5 FORTRAN によるプログラミング

計算機を用いたプログラム開発においては、多くの場合、プログラムの記述ではなく、記述したプログラムの誤りを直す作業 (**デバッグ**[1]) と改良にほとんどの時間が費される。したがって、最初からデバッグ・改良のし易いプログラムを作ることが大切である。プログラムを書く上でもっとも大切な心構えは、

分かりやすく書くこと

である。筋道を立てて分かり易く書かれたプログラムは、デバック・拡張もまた容易である。

計算機を用いて実験データの解析や理論計算を行う場合、正しいプログラミングの知識を持たない者が書いたプログラムによる解析・計算はとても信用されない事に銘記し、正しい知識の習得に努めて欲しい。

5.1 プログラミングのスタイル

ハードウエアが個々の構成部品の集合体で、それらの一つにでも設計ミスや製作ミスがあると正常に動作しないのと同様に、ソフトウエアの場合も小さなプログラムの集合体で、それらがよくデバックされた信頼性のあるものであって初めて大きなプログラムの製作が可能となる。

したがってプログラムを作る場合は、自分が解こうとする問題がどのような道具の組み合せで解決されるべきかをよく考える必要がある。まず抽象的にプログラムの流れを考え[2]、大まかな段階を決め、その後にしだいにこまかな段階に降りていくと良い[3]。そして、自分や友人、さらにはシステムの道具箱を調べ、可能な限り既存のものを使う[4]。どうしても必要な場合にのみ、あらたな道具の開発に着手すればよい。そうすることで、プログラムの開発時間を短縮する事ができる。

道具を作る際には、個々の道具はなるべく単機能にすべきである。こうすることにより、個々の道具は容易に信頼性のあるものにする事ができ、且つ相互の組み合せも容易になり、有機的につながった大きなプログラムを作る事ができる。こうして作られた有益な道具は資産となり、良く整備された道具をどれだけ手元に持つかで、より本質的な問題に取り組めるかが決まる。講義・演習を通じて、良い道具を揃えて欲しい。

どのようなプログラムが**良いプログラム**であるか、あるいはどうしたら**良いプログラム**を書けるかについては、例えば B.W. Kernighan による “The Elements of Programming Style” (邦訳:「プログラム書法」) に、FORTRAN や PL/I のプログラムの具体例とともに気の利いた規則 (rule) が述べられており、参考になる[a]。同じ著者による “Software Tools” (邦訳:「ソフトウエア書法」) ではフィルタの概念の大切さが述べられている。将来計算機のプロを目指す者は、この本 (の序文程度は) を読んでおくと良い。

[a] オブジェクト指向が叫ばれる昨今の状況下では、述べられている規則は目新しい事の無い「当然の事」と感じるかも知れない。もし「当然の事」と感じるのであればそれはそれで素晴しい事であり、“Software Tools” 等のより進んだ書で勉強すれば良い。

[1] debug : bug は虫、debug は虫取り。

[2] フローチャートと呼ばれる、作業や処理の手順を特定の記号を用いて図式的に表現したものがよく用いられる。

[3] この辺は、レポートや論文を書くとき等、多くの事柄に共通している。

[4] ただし、よくデバックされた信頼性のあるものである事が必要である。またその道具の使い方に精通している必要もある (道具も使い方を誤ると大きな間違いを犯す)。

5.2 プログラムの例と実行

プログラムを書く際に最も大切な点は分かり易さである。一見して読むのを放棄したくなるプログラムは大抵誤り (バグ) を含んでいるか、改良しにくいものである。分かり易いプログラムは、

- **読みやすい**
- **構造がはっきりりていて、機能毎に分けられた道具から整然と組み立てられている**

といった特徴を持つ。プログラムそのものがそのまま文章として読めるようなものが一つの理想形である。この理想形を目指しつつ、適宜注釈 (コメント) を付記する事により、プログラムを分かり易いものにするよう努めて欲しい。

