IRAF講習会
～CL Scriptの基礎と実演～

広島大学宇宙科学センター　川端弘治

日時：２０１１年５月１７日
於：国立天文台天文データセンター

注：本テキスト中に出てくる“”（引用符）などの「向き」は無視して下さい

CL-Script

- IRAFのcl（command language）で動かせるスクリプト言語
- C言語の制御構造に「ある程度」似ている
- １つのファイルに1つのスクリプト（関数）を記述する。
  - C言語のように1つのファイルにいくつも関数を記述できる訳ではない。
- C言語に慣れた人からすると一癖も二癖もあるので注意する
  - えっと思うような些細な事でエラーを吐く
  - エラーの場所がわかりにくい

本講習の流れ

- **cl script の実例を紹介（まず実感してもらう）**
  - **単独スクリプト　lacos**
  - **パッケージ trired**
- **cl script のチュートリアル**
- **cl script を書いて動かしてみる**

clscriptの例

- 宇宙線イベント除去 L.A.Cosmic（P.G. van Dokkum）
  - http://www.astro.yale.edu/dokkum/lacosmic/
  - 撮像画像用と分光画像用とがある
  - アルゴリズム解説は PASP 113, 1420 (2001).
- 今回用いるパッケージ trired
  - TRISPEC　可視偏光分光データリダクションパッケー
  - HBS用に'90年代に作ったhbsredを、TRISPEC/IRAF新バージョン用に改編（川端、秋田谷@広島大、磯貝@京産大）

TRISPEC: 可視赤外線3色同時撮像分光装置（名古屋大Z研で開発）
2006年～ 広島大1.5m望遠鏡に設置　今実習では可視データのみ扱う

ファイル群の解凍

ワークディレクトリを作成し、その中に、サンプルファイル一式を解凍しましょう

```
% mkdir /adc/data/[user ID]
% cd /adc/data/[user ID]
% tar zxvf /data/kawabata/kawabata.tar.gz
% ls
data1       data2       trispec
```

lacos_spec.cl を動かしてみよう

- cd data1
- ds9 &
  ※FOCAS_sample.fits を open し、[scale]>[zscale] で見てみる。
- lacosのスクリプトをホームページからコピーする（…講習ではコピー済みです）
  wget http://www.astro.yale.edu/dokkum/lacosmic/lacos_spec.cl
- xgterm –sb &

以降、立ち上げた xgterm 上で実行する

- mkiraf　※terminal は xgterm で
- cl

> task lacos_spec = ./lacos_spec.cl　※CLスクリプトのタスク登録

> stsdas　※stsdasパッケージのロード（lacosに必要）

> lacos_spec FOCAS_sample.fits FOCAS_sample.cr.fits FOCAS_sample.crmap.fits gain=2.1 readn=4.

以上で、2枚の画像ファイル *.cr.fits, *.crmap.fits が出来るはず

lacos_spec.cl の結果を見てみよう

- 先ほど立ち上げたds9上で、[frame]>[tile]を選択する
- さらに、[new] を **2回** 押す
- メニューバーで Frame > Frame Parameters > Tile > Rows を選択する
- 同じメニューバーで Frame > Tile Frames を選択する

※実は、ds9 -tile row -scale zscale FOCAS_sample*.fits & ならウマー

- 新たに出来た2つのサブウィンドウで *.cr.fits および *.crmap.fits を表示させる（必要に応じて、[scale]>[zscale]で見やすくする）

FOCAS_sample. fits	もとの画像
FOCAS_sample.cr.fits	宇宙線除去後の画像
FOCAS_sample.crmap.fits	宇宙線にIDされたピクセルのマップ

