数値計算法

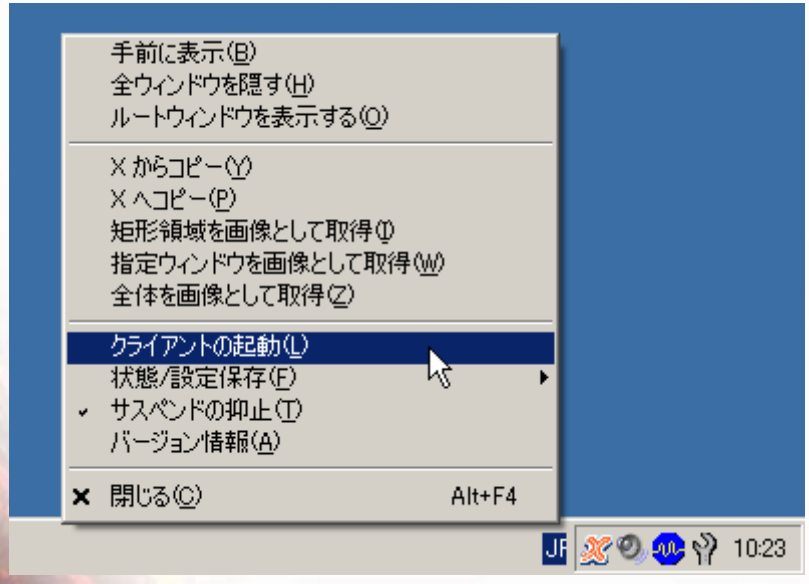
第二回
UNIXシステムとプログラミング言語

若狭　智嗣

粒子物理学講座

Xでの接続方法（ASTEC-Xの使い方）

- Windowsで、
  スタート → プログラム → ASTEC-X → ASTEC-X
  として、ASTEC-Xを起動する
- タスクトレイ内のASTEC-Xのアイコン（X文字）上で右クリックして、「クライアントの起動」を選択
- クライアントの起動ウインドウで、ユーザー名とパスワードを入力して、OKボタンを押す



エディタ(mule)を使ってみよう

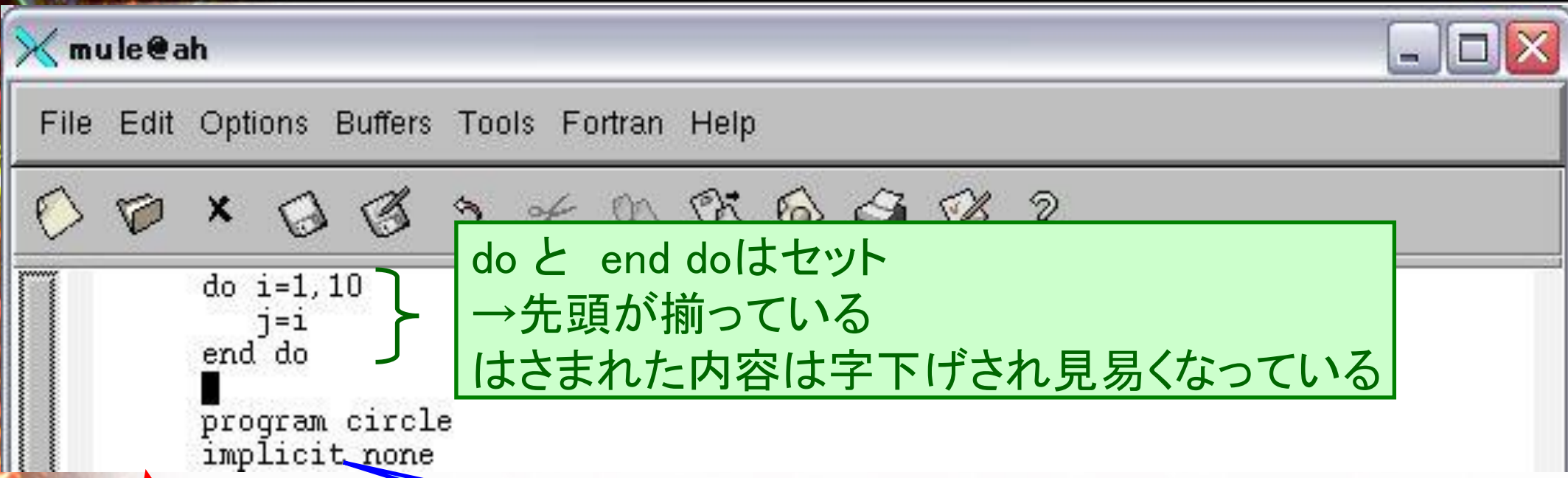
- Windows上でのメモ帳やワードパッドに相当するエディタがmule
- FORTRANのプログラムをコピーして、muleで開く

```
% pwd
/home/06nen/sc106000
% mkdir No2
% cd No2
% cp /home/teacher/z6wt01in/SAMPLE/circle.f .
% mule circle.f &
```

