東北大学サイバーサイエンスセンター講習会
第8回 Fortran入門（2021年9月10日）

第１部

Fortranプログラミングの基本

日本原子力研究開発機構　田口 俊弘

本講習会の基本方針

- Fortranという言語を利用して，主として数値計算を目的としたプログラムを書く手法を説明する
- プログラミングの初心者でもすぐにプログラムが書けるように，例題を出し，その解答例を示しながら説明する
- 計算機のしくみを考えながら，高速なプログラムになるような技法を紹介する
- 数値計算を効率良く行うプログラミング手法もいくつか紹介する
- 将来，計算機シミュレーションのような複雑で長いプログラムになってもメンテナンスをしやすいような書式を紹介する
- このため，文法的には必ずしも書く必要があるとは限らない書式も含まれている

本講習会の進め方

- １部１時間（質疑，休憩込み）の３部構成
- まず大まかなプログラムの書き方を教える
- 次に例題を示して，その解答を示しながら書き方のコツを教える
- その後で，より詳しい文法を教える
- 資料は少し細かく書いているので，時間が足らない場合は省略する
- 講習会では教える時間がないが，より多様なプログラムを書くための「知っておくと便利な文法」をいくつかセレクトして補足説明として付けている．参考にして自習してもらえたらと思う
- さらに進んだ勉強をしたい場合は，東北大学からもう少し詳しい内容を含めた解説書をダウンロードできる
- 数値計算法も含めて勉強したいときには拙著もあります

第１部の講習内容

１．Fortranの基礎知識

２．メインプログラムの書き方

３．変数の使い方

４．数値計算の基本的テクニック

付録：知っておくと便利な文法１

1．Fortranの基礎知識

```
.........
write(*,*) 'Start calculation'
do ns = 1, ntime+nmd
   nsion = mod(ns,ionstep)
   if (ns >= nostart) nst = 1
   if (mod(ns,ndstep) == 0) write(*,*) 'ns=',ns
   call stepbfield
   if (nst /= 0) then
      call eaccelerate
      call evelocity
      if (nsion == 0) then
         call iaccelerate
         call ivelocity(0)
      endif
      call makeecurrent
      call emovehalf
      call makeedensity
      if (nsion == 0) then
         call makeicurrent
         call makeidensity
      endif
   endif
.........
```

Fortranの特長

- 数値計算用のコンパイラとして古くから存在する
- スーパーコンピュータで高速計算をするときは，Fortranを使うことが多い
- 数値計算用として便利な書式が用意されている
- Fortran95レベルであればパソコン用のフリーコンパイラがあるので自由に使える
- 比較的エラーが見つけやすい
- プログラムの書式が，少し柔軟性に欠ける（古い）

Fortranの歴史

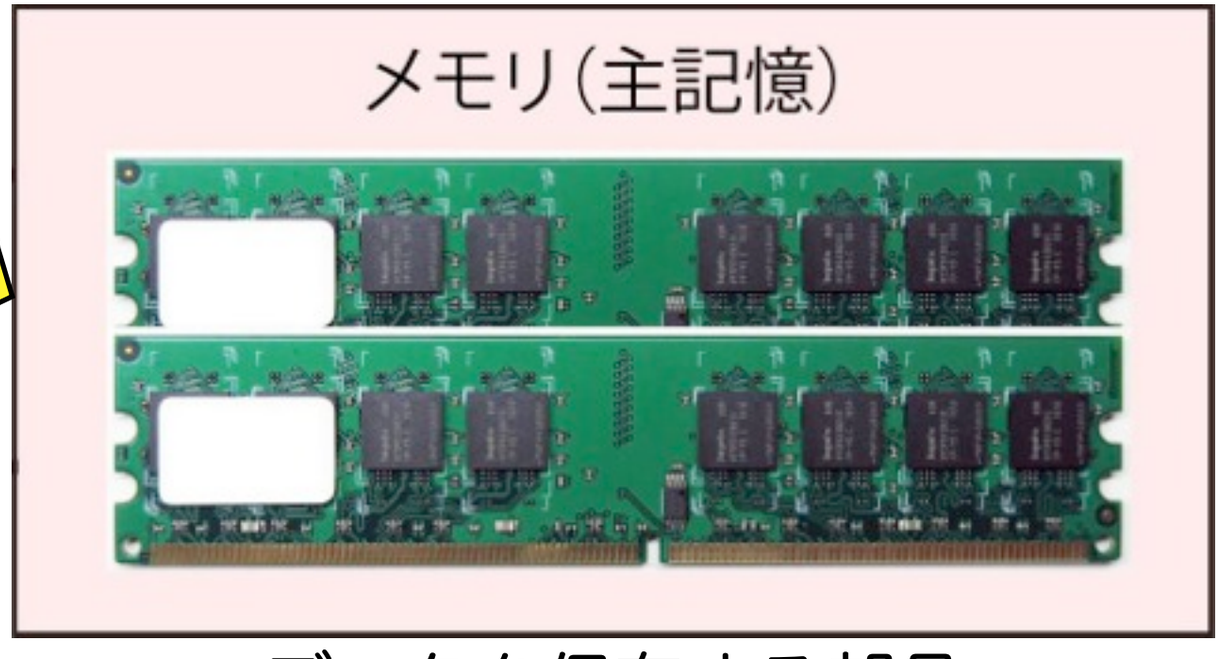
- FORmula TRANSlation（数式変換）が語源
- 1956年に最初のマニュアルリリース
- 1957年に最初のコンパイラ開発
- FORTRAN IV　（私が最初に学んだバージョン）
- FORTRAN 77　（構造化プログラミングが可能に）

- Fortran 90　（大幅に改訂，プログラミングが楽になった）
- Fortran 95　（マイナーチェンジ）　→　**今回の学習レベル**
- Fortran 2003（オブジェクト指向導入）
- Fortran 2008（並列プログラミングが可能に）
- Fortran 2018（……）