5.2.1 プログラムの例

例として、半径を与え、円の面積と球の体積を計算し、結果を画面に表示するプログラムは、

```
      program circle
      implicit none
c constant:
      real PI                                ! 円周率 PI の値を入れる変数
      parameter (PI=3.141593)                ! 円周率 PI を定義
c local variables:
      real radius                            ! 半径の値を入れる変数
      real area,volume                       ! 面積、体積の値を入れる変数
c begin:
      radius = 10.0                          ! 半径として 10.0 を入れる
      area   = PI * radius**2                ! *は乗算の、**はべき乗の記号
      volume = 4.0 * PI * radius**3 / 3.0 ! /は除算の記号
      write(*,*) 'radius = ',radius          ! 半径を画面に表示
      write(*,*) 'area   = ',area            ! 面積を画面に表示
      write(*,*) 'volume = ',volume          ! 体積を画面に表示
      stop
      end
```

と書くことができる。**各行が 7 文字目から始まっている**事に注意せよ。また 1 文字目に c があると、その行は注釈 (comment) 行となりコンパイラは解釈しない。

プログラムはプログラム名の宣言 (program circle) から始まり、定数や変数の宣言が行われた後、本文となる。write(*,*) はその後に書いてある式や変数の値、あるいは文字列を画面に表示する時のおまじないである。最初の*は画面を意味し、2 番目の*は、数の書き方 (**書式**、Format という) を計算機に任せることを意味している。

FORTRAN では、変数を宣言すること無く使用する事が出来る。最初の一文字が I〜N(または i〜n) で始まる変数 (例えば number) は整数、それ以外が実数を表すという決まりになっている。しかし、このような暗黙の了解は間違いを引き起こしやすい為、本講義ではプログラム中の変数全てを宣言する事にする。具体的には implicit none とプログラム名の宣言の後に宣言しておくと、

未定義の変数 (多くの場合はタイプミス) があるとコンパイル時にエラーメッセージがでるため、タイプミス等の誤り (バグのもと) を効果的にふせぐ事が出来る。

5.2.2 プログラムの実行

このプログラムは、

```
/home/teacher/z6wt01in/SAMPLE/circle.f
```

にあるので、各人のディレクトリにコピーしてコンパイル・実行してみて欲しい。
コンパイル、

```
% frt -o circle circle.f
```

を行うと、実行ファイル `circle` が出来るので、

```
% circle↵
```

とすると実行できて、

```
radius = 10.0000000
area = 314.159302
volume = 4188.79053
```

という結果が得られる。

5.3 FORTRAN の約束ごと

プログラムとは、「変数を定義し、変数間で演算し、結果を出力する」と極言する事ができる。変数は読みやすさのため、出来るだけ意味のある名前を付けるように心がける。定数 (円周率や超微細構造定数等) は積極的に `parameter` 文で宣言する[5]。これにより、プログラムが読みやすくなるだけではなく、後で変更する際にプログラム中にちりばめられた数を一々修正せずに `parameter` 文などの定義のみを換えればよく改良が容易となる。

変数以外にも、コーディングを工夫することにより読みやすいプログラムを書くことができる。`mule` を使えばインデント (字下げ) 等は自動で行ってくれ、読みやすいプログラムを簡便に書く事ができる。本講義で用いる拡張版 FORTRAN 77 では 3.7.2 節で述べたような拡張が FORTRAN 77 に対して施されており、自由度の高い (分かりやすい) コーディングが可能となっている。しかしながら次の規則は最低限守らなければならない (既出の事も重要な事は再掲しておく)。

- FORTRAN のプログラムを構成する文 (statement) は 1 行の 7 文字目から 72 文字目までにおさめられなければならない。`mule` を使い (`Fortran`) モードで編集している場合は、カーソルが行頭にある状態で Tab キーを押すと 6 文字のスペースが挿入され、カーソルは 7 文字目に移動するため便利である。
- 6 文字目に`&`などの文字を入れることにより、上の行からの継続行であることを表す。

[5] 例題の様に、定数は大文字にしておくと、プログラムは読みやすいものとなる。

表 5.1: FORTRAN が扱えるデータの型。

型	プログラム中での定義	範囲	有効数字
整数	`integer*2`	-32768 から 32767 まで	
	`integer` または `integer*4`	-2147483648 から 2147483647 まで	
実数	`real` または `real*4`	$0.0, \pm 0.29 \times 10^{-38}$ から $\pm 1.7 \times 10^{38}$ まで	7 桁
	`real*8`	$0.0, \pm 0.29 \times 10^{-38}$ から $\pm 1.7 \times 10^{38}$ まで	15 桁
	`real*16`	$0.0, \pm 0.84 \times 10^{-4932}$ から $\pm 0.59 \times 10^{4932}$ まで	33 桁
複素数	`complex` または `complex*8`		7 桁
	`complex*16`		15 桁
論理型	`logical*1`	真 (`.true.`) または偽 (`.false.`)	
	`logical*2`	を表す論理値	
	`logical` または `logical*4`		
文字列	`character*`サイズ	サイズ分の長さを持つ文字列	

