38th EuroForth Conference

September 16-18, 2022

Preface

EuroForth is an annual conference on the Forth programming language, stack machines, and related topics, and has been held since 1985. The 38th EuroForth finds us again mostly at home, and the conference is being held on the Internet. The two previous EuroForths were also held online. Information on earlier conferences can be found at the EuroForth home page (`http://www.euroforth.org/`).

Since 1994, EuroForth has a refereed and a non-refereed track. This year there has been one submissions to the refereed track, which was accepted (100% acceptance rate). For more meaningful statistics, I include the numbers since 2006: 30 submissions, 22 accepts, 73% acceptance rate. This year the only submission was authored by the program chair; Ulrich Hoffmann served as secondary chair and organized the reviewing and the final decision for that paper. The reviews of all papers are anonymous to the author: The paper was reviewed and the final decision taken without involving the author. I thank the program committee for their paper reviews.

In addition to the papers and presentation handouts available before the conference, these online proceedings also contain presentation handouts that were provided at or after the conference. I thank the authors for their papers and slide handouts.

You can find these proceedings, as well as the individual papers and slides, and (when they become available) links to the presentation videos on `http://www.euroforth.org/ef22/papers/`.

Workshops and social events complement the program. This year's EuroForth has been organized by Gerald Wodni.

Anton Ertl

Program committee

M. Anton Ertl, TU Wien (chair)
Ulrich Hoffmann, FH Wedel University of Applied Sciences
Matthias Koch, Institute of Quantum Optics, Leibniz University Hannover
Jaanus Pöial, Tallinn University of Technology
Bradford Rodriguez, T-Recursive Technology
Bill Stoddart
Reuben Thomas

Contents

Refereed Papers

M. Anton Ertl: Memory Safety Without Tagging nor Static Type Checking . . . 5

Presentation Slides

Nick J. Nelson: Better Values: Improvements to the implementation of the extended Forth Value concept . . . 16

Nick J. Nelson: Forth Query Language (FQL): Refining the result set analysis using dynamically generated local functions . . . 20

Ulrich Hoffmann: Fuzzing Forth . . . 24

Krishna Myneni: Progress Towards Porting EISPACK to Forth . . . 33

Glyn Faulkner: Tales from the Left-Hand Path: Dark Confessions of a Forth Hobbyist . . . 37

Klaus Schleisiek: German Academia and Forth . . . 42

Bernd Paysan: Gforth 1.0 . . . 43

Klaus Schleisiek: uCore progress . . . 44

Ulrich Hoffmann: Standard Report . . . 45

Nate Morse: Joy to the Web: A Zero Install version of Joy (not a production) Language called Pounce . . . 46

M. Anton Ertl: Are locals inevitably slow? . . . 48

Ulrich Hoffmann: Enums in Forth . . . 50

Nick Nelson: Resource embedding in Forth . . . 53

Howerd Oakford: Encoding ASCII into cf2022 colorForth tokens . . . 55

Memory Safety Without Tagging nor Static Type Checking

M. Anton Ertl[*]
TU Wien

Abstract

A significant proportion of vulnerabilities are due to memory accesses (typically in C code) that memory-safe languages like Java prevent. This paper discusses a new approach to modifying Forth for memory-safety: Eliminate addresses from the data stack; instead, put object references on a separate object stack and use `value`-flavoured words. This approach avoids the complexity of static type checking (used in, e.g., Java and Factor), and also avoids the performance overhead of dynamic type checking for non-memory operations. This paper discusses the consequences of this approach on the language, and on performance.

1 Introduction

Accessing arrays with an index outside the array bounds is usually a bug. Other memory access errors include reading an uninitialized location, accessing a field of the wrong structure, and accessing memory that has been freed. Memory access errors are common as sources of program crashes, but they are also a common source of vulnerabilities: Gaynor reports [Gay20] that at least 65% of vulnerabilities in various software environments are due to memory safety issues.

From pretty early on there were programming languages where such bugs could not happen or would be caught and reported, e.g., Lisp and Algol 60. Such programming languages are called *memory-safe languages*.

Forth, on the other hand, allows the programmers to shoot themselves in the foot, and relies on careful programmers to avoid such bugs; while this has some advantages, there are occasions when some programmers prefer memory safety. The Forth world has some answers in this area, e.g., Factor and Oforth, but they rely on static type checking and/or tagging all data for dynamic type checking.

In the present work, we introduce the approach of avoiding some of that type checking by eliminating addresses from the data stack, and instead keeping object[1] references on an object stack. The benefits of this approach are that it avoids both the complexity of static type checking as well as the slowdown of tagging and tag-checking.

In Section 2 we look at the problem and how others have solved it. Section 3 describes our own approach; in particular, the main contribution of this work is to keep object references separate from other data through separate stacks (Section 3.1) and value-flavoured words (Section 3.6). The rest of Section 3 describes various other changes necessary for memory safety, but most of that is relatively straightforward. Section 4 gives an idea about the difference between Forth and Safe Forth by dividing the Core wordset of standard Forth into words that are unchanged in Safe Forth, and words that need various changes. Section 5 discusses topics beyond 1:1-correspondence between Forth and Safe Forth. We present implementation approaches for some common operations in Safe Forth and their theoretical effect on performance in Section 6. Safe Forth is currently a paper design (Section 7). Finally, Section 8 discusses some related work.

1.1 Safe Forth or a memory-safe Forth?

The intention of this paper is to explore the idea of memory-safety without tagging and without static type checking in general. One can build a number of different languages on that foundation, which makes the exposition challenging: Describing only one of these languages loses generality, but makes the description easier to understand. Therefore we take this approach, and we call that language *Safe Forth*, but we occasionally describe alternative variants.

Safe Forth is object-oriented; this plays a minor role in much of this paper, but occasionally shines through. A more detailed description of the object model is given in Section 3.8, where the topic is first discussed in more depth.

[*]`anton@mips.complang.tuwien.ac.at`

[1]Here *object* refers to a piece of memory, similar to a contiguous region in standard Forth, or an object in the C standard. The division of memory into pieces instead of providing a flat address space is universal in memory-safe languages. However, elsewhere in this paper *object* typically is used in the object-oriented sense.

2 Existing approaches

Most languages in wide use today are memory-safe, with the exceptions being C, C++, assembly language and Forth; languages like Rust and Ada have unsafe escape hatches, and many others have interfaces to C that can also serve as escape hatches (but into C, not an unsafe dialect of the base language).

As an example, Java has primitive types (such as `int` or `double`) and reference types (objects including arrays). Java has static type rules and the Java compiler applies them and knows which type an expression has. On the level of objects, the type checker often does not know the exact class/type of an object, just that it is a subclass of a certain class; the exact class information is present at run-time in a field at the start of each object and is used for method dispatch and for subtype checking.

This scheme is roughly followed by all object-oriented languages, but in many language implementations the primitive types and object references are not distinguished by static type checking, but by tagging: A bit or a few bits of a machine word (a cell in Forth terminology) are reserved for indicating the type of the machine word: each primitive type represented as a machine word gets a different tag, and object references also get their tag (usually one tag for all object references, with more information available at the start of the object).[2] There are also cases like the Ocaml interpreter that uses tagging in a statically type-checked language to simplify the garbage collector.

In the world of Forth-like languages, Factor uses static type checking, while Oforth uses tagging.

These implementations ensure memory-safety as follows:

Out-of-bounds access: Array accesses are bounds-checked. Sophisticated compilers can eliminate a significant proportion of these bounds checks [BGS00].

Uninitialized location: All locations are initialized. Java and Oforth zero all locations: this initializes integers to 0, FP variables to 0.0, and reference types to `null`.

Access to the wrong field: Java only allows accesses to fields that belong to the (statically known) class of the object. Java allows casting to a class C in order to establish that static knowledge, but then Java tests during the cast (at run-time) whether the object is an instance of C. Likewise, other languages either have to establish static knowledge or check on every field access that the object has that field. Like for bounds checks, there are ways to reduce the number of necessary checks by increasing compiler sophistication.

Use after free: The common approach to avoiding use-after-free bugs is to let the language perform automatic storage reclamation, either through garbage collection or through reference counting (Python). Rust employs a sophisticated static type system that ensures that no reference to an object remains when the object is freed.

[2]There is also a technique called NaN-boxing which for the purposes of this paper is a variant of tagging and is not discussed further here.

3 Safe Forth

This section discusses how Safe Forth differs from Forth, and how its approach achieves memory-safety.

3.1 Basic approach

The basic idea in the present approach is to separate object references from other data types by putting object references on a separate objects stack, and no addresses on the data stack. Words that work with object references take them from the objects stack, and words that work with non-reference data take them from the data and FP stacks.

This makes it unnecessary to use tags or sophisticated type checkers, and yet (with the right setup) it is impossible to perform address arithmetic and similar things that are incompatible with memory safety.

In particular, consider the twenty dynamically most frequently executed primitives in the statistics at http://www.complang.tuwien.ac.at/forth/peep/sorted:

1	13.5%	`;s`
2	13.2%	`col:`
3	**9.0%**	**`@`**
4	*5.1%*	*`?branch`*
5	4.6%	`lit`
6	3.4%	`var:`
7	3.4%	`dup`
8	3.2%	`user:`
9	3.0%	`swap`
10	*2.8%*	*`+`*
11	2.5%	`con:`
12	2.0%	`>r`
13	1.9%	`r>`
14	*1.8%*	*`0=`*
15	*1.3%*	*`and`*
16	**1.3%**	**`c@`**
17	**1.2%**	**`!`**
18	1.2%	`over`
19	*1.1%*	*`cells`*
20	1.1%	`rot`

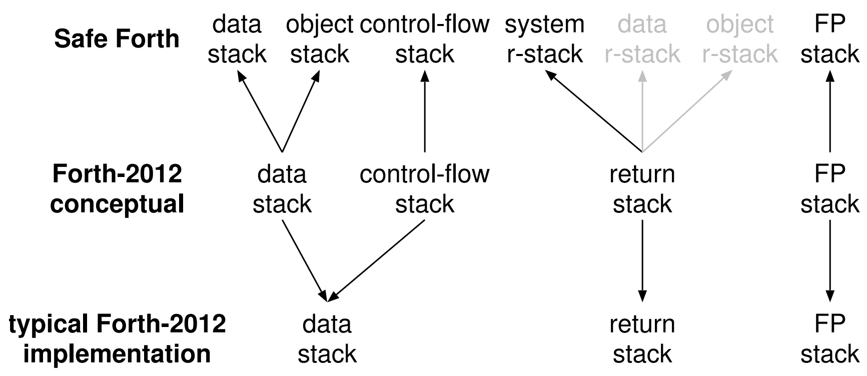


Figure 1: Stacks in Forth and Safe Forth

The black numbers are for words that do not need to handle tags in a tagged implementation, either because they don't handle data, they handle arbitrary data, or they push pre-tagged data (e.g., `lit`). The *slanted blue* lines are for words that would need to deal with tags in a tagged system, but do not need that with our approach (or with static type checking). The **bold red** lines are for words that deal with addresses; in a tagged system like Oforth they are replaced with words that have to check both the tags and the class descriptors of the object references, while with our approach (and with static type checking) you avoid the tag checking (but still have to check the class descriptor).

For the four problems mentioned above our approach is:

Out-of-bounds access: Array accesses are bounds-checked (and first the object is checked to be an array of the appropriate type).

Uninitialized location Values, arrays, and structures are zeroed on creation.

Access to the wrong field A field access checks whether the object actually has that field. It is possible to optimize this check away in some cases with relatively little complexity; e.g., in a method of class C we know that `this` is an instance of C.

Use after free Use garbage collection. Other approaches may be possible, but are unlikely to be simpler.

What this approach does not give us is type checking of the data on the data stack. If you want to add 1 to the letter `A`, you still can[3]. You also don't have to pay the compiler complexity or run-time cost of such type checking.

[3]While Safe Forth is probably not something Chuck Moore is interested in, at least in this respect it follows his preferences [RCM96]

3.2 Stack underflows and overflows

Stack overflows and underflows undermine memory safety if they are not caught. Fortunately they can be caught at no run-time cost on systems with a paged MMU, by putting unaccessible guard pages around each stack. This approach has been used by Gforth since almost its inception, and is a good approach for Safe Forth.

Gforth-fast keeps stack items in registers; to avoid spurious underflow exceptions from traffic between these stack items and the stack in memory, several memory slots below the stack bottom are left accessible. In a Safe Forth using this technique, the registers and the extra memory slots of the object stack have to be initialized to safe values (e.g., `null`); then, while accessing stack items below the bottom is possible, this cannot subvert memory safety.

If MMU-based stack bounds checking is unavailable, the stack pointer can be bounds-checked (more expensively) with code every time it is changed.

3.3 Return stack

In addition to keeping object references on a separate stack, we also have to ensure that we don't open a memory-safety hole through the return stack. Forth systems typically keep return addresses on the return stack, as well as counted-loop parameters and cells moved by the program from the data stack with words like `>r`. In ordinary Forth, a program can execute arbitrary code with `>r exit`, which is not compatible with memory safety.

Our solution to this problem is to just exclude the return stack words like `>r` from Safe Forth.

Another solution is to split the return stack into three stacks: A system return stack for system-execution types (return addresses and loop control paramaters), a data return stack for stashing away data stack items, and an object return stack for stashing away object stack items. However, if you implement locals, the additional benefit of the latter two stacks does not seem to be worth the cost.

In either case, `exit` inside a counted loop poses a problem (unless we split return addresses and counted loop parameters into using different stacks). Standard Forth has `unloop` to remove counted-loop parameters, but memory-safety can be violated in typical implementations by using `unloop` in a non-standard way. However, because the stack for system-execution types now only contains these types, it is relatively straightforward to implement `exit` without needing `unloop`: Just count the number of counted loop nests, and let `exit` compile code for dropping the corresponding number of loop parameters before compiling the return.

Figure 1 shows the relation between the stacks in Forth and Safe Forth. Optional stacks are shown in light gray.

3.4 Control-flow stack

The control-flow stack contains information about incomplete control structures during compilation. On most systems it is implemented on the data stack (although the standard does not guarantee that).

We considered putting the control-flow stack items on the object stack in Safe Forth. However, in Safe Forth we have to ensure that `do`-like words are matched with `loop`-like words not only during compilation, but also at run-time (to avoid memory corruption by mixups of loop-control parameters and return addresses).

Eventually the simplest way to achieve this is to have a separate control-flow stack.

3.5 `Null`

`Null` is a value on the object stack without object; trying to access through a `null` reference throws an error. `Null` is implemented as address 0 (to make type-ignorant initialization possible). Programs can test whether a reference is `null`.

3.6 No `variables`

`Variables` push an address, so Safe Forth eliminates them. They are replaced with `values`. In addition to the classical values that communicate with the data stack, there are `ovalues` for holding object references; an ovalue `ov` pushes its content on the object stack, and `to ov` consumes an element from the object stack.

In all other areas variable-flavoured words for storing data are eliminated and replaced with value-flavoured words, with both data-stack and object-stack variants. In particular, field-access words are value-flavoured (Forth-2012 only provides variable-flavoured field words).

Locals are value-flavoured in Forth-2012, which is kept in Safe Forth. The object-stack variant is defined by using `o:` in front of the local. E.g.

```
{: a  o: b  c :}
```

defines two data-stack locals `a` and `c` and an object-stack local `b`.

All these value-flavoured words for storing data initialize the data to `0`, `0e` or `null` if there is no initialization value given (e.g., for locals after `|`).

3.7 Fields in structures

This section discusses the access of fields in structures in a non-object-oriented memory-safe Forth. We use the Forth-2012 structure syntax in this section, with modifications.

A (value-flavoured) field is defined with one of the defining words `ovalue: value: cvalue: fvalue:` etc.[4] Fields are defined in the context of a structure.

For a field `f`, performing `f` takes a reference from the object stack and checks if the reference points to a structure/object that has a field `f`. If so, it pushes the content of that field in the structure on the appropriate stack (data, object, or FP stack). If not, it throws an exception.

Performing `to f`[5] is very similar, except that it takes a value from the appropriate stack and stores it in the field.

Forth's field-defining words keep the offset on the data stack during the definition, but that would mean that the program could change offsets and thus undermine memory-safety, so in Safe Forth the offset is kept in a hidden variable.[6] This means that we have to use `begin-structure ... end-structure` rather than the alternative `0 ... constant`.

The structure name pushes an object representing the structure type on the object stack, and `new` allocates an object of that type and initializes it with the type, and all fields zeroed.

Figure 2 shows an example of a structure. We discuss the representation of the type in Section 6.

3.8 Objects and fields

Safe Forth uses an object model with single inheritance of fields and method implementations, where every method selector can be used with any class (duck typing). When invoking a method, the top of

[4]Many of these defining words are also defined for Forth in Gforth's `struct-val.fs`.

[5]The alternative syntax `->f` is already implemented in Gforth.

[6]Alternative: an opaque object referenced through the object stack.

```
begin-structure intlist
  ovalue: intlist-next
  value:  intlist-val
end-structure

intlist new ovalue x
5 x to intlist-val
```

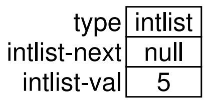


Figure 2: Source code for creating a structure and the resulting memory for the structure

the object stack O is removed, put into `this`, and used for method dispatch (calling the right method implementation for the combination of the class of O and the method selector).

Fields in objects are similar to fields in structures, with one difference: they refer to the object in `this` rather than the object stack, and do not consume that object. Because `this` is set on method entry, we know that the class D of the object is a subclass of the class C for which the method implementation was defined; if `f` is defined in a superclass B of C, no checking is necessary, and the check can be eliminated when compiling `f` inside that method.

Note that, e.g., `postpone f` can put `f` in a context where the class of `this` is not guaranteed to be a subclass of B, so it's better to make this optimization dependent on the compilation context. The alternative is to ensure through language restrictions that `f` can only be used in the right context, but one might overlook some corner case, and the restrictions may be too limiting.

3.9 Arrays

Arrays are accessed through a reference on the object stack. They are either object arrays that contain only object references and `null`, or they are data arrays that contain only data, no object references. Safe Forth has typed data arrays, i.e., they contain only cells, only characters, or only floats.

Safe Forth uses polymorphic access words:

```
[]   ( u array -- v )
->[] ( v u array -- )
```

The type of `v` depends on the type of the array, e.g., for an FP array `v` is a FP value on the FP stack, for an object array `v` is an object reference on the object stack.[7] If the index `u` is outside the array bounds, an exception is thrown.

