Forth Recognizers in SwiftForth

And initial implementation of the proposed Forth Recognizers extension in SwiftForth

Background

- There is no standard method for extending the Forth interpreter’s handling of text tokens
- Most systems have hooks to extend parts of the interpreter
- The hooks are non-standard and differ from system to system
- The classic Forth interpreter handles Forth words (execute or compile) and numbers (push on stack or compile as literals)

Overview

The recognizer implementation divides the classic Forth interpreter into three blocks:

- *Interpreter*. Maintains STATE and organizes the work.
- *Token recognizer*. Called from the interpreter and analyzes each text token to see if it matches the criteria for a certain data type.
- *Handler*. Result of the parsing words is handed over to the interpreter with a pointer to the data-specific handling methods.

Handlers

There are three methods for each data type:

- Interpret
- Compile
- Postpone

The Interpret and Compile methods are called from within the interpreter loop based on STATE.

The Postpone method is called directly from POSTPONE.

Recognizers

The combination of a parsing word and the set of data handling words is called a *recognizer*.

There is no strict one-to-one relation between the parsing words and the data handling sets.

For example, the data handling set for single-cell numbers can be used by different parsing words.

Interpreter Loop

The simplified (and extensible) interpreter loop looks like this:

```
BEGIN ?STACK PARSE-NAME DUP WHILE
   RECSTACK RECOGNIZE
   STATE @ 2+ CELLS + @EXECUTE
REPEAT 2DROP
```

- For each token, we call RECOGNIZE with the system recognizer sequence RECSTACK
- The result of RECOGNIZE is an execution vector indexed by STATE

Interpreter Loop

The simplified (and extensible) interpreter loop looks like this:

```
BEGIN ?STACK PARSE-NAME DUP WHILE
   RECSTACK RECOGNIZE
   STATE @ 2+ CELLS + @EXECUTE
REPEAT 2DROP
```

- For each token, we call RECOGNIZE with the system recognizer sequence RECSTACK
- The result of RECOGNIZE is an execution vector indexed by STATE

Interpreter Loop

The simplified (and extensible) interpreter loop looks like this:

```
BEGIN ?STACK PARSE-NAME DUP WHILE
   RECSTACK RECOGNIZE
   STATE @ 2+ CELLS + @EXECUTE
REPEAT 2DROP
```

- For each token, we call RECOGNIZE with the system recognizer sequence RECSTACK
- The result of RECOGNIZE is an execution vector indexed by STATE

Token Recognizers

Each token recognizer has this stack effect:

```
REC-SOMETYPE ( c-addr len -- i*x addr1 | addr2 )
```

It takes the text token address and length (c-addr len) as inputs and if it recognizes the token, returns the address of the handler vector (addr1) along with any data required by the interpret, compile, or postpone behaviors in the handler vector.

Otherwise, returns addr2, the address of REC-NONE (the "unrecognized" handler).

Token Recognizers

Each token recognizer has this stack effect:

```
REC-SOMETYPE ( c-addr len -- i*x addr1 | addr2 )
```

It takes the text token address and length (c-addr len) as inputs and if it recognizes the token, returns the address of the handler vector (addr1) along with any data required by the interpret, compile, or postpone behaviors in the handler vector.

Otherwise, returns addr2, the address of REC-NONE (the “unrecognized” handler).

Token Recognizers

Each token recognizer has this stack effect:

```
REC-SOMETYPE ( c-addr len -- i*x addr1 | addr2 )
```

It takes the text token address and length (c-addr len) as inputs and if it recognizes the token, returns the address of the handler vector (addr1) along with any data required by the interpret, compile, or postpone behaviors in the handler vector.

Otherwise, returns addr2, the address of REC-NONE (the “unrecognized” handler).

Token Recognizers

Each token recognizer has this stack effect:

```
REC-SOMETYPE ( c-addr len -- i*x addr1 | addr2 )
```

It takes the text token address and length (c-addr len) as inputs and if it recognizes the token, returns the address of the handler vector (addr1) along with any data required by the interpret, compile, or postpone behaviors in the handler vector.

