NAOJ/ADC IDL 講習会資料 (2009 Jan)

# IDL 初〜中級者のための天文データ解析用 IDL講座

大山陽一

ASIAA（台湾）

# 重要なお断り

- IDLは、RSI(->ITT)（とその販売代理店）の商品です。大山は彼らとはいっさい関係なく、金銭その他の利便の提供も受けていません。
- IDLに義理は無いので、IDLの宣伝はしません。
  - 大山は1ユーザーとしてIDLの良さを認識して、皆さんにお勧めしますが、買ってくれなくても結構です。
- IDL のライセンス料が高いという苦情は、私は受け付けませんし、どうにもできません。
  - 最初は ADC のユーザーとして IDL を使用したら良いでしょう。
- 今後の IDL 使用環境については、ADC のスタッフと相談してください。
  - 大山は ADC の経営企画に関与していません。

# なにはなくとも参考書

- Practical IDL programming
  - By L. E. Gumley (MORGAN Kaufmann 発行）
  - 超おすすめ。1教室に1冊常備。¥8103 at Amazon
  - ユーティリティーも気が利いていて、使える。
- IDLHELP (online 版) ‘Idlhelp’ on unix shell
    - 昔はぶ厚い本が大量に計算機室に転がっていたのだが。。。
    - IDL をインストールすれば、自動的についてくる。
  - 検索機能が強力。使える。
  - 最初に見るのは、Getting started with IDL
    - でも、これを見て絶望する人がいるかも。
  - 常用するのは、Reference guide
- Coyote’s Guide to IDL programming
  - By David Fanning
  - http://www.dfanning.com/index.html
    - Web 上の tips 集も面白い。
  - 中級以上？

# 講習内容の大枠

- 春の講習会：初級者コースの講習
- 今回の講習会：より実践的な講習
  - 天文解析アプリへの応用を念頭に。。。
  - IDL 基礎・特徴のおさらい
  - IDL の特徴を活かしたプログラミング
  - デバッグの基礎
  - サブルーチン化
  - 実習

# IDL 言語の特徴

- IDL の基礎の基礎の復習です。
- IDL の特徴が活かせるアプリケーションとは？
- キーワード
  - 「超」高級言語
  - Interactive Data Language
  - Data Visualization Tools

# IDL 言語の特徴1

- インタプリタ
  - 柔軟なプログラム開発環境。
  - ただし速度は遅い。
- アレイ言語
  - IDL に読み込んでしまえば、
    - FITS 画像も jpeg もアレイ。
    - FITS table も ascii table もアレイ。
  - アレイを使うと、
    - プログラムが分かりやすい、見やすい。
    - やりたい処理をそのまま書ける。
  - アレイのサブスクリプト（インデックス）を処理する、という概念を理解すべし。
- 強力なデータ処理言語＋データ表示
  - C -> file output on disk -> gnuplot では非効率。

# IDL 言語の特徴2

- Procedure/Function は ascii file で配布。
  - コンパイル済みプログラム or 実行形式 (.exe) は、基本的に存在しない。
  - 他人のプログラムも、全て中身が読める。
    - コード内容は、全公開。ブラックボックス処理は、ない。
    - 他人のプログラムを改良・応用して活かす。
  - 欲しいプログラムは、まずネットで探す。
- すべてはオン・メモリ処理。
  - メモリは可能な限り増設しましょう。
    - C.f. File I/O or disk space for IRAF

# IDL vs. IRAF

- IDL はプログラム言語
- IRAF は解析環境＋貧弱なシェル
- IDL はメモリベース
- IRAF はファイルベース
  - 結果は一旦ファイルに書き込む。
  - 次のルーチンは、ファイルを読む所から始まる。
- IRAF は良･くも悪くもブラックボックス
- IDL は低レベルのルーチンまでハッキング可能
  - 新しい装置の新しい解析手法も作れる。
- IRAF は無償 (NOAO: 天文台)
- IDL は有料 (RSI->ITT: 営利企業)
  - ただし無償のツール･コードが出回っている。