[7]A disadvantage of this approach is that `[]` and `->[]` have a stack effect that depends on the passed array, which makes the code harder to analyse. Alternatively, we could have stack-effect-specific array access words such as `o[] ->o[] [] ->[] f[] ->f[]`.

```
50 farray oconstant a
3e 1 a ->[]
a 1 3 slice oconstant b
5e 1 b ->[]
```

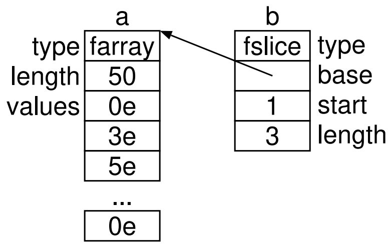


Figure 3: Source code for creating an array and an array slice, and the resulting memory

An alternative is to have object arrays and, for non-objects, untyped arrays of bytes with typed access words that use byte offsets rather than element indices. This approach would be closer to Forth and still memory-safe, but users of an object-oriented memory-safe Forth probably prefer the approach outlined for Safe Forth.

Figure 3 shows an example of an array. We discuss the representation of the type in Section 6.

The word `farray` creates a new FP array with the length (number of elements) given on the data stack; the elements are initialized to 0e.

3.10 Array slices

Forth supports representing parts of arrays by giving the start address and number of elements (or number of address units), the same representation as the full array. In Forth address arithmetic is used for that, but we cannot use that in Safe Forth, so we provide array slices instead. In the simplest case an array slice starts at some element of the array it is based on (with index 0 in the array slice), and has a length at most as long as the remaining elements of the base array.

Figure 3 shows an example of an array slice. This assumes (object-oriented) dynamic dispatch for `->[]` to work. Once we have polymorphism for array access words, we can have more fancyful kinds of array slices beyond what Forth can represent with address and length, e.g., with strided access.

3.11 Strings and String Buffers

Standard Forth uses *c-addr u* to represent both strings (e.g., in `type`) and string buffers (e.g., in `read-file`). In some cases (e.g., `move`), string buffers are represented by the address alone, and the word cannot prevent buffer overflows.

```
16 stringbuf oconstant s
s" abc" s move
s type
```

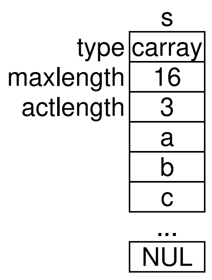


Figure 4: Source code for creating a string buffer, and for storing a literal string there, and the resulting memory

The use of two cells for this purpose leads to deep stacks and related complications, and so a significant number of programmers advocates counted strings or other alternative single-cell representations. So in Safe Forth we use a single reference on the object stack for representing a string or a string buffer.

A string buffer has a maximum length (used for words that write to the buffer, such as `read-file`), and an actual length (used for words that read from the string, such as `type`); see Fig. 4. Trying to store a too-long string into a string buffer results in an exception.

For read-only strings the maximum length is unnecessary. Such a specialized representation can easily be implemented in an object-oriented Safe Forth in addition to string buffers. Trying to write to a read-only string results in an exception.

3.12 Execution tokens

In Forth, `execute`, `compile,` and `defer!` take an xt from the data stack. In Safe Forth, we need to put execution tokens on the object stack; however, the execution tokens in Safe Forth will likely have a different representation than those in Forth, because it needs a type field like other objects referenced from the object header. One way to achieve this on top of an unchanged Forth system is to create objects that contain a type field and the Forth-level execution token. In Gforth a more efficient approach (one memory access less) would be to add a type field to the word header [PE19], but that requires deeper changes.

3.13 `does>` and `>body`

`Create` is eliminated. Instead, `does>` is combined with `oconstant`[8] and the code behind `does>` starts with the value of the oconstant pushed onto the object stack. Correspondingly, `>body` produces the value of the oconstant from its xt. E.g., the following is a possible implementation of `constant` in a non-object-oriented memory-safe Forth (Section 3.7):

```
begin-structure const-struct
  value: val
end-structure

: constant ( n "name" -- )
  const-struct new odup to val oconstant
does> ( -- n )
  ( const-struct ) val ;
```

3.14 Garbage collection

There are conservative garbage collectors that do not move the data around, and do not need to know if a cell contains a memory reference or some other data. The existing Forth garbage collector[9] is conservative.

Alternatively, precise garbage collection needs to know which cells contain memory references and which don't; the benefit is that it can move the data around, which eliminates fragmentation and makes allocation faster.

In the absence of tagging, it is difficult to keep track of all memory references in all situations: It's relatively straightforward to know it for object fields: objects are headed with a class address, and the class can contain the necessary information. But for locals, things are more difficult; either we have separate locals stacks for data and for object references, or we need some way to know which cells on the locals stack or return stack are references and which are data. These ways either require some runtime overhead or significant compiler complications.

To avoid such complications, we stick to conservative garbage collection.

4 Words

In order to see how similar and how different Safe Forth is from Forth, in this section we look at the Forth-2012 core words, and determine which are unchanged, which are changed, and how substantial the needed changes are.

[8] `Oconstant` is the object-stack equivalent of `constant`. An alternative view is that it is like `ovalue` except that you cannot use `to` on its children. Note that you can put mutable objects into oconstants

[9] http://www.complang.tuwien.ac.at/forth/garbage-collection.zip

In some cases one might choose the break in compatibility to perform other changes as well, e.g., to change the input stream handling, but in this paper we do not go into that and only work out changes that are related to the requirements of memory-safety.

4.1 Unchanged

These words typically work just on the data stack, but in some cases some stack items are taken off or pushed on the object stack, FP stack, or system-return-stack as discussed above: execution tokens are on the object stack, system compilation types (e.g., control-flow stack items like orig) are on the control-flow stack, system execution types (e.g., nest-sys, i.e., return addresses) are on the return stack, and floating-point values are on the FP stack.

```
 # #S ' ( * */ */MOD + +LOOP - .  ." /
/MOD 0< 0= 1+ 1- 2* 2/ 2DROP 2DUP 2OVER
2SWAP : ; < <# = > >IN ?DUP ABORT ABORT"
ABS AND BEGIN BL CHAR CELL+ CELLS CHAR+
CHARS CONSTANT CR DECIMAL DEPTH DO DROP
DUP ELSE EMIT EXECUTE FM/MOD HOLD I IF
IMMEDIATE INVERT J KEY LEAVE LITERAL
LOOP LSHIFT M* MAX MIN MOD NEGATE OR OVER
POSTPONE QUIT RECURSE REPEAT ROT RSHIFT
S>D SIGN SM/REM SPACE SPACES SWAP THEN U.
U< UM* UM/MOD UNTIL WHILE XOR [ ['] [CHAR]
]
```

`CELL+ CELLS CHAR+ CHARS` are listed above, because they can be used to compute sizes or offsets, but they are not very useful in most Safe Forth variants (except those that use sizes and offsets for arrays instead of numbers of elements or indexes).

4.2 Adapted stack effects

```
#> ( xd -- string )
>BODY ( xt -- object )
>NUMBER ( ud1 string -- ud2 stringslice )
ACCEPT ( stringbuf -- )
COUNT ( string -- c stringslice )
ENVIRONMENT? ( string -- false | ... true )
EVALUATE ( ... string -- ... )
FILL ( stringbuf c -- )
FIND ( string -- xt +-1 | string 0 )
MOVE ( string stringbuf -- )
S" ( run-time: -- string )
SOURCE ( -- string )
TYPE ( string -- )
WORD ( c -- string )
```

Among these, `count` is useful only in its meaning as alias for `c@+`.

4.3 Slight changes

`DOES>` now works on oconstants.

`EXIT` now works in counted loops without preceding `unloop`.

4.4 Substantial changes

`@` is deleted. Values and value-flavoured words push their value, and `[]` loads a value from an array.

`!` is deleted. `to` is used for storing into values and fields, `is` for storing into deferred words, and `->[]` for storing into arrays.

`+!` is deleted. The system may have `+to`.

`Allot` is deleted. Words like `new` or `array` are used instead.

`State` is replaced with the value `state@`.

`Base` is replaced with the value `base@`.

4.5 Deleted words

```
 !  +!  , 2!  2@ >R @ ALIGN ALIGNED ALLOT
BASE C! C, C@ CREATE HERE R> R@ STATE
UNLOOP VARIABLE
```

4.6 New words

```
base@ ( -- u )
null= ( o -- )
o= ( o1 i2 -- )
odup? ( o -- null | o o )
oconstant ( o "name" -- )
odrop ( o -- )
odup ( o -- o o )
oliteral ( o -- ) ( run-time: -- o )
oover ( o1 o2 -- o1 o2 o1 )
orot ( o1 o2 o3 -- o2 o3 o1 )
state@ ( -- f )
oswap ( o1 o2 -- o2 o1 )
[] ( u array -- v )
->[] ( v u array -- )
```

This set of words only allows individual values and array data. You typically also want to add some words for defining classes, fields and methods (not listed here).

4.7 Statistics

Relative to a base of 133 Core words in Forth-2012, 96 (72%) are unchanged, 14 (11%) have adapted stack effects and 2 (2%) have other small changes. 21 (16%) are deleted, and 14 (11%) are new.

5 Advanced topics

5.1 Looping over arrays

Arrays are often accessed in loops, e.g., iterating from the first to the last element. The bounds checks (and type checks) for these accesses can often be optimized away, and many memory-safe languages employ compiler optimization based on some form of data-flow analysis [BGS00] to achieve that.

Of course we want to use a simpler way in Forth. One way is to have array-walking words that walk the whole array (or array slice) and don't need to perform bounds checking. This could look as follows:

```
: type ( string -- )
  arraydo emit arrayloop ;
```

`Arraydo` would take an array and push one element of the array (from first to last) in every iteration; the loop would be closed with `arrayloop`.

In addition one might want variants that write an element in every iteration, or that push an element from one array and store a result into another array, etc. You may also want to work with a strided array slice, possibly with negative stride (for walking the array backwards). One problem with flexible `arraydo`s that can take arrays of various types, and array slices with all kinds of strides is that the way the array is handled is only known at run-time, so one typically has to call a run-time-dispatched handler. One could avoid that by having a variant of `arraydo` for every kind of array (slice) it can be applied to, but that leads to an explosion of words and would result in less flexible words that use these words.

An alternative to a `do`...`loop`-like control structure is to define a word like Postscript's `forall` or Oforth's `apply` that takes an array and an xt, and for each element of the array, pushes the element and calls the xt. One disadvantage of this approach is that either the code in the xt cannot access locals of the definition containing the `forall`, it requires explicit passing of the locals [EP18], or a complex implementation; by contrast, `arraydo`...`arrayloop` supports access to locals in the loop body.

5.2 Escape hatch

Memory-safe languages often have an escape hatch into code that is not guaranteed to be safe: E.g., Modula-2 has the module `SYSTEM` that allows access to low-level facilities, Java has the Java native interface (JNI) that allows calling C functions, and Rust has unsafe blocks.

The purpose of such an escape hatch is twofold:

- It allows doing things that are impossible in the memory-safe language, such as low-level access to I/O devices.
- It allows doing things more efficiently that are inefficient in the memory-safe language. E.g., `arraydo`...`arrayloop` could be implemented in Safe Forth by accessing each element individually and incurring a bounds check for each element, but it is more efficient to write these words in Forth without bounds checks (all accesses are within the array).

The typical approach is not to use the escape hatch at each place in application code that needs to deal with, e.g., I/O devices, but to define a memory-safe interface to these I/O devices, and use the escape hatch to implement this interface. This keeps the unsafe code to a minumum.

Two people reading a draft version of this paper suggested to include a way to specify absolute addresses for memory-mapped I/O in Safe Forth. We know of no way to achieve this without opening a hole that may subvert the memory-safety guarantees of Safe Forth. But using the escape hatch to achieve this is appropriate and seems simple enough: E.g., one could define words for the individual registers of the I/O device, or alternatively define the I/O device as a slice in an array of bytes (specifying the address through the escape hatch), and use offsets in that array to access various registers. In either case, the programmer of code beyond the escape hatch would guarantee the memory-safety of the words added to Safe Forth, or of the data structures modified through the escape hatch.

An advantage of having an escape hatch is that Safe Forth does not need to include words for every contingency; instead, you can define them as needed.

For Safe Forth implemented on top of Forth, the obvious escape hatch language is Forth, and the escape hatch is relatively simple to implement: Safe Forth provides its words in a wordlist different from the wordlists providing unsafe words, and in Safe Forth only this wordlist and wordlists defined in memory-safe code are on the wordlist. The escape-hatch word `forth` makes the Forth wordlist available again.

There may also be cases where you want to weld the escape hatch shut, in particular when you want to process untrusted code (e.g., coming from the Internet). It is straightforward to have a mechanism for disabling the escape hatch.

5.3 Multi-threading

Some of the challenges of multi-threaded multi-tasking are: Single-threaded stop-the-world garbage collection is relatively simple and is implemented in an existing garbage collector for Forth; extending it for a single-threaded cooperative multi-tasker is relatively easy, but has the unpleasant effect of stopping all tasks while it collects. Things become really complex when multi-threading is involved, because other threads may be in states that are not safe for garbage collection. There are solutions to these problems, but they are quite complex.

An alternative is to set the system up such that only per-thread/task garbage collection is necessary. One way to do that is that every thread/task has its own memory and collects its own memory. The tasks communicate through channels, mailboxes, or the like, but cannot pass object references across these communication channels. Instead, if you want to pass a data structure to another task, it has to be marshalled (serialized) on the sender and unmarshalled on the receiver.

The disadvantage of this approach is that the marshalling and unmarshalling constitutes an overhead; the advantage is that programs organized in this way can put the tasks into different processes, and actually even different computers.

Alternative approaches are to avoid garbage collection, e.g., with a reference-counting scheme, or to implement full-blown multi-threaded garbage collection.

6 Implementation efficiency

This section does not describe the implementation in detail, but addresses performance concerns you may have. The implementation approaches suggested in this section are designed to go with an object-oriented system where every selector can be used with any class (duck typing), with single inheritance (especially of fields), where all methods of a class are defined before the first object of the class is created.

In this section, we describe the implementation of some Safe Forth features as Forth code, but it could just as well be some other lower-level language, e.g., assembly language.

The performance claims are not supported empirically in this paper, possibly in a future paper.

6.1 Type equality

Field accesses to structures that do not support extensions only have to check the structure type for equality. So an access to field `intlist-val` in Fig. 2 could be implemented in Forth as follows:

```
: intlist-val ( o -- n )
  o> dup @ intlist = if
    2 cells + @ exit then
  type-error throw ;
```

On processors with out-of-order cores (such as the performance cores on desktop, server, and current smartphone CPUs) the `if` will usually be predicted correctly, which means that the code needed to compute the condition (in **bold red**) is not on the critical path; it costs resources (of which most such CPUs have plenty), but not latency. Only the *slanted blue* part is on the critical path for computing the result. So, on this class of CPUs this checked version will be about as fast as the unchecked version, at least in latency-limited code (the usual case).

```
selector sum
selector foreach

object class
  ovalue: next
  m: foreach ... ;m
end-class list

list class
  value: val
  m: sum ... ;m
end-class intlist
```

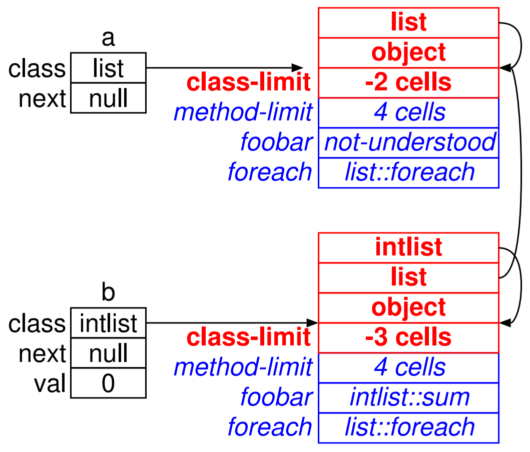


Figure 5: Simple example of some classes and objects. The **bold red** parts are used for subtype testing, the *slanted blue* parts for method dispatch.

6.2 Subtype test

If the Safe Forth supports structure extension, or equivalently inheritance of fields in classes, we need to check if the type of the structure/object is a subtype of the type the field was defined for.

We use Cohen's approach to subtype testing, [Coh91], see Fig. 5: Each class has a subtype table (shown in **bold red**) containing its own address, and the addresses of all its ancestor classes. The primal ancestor (`object`) has offset -1 cells, the next generation (`list` in the example) has offset -2 cells, etc. For each class the offset at which it is to be found is known. E.g., `list` and all of its descendents will find the address of `list` at offset -2 cells in their subtype table; nondescendents will either have a higher (closer-to-zero) `class-limit`, or will have a different class at offset -2 cells. So the code for an access to field `next` with a subtype check looks as follows:

```
: next ( o1 -- o2 )
  o> dup @ dup @ -2 cells <= if
    -2 cells + @ list = if
      cell+ @ >o exit then then
  type-error throw ;
```

Concerning performance, the latency in the usual correctly-predicted case is again the same as for the unchecked case, but the check needs more resources than a type equality check.

Note that when accessing a field of `this` (the object used for method dispatch), an ancestor class of `this` is known, and if the field belongs to an ancestor of that class (the usual case), no check at run-time is needed.

6.3 Method dispatch

We use a per-class method table with bounds checking (a bounds-checked version of unhashed general selectors [Ert12, Section 3.1]). The table is shown in the blue part of Fig. 5: `method-limit` contains the offset from the start of the class descriptor (where the class addresses point to) to just beyond the last method implemented for the class. The code for a method dispatch looks as follows:

```
: sum ( o -- n )
  o> dup @ dup cell+ @ 3 cells u>= if
    2 cells + @ execute exit then
  not-understood ;
```

Again, the bounds check (in red) does not cost latency on an out-of-order CPU, and the whole dispatch is usually as fast as the unchecked version.

6.4 Array access

An array access as a colon definition has to check the type, and the bounds. If the array access is implemented as a method, the method dispatch has to be performed, but there is no need to check the type. The bounds check has to be implemented in any case. Here's how `f[]` implemented as method for `farray` (Fig. 3) might look:

```
m: f[] ( n farray -- r )
  ( farray is in THIS )
  dup length u< if
    floats values + f@ exit then
  bound-error throw ;m
```

The bounds check is in red, and if the `if` is correctly predicted (the usual case), it does not contribute to the latency. A correctly predicted method dispatch also does not contribute to the latency.

7 Status and further work

For now Safe Forth is a paper design. Implementing it and evaluating this implementation is on my agenda, but currently not at the top, and it depends on the interest of the community if it ever reaches the top.