- 1 文字目から 5 文字目までの数字列は行番号 (`goto` 文の飛び先など) を表す。
- 1 文字目に `c` あるいは`*`の文字のある行はコメント行とみなす。また、`!`以降行末までをコメントとして無視する。
- 31 文字までの、長い変数名やサブルーチン名が使える。
- 大文字小文字を区別しない。

5.4 FORTRAN が扱うデータの型

FORTRAN が扱えるデータの型とプログラム中での定義を表 5.1 にまとめる[6]。

ここで `real*4` は**単精度型**、`real*8` は**倍精度型**と呼ばれ[7]、`*4` あるいは`*8` などは実際に使用するバイト数を表す[8]。

単精度の定数は `1.0, 1.0e+00` のように表され、倍精度の定数は `1.0d+00` のように表される。定数であっても有効数字の精度しか持ち得ないため、倍精度型の変数 `x` に値を代入する時は、`x=1.0d+00` のように型通りに記述しないと倍精度が保証されないので注意が必要である[9]。

FORTRAN では異なった型の間での演算が保証されている。例えば `real x,y` および `integer i` と変数が宣言されているとき、`y=x*i` とすると、原則として `i` は一度単精度実数に変換されてから `x` との掛算が行われる。また後述のように、プログラム中で型変換を明示して演算することもできる。

[6]計算機の種類が変わると異なる場合がある。例えば整数では、32 ビットマシンと 64 ビットマシンでは表すことの出来る最大値 (最小値) が異なる。

[7]`real*16` あるいは `complex*16` の演算を行うハードウエアを内臓していない計算機では、これらの変数を用いると実行速度が非常に遅くなるので注意が必要である。

[8]1 ワード (word)=2 バイト (byte)=16 ビット (bit) である。4 バイト=32 ビットである。

[9]例えば、`x=1.0` とすると、小数点以下 9 桁以降にでたらめな数字が入る場合がある。

5.5 プログラムで扱える構造

FORTRANには条件分岐を表す構文としてif(条件)then〜else if(条件)〜else〜end ifが、繰り返しを表す構文としてdo〜end doあるいはdo while(条件)〜end doなどが用意されている。

以下、代表的な構造を概観する。

条件判断を行う

```
      if (a.gt.b) then                  ! aがbより大きい(.gt.)場合
         statement 1                    ! →statement1を実行
      else if (a.eq.b) then             ! aとbが等しい(.eq.)場合
         statement 2                    ! →statement 2を実行
      else                              ! 上記何れの条件にも合致しない場合
         statement 3                    ! →statement 3を実行
      end if                            ! endifと書いても良い
```

else if(elseifと書いても良い)以降の条件判断は、それ以前の条件判断が真の場合は判断されない事に注意せよ。

場合分けを行う

例えば、キーボードから数値を入力し、入力された値によって処理を替える雛型は以下の通り。

```
      integer i                         ! iを整数型として定義
c
      read(*,*) i                       ! キーボードから数をiに読み込む
      if (i.eq.1) then                  ! iが1の場合(iが1に等しい(.eq.))
         statement 1                    ! →statement1を実行
      else if (i.eq.2) then             ! iが2の場合
         statement 2                    ! →statement 2を実行
      else                              ! その他の場合
         statement 3                    ! →statement 3を実行
      end if                            !
```

read(*,*)はその後に書いてある変数に、キーボードから値や文字列を読み込み代入する時のおまじないである。最初の*はキーボードを意味し、2番目の*は、数の読み方(書式)を計算機に任せることを意味している。

単純な繰り返し

以下の例では、iを1,2,3,...,10と変化させながら、中の文(statement 1,statement 2)を10回繰り返し実行する。

```
      do i=1,10                         ! iを1から10まで(1づつ)増やす
         statement 1                    !
         statement 2                    !
      end if                            !
```

statement 1,statemet 2 の中で i を参照する事は出来る (y=x*i など) が、i の値を変える事は出来ない (i=i+1 など) 事に注意せよ。