# IDL vs. C (科学技術計算の場合)

- コードの読みやすさ:IDL = 100 X C
  - C はそもそもシステム開発言語。
  - C の”ポインタ”、”*”、“&” などは、忘れましょう。
    - 忘れました。
    - Printf で悩んだ時代が懐かしい。
- デバック速度:IDL = 100 X C
  - インタープリタなので、デバッグはとても楽。
  - 試行錯誤によるコードのアップデート作業も、楽。
- 計算速度:IDL = 1/10 X C
  - IDL に速度を求めてはいけません。
  - 最終手段:高速化するルイーチンだけ C で書いて、それを IDL から呼び出す、という技はある。
    - 自分で使った事は無いが、見たことがある。

## これだけ知っていれば読める！ IDL 文法の特徴

- 初心者講習会に出た方は、スキップしてください。
- でも、少しは発見があるかも。

## 大文字と小文字

- 全く関係無し。
  - もちろん String 以外。
  - 個人の見やすい書き方で。
    - 私は小文字派。

# 0 から数えよ

- 画像の左下のコーナーは、
  - IDL では [0,0]
  - IRAF では [1,1]
  - DS9 や SKYCAT は [1,1]
- For loop を使うときは、
  - 0 から n_elements(x)-1 まで。

# , と $ と & と ; と

- ‘,’: すべての argument の区切り。
  - 一番よく使う。
  - または、array index の区切り。
- ‘$’: 複数の行を1行コマンドとして使う時のおまじない。
  - 基本は、1行1コマンド
- ‘&’: 一行に複数のコマンドを書く時のおまじない。
  - 基本は、1行1コマンド
- ‘;’: コメント行。
  - 一般に、行の頭につける。
  - しかし、行の最後につけても良い。
- ‘:’: array の index の範囲指定
  - Image[1:10,10:20] --- 大きな image array のうちの x=1~10, y=10~20 のサブ・アレイ

# [ ] と ( )

- [ ]: array の座標（インデックス）
  - x[0]: x array の最初の内容
  - y[0,10]: y array の x=0, y=10 の座標の内容
  - Atv,image[*,*,1]: 2次元画像のスタック・キューブ（3次元）から、必要な画像を抜きだす。
  - などなど。
- ( ): 関数 (function) の argument 部。
    - または、数学の計算順序を示すカッコ。
  - Y＝f(x)など。
- 非常に古いバージョンの IDL では、( ) を array の index として使っていたため、古いコードでは x(0,10) などと書いてある場合がある。
  - これでも動くが、お勧めできない。
    - 混乱の元。

# && と AND, || と OR, ~ と NOT

- 集合の論理積などを表すのが、&&, ||, ~
    - 「ベン図」を思い出そう。
  - IF 文で多用。
    - もし A かつ B であれば、これを実行、などなど
- ビット単位の理論積など表すのが、AND, OR, NOT
  - 2ビットの計算例
    - 000 AND 111 = 000
    - 000 OR 111 = 111
  - Where 文で多用。
    - A かつ B の条件を満たすインデックスを求める、などなど
- 経験的に、where 文で誤って && など使ってしまい、気づかずにデバッグで悩むことが多い。
  - 間違っても期待した挙動を示す場合もあり、気づきにくい。
  - where は特別、と割り切って覚えても、多分問題は無い。

# EQ と =, GT と <

- EQ, GT などは、比較演算子。
  - IF 文などで多用。
    - IF A GT B then…
- = は代入文。
  - A=B など。(A に B の内容をコピーする)
- <, > は特殊な演算子。
  - A = (B < 5) などと使う。
    - もし B が 3 なら A=B=3
    - もし B が 10 なら A=3
  - Index の範囲指定などで使うと便利。
    - A_cut=A[b > 0:c < (x_max-1)] など。
      - 仮に b や c が A array の index の範囲外を指定したとしても、エラーを回避できる。

# “ “ と ‘ ‘

- どちらも “string” の範囲指定をするもの。
  - ただし、‘’ は特殊な用途があるので、”” がおすすめ。
    - 例えば、string が数字で始まる場合、問題が発生する。
      - IDL> print,'0a'
      0a
      - IDL> print,"0a"
      print,"0a"
      ^
      % Syntax error.
      - IDL> print,'aa'
      aa
      - IDL> print,"aa"
      aa
    - 昔これで悩んだことがある。

## variable の振るまい1

- あらかじめ定義する必要はない。
  - C の様に、最初に define する必要はない。
  - インタプリタだから。
- 代入される段階で、初めて定義される。
  - Variable のタイプは、代入相手によって決まる。
  - A=1 -> A は integer
  - A=1.0 -> A は float
  - A=fltarr(10) -> A は float の array (size=10)
  - A='1.0' -> A は string
- Variable のタイプは、再設定できる。
  - A=1.0 の後に A='1.0'値と違うタイプに再設定できる。
    - 結果は A は string に変わる。