One could implement it as a layer on top of Forth relatively easily, but that would mean that the object-stack access is slow. For decent performance, the object stack needs to be implemented with its stack pointer in a register and maybe an OTOS (top-of-object-stack) register, and a `this` register, which requires changing an existing system at a pretty basic level and is somewhat more involved.

8 Related work

For the main contribution of the present work, the use of a separate stack and value-flavoured words to reduce the need for static or dynamic type checking, there is surprisingly little related work. A number of Forth systems have a separate FP stack (standardized in Forth-2012), but the reason for that is not type-safety, and the separation ends as soon as the stack contents are stored to memory.

Similarly, a string stack has often been proposed for dealing with strings [MM81], but again, type safety has not been a primary goal. A vector stack has been used for dealing with vectors [Ert17]; there the main point is to provide vectors as an opaque data type. Still, the object stack can be seen as a generalization of the string and vector stacks.

There are several memory-safe languages based on Forth, e.g., Oforth[10] and Factor [PEG10]. However, Oforth uses type tagging and Factor uses static type checking in combination with run-time type checking, while Safe Forth uses the division between object references and integer/FP data to get rid of a large part of the type checking.

Apart from that, many design decisions in Oforth[11] are similar to those in Safe Forth: Oforth initializes all locations to zero. You can only read from and write to fields (with `@field !field`), not take their address; field accesses are allowed only to fields of `self`'s class, and no class check is necessary. Oforth keeps only system-defined stuff on the return stack, and it uses a separated control-flow stack. Oforth has no variables, only (task-local) values. `Execute` is a method selector for objects. `Does>` is eliminated, because it can be replaced with object-oriented dispatch. Unlike Safe Forth, Oforth implements control flow similar to Factor or Postscript: by passing closures (quotations with lexical scoping)

[10] http://www.oforth.com/
[11] Personal communication with Franck Bensusan

to control-flow words that `execute` the control-flow word multiple times).

There is a large body of work on subtype testing and method dispatch; Ducournau [Duc11] gives an excellent, although somewhat abstract overview. We choose Cohen's subtype testing approach [Coh91] because we comply with its single-inheritance limitation and because it is simple to implement incrementally. For method dispatch in objects2 [Ert12] we proposed using a number of these techniques (to allow different space/time tradeoffs), here we select one of those and enhance it with bounds checking similar to that used in Oforth [Ben18]; however, Oforth has a table of classes per selector, while the unhashed general selectors in objects2 and in the present work have a table of selectors per class.

9 Conclusion

Memory-safe languages eliminate a significant class of bugs and vulnerabilities. Forth can be turned into a memory-safe language by eliminating all operations involving addresses, e.g., `@` and `!`. In order to be still useful as a general-purpose language, we replace addresses with object references, but they have to be distinguished from other data. Other languages and virtual machines use tagging and/or static type checking to distinguish them; we instead put object references on an object stack, and have `ovalue` and other defining words that keep object references separated from other data.

Starting from this premise, and otherwise keeping relatively close to standard Forth results in surprisingly few changes to the core vocabulary.

Acknowledgments

Franck Bensusan and the reviewers provided helpful comments that helped improve the paper.

References

[Ben18] M. Franck Bensusan. Method dispatch in Oforth. In *34th EuroForth Conference*, pages 31–36, 2018. 8

[BGS00] Rastislav Bodik, Rajiv Gupta, and Vivek Sarkar. ABCD: eliminating array bounds checks on demand. In *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation*, pages 321–333, 2000. 2, 5.1

[Coh91] Norman H. Cohen. Type-extension type tests can be performed in constant time. *ACM Transactions on Programming Languages and Systems*, 13(4):626–629, October 1991. Technical Correspondence. 6.2, 8

[Duc11] Roland Ducournau. Implementing statically typed object-oriented programming languages. *ACM Computing Surveys*, 43(3):Article 18, April 2011. 8

[EP18] M. Anton Ertl and Bernd Paysan. Closures — the Forth way. In *34th EuroForth Conference*, pages 17–30, 2018. 5.1

[Ert12] M. Anton Ertl. Methods in objects2: Duck typing and performance. In *28th EuroForth Conference*, pages 96–103, 2012. 6.3, 8

[Ert17] M. Anton Ertl. SIMD and vectors. In *33rd EuroForth Conference*, pages 25–36, 2017. 8

[Gay20] Alex Gaynor. What science can tell us about C and C++'s security. Blog posting, `https://alexgaynor.net/2020/may/27/science-on-memory-unsafety-and-security/`, 2020. 1

[MM81] Michael McCourt and Richard A. Marisa. The string stack. *Forth Dimensions*, III(4):121–124, 1981. 8

[PE19] Bernd Paysan and M. Anton Ertl. The new Gforth header. In *35th EuroForth Conference*, pages 5–20, 2019. 3.12

[PEG10] Sviatoslav Pestov, Daniel Ehrenberg, and Joe Groff. Factor: a dynamic stack-based programming language. In William D. Clinger, editor, *Proceedings of the 6th Symposium on Dynamic Languages, DLS 2010, October 18, 2010, Reno, Nevada, USA*, pages 43–58. ACM, 2010. 8

[RCM96] Elizabeth D. Rather, Donald R. Colburn, and Charles H. Moore. The evolution of Forth. In *History of Programming Languages*, pages 625–658. ACM Press/Addison-Wesley, 1996. 3

Euroforth 2022

Better Values

Improving the extended Forth value concept

Euroforth 2022

Better Values

Example:

**STRINDEX
Indexed strings**

**State of play at
EuroForth 2020**

```
: STRINDEXCOMP, ( xt--- ) \ Compiling action of a child of STRINDEX
  >BODY                                           \ To the address of the data
  CLIT,                                           \ Compile that address
  ]] DUP @ ROT * + CELL+ [[                       \ Calculate address of string
  OPERATORTYPE @OFF CASE
    VMOD@    OF                             ENDOF \ Default returns address
    VMOD!    OF POSTPONE ZMOVE              ENDOF \ Writes to string
    VMODADDR OF                             ENDOF \ ADDR also returns address
    VMODSIZE OF ]] IP>NFA NAME> >BODY @ [[  ENDOF \ SIZEOF returns max length
         BAD-METHOD
  ENDCASE
;

: STRINDEXINTERP ( addr--- ) \ Interpret a child of STRINDEX
...
\ As for STRINDEXCOMP, but not postponed
...
;

: STRINDEXNEW ( arraysize,maxlen--- ) \ Prepare a new indexed string
  1+ DUP ,                          \ Set the string size (allow for 0 terminator)
  SWAP 1+ * ALLOT&ERASE             \ Allocate (allow for 0 or 1 based indexing)
  ['] STRINDEXCOMP, SET-COMPILER    \ When a child is being compiled
  INTERP> STRINDEXINTERP            \ When a child is being interpreted
;

: STRINDEX ( arraysize,maxlen <name>--- ) \ Create an indexed string from inline
  CREATE STRINDEXNEW
;

: ZSTRINDEX ( arraysize,maxlen,z$--- ) \ Create an indexed string from a zstring
  ZCOUNT ($CREATE) STRINDEXNEW
;
```

Euroforth 2022

Better Values

Good points…

- **Logical separation of operations**
- **No magic numbers**
- **All values initialised**
- **Tidy POSTPONEs**

```
: STRINDEXCOMP, ( xt--- ) \ Compiling action of a child of STRINDEX
  >BODY                                           \ To the address of the data
  CLIT,                                           \ Compile that address
  ]] DUP @ ROT * + CELL+ [[                       \ Calculate address of string
  OPERATORTYPE @OFF CASE
    VMOD@    OF                             ENDOF \ Default returns address
    VMOD!    OF POSTPONE ZMOVE              ENDOF \ Writes to string
    VMODADDR OF                             ENDOF \ ADDR also returns address
    VMODSIZE OF ]] IP>NFA NAME> >BODY @ [[  ENDOF \ SIZEOF returns max length
         BAD-METHOD
  ENDCASE
;

: STRINDEXINTERP ( addr--- ) \ Interpret a child of STRINDEX
...
\ As for STRINDEXCOMP, but not postponed
...
;

: STRINDEXNEW ( arraysize,maxlen--- ) \ Prepare a new indexed string
  1+ DUP ,                          \ Set the string size (allow for 0 terminator)
  SWAP 1+ * ALLOT&ERASE             \ Allocate (allow for 0 or 1 based indexing)
  ['] STRINDEXCOMP, SET-COMPILER    \ When a child is being compiled
  INTERP> STRINDEXINTERP            \ When a child is being interpreted
;

: STRINDEX ( arraysize,maxlen <name>--- ) \ Create an indexed string from inline
  CREATE STRINDEXNEW
;

: ZSTRINDEX ( arraysize,maxlen,z$--- ) \ Create an indexed string from a zstring
  ZCOUNT ($CREATE) STRINDEXNEW
;
```

Euroforth 2022

Better Values

Not so good points…

- No index checks
- No string length checks
- Data close to code
- Unlovely SIZEOF

```
: STRINDEXCOMP, ( xt--- ) \ Compiling action of a child of STRINDEX
  >BODY                                            \ To the address of the data
  CLIT,                                            \ Compile that address
  ]] DUP @ ROT * + CELL+ [[                        \ Calculate address of string
  OPERATORTYPE @OFF CASE
    VMOD@    OF                              ENDOF \ Default returns address
    VMOD!    OF POSTPONE ZMOVE               ENDOF \ Writes to string
    VMODADDR OF                              ENDOF \ ADDR also returns address
    VMODSIZE OF ]] IP>NFA NAME> >BODY @ [[   ENDOF \ SIZEOF returns max length
         BAD-METHOD
  ENDCASE
;

: STRINDEXINTERP ( addr--- ) \ Interpret a child of STRINDEX
...
\ As for STRINDEXCOMP, but not postponed
...
;

: STRINDEXNEW ( arraysize,maxlen--- ) \ Prepare a new indexed string
  1+ DUP ,                            \ Set the string size (allow for 0 terminator)
  SWAP 1+ * ALLOT&ERASE               \ Allocate (allow for 0 or 1 based indexing)
  ['] STRINDEXCOMP, SET-COMPILER      \ When a child is being compiled
  INTERP> STRINDEXINTERP              \ When a child is being interpreted
;

: STRINDEX ( arraysize,maxlen <name>--- ) \ Create an indexed string from inline
  CREATE STRINDEXNEW
;

: ZSTRINDEX ( arraysize,maxlen,z$--- ) \ Create an indexed string from a zstring
  ZCOUNT ($CREATE) STRINDEXNEW
;
```

Euroforth 2022

Better Values

Existing word for index checking

```
: VICHECK { pindex paddr -- pindex' paddr } \ Checks for valid index
\ paddr is the address of the data, the first cell of which contains the array size
  pindex 0 paddr @ WITHIN IF              \ Index is valid
    pindex paddr
  ELSE                                    \ Index is invalid
    Z" Invalid index " pindex ZFORMAT Z+
    Z"  for " Z+ paddr >NAME 1+ Z+        \ >NAME does not work for separated data
    Z"  length " Z+ paddr @ ZFORMAT Z+
    ERROR
    0 paddr                               \ Use zeroth index
  THEN
;
```

Could do better...

Euroforth 2022

Better Values

String length checking

```
: SLCHECK ( pz$,pindex,paddr---pz$,pindex,paddr ) \ Check for string length
  2 PICK ZCOUNT NIP                \ Get length of string
  OVER CELL+ @                     \ Get maximum length
  U> IF                            \ Overflow
    Z" String length overflow index "
    2 PICK ZFORMAT Z+
    Z"  for " Z+ OVER >NAME 1+ Z+ \ >NAME does not work for separated data
    FATAL                          \ A buffer overflow is fatal
  THEN
;
```

Buffer overflow is fatal!

Euroforth 2022

Better Values

Include the checks in the child compilation

```
: STRIADDR ( index,bodyaddr---elementaddr ) \ Calculate address of string element
  DUP CELL+ @ 1+ ROT * + 2 CELLS+ \ Optimises to 27 bytes, 8 instructions
;

: STRINDEXCOMP, ( ?,index,xt--- ) \ Compiling action of a child of STRINDEX
  >BODY                                            \ To the address of the data
  CLIT,                                            \ Compile that address
  POSTPONE VICHECK                                 \ Index check
  OPERATORTYPE @OFF CASE
    VMOD@    OF POSTPONE STRIADDR            ENDOF \ Default returns string address
    VMOD!    OF ]] SLCHECK STRIADDR ZMOVE [[ ENDOF \ Writes to string
    VMODADDR OF POSTPONE STRIADDR            ENDOF \ ADDR also returns address
    VMODSIZE OF ]] NIP CELL+ @ [[            ENDOF \ SIZEOF returns max length
         BAD-METHOD
  ENDCASE
;
```

Euroforth 2022

Better Values

Move to separated data part 1

```
: STRINDEXNEW ( arraysize, maxlen --- pdata ) \ Make a new child of STRINDEX
  2DUP 1+ SWAP 1+ * 2+ CELLS IRESERVE&ERASE    \ Allow for 0 term and 0/1 index
  TUCK CELL+ !                                 \ Set maximum length
  TUCK !                                       \ Set index size
;

: STRINDEX ( arraysize,maxlen <name>--- ) \ Create an indexed string from inline
  STRINDEXNEW                    \ Make a new child of STRINDEX
  CREATE ,                       \ Create header and set address of data
  ['] STRINDEXCOMP, SET-COMPILER \ When a child is being compiled
  INTERP> STRINDEXINTERP         \ When a child is being interpreted
;
```

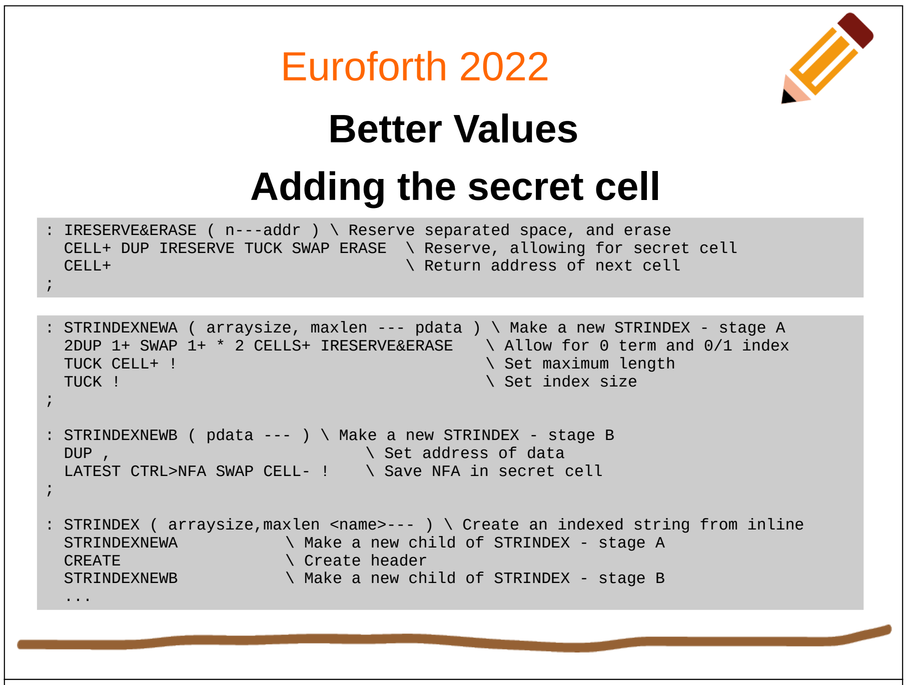
Euroforth 2022

Better Values

Move to separated data part 2

```
: SLCHECK ( pz$,pindex,p*addr---pz$,pindex,p*addr ) \ Check for string length
  2 PICK ZCOUNT NIP               \ Get length of string
  OVER @ CELL+ @                  \ Get maximum length
  U> IF                           \ Overflow
    Z" String length overflow index "
    2 PICK ZFORMAT Z+
    Z"  for " Z+ OVER >NAME 1+ Z+ \ >NAME does not work for separated data
    FATAL                         \ A buffer overflow is fatal
  THEN
;

: STRIADDR ( index,bodyaddr---elementaddr ) \ Calculate address of string element
  @ DUP CELL+ @ 1+ ROT * + 2 CELLS+
;
```

The double fetch problem



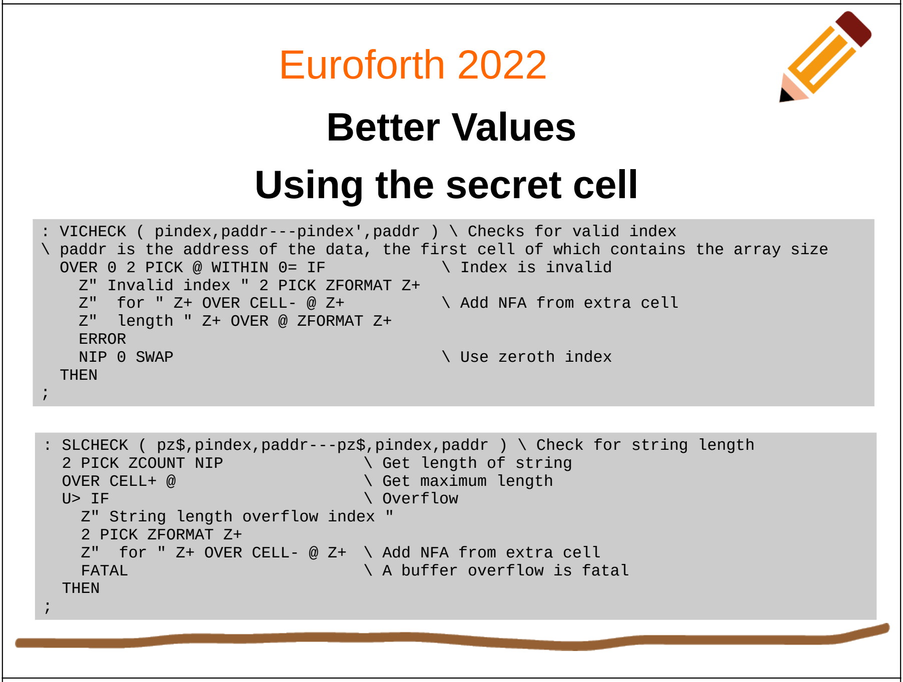
Euroforth 2022

Better Values

Adding the secret cell

```
: IRESERVE&ERASE ( n---addr ) \ Reserve separated space, and erase
  CELL+ DUP IRESERVE TUCK SWAP ERASE  \ Reserve, allowing for secret cell
  CELL+                               \ Return address of next cell
;
```

```
: STRINDEXNEWA ( arraysize, maxlen --- pdata ) \ Make a new STRINDEX - stage A
  2DUP 1+ SWAP 1+ * 2 CELLS+ IRESERVE&ERASE    \ Allow for 0 term and 0/1 index
  TUCK CELL+ !                                 \ Set maximum length
  TUCK !                                       \ Set index size
;

: STRINDEXNEWB ( pdata --- ) \ Make a new STRINDEX - stage B
  DUP ,                           \ Set address of data
  LATEST CTRL>NFA SWAP CELL- !    \ Save NFA in secret cell
;

: STRINDEX ( arraysize,maxlen <name>--- ) \ Create an indexed string from inline
  STRINDEXNEWA            \ Make a new child of STRINDEX - stage A
  CREATE                  \ Create header
  STRINDEXNEWB            \ Make a new child of STRINDEX - stage B
  ...
```



