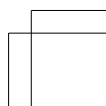
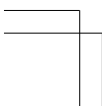
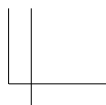


DARK FORTH

松本幸大 著

2019-08-30 版 発行



まえがき

FORTH は不思議なプログラミング言語だ。低レイヤでの手続きを記述するための至極単純な構文 (文法・表現) をもち、実行時に自らの意味論^{*1}を拡張できる。その特性ゆえに、ドメイン固有言語^{*2}(domain-specific language、DSL) を開発するための汎用言語として機能する。

FORTH は 1970 年頃から開発され、多くのプログラミング言語^{*3}に影響を与えながら、その仕様を洗練させてきた。処理系が非常にコンパクトであり、メモリ効率が良いため、現在も組み込み系やロボット制御において活用されている。

本書では、すでに何かしらの言語を用いてプログラミングを経験している人に向けて、FORTH による「スタック指向」プログラミングの基礎やメタプログラミングの初歩について解説していく。

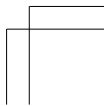
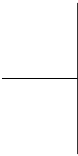
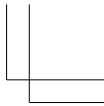
FORTH の多彩な意味論と計算機の織りなす世界に足を踏み入れるあなたに、一体何が待ち受けているのだろう。本書が、あなたにとって良い旅の道しるべとなることを願っている。

FORTH と共にあらんことを。

*1 ここでは、プログラムがどのような動作・結果を持つかを定式化したものを指す。

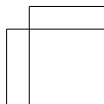
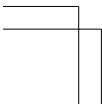
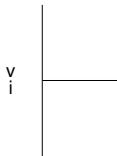
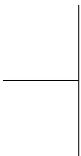
*2 特定のタスク向けに設計されたコンピュータ言語

*3 日本語プログラミング言語「Mind」、オブジェクト指向言語「Mops」「Factor」など



目次

まえがき	iii
第 1 章 はじめに	1
1.1 推奨する読み方	1
1.2 本文中での記法	1
1.3 準備	2
Windows	2
macOS	2
1.4 FORTH の歴史	2
1.5 役立つリファレンス	2
第 2 章 FORTH 序論	3
2.1 ワードとインタプリタ	3
2.2 スタック操作ワード	5
2.3 コメント	7
2.4 セル	7
2.5 スタック表記法	8
2.6 ワード定義の基本	8
2.7 まとめ	10
2.8 練習問題	10



第 1 章

はじめに

この章では、まず本書で推奨する読み方と、本文中での記法に関してのいくつかの約束事について紹介する。その後、処理系のインストールなどの準備について述べる。最後に FORTH の略史や役立つリファレンスを紹介する。

1.1 推奨する読み方

FORTH における用語はできる限り説明した上で導入するようにしているが、逐次実行・条件分岐・繰り返しといった手続き型プログラミングの基礎的な概念や、配列・スタックといった基本的なデータ構造については説明を省いている。2 章からはそれぞれ練習問題を設けている。本文中の内容を確認するために、実際に頭と手を動かすことを推奨する。

1.2 本文中での記法

```
." HELLO WORLD"
```

1.3 準備

GNU forth を使用する。

Windows

<http://gforth.org> からインストーラを入手する。

macOS

```
$ brew install gforth
```

1.4 FORTH の歴史

チャールズ・ムーアによって開発された。

1.5 役立ったりファレンス

また、FORTH ではなく iMops という派生言語について書かれているこのサイトも役に立つ。

第 2 章

FORTH 序論

この章では、FORTH の学習を始める上で把握しておくべき周辺の基礎概念と、FORTH の中核をなす諸概念について扱っていく。

2.1 ワードとインタプリタ

FORTH のシンタックスは、要は「文字列を空白文字で区切って並べただけ」だ。この極限まで単純化されたシンタックスが、後述する多彩な意味論と相まって、高い融通性を発揮する。

これから少しの間、Gforth を「電卓」として使っていく。Gforth を起動し、リスト 2.1 を書き写してみよう。入力が終わったら、改行してみてほしい。

リスト 2.1: はじめての FORTH プログラム

```
6 3 4 + *
```

正しく動いていれば、たった今書き込んだコードのすぐ左に ok 1 と表示されたはずだ。これは、Gforth が入力されたコードをすべて解釈実行し終

わったことを示している。Gforth は、この瞬間にどんな流れで、どうリスト 2.1 を解釈したのだろうか。

Gforth は手続きに必要なパラメータをスタック上で管理しながら、ソースコードを先頭から一直線に解釈する。リスト 2.1 を構成している文字列は 6, 3, 4, +, * の 5 つだ。