エディタ(mule)を使ってみよう
ータブ(tab)キーの動作に慣れるー

- カーソルを左上（program circleの行）に移動
- リターン・キーを押す
  - １行空行が入り、program circleの行は2行目へ移動
- カーソルを左上に移動
- タブ・キーを押す
  - カーソルが７文字目に移動
  - FORTRANは、７文字目－７２文字目の範囲に命令を書く約束
- ７文字目から以下の内容を入力
  do i=1,10
- リターン・キーを押すと２行目の先頭に移動
- ここでタブ・キーを押すと、１０文字目に移動
- １０文字目から以下の内容を入力
  j=i
- リターン・キーを押すと３行目の先頭に移動
- ここでタブ・キーを押すと、１０文字目に移動
- １０文字目から以下の内容を入力
  end do
- ここでタブ・キーを押すと、end doが左に３文字分移動
- リターン・キーを押すと４行目の先頭に移動
- ここでタブ・キーを押すと、７文字目に移動

エディタ(mule)を使ってみよう(続き)



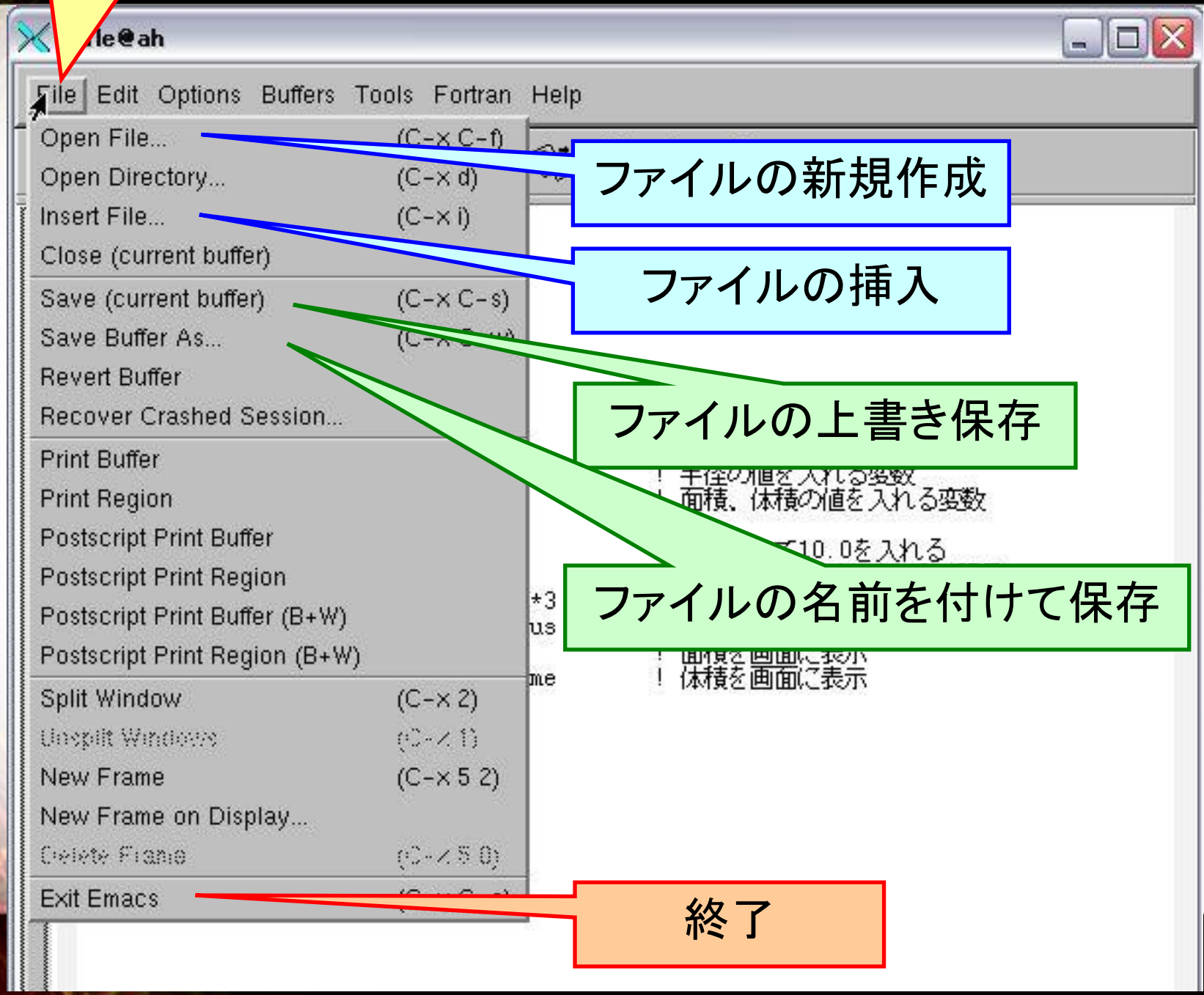
do と end doはセット
→先頭が揃っている
はさまれた内容は字下げされ見易くなっている

