J 言語 事典
Version 3.141592

須田祐司

January 20, 2011

Contents

Chapter 1

Prologue

1.1 J言語とは

J ホームページの冒頭の解説は大変短いものですが、示唆に富む文章です。私なりの翻訳で紹介します。

`http://www.jsoftware.com/`

J is a modern, high-level, general-purpose, high-performance programming language.

J言語は現代の高レベル、高機能、汎用プログラミング言語です。

J is particularly strong in the mathematical, statistical, and logical analysis of data. It is a powerful tool in building new and better solutions to old problems and even better at finding solutions where the problem is not already well understood.

J言語は特に、数学的、統計的、論理的データ処理・分析に強いパワーを持っています。古い課題を解くための新しい、より良い解決手段を構築したり、まだ解決手段の定まっていない課題の答えを発見するのに助けになるでしょう。

“ If you are interested in programming solutions to challenging data processing problems, then the time you invest in learning J will be well spent. ”

もし、あなたがデータ処理を伴う未知の課題を解くためのプログラム開発に興味があるのでしたら、J言語を習得するために費やす時間は十分価値あるものであり、決して無駄にはならないでしょう。

1.2　本書の内容

J 言語に内蔵されている関数はプリミティブと言われています。本書は、JAPLA のクイックレファレンス（初版と第 2 版) に記されているプリミティブから私が理解可能なものを抜粋し、各々の実施例と解説を再編したものです。分野別にプリミティブが下記の様式で記述されています。

　プリミティブ和名 (英語名) 用法　摘要　機能を理解するための実行例と解説

　J 言語ではプリミティブの引数は右と左に置く設計になっています。右引数は y で左引数は x で表されます。

　目次自体でも個々の要点をつかむことが出来るようになっています。オリジナル関数の作り方の章では、プリミティブを組み合わせてどうやって自分用のプログラムを作るかを記しました。GUI プログラム法はオリジナルプルグラムのパラメータ設定や起動をフォームを使って実現するのに有用で、グラフィックスの処理には必須のテクニックです。DLL は、速度が要求されるルーチンを実現するのに有用です。いずれも簡単な例題を通じて基本的な仕組みを解説しました。
　既に他のプログラミング言語の経験がある方は、目次を見ればビルトインされた機能を理解出来ると思います。配列処理については、他言語とはかなり異なりますので、本文の例題を通じて学んで下さい。
　なお、J 言語には、本書に記されたもの以外にも、たくさんの機能が搭載されています。詳細は下記 URL に登録されている JAPLA のクイックレファレンス (第 2 版) でご確認下さい。

http://japla.sakura.ne.jp/jlang/tutorial/doc/jqr_all.pdf

Chapter 2

データの表記法、変数と代入

2.1 数・文字列の表記法・配列データ作成法・式の評価順序

2.1.1 数の表記法。そのまま数字として表記する。

例）百二十三は 123

2.1.2 負の符号 (Negative-sign) 負の数は PASCAL,C 等と異なりアンダーバーで表す。

例）マイナス 10 は_10 と記す。

2.1.3 十進数の桁数表示 (exponential) e を使った十進数の桁数表示。

例）12000 は 1.2e4 と表す。

2.1.4 末尾 (end of number digits) 数字の末尾に x を付加して長い実数表示をさせる。

デフォルトでは 10 桁を超える数は有効桁数が 7 桁の桁数表示で表されるが、それをキャンセルして全ての桁をそのまま表示させる。例）12345678901 を入力すると 1.23456e10 と表示されるが、12345678901x とすると、そのまま、1234678901 と表示される。

2.1.5 複素数表示 (complex-number) 小文字の j を使って複素数を表わす。

例）3j4 は実部 3 虚部 4 の複素数。

2.1.6 文字列 の表記法 文字列はシングルクォート' で挟んで明示する。

例）'abcedfg' '0123456789' 空 (null) データは" となる。空データは数値と文字列の両者で使用可能

2.1.7 ボックス (Box)　ボックスはＪ言語特有のデータ型で、数値や文字を線で囲んだものである。

数値と文字は混在できないが、ボックスを使うと混在させることが出来る。

```
+-----+-----+
| 123 | abc |
+-----+-----+
```

数値 123 と文字列'abc' をまとめて一つのデータとして扱える。ボックスも配列として扱える。

2.1.8 データの扱い方：基本は配列

例）Ｊ言語では数値、文字列、ボックスは基本的に配列として扱われる。単一の数値はスカラー、一行に並んだ場合はベクトル、行列状態に並んだ場合は、マトリクス、そして、行列が複数ある場合は配列となる。プリミティブは配列の全ての要素に平等に働くように設計されている。さらに配列自体を処理するために、多くのプリミティブが準備されている。（配列処理の項参照）次に記載されている、変形はその一つである。

2.1.9 変形 (Reshape) x $ y 左引数の形に右引数を変形したものを返す

Ｊ言語ではデータは一般配列として表現する。数値一つであれば、スカラーであるが、これは　１行１列の行列とも言える。ベクトルは１行ｎ列の数字の集まり。行列はm行ｎ列の数字の集まり。さらに、ｌ（エル）個のｍｘｎ行列はｌｘｍｘｎの配列となる。これらのデータを作るために変形（Reshape）＄というプリミティブが用意されている。例）ベクトル　１２３４５６を 3x2 の行列に変更する。

```
   3 2 $ 1 2 3 4 5 6
1 2
3 4
5 6
```

2.1.10 数・ベクトル・行列・配列の表記法・作成法 数は要素が一つのベクトル、即ちスカラー。ベクトルは１行ｎ列の行列で要素の間にスペースを入れて表記する。m行ｎ列の行列は、まず、各行の要素をスペースを間に入れて１行ｍｘｎ列のベクトルで表し、変形のプリミティブ＄を使ってm行ｎ列に並べ変える。配列も同じくm行ｎ列の行列がｌ（エル）個ある場合、全てを１行ｍｘｎｘｌ個のベクトルにして、変形プリミティブ＄を使って配列の形にする。

例）数はスカラーでそのまま表記

```
   2
2
```

例）ベクトル　１２３

```
   1 2 3
1 2 3
```

例）3x2 の行列 変形プリミティブ＄で行列にする。

```
   3 2 $ 1 2 3 4 5 6
1 2
3 4
5 6
```

例）２つの 3x2 行列を有する配列　変形プリミティブ＄で配列にする

```
  2 3 2 $ 1 2 3 4 5 6 7 8 9 10 11 12
 1  2
 3  4
 5  6

 7  8
 9 10
11 12
```

2.1.11 式の評価順序　式は () で優先評価の指定がなければ右から順に評価される。

例）１＋２＊３％４－２
は最初に 4 - 2 が評価され、その答えに対して　３％ (答え）が評価され、次にその答えに対して　２* (答え) が評価され、最後にその答えに対して　１＋ (答え）が評価される。最終の答えは 4 である。

```
   1 + 2 * 3 % 4 - 2
4
   1 + 2 * (3 % 4) - 2
_1.5
   (1 + 2) * 3 % 4 - 2
4.5
```

2.2 変数と代入

2.2.1 変数

システム予約以外の任意の文字列数字の組み合わせで変数として、値（数値、文字列、ボックス）を次セクションの代入プリミティブを使って代入可能である。この際、変数の型宣言は不要である。J 言語では変数は基本的に配列となっている。スカラー、ベクトル、行列、配列いずれの型でも宣言なしに代入可能になっている。

2.2.2 代入 (Copula(is)) x =: y 右引数を左引数に代入する。=: は大局定義

大局変数（グローバル変数）にデータを代入するには =: を用いる。変数名は英文字（アンダーバーを含む）と数字の組み合わせで作成する。変数名の例。

```
a =: 123
a_1 =: 'abcdefg'
```

2.2.3 代入 (Copula(is)) x =. y 右引数を左引数に代入する。=. は局所定義

局所変数（ローカル変数）にデータを代入するには =. を用いる。変数名は英文字（アンダーバーを含む）と数字の組み合わせで作成する。関数定義の中で使う場合は関数の内部だけで有効である。変数名の例。

```
a_data =. 123
b_data =. 'abcdefg'
```

Chapter 3

基本演算（整数・実数・対数・三角関数・複素数・極座標）

3.1 整数・実数

3.1.1 加算 (Plus) x + y 左引数と右引数の和を返す

左引数と右引数との和を返す。左右のサイズは同じでなければならない。片方がスカラの時はスカラを拡張する。

```
   5 + 1 2 3
6 7 8
   1 2 3 + 4 5 6
5 7 9
```

3.1.2 減算 (Minus) x - y 左引数から右引数を引いた値を返す

左引数から右引数を引く

```
   8 _3 - 6 _5
2 2
```

複素数どうしの引算も可能

```
   3.5j0.5 - 1j1
2.5j_0.5
```

3.1.3 乗算 (Times) x * y 左引数と右引数との積を返す

左引数と右引数との積

```
   0 1 * 3 4
0 4
```

複素数どうしの掛算も可能

```
   3.5j0.5 * 1j1
3j4
```

3.1.4 逆数 (Reciprocal) % y 右引数の逆数を返す

% y は 1 % y と同じ。

```
   %0.25
4
```

複素数の逆数は大きさが逆数で、偏角が反対符号になる

```
   % 3j4
0.12j_0.16
   3j4 * % 3j4
1
   *. 3j4
5 0.927295
   *. % 3j4
0.2 _0.927295
```