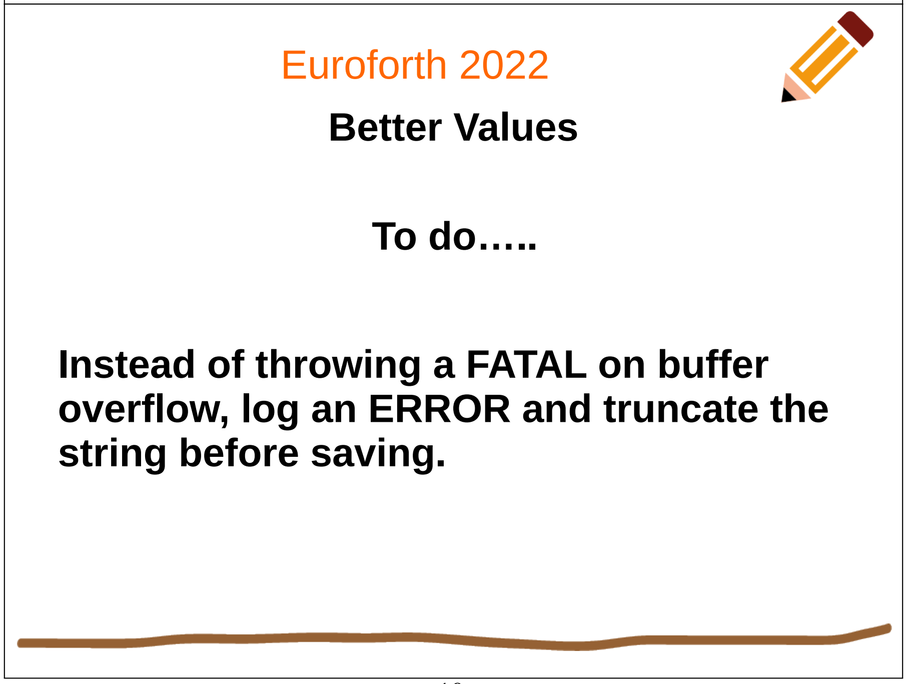
Euroforth 2022

Better Values

Using the secret cell

```
: VICHECK ( pindex,paddr---pindex',paddr ) \ Checks for valid index
\ paddr is the address of the data, the first cell of which contains the array size
  OVER 0 2 PICK @ WITHIN 0= IF            \ Index is invalid
    Z" Invalid index " 2 PICK ZFORMAT Z+
    Z"  for " Z+ OVER CELL- @ Z+          \ Add NFA from extra cell
    Z"  length " Z+ OVER @ ZFORMAT Z+
    ERROR
    NIP 0 SWAP                            \ Use zeroth index
  THEN
;
```

```
: SLCHECK ( pz$,pindex,paddr---pz$,pindex,paddr ) \ Check for string length
  2 PICK ZCOUNT NIP               \ Get length of string
  OVER CELL+ @                    \ Get maximum length
  U> IF                           \ Overflow
    Z" String length overflow index "
    2 PICK ZFORMAT Z+
    Z"  for " Z+ OVER CELL- @ Z+  \ Add NFA from extra cell
    FATAL                         \ A buffer overflow is fatal
  THEN
;
```



Euroforth 2022

Better Values

To do…..

Instead of throwing a FATAL on buffer overflow, log an ERROR and truncate the string before saving.

Euroforth 2022

FQL Result Set Analysis

Introducing Forth Local Functions

Euroforth 2022

FQL Result Set Analysis

Before...

```
: TAB-SETDATA { | pinnum poutnum pdescr piter[ GtkTreeIter ] -- } \ Set table data
...
  SQL| SELECT innum,outnum,notes                           \ Get table entries
       FROM tables
       WHERE tabnum = | CURRTAB FQL-N+ |
       ORDER BY innum
  |SQL> IF
    PROWS 0 ?DO
      FQL-NEXTROW IF
        0 PCOL ZDIGITS -> pinnum                           \ Get input number
        1 PCOL ZDIGITS -> poutnum                          \ Get output number
        2 PCOL         -> pdescr                           \ Get description
        TABTREESTORE piter[ NULL gtk_tree_store_append     \ Add a row
        TABTREESTORE piter[
        0 pinnum 1 poutnum 2 pdescr -1 gtk_tree_store_set  \ Put data into the row
      THEN
    LOOP
  THEN
...
```

Euroforth 2022

FQL Result Set Analysis

The goal...

```
: TAB-SETDATA { | piter[ GtkTreeIter ] -- } \ Set table data
...
  SQL| SELECT innum,outnum,notes                           \ Get table entries
       FROM tables
       WHERE tabnum = | CURRTAB FQL-N+ |
       ORDER BY innum
  |SQDO
    TABTREESTORE piter[ NULL gtk_tree_store_append         \ Add a row
    TABTREESTORE piter[
    0 r-innum 1 r-outnum 2 r-notes -1 gtk_tree_store_set   \ Put data into the row
  SQLOOP
...
```

Euroforth 2022

FQL Result Set Analysis

Rolling up the flow control

```
   IF
     PROWS 0 ?DO
       FQL-NEXTROW IF
...
       THEN
     LOOP
   THEN
```

```
: |SQDO ( zaddr--- ) \ Runs an SQL query, starts a DO..LOOP and fetches row
  NOINTERP ;            \ Cannot interpret
NDCS:                   \ Compiling
  IN-SQL OFF            \ Finished compiling SQL
  POSTPONE (|SQL>)      \ Run the query
  0 CLIT,               \ The initial value for the DO..LOOP
  s_?do,  3             \ Compile the DO
  POSTPONE FQL-NEXTROW  \ Get the next row
  s_?br>,  2            \ Compile the IF
;

: SQLOOP ( --- ) \ Complete a query analysis loop
  NOINTERP  ;           \ Cannot interpret
NDCS:
  2 ?PAIRS  s_res_br>,  \ Compile the THEN
  3 ?PAIRS  s_loop,     \ Compile the LOOP
;
```

Euroforth 2022

FQL Result Set Analysis

How to identify the column names?

Possibility 1:
Identify at execution time

Possibility 2:
At compilation time, create set of locals

Euroforth 2022

FQL Result Set Analysis

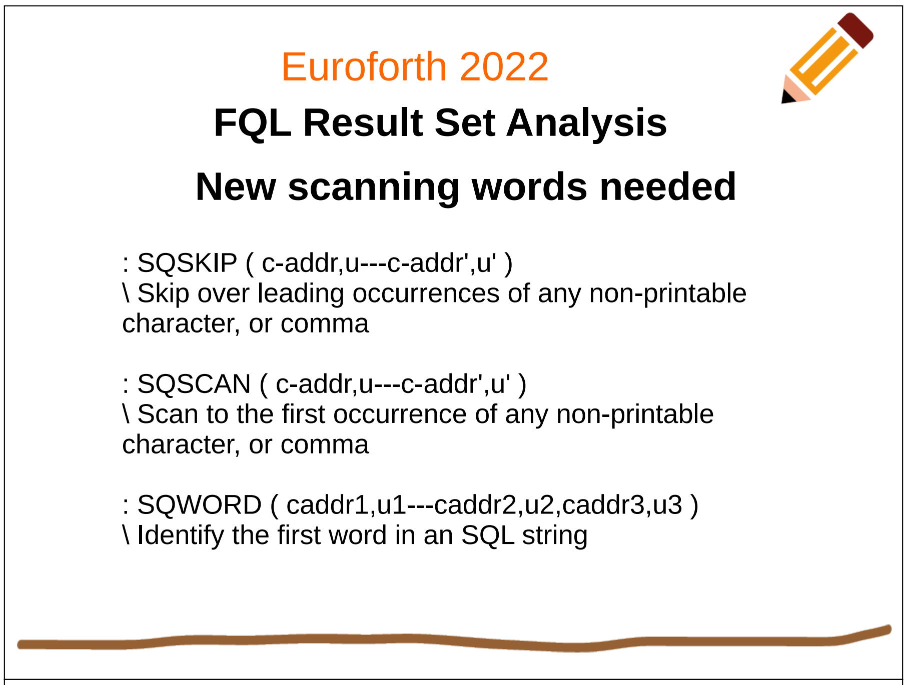
Parsing the SQL query

Easy

```
SQL| SELECT ircallm,ircalle,irspeed,irtimeswitch,timeswitches.tsresult
     FROM ironer
     ...
```

Hard

```
SQL| SELECT IFNULL(operator.name,CONCAT( | P" Unknown operator" ZSP Z+ FQL-Z$+ | ,
         work.opnum) )  AS 'Operator',
           DATE_FORMAT(start , | Z" %e/%m/%Y %H:%i" FQL-Z$+ | )     AS 'Clock in',
         DATE_FORMAT(end   , | Z" %e/%m/%Y %H:%i" FQL-Z$+ | )     AS 'Clock out',
         TIME_FORMAT(MAKETIME(worked/60,worked MOD 60,0),'%H:%i') AS 'Worked'
     FROM work
     ...
```


Euroforth 2022

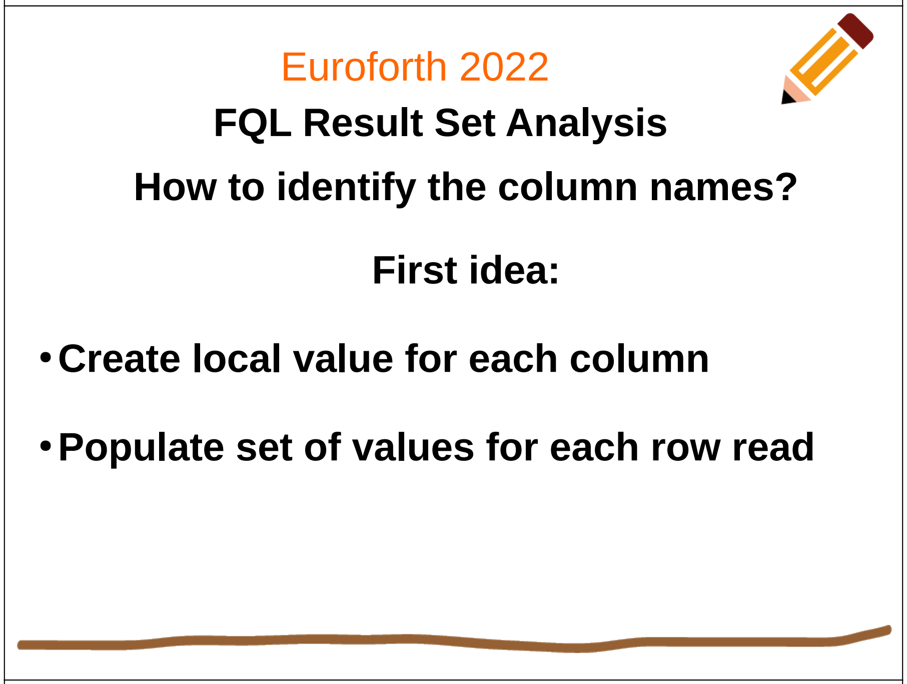
FQL Result Set Analysis

New scanning words needed

: SQSKIP (c-addr,u---c-addr',u')
\ Skip over leading occurrences of any non-printable character, or comma

: SQSCAN (c-addr,u---c-addr',u')
\ Scan to the first occurrence of any non-printable character, or comma

: SQWORD (caddr1,u1---caddr2,u2,caddr3,u3)
\ Identify the first word in an SQL string

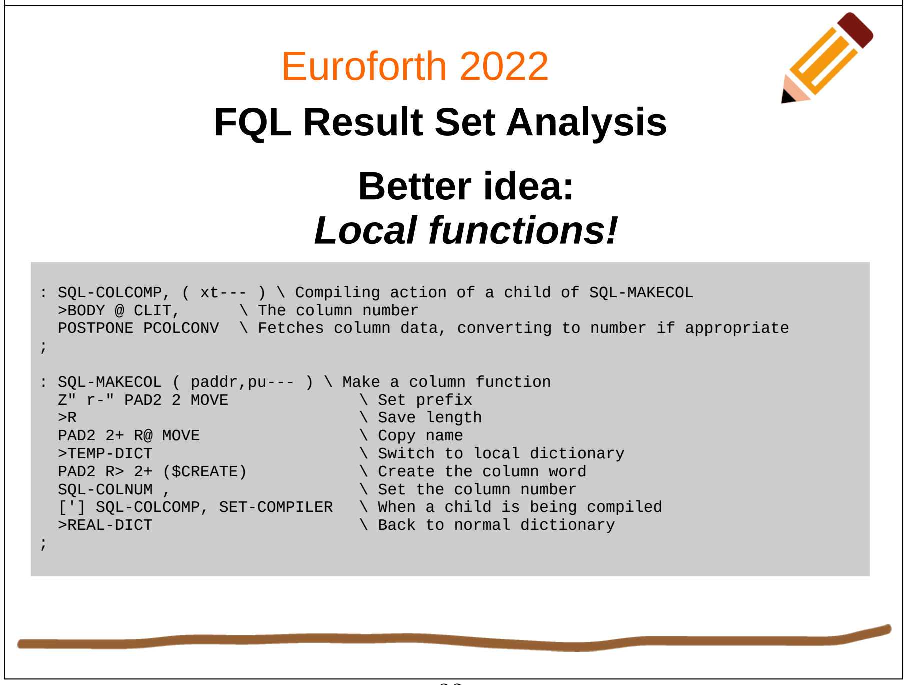

Euroforth 2022

FQL Result Set Analysis

How to identify the column names?

First idea:

- **Create local value for each column**
- **Populate set of values for each row read**


Euroforth 2022

FQL Result Set Analysis

Better idea:
Local functions!

```
: SQL-COLCOMP, ( xt--- ) \ Compiling action of a child of SQL-MAKECOL
  >BODY @ CLIT,      \ The column number
  POSTPONE PCOLCONV  \ Fetches column data, converting to number if appropriate
;

: SQL-MAKECOL ( paddr,pu--- ) \ Make a column function
  Z" r-" PAD2 2 MOVE             \ Set prefix
  >R                             \ Save length
  PAD2 2+ R@ MOVE                \ Copy name
  >TEMP-DICT                     \ Switch to local dictionary
  PAD2 R> 2+ ($CREATE)           \ Create the column word
  SQL-COLNUM ,                   \ Set the column number
  ['] SQL-COLCOMP, SET-COMPILER  \ When a child is being compiled
  >REAL-DICT                     \ Back to normal dictionary
;
```

Euroforth 2022

FQL Result Set Analysis

Start and cleanup

```
: ?START-LOCALS ( --- ) \ Set dictionary pointers to local, if not already done so
  TDPstart @ 0= IF   \ Not yet initialised
    START-LOCALS     \ No need to restore, it is done by ;
  THEN
;
```

```
: MICROSS-CLEANUP-LOCALS ( --- ) \ Clean up locals, if used - called by ; etc.
  TDPstart @ IF      \ Only compile if we had local values or functions
    FORGET-LOCALS    \ Lose local definitions
  THEN
; ' MICROSS-CLEANUP-LOCALS ' CLEANUP-LOCALS PATCHXT 2DROP DROP

: MICROSS-HASLVs? ( ---f ) \ True if current definition has local vals or funcs
  TDPstart @     \ Local values or functions exist
; ' MICROSS-HASLVs? ' HASLVs? PATCHXT 2DROP DROP
```

Euroforth 2022

FQL Result Set Analysis Demonstration

```
: SQTEST1 { | ptest -- } \ List details for step programs
  123 -> ptest
  SQL| SELECT program,name,miny,maxy,run
       FROM stepprogram
  |SQDO
    cr r-program . r-name z$. SPACE r-miny . r-maxy . r-run . ptest .
  SQLOOP
;

sqtest1
1 Test 1 100 120 0 123
2 Test 2 200 220 1 123  ok


: SQTEST2 { | ptest2 -- } \ Analyse operator privileges
  SQL| SELECT COUNT(*) AS numoperators, privil
       FROM operator
       GROUP BY privil
  |SQDO
    cr r-privil . r-numoperators .
    456 -> ptest2
  SQLOOP
  cr ptest2 .
;

sqtest2
0 6
1 204
2 1
3 14
4 2
456  ok
```

Euroforth 2022

FQL Result Set Analysis

New and unique feature of Forth!

Dynamically create a function that has scope only within the current definition

Fuzzing Forth

Apply Fuzz Tests to Forth

EuroForth'22 conference 2022-09

Ulrich Hoffmann

uho@ .de

Overview

Fuzzing Forth

- Introduction
- Correctness Notions
- Generators
- Mutators
- Sanitizers
- Tests
- Fuzzing
- Conclusion

Introduction

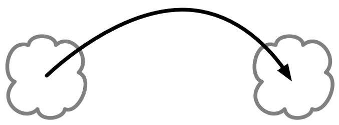
What fuzzing is all about?

- We assure quality of applications by **testing**
  - Manually, especially for embedded systems → time consuming
  - Automatically, correct functions, regression, TestDrivenDevelopment
  - We mostly test the good cases, infrequently behaviour in bad situations
- ***Fuzz Tests*** or ***Fuzzing*** tests applications with arbitrary data to see if they break

"*Crash often crash early!*" - but automated

Correctness Notions
What's in a word?

- When is a word correct?
  - need to describe the behaviour of a word
  - an approach: a word does a state transition
    - from a current state
    - to a next state
    - can be deterministic or non deterministic



States

- States can be complicated
  - not just labels as with finite state machines
  - **Forth System State**: includes the stack and return stack content, all dictionary content, the existing definitions, etc.
  - **Computer State**: contents of files, memory content, etc.
  - **Environment State**: relevant state of external components
- Think of states as huge records or vectors
- A state or sets of states can be described by conditions

"the (set of all) states that satisfy the condition"

State Transitions

- many aspects of a state are not relevant for the transition and stay as they are
- a transition (i.e. the behaviour of a word) can be described by a pre-condition P and a post-condition Q

Stack Comments

- We describe conditions already in Forth with stack comments

```
0< ( n -- flag )
   n : The condition TOS is a natural number in the current state
   -- : symbolizes the transition
   flag: the condition TOS is with all bits set (true) or all bits reset (false),
   i.e. a flag in the next state
```

- stack diagrams are not sufficient to specify the operation exactly
  0> has the same stack effect but a different behaviour.

