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 begin with the keyword "begin" and end with the keyword "end".

Example:

```
begin
      (* this block doesn't do anything *)
end
```

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)
array of (any of the above)

Arrays

Arrays are declared much like other variables:

```
variables
      names : array from 0 to 5 of string
```

All arrays are 1 dimensional and must have a range declared at definition time. Referencing a variable is done using square brackets:

```
      names[0] := "mphipps"
      write names[0]
```

Variables

Variable declarations are formed by 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.

Example:

```
variables
      variable1, a, foo9 : integer
      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.

```
constants
      myName = “mphipps”
      pi = 3.141
```

Type limits

Types can be limited at declaration time using “from” and “to”. Does not apply to booleans.

Example:

```
variables
      numberOfCards : integer from 0 to 52
      waterTemperature: real from 0.0 to 100.0
      shortString : string from 0 to 20 (* string has a length limit *)
```

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)
begin
      sum := x + y
end
```

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
begin
      write i
end

for j from 10 to 2
begin
      write j
end

while j < 5
begin
      j:=j+1
end

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

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

```
if a<5 then
begin
      repeat
      begin
            for j from 0 to 5
            begin
                  while k < 2
                  begin
                        k:=k+1
                  end
            end
            a := a + 1
      end
      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: not, and, or. The order of operations is not, and, or.

Characters have no operators.

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

Arrays have no operators.

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)

Array Functions

Start var start

start = the first index of this array

End var end

end = the last index of this array

Overall Format

```
define start (args : array of string)
constants
      pi=3.141
variables
      a,b,c : integer
begin
      a := 1
      b := 2
      c := 3
end
```

Future:

Enums, Records