3.1.5 階乗 (Factorial) ! y 1 から右引数整数までの積を返す

! y は 1, 2....., y の積 (階乗) を与える。

```
   ! 3 4 5
6 24 120
```

3.1.6 除算 (Devide-by) x % y 左引数を右引数で割った値を返す

左引数を右引数で割る

```
   0 8 % 1 0.4
0 20
```

複素数どうしの割算も可能

```
   3j4 % 1j1
3.5j0.5
   1j1 * 3j4 % 1j1
3j4
```

3.1.7 剰余 (Residue) x | y 右引数を左引数で割った時の余りを返す

右引数を左引数で割ったときの余り

```
   3| i.7
0 1 2 0 1 2 0
   3 | -i.7
0 2 1 0 2 1 0
   _3 | i.7
0 _2 _1 0 _2 _1 0
   _3 | -i.7
0 _1 _2 0 _1 _2 0
```

3.1.8 天井値 (Ceiling) >. y 右引数より大きな最小の整数を返す

右引数より大きな最小の整数 (切上げ)

```
   >. 0.4 _0.3 _3.4
1 0 _3
```

(最大値の取り出し)

```
   >./ 1 2 3 4 5
5
```

3.1.9 床値 (Floor) <. y 右引数を超えない最大の整数を返す

y を超えない最大の整数（切捨て)

```
   <. 0.4 _0.3 _3.4
0 _1 _4
```

(最小値の取り出し)

```
   <./ 1 2 3 4 5
1
```

3.1.10 累乗 (Power) x ^ y 左引数の右引数乗した値を返す

x を y 乗するに等しい

```
   2 3 4 ^ 1 2 3
2 9 64
```

3.1.11 指数 (Exponential) ^ y e の右引数乗した値を返す

e の Y 乗

```
   ^ 0 1 2
1 2.71828 7.38906
```

3.1.12 累乗根 (Root) x %: y 右引数の左引数乗根を返す

y の x 乗根で y ^

```
   2 3 4 %: 4 27 256
2 3 4
```

-1 乗根は逆数に等しい

```
   _1 %: 4
0.25
```

3.1.13 符号 (Signum) * y 右引数が実数の時はその符号を調べ、正には１、０には０、負には－１を返す。右引数が複素数の時は大きさが１で右引数と同じ偏角の複素数を返す。

実数に対しては、正には 1、0 には 0、負には 1 を付与。

```
   * _5 0 4
_1 0 1
```

複素数に対しては、大きさが 1 で、同じ偏角の複素数を返す。

```
   * 3j4
0.6j0.8
```

複素数の大きさと偏角を返すプリミティブ 長さ／偏角 (Length/angle) *. y で確認出来る。

```
   *. 3j4
5 0.927295
   *. 0.6j0.8
1 0.927295
```

3.1.14 逆符号 (Negate) - y 右引数の符号を反対にした値を返す

右引数の符号を反対にする

```
   - _5 0 1
5 0 _1
```

複素数では実部と虚部の符号を逆にする

```
   - 3j4
_3j_4
```

3.1.15 共役 (Conjugate) + y 右引数の共役を返す。実数の範囲内では同じ値を返す。複素数では実部が同じで虚部の符号が反対の複素数を返す。

実数の範囲内ではそのまま

```
   + 0.4 _5 0
0.4 _5 0
```

(共役複素数)

```
   + 3j4
3j_4
```

3.1.16 絶対値 (Magnitude) | y 右引数の絶対値を返す

符号をとって絶対値にする

```
   | 6 _5
6 5
```

複素数に対しては大きさを返す

```
   | 3j4
5
```

3.2 対数

3.2.1 対数 (log) x ^. y 左引数を底とする右引数の対数を返す

x を底とする y の対数に等しい

```
   10 ^. 125 100
2.09691 2
```

3.2.2 自然対数 (Natural log) ^. y 底が e の右引数の対数（自然対数）を返す

底が e の Y の対数 (自然対数)

```
   ^. 1 2
1 0.693147
```

3.3 三角関数

3.3.1 円周率倍 (Pi-times) o. y 円周率πと右引数の積を返す

πと y の積

```
   o.1
3.14159
```

3.3.2 正弦関数 (sin) 1 o. y 右引数（ラジアン）正弦を返す

sin 1 ラジアン

```
   1 o. 1
0.841471
```

度の場合はラジアンへ変換（度に 180 分のπを乗算して求める) してから計算する。 sin 90 度の場合は 90 * o. 1

```
   1 o. 90 * o. 1 % 180
1
```

3.3.3 余弦関数 (cos) 2 o. y 左引数の余弦を返す

cos 1 ラジアン

```
   2 o. 1
0.540302
```

度の場合はラジアンへ変換（度に 180 分のπを乗算して求める) してから計算する。 cos 60 度の場合は 60 * o. 1

```
   2 o. 60 * o. 1 % 180
0.5
```

3.4 複素数・極座標

3.4.1 虚数生成 (Imaginary) j. y 右引数の虚数をもつ複素数を返す

虚数生成
単位虚数と虚部大きさ 4 の虚数を生成

```
   j. 1
0j1
   j. 4
0j4
```

3.4.2 複素数生成 (Complex) x j. y 左引数を実部、右引数を虚部とする複素数を返す

複素数生成
実部 3、虚部 4 の複素数を生成

```
   3 j. 4
3j4
```

3.4.3 単位複素数 (Angle) r. y 右引数で与えられるラジアン値を偏角とする大きさ１の複素数を返す

1 ラジアンを偏角とする大きさ 1 の複素数を返す

```
   r. 1
0.540302j0.841471
```

2 ラジアンを偏角とする大きさ 1 の複素数を返す

```
   r. 2
_0.416147j0.909297
```

3.4.4 実部／虚部 (Real/Imaginary) +. y 複素数の実部と虚部を返す

複素数の大きさと偏角を返す

```
   *. 3j4
5 0.927295
```

3.4.5 逆符号 (Negate) - y 右引数の符号を反対にした値を返す

実部と虚部の符号を逆にする

```
   - 3j4
_3j_4
```

3.4.6 共役 (Conjugate) + y 右引数の共役を返す。実数の範囲内では同じ値を返す。複素数では実部が同じで虚部の符号が反対の複素数を返す

共役複素数　実部が同じで虚部の符号が反対の複素数を返す

```
   + 3j4
3j_4
```

3.4.7 絶対値 (Magnitude) | y 右引数の絶対値を返す

複素数の大きさを返す

```
   | 3j4
5
```

3.4.8 長さ／偏角 (Length/angle) *. y 右引数複素数の大きさと偏角を返す

複素数 3j4 の大きさと偏角を返す

```
   *. 3j4
5 0.927295
```

3.4.9 極座標表示 (Polar) x r. y 左引数の大きさで右引数の偏角の複素数を返す

大きさが 1 で偏角が 1 ラジアンの複素数を返す

```
   1 r. 1
0.540302j0.841471
```

Chapter 4

整数生成と基本演算の便利な短縮型

4.1 整数生成 (integers) i. y ０〜右引数-1 までの整数を生成する

```
   i. 10
0 1 2 3 4 5 6 7 8 9
   i. _4
3 2 1 0
   i. 2 3
0 1 2
3 4 5
```

4.2 ２倍 (Double) +: y 右引数の２倍を返す

```
   i.5
0 1 2 3 4
   +: i.5
0 2 4 6 8
```

4.3 ２分の１ (Halve) -: y 右引数の２分の１を返す

```
   i.5
0 1 2 3 4
   -: i.5
0 0.5 1 1.5 2
```

4.4 平方 (Square) *: y 右引数の平方を返す

```
   i.5
0 1 2 3 4
   *: i.5
0 1 4 9 16
```

4.5 平方根 (Square root) %: y 右引数の平方根を返す

```
   i.5
0 1 2 3 4
   %: i.5
0 1 1.41421 1.73205 2
```

4.6 １増 (Increment) >: y 右引数に１を加えた値を返す

```
   i.5
0 1 2 3 4
   >: i.5
1 2 3 4 5
```

4.7 １減 (Decrement) <: y 右引数から１を引いた値を返す

```
   i.5
0 1 2 3 4
   <: i.5
_1 0 1 2 3
```

Chapter 5

２進数、ｎ進数

5.1 ２進数 (Base-2) #. y ２進数で与えた右引数の１０進数値を返す

```
   #. 0
0
   #. 1
1
   #. 1 0
2
   #. 1 1
3
   #. 1 0 0
4
   #. 1 0 1
5
   #. 1 1 0
6
   #. 1 1 1
7
   #. 1 0 0 0
8
   #. 1 0 0 1
9
   #. 1 0 1 0
10
   #. 1 0 1 1
11
   #. 1 1 0 0
```

```
12
   #. 1 1 0 1
13
   #. 1 1 1 0
14
   #. 1 1 1 1
15
```

5.2 １０進化 (Base) x #. y 左引数を底とする右引数の加重合計値を返す

```
   10 #. 1
1
   10 #. 1 2
12
   10 #. 1 2 3
123
   2 #. 1
1
   2 #. 1 2
4
   2 #. 1 2 3
11
   3 #. 1
1
   3 #. 1 2
5
   3 #. 1 2 3
18
```

5.3 １０進数の２進化 (Antibase-2) #: y １０進数で与えた右引数を２進化値を返す

```
   i. 16
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
   #: i. 16
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
```

```
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1
```