コンピュータ内部の主要部品

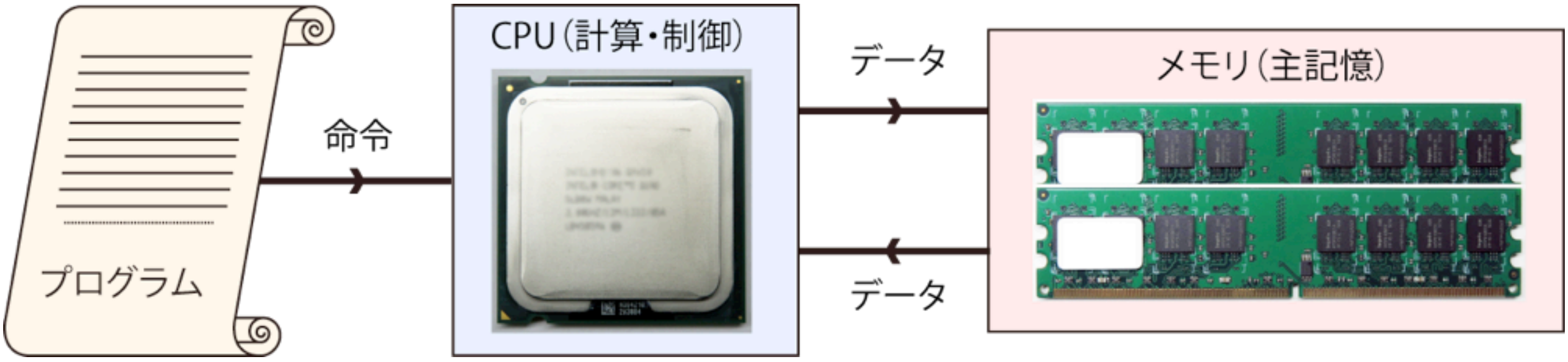
自作のコンピュータ

命令に応じて
計算や制御を行う部品



データを保存する部品

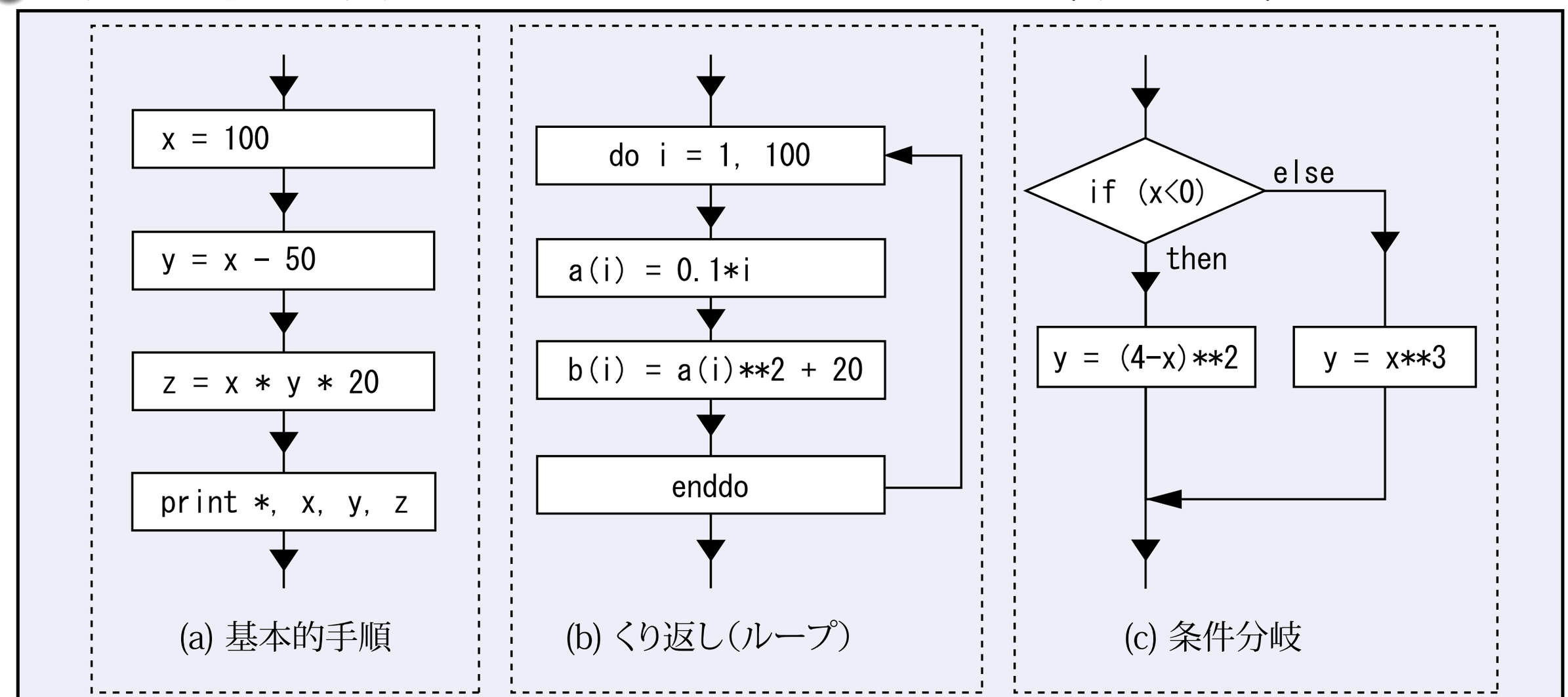
コンピュータとプログラムの関係



- プログラムを一行ずつ読み込んで実行する動作を自動的に行う
- 計算と制御を行うのが「CPU（Central Processing Unit）」
- データを一時保存するのが「メモリ（主記憶装置）」
  でプログラム上では「変数」
- 周辺機器（キーボード，ディスプレイ，ハードディスクなど）と
  のやりとりは「読み書き，入出力」のような付加的動作

