DARK FORTH

松本幸大　著

2019-09-07 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

v

目次

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。

本書での具体的な目的は「FORTH の特徴的な言語機能とその活用方法を理解して、これから Forth Standard を参照しながら探求していく土台をつくること」である。この目的のために、あえて FORTH の算術・文字列操作などの雑多な API には深入りせず、よりコアな言語機能の説明を優先している。その関係上、はじめは辞書的な読み方ではなく連続したチュートリアルとして、先頭の章から順番に読み進めることを推奨する。

1.2　Gforth のインストール作業

本書では、FORTH 処理系のひとつである GNU forth (以降、Gforth) を使用する。著者が使用している Gforth のバージョンは `0.7.9` である。`http://gforth.org` (図 1.1) でインストール方法に関する情報やチュートリアル (英語) が提供されている。

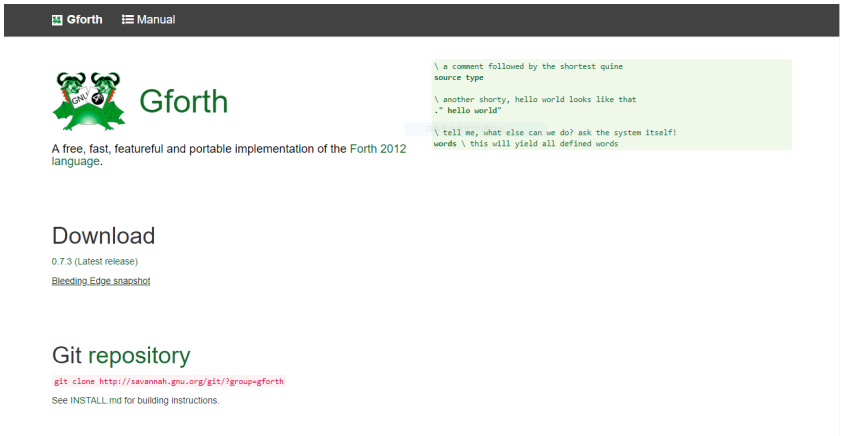

図 1.1: Gforth 公式サイト

Windows

`http://www.complang.tuwien.ac.at/forth/gforth/` から、「Win32 self installing」と記載されたインストーラを入手し、それを実行して指示に従ってインストールする。

macOS

Homebrew 経由で導入できる。

```
brew install gforth
```

Ubuntu

apt コマンドでインストールできる。

```
sudo apt install gforth
```

Arch linux

AUR [*1]で提供されているので、yay 等を利用してインストールする。

[*1] パッケージ詳細ページ: https://aur.archlinux.org/packages/gforth/

```
yay -S gforth
```

1.3　FORTH の歴史と標準

FORTH は、1958 年からチャールズ・ムーアが個人開発していたプログラミングシステムから考案された。1968 年に、このソフトウェアをミニコンピュータ上で FORTRAN で実装したものが FORTH の原型とされている。FORTH が他のプログラマに最初に公開されたのは 1970 年代で、アメリカ国立天文台 (NARO) にいたエリザベス・ラザーによって始められたものである。NARO にいた 2 人は制御用ソフト開発において FORTH を完成させ、後に FORTH, Inc. を設立した (1973 年)。ムーアの仕事の関係上、言語開発の初期段階では移植性の高さが求められていた。当時の FORTH は現在のスクリプト言語のような立ち位置ではなく、FORTH 処理系自体が軽量 OS として用いられていた。そういった処理系のひとつである MacFORTH は、アップル Macintosh の最初の常駐開発システムとして採用されていた。

その後も FORTH は様々な環境に移植され、1979 年に言語仕様の標準化が始まった。1979 年に成分化された FORTH-79、1983 年に成分化された FORTH-83 は「事実上の標準[*2]」である。これらの標準は 1994 年に ANSI によって統合され、ANS Forth と名付けられた。1979 年には ISO/IEC においても標準化されている[*3]。

本書で扱う Gforth は Forth 2012 Standard という規格に沿って現在も盛んに開発されており、Forth 2012 Standard は Forth 200x 標準化委員会によって策定されている。

[*2] デファクトスタンダードとも呼ばれる。標準化機関等が定めた規格ではなく、市場における競争や広く採用された結果として「事実上標準化した基準」を指している。

[*3] ISO/IEC 15145:1997 https://www.iso.org/standard/26479.html

1.4 役立つリファレンス

Forth Standard

`https://forth-standard.org/`

Forth 2012 Standard に関する情報が共有され、さらなる技術仕様 (プロポーザル) がまとめられている。

5

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。

7

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 `42 ok 0` 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 6 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 3 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のようになる。

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」を保存していくものだ。

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。
4. . ワードの解釈時意味論は「スタックの一番上の要素をポップして出力すること」であるから、スタックは空になり、 42 が出力される。

つまり、リスト 2.1 は $(3+4) \times 6$ を評価させて、結果を出力させるプログラムだったわけだ。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。

スタックの要素数と各要素の値を出力する .S ワードと、改行コードを出力する CR ワードも使用して、 リスト 2.1 でのスタックの変化を探ってみよう。(リスト 2.2)