5.4 n進化 (Base-n) x #: y １０進数で与えた右引数の左引数の底による進数表示を返す

０から１５までの２進数表示、０から１８までの３進数表示、５４２０秒の時、分、秒表示を以下に示す。、

```
   i. 16
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
   2 2 2 2 #: i. 16
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1

   i. 19
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
   3 3 3 3 #: i. 19
0 0 0 0
0 0 0 1
```

```
0 0 0 2
0 0 1 0
0 0 1 1
0 0 1 2
0 0 2 0
0 0 2 1
0 0 2 2
0 1 0 0
0 1 0 1
0 1 0 2
0 1 1 0
0 1 1 1
0 1 1 2
0 1 2 0
0 1 2 1
0 1 2 2
0 2 0 0

   24 60 60 #: 5420
1 30 20
```

5.5 ベース (base) xby（左引数と右引数と b の間にスペースは入れない）左引数に進数の値、右引数にその進数での値を置いたときの１０進数の値を返す

１６進数は 0 から f で表される。

```
   16b0
0
   16b1
1
   16b2
2
   16b3
3
   16b4
4
   16b5
5
   16b6
```

```
6
   16b7
7
   16b8
8
   16b9
9
   16ba
10
   16bb
11
   16bc
12
   16bd
13
   16be
14
   16bf
15
   16b10
16
   16bff
255
   16b100
256
```

Chapter 6

乱数

6.1 乱数 (Roll) ? y ０から右引数（整数）－１までの整数乱数を一個返す

```
   ? 10 10 10 10
0 7 8 9
   ? 10 10 10 10
2 2 5 5
   ? 10 10 10 10
5 3 8 0
   ? 10 10 10 10
0 5 9 6
   ? 10 10 10 10
4 0 6 4
   ? 10 10 10 10
9 8 5 7
   ? 10 10 10 10
4 0 1 2
```

6.2 乱数（シード固定） (Roll) ?. y シードを固定して０から右引数（整数）－１までの整数乱数を一個返す

シードを固定すると、いつも同じ乱数が発生する。

```
   ?. 10 10 10 10
6 5 9 2
   ?. 10 10 10 10
```

```
6 5 9 2
   ?. 10 10 10 10
6 5 9 2
   ?. 10 10 10 10
6 5 9 2
```

6.3 非重複乱数 (Deal) x ? y 重複をゆるさず０から右引数（整数）－１までの整数乱数を左引数個返す

```
  5 ? 10
7 4 9 0 2
  5 ? 10
2 0 5 9 3
```

6.4 非重複乱数（シード固定） (Deal) x ?. y 重複をゆるさずシードを固定して０から右引数（整数）－１までの整数乱数を左引数個返す

```
  5 ?. 10
6 9 1 4 0
  5 ?. 10
6 9 1 4 0
  5 ?. 10
6 9 1 4 0
```

Chapter 7

論理演算

7.1 等しい (Equal) x = y 左引数と右引数が等しければ１、そうでなければ０を返す

```
   1 = 1
1
   0 = 1
0
```

7.2 不等 (Not Equal) x ～: y 左引数と右引数が等しければ０、そうでなければ１を返す

```
   0 ~: 1
0
   0 ~: 1
1
```

7.3 大きいか等しい)Larger or equal) x >: y 左引数が右引数より大きいか等しければ１、そうでなければ０を返す

```
   2 >: 1
1
   2 >: 2
1
   2 >: 3
```

```
0
```

7.4 より大きい (Larger than) x ＞ y 左引数が右引数より大きければ１、そうでなければ０を返す

```
   2 > 1
1
   2 > 2
0
   2 > 3
0
```

7.5 小さいか等しい (Lesser or equal) x <: y 左引数が右引数より小さいか等しければ１、そうでなければ０を返す

```
   2 <: 1
0
   2 <: 2
1
   2 <: 3
1
```

7.6 より小さい (Less than) x ＜ y 左引数が右引数より小さければ１、そうでなければ０を返す

```
   2 < 1
0
   2 < 2
0
   2 < 3
1
```

7.7 小さい方 (Lesser of) x <. y 左引数と右引数の小さい方を返す

```
   2 <. 3
2
   2 <. 2
```

```
2
   2 <. 1
1
```

7.8 大きい方 (Larger of) x >. y 左引数と右引数の大きい方を返す

```
   2 >. 1
2
   2 >. 2
2
   2 >. 3
3
```

7.9 論理積 (And) x *. y 左右の引数が共に１のとき１、そうでなければ０を返す

```
   1 1 0 0 *. 1 0 1 0
1 0 0 0
```

7.10 論理和 (Or) x +. y 左右の引数が共に０のとき０、そうでなければ１を返す

```
   1 1 0 0 +. 1 0 1 0
1 1 1 0
```

7.11 否定・補数 (Not) -. y 右引数が１なら０、０なら１を返す

```
   -. 1 0
0 1
```

7.12 否定論理積 (Not and) x *: y 左右の引数が共に１なら０、そうでなければ１を返す

```
   1 1 0 0 *: 1 0 1 0
0 1 1 1
```

7.13 否定論理和 (Not Or) x +: y 左右の引数が共に０なら１、そうでなければ０を返す

```
   1 1 0 0 +: 1 0 1 0
0 0 0 1
```

Chapter 8

集合演算

8.1 一致 (Match) x -: y 左引数と右引数とが一致していれば１、そうでなければ０を返す

```
   1 2 3 -: 1 2 3
1
   1 2 3 -: 1 1 1
0
```

8.2 部分所属 (Raze in) e. y 右引数の各要素を照合してブール数で表示した行列を返す

```
   e. 'japan'
1 0 0 0 0
0 1 0 1 0
0 0 1 0 0
0 1 0 1 0
0 0 0 0 1
   e. '12341'
1 0 0 0 1
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
1 0 0 0 1
   e. '12345'
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
```

```
0 0 0 1 0
0 0 0 0 1
```

8.3 要素の所属 (Member in) x e. y 左引数の要素が右引数の中に含まれていれば１、そうでなければ０を返す

```
   2 e. 1
0
   2 e. 1 2
1
   2 e. 1 2 3
1
   4 e. 1 2 3
0
```

8.4 パターンの所属 (Member of interval) x E. y 右引数の中に左引数のパターンが含まれていれば、始まる位置に１を、そうでなければ０を返す

```
   'ka' E. 'kakao'
1 0 1 0 0
```

8.5 重複排除 (Nub) ~. y 右引数の重複要素を排除したものを返す

```
   ~. 1 1 2 2 3 3
1 2 3
   ~. 'japan'
japn
```

8.6 重複指示 (Nubsieve) ~: y 右引数の要素に重複があれば０を、なければ１を返す

```
   ~:1 1 2 2 3 3
1 0 1 0 1 0
   ~: 'japan'
1 1 1 0 1
```

8.7 自己分類 (Self Classify) = y 右引数の要素の重複を分類して判別し、１で重複、０で非重複を返す

```
   = 'a'
1
   = 'aa'
1 1
   = 'abc'
1 0 0
0 1 0
0 0 1
   = 'abca'
1 0 0 1
0 1 0 0
0 0 1 0
   = 'abcab'
1 0 0 1 0
0 1 0 0 1
0 0 1 0 0
```

Chapter 9

配列処理・配列演算

9.1 整数生成 (integers) i. y ０～右引数-1 までの整数を生成する。負の整数なら降順で生成する

```
   i. 10
0 1 2 3 4 5 6 7 8 9
   i. _4
3 2 1 0
   i. 2 3
0 1 2
3 4 5
```

9.2 インデックス (Index of) x i. y 左引数の中で右引数が最初に現れるインデックスを返す

```
   1 2 3 4 5 6 7 i. 1
0
   1 2 3 4 5 6 7 i. 3
2
   'abcdefg' i. 'e'
4
   'abcdefg' i. 'c'
2
```

9.3 変形 (Reshape) x $ y 左引数の形に右引数を変形したものを返す

```
   2 3 $ 1 2 3 4 5 6
1 2 3
4 5 6
   3 2 $ 1 2 3 4 5 6
1 2
3 4
5 6
```

9.4 形 (shape of) $ y 右引数の各ランクの要素数を返す

```
  a =. 2 3 $ 1 2 3 4 5 6
  a
1 2 3
4 5 6
  $ a
2 3
  a =. 3 2 $ 1 2 3 4 5 6
  a
1 2
3 4
5 6
  $ a
3 2
```

9.5 アイテム数 (Tally) # y 右引数のアイテム（右引数より１つランクの低いセル）の数を返す

```
   # 1 2 3 4 5 6
6
```

9.6 アイテム化 (Itemize) ,: y 右引数ランクを１つ増やした高次の配列を返す

```
   a =. 4 3 $ i.5
   a
```

```
0 1 2
3 4 0
1 2 3
4 0 1
   ,:a
0 1 2
3 4 0
1 2 3
4 0 1
   $ ,:a
1 4 3
```

9.7 リスト化 (Ravel) , y 右引数をリストに変形したものを返す

```
   a =. 3 2 $ i.3
   a
0 1
2 0
1 2
   ,a
0 1 2 0 1 2
   #,a
6
```

9.8 テーブル化 (Ravel Items) ,. y 右引数をテーブル化したものを返す

```
   a =. i.3
   a
0 1 2
   $a
3
   ,.a
0
1
2
   $ ,.a
3 1

   a =. 3 2 $ i.3
```

```
   a
0 1
2 0
1 2
   $a
3 2
   ,.a
0 1
2 0
1 2
   $ ,.a
3 2
```

9.9 複写 (Copy) x # y 右引数を左引数で指定した数だけコピーしたものを返す

```
   2 # 1
1 1
   3 # 1
1 1 1
   2 # 1 2
1 1 2 2
   2 # 1 2 3
1 1 2 2 3 3
```