コンピュータにおける基本的動作

- 「プログラム」もメモリに保存されていて，基本的には一つずつ順番に読み込んで，書かれている命令を逐次実行していく
- ジャンプ命令を使えば，実行開始場所を一気に変更することもできる
- バックしてくり返し動作をさせることもできる（ループ）
- 条件に応じて異なる動作をさせることもできる（条件分岐）



作成したプログラムをコンピュータで実行するまでの流れ

プログラム

↓ コンパイル（高級言語を機械語に翻訳）

機械語

↓ リンク（ライブラリとの結合）

実行形式

これが，アプリケーション

→ 実行

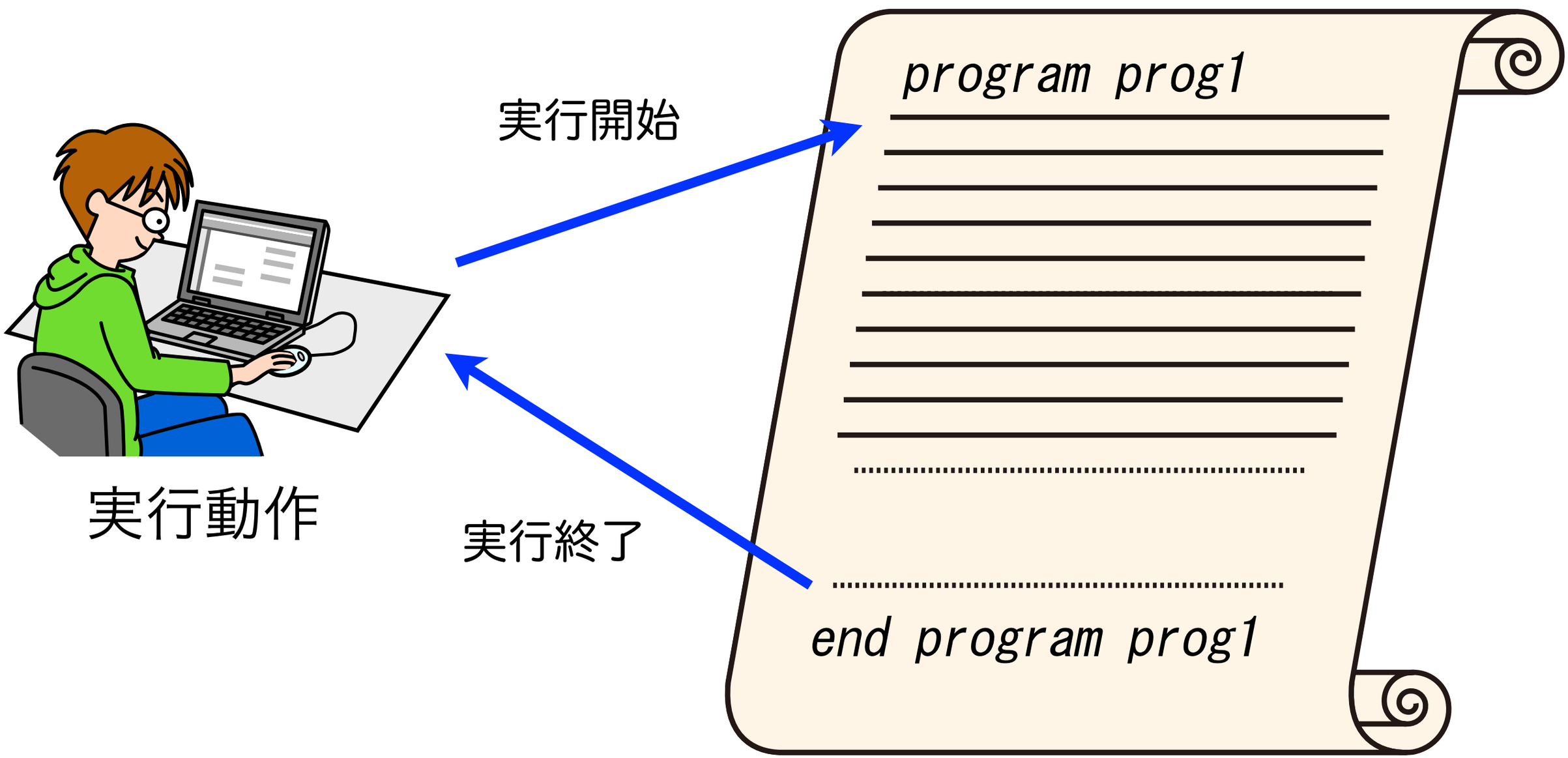
代表的なフリーのコンパイラ gfortran を使ったコンパイルから実行まで

```
$ gedit sample1.f90 &                              （プログラム作成）
$ gfortran -O -fdefault-real-8 sample1.f90         （コンパイル・リンク）
$ ./a.out                                          （プログラム実行）
```

1. gedit等のエディタを使って，拡張子「.f90」または「.f95」を付けた名前のファイルにプログラムを書き込む.
2. gfortranコマンドを使って，コンパイルとリンクを行う.
3. このときのオプションで，-O は最適化用，-fdefault-real-8 は自動倍精度化用で，どちらも付けた方が良い.
4. コンパイルとリンクが成功すると「a.out」という名の実行形式ファイルが作成される.
5. a.out を実行する.