- 最初の 3 つは 32 ビット符号付き整数値として解釈され、それぞれ順番にスタックにプッシュされる。
- あと 2 つの、数値として認識できない文字列は「ワード (word)」として解釈される。

ワードは、簡単に言えば手続きに名前をつけたものだ。これは、他の言語における関数やサブルーチンのようなものとして理解される。

FORTH 処理系がインタプリタとして動作しているときにワードを認識すると、対応する**解釈時意味論** (interpretation semantics) に従って呼び出しが発生する。後述するが、「～時意味論」のようにワードには複数の意味論が存在し、呼び出し時の処理系の状態によって、ワードの挙動は異なる (ように作為的にワードを定義できる)。

FORTH 処理系はデフォルトではインタプリタとして動作しており、この段階では解釈時意味論だけ考慮していれば問題ない。

リスト 2.1 の説明に戻るが、インタプリタの挙動は以下ようになる。

1. 6 3 4 まで解釈した直後には、スタックにはそのまま底から 6, 3, 4 が積まれている。
2. + を読み込んだインタプリタは、ワード名とその定義を紐付けて保存している「辞書 (dictionary)^{*1}」からその定義を見つけ出し、呼び出す。+ ワードはスタックの一番上に積まれている 2 つの数 3, 4 を

^{*1} 辞書は、実際には「ワードの定義が書き込まれたメモリ領域の先頭番地」を保存しているものだ。

ポップし、和 7 をプッシュする。

3. * を読み込んだインタプリタは同様に定義を見つけ出して呼び出す。

* ワードはスタックの一番上に積まれている 2 つの数 6, 7 をポップし、積 42 をプッシュする。

つまり、リスト 2.1 は $(3 + 4) \times 6$ を評価させて結果をスタックに積むプログラムだったわけだ。それを確かめるために、今のスタックの内容を出力させる .S ワードを呼び出して見てほしい。すぐ右側に <1> 42 ok 1 と出力されるはずだ。評価結果が正しく格納されていることがわかる。

FORTH において算術式の評価を表現すると、自然に「被演算子を先に並べて、最後に演算子を書く記法」になる。この記法は一般的には逆ポーランド記法と呼ばれており、スタックとの相性が良い。

今後は、特に指定のない限り、Gforth を起動した直後の状態を想定してサンプルコードを扱っていく。BYE ワードを呼び出せば Gforth を終了できる。

2.2 スタック操作ワード

FORTH にはスタックの内容を操作するための組み込みワードが用意されている。その 1 つが DUP ^{*2} ワードだ。スタックの一番上に積まれている要素を複製する。

リスト 2.2: DUP ワードの利用

```
8 DUP * .
```

リスト 2.2 を実行すると、すぐ右側に 64 ok 0 と表示される。(ドット) ワードを呼び出すと、スタックの一番上の要素をポップして出力する。

^{*2} アルファベットの大文字小文字の区別はないため、dup と入力しても構わない。

つまり DUP ワードによって複製が行われスタックにはふたつの 8 が残り、それらは * ワードによってポップされ 2 数の積 64 が残る。

他にもいろいろなスタック操作作用のワードが用意されている。

DROP

一番上の要素を破棄する。

NIP

一番上から 2 つ目の要素を破棄する。

SWAP

一番上の 2 つの要素の順序を入れ替える。

OVER

一番上から 2 つ目の要素をコピーして、それをプッシュする。

スタック上の 2 つの要素をまとめて操作するためのワードも用意されている。

2DROP

一番上から 2 つの要素を破棄する。

2SWAP

スタック上の (底の方から) a b c d を、c d a b に並び替える。

2OVER

スタック上の a b c d のうち a b をコピーしてプッシュする。プッシュ操作が完了すると a b c d a b のようになっている。

2DUP

一番上から 2 つの要素をコピーして、同じ順でプッシュする。

2SWAP ワードと 2OVER ワードの説明では、スタック上の要素に識別名をつけた上で、呼び出しの前後でそれがどう変化するかを示している。こういった記法については、後の「2.5 スタック表記法」でより厳密に定義して使っていくことにする。