変化幅は指定する事が可能で、例えば i を 1,3,5,7,9 と 2 毎に変化させるなら、

```
do i=1,10,2                          ! i を 1 から 10 まで 2 づつ増やす
   statement                         !
end if                               !
```

とすれば良い。なお、変化幅は負の値でも良い。

条件が満たされている間、繰り返す

以下の例では、i が 10 以下である間、中の文を繰り返し実行する。

```
i = 1
do while (i.le.10)                   ! i が 10 以下 (.le.) なら以下を実行
   statement 1                       !
   statement 2                       !
   i = i + 1                         !
end if                               !
```

この例では、statement 1,statemet 2 が i の値を変更しなければ「単純な繰り返し」と等価である。do while の場合は、statement 1,statemet 2 で i の値を変えても良い (i の値を変えるか否かが、どちらの構文を使うかの 1 つの目安となる)。

条件が満たされるまで、繰り返す

以下の例では、計算結果 (b = b * 2) がある条件 (関数 calc が判定) になるまで繰り返しの中の文を実行する。

```
logical status,calc                  ! 論理型変数・関数の定義
status = .false.                     ! 変数の初期化
do while (status)                    ! status が真になるまで以下を実行
   b = b * 2                         !
   status = calc(b)                  ! 関数 calc が b の値を判定
end if                               !
```

この例の応用として、ファイルからデータを 1 行ずつ読み込み処理する事を、ファイルの終わりまで行うといったものが挙げられる。

大小符号や等号

FORTRAN では、条件式において大小符号や等号を表 5.2 のように表す。

5.6 サブルーチン・関数

最初に出て来た例題を、半径を 1 から 2 刻みで 10 まで変化させて計算するプログラム (circle2.f) に変更する。同じ処理を何回も繰り返し行う場合は、**サブルーチン**や**関数**にその処理を下請けさ

表 5.2: FORTRAN での関係式の表し方。

関係式	FORTRAN での表現
$a = b$	a.eq.b
$a \neq b$	a.ne.b
$a < b$	a.lt.b
$a \leq b$	a.le.b
$a > b$	a.gt.b
$a \geq b$	a.ge.b
$a \wedge b$	a.and.b
$a \vee b$	a.or.b
$\neg a$	.not.a

せると良い。それ以外でも、書いているメインプログラムやサブルーチンの長さが 100 行を有意に越えたら長過ぎると判断して、プログラムを機能単位に分割する事ができないか考えるべきである。

```
      program circle2
      implicit none
c constants:
      real RADIUS_INI,RADIUS_MAX          ! 半径の初期値と最大値
      parameter(RADIUS_INI=1.0,RADIUS_MAX=10.0)
      real RADIUS_DELTA                   ! 半径の刻み幅
      parameter(RADIUS_DELTA=2.0)
c local variables:
      real radius                         ! 半径の値を入れる変数
      real area,volume                    ! 面積、体積の値を入れる変数
c begin:
      radius = RADIUS_INI                 ! 半径を初期化
      do while (radius.le.RADIUS_MAX)     ! 半径が最大値になるまで繰り返し
         call calc(radius,area,volume)    ! サブルーチンで面積・体積計算
         write(*,*) 'radius = ',radius    ! 半径を画面に表示
         write(*,*) 'area   = ',area      ! 面積を画面に表示
         write(*,*) 'volume = ',volume    ! 体積を画面に表示
         write(*,*)                       ! 空行を画面に表示
         radius = radius + RADIUS_DELTA
      end do
      stop
      end
```

プログラムの本体 (メインプログラム) では、半径に初期値を代入した後、決められたステップ毎に半径の値を増やして面積と体積を計算するサブルーチン calc を呼ぶ。FORTRAN では Pascal と異なり、メインプログラムとサブルーチンの宣言の順番に制限は無く、異なったファイルに定

