ICOT Technical Memorandom: TM-1138

A'UM-90 言語仕様

大西　弘一、柳田　伸二、
丹下　利雄（日電）

December, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191 ~5
Telex ICOT J32964

Institute for New Generation Computer Technology

A'UM-90 言語マニュアル

第1.00版 (1991年11月11日)

本マニュアルの構成

本マニュアルは、以下に示す2部から構成されている。

第一部 A'UM-90 言語仕様

A'UM-90 プログラミング言語のモデル、構文およびその意味を示したものである。

第二部 PAS (Parallel A'Um System) 参照マニュアル

A'UM-90 並列版処理系である PAS にあらかじめ組み込まれたメソッドの機能に関する参照マニュアルである。

A'UM-90 言語仕様

第1.3版

Sep. 18. 1991

Table of Contents

1 はじめに 1

2 モデル 3

2.1 オブジェクト 3
2.2 メッセージとメッセージID 3
2.3 ストリーム 4
2.3.1 ストリームの接続 4
2.3.2 順序なし合流 5
2.3.3 順序つき合流 5
2.3.4 方向 5
2.3.5 メッセージの送信 5
2.3.6 ストリームの閉鎖 6
2.3.7 ストリームの排水 6

3 字句要素 7

3.1 識別子 7
3.2 変数名 7
3.3 予約語 7
3.4 コメント 7

4 組込データ 9

4.1 シンボル 9
4.2 数値 9
4.2.1 整数 9
4.2.2 文字定数 9
4.2.3 浮動小数点数 10
4.3 真偽値 10
4.4 文字列 11
4.5 リスト 11
4.6 配列 12

5 メッセージ 13

6 変数 15

6.1 通常変数 15
6.2 無名変数 16

6.3 疑似変数 17

7 ストリーム操作 19

7.1 ストリームの接続 19
7.1.1 接続 19
7.1.2 連結 19
7.2 メッセージの送信 20
7.3 ストリームの閉鎖 20
7.4 ストリームの排水 21

8 スロット操作 23

8.1 入力端ストリームの参照 23
8.2 入力端ストリームの更新 23
8.3 出力端ストリームの参照 23
8.4 出力端ストリームの更新 24
8.5 対スロット更新 24

9 式 25

9.1 出力端式 25
9.1.1 一次式 26
9.1.2 負号 27
9.1.3 後置演算子 27
9.1.4 前置演算子 27
9.1.5 乗除演算子 28
9.1.6 加減演算子 28
9.1.7 シフト演算子 28
9.1.8 関係演算子 29
9.1.9 等価演算子 29
9.1.10 and 演算子 30
9.1.11 xor 演算子 30
9.1.12 or 演算子 30
9.2 入力端式 30
9.2.1 Who 式 31

10 アクション 33

10.1 自動閉鎖と自動排水 33
10.2 自己送信 34

11 メソッド定義 35

11.1 セレクタ 35
11.2 後継指示子 36

12 クラス定義 37

12.1 継承 38

12.2 ポラタイルクラス定義 38

13 プログラム 41

1 はじめに

本資料は、A'UM-90 処理系βリリースが提供する A'UM 言語の仕様である。

本資料の構成は以下の通りである。

1. はじめに
2. モデル
3. 字句要素
4. 基本データ
5. メッセージ
6. 変数
7. ストリーム操作
8. スロット操作
9. 式
10. アクション
11. メソッド定義
12. クラス定義
13. プログラム

本資料中の構文の表記には以下の記法を用いている。

ある構文要素の名前で始まる行の次の行に、その要素の構成が示される。次の例では、「配列」と「配列要素並び」の構成が示されている。

```
配列
        '{' [ 配列要素並び ] '}'
配列要素並び
        式 [ { ',' 式 } ]
```

'{'のように引用符で括られた記号は構成要素の一部である。一方、引用符で括られていない記号は構文表記の一部であり、以下の意味を持つ。

- [ ] で囲まれている部分は 0 回または 1 回現れる。
- { } で囲まれている部分は 1 回以上の任意の回数繰り返し現れることがある。

2 モデル

A'UM-90 の計算モデルは、並列オブジェクト指向モデルの一種であり、オブジェクト、メッセージ、およびストリームの3種の要素で構成されている。

このモデルでは、各オブジェクトの計算は、並行して行われる。オブジェクトは他のオブジェクトにメッセージを送ることができる。このとき、メッセージが経由する通信路がストリームである。

すべてのオブジェクトはある一つのクラスに属している。クラスは、それに所属するオブジェクトの振舞いを規定する。クラスは他のクラスを継承することができ、一つのクラスが複数のクラスを同時に継承すること（多重継承）ができる。

2.1 オブジェクト

オブジェクトはメッセージを受けとるためのストリーム（インターフェースストリーム）1本と、内部状態として0個以上のスロットを持つ。オブジェクトは、インターフェースストリームを介してメッセージを受けとって、そのメッセージに対応するメソッドを実行する。オブジェクトは必ずある一つのクラスに属し、オブジェクトの振舞いはそのクラスによって決まる。

スロットは、ストリームを保持する。スロットはメソッドによってのみアクセスされる。そのスロットの持ち主であるオブジェクトでも、スロットにアクセスするには自分自身にメッセージを送らなければならない。スロットは保持するストリームの方向によって入力端スロットと出力端スロットのどちらかとして宣言される。

メソッドは、メッセージの送信、オブジェクトの生成、およびストリームの接続の3種類の処理で構成される。これらの処理は並行に実行される。オブジェクトは複数のメソッドを並行に実行する。

クラスは、そのクラスに属するオブジェクトの、スロットの個数、名前、方向、メッセージとメソッドの対応、およびメソッドの定義を定める。クラスは0個以上の他のクラスを継承することができる。あるクラスを継承するクラスは、継承されたクラスのすべてのスロットとメソッドを継承する。ただし、継承されたスロットには継承されたメソッドによらなければアクセスできない。

2.2 メッセージとメッセージID

メッセージは、メッセージ名と0個以上の引数で構成される。引数として保持されるのはストリームである。メッセージの種類は、メッセージ名、引数の個数、および各引数の方向によって区別される。これらの項目をメッ

セージ ID という。メッセージによって起動されるメソッドは、そのメッセージのメッセージ ID によって決まる。

これ以降、メッセージ ID を表すために以下の記法を用いる。

メッセージ ID
　　メッセージ名 ['/' 方向]

方向
　　'+'
　　'-'

方向の順序は引数の順序に対応し、'+'と'-'はそれぞれ出力端と入力端の引数を表す。

以下に、メッセージとメッセージ ID の対応の例を示す。

例:

```
:foo(3, ^N)      ':foo/+-'

:bar             ':bar'
```

2.3 ストリーム

ストリームは、オブジェクト間の通信における通信路である。ストリームは基本的にはメッセージキューとして働き、単方向の非同期通信に用いられる。

ストリームに対して可能な操作は、接続、送信、閉鎖、排水、連結の 5 種類である。また、ストリームへの参照には 2 つのモードがある。各操作の操作対象は、特定のモードの参照でなければならない。このモードをストリームの方向と呼んでいる。

2.3.1 ストリームの接続

ストリームは他のストリームまたはオブジェクトに接続することができる。ストリームに送られたメッセージは、そのストリームが接続されるまではそのストリームに留まり、ストリームが接続されていれば、その接続先に送られる。一度接続されたストリームの接続先を変えることはできない。

複数のストリームの合流を形成する方法が2通りある。一方の合流を順序なしの合流、もう一方を順序つき合流と呼ぶ。

2.3.2 順序なし合流

順序なしの合流においては、複数のストリームが対等に一本のストリームに接続されている。上流のあるストリームからのメッセージは、そのストリームにおける順序を保ったまま他のストリームからのメッセージと合流する。この際、異なる上流のストリームからのメッセージ間の順序がどう決まるかについては、何も保証がない。

2.3.3 順序つき合流

順序つきの合流においては、複数のストリームが指定された順序で一本のストリームに接続される。ある上流のストリームが接続されると、そのストリームに送られるすべてのメッセージが接続先に流れ込んでから、すなわち、このストリームが閉鎖されてから、次の上流のストリームが接続される。

2.3.4 方向

ストリームへの参照は入力端と出力端の2種類に分けられる。一つのストリームに対する参照はいくつあってもよいが、そのうち入力端は一つだけでなければならない。

ストリームを接続する際、そのストリームへの参照は入力端でなければならない。したがって、一つのストリームは一度だけしか接続できない。このことにより、同じストリームに送られたメッセージは同じオブジェクトに届くことが保証されている。

2.3.5 メッセージの送信

ストリームには、メッセージを送信することができる。送信の宛先となるストリームの方向は、出力端でなければならない。ストリームに送られたメッセージは、宛先のストリームが接続されてから、その接続先に送られる。ストリームが受けとる複数のメッセージは、受けとった順序にしたがって接続先に送られる。ストリームが生成されてから、閉鎖されるまでの間に、任意の個数のメッセージをストリームに送ることができる。

2.3.6 ストリームの閉鎖

あるストリームの出力端としての参照がすべてなくなったとき、そのストリームは閉鎖されたという。閉鎖されたストリームには、それ以降メッセージが送られることはありえない。

出力端として参照されているストリームとは、以下のものを指す。

- 出力端変数の値
- 文字列、リスト、および配列の要素
- 出力端スロット
- 出力端として参照されているストリームの入力端と接続されているストリーム

2.3.7 ストリームの排水

排水溝オブジェクトにストリームを接続することを、ストリームの排水という。排水溝オブジェクトはメッセージを受けとると、その引数のうちの入力端を排水し、出力端を閉鎖する。

ストリームを排水するのは、通常、そのストリームに送られたメッセージを受けとるのに適当なオブジェクトが存在せず、それらのメッセージを処分したい場合である。

3 字句要素

3.1 識別子

識別子には以下の二種類がある。

- 英小文字から始まり、英大小文字、数字、および下線 ('_') で構成される文字列
- 両端を引用符 ('') で囲まれた文字列。途中に含まれる ''は直前にバックスラッシュ ('\') を置いて '\''と表す。