lacosの威力はいかが？

見終わったら、ds9, cl および xgterm は一旦落としましょう

trispecパッケージを動かしてみよう

まず、データファイルを見てみましょう

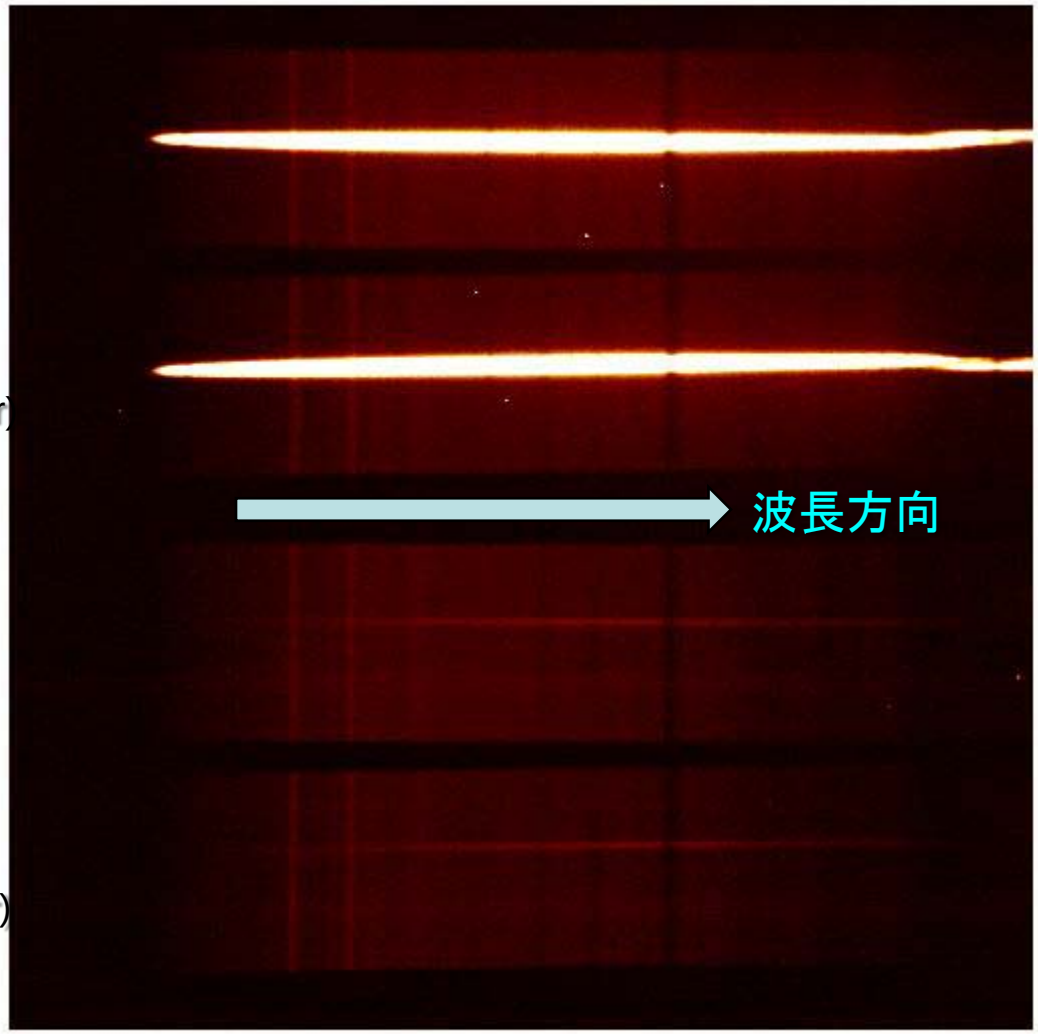
```
% cd /adc/data/[user ID]/data2
% ls
% ds9 &
```

※HD23512_sp_01_hwp000_00_OPT.fits を開いてみましょう

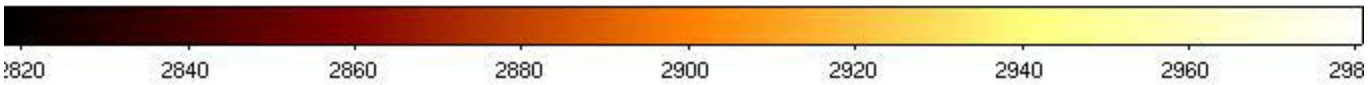
偏光分光モードのデータセットについて

- 2つのスリットのうち、片方に星を載せて撮っている
- 1つのスリットから2つ(常光・異常光)のスペクトル像が生成
- 半波長板(half-wave plate; hwp)の方位角0, 22.5, 45, 67.5度の4フレームで1セットのデータ
- HD 23512, HD25443, HD42807 の3星、およびダーク(bias)

HD23512_sp_01_hwp000_00_OPT.fits



※常光と異常光では偏光ベクトルの向きが直交



スクリプトファイルの構成を見てみましょう1

triredのスクリプト一式を見てみましょう

% cd /adc/data/[ユーザー]/trispec

% ls

- cal キャリブレーションデータのディレクトリ
- cfitsio CFITSIOのソース(ver 1.4)
- clscript CLスクリプト群
- etc いろいろ(bad pixel情報、コンパリソン輝線リストなど)
- prog C言語プログラム(CFITSIO使用)
- standard 標準星データ
- loginuser.cl (IRAF個人設定用ファイル)
- triinit.cl (初期セットアップ用スクリプト)

