オレ様言語の作り方

LL魂 2007

Ｓｕｋｕｎａ編

gikoforth@s13.xrea.com

小原広之

本日の予定

- Forthって何？
- オレ言語への道1
- オレ言語への道2

Forthって何？

現在、雇ってくださる方、大絶賛募集中です。本気と書いてマジと読む。得意な言語はFo…

Forth（1970）

- Charles H. Moore
- スタック系言語の祖
- 後置記法、パラメタスタックとリターンスタック、ワードとディクショナリ、immediateワード（コンパイル時実行）
- コンパクトで高速な処理系、高い自己拡張性
- アメリカ国立電波天文台の電波望遠鏡の制御やデータ処置に用いられた(1971)

Forth

(1+2)*(3+4)

```
1 2 + 3 4 + *
```

Factorial

```
:fact
   dup 0=
   IF
      drop 1
   ELSE
      dup 1-recurse*
   THEN
;
```

Ｊｏｙ（２００１）

- Manfred von Thun
- 最初の純関数型Concatenative言語
- スタック系言語 = Concatenative言語
  - 「式の連結」＝「関数の合成」
- 代入がない。変数がない。仮引数もない。
  - それでもプログラムできる

"良いアイディアを探したければ、たくさんの言語のうち、簡潔さで名高い言語を調べると良いだろう：Forth, Joy, Icon等だ。"
-- Paul Graham (2002)『簡潔さは力なり』

Joy

Factorial

```
DEFINE
  fact ==
    [
      [pop 0 =]
      [pop pop 1]
      [[dup 1 -] dip dup i*]
      ifte
    ] dup i.
```

Fact(5)

```
5 [1] [*] primrec.
```

オレ言語への道1

オレ言語への道１（接触編）

- Macアプリを簡単に作成できるMopsというオブジェクト指向Forthのファンだった
- しかしMac OS Xへの移行の時期には不具合が多かった（現在は安定）
- なんとなくpForthを触り始める
- なんとなくオブジェクト指向ライブラリを入れてみる

オレ言語への道１（発動編）

- バグがあって上手く動かない！
  - ソースを読み、なんとか自力で修正
- 実装方法がなんとなく理解できた
- 思いつき → 実装
  - この項を幾度か繰り返す
- だんだん楽しくなってキタ
- なんだか得体の知れないモノに・・・

GikoForth（2001）

- pForth + OOP Library + α
- +αの部分
  - 日本語風プログラミング
    - Mind & ひまわりにインスパイヤされた
    - 非分かち書き日本語ソースコードで動作
      - コンパイル時にForth辞書を利用した最長一致で分かち書きに変換
  - ブロッククロージャ

反省点

- ポータビリティの喪失
  - ANS Forthの機能だけでは、もはや動作しない
- 型がない
  - スタック上の値が整数なのかオブジェクトへのポインタなのか、処理系は判断できない
- ＧＣがない
- メソッド呼び出しが遅い
  - コンパイル時にオブジェクトを特定できないLate bindだと毎回メソッド探索
  - メソッド探索がForthで書いてある

オレ言語への道2

オレ言語への道２（惑う男）

- （´・ω・｀）
- ポータビリティと互換性の問題
  - 自分しか使わないから問題なし
- 型がない
  - Rubyのようにタグビットをつける
- ＧＣがない
  - 保守的ＧＣにしてみる
- 遅い
  - とりあえずvtableを取り入れてみる
  - 速度の気になる部分はCで書く

オレ言語への道２（働く男）

- 既存の処理系をベースにする？
  - たとえばGforth
    - ソースを見てもよくワカラン
- スクラッチから書く？
  - Cでまともにプログラム書いたことない
    - 某2chでみつけたBrainF*ck処理系を参考に
- 2003年、開発スタート
  - 同時期にスタートしたFactorはおろか、Catにも鮮やかに抜き去られる（2007年現在）
  - ↑ 今、ココ

Ｓｕｋｕｎａ（２００３）

- No22の作っているToy言語
- Forth系、というか、ほぼForth
- ブロッククロージャ（のようなもの）
- クラスベース、単一継承＋Mixin
- オプショナルな型
- 静的型チェック（できる範囲で）
- 保守的GC
- 無限リスト（のようなもの）