6文字分の空白
FORTRANの約束

命令は7文字目～72文字目まで
FORTRANの約束

- キーボードから文字・数字・記号を入力し、またdeleteキー（Backspaceキー）を用いて文字を消去し、入力・消去に慣れる。

ファイルの保存と終了

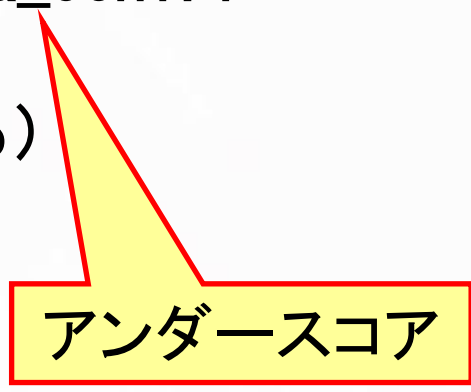

Fileを左クリック
File Edit Options Buffers Tools Fortran Help
Open File... (C-x C-f)
Open Directory... (C-x d)
Insert File... (C-x i)
Close (current buffer)
Save (current buffer) (C-x C-s)
Save Buffer As...
Revert Buffer
Recover Crashed Session...
Print Buffer
Print Region
Postscript Print Buffer
Postscript Print Region
Postscript Print Buffer (B+W)
Postscript Print Region (B+W)
Split Window (C-x 2)
New Frame (C-x 5 2)
New Frame on Display...
Exit Emacs
ファイルの新規作成
ファイルの挿入
ファイルの上書き保存
ファイルの名前を付けて保存
終了

練習

- circle.fを適当に編集
- save（ファイルの上書き保存)コマンドで保存
- lessコマンドで、circle.fの内容を確認
  - 編集した内容が反映しているか確認
  - lessコマンドはqを押すと終了
- circle.fを更に編集
- save as(ファイルの名前を付けて保存)コマンドで保存
  - Muleの画面下に、
    Write file: ~/No2/
    と表示されるので、~/No2/の後ろに適当なファイル名を入力
  - ファイル名は、例えばcircle2.f
- lsコマンドで、circle2.fが出来ている事を確認
- lessコマンドで、circle2.fの内容を確認
- Exit Emacs(終了)コマンドで終了