リスト 2.2: スタックの変化を探る

```
CR .S 6 3 4 CR .S + CR .S * CR .S
```

結果は リスト 2.3 のようになるはずだ。スタック上で正しく計算されていることがわかる。

リスト 2.3: 実行結果

```
<0>
<3> 6 3 4
<2> 6 7
```

```
<1> 42
```

2.2 基本的な算術

+ ワードや * ワードと同様に、以下のワードを用いて基本的な算術を行う。

- (マイナス)
: 減算

/ (スラッシュ)
: 除算

MOD
: 剰余

/MOD
: 「剰余」「商」を順にプッシュする

NEGATE
: 符号を反転する

```
23 8 - 3 / .
7 4 MOD .
7 4 /MOD .S
2 NEGATE .
```

2.3　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 2.5: DUP ワードの利用

```
8 DUP * .
```

リスト 2.5 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。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。

ROT
: 一番上から 3 つの要素を逆順にする。

[*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。

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「2.6 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。

2.4 コメント

リスト 2.6: 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、これらを呼び出すために、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。

2.5 セル

FORTH で扱われるデータのサイズの最小単位は**セル** (cell) である。スタックに積まれる要素のサイズもすべて 1 セルとなっている。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 2.7: 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 2.7 を実行すると 8 と出力されるはずだ。

2.6 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用される。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 2.8: 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。

2.7 ワード定義の基本

ユーザがワード定義する、つまり辞書に新たな項目を追加する方法を紹介する。基本的には、: ワードを使用する。

リスト 2.9: ワード定義

```
: square ( n -- n ) DUP * ;
```

リスト 2.9 では、新たに square というワードを定義している。(n -- n) の部分はコメントとして扱われ、プログラマに対して square ワードの挙動をスタック表記法で示す「ドキュメント」としての役割を担っている。

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 2.10: square ワードの利用

```
8 square .
```

square ワードの解釈時意味論は「スタックの一番上の要素を複製し、上 2 つの要素を掛け合わせること」となるため、実際に square ワードを呼び

出す リスト 2.10 を実行すると 64 が出力される。

: ワードは、呼び出されるとまず後続するソースコードを空白文字の直前まで読み込む。それを名前として保存した後、処理系を**コンパイル状態** (compilation state) に移行させる。コンパイル状態のときにワードを認識すると、先述した解釈時意味論ではなく**コンパイル時意味論** (compilation semantics) に従って呼び出しが発生する。

一部のワードを除いて、多くのワードのコンパイル時意味論は「自らの解釈時意味論を定義に追加すること」である。「処理系がインタプリタとして動作しているとき」の意味論と、「自らの呼び出しを定義に含む別のワードが呼び出されているとき」の意味論が異なっている場合、後者を**走行時意味論** (runtime semantics) と呼んで解釈時意味論と区別することがある。

2.8　ソースファイルの利用

2.9　Gforth パイプ

2.10　まとめ

2.11　練習問題

1. $6-7\times 8+9$ を Gforth 上で計算せよ。
2. 2SWAP ワードのスタック表記を示せ。
3. スタック上の x, y をポップし、$y\times 2+x$ をプッシュするワード word2-3 を定義せよ。
4. スタック上の x, y をポップし、$x \mod 3-y/4$ をプッシュするワード word2-4 を定義せよ。
5. NIP ワードと同じ挙動のワード word2-5 を定義せよ。ただし、定義に「 NIP ワードの呼び出し」を含めてはいけない。

第 3 章

文字列・印字

この章では、FORTH で文字列を扱う上で最低限知っておく必要のある言語機能について述べる。

17

3.1 HELLO WORLD

." ワードは、後続するソースコードを " まで切り出してそのまま出力する。いわゆる printf デバッグに用いられることが多い。

```
." HELLO WORLD"
```

3.2 通常の文字列

3.3 Unicode 文字列

第 4 章

フロー制御

4.1　条件分岐

4.2　繰り返し

4.3　再帰

第 5 章

データ・辞書

第 6 章

コンパイラ変形

付録 A

forth-jp について

筆者は、「FORTH 言語に興味がある技術者同士で情報を共有する場所」としてコミュニティ「forth-jp」を運営している。活動場所・内容は以下の通りである。

Slack ワークスペース

参加メンバー同士で、FORTH 言語の仕様や作成したプログラム、参考になるサイトなどについて共有する場所として使用している。

Scrapbox プロジェクト

`https://scrapbox.io/forth-jp`

FORTH に関する知見を蓄積していく場所として使用している。

参加するには、招待リンク (`https://bit.ly/2ZzTiYf`) から直接 Slack ワークスペースに入るか、または筆者の Twitter アカウント (`@0918nobita`) に連絡をとってほしい。

付録 B

練習問題の解答・解説

第 2 章

問題 1

```
6 7 8 * - 9 +
```

問題 2

```
( x1 x2 x3 x4 -- x3 x4 x1 x2 )
```

問題 3

```
: word2-3  ( n1 n2 -- n3 )  * + ;
```

問題 4

解答例 1

```
: word2-4  ( n1 n2 -- n3 )  SWAP 3 MOD SWAP 4 / - ;
```

解答例 2

```
: word2-4  ( n1 n2 -- n3 )  -4 / SWAP 3 MOD + ;
```


問題 5

```
: word2-5  ( x1 x2 -- x2 )  SWAP DROP ;
```

あとがき

索引

30

DARK FORTH

2019 年 9 月 7 日　発行
著　者　松本幸大