3.2 変数名

変数名は英大文字で始まる、英大小文字、数字および '_' (下線) の並びである。

3.3 予約語

'class', 'end', 'super', 'in', 'out', 'car', 'cdr', 'and', 'or', 'xor', 'not', 'class_of', 'mod', ''true', ''false'

上記の識別子はあらかじめ用途が予約されており、引用符で囲わない限り通常の識別子として用いることはできない。

3.4 コメント

'%'から行末まではコメントとして扱われる。ただし、文字列の中に '%'があっても、そこからコメントが始まることはない。

例:

```
% コメントの例
:foo(^X) ->
        X :bar.      % どこから始めてもよい
```

4 組込データ

基本的なデータには以下に示すものがある。

- 単純データ
  - シンボル
  - 整数
  - 単精度浮動小数点数
  - 真偽値
- 組込データ
  - 単純データ
  - 倍精度浮動小数点数
  - 文字列
  - リスト
  - 配列
  - メッセージオブジェクト

4.1 シンボル

シンボルは識別子で表される。引用符で囲まれた識別子の場合、両端の引用符はシンボル名の一部ではない。たとえば、'abc'と''abc''は同じシンボルを表す。

4.2 数値

- 数値
  - 整数
  - 浮動小数点数

4.2.1 整数

このシステムで扱える整数の範囲は、-536870912 (-2^{30}) から 536870911 ($2^{30}-1$) までである。

4.2.2 文字定数

'&'から始まる文字列によって、文字に対応する文字コードを表すことができる。このシステムではこの表記でASCII 文字に対する ASCII コードを表すことができる。

'&'の後に'\'以外の英文字が続く場合、その表記はその文字に対応するASCIIコードを表す。例えば'&A'は、文字'A'に対応するASCIIコードを表し、'& 'は文字' '(空白)に対応するコードを表す。

その他特別な場合を以下に示す。

'&\n' 改行コード。

'&\t' タブコード。

'&\0' ヌルコード。

'&\\' 文字'\'に対応するコード。

'&\ddd' (dddが3桁の8進数の時)その8進数が表す値

4.2.3 浮動小数点数

浮動小数点数
- 単精度浮動小数点数
- 倍精度浮動小数点数

単精度浮動小数点数は仮数部22ビット(符号1ビット+絶対値21ビット)、指数部7ビットを持つ。

倍精度浮動小数点数の詳細は処理系に依存するが、単精度より少なくはないビット数を持つ。

単精度は単純データであるが、倍精度は単純データではないので注意が必要である。

浮動小数点数は、'.'(小数点)を含む数字並びか、または仮数を表す上記の数字並びの後ろに文字'E'または'e'をはさんで指数を表す数字並びが続く文字列で表される。

例:

'3.141592'

'.25' 0.25を表す。0.25と書いても同じ。

'6.63e-34'
6.63かける10のマイナス34乗を表す。

4.3 真偽値

```
真偽値
        'true
        'false
```

真偽値に属するオブジェクトは上記の二つだけである。

4.4 文字列

文字列は、'"'（二重引用符）で囲まれた任意の文字列である。文字列に含まれる二重引用符は、直前に'\'（バックスラッシュ）を置いて'\"'と表す。

4.5 リスト

リストを表す構文は以下の通りである。

```
リスト
        '[' リスト要素並び [ リスト cdr 部 ] ']'

リスト要素並び
        式 [ { ',' 式 } ]

リスト cdr 部
        '|' 式
```

'リスト cdr 部'がない場合、そのリストの末尾の cdr 部の値は'[]'という印字名を持つシンボルになる。このシンボルを空リストという。空リストを表す次の構文が用意されている。

```
空リスト
        '[' ']'
```

リストの要素、および cdr 部には入力端、出力端のどちらでも保持できる。これらの値に個別にアクセスするための組み込みメソッドが用意されているが、その際には方向を指定する必要がある。値の更新の際には、元の値と異なる方向の値を書き込んでもよい。一方、値を参照する際に指定された方向が、実際の値の方向と異なる場合、実行時にエラーが発生する。

4.6 配列

配列を表す構文を以下に示す。

```
配列
        '{' [ 配列要素並び ] '}'
配列要素並び
        式 [ { ',' 式 } ]
```

配列の各要素は出力端でも入力端でもよい。配列の要素を参照／更新する時には、方向を指定する必要がある。要素の更新の際には、元の値と異なる方向の値を書き込んでもよいが、要素を参照する場合には指定した方向と実際に参照される値の方向が違っていると、実行時にエラーを生じる。

5 メッセージ

メッセージは次の構文を持つ。

```
メッセージ
        ':' 単純データ
        ':' [シンボル] '(' メッセージ引数並び ')'

メッセージ引数並び
        式 [ { ',' 式 } ]
```

メッセージは名前を持ち、さらにいくつかのストリームへの参照を引数として持つことができる。メッセージをオブジェクトに送ることによって、そのオブジェクトにメソッドを実行させることができる。

6 変数

6.1 通常変数

通常の変数はストリームへの参照を保持する。ストリームへの参照は、方向によって入力端と出力端の2種類に分けられる。これにしたがって、変数にも入力端変数と出力端変数の2種類がある。

入力端変数
　　'^' 変数名

出力端変数
　　変数名
　　変数名 '$' 整数

変数の通用範囲（スコープ）は、通常のクラスの一メソッド定義内である。

一つのスコープに同じ変数名を持つ入力端変数と出力端変数がある場合、それらは基本的には同じストリームへの方向の違う参照を表す。

同じ変数名を持つ出力端変数が複数ある場合、それらは、一本のストリームに合流する複数のストリームの出力端を表す (see Section 7.1.2 [ストリームの接続], page 15)。これらと同じ名前を持つ入力端変数は、合流先のストリームの入力端を表す。

出力端変数が整数を含む場合は、その整数の値はストリームの合流における順序を表す。数値が小さいストリームほど先に接続される。数字列を含まない出力端変数は、順序1として扱われる。また、同じ順序のアウトレット変数が複数ある場合、それらは、順序なしの合流を形成する。

次の例で、2回現れている'X'は、合流する2本のストリームのアウトレットを表しており、合流先のストリームの入力端が'^X'によって表されている。それぞれの'X'に送られるメッセージ':car/-'と':setcdr/+'はどちらも配列'{a, b}'に届くが、その到着順序はこのプログラムでは指定されない。

例：

```
X :car(^Y),
{a, b} = ^X,
X :setcar(c)
```

次の例では、'X'と'X$2'が順序つきの合流を形成する。すなわち、'X$2'が表すストリームは、'X'が閉鎖さ

れてから、合流先に接続される。その結果、メッセージ':setcar/+'はメッセージ':car/-'よりも先に配列'{a, b}'に到着する。

例:

```
X$2 :car(^Y),
{a, b} = ^X,
X :setcar(c)
```

同じ変数名を持つ入力端変数は、原則として2回以上現れてはならない。2回以上現れてもよいのは、各入力端変数が、同じボラタイルクラスの異なるメソッド定義中にある場合である。

例を示すため、次のようなプログラムを考える。クラスcがメソッドpを持っている。pはボラタイルクラスv1を持つ。v1はメソッドqとrを持つ。qはボラタイルクラスv2を持つ。v2はメソッドsとtを持つ。図で示すと次のようになる。

```
c :p
    v1 :q
         v2 :s
            :t
       :r
```

'^X'はqとrに同時に現れてよい。sとt、あるいはsとrという組み合わせでもよい。最後の例では、sに現れる変数は、同時にqの中に現れていると見なしている。

しかし、qとtに'^X'が同時に現れる場合、それは正しいプログラムではない。二つの'^X'はどちらもqの中に現れていることになるからである。qとsや、pとtなどの場合も同様である。

'_'一文字からなる変数名を持つ変数は無名変数と呼ばれ、特別な用途を持ち、特別な扱いを受ける。

'$'の後ろに小文字で始まる名前(識別子)が続く表記は、一次式であり、疑似変数と呼ばれる。疑似変数は、システムが提供する特別な機能へのアクセス手段として用意される。

6.2 無名変数

無名変数は、使われない入力端や出力端を処分するために用いられる。同じスコープ内に複数の入力端無名変数があってもよい。

入力端無名変数と接続された出力端は閉鎖される。出力端無名変数に接続された入力端は排水される。

無名変数は例えばメソッド定義のセレクタの引数として現れて、そのメソッド内で使われない引数を安全に処分するために用いられる。

例：

```
:foo(^X, ^_) ->                % 第二引数は使われず、直ちに閉鎖される。
    X :bar.
```

6.3 疑似変数

'$self'は、メソッドを実行しているオブジェクトのインターフェースストリームに接続されるストリームの出力端を表す。一メソッドの中に複数回'$self'が現れるとき、それらが表すストリームは出現順序にしたがって、インターフェースストリームへの順序つきの合流を形成する。つまり、先に出現する'$self'に送られるメッセージは、それ以降の出現に送られるメッセージよりも先に到着する。

7 ストリーム操作

7.1 ストリームの接続

接続式
　　出力端式 '=' 入力端式
　　出力端式 '\' 入力端式

7.1.1 接続

出力端式と入力端式の間に等号 ('=') が置かれた表記は出力端式であり、ストリームの接続を表す。左辺が表す出力端に、右辺が表す入力端が接続される。この式の値は、左辺が表す出力端である。

例：

```
3 = ^X = ^Y
```

この例では、ストリームXとYはともに整数3につながるストリームに接続され、XとYは順序なしの合流を形成する。

7.1.2 連結

出力端式と入力端式の間にバックスラッシュ ('\') が置かれた表記は出力端式であり、ストリームの連結を表す。左辺が表す出力端に、右辺が表す入力端が接続される。この式は値として新たなストリームの出力端を返す。このストリームは右辺のストリームが閉鎖された後に左辺に接続される。つまり、右辺のストリームと値として返されるストリームは順序つきの合流を形成する。

例：

```
3 \ ^X = ^Y
```

この例では、ストリームXは整数3につながるストリームに接続される。Yは式'3 \ ^X'の値であるストリームに接続されるので、XとYが順序なしの合流を形成する。すなわち、Yに送られたメッセージは、Xが閉鎖されてから3に届く。より多くのストリームを順序つきで合流させるには、次の例のように記述する。