## variable の振るまい2

- A が「定義されているかどうか」をプログラム上で調べることができる。
  - Undefined を使った IF 条件分岐、など。
  - Procedure/function のオプションの有無の判定など。
- IDL のパラメーター割当はルーズなので、便利な反面、混乱の元。
  - 統一的な命名法や、サブルーチンを使った variable の整理、などがおすすめ。

# 数字（アレイ）の計算

- 基本は、まったく普通の感覚で。
  - +-*/^
  - alog10(), sqrt(), sin(radian)
  - A^2.,10^-3
  - 3*(1+2)=9
- 数字 -> 「数字のアレイ」として使っても、同じに動く
  - 上記で、A はスカラーでも良いし、ベクターでも良いし、イメージ（2次元アレイ）でも良い。
  - ただし、アレイのディメンションは合せておく事。
    - 良くあるエラーの元。

# 特殊な計算

- 何でもできる。
  - でも気をつけて。
  - IDL は落ちずにプログラムは続いて行く。。。
    - Print,1/0
    - 1
    - % Program caused arithmetic error: Integer divide by 0
    - Print,1*!values.f_nan,1*!values.f_infinity
    - % Program caused arithmetic error: Floating illegal operand
  - 文句は言われるが、これらはエラーストップではない。
    - うまく活用しましょう。
  - 逆に、arithmetic error は頻出するので、あまり気にしすぎる必要はない。

# よく使う、数学関数

- Total
- Median,mean,stddev,variance
  - Moment 関数で一発。
- Sin/cos
- Max,Min
- Finite
  - Argument が有効な数字かを調べる関数。
    - 無効な数字：無限大、無限小、Not A Number (NAN)
  - 知っていると意外と便利
    - Mask 処理に応用できる。
      - 1 or 0 mask ではなく、1 or NaN mask とする。

# たまに使う、文字列関数

- Strcmp, strtrim, strlen
  - IDL の科学計算ではあまり必要ないですが、たまに file I/O などで必要な場合があります。
  - あまり強力ではないですが、一通りのことは出来ます。

# Array を使いこなそう

- 1次元：A=[1,2,3] --- 3 要素ベクトル
- 2次元：A=[[1,2,3],[1,2,3]] --- 3X2 array
- 範囲指定：A[1:2], b[1:3,2:5], c[*,2:5], d[2:*]
- アレイインデックスがアレイ
  - A=[1,2,3,4,5] & b=[0,1,2] -> A[b]

- Help で結果を確かめよう。
  - Help,a,a[1:3] など。
- (A+B)[0:2] という技もあり。
  - C=A+B & C[0:2] という意味。

# Where を使いこなそう

- Index 使いになろう
  - For loop で array 内容をスキャンしながら IF 文を掛けるのは、古いです。

```
For x=0,99 do begin
    For y=0,99 do begin
        If image[x,y] LT -100 then image[x,y]=!values.f_nan
    Endfor
Endfor

bad_index=where(image LT -100)
Image[bad_index]=!values.f_nan
```

  - 「ベン図」をイメージしながら、where の中に条件を書き込むだけ。

# 最低限知っておきたい IDL コマンド、関数

- アレーを作る
  - Fltarr, intarr, …
  - Findgen, indgen, …
- アレーを調べる
  - Where
  - Size, n_elements
  - help
- 結果を表示する
  - print
  - Plot (oplot)
  - atv
- Procedure/function のキーワードを調べる
  - Keyword_set
- デバッグ
  - Message
  - .reset

# 最低限知っておきたい IDL 言語の制御構造

- 条件分岐
  - IF 文
  - Case 文
- ループ
  - For 文
  - Break, continue
- ジャンプ
  - Goto 文
- どれも簡単なので、参考書を見てください。