◎ オプションで，実行形式の名前を a.out 以外にすることも可能

2．メインプログラムの書き方



プログラムの一般的書式

動作指示語 ＋ 動作制御パラメータ

```
real x,y,z
do i = 1, 100
print *,' x =  ',x(i),y(i)
if (x > 5) y = x**2
```

代入文

```
x = 100 + y/10
a(i) = b(i)*z(i, j*2) + cos(y)
```

動作指示語のみ

```
stop
return
exit
cycle
```

プログラムの実行開始後，最初に動作するのが
メインプログラム

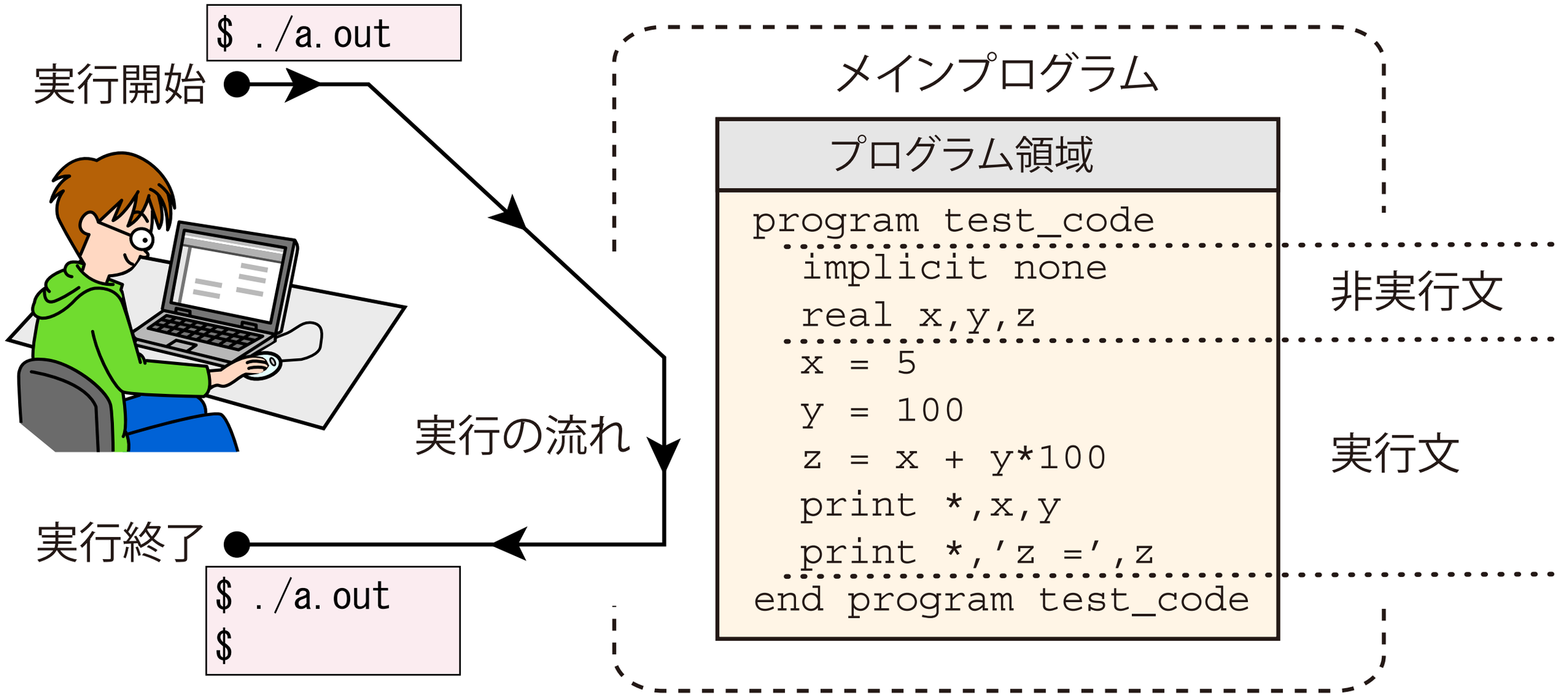
☆メインプログラムは program文と end program文の間に書く

```
program プログラム名      ← program文
  .......
  .......
  .......
  .......
end program プログラム名      ← end program文
```

- program文は書かなくてもメインプログラムと認識されるが，書いた方が良い
- 「end program プログラム名」は「end」だけでもエラーではないが，書いた方が良い
- プログラム名は，先頭がa～zのどれかで，後は数字が混じっても良い
  また，スペースは入れられないが，アンダースコア “_” は入れられる
- プログラム名では，大文字・小文字の区別はしない
- 途中で終了したいときには，stop文を書く

```
stop          ← stop だけの文が，stop文
```

メインプログラムの構造と動作の流れ



- 実行文とはコンピュータの動作を示す命令文で，これが本体
- 非実行文とは，後で説明する宣言文のように，実行するのに必要な準備に関する記述
- 実行を開始すると最初の実行文から順次１行ずつ実行する
- 最後のend program文に到達したらプログラムの動作が終了する

プログラムを書いてみよう

1-A　電卓的プログラム

例題1-A

以下の計算をFortranプログラムを使って計算し，結果を出力せよ

① $100 \times 25 + \dfrac{78}{30} - 72 \times 83$

② $-\dfrac{3.14 \times 2.3^2}{4 \times 0.28^3}$

③ $0.125 \times 10^{-5} \times (3.1 \times 10^{15} - 5.23 \times 10^{18})$

☆プログラム作成時のヒント☆

★ とりあえず出力するときには print文を使う

```
print *, 数値1, 数値2, ...
```

★ 最初の「*」 は「標準出力形式」を表しているので必ず書く

★ 数値１，数値２，の順に横並びで１行に出力する

1-A　電卓的プログラム の解答例

```
program answer1a
    print *, 100*25 + 78.0/30.0 - 72*83
    print *, -3.14*2.3**2/(4*0.28**3)
    print *, 0.125e-5*(3.1e15 - 5.23e18)
end program answer1a
```

☆書き方のポイント☆

★ 四則演算は，ほぼ数学的な記述で書ける

★ 掛け算は「*」で，割り算は「/」で，べき乗は「**」で表す

★ 100とか-72のような整数はそのまま書いても良い

★ 小数点付きの数字，-3.14とか，0.28もそのまま書ける

★ ただし，整数の割り算には注意がいるので，割り算を書くときは小数点を付ける（理由は後で説明する）

★ $A\times10^B$という指数表示の数値は，1.6e18，3.14e-7 のようにAeBと書く

★ 1.6e18を1.6*10.0**18と書くと，計算を実行するので時間がかかる

★ print文は１回の実行あたり１行ずつ出力する

★ 内部に記述するプログラムは，program 文より少し右にずらす（字下げ）

基本的な演算の書き方

演算記号	演算の意味	使用例	使用例の意味
+	足し算	x+y	x + y
−	引き算	x−y	x − y
*	掛け算	x*y	x ×y
/	割り算	x/y	x ÷y
**	べき乗	x**y	x^y
−	マイナス	−x	− x