FORTRANのプログラムを読んでみる

- ファイル、
  /home/teacher/z6wt01in/SAMPLE/keta_ochi.f
  を各人のディレクトリにコピー
  （必要なら、ディレクトリを適当に作成する）
- Muleを使って、keta_ochi.fを開く



keta_ochi.fの説明

左は6文字空いている

mule@ab

File Edit tions Buffers Tools Fortran Help

```
      program keta_ochi
      implicit none
c
      real   a,b,x,y
c
      a = 1.0e4
      b = 1.0e-4
      x = a+b
      y = a-b
      write(*,*) x-y
      stop
      end
```

1行目はプログラム名(任意)を宣言

全ての変数は定義する事を宣言

変数　a, b, x, yを実数(real)型で定義

aに1.0×10^{4}を
bに1.0×10^{-4}を代入(=)
$\times10^{n}$はen(またはEn)と表す

1文字目がcの場合、その行はコメントとみなす

keta_ochi.fの説明（続き）

mule@ah

File Edit Options Buffers Tools Fortran Help

```
      program keta_ochi
      implicit none
c
      real   a,b,x,y
c
      a = 1.0e4
      b = 1.0e-4
      x = a+b
      y = a-b
      write(*,*) x-y
      stop
      end
```

xにa+bを
yにa-bを代入

x-yを画面に出力
Write(*,*)はその後のもの（x-y）を
画面に出力するおまじない

プログラム終わり

プログラムの実行を終わる
（実行時はこの後は処理されない）

期待すること

- aとbの値
  - $a = 1.0 \times 10^{4}$
  - $b = 1.0 \times 10^{-4}$
- xとyの値
  - $x = a+b$
  - $y = a-b$
- x-yの値
  - $x-y = 2b = 2.0 \times 10^{-4}$
- したがって、x-yの値として、2.0×10^{-4}が表示されるハズ

コンパイルしてみよう

- FORTRANのプログラムは、人間は読める(理解出来る)が、計算機(CPU)は理解できない
- 計算機が理解できる形式（実行形式という）に変換することを、コンパイルするという
- コンパイルするときのコマンドは
  frt
  で、形式は、
  % frt -o 実行ファイル名 FORTRANプログラムファイル名
  である。今の場合、
  % frt -o keta_ochi keta_ochi.f
  とすると、実行ファイル keta_ochi ができる
  （lsで確かめると、keta_ochi*があるはず）

実行形式を表す

- 一般的約束
  - FORTRANプログラムは、拡張子として.fを付ける
  - 実行形式には、拡張子は付けない

実行してみよう

- 実行ファイルを実行するには、**単に実行ファイル名を入力すればよい**

  % keta_ochi（リターンキーを押す）

- **何が表示されるだろうか？**
- aとbの値を以下のように変更して、実行してみよ。
  - $a = 1.0 \times 10^{3}$、$b = 1.0 \times 10^{-3}$
  - $a = 1.0 \times 10^{2}$、$b = 1.0 \times 10^{-2}$
  - $a = 1.0 \times 10^{1}$、$b = 1.0 \times 10^{-1}$
  - $a = 1.0 \times 10^{0}$、$b = 1.0 \times 10^{-0}$

精度と誤差

- **実数**
  - 整数1は実数では1.000…と0が無限に続く数
  - 一般に実数を正確に表すには無限桁必要
- **計算機**
  - CPUの演算部分もメモリも有限
  - （本来無限桁必要な）実数は近似的にしか扱えない
  - 結果が常に誤差を含んでいる
    （ある限られた精度しか持ち合わせない）
- **誤差とエラー（間違い）は異なる**

誤差の例―丸め誤差―

- 計算機は数値を電気信号のオン･オフの組み合わせ、つまり2進数で記憶
- FORTRANの場合、
  integer または integer*4で定義される整数変数
  real または real*4で定義される実数変数
  には32ビット（4バイト）が割り当てられている。
- 実数xの場合、