Stack Comments

```
0< ( n -- flag )
   ...
   flag: the condition TOS is with all bits set (true) or all bits reset (false),
   i.e. a flag in the next state
```

- the post-stack-condition is too weak
- stronger post condition for 0<:
  flag where TOS'=true if TOS<0, TOS'=false if TOS>=0
- appropriate pre- and post-condition can describe the behaviour of a word as precisely as desired, but they may be difficult to specify

Partial Correctness of a word (a transition)

A word ist partially correct with respect to conditions P and Q:

- if the current state satisfies the pre-condition P and **if the word terminates** (i.e. does not crash) then the next state satisfies the post-condition Q

if the current state does not satisfy the pre-condition then the next state is undefined (computer scientists model this often with non-terminating programs or arbitrary results).

Total Correctness of a word (a transition)

A word is totally correct with respect to conditions P and Q:

- if the current state satisfies the pre-condition P **then the word terminates** (i.e. does not crash) **and** the next state satisfies the post-condition Q

if the current state does not satisfy the pre-condition then the next state is undefined.

Robustness of a word (a transition)

A word is robust with respect to P and Q

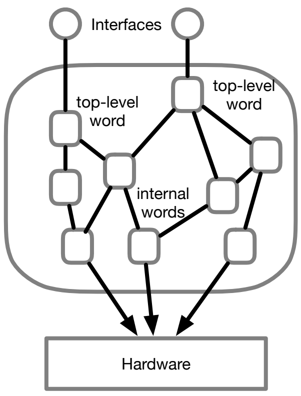
- if it always does a transition to a next state.
- is totally correct with respect to P and Q
- if the current state does not satisfy the pre-condition then *an error is signalled.*
  - *throw an exception*
  - *return an distinct error value*
  - *...*

Fuzzing checks if an application i.e. its top-level words are robust

Fuzzing
What the fuzz?

Fuzzing checks if an application i.e. its top-level words are robust

- top level words are accessible for outside
- realize outside interfaces
- **Fuzzing**
  - **invoke a top-level words with arbitary data**
  - **check if the system crashes**
- Random data → generators and mutators
- best no crashes → sanitizers



Generators
How to create random data?

- We need to generate random (stack) items.
- even distribution but also normal and other distributions
- classical Starting Forth random number generator

```
( Random number generation -- High level )

VARIABLE rnd   HERE rnd !
: RANDOM  rnd @ 31421 *  6927 +  DUP rnd ! ;
: CHOOSE  ( u1 -- u2 )  RANDOM UM*  NIP ;
```

- linear congruence generator not good as factor for normal distribution

[1] http://www0.cs.ucl.ac.uk/staff/D.Jones/GoodPracticeRNG.pdf

Generators
How to create random data?

- KISS generators are simple and have better properties
- JKISS32 [1] is based on 32-bit integer arithmetic
  - passes all of the Dieharder tests and the BigCrunch tests

```
\ JKISS32 for 32Bit Systems (algorithm by David Jones)

Variable x  123456789 x !
Variable y  234567891 y !
Variable z  345678912 z !
Variable w  456789123 w !
Variable c  0 c !

: kiss ( -- x )
   y @  dup 5 lshift xor   dup 7 rshift xor   dup 22 lshift xor   y !
   z @ w @ + c @ +     w @ z !     dup 0< 1 and 0= 0= 1 and c !   2147483647 and w !
   x @ 1411392427 + x !
   x @ y @ + w @ + ;
```

Generators
How to create random data?

- create normal distribution from even distribution
- different algorithms such as Marsaglia polar method or Box Muller Transform require floating point
- Rule of 12 (sum and average) is simple and works on integers
  - but only valid if random delivers independent random values
  - linear congruence generators (such as random earlier) do not have this property, KISS does.

```
: CHOOSE  ( u1 -- u2 )  KISS UM*  NIP ;

: NORMAL ( u1 -- u2 )
    0  12 0 DO  OVER CHOOSE +  LOOP  12 / SWAP DROP ;
```

Generators

How to create random data?

- from CHOOSE and NORMAL we can build generators for typical Forth data
- cell data on the stack

```
: cgen ( -- b )  256 choose ;
: ngen ( -- n )  -1 choose ;
: +ngen ( -- n )  -1 1 rshift choose ;
: ugen ( -- u )  -1 choose ;
```

or also with specific distribution

- strings of given exact or maximal length

```
: 'x'gen ( -- c )  128 bl - choose bl + ;

: $=gen ( u1 -- c-addr u2 ) \ string is excactly u1 characters. allocates, must be freed after use
   dup allocate throw swap  2dup bounds ?DO  'x'generate I c! LOOP ;

: $gen ( u1 -- c-addr u2 ) \ string is shorter than u1 characters. allocates, must be freed after use
   choose $=gen ;
```

Generators

How to create random data?

- Generate random composed data structures such as
  - structs and
  - arrays or
  - linked lists, etc.

  by defining appropriate (recursive) generators.

```
: person-gen ( -- addr ) ... ;
```

Mutators

How to change existing data?

- Mutators are similar to generators but modify existing data
- cell data on the stack
  - **`cmut`** (char1 rate -- char2)
  - **`nmut`** (n1 rate -- n2)
  - **`+nmut`** (+n1 rate -- +n2)
  - **`umut`** (u rate -- u2)
- or strings
  - **`$=mut`** (c-addr1 u rate -- c-addr2 u) just changes the characters
  - **`$mut`** (c-addr1 u1 rate -- c-addr2 u2) changes length and character

Mutators
How to change existing data?

- Modify random composed data structures such as
  - structs and
  - arrays or
  - linked lists, etc.

  by defining appropriate (recursive) mutators.

```
: person-mut ( addr1 rate -- addr2 ) ... ;
```

Sanitizers
How to detect and signal crashes?

- If application crash, then it is hard to monitor them.

  → Turn crashes into reported errors
- **Sanitizers** check if inputs are valid
- Memory sanitizers - detect illegal memory access , throw for memory faults
- Stack Sanitizers - detect stack over and underflow
- Control flow sanitizers - check for valid return addresses on EXIT
- ...

Sanitizers
How to detect and signal crashes?

- Memory sanitizers - detect illegal memory access , throw for memory faults
  - @ ! c@ c! with valid memory test, throw memory-fault on invalid address
  - linked list of valid regions that are checked on access
  - bit-field of valid memory words or bytes

```
: ?valid ( addr -- addr ) dup valid? 0= #memory-fault and throw ;

: @ ( addr -- x ) ?valid @ ;
: ! ( x addr -- ) ?valid ! ;
```

Sanitizers
How to detect and signal crashes?

- Stack Sanitizers - detect stack underflow

```
: arguments ( i*x u -- i*x )
    >r depth r> u< #stack-underflow and throw ;

1 2 3    3 arguments   . . .   ( 3 2 1 )
```

- and overflows
  - much harder to detect, every push must check
  - might need hardware support

Sanitizers
How to detect and signal crashes?

- Control flow sanitizers - check for valid return addresses on EXIT

```
: exit ( i*x u -- i*x )  rdrop
   r@  invalid-return-addess?  #return-stack-imbalance and throw
;
```

- others:
  - check exception stack
  - ...

[1] ftp://ftp.taygeta.com/pub/Forth/Applications/ANS/tester.fr

Tests
How do we test?

- A popular framework for tests of forth words are testers derived from John Hayes ANS Forth tester [1]

```
{ 3 4 + -> 7 }
```

- Tests + on a single value.
- We can elaborated test suites from this, as Gerry Jackson does for Forth200x compliance.

Fuzzing
What the fuzz?

- In order to fuzz our applications
  - run the Hayes style unit tests - fix all bugs
  - define the top-level words (TLW) using sanitizers
  - run the Hayes style unit tests - no issue expected
  - run fuzz tests in this style

```
\ assuming TLW ( c-addr1 u1 c-addr2 u2 -- c-addr3 u3)

many times DO
  { 100 $gen  S" secret" 75 $mut TLW clearstack -> }
LOOP
```

Fuzzing Forth
Apply Fuzz Tests to Forth

Conclusion
What's below the bottom line?

- Correctness Notions - partial, total correctness, robustness
- Generators - KISS generator, rule of 12
- Mutators - change existing data
- Sanitizers - make crashes into reported errors
- Tests - Hayes style testing
- Fuzzing - stressing top level words (interfaces)

Questions?

progress towards porting EISPACK to forth

Krishna Myneni

EuroForth 2022

V1.0

- numerical linear algebra source libraries for scientific computing
- numerical linear algebra modules in the FSL
- basic forms of linear systems problems I
- basic forms of linear systems problems II
- overview of EISPACK
- goals of porting EISPACK to Forth
- challenges of translating unstructured Fortran to Forth
- status of EISPACK port to Forth port
- what remains to be done?

numerical linear algebra source libraries for scientific computing

$$\begin{tikzcd}
 & \textbf{LAPACK}\text{ (1992, UTenn)} & \\
 & & \textbf{LINPACK}\text{ (1979, ANL)} \arrow[ul] \\
\textbf{EISPACK}\text{ (1974, ANL)}^{\dagger} \arrow[uur] \arrow[urr] & & \textbf{BLAS}^{\ddagger}\text{ Level 1 (1979,JPL)} \arrow[u]
\end{tikzcd}$$

- EISPACK and LINPACK were developed in 1970s – 1980s to provide *well-documented*, *well-tested* source libraries for scientific computing.
- EISPACK solves eigensystems of equations. Code is translated from *Algol*[¥] *→ Fortran (→ Forth)*.
- LINPACK solves linear systems of equations. Uses BLAS (Basic Linear Algebra Subprograms) Level 1.
- LAPACK combines functionality of LINPACK and EISPACK. It factors core matrix and vector computations (BLAS Level 3). Matlab, R, and other software use LAPACK.

† T. Haigh, *"An interview with Jack J. Dongarra,"* 26 April 2004, Soc. Industr. Appl. Math.; http://history.siam.org/pdfs2/Dongarra_ returned_SIAM_copy.pdf
‡ C. L. Lawson, et al., ACM Transactions on Math. Software 5, pp 308–323 (1979); https://doi.org/10.1145/355841.355847
¥ J. H. Wilkinson and C. Reinsch, Handbook for Automatic Computation: vol II Linear Algebra, Part 2, Springer-Verlag, New York 1972.

numerical linear algebra routines in the Forth Scientific Library†

module	description
`lufact`	factor a matrix **A** into a product of lower triangular (**L**) and upper triangular (**U**) matrices.
`dets`	find determinant of a matrix which has been factored in LU form.
`backsub`	solve linear system of equations using LU factorization: **A X** = **B**, where **A = L U**
`invm`	find the inverse of a matrix using LU factorization.
`gaussj`	provides tools for matrix arithmetic, finding inverse, solving linear system of equations and least-squares problems.
`svd`	solve matrix equations involving nearly singular matrices.

- **FSL** provides some **LINPACK** functionality.
- **EISPACK** functionality is completely missing from the **FSL**!

† The Forth Scientific Library, https://www.taygeta.com/fsl/scilib.html

basic forms of linear systems problems I

I. A X = **B** : **A** and **B** are given; solve for **X**

simple case:

$2x_0 + 3x_1 = -6$

$4x_0 + 8x_1 = 10$

matrix form:

$$\begin{pmatrix} 2 & 3 \\ 4 & 8 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} -6 \\ 10 \end{pmatrix}$$

solve using Gauss-Jordan elimination with FSL **`gaussj`**:

```
2 2 float matrix a{{
2 1 float matrix b{{
 2.0e0  3.0e0
 4.0e0  8.0e0 2 2 a{{ }}fput  \ init matrix a{{
-6.0e0 10.0e0 2 1 b{{ }}fput  \ init matrix b{{
a{{ b{{ 2 1 gaussj .          \ solve and print error (0 = no error)
2 1 b{{ }}fprint              \ print solution x0 and x1:
                              \ -19.5
                              \ 11
```

basic forms of linear systems problems II

II. A X = λ**X** : **A** is given; solve for λ's and corresponding **X**'s

simple case:

$2x_0 + 3x_1 = \lambda\, x_0$

$3x_0 + 4x_1 = \lambda\, x_1$

matrix form:

$$\begin{pmatrix} 2 & 3 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \lambda \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$$

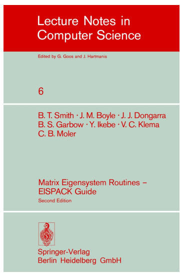
solve using matrix tridiagonalization and QL reduction with EISPACK **`tred2`** and **`imtql2`** :

```
2 2 float matrix A{{
 2.0e0  3.0e0  3.0e0  4.0e0   2 2 A{{ }}fput

2 float array diag{  2 float array subdiag{  2 2 float matrix ot{{

2 2 A{{ diag{ subdiag{ ot{{ tred2  \ tridiagonalize the matrix
2 2 diag{ subdiag{ ot{{ imtql2 .   \ find λs and eigenvectors; print error code
2 diag{ }fprint                    \ print eigenvalues (λs): -0.162278 6.16228
2 2 ot{{ }}fprint                  \ print corresponding eigenvectors:
                                   \ 0.811242 0.58471
                                   \ -0.58471 0.811242
```

overview of EISPACK

“EISPACK is a systematized collection of [Fortran] subroutines† which compute the eigenvalues and/or eigenvectors of six classes of matrices…”

1 *complex* general
2 *complex* Hermitian
3 *real* general
4 *real* symmetric
5 *real* symmetric tridiagonal
6 *real* special tridiagonal


Lecture Notes in Computer Science
Edited by G. Goos and J. Hartmanis
6
B. T. Smith · J. M. Boyle · J. J. Dongarra
B. S. Garbow · Y. Ikebe · V. C. Klema
C. B. Moler
Matrix Eigensystem Routines – EISPACK Guide
Second Edition
Springer-Verlag
Berlin Heidelberg GmbH

EISPACK Guide‡ provides recommended calling sequence of routines, the “EISPACK path”, for a given problem class, e.g.

```
call balanc( … )
call elmhes( … )
call eltran( … )
call hqr2( … )
call balbak( … )
```

to find all eigenvalues and eigenvectors for a real general matrix.

† Fortran source library from August 1983 release is at https://netlib.org/eispack/
‡ B. T. Smith, et al., Matrix Eigensystem Routines – EISPACK Guide 2nd ed., Springer-Verlag 1976.

goals of porting EISPACK routines to Forth

- provide library of eigensystems solvers to support scientific computing in Forth
- provide source library in Forth for portability, ease of use, ease of debugging, and ability to modify/adapt the code
- translate unstructured Fortran to structured Forth for improved source comprehensibility
- work with FSL style matrices and arrays
- provide test code and examples in library
- check results of Forth computations against “original” Fortran

challenges of translating unstructured Fortran to Forth

```
      subroutine imtql1(n,d,e,ierr)
    :
    :
c     look for small sub-diagonal element
  105    do 110 m = l, n
    :
            if (r .eq. 0.0d0) go to 210
    :
  200    continue
c
         d(l) = d(l) - p
         e(l) = g
         e(m) = 0.0d0
         go to 105
c     recover from underflow
  210    d(i+1) = d(i+1) - p
         e(m) = 0.0d0
         go to 105
c     order eigenvalues
  215    if (l .eq. 1) go to 250
c     for i=l step -1 until 2 do
         do 230 ii = 2, l
            i = l + 2 - ii
            if (p .ge. d(i-1)) go to 270
            d(i) = d(i-1)
  230    continue
c
  250    i = 1
  270    d(i) = p
  290 continue
```

$\longrightarrow$

```
: imtql1 ( n d e -- ierr )
   :
   :
\  look for small sub-diagonal element
      BEGIN
        N I DO
   :
        uflow IF
\  recover from underflow
          d{ ii 1+ } f@ p f@ f- d{ ii 1+ } f!
          false to uflow
        ELSE
          d{ I } f@ p f@ f- d{ I } f!
          g f@ e{ I } f!
        THEN
        0.0e0 e{ m } f!
      REPEAT

\  order eigenvalues
      I 0 = IF
        p f@ d{ 0 } f!
      ELSE
\ for i=l step -1 until 2 do
        I 2+ 1 DO
          J 1+ I - to ii
          p f@ d{ ii 1- } f@ f>= IF
            LEAVE
          THEN
          d{ ii 1- } f@ d{ ii } f!
        LOOP
        p f@ d{ ii } f!
      THEN
    LOOP   \ end main loop
```

status of EISPACK port to Forth

word	description	In Pr	C/N.T.	C/T	Demo
`balanc`	balance a real matrix	✓			
`balbak`	form eigenvectors of a general real balanced matrix		✓		
`elmhes`	reduce submatrix of real general matrix to upper Hessenberg form		✓		
`eltran`	accumulate similarity transforms for reduction of real general matrix		✓		
`hqr2`	find eigenvalues and eigenvectors of real general matrix	✓			
`htribk`	form eigenvectors of complex Hermitian matrix			✓	`cherm-01.4th`
`htridi`	reduce complex Hermitian matrix to real symmetric tridiagonal			✓	`cherm-01.4th`
`imtql1`	find the eigenvalues of a real symmetric tridiagonal matrix			✓	`rsymm-01.4th`
`imtql2`	find eigenvalues and eigenvectors of real symmetric tridiag matrix			✓	`rsymm-02.4th`
`tred1`	reduce real symmetric matrix to tridiagonal matrix			✓	`rsymm-01.4th`
`tred2`	reduce real symmetric matrix to symmetric tridiagonal matrix			✓	`rsymm-02.4th`

In Pr = *Fortran* → *Forth* translation in progress
C/N.T. = Completed translation, **not tested**
C/T = Completed translation, **tested**

C/T Forth code may be found at
https://github.com/mynenik/kForth-64/tree/master/forth-src/eispack

complex general
complex Hermitian ✓
real general
real symmetric ✓
real symmetric tridiagonal ✓
real special tridiagonal

what remains to be done?

- complete translation and testing of words needed to solve eigensystems for *real* general matrices (target date: end of 2022)
- begin translation of words for solution of *complex* general matrices (2023)
- write demo programs to test and illustrate use of EISPACK in Forth (2023 – 2024)
- testing, testing, testing …

applications are the payoff!

