Program structures#

Table of Contents

Wil Baden, which you encounter in the English literature often has, in Escaping his contribution FORTH stated the following: There are four types of control statements:

- The sequence of instructions
- The selection of program parts,
- The repetition of instructions and parts of the program,
- The demolition.

The first three possibilities are absolutely necessary and in the older languages such as PASCAL exclusively available. According to a statement is in the people-FORTH for the selection of program parts are available, the execution is the result of a logical expression subject:

```
IF flag THEN <Anweisungen>
IF flag THEN <Anweisungen> ELSE <Anweisungen>
```

Where, however, the program made a return for instructions wiederho1t run is used for a given number of times this statement, with the current index of I and J is available:

```
<Grenzen> DO /? <Anweisungen> DO LOOP
<Grenzen> DO /? <Anweisungen> <Schrittweite> DO + LOOP
```

If a repetition of instructions to be executed without the number of runs known 1st, it is an index variable board or otherwise to the result of a logical expression to come. The following construction provides an infinite loop:

```
BEGIN REPEAT <Anweisungen>
```

The repetition of statements are so far balanced that a statement is so long (while running), such an expression is true, or a statement is repeated, right up to (until) one expression.

```
BEGIN UNTIL <Anweisungen> flag
<Anweisungen> Flag BEGIN WHILE REPEAT <Anweisungen>
```

Both possibilities can in populous FORTH combine, including several (multiple) WHILE may occur in a control statement.

```
BEGIN WHILE <Anweisungen> <Anweisungen> flag flag UNTIL
```

Now occurs in applications often the case that a control instruction to be abandoned, because something has happened.

Then the fourth situation, the demolition, given the programming language "C" provides for the functions:*break*,*continue*,*return*and*exit*available, state-FORTH offers *exit*,*leave*,*endloop*,*quit*,*abort*,*abort "abort*and*(*to. In FORTH EXIT to used to leave the definition, in which it appears, however, the smallest enclosing leaves LEAVE DO ... LOOP.

Glossary#

As of version 3.81.8 has state-FORTH on additional control instruction for the compiler, conditional compilation in the form:

```
<word> have not. <action1> IF. <action2> ELSE. THEN
```

DieseWorte be used outside of colon definitions and replace the\needs of earlier versions.

- have
- exit
- ?exit
- 0=exit
- if
- .IF
- then
- .THEN
- else
- .ELSE
- do
- ?do
- loop
- +loop
- I
- J
- leave
- endloop
- bounds
- begin
- repeat
- until
- while
- execute
- perform
- case?
- stop

Words for error handling#

They work well as control statements, as the definitions of*ARGUMENTS*and*IS*DEPTH::

```
is-depth (n -)
  depth 1 - - abort "wrong number of parameters!" ;
```

IS-DEPTH review the stack on a given number of stack elements (depth point).

- Abort
- 'Abort
- Abort "
- Error

- ErrorHandler
- (Error
- R #
- Scr
- Quit
- ? Pairs

Fallunterscheidung in FORTH#

With IF ELSE structuring THEN / ENDIF

It is worth briefly the various possibilities are shown, which can merde made a case distinction in FORTH. Characteristic of such a program situation is that just from a different possibilities of the program flow to be chosen.

Starting from a clear problem, a game, are described based on the necessary definitions and the development of the above-described control structure.

An example is a game with simple rules:

This drinking game, which according to the article "Ultimate CASE Statement" (Fourth Dimension 2 / 87, page 40 ff) also called CRAPS is, it is about to distribute a supply of jars filled with the players with the help of the cube and leerzutrinken:

- When ONE was taken a glass out of the stock in the middle of the table and placed before him.
- For a TWO or THREE got the neighbor / neighbor left zugesehoben a glass of its own stock.
- If the FOUR or FIVE was the neighbor about the neighbor on the right set before a glass of its own stock.
- At a SIX, all glasses, the gamer had emptied before him.

Assignment is: 1 = accept, 2 / 3 = left, 4 / 5 = right, drink 6 = and according to the number of the cube is one of six possible actions are executed. The program should be limited to, read and evaluate the outcome of dice. A message is issued to perform which of the six acts.