義されていてもよい[10]。

実際の計算は以下のサブルーチン calc の中で行われる。

```
      subroutine calc(radius,area,volume)
      implicit none
c constant:
      real PI                              ! 円周率 PI の値を入れる変数
      parameter (PI=3.141593)              ! 円周率 PI を定義
c input:
      real radius                          ! 半径の値を入れる変数
c outputs:
      real area,volume                     ! 面積、体積の値を入れる変数
c begin:
      area   = PI * radius**2              ! *は乗算の、**はべき乗の記号
      volume = 4.0 * PI * radius**3 / 3.0 ! /は除算の記号
      return                               ! 呼んだ上位のプログラムに戻る
      end
```

ここで radius,area,volume は**引数**(argument) と呼ばれ、上の例ではサブルーチンへの入力として radius を受け取り、サブルーチンの内部で演算を行い area と volume に結果を代入して返す。サブルーチンへの引数は、

入力、内部で変更を受けるもの、出力

の順に並べるように決めておくと、道具として用いる際に混乱しなくて良い。また、引数はサブルーチンの中で最初に宣言し、自明な場合以外はコメントを付けると道具としての利用価値が上がる。

FORTRAN の場合、サブルーチンへの引数の受け渡しは**アドレス渡し**(call by address) と考えてよい[11]。その際、**引数の数はもとより、引数の型をサブルーチンを呼ぶ側で定義された型とサブルーチン本体側とで一致させなければならない**。但し、引数の名前は同じである必要はなく、

```
      call calc(radius1,area1,volume1)
      call calc(radius2,area2,volume2)
      call calc(radius3,area3,volume3)
```

のように異なった名前の変数と受け渡しを行ってもよい。

5.7 グローバル変数とローカル変数

Pascal ではメインプログラムまたは外側の手続きで定義された変数は、その内側の手続きの中で有効であった。このような変数を**グローバル変数**という。FORTRAN では変数は変数宣言を行ったメインプログラムまたはサブルーチンの中でだけ有効な変数 (**ローカル変数**) である[12]。

[10] サブルーチン (道具) を別々のファイルに分割し、部品毎にコンパイルしてテスト出来る事は、大きなプログラムを書く上で必須の機能である。

[11] アドレスを陽に考えなくても良い点が、FORTRAN を他の言語 (C など) に比べて、初学者にとっての落し穴が少なく、習得が比較的用意なものにしている。

[12] 後述の Common 文を使えばグローバル変数として扱える。

サブルーチンの中で定義された変数は

```
real result
save result
```

のように save と指定しておくと、サブルーチンが再度呼ばれた時に前の値を保存しておいてくれる。また、

```
integer n /1/
```

のようなサブルーチン内の変数の初期化は最初にサブルーチンが呼ばれたときだけ有効である。毎回初期化したい場合は

```
n = 1
```

のようにあらわに代入しなければならない。

5.8 端末からの読み込み

端末から半径を読み込み、これまでと同様に円の面積と球の体積を計算して画面に出力するプログラム circle3.f は

```
      program circle3
      implicit none
c local variables:
      real radius                       ! 半径の値を入れる変数
      real area,volume                  ! 面積、体積の値を入れる変数
c begin:
      write(*,*) 'radius? '             ! 画面に radius?と表示
      read(*,*) radius                  ! 半径を読み込む
      call calc(radius,area,volume)     ! サブルーチンで面積・体積計算
      write(*,*) 'radius = ',radius     ! 半径を画面に表示
      write(*,*) 'area   = ',area       ! 面積を画面に表示
      write(*,*) 'volume = ',volume     ! 体積を画面に表示
      stop
      end
```

と書ける。read(*,*) は端末 (キーボード) から変数を読み込むときに用いる文であり、read(*,*) の後に必要な変数を並べて書いておき、入力時はその順番に従って打ち込めば良い。

プログラムを実行すると分かるが、このままでは radius? と画面に表示した後改行してしまう。改行させないためには、

```
write(*,'("radius?  ",$)')
read(*,*) radius
```

あるいは、

表 5.3: FORTRAN の書式の例。

書式	書式	出力例	備考
整数	I8	32768	8 桁分の場所を確保して右詰め
	I8.6	032768	6 桁分について 0 を空白なら付ける
	I8.8	00032768	0 を前に詰める
実数 (浮動小数)	F15.6	197.326968	符号・小数点を含め全体で 15 桁、そのうち小数点以下に 6 桁使用
	E15.6	0.197327e+03	10 のべき乗の形で表し、小数点以下に 6 桁使用
	G15.6	0.123456e-01	数の大きさに応じて E(e)Format、F(f)Format を使い分ける
		1.234567	
文字列	A		文字列の取り扱いの節を参照
空白	nX		n 文字分の空白

```
write(*,'(a,$)') 'radius?  '
read(*,*) radius
```