The Left-Hand Path
dark confessions of a Forth hobbyist

Glyn Faulkner
EuroForth 2022

2022-09-17

This is the gateway to Hell, baby...
Welcome to The Underworld.
— *Kassandra Cross*

The Left-hand Path

In Western Esotericism...

Right-hand path	magic used for good, or guided by a code of ethics
Left-hand path	magic used for evil, or without consideration of morality

In Forth...

The Right-hand Path	Forth used as a powerful tool to solve real-world problems quickly and efficiently
The Left-hand Path	writing many Forth and Forth-adjacent language interpreters that the world *definitely* doesn't need

Joy: the Gateway Drug

The original concatenative functional language by Manfred von Thun

```
calc ==
  [ numerical ]
  [ ]
  [ unswons
    [ dup [+ - * /] in ]
    [ [ [calc] map uncons first ] dip call ]
    [ "bad operator\n" put ]
    ifte ]
  ifte;
```

From Joy to Funeral

A concatenative functional language in Polish notation

```
def reverse [ fold 'cons [] ]
def odd [= 1 % 2]
def even [= 0 % 2]

-- fold [fun] start [list]
def *fold [ fold dig 2 'dup ]
def fold [
    doif
        [ dip 'drop drop ]
        [ fold *apply dip 'swap dig 2 'uncons ]
    dip 'unrot *null rot
]
def rot   [ exhume 2 ]
def unrot [ bury 2 ]
```

Funeral (2011-12)

Used for HTML generation

```
html body div h1 "Hello World"

def html    [ newTag "html" setDefault
                xmlns="http://www.w3.org/1999/xhtml" ]
def body    [ newTag "body" ]
def div     [ newTag "div" ]
def h1      [ newInlineTag "h1" ]
```

including some ugly proprietary markup needed for a work project

```
def guess_value_from_name [
    doif
        [prepend "<<IPF~" append ">>" drop]
        [prepend "<<" append cons .. append ">>" ]
    = "IPQ" dup take 3 dup
]
```

Cantilever (2014)

An indirect-threaded Forth-like written in 32-bit x86 assembly, inspired by JonesForth with influences from Joy and Funeral

```
: --    [ '\n' = ] scan-in    ;    #immediate
-- and now we have comments. Yay!
```

First class dates

```
 0,1 ok 2022-02-28  1 + putdate  nl
        2020-02-28  1 + putdate ;
2022-03-01
2020-02-29
```

And times

```
 0,1 ok 11:33:30  32 +  puttime ;
11:34:02
```

The downward spiral

HackForth (2014)

```
word NextWord 8 "nex"   # ( -- label )
    call SkipSpaces
    call MakeLabel
    addl %ecx, %edx
    movl %edx, (next_input)
end
```

Thing (2014)

```
prim compile_lit ",lit"  # ( n -- )
    m_dup
    movl $lit, %eax
    call compile_call
    stosl
    m_drop
ret
```

The Quest for Minimalism

STTW (2015)

```
op fetch  "@"      _dup ; mov (%edx), %eax
op store  "!"      mov %eax, (%edx) ; _drop
```

TinyASM (2018)

```
( ?0 w1 w2 ...  if x is non-zero skip w1 )
: ?0  ( x -- ) 0<> cell-size and >r + r> ;
```

FifthWheel (2018)

```
?: dup ( n -- n n ) dsp@ @ ;
?: drop ( x -- ) dsp@ cell+ dsp! ;
```

Rage-coding

Projects I started due to anger or frustration, then quickly abandoned once I had calmed down.

WebOfHate (2018)

A small memory-footprint web browser that puts the user, instead of coporations, back in control (reaction to trying to compile Chromium from source)

BootstrapFromMBR (2020)

Let's throw our operating systems away and return to the stone-age (reaction to *all* modern operating systems!)

Wide (2021)

A tiny Forth IDE, intended to include compiler, debugging tools and full-featured editor (reaction to learning that the Atom editor exists)

The Need for Speed (of development)

OneDayProject (2022-03-08)

A native code compiler in approximately sixty x86 machine instructions.

```
despatch: # read a 16-bit token and despatch
          # based on the two high-bits.
    _dup
    xor %eax, %eax
    mov _src, %esi
    subl $1, _slen
    jc bye
    lodsw
    mov %esi, _src
    xor %ecx, %ecx
    shld $2, %ax, %cx
    shl $2, %ax
    mov handlers(,%ecx,4), %ecx
    jmp *%ecx
```

Lessons learned walking the left-hand path

- Replacing `lods` with a separate move from memory and add is often a performance gain for ITC code
- Replacing `lods` with `pop` also works (bigger difference on Intel)!
- You *can* implement direct-threading using `ret` as `NEXT` and ESP as the instruction-pointer. But just don't.
- Binary source code generally isn't a great idea...
- ... *but* having a binary-token intermediate representation simplifies your compiler and speeds the process of bootstrapping a new Forth.
- It does not appear to be possible to fit a useful Forth system into the 510 bytes available on an x86 boot sector.
- You can write an assembler in Forth using only `c,`... if you enjoy pain.
- Forth can be bootstrapped using a subset of Forth, without the need for compile-time execution, as `if` and `recurse` represent predictable sequences of instructions.
- It is possible to bring-up a rudimentary Forth system in one day, even in assembly.
- GNU assembler isn't as bad as you think.
- Rage-programming is rarely productive!

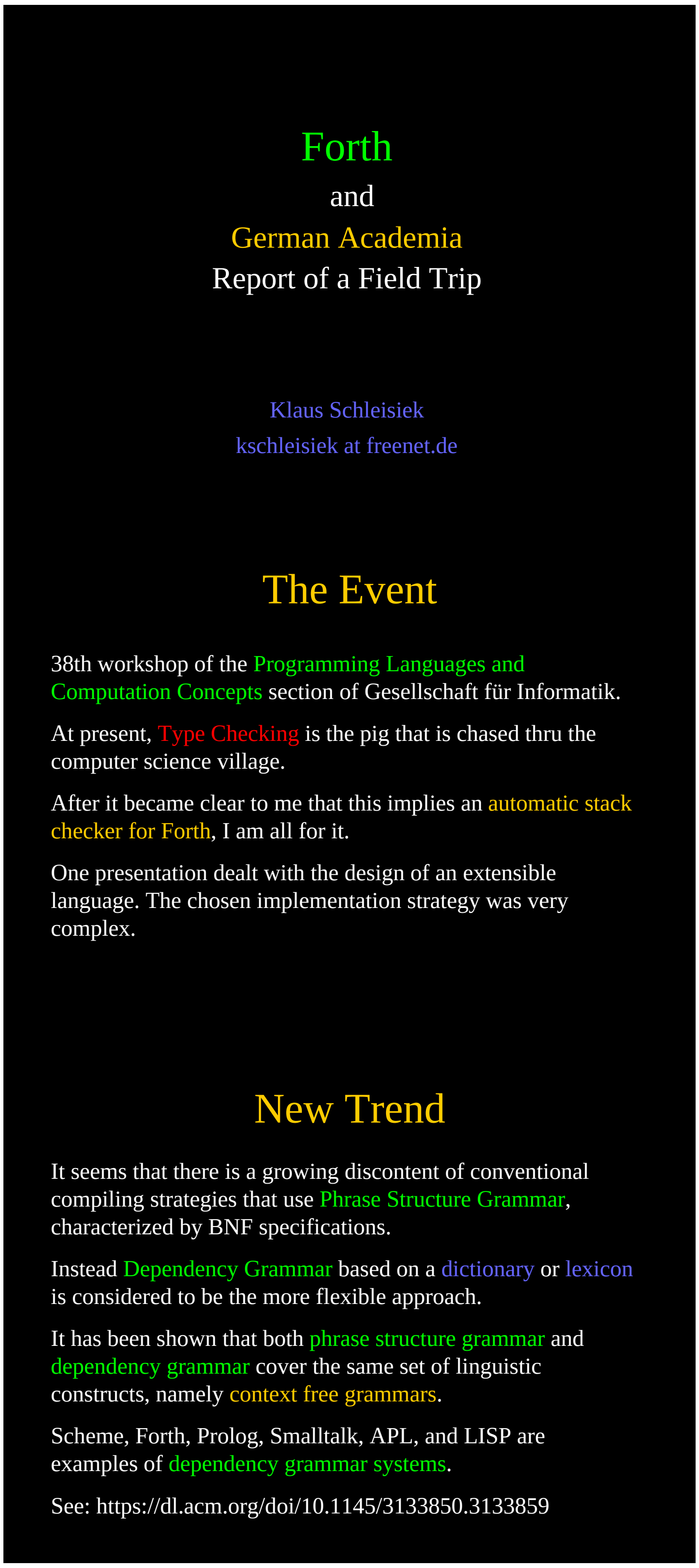
Where next?

How about a parameterised Forth interpreter generator?

```
[marsu@celaeno 4g]$ ./4g -t ITC -T -m ANSI -o forth
Indirect-threaded x86_64 Linux ANSI Forth
Options: top-of-stack in register, linked-list dictionary
Generating forth.S
gcc -m64 forth.S -o forth
Done
[marsu@celaeno 4g]$ ./forth
```

Ask me how this is going next year!

Comments/Questions?



Forth

and

German Academia

Report of a Field Trip

Klaus Schleisiek

kschleisiek at freenet.de

The Event

38th workshop of the Programming Languages and Computation Concepts section of Gesellschaft für Informatik.

At present, Type Checking is the pig that is chased thru the computer science village.

After it became clear to me that this implies an automatic stack checker for Forth, I am all for it.

One presentation dealt with the design of an extensible language. The chosen implementation strategy was very complex.

New Trend

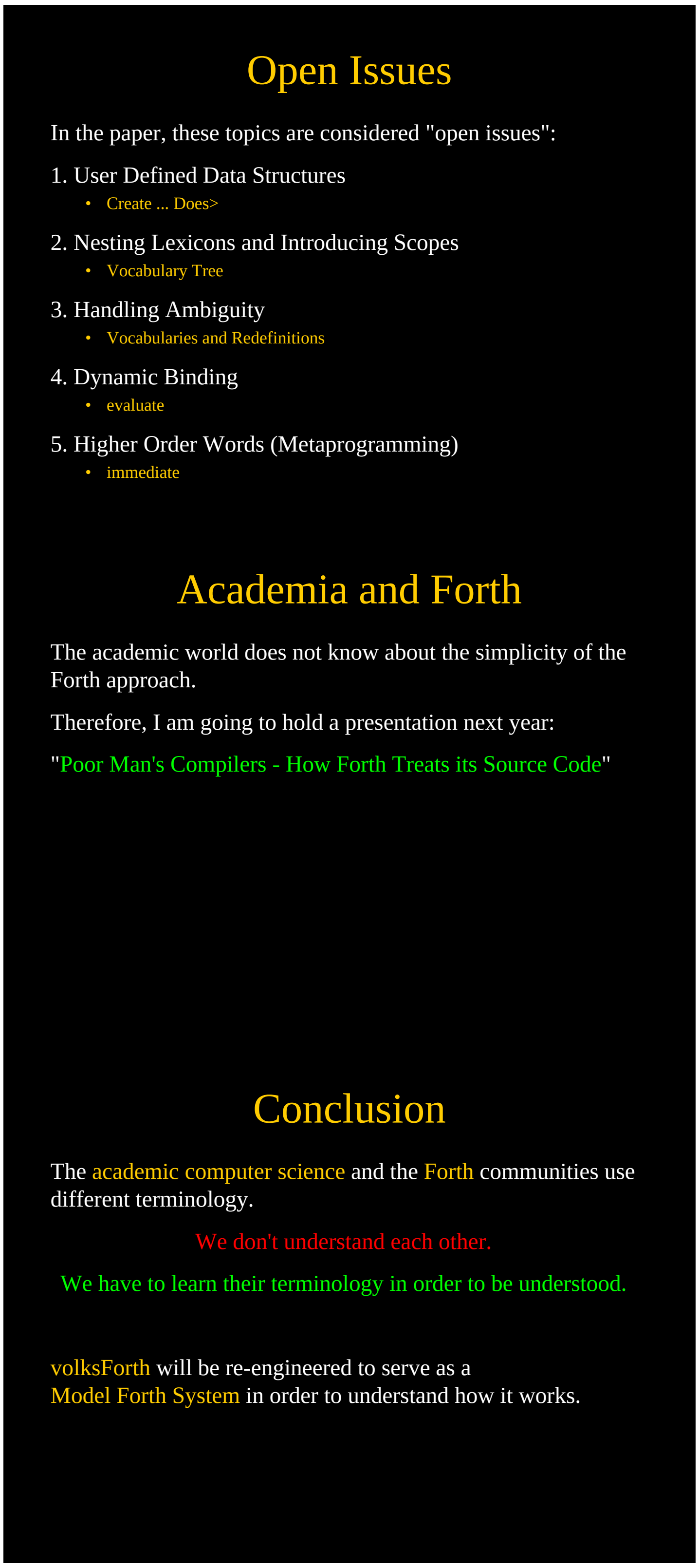
It seems that there is a growing discontent of conventional compiling strategies that use Phrase Structure Grammar, characterized by BNF specifications.

Instead Dependency Grammar based on a dictionary or lexicon is considered to be the more flexible approach.

It has been shown that both phrase structure grammar and dependency grammar cover the same set of linguistic constructs, namely context free grammars.

Scheme, Forth, Prolog, Smalltalk, APL, and LISP are examples of dependency grammar systems.

See: https://dl.acm.org/doi/10.1145/3133850.3133859



Open Issues

In the paper, these topics are considered "open issues":

1. User Defined Data Structures
   - Create ... Does>
2. Nesting Lexicons and Introducing Scopes
   - Vocabulary Tree
3. Handling Ambiguity
   - Vocabularies and Redefinitions
4. Dynamic Binding
   - evaluate
5. Higher Order Words (Metaprogramming)
   - immediate

Academia and Forth

The academic world does not know about the simplicity of the Forth approach.

Therefore, I am going to hold a presentation next year:

"Poor Man's Compilers - How Forth Treats its Source Code"

Conclusion

The academic computer science and the Forth communities use different terminology.

We don't understand each other.

We have to learn their terminology in order to be understood.

volksForth will be re-engineered to serve as a Model Forth System in order to understand how it works.

Gforth 1.0

Header & Recognizers & IDE & SWIG & MINΩΣ2

Bernd Paysan

EuroForth 2022, Video–Konferenz

IDE

Locate & Help & Where & Backtraces

Locate	Browse the source code
Help:	Browse the manual
Where	Show where words are used: nw/bw+cursors
Backtrace	Investigate a crash: bt/nt+cursors

New Headers 1

The Big Header Unification nt = xt = body

Name -c-4	Name comes first
flags+counts -4	Flags: up to 8 bits, count rest of the cell
Link Field -3	To next header
Code Field -2	Moved here
Name–HM -1	Header method table, see next page
Body 0	This is where the xt points to

SWIG

Generate C bindings automatically

%.i file:	Helps SWIG find & understand C files
%-fsi.c file:	Intermediate file, compiled with C
%.fsx file:	Generates Forth bindings
%.fs file:	Forth bindings, generate binding library
Outlook	Create complete binding libraries with reflections

New Header 2

Header Method Table

Link	Pointer to previous VTable
compile,	method to compile the word
to	method to apply IS or TO
defer@	method for DEFER@
extra	method for DOES> (or other extras)
name>int	convert name token to interpretation semantics
name>comp	convert name token to compilation semantics
name>string	convert name token to string (if any)
name>link	follow link field (if any)

MINΩΣ2

Lightweight GUI library

Classes:	actors, widgets, boxes, viewports & animations
Widgets:	glue, tile, frame, icon, image, text, edit, part-text canvas, (video)
Boxes:	hbox, vbox, zbox, slider, parbox, (grid)

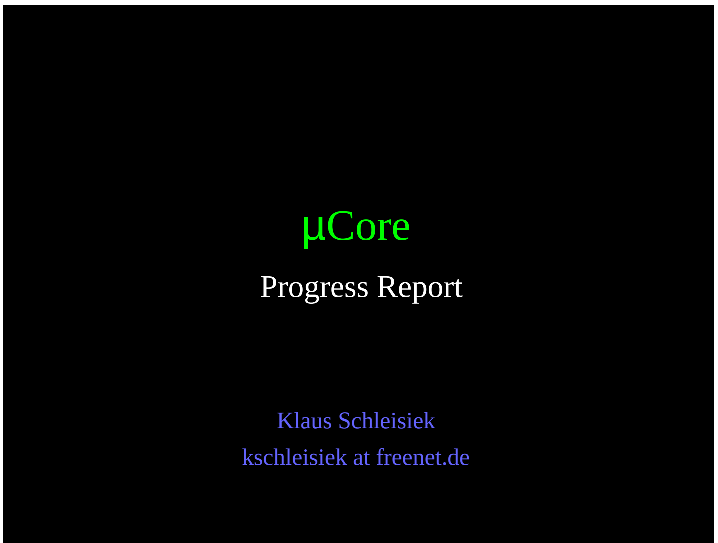
Recognizer

Minimalistic Core & Sequences & Unification

forth-recognize	Default–Recognizer as deferred Word
recognizer-sequence:	Sequence of recognizers
wordlists	are executable and recognizers
search order	is a rec-sequence:
translators	are executable, take (data -- ...)

