5– a new programming language for I64 VMS (and more)

Dr. Bernd Ulmann

17-SEP-2010

Hochschule fuer Oekonomie und Management, Frankfurt

Introduction

Do you miss VAX APL?

No: You obviously missed something!

Yes: Maybe **5** will make you happy again!

5 at a glance

What is **5**?

- **5** is a dynamic programming language.
- **5** incorporates the main features of APL and Forth.
- The **5**-interpreter is written in Perl and easily portable.
- **5** runs out of the box on OpenVMS.
- **5** may be used to replace the much missed VAX APL interpreter.
- **5** is free software.
- **5**'s home is located at `lang5.sourceforge.net`.

5 at a glance

What is **5**, now?

- It is a stack oriented language just like Forth.
- The stack can hold scalars as well as nested data structures.
- The language can be extended by itself making use of so called *User Defined Words*.
- User defined words can act as unary or binary operators and are thus equivalent to builtin operators.
- Unary and binary operators are automatically applied in an element wise fashion on the elements of nested data structures.

How to install **5** on your OpenVMS system?

- Make sure you have a Perl interpreter running on your system.
- Download the **5** distribution kit from `https://sourceforge.net/projects/lang5/files/`. This kit contains the interpreter, a lot of examples and the complete documentation in PDF format.
- Unpack the distribution kit at a location suitable for your environment as the following example shows:

```
$ SET DEF DISK$SOFTWARE:[000000]
$ UNZIP DISK$SCRATCH:[SYSTEM]5.ZIP
```

- Define a foreign command for invoking **5** by adding a line like this to your SYS$MANAGER:SYLOGIN.COM:

```
$ FIVE :== PERL DISK$SOFTWARE:[5]5
```

- Now users can invoke the **5**-interpreter like that:

```
ULMANN:FAFNER$ five
----> loading mathlib.5
----> loading stdlib.5
5> 100 iota 1 + '+ reduce .
5050
5>
```

First steps

The following slides show some simple **5** programs to give an impression of the language. These examples are very basic – the power of the language is much greater than that shown here.

Example 1 – calculating $\sum_{i=1}^{100} i$ and 100!:

gauss_factorial.5

```
1  # Define two new words "gauss" and "factorial":
2  : gauss iota 1 + '+ reduce ;
3  : factorial iota 1 + '* reduce ;
4
5  # Use these new user defined words:
6  100 dup gauss . factorial .
```

gauss_factorial.5

Example 2 – the ubiquituous Fibonacci series:

fibr_apply.5

```
1  : fib{u}
2    dup 2 < if drop 1 break then
3    dup 1 - fib swap 2 - fib +
4  ;
5
6  10 iota fib .
```

fibr_apply.5

```
alberich$ 5 fibr_apply.5
----> loading mathlib.5
----> loading stdlib.5
loading fibr_apply.5
[    1     1     2     3     5     8    13    21
    34    55  ]
```

Sum of cubes

Recently I found the following Fortran-example program[1] which prints all numbers between 1 and 999 which are equal to the sum of the cubes of their digits:

sum_of_cubes.for

```
1  program sum_of_cubes
2  implicit none
3  integer :: H, T, U
4  do H = 1, 9
5    do T = 0, 9
6      do U = 0, 9
7        if (100*H + 10*T + U == H**3 + T**3 + U**3) &
8          print "(3I1)", H, T, U
9      end do
10   end do
11 end do
12 end program sum_of_cubes
```

sum_of_cubes.for

[1] Cf. [?][p. 41]

Sum of cubes

The **5**-solution is a bit shorter ("Look Mom, no Loops!"):

Example 3 – sum of cubes:

sum_of_cubes.5

```
1 : cube_sum{u} "" split 3 ** '+ reduce ;
2 999 iota 1 + dup dup cube_sum == select .
```

sum_of_cubes.5

```
alberich$ 5 sum_of_cubes.5
----> loading mathlib.5
----> loading stdlib.5
loading sum_of_cubes.5
[    1   153   370   371   407  ]
```

Example 4 – generate a list of primes (between 2 and 100 eg.):

primes.5

```
1  : prime_list
2    1 - iota 2 + dup dup dup
3    '* outer
4    swap in not
5    select
6  ;
7
8  100 prime_list .
```

primes.5

```
alberich$ 5 prime.5
----> loading mathlib.5
----> loading stdlib.5
loading prime.5
[    2     3     5     7    11    13    17    19    23    29    31    37
    41    43    47    53    59    61    67    71    73    79    83    89
    97  ]
```

Example 5 – plot on an ASCII-Terminal:

sine_curve.5

```
1  : print_dot{u} " " 1 compress swap reshape "*\n" append "" join . ;
2  21 iota 10 / 3.14159265 * sin 20 * 25 + int
```

sine_curve.5

```
alberich$ 5 sine_curve.5
----> loading mathlib.5
----> loading stdlib.5
loading sine_curve.5
                          *
                               *
                                    *
                                         *
                                            *
                                             *
                                            *
                                         *
                                    *
                               *
                          *
                     *
                *
           *
      *
      *
      *
           *
                *
                     *
                          *
```

What's it good for?

What's it good for?

What can one do with **5**?

- Use it as a very advanced command line calculator.
- Perfom ad-hoc data analyses.
- Experiment with complex functions and explore their behaviour.
- Taylor the language to suit your tasks by defining new user defined words which can be placed into libraries and will be loaded automatically.

The author can be reached at `ulmann@vaxman.de`.

Thank you for your interest.