For such a program a number entry is required. This was realized here with the word-F83 NUMBERS:

```
: F83-number? ( string -- d f )
number? ?dup
IF
0< IF extend THEN
true exit
THEN
drop 0 0 false ;

: input# ( string -- n )
pad c/l 1- >expect
pad F83-number? 2drop ;
```

The definition of the words that are to carry six above-mentioned actions symbolically depends on the rules that dictate exactly one result for each cube action:

```
\ Take drink push left right

: Take bright. "Take a glass of" normal two spaces;
: Drink bright. "Drink all the glasses" normal two spaces;
```

```
spaces: left bright. "a glass to the left" normal 2;
spaces: the right bright. "a glass to the right" normal 2;

: Slide;
```

PUSHINGis a dummy procedure, a filler, the necessity arises only very late. For dialogue with countries will deflniert users:

```
: Cr request. "If you take drink or move?"
             cr. "Please your eyes and <cr> number:";

: Congratulations cr. "Good luck on the next roll ..." ;
```

The word*RESULTS*is to perform in accordance with a selector just one of 6 possible procedures. So we will examine whether this or this or ... the Mogiichkeiten comes into question. Add to that the test whether the passed parameter was between (between) 1 and 6.

The Def nit! On is of*BETWEEN*according to state-FORTH-quite short:

```
(Lower limit value ceiling - false or)
(- True if lower <<= value = upper limit)
: Between 1 + uwithin;

: Auswertung.1 (draft results -)
  dup 1 = IF ELSE take
          dup IF 2 = move left ELSE
          dup = 3 ELSE IF one left
                  dup 4 ELSE IF the right move
                  dup 5 ELSE IF the right move
                        dup 6 = IF THEN drink
                        THEN
                        THEN
                  THEN
                  THEN
          THEN
  1 6 between IF not inversely. "Fraud!" THEN normal;
```

Since such a test for equality in the programming practice is often the case, which provides for the word volks4TH case? available. case? compares the top two stack values together. When inequality is the test value (selector) received, so that the words of DUP and = are replaced by it.

```
: Auswertung.3 (throwing outcome -)

   A case? IF THEN take exit
   2 case? IF shift left exit THEN
   3 case? IF shift left exit THEN
   4 case? IF THEN exit right move
   5 case? IF THEN exit right move
   6 case? IF THEN drink exit

   drop inversely. "Fraud!" normal;
```

In this analysis from the QueHtext clear enough that in TWO THREE and the same action is executed, as well as FOUR and FIVE will have the same action to follow.

OR = n2 examine why a test value for equality with a lying under a flag f1 number n2. The result of this test is already available with the Flag OR-linked. This new flag f2 and pass the test value will nl:

OR = definition in Forth

```
  : = Or (f1 n1 n2 - n1 f2)
    2 pick
    = Or;
```

OR = definition in 8086 assembler

```
   code 0or (f1 n1 n2 - n1 f2)
      A D D xchg pop
      S W mov
        W) A cmp
        0 =?] [-1 Mov # D?
      next
   end-code
```

This word brings a significant improvement in the code:

```
: Auswertung.4 (throwing outcome -)
  dup
  1 6 between IF
              dup 1 = IF THEN take
              dup 2 = 3 = or IF THEN push left
              dup 4 = 5 = or IF THEN push right
              dup 6 = IF THEN drink
            ELSE
              inverse. "Fraud!" normal
            THEN
   drop;
```

This created a case statement without a very clear control of the program flow. The Plausibilitatsprüfung whether the number entered 1-6 was, is here moved to the beginning and is processed in a single ELSE branch.

Treatment of a CASE - Situation

Structural CASE

Many programming languages provide a CASE statement is available that evaluates as in PASCAL using a case-by-case indices a list of constants and executes a corresponding statement. Although such a CASE construct - as shown above - is not necessary, it makes programs easier to read and is in problems such as the assessment of a given index actually closer. This is in "Wil Baden - Ultimate CASE Statement - VD 2 / 87, p.40 ff" has been discussed in detail, but with the older Esker CASE (Dr. Charles Eaker - Just in CASE (DIM FORTH II / 3)) by Dr. Charles Eaker certainly the better known, is the often found in the literature and source code reference and use.