$$x = \pm M \times 10^{e}$$

と仮数部M、指数部e、全体の符号の3つに分解し

- 仮数部M　　24ビット
- 指数部e　　7ビット
- 全体の符号　　1ビット

が割り当てられている。

4バイトの意味

1バイト＝8ビット

32ビット表示の例（12．5の場合）

- **約束事**
  - 指数部はバイアス表示により、

$$e+2^{7-1}=e+2^{6}$$

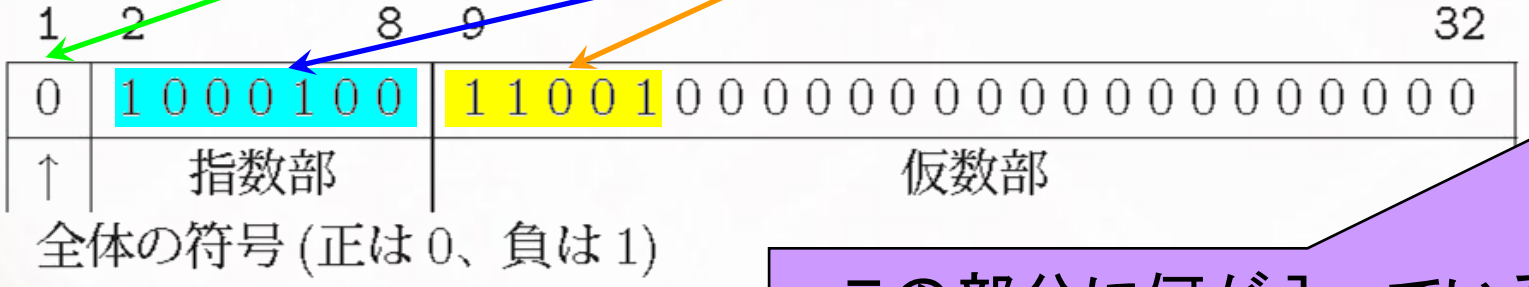
を表示する。

- **12．5の場合**

$$12.5=8+4+0.5=2^{3}+2^{2}+2^{-1}$$

$$=(1100.1)_2=(0.11001)_2\times 2^{4}$$

$$4+2^{6}=2^{2}+2^{6}$$



この部分に何が入っているか
（0か1か）は不定→丸め誤差

桁落ち－なぜ、計算結果が予想と異なったか？－

- aとbの値
  - a = 1.0×10^{4} $(0.10011100010000000000000)_2\times10^{14}$
  - b = 1.0×10^{-4} $(0.110100011001\cdots)_2\times10^{-13}$

27桁違う

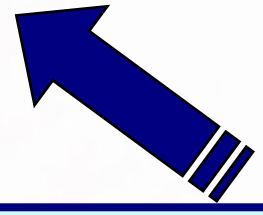
仮数部は24ビットしかない
→xやyには反映しない（桁落ち）

- xとyの値
  - x = a+b → a= $(0.10011100010000000000000)_2\times10^{14}$
  - y = a−b → a= $(0.10011100010000000000000)_2\times10^{14}$
- x−yの値
  - x−y = ~~2b = 2.0×10^{-4}~~ →桁落ちによりx−y=a−a=0

桁落ち

- **桁落ちが問題になる場合**
  - 大きな数に小さな数を足したり引いたりする場合
  - 大きな数から大きな数を引いて、小さな値をだす場合
- **多くは、計算のアルゴリズムで問題を回避できる**
  - 例題

$$\frac{1}{\sqrt{a+b}-\sqrt{a}} \quad \text{数学的に等価} \quad \frac{\sqrt{a+b}+\sqrt{a}}{b}$$



$a \gg b$の場合、大きな数から大きな数を引くことになり、
桁落ち（計算精度）が問題になる

実習

- 例題の式を計算するプログラムを、
  /home/teacher/z6wt01in/SAMPLE/keta_ochi2.f
  として置いた。
  - 各人コピーして実行しなさい。
  - aやbの値を変えて計算しなさい。
  - 正解（筆算での値）と比較しなさい。