お題：FizzBuzz (1)

```
{ FizzBuzz
  [  ""
    over 5 mod 0= [ "Fizz" :+ ] ift
    over 3 mod 0= [ "Buzz" :+ ] ift
    dup "" = ifelse
  ] :map
}
1 to: 100 FizzBuzz [ :print cr ] :each
```

お題：FizzBuzz（2）

```
( "" 2 times "Fizz" ) :cycle -> Fizz
( "" 4 times "Buzz" ) :cycle -> Buzz

( 1 ;; 1+ ) Fizz Buzz [ :+ ] :zipwith
[ dup "" = ifelse ] :zipwith -> FizzBuzz

100 FizzBuzz :take [ :print cr ] :each
```

地味な実演（時間があれば）

ダメなところ

- open classじゃない
- ライブラリが貧弱だ
- エラーメッセージがテキトーだ
- 型チェッカーがテキトーだ
- 実用的に使えない
- ソースコードがはずかしい
  - ダウンロードしている人がたまにいる
    - → 穴があったら入りたい

それでもなぜ作るのか？

- こつこつ作る → 小さな達成感
- 車輪の再発明 → 勉強になる（気がする）
- バグ退治 → パズル的に楽しむ
  - 退治できないとき →「仕様です！」
  - 趣味言語なら人に迷惑かからない（と思う）
- なにより楽しい
- 楽しければそれでいい

まとめ

- やくにたたなくたっていいぢゃない。

にんげんだもの。

参考ＵＲＬ

- The Evolution of Forth
  - http://www.forth.com/resources/evolution/index.html
- Main page for the programming language JOY
  - http://www.latrobe.edu.au/philosophy/phimvt/joy.html

付録

Forth: 後置記法

- Ruby

```
p 1+2

p (1+2)*(3+4)
```

- Forth

```
1 2 + .

1 2 + 3 4 + * .
```

Forth: どこで使われてるの？

- 主にメモリの制約が厳しい組み込み分野
- AppleやSunのOpenFirmwareで使用
  - command + option + O + F 同時押し起動
  - でもIntel Mac以降はEFIが使われている
- シャトルのロボットアームのシミュレータ
- Forthプロセッサ
  - Harris RTX 2000, 2010 等
  - 惑星探査機などに搭載される観測機器の制御など
  - NEAR, CASSINI, Deep Impact, ROSETTA …

Forth: 関数定義（ワード定義）

- Ruby

```
def addOne(x)
  x+1
end
```

- Forth

```
: addOne
  1 +
;
```

Forthの影響を受けた言語

- PostScript (1982)
  - John Warnock & Chuck Geschke
  - Adobeの開発したページ記述言語
  - アップルのLaser Writerに搭載された
- colorForth (2001)
  - Forthの産みの親、Moore自身が開発
  - ワードの属性をソースコードの色で指定
  - ハフマン符号化されたソースコードを解釈実行
  - 言語コアは2Kバイト

PostScript & colorForth

- PostScript

```
/addOne {
   1 add
} def
```

- colorForth

```
addOne
  1 + ;
```

Forthの影響を受けた言語

- Mind (1985)
  - 片桐 明 氏
  - 高性能なスタック系日本語プログラム言語
  - CGIや全文検索エンジンなど実用プログラムを開発可
  - 1999年GPLライセンスに

Ｍｉｎｄ

サンプル

午前？とは
　　時刻を得て
　　時が　１２より　小さいこと。

メインとは
　　午前？
　　　　ならば　「おはよう」を
　　　　さもなければ
　　　　　　「こんにちは」を
つぎに
　　表示し　改行すること。

Ｊｏｙの影響を受けた言語

- Factor (2003)
  - Slava Pestov
- Cat (2006)
  - Christopher Diggins
- V (2007)
  - Rahul

Joy & Cat

- Joy

```
DEFINE addOne ==
  1 +
.
```

- Cat

```
define addOne {
  1 +
}
```

Ｎｅｏｎ（１９８５）

- Kriya Systemsから発売されたＭａｃ用の開発環境
- Ｓｍalltalkにインスパイヤされたオブジェクト指向Forth
- クラスの継承関係と無関係に多態可能
- サポート停止後、Public domainに(1991)

