Forth Golfscript Interpreter

Golfscript

Golforth

Code Golf

- shortest possible source code that implements an algorithm
- solving problems (holes) in as few keystrokes as possible

Golforth

Code Golf

- shortest possible source code that implements an algorithm
- solving problems (holes) in as few keystrokes as possible

Golfscript

- stack oriented, variables exist
- single symbols represent high level operations
- strong typed
- heavy use of operator overloading and type coercion

Golforth

Golfscript Types

- Integer: `1 2`
- Arrays: `[1 2 3] [3]`
- Strings: `"one two three"`
- Blocks: `{1+}`

Golforth

Golfscript Types

- Integer: `1 2`
- Arrays: `[1 2 3] [3]`
- Strings: `"one two three"`
- Blocks: `{1+}`

Golfscript Operator Example

- `12 3 * -> 36`
- `[50 51 52]' '* -> "50 51 52"`
- `[1 2 3]{1+}/ -> 2 3 4`
- `{.@\%.}do; ( n1 n2 -- gcd )`

Forth Implementation

Golforth

Typesystem

- Values as scalar references on stack
- Anonymous functions
  - `: anon_int { u -- typext } :noname u POSTPONE LITERAL POSTPONE typeno_int POSTPONE ; ;`

```
12 anon_int s" foo" anon_str
```

$$\downarrow$$

```
2 elements on stack (12 and 'foo')
```

Golforth

Parser

- translates golfscript to forth execution tokens
- based on regular expression of reference implementation
- Responsible for:
  - infer initial type from syntax
  - symbol table for variable tracking
  - note that every value can be a variable!

```
12 'foo':x
```

$\downarrow$

(creating x in symbol table)

$\downarrow$

```
12 anon_int s" foo" anon_str dup x !
```

Golforth

Arrays

- Construction similar to postscript.
- **[** marks stack size, **]** collects back to marked size.
- Mark moves when stack becomes smaller:

```
1 2 [\] -> [2 1]
```

Golforth

Blocks

- Stored as execution tokens
- Operations: `2{1+}+` $\rightarrow$ `{2 1+}`
  implemented as function composition
- Execution via `execute`

Golforth

Conditionals and Loops

- `5{1-..}do` $\rightarrow$ 4 3 2 1 0 0
- `5{.}{1-.}while` $\rightarrow$ 4 3 2 1 0 0
- `5{.}{1-.}until` $\rightarrow$ 5
- implemented as words which consume code blocks

Golforth

Type Coercion and Overloading

- Typeorder for Coercion
- Coercion according to highest order type
- Heavy operator overloading results in wide range of functionality

Hier koennten die demos beginnen:
Operator in shell zeigen und evtl entsprechende implementierung

Golforth

~	100
‘	0
!	100
@	100
$	0
+	100
-	75 not for blocks
*	0
/	0
%	25
|	0
&	0
^	0
\	100
;	100
<	0
>	0
=	88
,	75
.	100
?	75
(	75
)	75
and or xor	0
print p n puts	0
do	100
while until	0
if	0
abs	0
zip	0
base	0

Golforth

Cutbacks

- Error Handling differs
- Probably not all operators implemented
- Block operations not completly implemnted

Golforth

Usage of Idiomatic Forth

- Stack paradigma mapped to typed language
- Wordlists for variable tracking
- Macros & anonym functions for language implementation
- Macros for operator implementation