とすると、ユーザーの入力が終ってから改行が行われる。

演習 プログラム circle3.f を、

```
/home/teacher/z6wt01in/SAMPLE/circle3.f
```

に置くので、calc.f と共にコンパイルして実行してみよ。また、ユーザーの入力が終ってから改行が行われるようにプログラムを修正してみよ。

5.9 結果をファイルに書き出す

計算結果を画面で確認した後、それをファイルにしまいたい時がよくある。複数のファイルを扱う時は後述の open 文を使えば良いが、単純な場合には、以下のように**リダイレクション**を用いて出力を振り替えればよい。

```
% circle > circle.dat
```

上記の例では write(*,*) 文の出力が、ファイル circle.dat に書き込まれる。

5.10 書式を整えて書き出す–format–

計算結果を出力する時に、表などの型で書式を整えたい時がある。その場合、write 文に埋め込む書式が用意されている。詳しくは参考書を参照する事にして、よく使うものを表 5.3 にまとめる。

他にもタブ揃えに似た書式も可能で、

```
write(*,'(X,I6,T25,F8.5)') id,value
```

は id の値を 6 桁の幅で書いた後、25 文字目まで飛んで (T25) から value を書く。

5.11 配列・行列を用いた計算

配列・行列の宣言

1 次元の配列、2 次元の配列 (行列) の宣言は以下のとおり。

```
real array(10)                          ! 10 個の要素を持つ配列
real matrix(2,2)                        ! 2 行 2 列の行列
real matrix(3,3)                        ! 3 行 3 列の行列
```

行列の和

2 つの 3 行 3 列の行列 a,b を与えた時、その和を c に入れるプログラムは以下のように書ける。

```
real a(3,3),b(3,3),c(3,3)               ! 3 行 3 列の行列
integer i,j
do i=1,3                                ! 行に対するループ
  do j=1,3                              ! 列に対するループ
    c(j,i) = a(j,i) + b(j,i)
  end do
end do
```

行列の積

2 つの 3 行 3 列の行列 a,b を与えた時、その積を c に入れるプログラムは以下のように書ける。

```
real a(3,3),b(3,3),c(3,3)               ! 3 行 3 列の行列
integer i,j,k
do i=1,3                                ! 行に対するループ
  do j=1,3                              ! 列に対するループ
    c(j,i) = 0.0
    do k=1,3
      c(j,i) = c(j,i) + a(j,k)*b(k,i)
    end do
  end do
end do
```

行列のサブルーチンや関数との受け渡し

3 行 3 列の行列 a をサブルーチン sub に渡すには、メインプログラムでは、

```
real a(3,3)                             ! 3 行 3 列の行列
...
call sub(3,a)                           ! sub への引渡し
```

とし、サブルーチン sub では

```
      subroutine sub(n,a)
      implicit none
      integer n
      real    a(n,n)
      ...
```

とすれば良い[13]。

サブルーチン sub は、行列の大きさをあらわに指定して

```
      subroutine sub(a)
      implicit none
      real    a(3,3)
      ...
```

と書くこともできるが、最初の例の方が行列のサイズの変更に際しプログラムの変更を要しないので良い作法である。

5.12 組み込み算術関数

FORTRAN には標準で多くの算術関数が組み込まれており、特別な宣言をしたり、リンクの際に特別に指定しなくても使う事ができる。多くの場合、引数の型のうち単精度と倍精度は自動的に認識されるので、例えば sin(1.0E0) は単精度に、sin(1.0D0) は倍精度になる[14]。表 5.4 によく用いる算術関数をまとめる。

5.13 文字列

FORTRAN では character 文により文字列を表す変数を定義する。parameter 文での定義や /.../による初期値の設定などは他の変数と同様で、以下のようにすれば良い。

```
c constant
      character*6 STRING                    ! 6 文字の変数
      parameter(STRING='Hello!')            ! 文字を定義
c local variables
      character*80 line/'Initial String'/ ! 80 文字の変数の初期化
      character*80 line2,line3              ! 80 文字の変数
      character*32 file_name(20)            ! 32 文字の変数を 20 個用意
```