例:

```
3 \ ^X \ ^Y \ ^Z = ^W
```

この例ではストリーム X, Y, Z, W はこの順で、3 につながるストリームに合流する。

7.2 メッセージの送信

送信式
　　後置式 メッセージ

後置式の後ろにメッセージが続く表記は後置式であり、メッセージの送信を表す。後置式の表す出力端にメッセージが送信される。式の値は後置式の表す出力端である。

例:

次の例では、メッセージ':foo', ':bar', ':noo'がこの順に 'X'の表す出力端に送られる。

```
X :foo :bar :noo
```

次の例では、ストリーム'Second'の入力端は、メッセージ':bar'を受けとったストリーム'First'の出力端に接続される。その結果、'Second'に送られるメッセージ':foo'は、':bar'の到着するオブジェクトと同じオブジェクトに、':bar'よりも遅く届く。

```
Second :foo,
First :bar = ^Second,
```

7.3 ストリームの閉鎖

閉鎖は、ほとんどの場合自動的に行われる。明示的に閉鎖を記述する方法には次の二つがある。

例:

無名変数を接続する。次の例では、'X'が無名の入力端と接続されることにより、閉鎖される。
(see Section 6.2 [無名変数], page 13)

```
X = ~_
```

閉鎖しようとする出力端を独立したアクションとすることによって、自動閉鎖を起こさせる。次の例では、'X'が単独でアクションとなっているので、自動閉鎖の対象になる。
(see Section 10.1 [自動閉鎖と自動排水], page 27)

```
:foo(~X) -> X.
```

7.4 ストリームの排水

排水も、閉鎖同様、ほとんどの場合自動的に行われる。明示的に排水を記述する方法には、次の三つがある。

例:

無名変数と接続する。次の例では、'~X'が無名の出力端に接続されることにより、排水される。
(see Section 6.2 [無名変数], page 13)

```
_ = ~X
```

排水しようとする入力端を独立したアクションとすることによって、自動排水を起こさせる。次の例では、'~X'が単独でアクションとなっているので、自動排水の対象になる。
(see Section 10.1 [自動閉鎖と自動排水], page 27)

```
:bar(X) -> ~X.
```

排水しようとする疑似変数'$self'と接続する (see Section 6.3 [Pseudo Variables], page 14)。

```
$self = ~X
```

8 スロット操作

スロットは保持するストリームの方向によって入力端スロットと出力端スロットの2種類に分けられる。

8.1 入力端ストリームの参照

入力端スロット参照
'@' 識別子

単価記号('@')に識別子が続く表記は入力端スロットの参照を表す。識別子は入力端スロット名として宣言されている必要がある。入力端スロットは、一度参照されると自動的に未設定の状態になる。未設定の入力端スロットが参照されると、実行時にエラーを生じる。

8.2 入力端ストリームの更新

入力端スロット更新
'@' 識別子 '=' 入力端式

単価記号('@')の後ろに識別子が並び、さらにその後ろに等号('=')と入力端式が続く表記は、アクションであり、入力端スロットの更新を表す。'@'の後ろの識別子は、入力端スロット名として宣言されていなければならない。この式によって、'@'の後ろの識別子で指定されたインレットスロットの値が、'='の右側の入力端式が表す入力端で更新される。更新前の値は排水される。

例:

右辺のWho式の値の入力端が右辺の入力端スロット'@bar'の値として設定される。

```
@bar = (X > 3) ?
```

8.3 出力端ストリームの参照

出力端スロット参照
'!' 識別子

感嘆符 ('!') に識別子が続く表記は出力端スロットへの参照を表す。識別子は出力端スロット名として宣言されている必要がある。

8.4 出力端ストリームの更新

出力端スロット更新
　　出力端式 '=' '!' 識別子

出力端式の後ろに等号 ('='), 感嘆符 ('!'), および識別子が続く表記は出力端式であり、出力端スロットの更新を表す。最後の識別子は出力端スロット名として宣言されたものでなければならない。右辺の出力端スロットの値が左辺の出力端に置き換えられる。式の値は左辺が表す出力端である。

8.5 対スロット更新

対スロット更新
　　'@' 識別子 '=' '!' 識別子

等号 '=' の左辺に、'@' を前に伴う識別子があり、一方右辺に '!' を前に伴う識別子がある表記は、アクションであり、二つのスロットの更新を表す。左辺の識別子は入力端スロット名として、また右辺のはアウトレットスロット名として宣言されていなければならない。この式によって、新しいストリームが 1 本生成され、その入力端が左辺で指定される入力端スロットの値として設定される。一方、新しいストリームの出力端は、右辺で指定される出力端スロットの値として設定される。更新前の値はそれぞれ閉鎖、排水される。

例：

```
@foo = !bar
```

9 式

式は、出力端または入力端のどちらかを表す。

式
　出力端式
　入力端式

9.1 出力端式

出力端式
　*or*式
　接続式
　連結式
　出力端スロット更新
　出力端式 '=' 後置式 '[' 出力端式 ']'
　出力端式 '=' car 前置式
　出力端式 '=' cdr 前置式

接続式、連結式については see Section 7.1.2 [ストリームの接続], page 15

出力端スロット更新については see Section 8.4 [出力端スロットの更新], page 19

出力端式の後ろに'='が続き、さらにその後ろに後置式と、鍵括弧'□'で囲まれた出力端式が続く表記は出力端式であり、メッセージID'set/++'を持つメッセージの送信を表す。メッセージの宛先は右辺の後置式が表すストリームであり、メッセージの第一引数は右辺の出力端式、第二引数は左辺の出力端式である。式の値は左辺が表す出力端である。

例：

```
X = V[I]
```

は、式の値を除けば次の式と等価である。

```
V :set(I, X)
```

出力端式の後ろに等号、car、および前置式が続く表記は、出力端式であり、メッセージID'setcar/+'を持つメッセージの送信を表す。また、carの代わりにcdrが並ぶ表記も出力端式であり、メッセージID

'setcdr/+' を持つメッセージの送信を表す。どちらの場合も、メッセージの宛先は右辺の前置式であり、メッセージの第一引数は左辺のアウトレットである。式の値は左辺が表す出力端である。

例：

```
X = car L, X = cdr L
```

は、式の値を除き、次のプログラムと等価である。

```
L :setcar(X), L :setcdr(X)
```

9.1.1 一次式

一次式
　基本データ
　出力端変数
　出力端スロット参照
　'$' 識別子
　'#' 識別子
　'(' 出力端式 ')'

一次式には基本データ、出力端変数、出力端スロット、疑似変数、インスタンス生成式、および括弧で囲まれた式がある。

基本データについては see Chapter 4 [基本データ], page 7

出力端変数については see Chapter 6 [変数], page 12

出力端スロット参照については see Section 8.3 [出力端スロットの参照], page 18

ドル記号 ('$') に識別子が続く表記は疑似変数を表す。識別子は sink, self, systemのどれかでなければならない。疑似変数の振舞いは、各識別子ごとに様々である (see Section 6.3 [疑似変数], page 14)。

シャープサイン ('#') に識別子が続く表記はインスタンス生成を表す。識別子はクラス名でなければならない。そのクラスの新しいインスタンスオブジェクトが生成され、そのオブジェクトのインターフェースストリームの出力端がこの式の値となる。

括弧 ('( )') で囲まれた式は、括弧がない場合とまったく同じものを表す。

9.1.2 負号

```
負号式
        一次式
        '-' 一次式
```

負号('-')は二つ以上並ぶことはない。ただし、一次式が数値の場合には数値が'-'を含むため、見かけ上負号が連続するように見えることがある。メッセージID':minus/-'を持つメッセージが送信される。宛先は右側の一次式が表す出力端である。第一引数は新しいストリームの入力端で、このストリームの出力端が負号式の値となる。

9.1.3 後置演算子

```
後置式
        負号式
        送信式
        後置式 メッセージ
        後置式 '[' 出力端式 ']'
```

送信式はメッセージの送信を表す (see Section 7.2 [メッセージの送信], page 16)。

後置式の後ろに、鍵括弧('[]')で囲まれた出力端式が続く表記は後置式であり、通常、添字によって指定された、配列の要素の参照を表す。実際にはメッセージID'elt/+-'を持つメッセージが送信される。宛先は後置式であり、第一引数は'[]'の中の出力端式が表す出力端である。第二引数は新しいストリームの入力端であり、そのストリームの出力端が式の値となる。

9.1.4 前置演算子

```
前置式
        後置式
        前置演算子 前置式
前置演算子
        以下に示すものがある

        not car cdr classof
```

前置演算子は右から左に結合する。

どの前置演算子においても、メッセージが送信される。宛先は右側の前置式で、第一引数は新しいストリーム

の入力端である。このストリームのアウトレットが式の値となる。メッセージIDは、notでは':not/-'、carでは':car/-'、cdrでは':cdr/-'、classofでは':classof/-'である。

9.1.5 乗除演算子

乗除算式
　　前置式
　　乗除算式 '*' 前置式
　　乗除算式 '/' 前置式

乗除演算子は左から右に結合する。

演算子'*'についてはメッセージID':mul/+-'を持つメッセージが、また演算子'/'については、':div/+-'を持つメッセージが送信される。どちらの演算子についても、宛先は右側の乗除算式が表すストリームで、第一引数は左側の前置式が表すストリームである。第二引数は新しいストリームのインレットである。このストリームの出力端が式の値となる。

9.1.6 加減演算子

加減算式
　　乗除算式
　　加減算式 '+' 乗除算式
　　加減算式 '-' 乗除算式

加減演算子は左から右に結合する。

演算子'+'に対してはメッセージID':add/+-'を持つメッセージが、演算子'-'に対しては':sub/+-'を持つメッセージが送信される。どちらの演算子についても、宛先は右側の加減算式が表すストリームで、第一引数は左側の乗除算式が表すストリームである。第二引数は新しいストリームの入力端である。このストリームの出力端が式の値となる。