2.3 コメント

リスト 2.3: コメントの書き方

```
\ コメント  
( コメント )
```

上のような書き方をすれば、コメントの部分は無視される。`\` と `(` はそれぞれひとつのワードとして独立に定義されており、これら呼び出すために、後ろに空白文字が必要になる。`\` ワードは、後続するソースコードを行末まで読み飛ばす。`(` ワードは、後続するソースコードを `)` まで読み飛ばす。この 2 つのワードのように、後ろに続いているソースコードをパースして引数に取ることが可能なワードも存在している。後述するが、そのようなワードをユーザが定義することも可能である。

2.4 セル

FORTH で扱われるデータのサイズの最小単位は**セル**である。スタックに積まれる要素のサイズもすべて 1 ワードとなっている。1 セルあたりの実際のサイズは処理系依存だが、標準仕様に含まれている `CELLS` ワードを用いて、使用中の処理系でのサイズを知ることができる。

リスト 2.4: 1 セルあたりのバイト数を調べる

```
1 CELLS .
```

`CELLS` ワードは、スタックに積まれている整数値 1 つをポップし、それ

に 1 セルあたりのバイト数を掛けてプッシュする。Gforth では 1 セル = 8 バイトなので、リスト 2.4 を実行すると 8 と出力されるはずだ。

2.5 スタック表記法

ワードを呼び出した際のスタックへの影響を記述するための記法を紹介する。これは、FORTH のリファレンス等でワードの挙動を説明する際に多 사용되는。基本的には、以下のようなフォーマットになっている。

リスト 2.5: スタック表記法

```
( before -- after )
```

before, after の部分には、スペース区切りでシンボルが 0 個以上並ぶ。シンボルは、FORTH におけるデータ型や、その値のサイズを表現するための記号である。Forth 2012 Standard において用いられているシンボルについて、表 2.1 にまとめておく。現段階では扱っていない言語機能に関するシンボルも含まれているが、今後参照する資料として一旦読み流しておいてほしい。

2.6 ワード定義の基本

表 2.1: スタック表記法で用いられるシンボル

シンボル	データ型	サイズ
flag	フラグ	1 セル
char	文字	1 セル
n	符号付き整数	1 セル
u	符号無し整数	1 セル
x	指定のない 1 セル分のデータ	1 セル
xt	エグゼキューショントークン	1 セル
addr	アドレス	1 セル
a-addr	アラインメントされた領域のアドレス	1 セル
c-addr	文字用にアライメントされた領域のアドレス	1 セル
d	2 セル符号付き整数	2 セル
ud	2 セル符号無し整数	2 セル
colon-sys	定義コンパイル	処理系依存
do-sys	DO-LOOP 構造	処理系依存
case-sys	CASE 構造	処理系依存
of-sys	OF 構造	処理系依存
orig	コントロールフロー原点	処理系依存
dest	コントロールフロー方向	処理系依存
loop-sys	ループコントロール引数	処理系依存
nest-sys	定義セル	処理系依存
i * x, j * x, k * x	任意	0 セル以上

リスト 2.6: ワード定義

```
: SQUARE DUP * ;
```

2.7 まとめ

2.8 練習問題

索引

* (アスタリスク), 5

+, 4

. (ドット), 5

.S, 5

2DROP, 6

2DUP, 6

2OVER, 6

2SWAP, 6

a-addr, 9

addr, 9

BYE, 5

c-addr, 9

case-sys, 9

CELLS, 7

char, 9

colon-sys, 9

d, 9

dest, 9

do-sys, 9

DROP, 6

DUP, 5

flag, 9

loop-sys, 9

n, 9

nest-sys, 9

NIP, 6

of-sys, 9

orig, 9

OVER, 6

SWAP, 6

u, 9

ud, 9

x, 9

xt, 9

解釈時意味論, 4

逆ポーランド記法, 5

シンボル, 8

DARK FORTH

2019 年 8 月 30 日 発行

著 者 松本幸大

(C) 2019 Kodai Matsumoto