Otherwise, returns addr2, the address of REC-NONE (the "unrecognized" handler).

Postpone

Although the primary use of Recognizers in this implementation is in the interpreter loop, any string can be passed to a recognizer word.

This is also how POSTPONE is implemented.

```
: POSTPONE ( "name" -- )
   PARSE-NAME RECSTACK RECOGNIZE
   @EXECUTE ;  IMMEDIATE
```

Handler Vectors

The defining word RECTYPE: compiles the three-element vector table that handles a specific recognizer type ("rectype").

```
RECTYPE: ( xt1 xt2 xt3 "name" -- )
```

RECTYPE: defines a recognizer vector table and compiles the cells xt1, xt2, and xt3 in this order:

CELL OFFSET	VECTOR	ACTION
0	xt3	Postpone
1	xt2	Compile
2	xt1	Interpret

Recognizer Sequences

- A recognizer sequence is a “stack” of token recognizers the specifies
- The first cell of the sequence is the number of recognizers in the sequence
- Subsequent cells in the sequence are the execution tokens of the token recognizers
- The token recognizers in the sequence are executed in order until one of them does not return RECTYPE-NONE

Recognize

```
: RECOGNIZE ( c-addr len addr1 -- i*x addr2 )
   DUP @ 0 DO  CELL+  3DUP >R 2>R @EXECUTE
      DUP RECTYPE-NONE <> IF  2R> R> 3DROP
      UNLOOP EXIT  THEN DROP
   2R> R> LOOP  DROP RECTYPE-NONE ;
```

RECOGNIZE takes the token string c-addr len and the recognizer list addr1, runs through the list until it gets a hit (and exits the loop), or falls out the end of the list and returns RECTYPE-NONE.

Recognize

```
: RECOGNIZE ( c-addr len addr1 -- i*x addr2 )
   DUP @ 0 DO  CELL+  3DUP >R 2>R @EXECUTE
      DUP RECTYPE-NONE <> IF  2R> R> 3DROP
      UNLOOP EXIT  THEN DROP
   2R> R> LOOP  DROP RECTYPE-NONE ;
```

RECOGNIZE takes the token string c-addr len and the recognizer list addr1, runs through the list until it gets a hit (and exits the loop), or falls out the end of the list and returns RECTYPE-NONE.

Recognize

```
: RECOGNIZE ( c-addr len addr1 -- i*x addr2 )
   DUP @ 0 DO  CELL+  3DUP >R 2>R @EXECUTE
      DUP RECTYPE-NONE <> IF  2R> R> 3DROP
      UNLOOP EXIT  THEN DROP
   2R> R> LOOP  DROP RECTYPE-NONE ;
```

RECOGNIZE takes the token string c-addr len and the recognizer list addr1, runs through the list until it gets a hit (and exits the loop), or falls out the end of the list and returns RECTYPE-NONE.

Recognize

```
: RECOGNIZE ( c-addr len addr1 -- i*x addr2 )
   DUP @ 0 DO  CELL+  3DUP >R 2>R @EXECUTE
      DUP RECTYPE-NONE <> IF  2R> R> 3DROP
      UNLOOP EXIT  THEN DROP
   2R> R> LOOP  DROP RECTYPE-NONE ;
```

RECOGNIZE takes the token string c-addr len and the recognizer list addr1, runs through the list until it gets a hit (and exits the loop), or falls out the end of the list and returns RECTYPE-NONE.