◎ 演算の優先順位は，数式と同じ

かっこ ＞ べき乗 ＞ 乗算または除算 ＞ 加算または減算

◎ ただし，同レベルの演算は左から行うので注意が必要

例えば，$f = x + \dfrac{y}{ab}$ をプログラムで書く場合

`f = x + y/a*b` と書くと `f = x + (y/a)*b` を意味する

分母を ab にしたいなら `f = x + y/(a*b)` と書かねばならない

コンピュータの数値には「型」がある！

- コンピュータ内部では，計算に使う数値に「型」があるので，使い分けなければならない

（1bitとは２進数１桁のことなので，4bitなら2^4個の数値を表せる）

1.整数型（4byte）

★ 1とか，512とか，-1123とか，小数点のない数値

★ -2^{31}～$2^{31}-1$ の範囲の数値しか扱えない

◎ 1byte=8bitなので，4byteなら2^{32}まで識別可能

★ 足し算，引き算，掛け算は厳密に計算する

◎ 最大値が比較的小さいので大きな数の掛け算には気をつけよう

★ 割り算は切り捨てになる

◎ 例題の78÷30を78/30と書いたら答えは 2 になる

★ 数値の比較が確実に行える

基本的に数値計算は実数型で行う

2.実数型

★ 10.05，-0.25，のように小数点を付けた数値

◎ 23.0 は，23．でも良い

★ 5×10^{20}や1.6×10^{-19}のような指数表示の数値を表現するときは
5e20とか1.6e-19のようにAeBの形で書けるが，これも実数型

★ 数値を比較するときは注意が必要！

◎ 例えば，0.1を10個加えた結果は 1.0 に等しくなるとは限らない

☆ これは，コンピュータが２進数で計算しているため，
0.1 は無限小数になり，コンピュータ内部では近似値だから

★ 実数型と整数型の演算結果は実数型になる

◎ このため，2/3*1.5 と 1.5*2/3 は，結果が異なる
2/3*1.5 → 0*1.5 → 0.0， 1.5*2/3 → 3.0/3 → 1.0

☆ ちなみに，0.5=2^{-1} なので，２進数でも有限小数
よって，1.5*2 は 3 に等しい

実数型には有効数字の異なる単精度実数と倍精度実数がある

- Fortranで特に指定がなければ単精度（4byte），有効数字7桁程度
- 数値計算は倍精度（8byte）で行った方が良い，有効数字15桁程度
- 7桁あれば十分と思うかもしれないが，数値計算プログラムでは様々な計算を大量に行うことが多いので，精度が高い数値を使った方が良い
- 倍精度計算が遅いとは限らないので速度を気にする必要はあまりない
- 最近のFortranコンパイラではオプションでデフォルトの実数型を倍精度にすることができるので，この講習会では区別をせず「実数型」として説明する

◎ gfortran の -fdefault-real-8 が自動倍精度化オプション

プログラムを書いてみよう

1-B　関数電卓的プログラム

例題1-B

以下の計算をFortranプログラムを使って計算し，結果を出力せよ

① $0.5\sin(3.4) - 0.115\cos(0.18)$

② $\dfrac{50\tan(2.1)}{80\sqrt{2}}$

③ $3.1\times10^4 e^{-3} - 523.5\log_e 2.8 + 40^{3.2}$

ここで，sin, cos, tan の引数値はラジアンを単位とする

1-B　関数電卓的プログラム の解答例

```
program answer1b
    print *, 0.5*sin(3.4)-0.115*cos(0.18)
    print *, 50*tan(2.1)/(80*sqrt(2.0))
    print *, 3.1e4*exp(-3.0)-523.5*log(2.8)+40.0**3.2
end program answer1b
```

