Shank Language Definition

Shank is different in some significant ways from languages that you many already know.

The biggest differences are:

1) Shank functions don't have return values.
2) Shank functions can alter variables that are passed into the function, when the variable is marked as "var"
3) Shank doesn't use curly braces {} for blocks – it uses indentation.
4) Shank for loops are much simpler than Java or C for loops.
5) Assignment uses :=, but comparison uses =

Overall Function Format

The overall format of a function:

function definition (define)
constants (optional)
variables (optional)
body

For example:

```
define start ()
constants pi=3.141
variables a,b,c : integer
      a := 1
      b := 2
      c := 3
```

Comments

Comments can span multiple lines. They start with { and end with }

Example:

```
{ This is a comment }
```

Blocks

Blocks are one or more statements that are run one after another. Blocks are indented one level more than the "owner" of the block. To end a block, you simply "un-indent" one level.

Indentation on empty lines or lines that contain only a comment is not counted.

Example:

```
for a from 1 to 10
      write a
      write a+1
write "not in the block"
```

Built-in types:

integer (32-bit signed number)
real (floating point)
boolean (constant values: true, false)
character (a single number/letter/symbol)
string (arbitrarily large string of characters)

Variables

Variable declarations are defined by the keyword variables, a list of names, then a ":" and then the data type. A name must start with a letter (lower or upper case) and then can have any number of letters and/or numbers. There can be more than one variables line in a function.

Example:

```
variables variable1, a, foo9 : integer
variables name, address, country : string
```

Constants

Constants are variables that are set at definition and cannot be changed after definition. They do not require a data type since the data type is inferred from the value. There can be more than one per function. The data type of the function need not be consistent in the single line.

```
constants myName = "kmeltzer"
constants pi = 3.141
constants year = "2023", month = 5
```

Functions (also known as: Procedures/Methods/Subroutines)

A function is an (optional) constant section, an (optional) variable section and a block. Functions have a name and a set of parameters; this combination must be unique.

Function parameters are read-only (treated as constant) by default. To allow them to be changed, we proceed them by the keyword "var" both in the function declaration and in the call to the function.

Example:

```
define addTwo(x,y : integer; var sum: integer)
      sum := x + y
```

To call this function:

```
addTwo 5,4,var total { total was declared somewhere else }
```

The var keyword must be used before each variable declaration that is alterable.

```
define someFunction(readOnly:integer; var changeable : integer; alsoReadOnly
: integer)
someFunction someVariable, var answer, 6
```

In contrast to other languages, functions never return anything except through the “var” variables. While this is unfamiliar to people who have used other languages, it is actually powerful, since you can return as many values as you choose.

```
define average(values:array of integer; var mean, median, mode : real)
```

When the program starts, the function “start” will be called.

Control structures and Loops

The only conditional control structure that we support is “if-elsif-else”. Its format is:

if booleanExpression then block {elsif booleanExpression then block}[else block]

Examples:

```
if a<5 then
begin
      a := 5
end
end if


if i mod 15=0 then
begin
      write “FizzBuzz “
end
elsif i mod 3=0 then
begin
      write “Fizz “
end
elsif i mod 5 = 0 then
begin
      write “Buzz “
end
else
begin
      write I, “ “
end
```

There are three types of loops that we support:

for integerVariable from value to value
block

while booleanexpression
block

repeat
block
until booleanExpression

Note – the control variable in the for loop is **not** automatically declared – it must be declared before the for statement is encountered.

Examples:

```
for i from 1 to 10
      write i { prints values 1 … 10 }

for j from 10 to 2 { this loop will never execute}
      write j

while j < 5
      j:=j+1

repeat
      j:=j-1
until j = 0
```

Since these are statements, they can be embedded within each other:

```
if a<5 then
      repeat
            for j from 0 to 5
                  while k < 2
                        k:=k+1
            a := a + 1
      until a=6
end
```

Operators and comparison

Integers and reals have the following operators: +,-,*,/, mod. The order of operations is parenthesis, *,/,mod (left to right), then +,- (also left to right).

Booleans have no operators.

Characters have no operators.

Strings have only + (concatenation) of characters or strings.

Comparison can only take place between the same data types.

= (equals), <> (not equal), <, <=, >, >= (all done from left to right).

Built-in functions

I/O Functions

Read var a, var b, var c (* for example – these are variadic *)

Reads (space delimited) values from the user

Write a,b,c (* for example – these are variadic *)

Writes the values of a,b and c separated by spaces

String Functions

Left someString, length, var resultString

ResultString = first length characters of someString

Right someString, length, var resultString

ResultString = last length characters of someString

Substring someString, index, length, var resultString

ResultString = length characters from someString, starting at index

Number Functions

SquareRoot someFloat, var result

Result = square root of someFloat

GetRandom var resultInteger

resultInteger = some random integer

IntegerToReal someInteger, var someReal

someReal = someInteger (so if someInteger = 5, someReal = 5.0)

RealToInteger someReal, var someInt

someInt = truncate someReal (so if someReal = 5.5, someInt = 5)