9.10 連結 (Append) x , y 右引数を左引数の最大ランクの方向に連結したものを返す。左引数がリストなら横に、テーブルなら下に連結する。要素数が不足なら０が付加される

```
   a =. 3 2 $ i. 5
   a
0 1
2 3
4 0
   a , 1
0 1
2 3
4 0
1 1
```

9.11 縦連結 (Stitch) x ,. y 右引数を左引数に縦に連結したものを返す

```
   a =. 3 2 $ i. 5
   a
0 1
2 3
4 0
   a ,. 1
0 1 1
2 3 1
4 0 1
```

9.12 層連結 (Laminate) x ,: y 右引数を、左引数が１つ高いランクになるように連結したものを返す

```
   a =. 3 2 $ i. 5
   a
0 1
2 3
4 0
   a ,: 1
0 1
2 3
4 0

1 1
1 1
1 1
   a ,: 1 2
0 1
2 3
4 0

1 2
0 0
0 0
   a ,: 1 2 3
0 1 0
2 3 0
4 0 0
```

```
1 2 3
0 0 0
0 0 0
   a ,: 1 2 3 4
0 1 0 0
2 3 0 0
4 0 0 0

1 2 3 4
0 0 0 0
0 0 0 0
   a
0 1
2 3
4 0
   $ a
3 2
   a ,: a
0 1
2 3
4 0

0 1
2 3
4 0
   $ a ,: a
2 3 2
```

9.13 カタログ (Catalog) { y 右引数のボックスで与えられた要素の全ての組み合わせを返す

```
a =. 1 2 ; 3 4
   a
 +-----+-----+
 | 1 2 | 3 4 |
 +-----+-----+
   $a
2
   >a
1 2
3 4
   $>a
```

```
2 2

   {a
 +-----+-----+
 | 1 3 | 1 4 |
 +-----+-----+
 | 2 3 | 2 4 |
 +-----+-----+
   ${a
2 2
   >{a
1 3
1 4

2 3
2 4
   $>{a
2 2 2
```

9.14 選択 (From) x { y 右引数から左引数で指定したアイテムを取り出して返す

```
   a =. i. 5
   a
0 1 2 3 4
   0 { a
0
   1 { a
1
   1 2 { a
1 2
   1 2 4{ a
1 2 4

   b =. 5 4 $ i.20
   b
 0  1  2  3
 4  5  6  7
 8  9 10 11
12 13 14 15
16 17 18 19
```

```
   0 { b
0 1 2 3
   0 1 { b
0 1 2 3
4 5 6 7
   2 4 { b
 8  9 10 11
16 17 18 19

   b =. 4 5 $ i.20
   b
 0  1  2  3  4
 5  6  7  8  9
10 11 12 13 14
15 16 17 18 19

   (<1 2;1 2){b
 6  7
11 12

   (<2 3;1 2){b
11 12
16 17
```

9.15 先頭取り (Head) {. y 右引数の先頭アイテムを取り出して返す

```
   a =. i. 5
   a
0 1 2 3 4
   {.a
0
```

9.16 取り (Take) x {. y 左引数が正整数の時は右引数の先頭から、負なら末尾から左引数の絶対値の個数のアイテムを取り出して返す

```
   a =. i.5
   a
0 1 2 3 4
   1 {. a
```

```
0
   4 {. a
0 1 2 3
   2 {. a
0 1
```

9.17 先頭落とし (Behead) } . y 右引数の先頭アイテムを落としたものを返す

```
   a =. i.5
   a
0 1 2 3 4
   }. a
1 2 3 4
```

9.18 落とし (Drop) x } . y 左引数が正整数の時は右引数の先頭から、負なら末尾から左引数の絶対値の個数のアイテムを落としたものを返す

```
   a =. i.5
   a
0 1 2 3 4
   2 }. a
2 3 4
   _2}. a
0 1 2
```

9.19 末尾取り (Tail) {: y 右引数の最終アイテムを取り出して返す

```
   a =. i.5
   a
0 1 2 3 4
   {:a
4
```

9.20 末尾落とし (Curtail) } : y 右引数の最終アイテムを落としたものを返す

```
   a =. i.5
   a
0 1 2 3 4
   }:a
0 1 2 3
```

9.21 アイテム修正 (Item Amend) m } インデックスmで右引数の要素を並べ変える

```
a =. 'japan','korea','china'
   a
japankoreachina
   3 5 $ a
japan
korea
china
   1 0 1 2 0 } 3 5 $ a
karnn

   b =. 4 5 $ i.20
   b
 0  1  2  3  4
 5  6  7  8  9
10 11 12 13 14
15 16 17 18 19
  0 1 2 3 3 } b
0 6 12 18 19
  2 } 0 1 2 3 3 } b
12
```

9.22 修正 (Amend) x m } インデックスmで指定した右引数の要素を左引数に変更したものを返す

```
   '*' 1 3 5 7} 'abcdefghij'
a*c*e*g*ij
   'BD' 1 3 } 'abcd'
aBcD
```

```
  35 (0 2 4)  } (10 20 30 40 50)
35 20 35 40 35

   b =. 4 5 $ i.20
   b
 0  1  2  3  4
 5  6  7  8  9
10 11 12 13 14
15 16 17 18 19
   (<1 2) { b
7
   20 (<1 2) } b
 0  1  2  3  4
 5  6 20  8  9
10 11 12 13 14
15 16 17 18 19
```

9.23 逆順 (Reverse) |. y 右引数の要素を逆順にしたものを返す

```
   a =. i.5
   a
0 1 2 3 4
   |.a
4 3 2 1 0
   b =. 5 4 $ i.20
   b
 0  1  2  3
 4  5  6  7
 8  9 10 11
12 13 14 15
16 17 18 19
   |.b
16 17 18 19
12 13 14 15
 8  9 10 11
 4  5  6  7
 0  1  2  3
```

9.24 回転 (Rotate) x |. y 右引数のアイテムを左引数回だけ回転したものを返す。左引数が負なら後ろから回転する

```
   a =. i.5
   a
0 1 2 3 4
   1 |. a
1 2 3 4 0
   2 |. a
2 3 4 0 1
   3 |. a
3 4 0 1 2
   _1 |. a
4 0 1 2 3
   _2 |. a
3 4 0 1 2
   _3 |. a
2 3 4 0 1

   b =. 5 4 $ i.20
   b
 0  1  2  3
 4  5  6  7
 8  9 10 11
12 13 14 15
16 17 18 19
   1 |. b
 4  5  6  7
 8  9 10 11
12 13 14 15
16 17 18 19
 0  1  2  3
   2 |. b
 8  9 10 11
12 13 14 15
16 17 18 19
 0  1  2  3
 4  5  6  7
   _1 |. b
16 17 18 19
 0  1  2  3
 4  5  6  7
```

```
 8  9 10 11
12 13 14 15
   _2 |. b
12 13 14 15
16 17 18 19
 0  1  2  3
 4  5  6  7
 8  9 10 11
```

9.25 転置 (1) (Transpose) |: y 右引数の要素を転置させたものを返す

```
   b =. 3 4 $ i. 12
   b
0 1  2  3
4 5  6  7
8 9 10 11
   |: b
0 4  8
1 5  9
2 6 10
3 7 11
```

9.26 転置 (2) (Transpose) x —: y 左引数で指定した軸が最も若い軸（0軸）になるように右引数を転置して返す

```
   a =. 2 3 4 $ i. 24
   a
 0  1  2  3
 4  5  6  7
 8  9 10 11

12 13 14 15
16 17 18 19
20 21 22 23
   $a
2 3 4
   0 |: a
 0 12
```

```
 1 13
 2 14
 3 15

 4 16
 5 17
 6 18
 7 19

 8 20
 9 21
10 22
11 23
   $ 0 |: a
3 4 2
   1 |: a
 0  4  8
 1  5  9
 2  6 10
 3  7 11

12 16 20
13 17 21
14 18 22
15 19 23
   $ 1 |: a
2 4 3
   2 |: a
 0  1  2  3
 4  5  6  7
 8  9 10 11

12 13 14 15
16 17 18 19
20 21 22 23
   $ 2 |: a
2 3 4
```

9.27 昇順 (Grade up) /: y 右引数を昇順に並べるインデックスを返す

```
   a =. ? 10 # 10
   a
5 7 4 0 1 2 7 9 3 2
   /:a
3 4 5 9 8 2 0 1 6 7
   (/:a) { a
0 1 2 2 3 4 5 7 7 9
```

9.28 昇順ソート (Sort) x /: y 左引数を右引数で与えた指標にしたがって並べ替えて返す

```
   a =. 10 20 30 40
   a
10 20 30 40
   a /: 3 1 0 2
30 20 40 10

   3 1 0 2 { a
40 20 10 30
   /: 3 1 0 2 { a
2 1 3 0
   (/: 3 1 0 2 { a) { a
30 20 40 10
```

9.29 降順 (Grade down) ¥: y 右引数を降順に並べるインデックスを返す

```
   a =. ? 10 # 10
   a
1 2 9 5 3 9 6 9 0 6
   \:a
2 5 7 6 9 3 4 1 0 8
   (\:a){a
9 9 9 6 6 5 3 2 1 0
```

9.30 降順ソート (Sort) x ¥: y 左引数を右引数で与えた指標にしたがって並べ替えて返す

```
   a =. 10 20 30 40
   a
10 20 30 40
   a \: 3 1 0 2
10 40 20 30

   3 1 0 2 { a
40 20 10 30
   \: 3 1 0 2 { a
0 3 1 2
   (\: 3 1 0 2 { a) {
10 40 20 30
```