☆書き方のポイント☆

★ 関数の引数を記述するとき，両かっこは必ず必要

★ sin, cos, tan の引数はラジアンが単位

★ $\sqrt{x}$ は sqrt(x) と書く

★ log(x) は自然対数．常用対数のときは log10(x) を使う

★ ** は実数のべき乗も可能

★ 実数関数の引数は実数で書くこと

◎ **sqrt(2)やexp(-3)と書けば，エラーになる可能性がある**

Fortranで使える数学関数（組み込み関数）

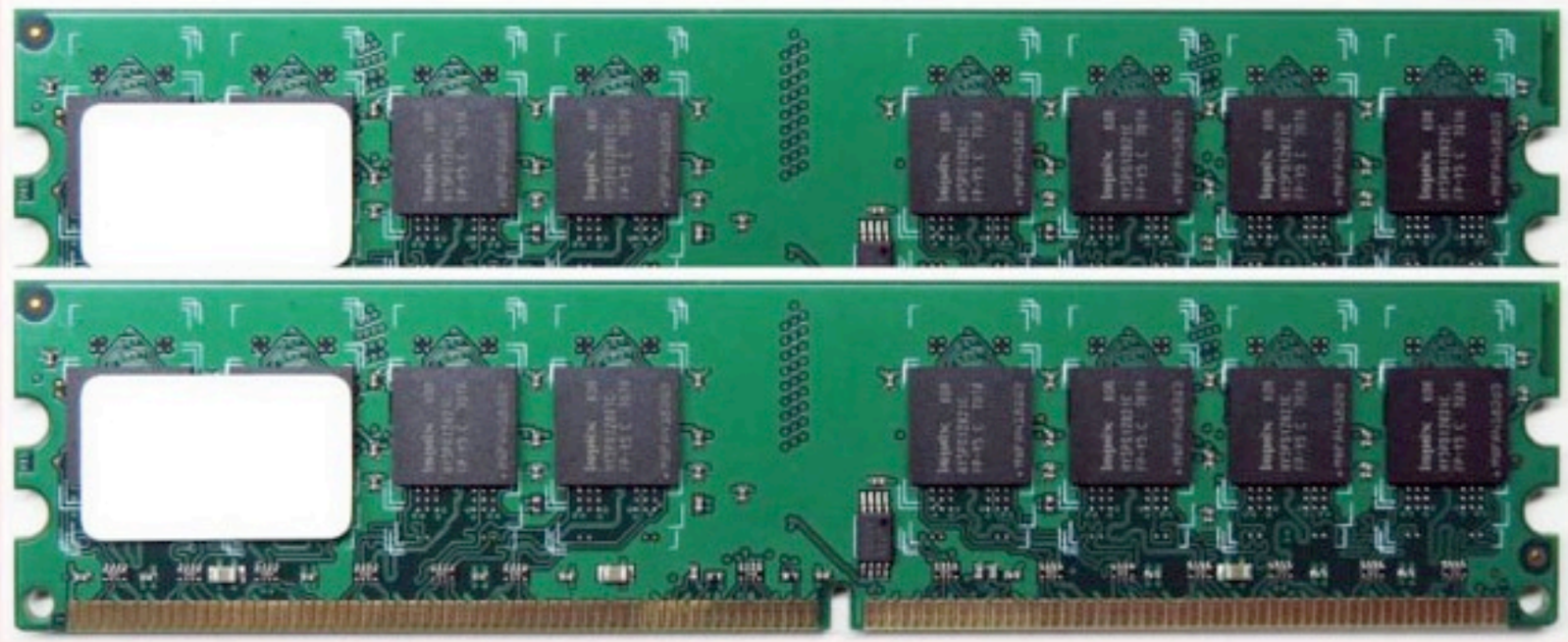
組み込み関数	名　称	数学的表現	必要条件	関数値 f の範囲
sqrt(x)	平方根*	$\sqrt{x}$	$x \geqq 0$	
sin(x)	正弦関数*	$\sin x$		
cos(x)	余弦関数*	$\cos x$		
tan(x)	正接関数	$\tan x$		
asin(x)	逆正弦関数	$\sin^{-1} x$	$-1 \leqq x \leqq 1$	$-\pi/2 \leqq f \leqq \pi/2$
acos(x)	逆余弦関数	$\cos^{-1} x$	$-1 \leqq x \leqq 1$	$0 \leqq f \leqq \pi$
atan(x)	逆正接関数	$\tan^{-1} x$		$-\pi/2 < f < \pi/2$
atan2(y,x)	逆正接関数	$\tan^{-1}(y/x)$		$-\pi < f \leqq \pi$
exp(x)	指数関数*	e^x		
log(x)	自然対数*	$\log_e x$	$x > 0$	
log10(x)	常用対数	$\log_{10} x$	$x > 0$	
sinh(x)	双曲線正弦関数	$\sinh x$		
cosh(x)	双曲線余弦関数	$\cosh x$		
tanh(x)	双曲線正接関数	$\tanh x$		

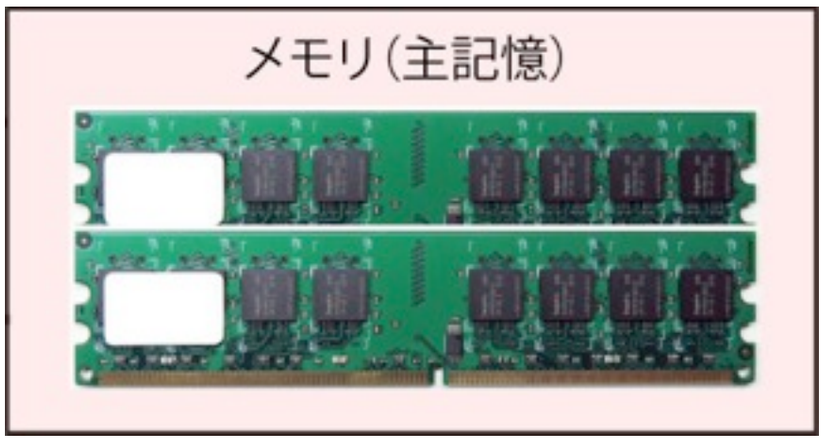
★ acosやatanは，πを計算するときにも使える

★ 表の＊の付いた関数は引数の数値型に対応した関数計算をする

3．変数の使い方

メモリ（主記憶）





変数とはなにか

- プログラム中での「変数」は，数値を入れる「箱」
- 「箱」を指定するのが「変数名」
- その実体は，数値を保存するメモリの番地（メモリアドレス）
- プログラムで必要なメモリを確保するのが「宣言」
- メモリに数値を保存する，すなわち箱に入れるのが「代入」
- もしすでに数値が入っている変数に別の数値を代入したら，数値が入れ替わる．すなわち，以前の数値は消える

変数の宣言によるメモリの確保

1.暗黙の宣言を禁止する（全ての変数は宣言して使う）

```
program code1
   implicit none      ← 最初に必ずこの文を挿入する
   .........
```

★ この文を書いておくと，宣言しない変数があるとエラーになる

★ 予期せぬ変数の書き間違いを防ぐ

★ 変数の型を常に意識することができる

2.整数型変数の宣言

```
integer 変数１,変数２,...
```

3.実数型変数の宣言

```
real 変数１,変数２,...
```

☆ 宣言文は，非実行文なので，プログラムの最初の方に書かねばならない

代入文は変数に数値を保存する「動作」

変数への保存はイコールで記述する（代入文）

変数 = 数値や数式

この式は，右辺を計算し，結果を左辺の変数に保存するという動作を表す

代入のイメージ

```
x = 10
y = 3.14*x**2
```

real x, y

	メモリアドレス	メモリ領域
x →	101:	-5.0
	102:	0.0
	103:	0.1
y →	104:	10.4
	105:	1e-3
	....	

real x, y

	メモリアドレス	メモリ領域
x →	101:	-5.0
	102:	10.0
	103:	0.1
y →	104:	314.0
	105:	1e-3
	....	