Mr. H. Sehnitter has implemented this Eaker-CASE for the volks4TH and it made changes in the structure and improvements in the application.

```
\ Caselist initlist> marklist> resolvlist

| Variable caselist
|: Initlist (list - addr)
    dup @ swap off;
```

```
|:> Marklist (list -)
    here over @ swap! ;

|:> Resolvelist (addr-list)
    BEGIN dup @
    Dup dup dup WHILE @ @ red! > Resolve
    REPEAT! ;

\ Case elsecase endcase

: CASE caselist initlist 4; immediate restrict
: ELSECASE 4? Pairs compile drop 6; immediate restrict
: ENDCASE dup 4 =
                IF compile drop drop
                6 ELSE? Pairs
                THEN caselist> resolvelist
  ; Immediate restrict

\ Of EndOf

: OF 4? Pairs compile over
                  compile =
                  compile? branch> mark
                  compile drop 5; immediate restrict

: EndOf 5? Pairs compile branch caselist> marklist> resolve 4; immediate restrict
```

This implementation of the Eaker-CASE is an improvement over the original by Mr. Reaper has extended the control structure to ELSECASE. Seibstverständlich the new version is fully upward compatible with the original version.

Verbesserung:

In the original version of the CASE structure, it is not possible to place between the last and EndOf ENDCASE to a value or a flag on the stack, as ENDCASE principle the "Top of Stack" remote.

In the improved version ELSECASE cleans the stack. ELSECASE must not be invoked, in which case compiled ENDCASE as before a DROP. It is now possible to place between the words and ELSECASE ENDCASE - as well as between OF and EndOf - a value on the stack and to use these outside the CASE control structure.

Änderung:

The forward references are not resolved on the stack, but on a linked list. The variable **caselist** contains the start address for unknown transfer addresses. The nesting of several CASE structures is arbitrary, and solved by **initlist**. **> Marklist** fills compile the list of forward references and **> resolvelist** solve them again.

Anwendungshinweis:

If these definitions are loaded outside the compilation of the work system should be removed with the | as headerless selected words with **clear** after compiling the names.

The example of a Tastatuabfrage on CTRL-key shows (MS-DOS), is how to use this CASE construct. It is important that the **OF** even checks the equality of the two present values and executes the instructions in this case between **OF** and **ENDOF**.

```
: Control bl word 1+ c@ $BF and state @ IF [compile] Literal THEN : immediate

: Tastaurabfrage
." exit mot ctrl x" cr
BEGIN key
CASE control A OF ." action ^a " cr false ENDOF
control B OF ." action ^b " cr false ENDOF
control C OF ." action ^c " cr false ENDOF
control D OF ." action ^d " cr false ENDOF
control X OF ." exit 2" true ENDOF
ELSECASE
." befehl unbekannt " CR false
ENDCASE
UNTIL ;
```

This CASE statement can be the assignment of the six ways to write the six directions, as in Pascal, for example 0 .255 only areas as case constants not allowed.

```
: Auswertung.5 (throwing outcome -)
   CASE
      1 OF EndOf take
      2 OF shift left EndOf
      3 OF shift left EndOf
      4 OF slide right EndOf
      5 OF slide right EndOf
      6 OF EndOf drink
   ELSECASE
      inverse. "Fraud!" normal
   ENDCASE;
```

The complete program can be written, with the typical three-part "input-processing-output" is clear:

```
: Craps (-)
  cr cr request
     input #
     Evaluation
  Congratulations cr
;
```

Wil Baden has "Ultimate CASE Statement stated" that a CASE statement is just syntactic sugar for a program and, ultimately, is nothing more than compiling a nested IF ... THEN statement. Such Implemetation for which was written by Mr. Klaus volksFORTH83 Schleisiek:

```
\ CASE OF EndOf ENDCASE

: CASE (n1 - n1 n1) dup;
: OF [compile] compile IF drop; immediate restrict
: Restrict EndOf [compile] ELSE 4 +; immediate
: ENDCASE compile drop BEGIN 3 case? WHILE> resolve REPEAT; immediate restrict
```

Wil Baden's implementation adheres closely to the logical foundations, with the difference to Eaker-CASE main object is that here runs every TRUE flag the statement part between OF and EndOf, the original version did not conduct any testing for equality, but any expression can lead to a flag that is then evaluated by OF. Thus, the evaluation of the index case is more variable than the Eaker-CASE:

```
: Auswertung.6 (throws litter -)
  dup
     Not between 1 6
         IF inversely. "Fraud!" normal drop exit THEN
  CASE OF 1 = take EndOf
  CASE OF 6 = drink EndOf
  CASE 4 <OF shift left EndOf
  CASE 3> OF slide right EndOf
  ENDCASE;
```