スクリプトファイルの構成を見てみましょう2

```
% more triinit.cl        ← 初期セットアップスクリプト

% cd clscript     ← cl script一式
% ls
% more tri1d.cl     ← 例えばこんな感じです


% cd ../prog      ← C言語プログラム(FITSファイルの読み書
% ls                 きにはCFITSIOを使用)
% more tri1dsub.c   ← 例えばこんな感じです
```

一通り動かしてみましょう 1

- cfitsio（ver 1.4）のコンパイル
  - % cd /adc/data/[user ID]/trispec/cfitsio

    ※ここに cfitsioのホームページなどから cfitsio142.tar.Z を持ってきてください。例えば wget ftp://heasarc.gsfc.nasa.gov/software/fitsio/c/oldvers/cfitsio142.tar.Z （…講習ではコピー済みです）
  - % tar zxvf cfitsio142.tar.Z
  - % ./configure
  - % make

- unix実行パス・ライブラリパスの設定　　csh, tcshの場合
  - % emacs ~/trispec.cshrc として、以下のように記述

    set path=(/adc/data/[user ID]/trispec/prog $path)

    setenv LD_LIBRARY_PATH /adc/data/[user ID]/trispec/cfitsio:$LD_LIBRARY_PATH
  - % emacs ~/.cshrc を編集し、ファイルの最後に以下のエントリを追加

    source ~/trispec.cshrc
  - % source ~/.cshrc (編集した.cshrcを反映)

一通り動かしてみましょう２

- trired実行用の設定
  - /adc/data/[user ID]/trispec/triinit.cl を編集

    「/home/lecture/kawabata」５か所すべてを、自分の対応するディレクトリ（/adc/data/[user ID]）に置換する

    ※ emacs で開き、置換（[esc]-[%]）を利用すると便利
- IRAF初期設定
  - cd /adc/data/[user ID]/trispec
  - mkiraf

    ターミナルは xgterm を選択

今後、IRAFは /adc/data/[user ID]/trispec で起動することををお忘れなく。

一通り動かしてみましょう３

- xgtermを起動し、その上でiraf (cl) を起動
  - % xgterm -sb &
  - echo $path してみて、/adc/…trispec/prog が含まれていなければ

    % source ~/trispec.cshrc
  - % cl
  - > triinit
    - 各種環境変数設定、Cプログラムのコンパイル
    - 何かに失敗したようなら、logoutして、必要な対処をしてからclを再立ち上げし、triinitを起動し直すこと（cl上でtriinitを二重に起動することは不可）
    - cfitsioのライブラリがうまく参照されない場合は、trispec/prog/trigccをエディタで開いて、gccのオプションを以下のように直す
      - 全ての/home/lecture/kawabataを/adc/data/[User ID]に直す
      - 全てのcfitsio/cfitsioをcfitsioに置換
  - > cd ../data2
  - > ls ← どんなファイルがあるか一応確認
  - > tired ← スクリプトの実行

一通り動かしてみましょう４

- [RET]:Continue.(from 'trimktable') [1]:Customize >>
  には [1] を入力した後、Beginning.. には[1]、End.. には[4]と入力して、1番（trifconv） から4番（tridarksub）まで実行する

- --- OBJECT TRI.OBJ.h23512.1 2S2L 77 NO ---
  Binning mode [RET]:org [1]:const error [2]:sum [3]:mean [4]:def.file >>
  には、[1] > [1] > [0.04] > [Enterのみ] > [h23512] > [2.5] > [1] と入力（h23512はオブジェクト名）

- 他の2つのオブジェクトも同じく入力（オブジェクト名だけ、上記の青字の部分の通りに変えて入力しておく）

ここでは、各オブジェクトのグループに対し、結果をどのようにまとめるか（平均化とσ-リジェクション、波長ビンニング、結果ファイル名）を指定しています

一通り動かしてみましょう５

trired で実行されるマクロコマンド

0. “triinit” 各種初期設定（task登録、ディレクトリ、環境変数設定、Cプログラムコンパイル等）
1. “trifconv“ リストファイルの情報に従いFITSファイルのヘッダーへキーワードを追加
2. “trigroup“ 同じモード（オブジェクト、積分時間等）のフレームをデータタイプ毎にグループ化
3. “tridarkave“ 各ダーク・グループの画像を平均化してダークフレームを生成
4. “tridarksub“ 各グループに該当するダークフレームを差し引く
5. “tritrim“ 画像周縁部の不要な箇所をトリミング
6. “tribadpix“ バッドピクセル・バッドコラム補正
7. “tripixsens“ ピクセル感度ムラ補正（第一フラット）
8. “triextract“ 波長補正 兼 画像歪み補正、及び常光、異常光それぞれの二次元スペクトルの切り出し
9. “triskysub“ 背景スカイ成分の差し引き
10. “triarrange“ 波長合わせ（ガイドエラーや望遠鏡のたわみによる波長方向の微少なズレの補正）
11. “triflat2“ 光学系起源の視野内効率ムラ補正（第二フラット）
12. “tri1d“ スペクトルの一次元化
13. “tribin“ 波長方向のビンニング
14. “triiqugroup“ 各グループ内で、偏光を計算するためのペア・小グループを自動決定
15. “triiqumake“ 各グループ内の各ペア・小グループごとに偏光パラメータを計算
16. “triiquave“ 各グループ内の平均化した偏光パラメータを導出
17. “trimktable“ 器械偏光等の補正を行った偏光データをテキストファイルへ出力