9.1.7 シフト演算子

シフト式
　　加減算式
　　シフト式 '<<' 加減算式
　　シフト式 '>>' 加減算式

シフト演算子は左から右に結合する。

演算子'<<'に対してはメッセージID':shtl/+-'を持つメッセージが、演算子'>>'に対しては':shtr/+-'を持つメッセージが送信される。どちらの演算子についても、宛先は右側のシフト式が表すストリームで、第一引数は左側の加減算式が表すストリームである。第二引数は新しいストリームのインレットである。このストリームの出力端が式の値となる。

9.1.8 関係演算子

関係式
　　シフト式
　　シフト式 関係演算子 シフト式
関係演算子
　　以下に示すものがある

　　'>' '<' '>=' '=<'

関係演算子は結合しない。

どの関係演算子においても、メッセージが送信される。メッセージIDは、'=<', '>'では':gt/+-'、'>=', '<'では':ge/+-'である。

'>'と'>='では、宛先は右側のシフト式で、第一引数は左側のシフト式である。'<'と'=<'については、宛先は左側のシフト式で、第一引数が右側のシフト式になる。どちらの場合も第二引数は新しいストリームのインレットであり、このストリームの出力端が式の値となる。

9.1.9 等価演算子

等価式
　　関係式
　　関係式 等価演算子 関係式
等価演算子
　　以下に示すものがある

　　'==' '!='

等価演算子には'=='と'!='がある。等価演算子は結合しない。'=='に対してはメッセージID'eq/+-'を持つメッセージが、'!='に対しては'neq/+-'を持つメッセージが送信される。宛先は左辺が表すストリームで、第一

引数は右辺の表すストリームである。第二引数は新しいストリームの入力端で、このストリームの出力端が式の値となる。

9.1.10 and 演算子

*And*式
　　等価式
　　*And*式 and 等価式

And 演算子は左から順に結合する。メッセージ ID'and/+-'を持つメッセージが送信される。宛先は左辺が表すストリームで、第一引数は右辺の表すストリームである。第二引数は新しいストリームの入力端で、このストリームのアウトレットが式の値となる。

9.1.11 xor 演算子

*Xor*式
　　*And*式
　　*Xor*式 xor *And*式

Xor 演算子は左から順に結合する。メッセージ ID'xor/+-'を持つメッセージが送信される。宛先は左辺が表すストリームで、第一引数は右辺の表すストリームである。第二引数は新しいストリームの入力端で、このストリームのアウトレットが式の値となる。

9.1.12 or 演算子

*Or*式
　　*Xor*式
　　*Or*式 or *Xor*式

Or 演算子は左から順に結合する。メッセージ ID'or/+-'を持つメッセージが送信される。宛先は左辺が表すストリームで、第一引数は右辺の表すストリームである。第二引数は新しいストリームの入力端で、このストリームのアウトレットが式の値となる。

9.2 入力端式

入力端式
　入力端スロット参照
　入力端変数
　後置式 '?'

入力端式には、入力端スロット参照、入力端変数、および Who 式がある。

入力端スロット参照については see Section 8.1 [入力端スロットの参照], page 18

入力端変数については see Chapter 6 [変数], page 12

9.2.1 Who 式

後置式の後ろに疑問符 ('?') が続く表記は入力端式である。メッセージ ID':who/+'を持つメッセージが送信される。宛先は後置式が表す出力端である。第一引数は、新しいストリームの出力端で、このストリームの入力端が式の値となる。

メッセージ ID':who/+'に対応するメソッドは、原則としてオブジェクトを識別するために使われることが想定されている。単純データには':who/+'に対応する組み込みメソッドが用意されている。このメソッドは、これを実行するオブジェクトを表すメッセージを第一引数の出力端に送信する。ここで送られるメッセージのメッセージ ID は、引数無しで、表すべきオブジェクトの表記をメッセージ名として持つ。以下に、単純データに対応するメッセージ ID の例を示す。

例:

'foo'	':foo'
'5'	':5'
''true'	':'true'

10 アクション

アクション
　　式
　　入力端スロット更新
　　対スロット更新
　　{ メッセージ }
　　ボラタイルクラス定義

アクションは、式、入力端スロットの更新、対スロット更新、自己送信、またはボラタイルクラス定義である。

式については see Chapter 9 [式], page 20

入力端スロット更新については see Section 8.2 [入力端スロットの更新], page 18

対スロット更新については see Section 8.5 [対スロット更新], page 19

またボラタイルクラスについては see Section 12.2 [ボラタイルクラス定義], page 32

10.1 自動閉鎖と自動排水

式が単独でアクションとして用いられる場合、その式の値であるストリームへの参照は自動的に処分される。すなわち、出力端式からなるアクションの場合、式の値である出力端は自動的に閉鎖される。また、入力端式の場合は、自動的に閉鎖される。

次の例では、'X'はメッセージ':bar'が送られた後、閉鎖される。'Y'は即座に閉鎖される。入力端スロット'@nue'の値は即座に排水される。

例：

```
:foo(^X, ^Y) ->
        X :bar, Y, @nue.
```

10.2 自己送信

任意の個数のメッセージの並びはアクションであり、そのアクションを実行するオブジェクト自身にメッセージが送信される。新しいストリームが生成され、このストリームが新しいインターフェースストリームになる。このストリームに指定されたメッセージが送信される。メッセージは、それらがプログラム中に出現する順序に従って送信される。すべてのメッセージが送信されてから、このストリームの出力端に従来のインターフェースストリームの入力端が接続される。

例：

次のメソッドでは、このメソッドを実行しているオブジェクト自身にメッセージ':foo1'、':foo2', ':foo3' がこの順番で送られる。

```
:begin(^X) ->
        :foo1 :foo2,
        :foo3.
```

11 メソッド定義

メソッド定義
　　セレクタ 後継指示子 アクション並び '.'
後継指示子
　　'->'
　　'-|'
アクション並び
　　アクション [{ ',' アクション }]

メソッド定義は、セレクタ、後継処理の有無、そしてコンマ (',') で区切られた任意の個数のアクションからなる。メソッド定義は、メソッドの振舞いを定義する。

メソッドは、セレクタで指定されたメッセージ ID を持つメッセージが受信されるか、インターフェースストリームが閉鎖されることによって起動される。メソッドが起動されると、セレクタのメッセージの引数になっている式およびアクション並び中のアクションがすべて並行に実行される。また、後継指定子の指示にしたがってインターフェースストリームが操作される。

11.1 セレクタ

セレクタ
　　メッセージ
　　'::'
　　入力端式

セレクタは、メッセージ、閉鎖セレクタ、または入力端式である。

セレクタがメッセージである場合、そのセレクタによって定義されるメソッドは、そのメッセージと同じメッセージ ID を持つメッセージによって起動される。セレクタの引数である式は、アクションと並行して実行され、式の値は実際に受信されるメッセージの対応する位置の引数と接続される。

メソッド定義のセレクタの引数の方向は反転している。セレクタの引数は受信したメッセージの引数と接続されるストリームを表しているからである。

二個並んだコロン ('::') は閉鎖セレクタである。閉鎖セレクタによって定義されるメソッドは、インターフェースストリームが閉鎖されたときに起動される。

セレクタとして入力端式を持つメソッドは、デフォルトメソッドである。デフォルトメソッドは、受けとった

メッセージに対応するメソッドが他にない場合に起動される。受けとったメッセージに対応するオブジェクトが生成され、これにセレクタの入力端式の値が接続される。

11.2 後継指示子

後継指示子は矢印('->') または行き止まり('-|') のどちらかであり、これによって宣言されるメソッドが実行された後のインターフェースストリームの扱いを指定する。通常、'->'が指定され、インターフェースストリームから次のメッセージが読み出され、それに対応する次のメソッドが実行される。'-|'の場合、インターフェースストリームは排水される。メソッドの中でインターフェースストリームが更新される場合、後継指示子はメソッド起動時のインターフェースストリームにのみ影響し、更新後のインターフェースストリームには影響しない。

注：現在のシステムでは、'-|'によって定義される振舞いは上記の記述とは異なる恐れがある。

例：

以下の例では後継指示子によって従来のインターフェースストリームは排水される。一方自己送信される':loop/+'は、新しく生成されるストリームに送られ、このストリームが新しいインターフェースストリームになる。その結果、他のオブジェクトからのメッセージはこのオブジェクトに届かなくなるが、':loop/+'が起動するメソッドは実行される。

```
:foo(^X) -|
        :loop(10).
```

12 クラス定義

```
クラス定義
      クラスヘッダ [ 属性宣言並び ] [ メソッド定義並び ] ‘クラスフッタ’
クラスヘッダ
      class 識別子 ‘.’
属性宣言並び
      { 属性宣言 }
属性宣言
      super 識別子並び ‘.’
      in 識別子並び ‘.’
      out 識別子並び ‘.’
識別子並び
      識別子 [ { ‘,’ 識別子 } ]
メソッド定義並び
      { メソッド定義 }
クラスフッタ
      end ‘.’
```

クラス定義は、クラスヘッダで始まり、その後ろに属性定義並び、メソッド定義並びが続き、クラスフッタで終る。属性定義並び、メソッド定義並びは無くてもよい。クラス定義はそのクラスに所属するオブジェクトの属性と振舞いを定義する。

クラスヘッダは予約語 classの後ろに識別子が続き、ピリオド(‘.’)で終る。識別子がここで定義されるクラスのクラス名となる。

属性宣言は、スーパークラス宣言、入力端スロット宣言、またはアウトレットスロット宣言である。これらはすべて予約語で始まり、これにコンマ(‘,’)で区切られた任意個の識別子が続き、‘.’で終る。

スーパークラス宣言は予約語 superで始まる。これに続く識別子はすべてクラス名でなければならない。ここで宣言されたクラス名はこの宣言で定義されるクラスのスーパークラスとなる。