Literatur & Links

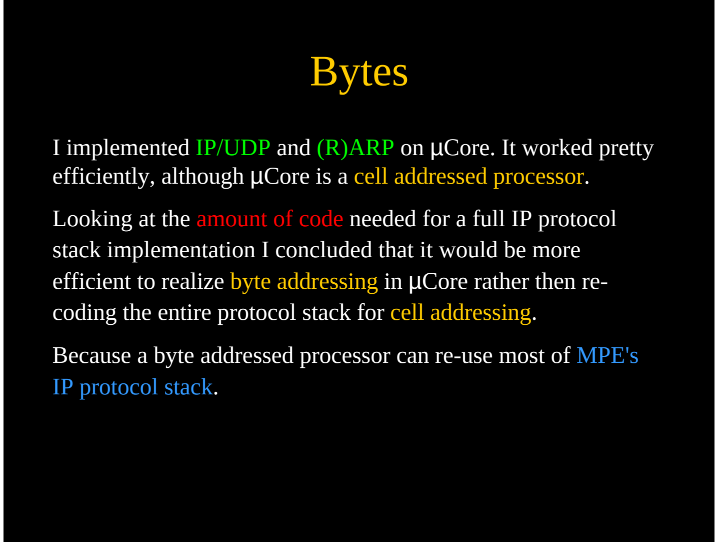
Gforth Team *Gforth Homepage*
https://gforth.org/



µCore

Progress Report

Klaus Schleisiek
kschleisiek at freenet.de

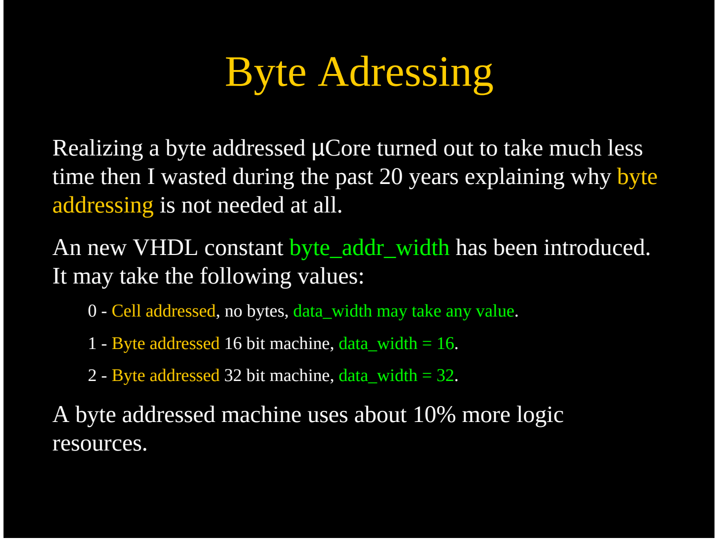


Bytes

I implemented IP/UDP and (R)ARP on µCore. It worked pretty efficiently, although µCore is a cell addressed processor.

Looking at the amount of code needed for a full IP protocol stack implementation I concluded that it would be more efficient to realize byte addressing in µCore rather then re-coding the entire protocol stack for cell addressing.

Because a byte addressed processor can re-use most of MPE's IP protocol stack.



Byte Adressing

Realizing a byte addressed µCore turned out to take much less time then I wasted during the past 20 years explaining why byte addressing is not needed at all.

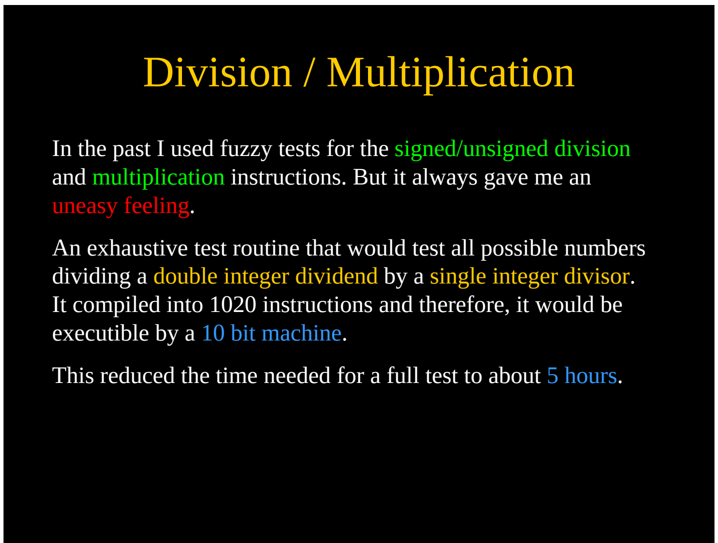
An new VHDL constant byte_addr_width has been introduced. It may take the following values:

0 - Cell addressed, no bytes, data_width may take any value.

1 - Byte addressed 16 bit machine, data_width = 16.

2 - Byte addressed 32 bit machine, data_width = 32.

A byte addressed machine uses about 10% more logic resources.

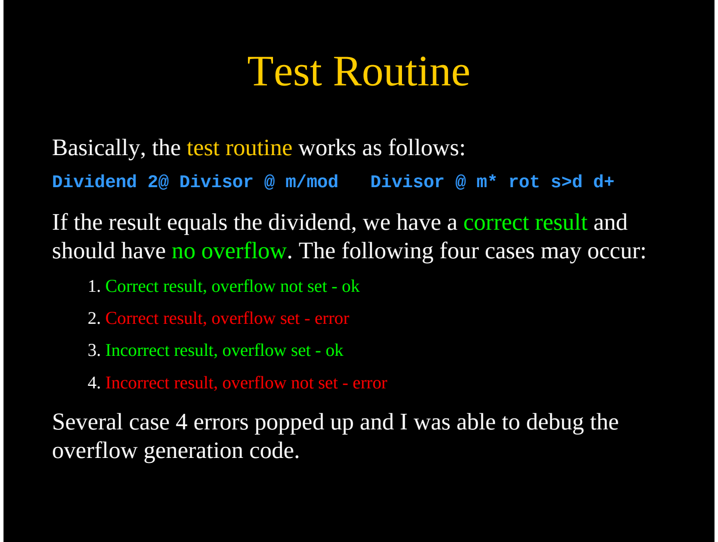


Division / Multiplication

In the past I used fuzzy tests for the signed/unsigned division and multiplication instructions. But it always gave me an uneasy feeling.

An exhaustive test routine that would test all possible numbers dividing a double integer dividend by a single integer divisor. It compiled into 1020 instructions and therefore, it would be executible by a 10 bit machine.

This reduced the time needed for a full test to about 5 hours.



Test Routine

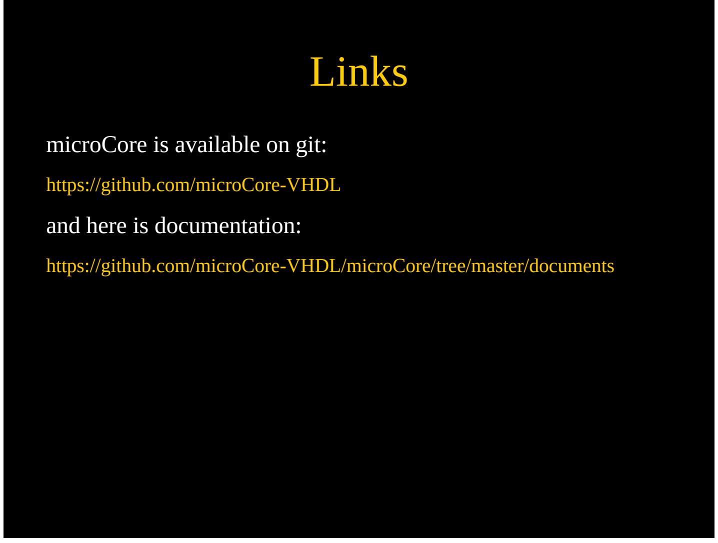
Basically, the test routine works as follows:

```
Dividend 2@ Divisor @ m/mod   Divisor @ m* rot s>d d+
```

If the result equals the dividend, we have a correct result and should have no overflow. The following four cases may occur:

1. Correct result, overflow not set - ok
2. Correct result, overflow set - error
3. Incorrect result, overflow set - ok
4. Incorrect result, overflow not set - error

Several case 4 errors popped up and I was able to debug the overflow generation code.



Links

microCore is available on git:

https://github.com/microCore-VHDL

and here is documentation:

https://github.com/microCore-VHDL/microCore/tree/master/documents

Forth200x Standard Report

Impromptu Talk

EuroForth'22 conference 2022-09

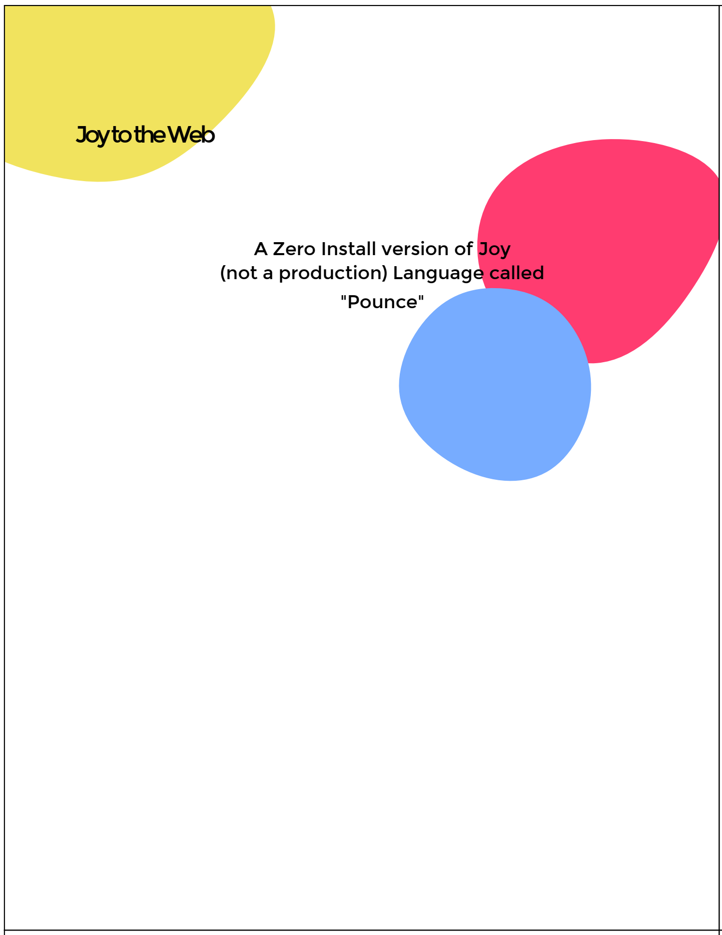
Ulrich Hoffmann

uho@

.de

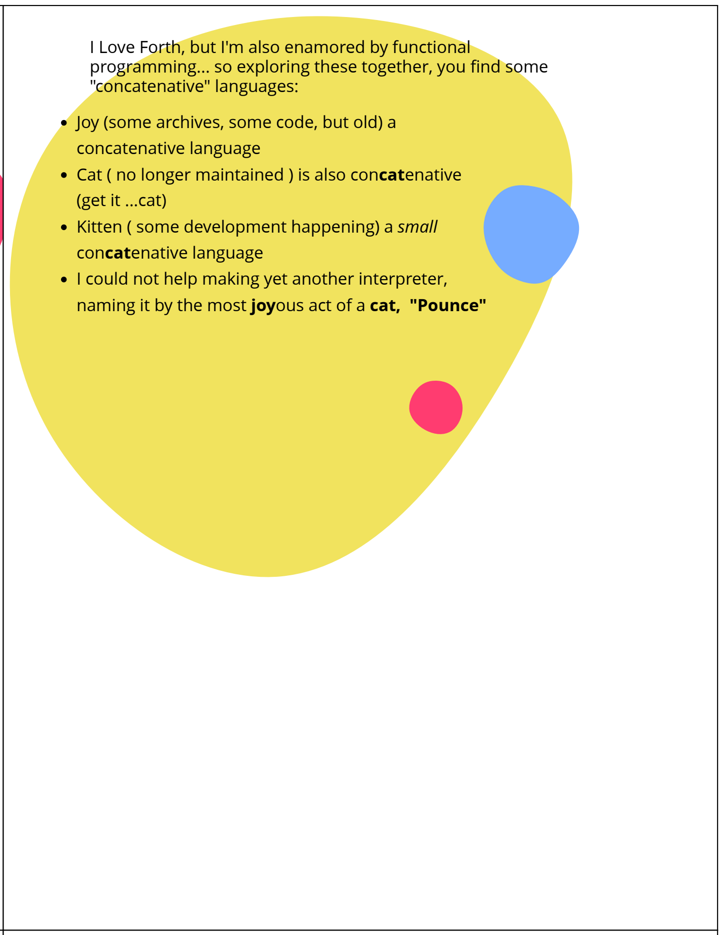
Forth200x Standard Report

- Meeting just before euroForth for 2 days
  - minutes at http://www.forth200x.org/meetings/2022-notes.html
- Call for participation your contribution is welcome, register below and/or contact me
- WEB sites forth200x.org, forth-standard.org
- Discussion Forum Mattermost at chat.forth-standard.org
- Involve the community new system for public voting
- Proposal: [160] minimalistic core API for recognizers
  https://forth-standard.org/proposals/minimalistic-core-api-for-recognizers#reply-892
- Next meetings: virtual on 2023-02-17, in person 2023-09-13 to 2023-09-15 please join



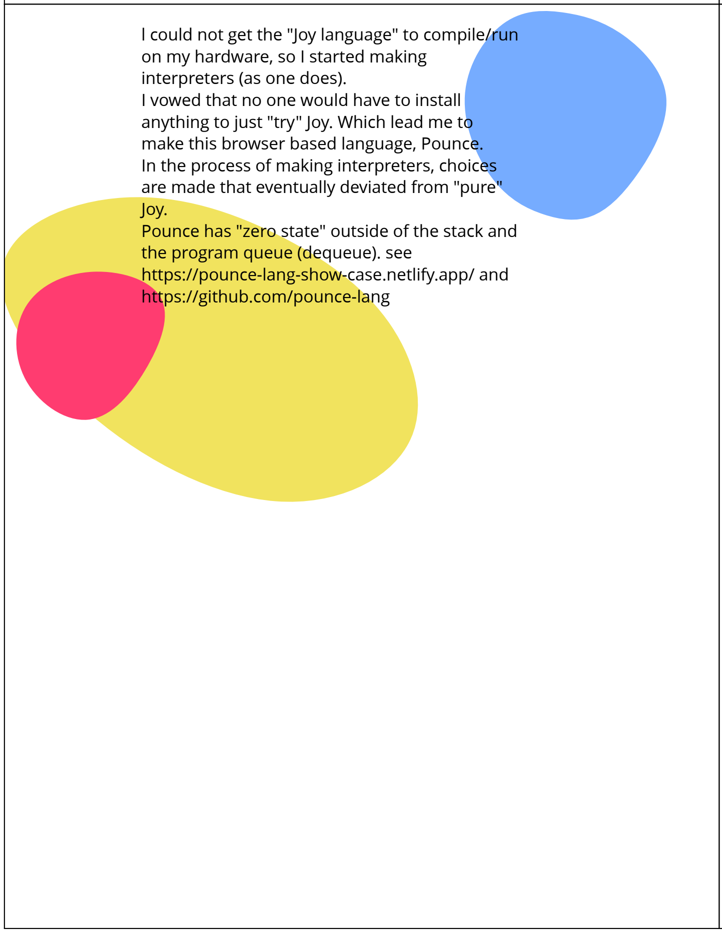
Joy to the Web

A Zero Install version of Joy
(not a production) Language called
"Pounce"

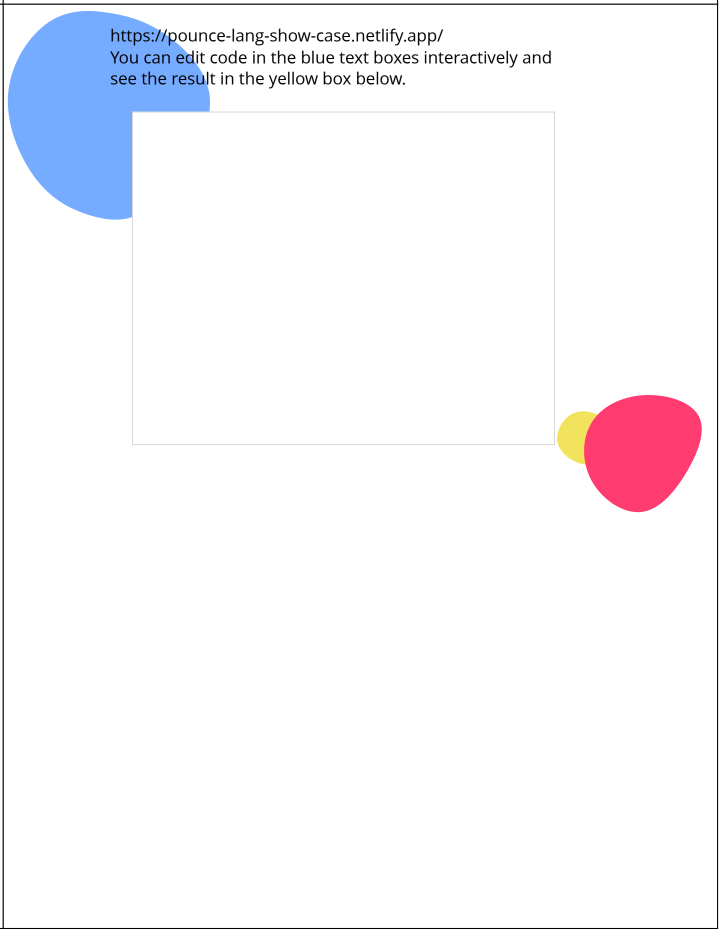


I Love Forth, but I'm also enamored by functional programming... so exploring these together, you find some "concatenative" languages:

- Joy (some archives, some code, but old) a concatenative language
- Cat (no longer maintained) is also con**cat**enative (get it ...cat)
- Kitten (some development happening) a *small* con**cat**enative language
- I could not help making yet another interpreter, naming it by the most **joy**ous act of a **cat**, **"Pounce"**



I could not get the "Joy language" to compile/run on my hardware, so I started making interpreters (as one does).
I vowed that no one would have to install anything to just "try" Joy. Which lead me to make this browser based language, Pounce.
In the process of making interpreters, choices are made that eventually deviated from "pure" Joy.
Pounce has "zero state" outside of the stack and the program queue (dequeue). see https://pounce-lang-show-case.netlify.app/ and https://github.com/pounce-lang



https://pounce-lang-show-case.netlify.app/
You can edit code in the blue text boxes interactively and see the result in the yellow box below.

https://pounce-lang-show-case.netlify.app/

https://github.com/pounce-lang

until next year

Are locals inevitably slow?