番外. “triparam” 各グループのリダクション・パラメータ（出力ファイル名、ビンニングの仕方、波長合わせ方法等）を選択

一通り動かしてみましょう 6

ここまでで、各オブジェクトフレームからダークが差し引かれた画像（*.d.fits）が生成されています

- `> imstat *.fits`

続き（5番から8番まで）を実行しましょう

- `trired`
  - `[1] > [5] > [8]` の順に入力

8番（triextract）では、元画像（*.px.fits）に写っていた4つのスペクトル像が、それぞれ矩形化（歪み補正等）されたあと、別々のファイル（*.px.1.fits, *.px.2.fits等）に切り出されます。

※`ds9 で HD23512_sp_01_hwp000_00_OPT.px.fits および HD23512_sp_01_hwp000_00_OPT.px.1.fits`表示してみましょう

一通り動かしてみましょう 7

最後まで（9番から17番まで）を実行しましょう

- `tired`
  - `[1] > [9] > [17]` の順に入力

10番（triarrange）では textronixターミナルに確認用スペクトル画像が表示されます（が自動的に進んでいきます）

最後の17番（trimktable）では、どの座標系で偏光を計算するか訊かれます。[1] > [n] の順に入力してください

結果のテーブルに書かれているもの

*0.xy　偏光スペクトルのデータ（ビンニングしたもの）

1 effective wavelength (angstrom)
2 I (ADU)
3 q (=Q/I) (%)
4 error of q
5 u (=U/I) (%)
6 error of q
7 P (%)
8 error of P
9 position angle (degree)
10 error of position angle
11 N frames used for this entry
12 center wavelength (angstrom)
13 physical wavelength width (angstrom)
14 effective wavelength width (= I(ADU) / (count at center wavelength))

頭の数行は、バンド観測との比較の便宜のため、バンドフィルターの透過曲線を使って重み付け平均をしたもの（疑似的なバンドデータ）を計算して載せている

*1.xy　フラックスのデータ（ビンニング無しの結果）

1 effective wavelength
2 I (ADU)
3 standard deviation of I (ADU)
4 I (erg/cm^2/sec/angstrom)
5 standard deviation of I (erg/cm^2/sec/angstrom)

- **TRI.log　tried実行全般のログ**
- TRI.darksub.log ダーク差し引き関連のログ
- TRI.xsft.log 各フレームの波長合わせ（Xシフト）量のログ
- TRI.ysft.log スペクトル一次元化に使われたY幅のログ