入力端スロット宣言は予約語 inで始まる。この宣言で定義されるクラスは、この宣言中の識別子を名前とする入力端スロットを持つ。

出力端スロット宣言は予約語 outで始まる。この宣言で定義されるクラスは、この宣言中の識別子を名前とする出力端スロットを持つ。

メソッド定義については see Chapter 11 [メソッド定義], page 29

クラスフッタは予約語 endの後ろに‘.’が続いた表記であり、クラス定義の終りを示す。

12.1 継承

クラスは任意の個数の他のクラスの定義を継承することができる。あるクラスを継承したクラスは、継承されるクラスのすべてのスロットとメソッドを継承する。

クラス間の継承関係は、継承されるクラスを子ノードとし、継承するクラスを親ノードとする木構造を形成する。これを継承木と呼ぶ。便宜上、スーパークラス宣言の中に現れるクラス名の順序に従って木の左右を定義する。

例：

次のプログラムは、クラスaを根とし、クラスb, c, dを葉とする継承木を定義する。'b'がもっとも左側の葉であり、'd'がもっとも右側の葉である。

```
class a.
    super b, c, d.
end.
```

継承木の中の複数のクラスで同じメッセージIDを持つメソッドが定義されている場合、継承木の左側深さ優先探索によって最初に見つかるメソッドが継承される。また、スロットアクセスはメソッドによって行われるので、複数のクラスで同じスロットが定義されている場合も、同じ探索によって最初に見つかるスロットだけが継承される。

12.2 ボラタイルクラス定義

```
ボラタイルクラス定義
        インターフェース式 '(' メソッド定義並び ')'
インターフェース式
        入力端式 '?'
        Who式
```

ボラタイルクラス定義は、インターフェース式の後ろにカッコ('( )')で囲まれたメソッド定義並びが続いたものである。この式によって、ボラタイルクラスが定義される。さらに実行時には、このボラタイルクラスのインスタンスが生成され、これに、インターフェース式の値である入力端が接続される。

インターフェース式は入力端式に疑問符('?')が続いた表記、または、Who式である。インターフェース式の値は、前者の場合は入力端式の値であり、後者ではWho式の値である。

ボラタイルクラス定義は、それを含むメソッドが定義するスコープに含まれる。
(see Chapter 6 [変数], page 12)。また、ボラタイルクラス定義の中では、それを含む通常クラスのスロットをアクセスできる。スロットへのアクセス順序はボラタイルクラス定義の内外には関係なく、構文上の出現順序による。ボラタイルクラス定義中の自己送信が生成する新たなストリームは、それを含む通常クラスのオブジェクトの新たなインターフェースストリームになり、従来のインターフェースストリームは新たなストリームの出力端に接続される。

例:

```
class bar.
    out noo.
:foo(X) ->
    (X > 0) ? (
    :'true ->
        !noo + 1 = !noo,      % クラス bar のスロット noo をアクセスできる。
        :foo(X - 1).          % :foo/+ はクラス bar のオブジェクトに送られる
    :'false -> .
    ),
    :zoo.                     % :zoo は :foo/+ の後から届く。
:zoo -> .
end.
```

13 プログラム

プログラム
　　{ クラス定義 }

プログラムは、一つ以上のクラス定義の並びである。

プログラムは、必ずクラス aumの定義を一つだけ含まなければならない。また、クラス aumの定義はメソッド ':go/+'の定義を含まなければならない。プログラムが起動されると、システムによってクラス aumのインスタンスが生成され、これに、メッセージ ':go/+'が送られる。

プログラムの実行が終了するのは、すべてのオブジェクトがメソッドを実行していない時点で、どのオブジェクトのインターフェースストリームにも受信されていないメッセージが残っていない状態になったときである。プログラムが終了したとき、すべてのインターフェースストリームが閉鎖されているとは限らない。

PAS (Parallel A'Um System)

参照マニュアル

第 1.14 版 (1991 年 9 月 18 日)

Table of Contents

1 概要 1

1.1 本マニュアルの使い方 1

1.1.1 説明の形式 1

1.1.2 引数の取り扱いについて 2

1.2 用語の説明 2

1.2.1 アトミックオブジェクト 2

1.2.2 ストリームの方向 2

2 組込オブジェクト 3

2.1 比較 3

2.2 その他 4

3 数 5

3.1 型変換のルール 5

3.2 数 5

3.3 整数 7

3.4 数学関係 9

4 シンボル 15

5 論理値 16

6 列 18

6.1 列 18

6.2 文字列 20

6.3 リスト 21

6.4 メッセージ 22

7 ライブラリ 23

7.1 ファイル 23

7.1.1 ファイルストリームの生成 23

7.1.2 ファイル操作メソッド 23

Method Index 27

1 概要

PAS (Parallel A'Um System) は並列オブジェクト指向言語 A'Um の処理系である. PAS はその名の示すように, 並列に A'Um 言語を実行する処理系で, その処理単位はクラスのインスタンスオブジェクトである. A'Um のクラスはメソッドと呼ぶ処理手順からなる. インスタンスオブジェクトはメッセージを受け取ることにより, クラスで定義されているメソッドを起動する. メソッドの中ではインスタンスオブジェクトのスロット値の参照/設定, 別なインスタンスオブジェクトの生成, そしてオブジェクトへのメッセージの送信を行なう.

PAS は非共有メモリシステム上で動作することを想定している. このため, オブジェクト間のメッセージ通信は共有メモリを必要としない方式をとっている. 同じ PE (Processing Element) 上にあるオブジェクト同士の通信では, A'Um のメッセージ通信は直接宛先のオブジェクトに送ることができる. これに対して, 送信するオブジェクトが異なる PE 上にあると, A'Um のメッセージを外部メッセージ形式に変換して送る. この共有メモリを想定しない通信は, 少い PE 数のシステムでは不利だと考えられるが, 大規模なシステムでは有効な方式である. また, この通信方式はネットワーク上に存在するワークステーションを一つの PE と見なすこともできるので, ネットワーク上で分散システムを構築するのにも利用できる.

1.1 本マニュアルの使い方

本マニュアルは, PAS が組み込んでいるメソッドの説明を行なっている. 各メソッドの説明はクラス毎に章単位にまとめており, 各章でメソッドはアルファベット順に並べてある.

メソッドはオブジェクトにメッセージが届くことにより起動されるので, メソッドの宛先に関するエラーの記述は行なっていない. しかし, メソッドの引数については, 各メソッド毎に記述してある.

1.1.1 説明の形式

メソッドは以下の項目ごとに説明してある.

'形式' メソッドの引数定義ならびにメッセージの形式を表している. オブジェクト X にメッセージ msg が送られた時に起動されるメソッドは X :msg の形式で記述しており, 引数が入力方向ならば仮引数を表す変数の前に '^' を付けている.

'マクロ表記' 慣例的に利用される形式を述べている. 例えば, 数の足し算を A'Um では X :add (Y,^Z) と表現するが, 通常は X + Y = ^Z と記述するのが普通である.

'機能説明' メソッドの処理内容を記述している.

1.1.2 引数の取り扱いについて

引数を持つ組込メソッドは引数に対する処理に次の二通りがある．ただしこれは出力方向の引数に対してのみ有効である．

1. 引数が何者にも接続されていない時，接続されるまでメソッドの実行を止めて待つ．メソッドの実行が副作用を残す場合に適用される．
2. 引数が何者にも接続されていない時，そのストリームにメッセージを流して次の処理に移る．例えば，次の整数の足し算の場合，

   ```
   X + Y = ^Z
   ```

   Y が未接続な場合，Y に :add(X, ^Z) のメッセージを送ることにより Y の値が確定するのを待たずに，Z の値を利用することができる．

1.2 用語の説明

本マニュアルで使用している用語について述べる．

1.2.1 アトミックオブジェクト

PAS では以下のデータをアトミックオブジェクトと呼ぶ．

- 整数
- 単精度浮動小数点数
- シンボル
- 論理値

1.2.2 ストリームの方向

オブジェクトからオブジェクトにメッセージを送信する状況において，メッセージの通り道をストリームと呼ぶ．そして，ストリーム内を流れるメッセージの方向をストリームの方向と呼ぶ．

通信路であるストリームには二つの端があり，メッセージの送る側を出力端と呼びメッセージを受け取る側を入力端と呼ぶ．出力端と入力端を繋げて２本のストリームを１本のストリームにすることができる．また入力端をオブジェクトに繋げて，ストリーム内のメッセージをオブジェクトに転送することができる．

2 組込オブジェクト

本章では PAS が提供する組込クラスにおいて定義されるメソッドについて述べる. この章に挙げた全てのメソッドは, PAS の全ての組込クラスのインスタンスオブジェクトで実行可能である.

2.1 比較

本節では二つのオブジェクトの比較を行なうメソッドについて述べる. 比較結果は論理値 'true, 'false によって表現される.

eq Method

形式 X :eq (Y, ^Z)

マクロ表記
```
X == Y = ^Z
```

機能説明 X と Y を比較し, 等しい時は 'true を, そうでない時は 'false をストリーム Z と接続する. Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合, Y に:eq (X, ^Z,) メッセージを送る.

ge Method

形式 X :ge (Y, ^Z)

マクロ表記
```
X >= Y = ^Z
Y < X = ^Z
```

機能説明 X と Y を比較し, X が Y より大きいかまたは等しいならば 'true を, そうでない時には 'false をストリーム Z と接続する. Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合, Y に:ge (X, ^Z,) メッセージを送る.

gt Method

形式 X :gt (Y, ^Z)

マクロ表記
```
X > Y = ^Z
Y <= X = ^Z
```

機能説明 X と Y を比較し, X が Y より大きいならば 'true を, そうでない時には 'false をストリーム Z に接続する. Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合, Y に:gt (X, ^Z,) メッセージを送る.

neq

Method

形式 X :neq (Y, ^Z)

マクロ表記 X != Y = ^Z

機能説明 X と Y を比較し，等しくない時は 'true を，そうでない時は 'false をストリーム Z に接続する．Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:neq (X, ^Z,) メッセージを送る．

2.2 その他

class_name

Method

形式 X :class_name (^Y)

機能説明 X のクラス名をシンボルとして，ストリーム Y と接続する．

class_of

Method

形式 X :class_of (^Y)

機能説明 X のクラス名をシンボルとして，ストリーム Y と接続する．X がユーザ定義クラスのオブジェクトの場合，最初そのクラスで参照できる class_of/- を探す．見つからなかった場合には，ユーザ定義クラスの名前をシンボルとしてストリーム Y と接続する．

copy

Method

形式 X :copy (^Y)

機能説明 X を複写して新しいオブジェクトを生成し，ストリーム Y と接続する．X と Y は同じ型となる．

who

Method

形式 X :who (Y)

マクロ表記 Y = X ?

