

Variables in Yabasic 3 (some preliminary ideas, version two)

Thomas Larsen

August 19, 2010

Hello all

- Here are some of my suggestions for how variables could work in Yabasic 3.
- Your comments and questions would be greatly appreciated! Just respond at the Yabasic blog: <http://yabasic.basicprogramming.org/blog>.
- Nothing has been concretely decided upon yet. All input is welcome.
- Please let me know if you think anything major is missing.

Variables

- Variables identify values.
- Initially, variables are typeless.
- Once a variable has been assigned a value, it can only hold values of the same type for the remainder of the program.
- Variables in action:
`pi = 3.1415`
`name = "Bob"`

Values

- There are two value types:
 - ◆ real numbers (for example, 0, 1, 3.14, -753)
 - ◆ character strings (for example, the empty string "", "z", "Hello, world!", "oranges").

Scalars

- Scalars are variables that can hold only one value:

```
town = "Tinyville"
```

```
population = 100
```

Arrays

- Arrays are variables that can hold more than one value:
town (1) = "Randomville"
population (1) = 242
town (2) = "Somewhereton"
population (2) = 583
- The position of an element in an array is specified using indices.
- Arrays can have any number of dimensions.
- Arrays expand and contract dynamically.

Arrays (continued)

- Array indices can be:
 - ◆ integers (0 and the natural numbers only)
 - ◆ character strings (excluding the empty string "")
 - ◆ a combination of integers and character strings.
- An error will be reported if a real number that is not an integer is used as an index.
- The order of array indices *does* matter.
- Here are some possible associative arrays:

```
crate ("fruit", "apples") = 93
population ("Random" + "ville") = 242
student_subject ("Alice", 1) = "CS101"
```

Records

- Records are implemented using associative arrays:
 - ◆ `a.b` is syntactical sugar for `a ("b")`
 - ◆ note that `a.b` is *not* equivalent to `a (b)`.
- Both `a` and `b` must be valid symbols for this to work.
- Take these records:

```
crate.fruit.oranges
```

```
crate ("fruit").oranges
```

```
crate.fruit ("oranges")
```

All of these actually represent:

```
crate ("fruit", "oranges")
```

Initialisation

- Variables are dynamic.
- Neither scalar nor array variables need to be declared before use.
- However, an error will be reported if a program tries to access a variable that has not previously been assigned a value.

Subroutine parameters

- Parameters passed to subroutines are stored in variables:
values (1, 2, 3)

```
sub values (a, b, c)
  print a, b, c // output will be "1 2 3"
  return
end sub
```

- Each parameter must correspond to a variable, and vice versa; an error will be reported if too many or too few parameters are passed to a subroutine.
- Scalars are always passed by value, arrays are always passed by reference.

Subroutine parameters (continued)

- Partial array references can be passed as subroutine parameters:

```
crate.fruit ("apples") = 253
crate.fruit ("oranges") = 221
count_fruit (crate.fruit)
```

```
sub count_fruit (fruit)
  print fruit.apples, "apples,", fruit.oranges, "oranges"
  return
end sub
```

Namespaces

- All variables are given a namespace:
 - ◆ variables in a subroutine are prefixed with the name of the subroutine and “:”
 - ◆ variables outside a subroutine (in the main body of a program) are prefixed with “main:”.
- A variable is said to be “local” to its namespace.
- To access a variable in *another* namespace, the namespace prefix must be prepended to the variable name.

Namespaces (continued)

- Here's an example to clarify how namespaces work:

```
a = 1000
```

```
value_1 (42)
```

```
sub value_1 (a)
```

```
  print a, main:a           // output will be "42 1000"
```

```
  value_2 ()
```

```
  return
```

```
end sub
```

```
sub value_2 ()
```

```
  a = 512
```

```
  print a, value_2:a, value_1:a, main:a
```

```
                                // output will be "512 512 42 1000"
```

```
  return
```

```
end sub
```