9.31 インデックスファースト (index of first) x i. y 右引数の要素の左引数における最初の位置を返す

```
   30 2 10 3 10 20 2 10 3 i. 2 3
1 3
```

9.32 インデックスラスト (Index of Last) x i: y 右引数の要素の左引数における最後の位置を返す

```
   30 2 10 3 10 20 2 10 3 i: 2 3
6 8
```

9.33 レス (Less) x -. y 左引数から右引数の要素を除いたものを返す

```
   30 2 10 3 10 20 2 10 3 -. 2 3
30 10 10 20 10
```

Chapter 10

ファイルシステムを使ったデータ入出力

J 言語は閉じた世界でデータ処理が行われる。ファイルシステムとのデータの読み書きは特別な外部接続プリミティブを使って行うことが出来る。外部アプリケーションとデータを授受するにはテキストファイルで授受することが可能である。

10.1 テキストファイル書き出し (write to file) x 1!:2 ＜y 右引数で指定したファイルに左引数（文字列に限る）を書き出す

例 1） ファイル名 textfile.txt に　文字列　'abcdefg' を書き込むには

```
'abcdefg' 1!:2 <'textfile.txt'
```

とする。ファイル名はシングルクォートで囲み、その前にボックス化のプリミティブ＜をつける。ボックスという表記はＪ特有のものである。(ボックス化は後述の配列処理プリミティブの内、要素の修正とカタログ作成プリミティブで使われる。) ボックス化は書き出し時のファイル名を指定するときの約束であり、必須である。書き込むデータも文字列なのでシングルクォートで囲む。データやファイル名が変数に代入してあればその変数名を指定する。
例 2） ファイル名 textfile2.txt に　数字列　'1 2 3 4 5 6' を書き込むには

```
'1 2 3 4 5 6' 1!:2 <'textfile2.txt'
```

とする。変数名を使うなら下記のようにする。

```
a1 =: '1 2 3 4 5 6'

b2 =: 'textfile2.txt'
```

```
   a1 1!:2 <b2
```

ファイルへの書き込みの特殊形として、スクリーンへの出力がある。書き出すファイル名を数字の 2 にすると、文字列を画面に表示する。

```
   'abc' 1!:2 <2
abc
abc
```

10.2 テキストファイルへの追加書き込み

(append to file) x 1!:3 ＜ y 右引数で指定したファイルに左引数（文字列に限る）を追加で書き足す例 2） ファイル名 textfile2.txt に　数字列　'1 2 3 4 5 6 ' を書き込み、さらに、書き足すには

```
   '1 2 3 4 5 6 ' 1!:2 <'textfile2.txt'
   '1 2 3 4 5 6 ' 1!:3 <'textfile2.txt'
```

10.3 テキストファイル読み込み (read from file) 1!:1 ＜y 右引数で指定したファイルから格納されているデータを全て文字列として読み込む。

ファイル名の指定はシングルクォートで挟み、その前に＜を付ける。
例) 書き込みの例で実行したファイルtextfile.txt とtextfile2.txt から内容を読み込むには

```
   1!:1 <'textfile.txt'
abcdefg
   1!:1 <b2
1 2 3 4 5 6
   1!:1 <'textfile2.txt'
1 2 3 4 5 6 1 2 3 4 5 6
```

となる。

Chapter 11

Ｊ言語特有の補助関数・文字数値変換・実行命令・システム定数・作業ディレクトリの管理

11.1 整数生成 (integers) i. y ０～右引数-1 までの整数を生成する

```
   i. 10
0 1 2 3 4 5 6 7 8 9
   i. _4
3 2 1 0
   i. 2 3
0 1 2
3 4 5
```

11.2 デフォルト書式 (Default Format) ": y 数値で与えた右引数を文字化して返す

```
   1 2 3
1 2 3
   $1 2 3
3
   ": 1 2 3
1 2 3
   $": 1 2 3
5
```

11.3 書式 (Format) x ": y 左引数の書式に従って右引数を文字化して返す

書式は複素数と同じ j で指定する。

```
   $ 6j0 ": 3.1234
6
   6j1 ": 3.1234
   3.1
   $ 6j1 ": 3.1234
6
   6j3 ": 3.1234
 3.123
   $ 6j3 ": 3.1234
6
```

11.4 挿入 (Insert) u/y アイテムの間に動詞 (u) を挿入した演算と同等

```
   +/ i. 10
45
   0+1+2+3+4+5+6+7+8+9
45
   */ i. 10
0
   0*1*2*3*4*5*6*7*8*9
0
   */ 1 + i. 10
3628800
   1 + i. 10
1 2 3 4 5 6 7 8 9 10
   1*2*3*4*5*6*7*8*9*10
3628800
```

11.5 ランク (1) (Rank) u"n y 動詞（u）の作用するランクセルのナンバーを指定する。単項動詞用

```
   a =. i. 2 3 4
   a
 0  1  2  3
 4  5  6  7
```

```
 8  9 10 11

12 13 14 15
16 17 18 19
20 21 22 23
   +/"0 a
 0  1  2  3
 4  5  6  7
 8  9 10 11

12 13 14 15
16 17 18 19
20 21 22 23
   +/"1 a
 6 22 38
54 70 86
   +/"2 a
12 15 18 21
48 51 54 57
   +/"3 a
12 14 16 18
20 22 24 26
28 30 32 34
```

11.6 ランク (2) (Rank) x u ^ n y 動詞（u）の作用するランクセルのナンバーを指定する。両項動詞用

```
   a =. 7 8 9
   b =. i. 2 3
   a
7 8 9
   b
0 1 2
3 4 5
   a,b
7 8 9
0 1 2
3 4 5
   a ,"1 b
7 8 9 0 1 2
7 8 9 3 4 5
   a ,"2 b
```

```
7 8 9
0 1 2
3 4 5
```

11.7 実行 (Do) ”. y 文字列で与えた右引数を実行する

```
   'i. 10'
i. 10
   ". 'i. 10'
0 1 2 3 4 5 6 7 8 9
```

11.8 コメント (comment) NB. NB. 以降はコメント。ラテン語　nota bene の略。英訳　mark well. 注意 (せよ) の意

```
NB. this is comment
```

コメントはプログラムを自作する場合、なるべく詳しく記述すると良い。時間が経つと、論理を忘れてしまうことが多々あり、コメントが残っていないと解読に苦労することになる。

11.9 アルファベット (alphabet) a. ２５６のキャラクタ、アスキー文字のシステム定数

```
   #a.
256
   a. i. '!'
33
   a. i. 'abcde'
97 98 99 100 101
   a. i. 'ABCDE'
65 66 67 68 69
```

11.10 システム定数

コントロール文字として CR 改行　LF　ラインフィードが定義されています。

```
   a. i. CR
13
```

```
   a. i. LF
10
   a. i. CRLF
13 10
```

11.11　作業ディレクトリの管理

テキストファイルで外部と情報交換する為に、作業ディレクトリを作成・指定・表示することが出来ます。ディレクトリ作成は１!:５、ディレクトリ移動は１!:４４、現在のディレクトリ表示は１!:４３　''です。ディレクトリ名はボックス化プリミティブ<を使って指定します。

```
   1!:5 <'c:\working'
1
   1!:44 'c:\working'

   1!:43 ''
'c:\working'
```

Chapter 12

プログラミング仕様

12.1 自作関数の作り方

以下のセクションに記述されているプログラムは jbooklet.ijs にありますので File ー＞ Open で開き、Run ー＞ Window で実行環境へロードしておくと続く記述を読みながら直ぐ試すことが出来ます。

12.2 関数定義の基本構造と変数・定数の取り扱い方

Ｊ言語での関数定義は引数の数で仕様が異なります。引数が一つの場合は関数名を foo とした場合、

```
foo =: 3 : 0
```

引数が２つの場合は関数名を foo とした場合、

```
foo =: 4 : 0
```

として関数定義開始を宣言します。その後、処理を記述し、閉じカッコで定義の終了を宣言します。引数一つの場合は右引数で与え、関数定義の中では y で参照します。引数２つの場合は左引数と右引数で与え左引数を x で参照します。この引数の仕様は、これまで記述されたプリミティブでも数多くありますので、理解出来ると思います。自作関数もプリミティブのように一つ、あるいは２つの引数を持たせることが出来るのです。

変数の宣言は不要です。なぜなら変数は全て配列だからです。変数名はシステム予約語以外の任意の文字列を使うことが出来ます。
関数は引数を取り込み、処理を行い、結果を返すものです。そして、関数定義の終了宣言である閉じカッコ直前に評価された内容がその関数の戻り値になります。引数を必要としない関数も可能です。それは丁度、PASCAL 言語の procedure に当たります。

12.3　平均値を求める関数作成例

平均値の計算を heikin という名前でプログラムする例を示します。まず、File -> New ijs で編集用ウインドウを開き、下記のように関数を記述します。このあと、File -> Save As で適当な名前を付けて保存しておきます。その後、Run-> Window で定義を実行ウインドウ ijx へロードします。

```
heikin_0 =: 3 : 0  NB. 右引数だけの関数定義は =: 3 : 0
a_kosu =. # y  NB. 要素数を求める
a_goukei =. +/ y NB. 合計を求める
a_goukei % a_kosu NB.  平均を計算
NB. かっこ閉じで関数の終了を宣言
)
```

これで、heikin __ 0 という関数が使えるようになります。

```
   heikin 1 2 3 4
2.5
```

計算の途中結果をモニタするために、機能を追加します。「テキストファイルを使ったデータの読み書き」の書き出しの項にファイル名を 2 にすると画面に表示するとありますので、それを使います。

```
   'mojiretu' 1!:2 <2
```