- triredの詳説は、hbsred (HBSリダクションソフトウェア）の解説書を参考願います。

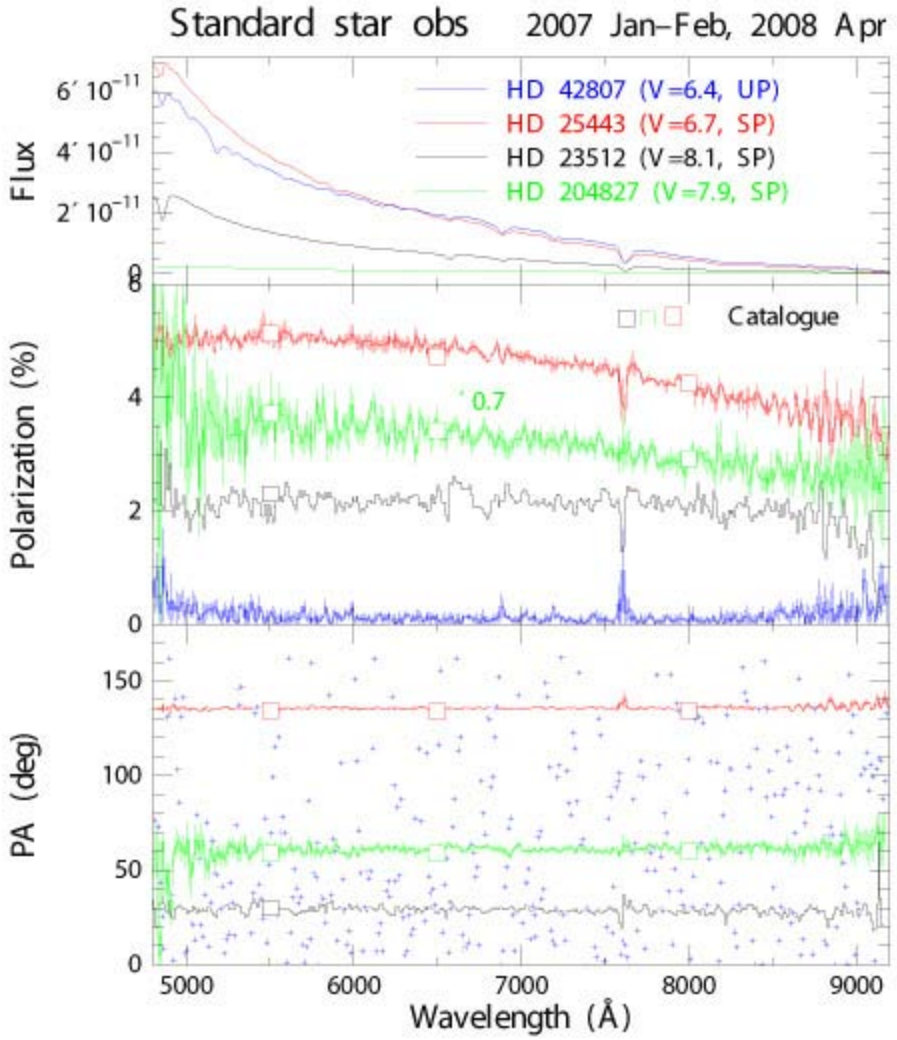
  http://1601-031.a.hiroshima-u.ac.jp/hbs/manual/redman.pdf
  http://1601-031.a.hiroshima-u.ac.jp/hbs/

結果を図示した例（別途グラフツールで作成）

フラックス

偏光度

偏光方位角



X： *1.xy の1カラム
Y： *1.xy の3カラム

X： *0.xy の1カラム
Y： *0.xy の7カラム

X： *0.xy の1カラム
Y： *0.xy の9カラム

clスクリプトを使い出すにあたり…

- **文法が融通利かない点多し**
  - 制御構造(for, while等)の中括弧の書き方とか、最終行の改行の有り無しでエラーが出るとか
  - 動いているサンプルスクリプトと見比べて書き方がどう違うかを調べるのが早い
- **エラーメッセージのエラー行はスクリプトの行数を表していない**
  - デバグは随所に変数を表示するタスクを入れることで原因を追い込む方針が良い

Cl script の構造　1

- % cd /adc/data/[user ID]/trispec
- 初期設定スクリプト triinit.cl を見てみましょう
  - 「#」その行のそれ以降はコメント文
  - set _caldir = “home$cal/200610spol_OPT/” ←IRAF環境変数 _caldir の設定
  - task $trigcc = “$foreign” ← unixシェルで動くコマンド（trigcc）をcl上で動くように
  - task $triflat2 = _clscript$triflat2.cl ←clスクリプトの登録（引数が無い場合）
  - task triscy = _clscript$triscy.cl　←clスクリプトの登録（引数を必要とする場合）
  - keep ←メモリ状態（IRAF環境変数、パッケージ情報）をそのスクリプトが終了しても保持させる
- cd clscript
- 歪み補正スクリプト triscy.cl を見てみましょう
  - procedure triscy(infile, ref_dir, …) ← タスク名と引数順指定
  - file infile { prompt = “Directry…” } 各引数の変数型と入力促進コメントを指定
  - begin ← その次の行からプリアンブル、スクリプト本体の順に記述
  - end ← スクリプト本体が終了（endの後に忘れず改行コードを入れること）

procedureとそれに続く変数指定、begin, スクリプト本体, end がclスクリプトの基本的構成
引数が無いタスクの場合は、変数指定が不要（tridarksub.cl を見てみましょう）

Cl script の構造　2

- つまり、cl script の基本形は

```
procedure aho( infile, param, outfile )
file infile { prompt = “Input image” }
int param { prompt = “Interporation order” }
file outfile { prompt = “Output image” }

begin
  string infn, outfn
  int interp_order
  real f
  string buf

  infn = infile
  interp_order = param
  outfn = outfile
      :
      :
end
```