M. Anton Ertl, TU Wien

How to code `3dup`?

```
: 3dup.3 {: a b c :} a b c a b c ;
```

instr.	bytes	system
41	158	Gforth AMD64
16	44	iforth 5.0.27 (plus 20 bytes entry and return code)
7	19	lxf 1.6-982-823 32-bit
41	149	SwiftForth 3.11.0 32-bit (calls LSPACE)
26	92	VFX Forth 64 5.11 RC2

```
\ lxf code
mov     eax , [ebp]
mov     [ebp-Ch] , eax
mov     eax , [ebp+4h]
mov     [ebp-8h] , eax
mov     [ebp-4h] , ebx
lea     ebp , [ebp-Ch]
ret     near
```

`VICHECK` from Nick Nelson's "Better Values"

```
: VICHECK {: pindex paddr -- pindex' paddr :} \ Checks for valid index
\ paddr is the address of the data, the first cell of which contains the array size
    pindex 0 paddr @ WITHIN IF \ Index is valid
        pindex paddr
    ELSE \ Index is invalid
        \ code for reporting the error elided
    THEN ;

: VICHECKs ( pindex paddr -- pindex' paddr ) \ Checks for valid index
\ paddr is the address of the data, the first cell of which contains the array size
    over 0 2 pick @ WITHIN IF \ Index is valid
        \ the stack already contains the stuff
    ELSE \ Index is invalid
        \ code for reporting the error elided
    THEN ;
```

`VICHECK` from Nick Nelson's "Better Values"

instructions		bytes		system
locals	stack	locals	stack	
21	9	68	27	lxf 1.6-982-823 32-bit
22	5	78	19	VFX Forth 64 5.11 RC2

Discussion and Conclusion

- Are locals inevitably slow? No
- lxf is analytical about the return stack (including locals) but only in straight-line code
- C compilers have been register-allocating locals for decades
  Even on architectures like IA-32 with 8 registers

Counterarguments

- Locals are against the Forth spirit
- Locals are not used enough to justify optimizing them

Enums in Forth

Best Practices and Alternatives
Impromptu Talk

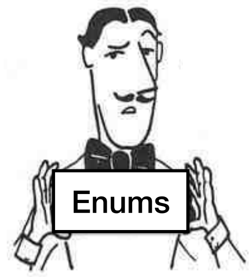
EuroForth'22 conference 2022-09

Ulrich Hoffmann

uho@ .de

Overview
Enums in Forth

- Enums in Forth
  - Explicit using *Forth Phrases™*
  - Nice Syntax - give names for phrases



Enums in Forth

- enumerations give names to values
- you don't remember all the numbers
- or they are likely to change
- or they are different on different systems

Nick Nelson:
"I don't like magic numbers."

- use CONSTANTs

Enums in Forth

- enumerations give Names to values - explicit with Constants

```
0 Constant black
1 Constant red
2 Constant green
3 Constant yellow

: .color ( c -- )
    dup black  = IF drop ." black"  EXIT THEN
    dup red    = IF drop ." red"    EXIT THEN
    dup green  = IF drop ." green"  EXIT THEN
    dup yellow = IF drop ." yellow" EXIT THEN
    ." color " . ;
```

Enums in Forth

- doing the calculations on your own

```
0
dup Constant black   1+ \ 0
dup Constant red     1+ \ 1
dup Constant green   1+ \ 2
dup Constant yellow  1+ \ 3
drop
```

- let the Forth interpreter do the calculation
- enum operations `dup` and `1+` are in different parts → combine them

```
dup Constant x    1+      →     dup 1+ swap Constant x
```

Enums in Forth

- doing the calculations on your own

```
0
dup 1+ swap Constant black   \ 0
dup 1+ swap Constant red     \ 1
dup 1+ swap Constant green   \ 2
dup 1+ swap Constant yellow  \ 3
drop
```

- explicit with Forth Phrases™
- a Forth Phrases is a sequence of inline forth words with no name
- Attention! Repeated phrases might be sign of bad factoring   **dup 1+ swap**
- factorization given a name to phrases   **over + swap -> bounds**

Enums in Forth

- name the calculation - use the name to do the calculation implicitly
- traditionally (math) names this *iota* (greek letter **ι**)

```
: iota ( x -- x+1 x )  dup 1+ swap ;           1 under+

0
iota Constant black    \ 0
iota Constant red      \ 1
iota Constant green    \ 2
iota Constant yellow   \ 3
drop
```

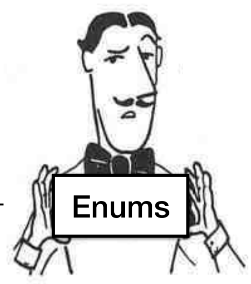
Enums in Forth

- name the calculation - use the name to do the calculation implicitly
- traditionally (math) names this *iota* (greek letter **ι**)

```
: ι ( x -- x+1 x )  dup 1+ swap ;              1 under+

0
ι Constant black   \ 0
ι Constant red     \ 1
ι Constant green   \ 2
ι Constant yellow  \ 3
drop
```

Enums in Forth

- Using a Defining word and capture Constant



```
: ι ( x -- x+1 x )  dup 1+ swap ;
: Enum ( n1 -- n2 )  ι Constant ;

0 Enum black    \ 0
  Enum red      \ 1
  Enum green    \ 2
  Enum yellow   \ 3
drop
```

: Enum (n1 -- n2) dup Constant 1+ ;
see SwiftForth

enum{ black, red, green, yellow };
see Vfx

Euroforth 2022

Resource Embedding

Euroforth 2022

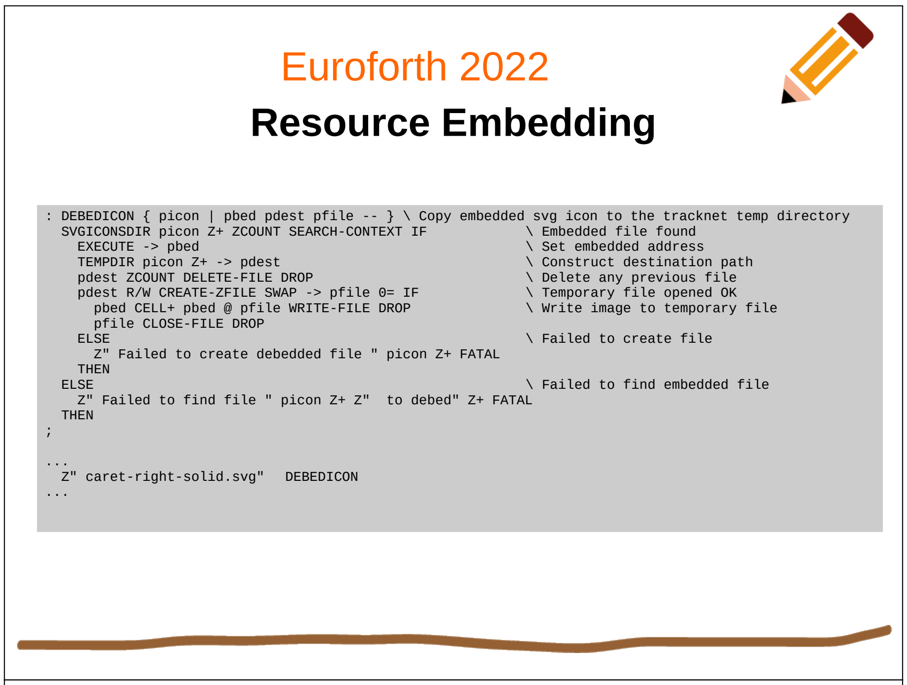
Resource Embedding

```
: EMBEDCOMP ( xt--- ) \ Compiler for children of EMBED
  >BODY @ POSTPONE LITERAL
;

: EMBED { pzpath | pfile plen ppos -- } \ Embed a file within the Forth directory
  pzpath R/O OPEN-ZFILE SWAP -> pfile 0= IF                          \ File opened OK
    pfile FILE-SIZE -ROT D>S -> plen 0= plen 0<> AND IF              \ File size ok and is non-zero
      plen CELL+ IRESERVE -> ppos                                    \ Get address of reserved space
      plen ppos !                                                    \ Save length
      ppos CELL+ plen pfile READ-FILE 0= SWAP plen = AND 0= IF       \ Read in OK
        CR Z" Cannot read file " pzpath Z+ Z"  to embed" Z+ Z$. ABORT
      THEN
    ELSE                                                        \ No file size
      CR Z" Failed to embed empty file " pzpath Z+ Z+ Z$. ABORT
    THEN
    pfile CLOSE-FILE DROP                                       \ Close file
  ELSE                                                          \ Cannot open file
    CR Z" Cannot open file " pzpath Z+ Z"  to embed" Z+ Z$. ABORT
  THEN
  CR Z" Embedding " pzpath z+ z$.
  pzpath ZCOUNT ($CREATE)                                       \ Create Forth word from file path
    ppos , ['] EMBEDCOMP set-compiler                           \ Returns the address of file length
  interp>
    @
;
```

Euroforth 2022

Resource Embedding

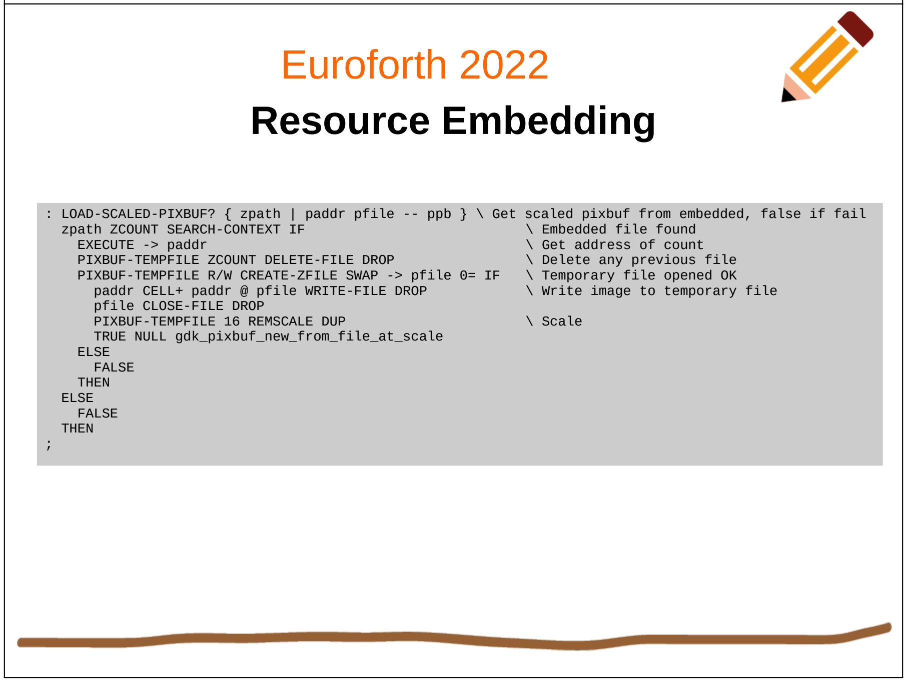
```
4 1 callproc: EMBEDDIRFUN { ppath pstat ptype pbuf -- n } \ Callback function from walk dir tree
  ptype 0= IF                                   \ Only files
    ppath ZCOUNT 1- + C@ [CHAR] ~ <> IF         \ No Glade backup files
      ppath EMBED                               \ Embed file
    THEN
  THEN
  FALSE                                         \ Continue
;

: EMBEDDIR { pzdir | pdir -- } \ Embed all files in directory
  pzdir EMBEDDIRFUN 0 0 nftw IF                                 \ Walk directory
    CR Z" Failed to embed files from directory " pzdir Z+ Z+ Z$. ABORT
  THEN
;

SVGICONSDIR   EMBEDDIR   \ Embed SVG icon files
```



Euroforth 2022

Resource Embedding

```
: DEBEDICON { picon | pbed pdest pfile -- } \ Copy embedded svg icon to the tracknet temp directory
  SVGICONSDIR picon Z+ ZCOUNT SEARCH-CONTEXT IF            \ Embedded file found
    EXECUTE -> pbed                                        \ Set embedded address
    TEMPDIR picon Z+ -> pdest                              \ Construct destination path
    pdest ZCOUNT DELETE-FILE DROP                          \ Delete any previous file
    pdest R/W CREATE-ZFILE SWAP -> pfile 0= IF             \ Temporary file opened OK
      pbed CELL+ pbed @ pfile WRITE-FILE DROP              \ Write image to temporary file
      pfile CLOSE-FILE DROP
    ELSE                                                   \ Failed to create file
      Z" Failed to create debedded file " picon Z+ FATAL
    THEN
  ELSE                                                     \ Failed to find embedded file
    Z" Failed to find file " picon Z+ Z"  to debed" Z+ FATAL
  THEN
;

...
  Z" caret-right-solid.svg"   DEBEDICON
...
```



Euroforth 2022

Resource Embedding

```
: LOAD-SCALED-PIXBUF? { zpath | paddr pfile -- ppb } \ Get scaled pixbuf from embedded, false if fail
  zpath ZCOUNT SEARCH-CONTEXT IF                           \ Embedded file found
    EXECUTE -> paddr                                       \ Get address of count
    PIXBUF-TEMPFILE ZCOUNT DELETE-FILE DROP                \ Delete any previous file
    PIXBUF-TEMPFILE R/W CREATE-ZFILE SWAP -> pfile 0= IF   \ Temporary file opened OK
      paddr CELL+ paddr @ pfile WRITE-FILE DROP            \ Write image to temporary file
      pfile CLOSE-FILE DROP
      PIXBUF-TEMPFILE 16 REMSCALE DUP                      \ Scale
      TRUE NULL gdk_pixbuf_new_from_file_at_scale
    ELSE
      FALSE
    THEN
  ELSE
    FALSE
  THEN
;
```

Encoding ASCII into cf2022 colorForth tokens

EuroForth 2022 Sep 18

Cakes and Biscuits

Flour plus :

Name	Fat	Egg(s)	Raising Agent	Topping
Pancake	0.5	1	0	Maple Syrup
Biscuit	0.5	0	Soda	Chocolate
Cake	1	2	Soda	Icing
Bread	0	0	Yeast	Butter
Jaffa Cake	0.25	0.5	Soda	Chocolate

Cakes and Biscuits

Computer plus :

Name	File	ASCII, Unicode	Raising Agent	Topping
C Source File	1	1, UTF-8	Pre-processor	Libraries
Word Docx	1	1, *	Templates	Styles
Forth Source	1	1, UTF-8	Forth	Source Database
PDF file	1	1, UTF-8	Forth-like	Zip, Signing
colorForth	0.01	0.01, UTF-8	Forth	Chocolate

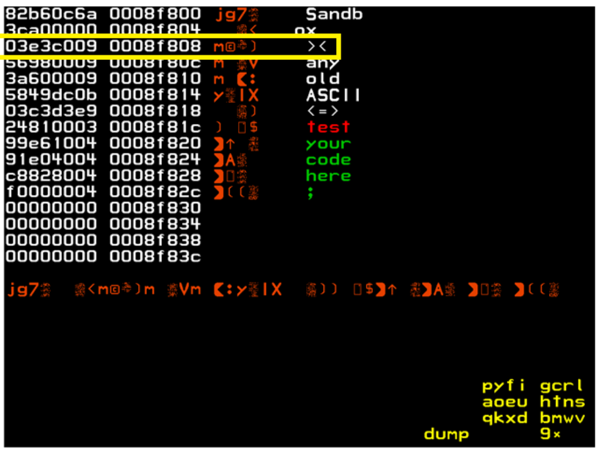
Layers

Choose your ingredients, add toppings.

Layers

Win32Forth	Native Forth	colorForth	JaffaCake Forth
Application	Application	Application	Application
File System	File System	Blocks	Blocks
OS	OS	-	OS/VM
Hardware	Hardware	Hardware	Hardware

```
; ASCII / UTF8 support. If the first Shannon-Fano encoded letter is a 4 bit NULL,
; display the next 24 bits as three ASCII characters.
; $03e3c009 is displayed as '><'
```



```
82b60c6a 0008f800 jg7▒      Sandb
3ca00000 0008f804   ▒<     ox
03e3c009 0008f808 m©▒)      ><
56980009 0008f80c m  ▒V     any
3a600009 0008f810 m ▒:      old
5849dc0b 0008f814 y▒IX      ASCII
03c3d3e9 0008f818   ▒)      <=>
24810003 0008f81c ) ▯$      test
99e61004 0008f820 ▒↑ ▒      your
91e04004 0008f824 ▒A▒       code
c8828004 0008f828 ▒▯▒       here
f0000004 0008f82c ▒((▒      ;
00000000 0008f830
00000000 0008f834
00000000 0008f838
00000000 0008f83c

jg7▒  ▒<m©▒)m ▒Vm ▒:y▒IX  ▒)) ▯$▒↑ ▒▒A▒ ▒▯▒ ▒((▒

                                        pyfi gcrl
                                        aoeu htns
                                        qkxd bmwv
                                   dump      9×
```

```
showShannonFano:    ; ( token -- ) \ display the Shannon-Fano encoded token on TOS
    ; ASCII / UTF8 support. If the first Shannon-Fano encoded letter is a 4 bit NULL,
    ; display the next 24 bits as three ASCII characters.
    mov _SCRATCH_, _TOS_            ; save the token value
    and _SCRATCH_, 0xF0000000
    cmp _SCRATCH_, 0x00000000
    jnz .forward
        ; display as three ASCII characters
        mov _SCRATCH_, _TOS_

        mov _TOS_, _SCRATCH_
        shr _TOS_, 20
        and _TOS_, 0x000000FF
        jz .null_terminator
            _DUP_
            call emit_

        mov _TOS_, _SCRATCH_
        shr _TOS_, 12
        and _TOS_, 0x000000FF
        jz .null_terminator
            _DUP_
            call emit_

        mov _TOS_, _SCRATCH_
        shr _TOS_, 4
        and _TOS_, 0x000000FF
        jz .null_terminator
            _DUP_
            call emit_

        ; arrive here if an ASCII character is an ASCII NULL, or if all three have been emitted
        .null_terminator:
        call space_                 ; display a space character at the end of the word
        _DROP_
        ret
```

Encoding ASCII into cf2022 colorForth tokens EuroForth 2022 Sep 18

```
lowercase:   ; display a white text word in normal lower-case letters
    call white
showSF_EDI_: ; ( -- ) \ display a Shanon-Fano encoded token pointed to by  edi  in the current colour
    _DUP_
    mov _TOS_, [ ( edi * 4 ) - 0x04 ]   ; fetch the next token - drops through to showShannonFano

call showShannonFano

    .forward:

    ; display as Shannon-Fano encoded token name
    and _TOS_, byte -0x10   ; and _TOS_, 0xFFFFFFF0 ignore token colour when displaying the letters

lowercasePrimitive: ; ( token -- ) \ display the given Shannon-Fano encoded word in the current colour
    call unpack
    jz lowercasePrimitiveEnd
    call emitSF_
    jmp lowercasePrimitive
lowercasePrimitiveEnd:
    call space_
    _DROP_
    _DROP_
    ret
```

Encoding ASCII into cf2022 colorForth tokens EuroForth 2022 Sep 18

This simple change to the colorForth pre-parsed source format, and the corresponding editor display words, allows the colorForth system to continue to use 32 bit „colored“ tokens, and also support ASCII / UTF-8 characters.

ToDo: Add keyboard support to type ASCII / UTF-8 characters !

Encoding ASCII into cf2022 colorForth tokens EuroForth 2022 Sep 18