という書式で文字列が表示できます。但し、数値の場合は文字化プリミティブ ": で文字化しておく必要があります。以下のように変更します。

```
heikin_1 =: 3 : 0  NB. 右引数だけの関数定義は =: 3 : 0
a_kosu =. # y  NB. 要素数を求める
(": a_kosu) 1!:2 <2 NB. 個数を表示
a_goukei =. +/ y NB. 合計を求める
(": a_goukei) 1!:2 <2 NB. 合計を表示
a_goukei % a_kosu NB.  平均を計算
NB. かっこ閉じで関数の終了を宣言
)
```

変更を CTRL+S で保存後、Run-> Window でロードし、

```
   heikin_1 1 2 3 4
4
10
2.5
```

これで途中経過が少し分かるようになりました。さらに、項目名をつけて見ましょう。

```
heikin_2 =: 3 : 0  NB. 右引数だけの関数定義は =: 3 : 0
(’与えられたデータ ’,": y) 1!:2 <2 NB. 与えられたデータを表示
a_kosu =. # y  NB. 要素数を求める
(’個数 ’,": a_kosu) 1!:2 <2 NB. 個数を表示
a_goukei =. +/ y NB. 合計を求める
(’合計 ’,": a_goukei) 1!:2 <2 NB. 合計を表示
’平均は’ 1!:2 <2
a_goukei % a_kosu NB.  平均を計算
NB. かっこ閉じで関数の終了を宣言
)
```

変更を CTRL+S で保存後、Run-＞ Window でロードし、

```
   heikin_2 1 2 3 4
与えられたデータ 1 2 3 4
個数 4
合計 10
平均は
2.5
```

これで、だいぶ分かりやすくなりました。

次に右、左に２つの引数を持つ関数の定義法を説明します。例としては先ほど使った画面に文字列を表示する関数を定義してみましょう。左右の２引数の関数定義は =: 4 : 0 で記述します。

```
disp_it =: 4 : 0  NB. 左右の２引数の関数定義は =: 4 : 0
a_komoku =. x NB. 左引数ｘは項目名の文字列
a_val =. y NB. 右引数ｙは表示する数値
a_data =. (a_komoku,’ ’,": a_val) NB. 表示内容の文字列化
a_data 1!:2 <2  NB.  画面に表示
NB. かっこ閉じで関数の終了を宣言
)
```

新しい表示関数を使って、heikin 関数を書き直すと

```
heikin_3 =: 3 : 0  NB. 右引数だけの関数定義は =: 3 : 0
’与えられたデータ ’ disp_it y NB. 与えられたデータを表示
a_kosu =. # y  NB. 要素数を求める
’個数 ’ disp_it a_kosu NB. 個数を表示
a_goukei =. +/ y NB. 合計を求める
’合計 ’ disp_it a_goukei NB. 合計を表示
’平均は’ disp_it a_goukei % a_kosu NB.  平均を計算して表示
NB. かっこ閉じで関数の終了を宣言
)
```

となります。結果は、

```
   heikin_3 1 2 3 4
与えられたデータ 1 2 3 4
個数 4
合計 10
平均は 2.5
```

となります。

式が右から順に評価される規則を積極的に利用してもう少しすっきり書くことも出来ます。

```
heikin_4 =: 3 : 0  NB. 右引数だけの関数定義は =: 3 : 0
'与えられたデータ ' disp_it y NB. 与えられたデータを表示
'個数 ' disp_it # y NB. 個数求めて表示
'合計 ' disp_it +/ y NB. 合計を求めて表示
'平均は' disp_it (+/y) % # y NB.  平均を計算して表示
NB. かっこ閉じで関数の終了を宣言
)
```

となります。結果は、

```
   heikin_4 1 2 3 4
与えられたデータ 1 2 3 4
個数 4
合計 10
平均は 2.5
```

で同じです。

12.4 ディレクトリ管理とファイルシステムの定義例

関数定義の例として、ディレクトリ管理のプリミティブを UNIX 風のコマンドに書き換えてみましょう。

```
mkdir =: 3 : 0 NB. make directory
1!:5 <y.
)

pwd =: 3 : 0 NB. pwd print working directory
1!:43 ''
)

cddir =: 3 : 0 NB. cd change directory
1!:44 y.
```

```
)

rff =: 3 : 0 NB.read one line string file
1!:1 <y.
)

wtf =: 4 : 0 NB.write string data x. to y. file
x. 1!:2 <y.
)

dir =: 3 : 0 NB. directory check
1!:0 <y.
)
```

CTRL-S で保存した後、Run-＞ Window で定義を実行ウインドウ ijx へロードします。 mkdir, pwd, cddir が使えるようになります。UNIX では cd が change directory のコマンドですが、J では cd が後に説明する DLL の処理用語にすでに登録されているので、cddir にしました。

```
   pwd ''
C:\Program Files\j602

   mkdir 'c:\testdir'
1

   cddir 'c:\testdir'

   pwd ''
c:\testdir
```

という具合に操作しやすくなります。

12.5 制御構文 (if文と while 文) 処理の流れを制御する構文は複数ありますが、if文による条件判断処理、while文による繰り返し処理の２つで十分実用的なプログラムを作ることが可能です

プログラムの流れは if 文で変更できます。また、実行の繰り返しは while 文で実現できます。

```
if. A do. B end.
while. A do. B end.
```

A が真なら B を実行という書式です。この 2 つの制御文を使うことで様々な複雑な処理を行うことが出来るようになります。例題として、1 2 3 .. 10 10 20 30 .. 100 100 200 300 .. 1000 までの 30 個の数字を作って見ます。

```
makenumber =: 3 : 0
 start_num =. y NB. get start number from y
 a_result =. '' NB. nil data
 a_counter =. 0 NB. initialize counter
 b_counter =. 0 NB. initialize counter
 c_data =. 0  NB. initialize data
 while. a_counter < 3 do.
  'a_counter' disp_it a_counter
  if. a_counter = 0 do. a_delta =. 1 end.
  if. a_counter = 1 do. a_delta =. 10 end.
  if. a_counter = 2 do. a_delta =. 100 end.
  'a_delta' disp_it a_delta

  while. b_counter < 10 do.
     c_data =. (start_num * a_delta) + a_delta * b_counter
     '' disp_it c_data
     a_result =. a_result , c_data NB. append data to result
     b_counter =. b_counter + 1 NB. increase b_counter
  end.
  b_counter =. 0 NB. reset b_counter
  a_counter =. a_counter + 1   NB. increase a_counter
 end.
a_result NB. return the result
)

   a =. makenumber 1
a_counter 0
a_delta 1
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
a_counter 1
a_delta 10
```

```
 10
 20
 30
 40
 50
 60
 70
 80
 90
 100
a_counter 2
a_delta 100
 100
 200
 300
 400
 500
 600
 700
 800
 900
 1000
   $a
30
   3 10 $ a
  1   2   3   4   5   6   7   8   9   10
 10  20  30  40  50  60  70  80  90  100
100 200 300 400 500 600 700 800 900 1000
```

Chapter 13

フォームとマウスを使ったGUIプログラミング

13.1 メインフォーム作成、実行と終了

ボタンやエディット等のコントロールの無いフォームの例。メインフォームを=: 0 : 0で定義し、そのフォーム名__ runとフォーム名__ closeを =: 3 : 0で下記のように定義すれば空フォームが出来上がります。blankform __ run 0でフォームが画面に表示されフォーム右上の×をクリックするとフォームが閉じます。

```
BLANKFORM=: 0 : 0
pc blankform; pn "テスト用空フォーム";
rem pc is parent creat. pn is parent name set command
)

blankform_run=: 3 : 0
wd BLANKFORM
wd 'pshow;'
)

blankform_close=: 3 : 0
wd 'pclose'
)
```

13.2 ボタンの配置とクリックイベント処理

このフォームにエディットコントロールを配置し、その内容を数字の１にします。先のボタンを「１加算」という名前に変更し、クリックイベントでエディッ

トの内容に＋１してエディット内容を書き変えるようにします。数値は文字化プリミティブ ":" で文字化する必要があります。この値をプログラム内で参照するには、エディット名そのものを参照します。この例ではエディット名は suuji です。

```
BLANKFORM1=: 0 : 0
pc blankform1;pn "加算ボタンのフォーム";
xywh 10 10 44 12;cc kasan button;cn "１加算";
xywh 10 26 44 12;cc suuji edit;
rem pas 6 6;
rem pcenter;
rem form end;
)

blankform1_run=: 3 : 0
wd BLANKFORM1
NB. initialize form here
wd 'set suuji *','1'
wd 'pshow;'
)

blankform1_close=: 3 : 0
wd'pclose'
)

blankform1_kasan_button=: 3 : 0
a_val =. 1 + ". suuji
wd 'set suuji *',":a_val
)
```

13.3 フォームアプリケーション　例題１　ベクトルの平均値を求めるフォーム

平均値を求める自作関数を GUI 化した例を示します。フォームを testform __ run 0 で実行し、データエディットコントロールに数値を入れてから各々のボタンをクリックすると、ボタンクリックで定義された機能が実行されます。先ずはデータの個数を計算するボタンのみを配置したものを示します。

```
TESTFORM=: 0 : 0
pc testform; pn "テスト用フォーム";

xywh 5  5 35 12;cc command1 button;cn " data  ";
xywh 5 19 35 12;cc command2 button;cn " 個数  ";
```