★ 数値を代入すると，その前に代入されていた数値は消える

★ もし変数に数値を一度も代入しないで使用すると，どんな数値が入っているかわからないので結果が保証されない（０の可能性はある）

代入文の左辺と右辺では変数の意味が異なる

変数 ＝ 数値や数式

- 右辺は，定数や変数の「中味」を使って計算する動作
- 左辺は，変数という箱の場所（アドレス）指定
- 右辺を計算した後に，その結果を左辺の変数に保存する

このため，

```
x = 9
x = x + 10
x = sin(x) + cos(x)
x = x**3 + 5*x**2 + 1
```

のように書いても問題はない

★ ただし，１行ごとに変数 x の中味が更新されているので，
３行目と４行目の右辺の x は 9 ではない！

☆こういうプログラム特有の書き方に慣れて下さい！

プログラムを書いてみよう

1-C　変数を使った計算

例題1-C

以下の計算手順をFortranプログラムを使って実行せよ

① まず変数 deg に $\dfrac{\pi}{180}$ の数値を代入する

② $50\tan 20^\circ - 4.8\cos 38^\circ$ の結果を変数 x に代入する
この計算を行うときに，変数 deg を利用する

③ 変数 y に $\dfrac{x^3}{3} + \sqrt{2x}$ の値を代入する

④ x と y の結果を１行で出力する

1-C　変数を使った計算 の解答例

```
program answer1c
    implicit none
    real deg, x, y
    deg = atan(1.0)/45     ! arctan(1.0) = π/4
    x = 50*tan(20*deg) - 4.8*cos(38*deg)
    y = x**3/3 + sqrt(2*x)
    print *, 'x = ', x, '   y = ', y
end program answer1c
```

☆書き方のポイント☆

★ program文の次に必ず「implicit none」を書く.

★ プログラム中で使う変数は全て宣言する

★ そのとき，必ず数値型を意識する

★ 関数の引数には数式を入れることも可能

1-C　変数を使った計算 の解答例で使ったテクニック

```
program answer1c
   implicit none
   real deg, x, y
   deg = atan(1.0)/45      ! arctan(1.0) = π/4
   x = 50*tan(20*deg) - 4.8*cos(38*deg)
   y = x**3/3 + sqrt(2*x)
   print *, 'x = ', x, '   y = ', y
end program answer1c
```

☆このプログラムで使った，良く使うFortranの書式

★ 「!」を書くと，それ以降の文字は行末まで無視される（コメント文）

★ 'x = ' のように，２個の「'」ではさんだ文字は「文字列」として扱われ，print文で数値の代わりに出力すると，スペースも含めてそのままの文字が出力される

◎私のパソコンでやってみた出力結果は以下の通り（倍精度）

```
 x =    14.4160600959979    y =    1004.03149196539
```

変数の用途と名前の選び方

☆変数はプログラムの主役である

- プログラム中の式において，2倍とか3で割るなどのような公式中の決まった数値以外の数値は基本的には変数で書く
- π や物理定数のような有効数字の大きな数値も変数に代入して使った方がわかりやすい
- よって，定数を使うのは変数へ代入するときだけにする方が良い

☆変数名の選び方

- 頭文字は a～z のローマ字，それ以外は数字や “_” を交えても良い

  ◎ a，b1，abc，ab12cd，abc_123，など

- Fortranでは，大文字と小文字は区別をしない

  ◎ ABC と abc と Abc は，全て同一変数

- 実数型変数は最も良く使うので，名前を特に制限する必要はない
- 整数型変数は用途が限られているので，頭文字を決めておく方が良い
  例えば，整数をイメージする i,j,k,l,m,n に限定する

  ◎ n，i0，jmax，k3s1，など

数値型の異なる変数への代入

代入する場合には右辺の数値を左辺の数値型に変換して代入する

実数型変数＝整数型数値	は右辺の値を実数として代入
整数型変数＝実数型数値	は右辺の値の小数点以下を切り捨てた整数を代入

例えば,

```
integer m, n, k
m = 10.31
n = -78.5/32
k = sin(2.0)
```

と書けば, m=10, n=-2, k=0 である

４．　数値計算の基本的テクニック

$$\tan\theta = \frac{\sin\theta}{\cos\theta}$$

$$y = \frac{ax - b}{cx^2 + d}$$

$$V = \frac{4}{3}\pi r^3$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$x^2 + y^2 = r^2$$

$$S = \pi r^2$$

$$E = mc^2$$

$$y = a_3x^3 + a_2x^2 + a_1x + a_0$$

プログラムを書いてみよう
1-D　多項式計算

例題1-D

x=1, 3, 10 の3個の数値に対して，以下の多項式の値を計算し，
それぞれの結果を出力せよ.

$$y = 5x^4 + 3x^3 - 7x^2 - 10x + 1$$

1-D　多項式計算 の解答例

```
program answer1d
    implicit none
    real x, y
    x = 1
    y = (((5*x+3)*x-7)*x-10)*x+1
    print *, 'x, y = ', x, y
    x = 3
    y = (((5*x+3)*x-7)*x-10)*x+1
    print *, 'x, y = ', x, y
    x = 10
    y = (((5*x+3)*x-7)*x-10)*x+1
    print *, 'x, y = ', x, y
end program answer1d
```

☆書き方のポイント☆

★ 整数べき乗「x**2」などは，「x*x」のように掛け算で書いた方が速い

★ 掛け算の回数を減らせば速くなる

★ 多項式の計算は，このようにかっこでくくった形に変形すると掛け算が少なくてすむ（horner法）

数値計算における注意事項

1．計算機には，演算の得手・不得手がある

加減算　≫　乗算　≫　除算　≫　べき乗

例えば，次のような変形をすると速くなる

```
x = a/b/c   →   x = a/(b*c)
```

```
x = a**2 + b**3   →   x = a*a + b*b*b
```

★ 関数計算も遅いので，変数に代入するなどして同じ計算をくり返さないようにした方が速い

2．差の小さい数字の引き算をすると有効数字が減る

```
2000.0612 - 2000.0000 = 0.0612
```

８桁が３桁になった！

★ これを「桁落ち」という

★ 桁落ちをできるだけ防ぐためにも倍精度実数を使う方が良い

プログラムを書いてみよう

1-E　２次方程式の解

例題1-E

① a=18，b=313, c=-35 として，次の２次方程式の２解を計算せよ.

$$ax^2 + bx + c = 0$$

② 解を計算したら，検算をして２次式が０に近いことを確かめよ.