機能説明 X を表すメッセージを作成し，それを Y に送る．これはオブジェクトをメッセージに変換する唯一の方法である．X がアトミックオブジェクトの場合，アトミックメッセージを Y に送る．

3 数

PAS では数として整数，単精度浮動小数点数そして倍精度浮動小数点数を扱うことができる．

3.1 型変換のルール

同じ型同士の数の演算結果は，元の数の型と同じである．しかし異なる型の数を演算した場合，自動的に型変換を行なう．型変換の規則を以下に述べる．

1. 整数と単精度浮動小数点数の演算結果は，単精度浮動小数点数となる．
2. 整数と倍精度浮動小数点数の演算結果は，倍精度浮動小数点数となる．
3. 単精度浮動小数点数と倍精度浮動小数点数の演算結果は，倍精度浮動小数点数となる．

3.2 数

本節では整数，単精度浮動小数点数および倍精度浮動小数点数に共通するメソッドについて述べる．

add Method

形式 X :add (Y, ^Z)

マクロ表記 X + Y = ^Z

機能説明 X と Y を加算し，その結果をストリーム Z と接続する．X と Y の『数』の型が異なる場合，型変換を行なう．Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:add (X, ^Z,) メッセージを送る．

div Method

形式 X :div (Y, ^Z)

マクロ表記 X / Y = ^Z

機能説明 X を Y で除算し，その結果をストリーム Z と接続する．Y は数でなくてはならない．X と Y の『数』の型が異なる場合，型変換を行なう．Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:rev_div (X, ^Z,) メッセージを送る．

double

Method

形式 X :double (^Y)

機能説明 X が数値を表すオブジェクトあるいは文字列のとき，X が表す数値の倍精度浮動小数点数を生成し，ストリーム Y と接続する．

float

Method

形式 X :float (^Y)

機能説明 X が数値を表すオブジェクトあるいは文字列のとき，X が表す数値の単精度浮動小数点数を生成し，それをストリーム Y と接続する．

int

Method

形式 X :int (^Y)

機能説明 X が数を表すオブジェクトあるいは文字列のとき，X が表す数値の整数を生成し，それをストリームに Y と接続する．

minus

Method

形式 X :minus (^Y)

機能説明 X の符号を反転したオブジェクトを生成し，それをストリーム Y と接続する．オブジェクト X とオブジェクト Y の型は同じとなる．

mul

Method

形式 X :mul (Y, ^Z)

マクロ表記 X * Y = ^Z

機能説明 X と Y を乗算し，その結果をストリーム Z と接続する．X と Y の『数』の型が異なる場合，型変換を行なう．Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:mul (X, ^Z,) メッセージを送る．

rev_div

Method

形式 Y :rev_div (X, ^Z)

機能説明 X を Y で除算し，その結果をストリーム Z と接続する．Y は数でなくてはならない．X と Y の『数』の型が異なる場合，型変換を行なう．X が未接続なジョイントの場合はエラーである．

rev_sub

Method

形式 *Y* :rev_sub (*X*, ^Z)

機能説明 X から Y を引算し，その結果をストリーム Z と接続する．X と Y の『数』の型が異なる場合，型変換を行なう．X が未接続なジョイントの場合はエラーである．

string Method

形式 *X* :string (^*Y*)

機能説明 X がシンボル，論理値または数ならば，その印字イメージを持つ文字列を生成し，ストリーム Y と接続する．

sub Method

形式 *X* :sub (*Y*, ^Z)

マクロ表記 X - Y = ^Z

機能説明 X から Y を引算し，その結果をストリーム Z と接続する．X と Y の『数』の型が異なる場合，型変換を行なう．Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:rev_sub (*X*, ^Z,) メッセージを送る．

3.3 整数

本節では整数オブジェクトのメソッドについて述べる．

abs Method

形式 *X* :abs (^*Y*)

機能説明 X の絶対値を表す整数を生成し，ストリーム Y と接続する．

and Method

形式 *X* :and (*Y*, ^Z)

マクロ表記 X and Y = ^Z

機能説明 X と Y がともに論理値ならば両者のブール代数の論理積をとり，結果をストリーム Z と接続する．X と Y がともに整数ならばビットの論理積をとり，結果をストリーム Z と接続する．Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:and (*X*, ^Z,) メッセージを送る．

mod Method

形式 *X* :mod (*Y*, ^Z)

機能説明 X と Y が整数ならば X を Y で除算し，その余りをストリーム Z と接続する．Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:rev_mod (X, ^Z,) メッセージを送る．

not Method

形式 *X* :not (^Y)

マクロ表記 not X = ^Y

機能説明 X が論理値なら論理否定を行い，結果をストリーム Y と接続する．X が整数ならビット反転を行い，結果をストリーム Y と接続する．

or Method

形式 *X* :or (*Y*, ^Z)

マクロ表記 X or Y = ^Z

機能説明 X と Y がともに論理値ならば両者のブール代数の論理和をとり，結果をストリーム Z と接続する．X と Y がともに整数ならばビットの論理和をとり，結果をストリーム Z と接続する．Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:or (*X*, ^Z,) メッセージを送る．

rev_mod Method

形式 *Y* :rev_mod (*X*, ^Z)

機能説明 X と Y が整数ならば X を Y で除算し，その余りをストリーム Z と接続する．X が未接続なジョイントの場合はエラーである．

rev_shtl Method

形式 *Y* :rev_shtl (*X*, ^Z)

機能説明 X と Y がともに整数ならば，X を Y ビット左シフトし，結果をストリーム Z と接続する．X が未接続なジョイントの場合はエラーである．

rev_shtr Method

形式 *Y* :rev_shtr (*X*, ^Z)

機能説明 X と Y がともに整数ならば，X を Y ビット右シフトし，結果をストリーム Z と接続する．X が未接続なジョイントの場合はエラーである．

shtl Method

形式 X :shtl (Y, ^Z)

マクロ表記 X << Y = ^Z

機能説明 X と Y がともに整数ならば，X を Y ビット左シフトし，結果をストリーム Z と接続する．Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:rev_shtl (X, ^Z,) メッセージを送る．

shtr Method

形式 X :shtr (Y, ^Z)

マクロ表記 X >> Y = ^Z

機能説明 X と Y がともに整数ならば，X を Y ビット右シフトし，結果をストリーム Z と接続する．Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:rev_shtr (X, ^Z,) メッセージを送る．

xor Method

形式 X :xor (Y, ^Z)

マクロ表記 X xor Y = ^Z

機能説明 X と Y がともに論理値ならば両者のブール代数の排他的論理和をとり，結果をストリーム Z に接続する．X と Y がともに整数ならばビットの排他的論理和をとり，結果をストリーム Z に接続する．Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:xor (X, ^Z,) メッセージを送る．

3.4 数学関係

本節では数学ライブラリのメソッドについて述べる．ここで挙げているメソッドは全ての数オブジェクトで有効だが，演算は倍精度浮動小数点数に変換して行なわれる．

acos Method

形式 X :acos (^Y)

機能説明 X の逆余弦値を倍精度浮動小数点数で生成し，ストリーム Y と接続する．

asin Method

形式 X :asin (^Y)

機能説明 X の逆正弦値を倍精度浮動小数点数で生成し，ストリーム Y と接続する．

atan Method

形式 *X* :atan (^*Y*)

機能説明 X の逆正接値を倍精度浮動小数点数で生成し，ストリーム Y と接続する．

atan2 Method

形式 *X* :atan2 (*Y*, ^*Z*)

機能説明 Y が倍精度浮動小数点数ならば X を Y で除算し，その逆正接をとった値を倍精度浮動小数点数で生成し，ストリーム Z と接続する． *Y* が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，*Y* に:rev_atan2 (*X*, ^*Z*,) メッセージを送る．

cabs Method

形式 *X* :cabs (*Y*, ^*Z*)

機能説明 X を実部 Y を虚部とした複素数の絶対値を倍精度浮動小数点数として生成し，ストリーム Z と接続する． Y は数でなくてはならない． *Y* が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，*Y* に:rev_cabs (*X*, ^*Z*,) メッセージを送る．

ceil Method

形式 *X* :ceil (^*Y*)

機能説明 X を越える最小の整数値を倍精度浮動小数点数として生成し，ストリーム Y と接続する．これは小数点以下の切り上げをするのと同じである．

cos Method

形式 *X* :cos (^*Y*)

機能説明 X の余弦を倍精度浮動小数点数として生成し，ストリーム Y と接続する．

cosh Method

形式 *X* :cosh (^*Y*)

機能説明 X の双曲線余弦を倍精度浮動小数点数として生成し，ストリーム Y と接続する．

exp Method

形式 *X* :exp (^*Y*)

機能説明 X の指数関数 (e^x) を倍精度浮動小数点数として生成し，ストリーム Y と接続する．

fabs

Method

形式 X :fabs (^Y)

機能説明 X の絶対値を倍精度浮動小数点数として生成し，ストリーム Y と接続する．

floor

Method

形式 X :floor (^Y)

機能説明 X を越えない最大の整数値を倍精度浮動小数点数として生成し，ストリーム Y と接続する．これは小数点以下の切り捨てをするのと同じである．

gamma

Method

形式 X :gamma (^Y)

機能説明 X の Log Gamma 関数の値を倍精度浮動小数点数として生成し，ストリーム Y と接続する．

hypot

Method

形式 X :hypot (Y, ^Z)

機能説明 式 $X^2 + Y^2 = Z^2 (Z \geq 0)$ を満たす Z を倍精度浮動小数点数として生成し，ストリーム Z と接続する．これは直角三角形の底辺と高さから斜辺を求める方法である．
Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:rev_hypot (X, ^Z,) メッセージを送る．

j0

Method

形式 X :j0 (^Y)

機能説明 X に対する第 1 種ベッセル関数の 0 項の値を倍精度浮動小数点数として生成し，ストリーム Y と接続する．

j1

Method

形式 X :j1 (^Y)