Ｍｏｐｓ/ＰｏｗｅｒＭｏｐｓ（１９９０）

- Michael Hore
- Neonから派生したPublic domainの開発環境
- Neonのコアをスクラッチから書き直した
  - Indirect threaded (Neon)
    → Subroutine threaded (Mops)
- 多重継承、ガベージコレクション、オブジェクトの永続化などをサポート

pForth（1994）

- Phil Burk, Larry Polansky, David Rosenboom
- C言語によるポータブルなForth実装 (Public domain)
- token threaded

オブジェクト指向ライブラリ

- class10.fth (1997)
  - Andrew McKewan
  - ポータブルなオブジェクト指向ライブラリ
  - NeonやMopsのモデルとほぼ同じ
  - 単一継承
- classM10.fth (1997)
  - Mike Hore
  - class10.fthを多重継承可能に

GikoForth

- addOne

```
単語「addOne」を
    1を足す
と定義
```

- 絶対値

```
単語「絶対値」{ 値 }を
    値が0より小さいならば-1を値に掛ける
    さもなければ値をつぎに
と定義
```

Ｎｅｏｎモデルへの批判

- Anton Ertl, 1997
- Neonモデルは構文的な問題点がある
  - スタック上のオブジェクトに対するメソッド呼び出しが不自然
- Neonモデルは実装が複雑な上、効率の良い実装が難しい
  - 継承関係とは関係なく多態可能

  → vtableが使えない

Ｎｅｏｎモデルの問題点

Late bindの場合

```
selecter:[object]
object selecter:[]
```

メソッドチェインしたい場合

```
selecter:[selecter:[object]]
object selecter:[]selecter:[]
```

↓こう書けるほうがForth的に自然では？

```
object selector selector
```

12行のオブジェクト指向拡張

mini-oof.fs

```
: method ( m v -- m 'v ) Create over , swap cell+ swap
 DOES> ( ... o -- ... ) @ over @ + @ execute ;
: var ( m v size -- m v' ) Create over , +
 DOES> ( o -- addr ) @ + ;
: class ( class -- class methods vars ) dup 2 @ ;
: end-class ( class methods vars -- )
 Create here >r , dup , 2 cells ?DO ['] noop , 1 cells +LOOP
 cell+ dup cell+ r> rot @ 2 cells /string move ;
: defines ( xt class -- ) >body @ + ! ;
: new ( class -- o ) here over @ allot swap over ! ;
: :: ( class "name" -- ) >body @ + @ compile, ;
Create object 1 cells , 2 cells ,
```

ワード定義

- addOne

```
{ addOne
   1 +
}
```

```
{ addOne | int: x — x 'x |
   x 1 +
}
```

少しJoy風味

- クォーテーション（のようなもの）
- コンビネータ（のようなもの）
- 例
  - Fact (recursion combinator)

```
[1[*]primrec]->[fact]
10 fact.
```

少しJoy風味

- Fact

```
[[
   [over 0=]
   [2drop 1]
   [[dup 1-] dip dup i.*]
   ifte
 ] dup i.
]-> [fact]

10 fact.
```

少しＭｏｐｓ風味

- クラス定義など

```
class Point super{ Object }
    int x accessor
    int y accessor
    {classNew: -> y -> x }
    {pos: x y }
end

10 20 Point :new -> p
p :x . p :y . cr
```

少しＭｏｐｓ風味(?)

- メソッド呼び出し(Ｍｏｐｓ風)

```
selector: object
```

- メソッド呼び出し(Ｍｏｐｓ風ではない)

```
object :selector
```

- 両方使える

わずかにRuby風味(?)

- Mixin

```
mixin foo requires{bar}
   {foo: self :bar}
end

class baz super{foo object}
   {baz: self :foo}
end
```

・・・全然似てないし

その他：無限リスト的なもの

```
( 1 ;; 1+ ) -> nats
```

自然数の列（1 2 3 4 5…）

```
( 1 1 ;; tuck + ) -> fibs
```

フィボナッチ数列（1 1 2 3 5 8 13 ..）

ところで {tuck | a b | b a b}

```
fibs [ dup * ] :map
[ 200 < ] :takeWhile
```

（1 1 4 9 25 64 169）が返る

その他：型チェック

```
{ foo 10 " bar" }
```

fooの型は |－int: string: |

```
{ add | int: x int: y | x y + }
```

addの型は | int: int: －int: |

```
{ bar foo add }
```

コンパイル時にtype mismatchと警告

```
{ bar foo swap add }
```

やっぱりtype mismatchと警告