1-E　２次方程式の解 の解答例

```
program answer1e
    implicit none
    real a, b, c, d, x1, x2, y1, y2
    a = 18
    b = 313
    c = -35
    d = sqrt(b*b-4*a*c)
    x1 = (-b+d)/(2*a)
    x2 = (-b-d)/(2*a)
    print *, 'x1, x2 = ', x1, x2
    y1 = (a*x1+b)*x1+c
    y2 = (a*x2+b)*x2+c
    print *, 'y1, y2 = ', y1, y2
end program answer1e
```

☆書き方のポイント☆

★ 関数計算（この場合はsqrt(x)）は時間がかかるので，共有できる計算値は１回の計算ですむようにする.

★ もし b の値が非常に大きいときには，どちらかの解が桁落ちする可能性がある．例えば，b=300000にして計算してみるとわかる

２次方程式における解計算のコツ

$ax^2 + bx + c = 0$ の２解の公式は,

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

であるが，$|4ac| << b^2$ のとき，$|b| - \sqrt{b^2 - 4ac}$ が０に近いので桁落ちする可能性がある

このため，$b>0$の時は x_1の分子が，$b<0$の時は x_2の分子が桁落ちする

そこで，例えば $b>0$の時は，まず上の公式を使って x_2 を計算してから

$x_1 x_2 = \frac{c}{a}$ の関係を使って $x_1 = \frac{c}{ax_2}$ で計算した方が精度が良い

第１部　終了

付録：知っておくと便利な文法１

1A. Fortran には複素数型が用意されている

3.複素数型

★ ２個の実数型を一つの数値「複素数型数」として扱うことができ，四則演算や関数計算ができる

★ 複素数定数は「（実部の実数，虚部の実数）」という形式

◎ `(0.0,1.0)`，`(1e-5,-5.2e3)`，`(-3200.,0.005)`　など

これらは，　i，　$10^{-5}-5.2\times10^{3}i$，　$-3200+0.005i$ を意味する

★ 変数宣言はcomplexで行う

```
complex cx, cy　など
```

★ 実部や虚部をxとかyのような変数や数式で表された複素数を作りたいときには，関数cmplxを使う（名前に注意！）

```
cx=cmplx(x,y)
cy=cmplx(x**3-1, 2*y)　など
```

複素数型が混在する計算をするときの型変換

- 整数型と複素数型の計算　→　複素数型
- 実数型と複素数型の計算　→　複素数型
- 代入する場合には右辺の数値を左辺の数値型に変換して代入する
  - ☆ 実数型変数＝複素数型　は右辺の実部を代入
  - ☆ 複素数型変数＝実数型　は複素数型の実部に代入して虚部は０

使用例

```
complex ci, cx, cy
real a, b
a  = 2.0
b  = 0.125
ci = (0.0,1.0)        !  i
cx = a + b*ci
cy = cmplx(a, b)      !  cy と cx は同じ複素数
a  = cx               !  実部が代入される
b  = imag(cx)         !  虚部を取り出す組み込み関数
```

1B. 様々な組み込み関数

型変換関数や絶対値など

組み込み関数	名　称	引数の数値型	関数値の数値型	関数の意味
real(n)	実数化	整数	実数	実数型に変換
abs(n)	絶対値	整数	整数	n の絶対値
mod(m,n)	剰余	2 個の整数	整数	m を n で割った余り
int(x)	整数化	実数	整数	整数型に変換（切り捨て）
nint(x)	整数化	実数	整数	整数型に変換（四捨五入）
sign(x,s)	符号の変更	実数	実数	$s \geqq 0$ なら $\lvert x \rvert$，さもなくば $-\lvert x \rvert$
abs(x)	絶対値	実数または複素数	実数	x の絶対値
mod(x,y)	剰余	2 個の実数	実数	x を y で割った余り
real(z)	複素数の実部	複素数	実数	z の実部
imag(z)	複素数の虚部	複素数	実数	z の虚部
cmplx(x,y)	複素数化	2 個の実数	複素数	$x+iy$
conjg(z)	共役複素数	複素数	複素数	z の共役複素数

赤い部分は整数型数値用関数，黄色い部分は実数または複素数型数値用関数

例えば，int(x+0.5)は四捨五入になる（ただし x>0 のとき）

1C. コメント文，継続行，複文

1．！ 以下の記述は無視される（コメント）

```
!   area of circles
    s = pi*r**2              !   円の面積
    v = 4*pi*r**3/3          !   球の体積
```

2．& を使って複数行で１つの文を記述できる（継続行）

```
    print *,alpha,beta,gamma &
                  ,delta,epsilon &
                  ,zeta,eta,iota
```

は，次の１行と同じ

```
    print *,alpha,beta,gamma,delta,epsilon,zeta,eta,iota
```

3．；を使って複数の文を１行で書ける（複文）

```
    x = 1; y = 2; z = 3
```

ただし，やたらに１行で書かない方が良い

1D. キーボードからのデータ入力

- 計算の途中で外部から変数に数値を代入するのが「入力」
- 入力は read 文で行う

キーボード入力用のread文

```
read *, データ1, データ2, ...
```

★ この文を実行すると，動作がストップして入力待ちになる

★ ストップした状態でキーボードから数値を入力すると実行を再開する

★ 要求する数値を全て入力するまで，次の動作に移らない

★ 入力を促す文字列を直前に出しておくと良い

```
print *,'Input X and Y :'
read *,x,y
```