Here, in this design, the Plausibilltätsprüfung is quite ahead to reach the ELSECASE case by one of the word EXIT. If none of the words from the drop-down list above, can be achieved with a different solution BREAK:

```
: BREAK compile exit
          [Compile] THEN; immediate restrict
```

The fact that BREAK is an EXIT from the word, an (implicit) ELSECASE is accomplished by clutching the instructions of the selection list OF and BREAK and lists the instructions for the case after ENDCASE ELSE:

```
: Auswertung.7 (cube number -)
  1 = CASE OF BREAK take
  CASE 2 = 3 = push left or OF BREAK
  CASE OF 4 = 5 = or slide right BREAK
  CASE OF 6 = drink BREAK
  ENDCASE inversely. "Fraud!" normal;
```

Positional CASE #

A completely different approach offers a positional CASE construct, in which case the distinction by the case-index is done in table form.

In previous solutions are still a number of comparisons between a case-index and a list of case-constants were made, now is the case-index itself is used to select the desired procedure. The use of the index case as selector also has advantages in the period since the comparisons eliminated.

If FORTH-words are to be stored in tables, there is the problem that a FORTH-word in his call usually runs the compiled words. When a table is not desired, and there is reasonably required that the starting address of the table is given in order to use the case index as an offset in this table.

This can in populous FORTH either in the traditional way of solving] and No InterWiki reference defined in properties for Wiki called "or the state-FORTH-specific __Create"! Take one left Push the right drink No InterWiki reference defined in properties for Wiki called ""! Does>; :: DOES> @ 0 = last abortion "without reference" (Does> corrent @ context! Hide 0]; }}}

This word **: DOES>** points to the last word on **Create** defined a runtime part. This word was programmed by Klaus Schleisiek also applies to point out after compiling to remove this with | headerless as declared by word **clear**.

```
Create: Auswertung.8
          take
            shift left
            right move
          ; Drink
```

```
: DOES move> right;
```

Without: DOES> are the table and access procedures separate words:

```
: CRAPS1
  cr cr request
    input #
    Glass slide properly
  Congratulations cr;
```

If one decides against it, to define both table and access procedure in a word, the result is the familiar appearance:

```
: CRAPS
  cr cr request
     input #
     Evaluation
  Congratulations cr;
```

For more frequent use of such tables are, the use of "positional CASE defining words to". Once again, the first volks4TH-compliant solution, then the traditional version:

```
: Case: (-)
   Create: Does> (pfa -) swap 2 + * perform;

\ Alternative definition for CASE:
: Case:
   : Does> (pfa -) swap 2 + * perform;
```

A very elegant way to handle the error handling in case of an implausible case-index has the word **Associative:** . This word **Associative:** searches a table for a match between a numerical value on the stack and the numerical values in the table and returns the index of the found number (match back). In case of failure (mismatch), the maximum index +1 (out of range = maxIndex + 1) is passed:

```
: Associative: (n -)
  Does Constant> (n - index)
  dup @ rot
  dup @ 0
  DO 2 + @ = 2dup
     IF 2drop drop I 0 0 LEAVE THEN
  LOOP 2drop;

Associative 6: Evaluate

            1,
          2, 3,
          4, 5,
            6,

      Case: Handler \ consists

            take
      links links
      right right
```

```
           drink
    ; Scold
```

Instead of Primitivabsicherung on MIN and MAX is an "out of range" error handling named **schimpfen** carried out on the table heading maxIndex +1.

Purpose #

This last part of the discussion of the ways to handle a situation CASE attacks, suggestions from the literature (E. Flögel, FORTH Guide (p. 109) and W. Waigaard, menus in FORTH, electronics, 9 / 88 (P. 109 ff)) to.

These two words are defined:

- CLS - clears the screen and
- CELLS - makes the calculation of the table significant access

```
: Cls full page;
: 2 * cells;
```

The content and structure of the table remain unchanged, only the treatment of an "out of range" situation is achieved with time and **min max** and twice entering the error routine **schimpfen**.

```
Create: action
          scold take links links
                      right to drink quite scold;

\ The execution of a list Flögel 7 / 86
: Select (addr n - * cfa) 2 arguments
   swap 0 max \ out of range MIN
        7 min \ out of range MAX
        cells +;

: Evaluate (n -) 1 arguments
   Select perform action;

:. All (-)
   8 0 DO cr I dup. Evaluating two spaces LOOP;
```

AUSWÄHLEN Passes at a given vector and a given index a pointer to the "code field address (cfa) of the corresponding word. **AUSWERTEN** Performs the so-selected word and **.ALL** only served as a control. Such a word-based data structures is on the screen should be in the development of a program always be there.