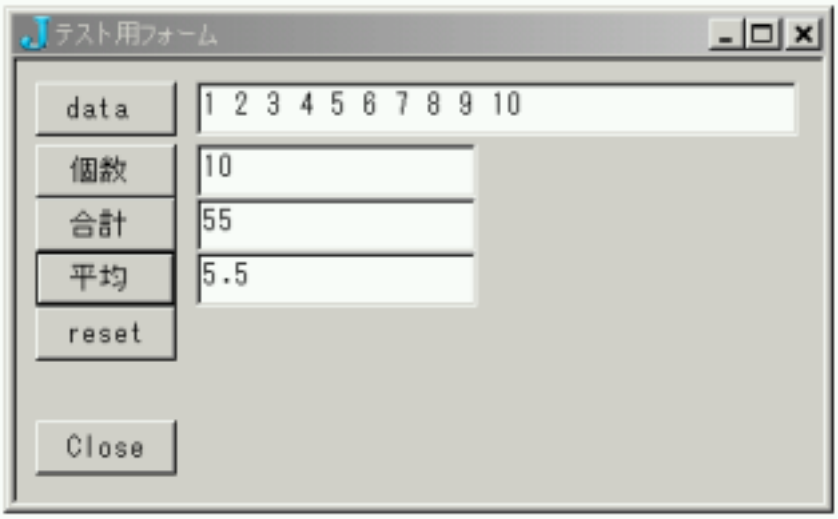

Figure 13.1: test form button and edit

```
xywh 5 55 35 12;cc command5 button;cn " reset ";

xywh 5 80 35 12;cc cancel button;cn "Close";

xywh 45  5 150 12;cc edit1 edit;
xywh 45 19  70 12;cc edit2 edit;

pas 6 6;pcenter;
rem form end;
)

testform_run=: 3 : 0
wd TESTFORM
NB. initialize form here
wd 'set edit1 *',":0
wd 'pshow;'
)

testform_close=: 3 : 0
wd'pclose'
)
```

```
testform_command1_button=: 3 : 0
a_num =. ". edit1 NB. convert content of edit1 to numeric
a_count =. # a_num NB. calculate number of element
wd 'set edit2 *',": a_count
        NB. convert count to character and set it edit2
)

testform_command2_button=: 3 : 0
a_num =. ". edit1 NB. convert content of edit1 to numeric
a_count =. # a_num NB. calculate number of element
wd 'set edit2 *',": a_count NB. convert count to character
                            NB. and set it edit2
)

testform_command5_button=: 3 : 0
wd 'set edit1 *',":0
wd 'set edit2 *',":0
)

testform_cancel_button=: 3 : 0
testform_close''
)
```

今度は、合計と平均値を求めるボタンを追加したものを示します。フォームにはデータ表示、個数表示、合計表示、平均を求めるボタン、フォームを閉じるボタンを配置します。下記ソースには各々コメントをつけてありますので、一行ずつ確認して下さい。Run ー> window でロード後、testform1 __ run 0 で実行します。

```
TESTFORM1=: 0 : 0
pc testform1; pn "テスト用フォーム";

xywh 5  5 35 12;cc command1 button;cn " data  ";
xywh 5 19 35 12;cc command2 button;cn " 個数  ";
xywh 5 31 35 12;cc command3 button;cn " 合計  ";
xywh 5 43 35 12;cc command4 button;cn " 平均  ";
xywh 5 55 35 12;cc command5 button;cn " reset ";

xywh 5 80 35 12;cc cancel button;cn "Close";

xywh 45  5 150 12;cc edit1 edit;
xywh 45 19  70 12;cc edit2 edit;
xywh 45 31  70 12;cc edit3 edit;
```

```
xywh 45 43  70 12;cc edit4 edit;

pas 6 6;pcenter;
rem form end;
)

testform1_run=: 3 : 0
wd TESTFORM1
NB. initialize form here
wd 'set edit1 *',":0
wd 'set edit2 *',":0
wd 'set edit3 *',":0
wd 'set edit4 *',":0
wd 'pshow;'
)

testform1_close=: 3 : 0
wd'pclose'
)

testform1_command1_button=: 3 : 0
a_num =. ". edit1 NB. convert content of edit1 to numeric
a_count =. # a_num NB. calculate number of element
wd 'set edit2 *',": a_count NB. convert count to character and
                            NB.set it edit2
a_sum =. +/ a_num NB. calculate number of element
wd 'set edit3 *',": a_sum NB. convert count to character and
                          NB.set it edit2
)

testform1_command2_button=: 3 : 0
a_num =. ". edit1 NB. convert content of edit1 to numeric
a_count =. # a_num NB. calculate number of element
wd 'set edit2 *',": a_count NB. convert count to character and
                            NB. set it edit2
)

testform1_command3_button=: 3 : 0
a_num =. ". edit1 NB. convert content of edit1 to numeric
a_sum =. +/ a_num NB. calculate number of element
wd 'set edit3 *',": a_sum NB. convert count to character and
                          NB.set it edit2
)
```

```
testform1_command4_button=: 3 : 0
a_count =. ". edit2 NB. calculate number of element
a_sum =. ". edit3 NB. calculate number of element
if. a_count ~: 0 do.
  wd 'set edit4 *',": a_sum % a_count
                      NB. calculate mean ,
                      NB. convert to character and set it edit4
end.
)

testform1_command5_button=: 3 : 0
wd 'set edit1 *',":0
wd 'set edit2 *',":0
wd 'set edit3 *',":0
wd 'set edit4 *',":0
)

testform1_cancel_button=: 3 : 0
testform1_close''
)
```

13.4 フォームアプリケーション　例題２　画像コントロール

ビットマップ画像のロード、色検査、点描、文字描画、画像のセーブの例を示します。グラフィックテスト用フォームはエディットコントロール２つ、ボタン４つ、キャンバス一枚とします。Load ボタンで画像をロードします。右クリックでその場所の座標とカラーコード整数と RGB をエディトに表示します。左クリックで RGB エディトの色で点描します。シフト左クリックで拡大点描。キーボートを押すとその文字をエディトの RGB でマウスポインタの位置に描画します。Save ボタンで画像を保存します。この例でロード、セーブするｂｍｐファイルは２４０ｘ２４０ピクセルに固定されています。グラフィック用のコントロールは isigraph という名前です。このコントロールではマウスイベント、キーボートイベントが処理できます。ｂｍｐファイル処理用のライブラリをあらかじめロードしておく必要があります。また、グラフィックスの縦横比をそろえるために定数 RATIO を定義する必要があります。

```
NB.  graphic sample 0

NB. RATIO =: 0.5 0.5
RATIO =: 0.5 0.446667
```