数値計算の誤差（丸め誤差や桁落ち）による エラー（間違い）を防ぐには

- **アルゴリズムを工夫する**
  - 桁落ちしにくい式（アルゴリズム）を用いる
- **たった1回の計算で満足しない。**
  - 今までの例でも、aやbの値が異なる場合を計算すると、違い（桁落ち）が顕著に現れた
  - 計算機を用いた数値計算のよいところは、
    いろいろな場合（aやbの異なる計算）を瞬時にできる
    点にあるので、系統的に数値計算を行い、結果の確からしさを検証することが必要
- **倍精度の変数を使う**
  - 今までの例で、
    real → real*8
    とすると、実数を64ビットで表すことになり、仮数部の桁数（ビット数）が増え、桁落ちがしにくくなる。

8バイト割り当てる

real*8 (倍精度)を使う際の注意

- real*8（倍精度）の変数を使うと、桁落ちしにくいプログラムが安易に作成できるが、
  - 倍精度の変数は単精度の場合の倍のメモリを消費（ビット数が倍）
  - 倍精度での計算は、計算の確かさの証明にはならない

  ので、必要に応じて使うこと。
- 実習
  - 今までの
    keta_ochi.f
    keta_ochi2.f
    で、realで宣言されている変数をreal*8で宣言して倍精度にして計算してみよ。
  - aやbの値を変えて実行し、どの程度まで桁落ちが起こらないか確認せよ。

FORTRANのプログラム

- 第5章には以下のような、FORTRANでプログラムを作成する上で必要な事項をまとめてある
  - 扱えるデータの型(integer, real, etc.)
  - 扱える構造(do…end do, if…end if, etc)
  - サブルーチン・関数
  - 組み込み算術関数
  - 配列・複素数・文字列の扱い
- 全てを今すぐ理解する必要はない(必要な時に戻って参照すればよい)
- 以下では、例題のプログラムについて解説する。プログラムは全て
  /home/teacher/z6wt01in/SAMPLE/
  に置いてあるので、各人コピーしてmuleで開いて見るなり、実行するなり、変更するなりすること。

circle.fの解説

左は6文字空いている

1行目はプログラム名を宣言

全ての変数は定義する事を宣言

```
      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
```

1文字目がcの場合、その行はコメントとみなす

文字列はカンマ(,)で括る。表示は、

volume = 体積値

となる

実習

- circle.fをコンパイルし、実行しなさい。
- 半径の値を変えて、実行しなさい。
- 円の周りの長さ、及び球の表面積を計算して表示するようにプログラムを変更して、実行しなさい。

ネイピア数（自然対数の底）を求める

- マクローリン展開

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \quad \Rightarrow \quad e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

- 階乗（factorial = n!）を　Fortran の ループ文で計算
  - n=0の場合
    factorial = 1
  - n≧1の場合
    factorial = 1
    do i=1,n
      factorial = factorial*n
    end do

do から end do までの間を
I を 1 から n まで、
1づつ増やしながら
繰り返しn回実行

do ループという

do ループ終了後は、factorial = n! になる

ネイピア数を計算するプログラム

- mule で新しいファイルを開く
  - % mule napier.f（リターン）
- 以下の内容を入力

```
      program calc_napier
      implicit none
c const:
      integer    LOOP    ! 計算する次数
      parameter(LOOP=10)
c variable:
      integer    i,n,factorial
      real       napier
```

整数を入れる変数を定義

LOOP という変数を定数 10 に定義

実数を入れる変数を定義

変数はカンマで区切る

ネイピア数を計算するプログラムーつづきー

```
c begin:
      napier = 1.0        ! 0次の項の値
      do n=1,LOOP         ! 1次の項以降を計算して加える
         factorial =1
         do i=1,n         ! factorial = n!の計算
            factorial = factorial*i
         end do
c 計算したn次の項を加える
         napier = napier + 1.0/float(factorial)
         write(*,*) n,napier   ! 結果を画面に出力
      end do
      stop
      end
```

factorial=n!

整数型を実数型に変換

$$napier = \sum_{i=0}^{n} \frac{1}{i!}$$

コンパイルして実行してみる

- コンパイル

```
% frt -o napier napier.f
```

- 実行

```
% napier
```

- ネイピア数（e）の値

  e = 2.71828 18284 59045 23536 02874 71352 …

- 問題
  - 計算結果は上記の値と一致するか？
    - 一致しない場合、何桁までは一致するか？
  - 級数の何次の項まで計算すると収束するか？