Another M5glichkeit, values entered into a vector, has represented Mr. Flögel in his book "FORTH Manual":

```
Create Table 8 cells allot
       : DOES> (i - addr) swap cells +;

'Scold 0 Table!
'Take a table!
'Links dup 2 Table!
             3 Table!
'Right dup 4 Table!
             5 Table!
```

```
'Drink 6 Table!
'Scold 7 Table!

: Evaluate (i -) 7 min 0 max table perform;

:. Action (i -)
   Table @> name bright. Name normal;

:. Table (-) 8 0 DO cr cr I. Action LOOP;
```

Here is with **.ACTION .TABELLE** and the possibility of the vector can be represented. Similarly, in the command-line editing CED entered the new actions in the input vectors.

A slight modification of "W. Waigaard Menus, in FORTH" should show the VerknQpfung a vector of words and a menu option:

```
Create function
    ] Noop noop noop noop
      noop noop noop noop [
    : DOES> (i - addr)
      7 min 0 max swap cells +;
```

FUNCTION is a vector execution, which is preset with **NOOP**. At runtime, it returns the address of the indexed element.

```
:. Action (i addr -)
  @> Name bright. Name normal;
```

.WORD Displays the name of a word, the CFA was entered into an address.

```
: Option (i -)
  R>
  dup 2 + R \ i * w.addr
  @ \ I w.addr
  stash swap function! \ I w.addr i addr
  function. action; \ i addr
```

option fetches the address of the **option** on the following word. The word should not be executed, but the following. Only the pointer on the word to be evaluated. After the passed index pointer is entered in **FUNCTION**. The name of the so-entered word is displayed!

```
\ Menu jrp 06feb89
: Menu
   option 0 scold
   take option 1
   2 option left
   3 option on the left
   4 option right
   5 option right
   option 6 drink
   7 option scold;
```

If the word **MENUE** is called, not only the options listed in the table, but also by name displayed on the screen. This technique lends itself to a menu bar at a fixed screen position, similar to the status bar of the state-FORTH. To change those menu items to offer the keys:

```
: Fkey (-)
  & 58 key function + abs perform;
```

FKEY supplies the pressure of a button returns a value from -59 to -68. This is for 10 function in the range -1 to -10 scale and made the absolute value.

Recursion #

Before the technique is shown to the recursion for the state-FORTH, is another word **.LASTNAME** show that the word is **LAST** with the literature often encountered **LATEST** identical: Both words provide the "name field address" (nfa) of the previously defined CURRENT-word in vocabulary. The word **LAST'** contrast, provides the "code field address (cfa) of the last defined word.

```
: Last name. @ Last. Name;
```

Recursion is a technique in which a word is repeatedly calling itself. One of the most famous examples is the calculation of the factorial of a positive integer n this is clear! from the product of all its predecessors.

In the state-FORTH Selbstaufruf a word is characterized by **RECURSIVE**, so that presents a program for calculating faculty as follows:

```
: Faculty (+ n - n!)
   recursive
   dup 0 <IF drop. "no negative numbers!" exit
          THEN
   ? Dup 0 = IF 1 \ special case 0
            ELSE dup 1 - faculty *
            THEN;

 4 cr faculty.
 faculty cr 5th
 faculty cr 6th
```

However, there is - especially in the fig-FORTH literature - a word **MYSELF**, which is identical to that encountered in FORTH83 **RECURSE** environments. Since this construction is used in MYSELF / RECURSE Platshalter as for the word name is often used, the possible definitions and a further form of FACULTY be displayed:

```
: @ Myself last name>; immediate

: Myself last '; immediate

: Myself recurse [compile]; immediate

'Myself Alias recurse immediate

: Faculty (+ n - n!)
  dup 0 <IF. "does not allow negative numbers!"
          ELSE? Dup 0 = IF 1
                        ELSE dup 1 - myself *
                        THEN
```

```
          THEN;
```

When using **RECURSE** is simply replaced by **MYSELF**:

```
...
          ELSE? Dup 0 = IF 1
                        ELSE dup 1 - recurse *
                        THEN
          THEN;
...
```