変数への代入も他の変数と同様に、

```
      line2 = 'This is a pen.'              ! 文字の代入
      line3 = STRING                        ! 変数から変数への代入
```

などとすれば良い。

[13]整合配列という。
[14]但し、戻り値は適した型の変数で受ける必要がある。

表 5.4: FORTRAN の組み込み算術関数の例。

関数名	戻り値
sqrt(x)	$\sqrt{x}$
log(x)	$\log_e x$
log10(x)	$\log_{10} x$
exp(x)	e^x
sin(x)	$\sin x$ (x の単位はラジアン)
sind(x)	$\sin x$ (x の単位は度)
cos(x)	$\cos x$ (x の単位はラジアン)
cosd(x)	$\cos x$ (x の単位は度)
tan(x)	$\tan x$ (x の単位はラジアン)
tand(x)	$\tan x$ (x の単位は度)
asin(x)	$\sin^{-1} x$ (結果はラジアン)
asind(x)	$\sin^{-1} x$ (結果は度)
acos(x)	$\cos^{-1} x$ (結果はラジアン)
acosd(x)	$\cos^{-1} x$ (結果は度)
atan(x)	$\tan^{-1} x$ (結果はラジアン)
atand(x)	$\tan^{-1} x$ (結果は度)
sinh(x)	双曲線正弦
cosh(x)	双曲線余弦
tanh(x)	双曲線正接
abs(x)	$\|x\|$
iabs(n)	$\|n\|$(引数は整数)
int(x)	x の整数部
nint(x)	x に最も近い整数
float(n)	整数 n を単精度変数 (real*4) に変換
real(x)	整数を含む変数 x を単精度変数 (real*4) に変換
dble(x)	整数を含む変数 x を倍精度変数 (real*8) に変換
qext(x)	整数を含む変数 x を 4 倍精度変数 (real*16) に変換
min(x,y,...)	$x, y, \ldots$ の最小値。引数の数はいくつでもよい
max(x,y,...)	$x, y, \ldots$ の最大値。引数の数はいくつでもよい
mod(x,y)	x を y で割った余り
ishft(n,m)	整数 n を m ビットシフト ($m > 0$ で左、$m < 0$ で右)
iand(n,m)	n と m に対してビット毎に論理 AND をとる
ior(n,m)	n と m に対してビット毎に論理 OR をとる

5.14 複素数

複素数は、科学技術計算用語として FORTRAN の得意とするところである[15]。以下に、簡単に複素数の扱い方をまとめる。

宣言・初期化・代入

```
c constant:
      complex CI                          ! 複素数の変数
      parameter(CI=(0.0,1.0))             ! 虚数単位の定義
c local variables:
      real    y
      complex phi,b                       ! 複素数の変数
      complex a/(1.0,-1.0)/               ! 複素数の変数の初期化
      b = (1.0,4.0)                       ! 複素数の変数への代入
```

四則演算・組み込み関数

```
      phi = CI * exp(a)/a +b**a * sin(b)
      y   = abs(phi)**2
```

exp や sin などの関数は、引数が複素数であれば複素演算であると了解される。複素数を引数とする関数 abs は、期待通り引数の絶対値を計算し、結果を実数で返す。

複素関数の定義

例えば、複素数 cx,cy の積の log を返す複素関数 cz は以下のように書ける。

```
      complex function cz(cx,cy)
      implicit none
c inputs:
      complex cx,cz
c begin:
      cz = log(cx*cy)
      return
      end
```

実数と複素数の変換

実数と複素数を変換する関数として、以下のように real,aimag,cmplex,conjg が用意されている。

[15]複素数は 2 つの要素をもつ配列であるが、演算においてその事を意識する必要なく使用できる点等において優れている。

```
c constant:
      complex CI                          ! 複素数の変数
      parameter(CI=(0.0,1.0))             ! 虚数単位の定義
c local variables:
      real    ar,ai
      complex cx,cy
c begin:
      ar = real(CI)                       ! 実部を取り出す
      ai = aimag(CI)                      ! 虚部を取り出す
      cx = conjg(CI)                      ! 複素共役を作る
      cy = cmplx(ar,ai)                   ! 実部と虚部から複素数を作る
```

計算・メモ用余白