←promptは、実行時にその引数が指定されていない場合に、入力を促す際に表示されるコメント（本体スクリプトで現れた順に入力が促される）

beginの直後にプリアンブル（変数設定等）を記述　変数設定はプリアンブルのみ可

← 本体用変数へprocedureの引数の順に代入
←　〃
←　〃
（本体スクリプトをここに記述）

※procedureの宣言で使われた引数変数は実行時にその引数が指定されていないと、本体スクリプト中でその引数変数が現れる度に入力を促されるので、一端、本体スクリプト用の変数に代入すると良い

といった感じ（endの後には必ず改行を入れる）

CI script のプリアンブル・変数型

- 変数の設定はプリアンブル部のみで可能（C言語:C99以前と同様）
- 変数の型
  - string　文字列
  - file　　文字列（stringとの違いは配列が不可な点のみ?）
  - int　　整数
  - real　　実数（倍精度らしい）
  - boolean　1/0 (+/-)
  - string buf = “initial value”　←初期設定値を指定する場合（1行に1つまで）
- 配列は1次元のみ
  - real a[100]　←　real a[100][10] は不可。file型は配列不可

※stringの配列形式を直接、タスクで使えない場合がある

その場合、いったん buf = fname[5] などと配列形式じゃない変数へ入れてから与える

※スクリプト本体中で文字列（数値）から数値への型変換は可能

- a = real(buf)　など　　※数値の文字化は str() を利用（後述）

cl scriptの文法

- 各タスクは1行に一つ記述
- 1タスクが長くなる場合は、行最後に「¥」を書いて、次行に折り返すことができる
- cl上でインタラクティブに入力する場合とおおよそ同じように記述していけば良いが、大きく異なる点として
  - 引数（パラメータ）は括弧の中にカンマで区切って記述

Cl script の制御構造

- 繰り返し(forループ)

```
for( i=1; i<n_list; i+=1 ) ← i=1からn_list-1 まで、1づつ足して繰り返し
 a[i] = a_init + i * 10.
```

※注 i+=1, i-=10は使えるが、i++は使えない

- 比較(if文)

```
if ( access( “tri.tmp.time” ) == 1 ) ← 括弧内が真の場合は以下を実施
{                                    ← 中括弧で範囲を区切る場合はこのようにifなどとは別の行で
 delete( “tri.tmp.time”, ver- )
}
```

access関数: ファイルが存在する場合は真(=1)

- ほか、while(括弧内が真の間はずっと繰り返す), switch など
- 比較の文法 「==」一致、「!=」非一致、「<」「>」「<=」「>=」不等号 (文字列OK)
- A && B はAND「AもBも真なら」、A || B はOR「AかBのいずれかが真なら」

Cl script 中での IRAFタスク実行

- 各タスクは1行に一つ記述
- 1行が長くなる場合は、行最後に「¥」を書いて折り返すことができる（但し制限有）
- cl上でインタラクティブに入力する場合と、大きく異なる点として、引数（パラメータ）は、括弧の中に、パラメータごとカンマで区切って記述する点がある。つまり、

- ```
  imarith( fn1, “-”, fn2, fnout, pixtype=“real”, ¥
        calctype=“real”, ver- )
  ```
  のように、タスク名の後のパラメータは括弧で閉じ、パラメータ間はカンマ区切り。
  （1行が長くなる場合、「¥」の後に改行を入れてもOK。ただ1行＝1タスクにつき上限は250-300文字くらいまで）

比較： cl上でこれと同等のタスクを入力する場合は

```
imarith in1.fits – in2.fits out.fits pixtype=“real”
calctype=“real” ver- のように改行せずに続けて入力
```

- unixシェルのタスクを（登録せず）使う場合は、頭に「！」を付けてunixシェル上で実行する形式そのまま記述　`!cp aho1.fits aho2.fits`
```

テキストファイルの入力

- テキストファイルからの読み込み

```
n_list = 0
list = “TRI.list”  ←ファイルのオープン(ポインタlistは固定)
while( fscan( list, buf ) != EOF )
{       ※fscanの返し値は、読み込めた変数の個数（かEOF）
  n_list += 1
  fn[n_list] = buf
}
```

```
% cat TRI.list
ngc1000_0_OPT.fits
ngc1001_0_OPT.fits
     :
```

注：ファイルハンドラは、ひとつ(list)しか使えない。

つまり、同時に1つしかファイルを開くことができない。2つ以上のファイルから読む場合は1つ目のファイルを開いて必要な情報を変数（配列変数）へ入れるなどしてから、次のファイルを開く

ファイルをクローズする命令は無い。つまりlist=…が宣言されるたび、ファイルポインタが新しく読み込むファイルの先頭へと移動する

cl-scriptを書いてみよう1

あるテキストファイル（TRI.list）に画像ファイルのリストを与えて、画像一つずつヘッダーからエアマス値を読み出して表示し、且つ統計値（median=中央値）を表示するスクリプトを作ってみましょう

