sys”)

func Test() {
        sys.Write ( sys.String( 150 ) )
}
```

Custom Types

In Scribble it is possible to create your own types. Either as a previous type or with a new data structure. This is done using the syntax.

```
type Name := type;
```

or

```
type Name := struct {
        First : string,
        Last : string
}
```

A type set to another type is initialized in the same way the other type would be. A structure is initialized using a special syntax. The syntax is in the form StructureName { Element1Value, Element2Value, … }

so to initialize the structure above you would use the syntax

```
var structInstance := Name { “John”, “Smith” };
```

To access an element within a custom structure you use the → operator. So after executing the code

```
var structInstance := Name { “John”, “Smith” };
var name := structInstance->First;
```

The value of the variable name would be “John"

Working Example

The file copied in below gives a working example of bubble sort demonstrating some flow control, variables arrays and name spaces as well as some of the default system calls. It can be found in the repository under examples/sorts/bubble.

```
import ("sys")
import ("populate")


func Sort( n : array(int) ) {

 var swapped := true;

 for var i := len(n) - 1; (i > 0) & (swapped = true); i-- do {

      swapped := false;

      for var j := 0; j < i; j++ do {

            if (n[j]) > (n[j+1]) then {
```

```
                var temp := n[j];
                n[j] := n[j+1];
                n[j+1] := temp;
                swapped := true;
            }
        }
    }
}

func Test() {

    var list := [100]int;

    populate.PopulateWorstCase(list);
    Sort(list);

    for var i := 0; i < len(list); i++ do {
        sys.Write(sys.String(list[i]));
        sys.Write(" ");
    }

    sys.Write("\n");

}
```