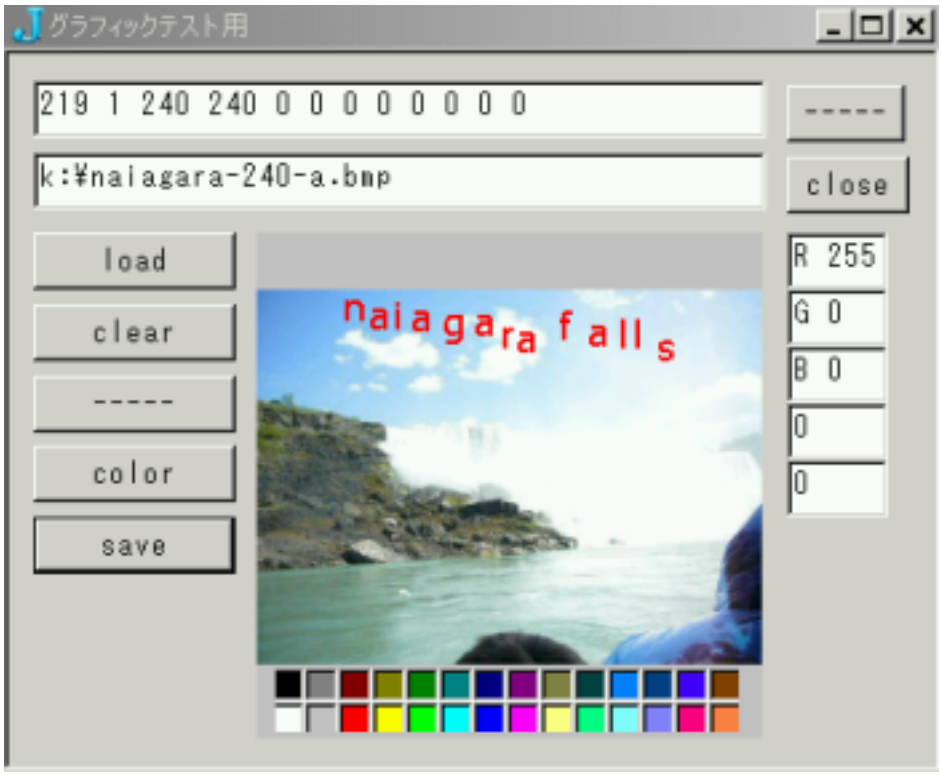

Figure 13.2: test form graphic control

```
NB. system constanct ration of isigraph

load 'trig numeric plot gl2 graph bmp dll'
coinsert 'jgl2'

BMPTEST=: 0 : 0
pc bmptest; pn "グラフィックテスト用";

xywh 185  7  28  12;cc ok button;cn "-----";
xywh 185 22  29  12;cc cancel button;cn "close";
xywh  59 38 120 107;cc canvas isigraph;
xywh   6  6 174  12;cc edit1 edit;
xywh   6 21 174  12;cc edit2 edit;
xywh 185 38  24  12;cc edit3 edit;
xywh 185 50  24  12;cc edit4 edit;
xywh 185 62  24  12;cc edit5 edit;

xywh 185 74  24  12;cc edit6 edit;
xywh 185 86  24  12;cc edit7 edit;

xywh   6 38  48  12;cc cmd1 button;cn "load";
xywh   6 53  48  12;cc cmd2 button;cn "clear";
xywh   6 68  48  12;cc cmd3 button;cn "-----";
xywh   6 83  48  12;cc cmd4 button;cn "color";
xywh   6 98  48  12;cc cmd5 button;cn "save";
pas 6 6;pcenter;
rem form end;
)

bmptest_run=: 3 : 0
wd BMPTEST
NB. initialize form here
wd 'set edit3 *','R 0'
wd 'set edit4 *','G 0'
wd 'set edit5 *','B 0'
wd 'set edit6 *','0'
wd 'set edit7 *','0'
wd 'pshow;'
)

bmptest_close=: 3 : 0
wd'pclose'
)
```

```
bmptest_cancel_button=: 3 : 0
bmptest_close''
)

bmptest_canvas_mmove =: 3 : 0
mouseinfo =: sysdata
wd 'set edit1 *','' NB. erase first
wd 'set edit1 *',": mouseinfo
)

bmptest_canvas_char =: 3 : 0
a_chr =. sysdata
NB. wd 'set edit2 *',a_chr
NB.  wd 'setfocus canvas'
  FONT=: 'Tahoma'
  FONTSIZE=: 12
  glfont FONT,' ',":FONTSIZE
NB. gltext text ; write text
NB. in glfont and gltextcolor at gltextxy
  NB. gltextxy 0 1 { ". edit1
  cx =. 0 { ". edit1
  cy =. 1 { ". edit1
  gltextxy cx,cy
  '' disp_it 0 1 { ". edit1
  glrgb 255 0 0
  if. edit3 ~: '0' do.
     a_r =. ". 2 }. edit3
     a_g =. ". 2 }. edit4
     a_b =. ". 2 }. edit5
     glrgb a_r,a_g,a_b
    end.
  gltextcolor '' NB. 0 0 0 NB. black
  '' disp_it a. i. a_chr
  gltext a_chr
  NB. gltextxy  (cx+FONTSIZE),cy
  NB. glpixel 0 1 { ". edit1
  NB. gltext a_chr  NB. print one more
  glpaint ''  NB. paint buffered data with this command
)

bmptest_cmd1_button =: 3 : 0
NB. load test bmp
NB. bmpfile =. 'bmptest120.bmp'
bmpfile =. wd 'mbopen  "select bmp file with less than
```

```
         240x240 pixels"
         ""  ""  "bmp(*.bmp)|*.bmp" ofn_filemustexist'
ah =. readbmphdr bmpfile
bmpd =. 0 { ah
bmph =. 1 { ah
bmpw =. 2 { ah
wd 'set edit2 *',": ah
if. (bmpw >. bmph) <: 240 do.
 a =: readbmp bmpfile
 glmap MM_RAW
 NB. wd 'setxywh canvas 59 38 ',": RATIO * bmpw,bmph
 pxywh =. 0,0,bmpw,bmph
 glpixels pxywh,,a
 glpaint ''
end.
)

bmptest_cmd2_button =: 3 : 0
NB. clear isigraph
glclear ''
glpaint ''
wd 'set edit2 *',''
)

bmptest_cmd3_button =: 3 : 0
NB. clear isigraph
glclear ''
glpaint ''
wd 'set edit2 *',''
)

bmptest_cmd4_button =: 3 : 0
NB. color selection
a_color =. wd 'mbcolor'
wd 'set edit2 *',": a_color
wd 'set edit3 *','R ',": 0 { ". a_color
wd 'set edit4 *','G ',": 1 { ". a_color
wd 'set edit5 *','B ',": 2 { ". a_color
)

bmptest_cmd5_button =: 3 : 0
NB. save bmp
bmpfile =. wd 'mbsave  "bmp file specify"  ""  ""
                                  "bmp(*.bmp)|*.bmp"
```

```
                                    ofn_filemustexist'
wd 'set edit2 *',bmpfile
b =. glqpixels 0,0,240,240
b =. 240 240 $ b
b writebmp <bmpfile
)

bmptest_ok_button =: 3 : 0
NB. clear isigraph
glclear ''
glpaint ''
wd 'set edit2 *',''
)


bmptest_canvas_mbldown =: 3 : 0
a_ij =. 0 1 { ". edit1
if. edit3 ~: '0' do.
     a_r =. ". 2 }. edit3
     a_g =. ". 2 }. edit4
     a_b =. ". 2 }. edit5
     glrgb a_r,a_g,a_b
end.
glpixel a_ij
a_shift =. 7 { ". edit1 NB.check shift key
if. a_shift = 1 do.
a_i =. 0 { a_ij
a_j =. 1 { a_ij
glpixel (a_i + 1),a_j
glpixel a_i ,a_j + 1
glpixel (a_i + 1),a_j+ 1
end.
a_ctl =. 6 { ". edit1 NB.check shift key
if. a_ctl = 1 do.
a_i =. 0 { a_ij
a_j =. 1 { a_ij
glpixel (a_i + 1),a_j
glpixel a_i ,a_j + 1
glpixel a_i ,a_j - 1
glpixel (a_i -1),a_j
end.

glpaint ''
)
```

```
bmptest_canvas_mbrdown =: 3 : 0
a_ij =. 0 1 { ". edit1
a_rgb =. query_ij a_ij
wd 'set edit2 *',": a_ij,a_rgb,,256 256 256 #: a_rgb
a_data =. ,256 256 256 #: a_rgb
wd 'set edit3 *','R ',": 0 { a_data
wd 'set edit4 *','G ',": 1 { a_data
wd 'set edit5 *','B ',": 2 { a_data
)

query_ij =: 3 : 0 NB. check value at i,j on g2_isigraph
wd 'setfocus canvas' NB. set focus at g2
i =. 0 { y
j =. 1 { y
glqpixels i,j,1,1
)
```

Chapter 14

Ｊ言語でDLLを使う方法 (Windows版)

J 言語はインタプリタですので処理速度には限界があります。高速処理を要求されるルーチンが必要な場合は C や PASCAL 等のコンパイラ言語で DLL を作成すると速度を上げることが出来ます。アルゴリズムを J 言語で試作し、出来上がった時点で、コンパイラ言語で書き換えて DLL にし、高速処理を J 言語の関数に取り込みます。

14.1 DLLの作り方 私はDELPHI (PASCAL COMPILER) での経験がありますので、簡単なサンプルで紹介します

free pascal complier を使って dll を作ってみましょう。以下のファイルを testdll.pp の名前で保存して fpc で読み込み F9 を押して make すると testdll.dll が出来ます。

```
{ library test read from msg.txt and write it twice to msgout.txt}
library testdll;

var
 fd1,fd2:text;
 num1,num2:integer;

procedure read_and_write2;stdcall;
begin
 assign(fd1,'msg.txt');reset(fd1);
 read(fd1,num1);
 close(fd1);
```

```
 assign(fd2,'msgout.txt');rewrite(fd2);
 write(fd2,num1);write(fd2,' ');write(fd2,num1);
 close(fd2);
end;

procedure read_and_write4;stdcall;
begin
 assign(fd1,'msg.txt');reset(fd1);
 read(fd1,num1);
 close(fd1);
 assign(fd2,'msgout.txt');rewrite(fd2);
 write(fd2,num1);write(fd2,' ');write(fd2,num1);write(fd2,' ');
 write(fd2,num1);write(fd2,' ');write(fd2,num1);
 close(fd2);
end;

exports
 read_and_write2 index 1,
 read_and_write4 index 2;

begin
end.
```

14.2 DLL の呼び出し方 J 言語で DLL を呼び出すには外部接続詞 15!:を使います

DLL を使うためのライブラリが付属しているので、load 'dll' でツールをロードします。すると cd (call dll) という関数が使えるようになります。

```
   rff 'msg.txt'
123
   load 'dll'
   '"testdll.dll" read_and_write2 n' cd ''
+---+
| 0 |
+---+
   rff 'msgout.txt'
123 123
   '"testdll.dll" read_and_write4 n' cd ''
+---+
| 0 |
+---+
   rff 'msgout.txt'
```

```
123 123 123 123
```

Chapter 15

Epilogue

Ｊ言語は使えば使うほどその威力に驚かされます。その強力な「配列処理」とスタック型記述仕様は、Ｊ言語の高い生産性を可能にしています。自分の考え・意図が容易に記述出来きることは心地よく、インタプリタなので直ちにプログラムが実行出来、結果を見て必要な訂正を加えることも容易です。この記述と試行錯誤のスピードが速いことが、問題解決までのスピードアップにつながります。私は本書に記した機能を使って十分実用的なプログラミングをすることが出来ました。本書とＪ言語例題集（未発表）は、ＢＡＳＩＣやC，ＰＡＳＣＡＬ言語でプログラムを作っている中学、高校、大学、大学院、そして社会人の人々にＪ言語の案内をするために作りました。Ｊ言語はＢＡＳＩＣやＰＡＳＣＡＬとどこが違っていて、どんなところが便利なのか、その要点を伝えたいと思ったのです。Ｊ言語で一体何が出来るのか、オリジナルプログラムはどう作ればいいのか、このレファレンスが役立ては幸いです。ひとりでも多くの方がＪ言語を活用できるようになることを願っています。