```
cl> cd home$testdata
cl> !xemacs clsample1.cl &     ※以下のように記述して下さい（「←」の後はコメント）

procedure clsample1    ←プロシージャ名はファイル名（拡張個）と一緒
begin
  string inlist = “TRI.list”
  file fn1
  real am

  list = inlist     ←ファイル（string変数inlistに書かれている名前）のオープン
  while( fscan( list, fn1 ) != EOF )    ←ファイル終了まで
  {
    imgets( fn1, ‘AIRMASS’ )  ←ヘッダーキーワード（AIRMASS）の値を読み込む
    am = real( imgets.value ) ←文字列型を、実数型へ変換
    printf( “Image: %s  AIRMASS %5.3f  Median “, fn1, am ) ←結果表示1
    imstatistics( fn1, field=“midpt”, format- )             ←結果表示2
  }
end
```

cl-scriptを実行してみよう1

タスク登録

```
cl> task $clsample1 = ./clsample1.cl
```

タスク実行

```
cl> files *OPT.fits > TRI.list
cl> clsample1
```

どう表示されたか？（スクリプト中の命令との対応が判るか？）
正しく処理されたか？

うまく行けば以下のように表示されるはず

```
ecl> clsample1
Image: HD23512_sp_01_hwp000_00_OPT.fits  AIRMASS 1.000152  Median 2835.355
Image: HD23512_sp_01_hwp225_00_OPT.fits  AIRMASS 1.000152  Median 2837.205
  :
```

cl script 中の演算・文字列操作

- 数値については、+, -, *, / の四則演算、%(余り)、通常の数学関数（sin, cos, tan, log, log10, asin, acos, atan, atan2などなど）が使える。int と realが混じっている場合、realで計算されることもC言語と一緒
- 文字列操作
  - 文字列も「=」で代入できる
  - 単純な結合は「//」

```
例1: fn1 = buf[i]//".fits"
例2: imcopy( fname//"["//x1//":"//x2//","//y1//":"//y2//"]",¥
             outname, ver- )
     cf. imcopy test1.fits[2:200,3:400] test2.fits（XY領域指定）
```

  - 文字列関数(cl上で help stringsと打てばヘルプが表示)
    - buf = str(a) 数値を文字列化
    - a = strlen(buf) 文字列の長さを返す
    - buf = substr(buf1, strlen(buf1)-3, strlen(buf1)) buf1の一部を切り出し（この例ではbuf1の末尾の4文字をbufへ代入）

テキストファイル（や変数）への出力

- clへ標準出力するコマンドの最後の部分へ、unixシェルのリダイレクト形式で指定
  - 例

```
printf( “Angle %-07.2f”, angle, >> “angle.log” )
imstat( tmp_fn1, fields=“midpt”, format-, > tmp_fn2 )
```

※unixシェルと同様に「>」は新規上書き、「>>」は追記

※「>」を使う場合、そのファイルが存在するとエラーが出る（場合がある）ので
access() を利用して存在する場合はあらかじめ削除しておく

- パイプを使って変数へ直接入力することも可能
  - 例

```
real mflux
imstat( tmp_fn1, fields=“midpt”, format- ) | scan( mflux )
```

Cl-script で有用な命令・タスク1

- bye　スクリプトを途中終了する
- next　繰り返しの制御構造(for, while等)で次ターンとしてループ先頭へ
  C言語の continue; と同等
- goto　指定したラベルの行へジャンプ

```
if( n < 1 )
  goto nextpoint
nextpoint: (←ラベルはこのようにコロンを付けて指定)
```

- envget　IRAF環境変数の中身を返す

```
例: buf = envget( “home” )
```

- imgets　画像ヘッダーのキーワードを(文字列形式で)読む

```
例: imgets( buf//”od.e.fits”, “POS-HWP” )
    theta = real( imgets.value ) ← 実数化して変数へ代入
```

- sections ある文字列を含んだファイル名の一覧を作成