機能説明 X に対する第 1 種ベッセル関数の 1 項の値を倍精度浮動小数点数として生成し，ストリーム Y と接続する．

jn

Method

形式 X :jn (Y, ^Z)

機能説明 X の Y 次第 1 種ベッセル関数の値を倍精度浮動小数点数として生成し，ストリーム Z と接続する．Y は数でなくてはならない．Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:rev_jn (X, ^Z,) メッセージを送る．

log Method

形式 X :log (^Y)

機能説明 X の自然対数の値を倍精度浮動小数点数として生成し，ストリーム Y と接続する．X の値は正でなければならない．

log10 Method

形式 X :log10 (^Y)

機能説明 X の 底を 10 とする対数の値を倍精度浮動小数点数として生成し，ストリーム Y と接続する．X の値は正でなければならない．

pow Method

形式 X :pow (Y, ^Z)

機能説明 X の Y 乗 (X^Y) を倍精度浮動小数点数として生成し，それをストリーム Z と接続する．Y は数でなくてはならない．Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:rev_pow (X, ^Z,) メッセージを送る．

rev_atan2 Method

形式 Y :rev_atan2 (X, ^Z)

機能説明 X と Y が倍精度浮動小数点数ならば X を Y で除算し，その逆正接をとった値を倍精度浮動小数点数で生成し，ストリーム Z と接続する．X が未接続なジョイントの場合はエラーである．

rev_cabs Method

形式 Y :rev_cabs (X, ^Z)

機能説明 X を実部 Y を虚部とした複素数の絶対値を倍精度浮動小数点数として生成し，ストリーム Z と接続する．X が未接続なジョイントの場合はエラーである．

rev_hypot Method

形式 Y :rev_hypot (X, ^Z)

機能説明 式 $X^2 + Y^2 = Z^2 (Z \geq 0)$ を満たすZを倍精度浮動小数点数として生成し，ストリームZと接続する．これは直角三角形の底辺と高さから斜辺を求める方法である．Xが未接続なジョイントの場合はエラーである．

rev_jn Method

形式 Y :rev_jn (X, ~Z)

機能説明 Xの Y 次第1種ベッセル関数の値を倍精度浮動小数点数として生成し，ストリームZと接続する．Xが未接続なジョイントの場合はエラーである．

rev_pow Method

形式 Y :rev_pow (X, ~Z)

機能説明 Xの Y 乗 (X^Y) を倍精度浮動小数点数として生成し，それをストリームZと接続する．Xが未接続なジョイントの場合はエラーである．

rev_yn Method

形式 Y :rev_yn (X, ~Z)

機能説明 Xの Y 次第2種ベッセル関数の値を倍精度浮動小数点数として生成し，ストリームZと接続する．Xが未接続なジョイントの場合はエラーである．

sin Method

形式 X :sin (~Y)

機能説明 Xの正弦の値を倍精度浮動小数点数として生成し，ストリームYと接続する．

sinh Method

形式 X :sinh (~Y)

機能説明 Xの双曲線正弦の値を倍精度浮動小数点数として生成し，ストリームYと接続する．

sqrt Method

形式 X :sqrt (~Y)

機能説明 Xの平方根の値を倍精度浮動小数点数として生成し，ストリームYと接続する．Xの値は正でなければならない．

tan Method

形式 X :tan (^Y)

機能説明 X の正接の値を倍精度浮動小数点数として生成し，ストリーム Y と接続する.

tanh Method

形式 X :tanh (^Y)

機能説明 X の双曲線正接の値を倍精度浮動小数点数として生成し，ストリーム Y と接続する.

y0 Method

形式 X :y0 (^Y)

機能説明 X に対する第 2 種ベッセル関数の 0 項の値を倍精度浮動小数点数として生成し，ストリーム Y と接続する.

y1 Method

形式 X :y1 (^Y)

機能説明 X に対する第 2 種ベッセル関数の 1 項の値を倍精度浮動小数点数として生成し，ストリーム Y と接続する.

yn Method

形式 X :yn (Y, ^Z)

機能説明 X の Y 次第 2 種ベッセル関数の値を倍精度浮動小数点数として生成し，ストリーム Z と接続する. Y は数でなくてはならない. Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に :rev_yn (X, ^Z,) メッセージを送る.

4 シンボル

本章ではシンボルオブジェクトが認識できるメソッドについて述べる.

string Method

形式 X :string (^Y)

機能説明 X がシンボル, 論理値または数ならば, その印字イメージを持つ文字列を生成し, ストリーム Y と接続する.

5 論理値

本章では二つの論理オブジェクト間の論理演算を行なうメソッドについて述べる.

and Method

形式 X :and (Y, ^Z)

マクロ表記 X and Y = ^Z

機能説明 X と Y がともに論理値ならば両者のブール代数の論理積をとり，結果をストリーム Z と接続する. X と Y がともに整数ならばビットの論理積をとり，結果をストリーム Z と接続する. Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:and (X, ^Z,) メッセージを送る.

not Method

形式 X :not (^Y)

マクロ表記 not X = ^Y

機能説明 X が論理値なら論理否定を行い，結果をストリーム Y と接続する. X が整数ならビット反転を行い，結果をストリーム Y と接続する.

or Method

形式 X :or (Y, ^Z)

マクロ表記 X or Y = ^Z

機能説明 X と Y がともに論理値ならば両者のブール代数の論理和をとり，結果をストリーム Z と接続する. X と Y がともに整数ならばビットの論理和をとり，結果をストリーム Z と接続する. Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に:or (X, ^Z,) メッセージを送る.

string Method

形式 X :string (^Y)

機能説明 X がシンボル，論理値または数ならば，その印字イメージを持つ文字列を生成し，ストリーム Y と接続する.

xor Method

形式 X :xor (Y, ^Z)

マクロ表記 X xor Y = ^Z

機能説明　X と Y がともに論理値ならば両者のブール代数の排他的論理和をとり，結果をストリーム Z に接続する．X と Y がともに整数ならばビットの排他的論理和をとり，結果をストリーム Z に接続する．Y が宛先の決まっていないジョイントあるいはユーザオブジェクトの場合，Y に :xor (X, ~Z,) メッセージを送る．

6 列

本章では列の操作メソッドについて述べる. PASでは以下のデータ型を列と呼ぶ.

- 文字列オブジェクト
- リストオブジェクト
- 配列オブジェクト
- メッセージオブジェクト

6.1 列

本節では列オブジェクトに対して共通な操作について述べる.

dir Method

形式 `X :dir (Y, ^Z)`

機能説明 Xの Y 番目の要素の方向を求めストリーム Z と接続する. ストリーム Z には シンボル 'in' または 'out' が接続され, それぞれストリームの入力方向(入力端), 出力方向(出力端)を表す. Y は整数でなくてはならない. Y がジョイントの場合が何かに接続されるまで待つ. X が文字列の場合は常に 'out' となる.

elt Method

形式 `X :elt (Y, Z)`

機能説明 Xの Y 番目の入力端ストリームを取り出し Z と接続する. 取り出した後の Y 番目の要素は初期化されていない入力端となる. Y は正の整数でなくてはならない. Y が未接続のジョイントの場合 Y 何かに接続されるまで待つ. それ以外のデータ型の場合はエラーである. Y 番目の要素が出力端ならばエラーであり, X はリスト, 配列あるはメッセージでなくてはならない.

elt Method

形式 `X :elt (Y, ^Z)`

機能説明 Xの Y 番目の要素をストリーム Z と接続する. Y は正の整数でなくてはならない. Y が未接続のジョイントの場合 Y が何かに接続されるまで待つ.

fix

Method

形式 *X* :fix (*Y*, *Z*)

機能説明 未定義のジョイントが確定されるまで待った後の後処理を行なう．Xがベクタの場合，待ちの発生したベクタに送られたメッセージはベクタの第0要素にとして保持される．Zはもともとのベクタ第0要素である．Yに保持されていたメッセージ列を取り出す時に第0要素をZに置き換える．

fix

Method

形式 *X* :fix (*Y*, ^*Z*)

機能説明 未定義のジョイントが確定されるまで待った後の後処理を行なう．Xがベクタの場合，待ちの発生したベクタに送られたメッセージはベクタの第0要素にとして保持される．Zはもともとのベクタ第0要素である．Yに保持されていたメッセージ列を取り出す時に第0要素をZに置き換える．

length

Method

形式 *X* :length (^*Y*)

機能説明 Xが文字列であれば文字列の長さを，配列であれば配列の要素数を，リストであばリストの長さを，メッセージであれば引数の数をストリームYと接続する．Xがリストでかつ cdr 部が接続されていないストリームの場合には，ストリームが何かに接続されるのを待つ．

set

Method

形式 *X* :set (*Y*, *Z*)

機能説明 XのY番目の要素を出力端Zとする．Yは正の整数でなくてはならない．Yが未接続のジョイントの場合Yが何かに接続されるまで待つ．

set

Method

形式 *X* :set (*Y*, ^*Z*)

機能説明 XのY番目の要素に入力端ストリームZをセットする．Yは正の整数でなくてはならない．Yが未接続のジョイントの場合Yが何かに接続されるまで待つ．オブジェクトXはリスト，配列あるいはメッセージでなくてはならない．

subseq

Method

形式 *X* :subseq (*Y*, *Z*, ^*S*)

機能説明 X が文字列，リストまたは配列ならば Y 番目から Z 番目までの部分列を新たなる同じ型のオブジェクトとして生成し，ストリーム S と接続する．Y，Z は正の整数でなくてはならない．それらが未接続のストリームの場合何かに接続されるまで待つ．Y と Z が等しいとき，X が文字列ならば ""，リストならば '[]'，ベクタならば '{}' がストリーム S と接続される．【注意】本来ならばメッセージオブジェクトも扱えるはずであるが，現処理系では新しいメッセージを作成する際に制限があるため省いている．将来的には subseq でメッセージも扱えるようになる．

6.2 文字列

本節では文字列操作のメソッドについて述べる

append Method

形式 *X* :append (*Y*, ^Z)