Example: REC-FIND

- Recognizes Forth words in the current dictionary search order
- Note the cases for immediate and non-immediate words

```
' EXECUTE ' COMPILE, ' POSTPONE, RECTYPE: RECTYPE-WORD
' EXECUTE ' EXECUTE ' COMPILE, RECTYPE: RECTYPE-IMM

: REC-FIND ( c-addr len -- xt addr1 | addr2 )
  (FIND) CASE
     -1 OF  RECTYPE-WORD  ENDOF
     1 OF  RECTYPE-IMM  ENDOF
     0 OF  RECTYPE-NONE  ENDOF
  ENDCASE ;
```

Example: REC-FIND

- Recognizes Forth words in the current dictionary search order
- Note the cases for immediate and non-immediate words

```
' EXECUTE ' COMPILE, ' POSTPONE, RECTYPE: RECTYPE-WORD
' EXECUTE ' EXECUTE ' COMPILE, RECTYPE: RECTYPE-IMM

: REC-FIND ( c-addr len -- xt addr1 | addr2 )
   (FIND) CASE
      -1 OF  RECTYPE-WORD  ENDOF
      1 OF  RECTYPE-IMM  ENDOF
      0 OF  RECTYPE-NONE  ENDOF
  ENDCASE ;
```

Example: REC-FIND

- Recognizes Forth words in the current dictionary search order
- Note the cases for immediate and non-immediate words

```
' EXECUTE ' COMPILE, ' POSTPONE, RECTYPE: RECTYPE-WORD
' EXECUTE ' EXECUTE ' COMPILE, RECTYPE: RECTYPE-IMM

: REC-FIND ( c-addr len -- xt addr1 | addr2 )
   (FIND) CASE
      -1 OF  RECTYPE-WORD  ENDOF
      1 OF  RECTYPE-IMM  ENDOF
      0 OF  RECTYPE-NONE  ENDOF
  ENDCASE ;
```

Example: REC-FIND

- Recognizes Forth words in the current dictionary search order
- Note the cases for immediate and non-immediate words

```
' EXECUTE ' COMPILE, ' POSTPONE, RECTYPE: RECTYPE-WORD
' EXECUTE ' EXECUTE ' COMPILE, RECTYPE: RECTYPE-IMM

: REC-FIND ( c-addr len -- xt addr1 | addr2 )
   (FIND) CASE
      -1 OF  RECTYPE-WORD  ENDOF
      1 OF  RECTYPE-IMM  ENDOF
      0 OF  RECTYPE-NONE  ENDOF
  ENDCASE ;
```

Example: REC-NUM

- Recognizes single- and double-cell numbers

```
' DROP ' EXECUTE ' NOPOST RECTYPE: RECTYPE-NUM

: REC-NUM ( c-addr len -- i*x xt addr1 | addr2 )
   ANY-NUMBER? CASE
      0 OF  RECTYPE-NONE  ENDOF
      1 OF  ['] LITERAL RECTYPE-NUM  ENDOF
      2 OF  ['] 2LITERAL RECTYPE-NUM  ENDOF
   ENDCASE ;
```

Example: REC-NUM

- Recognizes single- and double-cell numbers

```
' DROP ' EXECUTE ' NOPOST RECTYPE: RECTYPE-NUM

: REC-NUM ( c-addr len -- i*x xt addr1 | addr2 )
  ANY-NUMBER? CASE
     0 OF  RECTYPE-NONE  ENDOF
     1 OF  ['] LITERAL RECTYPE-NUM  ENDOF
     2 OF  ['] 2LITERAL RECTYPE-NUM  ENDOF
  ENDCASE ;
```

Example: REC-NUM

- Recognizes single- and double-cell numbers

```
' DROP ' EXECUTE ' NOPOST RECTYPE: RECTYPE-NUM

: REC-NUM ( c-addr len -- i*x xt addr1 | addr2 )
   ANY-NUMBER? CASE
      0 OF  RECTYPE-NONE  ENDOF
      1 OF  ['] LITERAL RECTYPE-NUM  ENDOF
      2 OF  ['] 2LITERAL RECTYPE-NUM  ENDOF
   ENDCASE ;
```

Sequence Operations

- Set and get the current recognizer sequence

```
SET-RECOGNIZERS ( xtn ... xt1 n -- )
GET-RECOGNIZERS ( -- xtn ... xt1 n )
```

- Append and delete a single recognizer

```
+RECOGNIZER ( xt -- )
-RECOGNIZER ( -- )
```

www.forth.com/recognizers