```
例: sections( “upave*.fits”, option=“fullname”, >> tmp_list )
    if( sections.nimage == 0 )
    {
      printf( “Cannot find appropriate files¥n” )
      bye
    }
```

Cl-script で有用な命令・タスク2

- access　ファイルが存在するかどうか判定（存在すれば真=1）

  例 if(access(outfn[i]//".fits") == 1)

- printf　フォーマット付き表示の構文はC言語と殆ど同じ

  printf("%s %10e %-05.2f¥n", buf, a, b, >> logfn) とか

- scanf　コンソールからの入力

  例: c = scan(buf)
  while(buf != "y" && buf != "n")
  c = scan(buf)
  if(buf == "y")

- hedit　ヘッダー情報の編集

  例: hedit(fn, "HISTORY", "Dark ("//dark_fn//") subtracted", add+,¥
  delete-, verify-)

- mktemp　テンポラリファイルのファイル名候補を作成

  例: tmp_fn = mktemp("tri.tmp.")//".fits"

cl-scriptを書いてみよう2

```
cl> cd home$testdata
cl> !emacs clsample2.cl &   へ以下のように記述

procedure clsample2
begin
  string inlist = “TRI.list”
  string fnhead[100]  ← 100個の配列を持った文字列変数を定義
  int n_list  ← リストファイル中に書かれたエントリ数を扱う変数
  file buf, fn1, fn2, tmpfn
  int x1=1
  int x2=30

  task $sed = “$foreign”
  tmpfn = mktemp( “tri.tmp.” )

  sed( ‘s/¥.fits//’, inlist, > tmpfn )  ←sed(ストリームエディタ)を使ってリストファイル中の
                                          「.fits」を「」に置換(=削除)したファイルを生成
  n_list = 0
  list = tmpfn      ←生成したファイルをオープン
  while( fscan( list, buf ) != EOF )
  {
    n_list += 1
```

リストファイルからファイル名を一部変換した状態で読み出し、配列変数に保管しておいて、順次それを基に入力ファイル名と出力ファイル名を決めて処理を行う。ここでは1ピクセル列ごとに関数をフィットして差し引くタスクbackgroundを用いて背景成分を差し引く。

```
    fnhead[n_list] = buf    ← ファイル中のエントリを一個ずつ、配列変数へ入れていく
  }
  delete( tmpfn, ver- )
  printf( “%d files registered.¥n”, n_list )
  for( i=1; i<=n_list; i+=1 )  ← 繰り返しの制御構造
  {
    fn1 = fnhead[i]//”.fits”      ←入力ファイル名を定義
    fn2 = fnhead[i]//”.os.fits”  ←出力ファイル名を定義
    printf( “%s(“, fn1 )
    if( access( fn1 ) == 0 )  ←入力ファイルが見つからない場合
    {
      printf( “Not found. Skipped.¥n) “ )
      next
    }
    if( access( fn2 ) == 1 )   ←出力ファイルが既に存在する場合
    {
      printf( “Overwrite…” )
      imdelete( fn2, ver- )
    }
    background( fn1, fn2, axis=1, sample=x1//”:”//x2, naverage=1,¥
                function=“legendre”, order=1, low_reject=1.5,¥
                high_reject=1.5, niterate=3, interac- ) ←背景成分の差し引き
    printf( “Done)¥n” )
  }
  printf( “¥n” )
end
```

処理のメインはこの background

cl-scriptを実行してみよう2

タスク登録

```
cl> task $clsample2 = ./clsample2.cl
```

タスク実行

```
cl> del TRI.list
cl> !echo ahoaho.fits > TRI.list
```

←実在しないファイルでどうなるか見るため

```
cl> files *OPT.fits >> TRI.list
cl> imred
cl> generic
cl> clsample2
```

どう表示されたか？　正しく処理されたか？

もう一回 clsample2 を実行すると表示がどう変わるか？

```
cl> ls
```

オリジナルのファイル（*OPT.fits）と処理済みのファイル（*OPT.os.fits）との違いを、ds9で表示したり、implot、imstatを実行して比べよう

cl-script の応用的利用

- あるフィーチャーをidentifyを使ってマークしてもらい、その位置を database中のid*ファイルをlist=で読み取る

  triredの中の　triarrange.cl で試してみよう
  - triparam を実行し直して、

    最後の　Feature identification で [2]:manual を選択
  - triredのtriarrange(10番)を再実行する
  - フィーチャー種類を選択、textro上でマークする

- スペクトルの波長ズレを fxcor で求めて、その結果をテキストに出力し、それを list= で読み取って、imshiftで揃える

まとめ・最後に

- cl-scriptの基本構造とタスク登録、実行方法を覚えた（引数がある場合と無い場合では少し異なる）
- スクリプトの制御構造を覚えた

- cl-scriptでは変数系が弱い　大量の情報を扱う場合は、ファイルを介した入出力が便利
- 文法に癖があるため、本格的なスクリプトを組む場合はバグだしが大変だが、既に動いているものと比べることで、デバッグの労力が軽減