機能説明 X と Y の型が等しく，型が文字列かリストならば，二つのオブジェクトを連結したオブジェクトを生成し，ストリーム Z と接続する．連結を行なう時，オブジェクトの型によって要素のコピー方法が異なる．型が文字列の時には，二つのオブジェクトの長さの和を持つ文字列を生成し，X の内容そして Y の内容を設定する．型がリストの時には，X の内容を持ちかつ最後の要素の cdr 部が Y であるリストを生成する．

double Method

形式 *X* :double (^Y)

機能説明 X が数値を表すオブジェクトあるいは文字列のとき，X が表す数値の倍精度浮動小数点数を生成し，ストリーム Y と接続する．

float Method

形式 *X* :float (^Y)

機能説明 X が数値を表すオブジェクトあるいは文字列のとき，X が表す数値の単精度浮動小数点数を生成し，それをストリーム Y と接続する．

int Method

形式 *X* :int (^Y)

機能説明 X が数を表すオブジェクトあるいは文字列のとき，X が表す数値の整数を生成し，それをストリームに Y と接続する．

6.3 リスト

本節ではリストオブジェクトの操作メソッドについて述べる

append Method

形式 *X* :append (Y, ~Z)

機能説明 X と Y の型が等しく，型が文字列かリストならば，二つのオブジェクトを連結したオブジェクトを生成し，ストリーム Z と接続する．連結を行なう時，オブジェクトの型によって要素のコピー方法が異なる．型が文字列の時には，二つのオブジェクトの長さの和を持つ文字列を生成し，X の内容そして Y の内容を設定する．型がリストの時には，X の内容を持ちかつ最後の要素の cdr部が Y であるリストを生成する．

car Method

形式 *X* :car (*Y*)

機能説明 リストの car 部の入力端ストリームを Y する．X の car 部が出力方向ならばエラーである．取り出した後の頭部要素は初期化していない入力端となる．

car Method

形式 *X* :car (^Y)

機能説明 リストの car 部をストリーム Y と接続する．X の car 部が入力方向ならばエラーである．

cdr Method

形式 *X* :cdr (*Y*)

機能説明 リストの cdr 部の入力端ストリームを Y とする．X の cdr 部が出力方向ならばエラーである．取り出した後の cdr 部は初期化していない入力端となる．

cdr Method

形式 *X* :cdr (^Y)

機能説明 リストの cdr 部をストリーム Y と接続する．X の cdr 部が入力方向ならばエラーである．

setcar Method

形式 *X* :setcar (*Y*)

機能説明 リスト X の car 部に Y をセットする．

setcar Method

形式 *X* :setcar (^Y)

機能説明 リスト X の car 部に Y をセットする.

setcdr Method

形式 *X* :setcdr (Y)

機能説明 リスト X の cdr 部に Y をセットする.

setcdr Method

形式 *X* :setcdr (^Y)

機能説明 リスト X の cdr 部に Y をセットする.

6.4 メッセージ

本節ではメッセージオブジェクトの操作メソッドについて述べる.

get_mode Method

形式 *X* :get_mode (^Y)

機能説明 X の各要素の方向を表す文字列を生成しストリーム Y と接続する. 入力方向, 出力方向はそれぞれ文字'-', '+' によって表現する.

get_name Method

形式 *X* :get_name (^Y)

機能説明 X のメッセージ名を文字列として生成し, ストリーム Y と接続する.

7 ライブラリ

7.1 ファイル

7.1.1 ファイルストリームの生成

ファイルオブジェクトは以下のメソッドによって作ることができる.

open Method

形式 *#file* :open (*X*, *Y*, ^Z)

機能説明 名前 X のファイルをモード Y で開く. そして新たに生成したファルオブジェクトを Z と接続する. X と Y は文字列でなくてはならない. モード Y は以下の意味を持つ.

"r"	読み込みでファイルを開く.
"w"	書き込みでファイルを開く.
"a"	追加書き込みでファイルを開く.
"r+"	読み込み／書き込みの両方でファイルを開く. ファイルの位置は先頭となる.
"w+"	読み込み／書き込みの両方でファイルを開く. ファイルは新しく作成するか, または大きさ 0 となる.
"a+"	読み込み／書き込みの両方でファイルを開く. ファイルの位置は末尾となる.

X あるいは Y が未接続のジョイントの場合, 何かに接続されるまで待つ.

7.1.2 ファイル操作メソッド

本節ではファイルオブジェクトの操作メソッドについて述べる.

fflush Method

形式 *X* :fflush

機能説明 X のバッファに溜っているデータを吐き出す.

fprintf Method

形式 X :fprintf (Y, Z)

機能説明 Yが文字列そしてZが配列ならば，配列Zに格納されているオブジェクトをフォーマットYに従って文字列化し，ファイルXに出力する．以下にフォーマットの機能を示す．

'%o'	整数オブジェクトの8進表示
'%d'	整数オブジェクトの10進表示
'%x'	整数オブジェクトの16進表示
'%g'	数値オブジェクトの実数表示
'%f'	数値オブジェクトの実数表示
'%e'	数値オブジェクトの指数表現による表示(e)
'%E'	数値オブジェクトの指数表現による表示(E)
'%s'	文字列オブジェクトの表示
'%a'	シンボルオブジェクトの表示
'%c'	整数値で示される文字コードの文字の表示
'%l'	リストオブジェクトの表示
'%v'	配列オブジェクトの表示
'%%'	文字%の表示
'%bs'	論理値の短形式表示('true⇒t, 'false⇒f)
'%bl'	論理値の長形式表示('true⇒'true, 'false⇒'false)

YとZが未接続のジョイントの場合，何かに接続されるまで待つ．

fscanf Method

形式 X :fscanf (Y, ^Z)

機能説明 Yが文字列ならば，フォーマットYに従い入力を行なう．次に読み込んだオブジェクトを要素とする配列を生成し，入力端ストリームZと接続する．以下にフォーマットの機能を示す．

'%o'	整数オブジェクトの8進入力
'%d'	整数オブジェクトの10進入力
'%x'	整数オブジェクトの16進入力
'%g'	数値オブジェクトの小数点数記法による入力
'%f'	同上
'%e'	数値オブジェクトの指数表現による入力(e)
'%E'	数値オブジェクトの指数表現による入力(E)
'%s'	文字列オブジェクトの入力
'%c'	整数値で示される文字コードの文字の入力

'%bs' 論理値の短形式入力 ('true⇒ t, 'false⇒ f)

'%bl' 論理値の長形式入力 ('true⇒ 'true, 'false⇒ 'false)

'%%' 文字 % の入力

Y が未接続のジョイントの場合，Y が何かに接続されるまで待つ.

fseek Method

形式 *X* :fseek (Y, Z)

機能説明 Y と Z が整数ならば X のファイルポインタを指定した位置に移動させる．ファイルポインタの位置とは，次の入力あるいは出力演算の位置を表す．整数 Y の示す値は開始位置 Z からのバイト数である．開始位置は Z の値により以下の様に定義している.

'0' ファイルの先頭からのバイト数

'1' ファイルの現在位置からのバイト数

'2' ファイルの最後からのバイト数

Y と Z が未接続のジョイントの場合，何かに接続されるまで待つ.

getc Method

形式 *X* :getc (^Y)

機能説明 X から 1 文字を入力し，ストリーム Y と接続する．Y のデータ型は整数となる.

gets Method

形式 *X* :gets (^Y)

機能説明 X がファイルならば X から 1 行を入力し，ストリーム Y と接続する．Y のデータ型は文字列となる.

iseof Method

形式 *X* :iseof (^Y)

機能説明 X の状態が EOF (end of file) かどうかを検査する．もし，EOF ならば 'true をそうでなければ 'false をストリーム Y と接続する.

put Method

形式 *X* :put (Y)

機能説明 オブジェクト Y の印字イメージを X に出力する.

putc Method

形式 X :putc (Y)

機能説明 Yの整数値を文字コード(ASCIIコード)としてXに出力する.

puts Method

形式 X :puts (Y)

機能説明 Yの印字イメージをXに出力する.

ungetc Method

形式 X :ungetc (Y)

機能説明 Yの整数値を文字コード(ASCIIコード)としてファイルXに押し戻す. 結果として, 次に呼ばれる入力演算の先頭文字はYの文字表現と同じになる. 【注意】C言語の一文字押し戻し入力とは, 異なった振る舞いをする. というのはもし文字として2バイトコード(日本語)がYに与えられた場合, 押し戻し入力の効果を得たい為に文字コードのバイトサイズ分だけ, ファイル位置を移動させるからである.

Method Index

!

!=....3

*

*....6, 7

-

-....7

/

/....5, 6

=

=....4
==....3

+

+....5

>

>....3
>=....3
>>....9

<

<....3
<=....3
<<....8

A

abs....7
acos....9
add....5
and....7, 16
append....20, 21
asin....9
atan....9, 10
atan2....10

C

cabs....10
car....21
cdr....21
ceil....10
class_name....4
class_of....4
copy....4
cos....10
cosh....10

D

dir....18
div....5
double....5, 6, 20

E

elt....18
eq....3
exp....10

F

fabs....11
fflush....23
fix....18, 19
float....6, 20
floor....11
fprintf....23, 24
fscanf....24
fseek....25

G

gamma....11
ge....3
get_mode....22
get_name....22
getc....25
gets....25
gt....3

H

hypot 11

I

int 6, 20
iseof 25

J

j0 11
j1 11
jn 11

L

length 19
log 12
log10 12

M

minus 6
mod 7
mul 6

N

neq 3, 4
not 8, 16

O

open 23
or 8, 16

P

pow 12
put 25
putc 25, 26
puts 26

R

rev_atan2 12
rev_cabs 12
rev_div 6
rev_hypot 12
rev_jn 13
rev_mod 8
rev_pow 13
rev_shtl 8
rev_shtr 8
rev_sub 6
rev_yn 13

S

set 19
setcar 21, 22
setcdr 22
shtl 8
shtr 9
sin 13
sinh 13
sqrt 13
string 7, 15, 16
sub 7
subseq 19

T

tan 13
tanh 14

U

ungetc 26

W

who 4

X

X 8, 16
xor 9, 16

Y

y0 14
y1 14
yn 14