## 知っておきたい外部function/procedure

- ネットの向こうにいる開発者に、感謝。
- IDLASTRO (http://idlastro.gsfc.nasa.gov)
  - 天文向け IDL 外部ルーチン総本家。検索やリンクもある。
  - 各種そろっているが、以下はマスト。
    - FITS read/write; ASCII table read/write
- Others
  - Atv: 2次元画像ビュワー or ds9 for IDL.
    - http://www.physics.uci.edu/~barth/atv/ by Aaron Barth
    - ちょっと機能が足りないが、ないと大変困る。
    - 各方面で改造されて、応用されている。
  - Loadcolors:プロット時の色を定義する。
    - 最初に出てきた参考書に集録
  - Saveimage, pson (psoff)
    - PS/PNG/JPEG dump of plot window
    - 最初に出てきた参考書に集録

## IDL プログラミングを始める前に知ってほしい事柄

- これから本格的に IDL をいじりたい方が、その環境を準備するときに役に立つと思われる、ちょっとした情報。
- 自分の研究室に戻って設定するときに、役立ててください。

# 開発環境

- Idlde
  - IDL 開発会社が作った開発環境 (developing environment)
  - 良くできていて、愛用者もいるが、大山は好きでない。
    - 初期のころは buggy すぎて、使えなかったので。
    - 最近はだいぶマシになったらしい。
  - つい最近、eclipse 環境になりつつあるようだ。
    - ただし、最初のバージョンは buggy すぎて使えないらしい。
- コマンドライン on ターミナル + 汎用エディタ
  - 簡単・簡便・必要十分
  - エディタは Emacs + IDLWAVE 環境がおすすめ。
    - IDLWAVE: http://www.idlwave.org/
      - IDLWAVE is an add-on mode for GNU Emacs and XEmacs which enables feature-rich development and interaction with IDL…
    - 構文の色付け、1行ヘルプ、などなど機能多数。
  - お好きな組み合わせで、どうぞ。

# IDL 起動前の設定

- IDL のインストール: プロに任せる。
- .cshrc; IDL_PATH の設定
  - 自分の IDL ライブラリ path を追加せよ。
  - Source .cshrc を忘れずに。
- 'Idl' or 'idlde' で起動。
- "Window" して、graphic 画面が出れば OK.
  - エラーが出たら、X11 の設定に問題がある。
    - プロに相談せよ。
- IDL を起動中は IDL_PATH を追加できない。
  - 一旦 exit して、.cshrc 編集、source .cshrc の後、再起動せよ。
    - 本当は IDL 起動のままで書き替える方法もある。
- おまけ: IDL_rbuf_size を設定しておくと、便利。
  - コマンドラインのヒストリーを何行分覚えておくか？
  - 本気でデバッグするなら、大きめの方が良い。

## マニュアルはこれだけ<br>オンライン・ヘルプを使うべし

- Unix shell から ‘idlhelp’
- Reference guide -> command reference（IDL文法辞典）だけでよい。
- まずは index か search で探す。
- Example が便利。
- See also… を、たどってみよう。

- 外部 procedure/function については、そのソースコード自身にヘルプが書いてあるのが慣例。
  - コードの先頭部（だけ）を読め。

## IDL本体（基礎）に集中すべし。<br>おまけは当面忘れる。

- No GUI
- No iTOOL
- No Object programming
- Etc…

- GUI だけやむなく使ったことがあるが、他は全く使用経験ゼロ。
  - たまに外部ライブラリだけ使うことがある。

# デバッグの達人への道

- IDL ならデバッグは簡単。その特徴を活かす際の基礎知識集です。

# IDL プログラム実行時の、コンパイルとエラーの種類

- コンパイルエラー
  - インタプリタでも、コンパイルする。
    - コンパイル：最低限の構文チェックなどを指す。
  - サブルーチンは、呼び出されると自動コンパイルされる。
    - したがって、メインプログラムの途中でコンパイルエラーが起き得る。
    - 前もってコンパイルしておくこともできる。
- 実行時エラー
  - コンパイルが通っても、実行時エラーは発生する。
    - 例：Array の割り当て等は、実行時に決まるので。
  - 実行時エラーが起きると、そのままのメモリ状態で IDL シェルに落ちる。
    - インタラクティブモードになる。

# エラーメッセージを読むべし

- 知るべき事
  - エラーの種類
  - エラーが発生したサブルーチン
    - 今自分はどこ（どのルーチン）にいるのかを知る。
    - 大きなプログラムでは、自分の位置を見失いがち。
  - エラーが発生したプログラムファイル
    - 1つの xxx.pro に複数の procedure/function がある場合もあるので、注意。
  - エラーが発生したプログラム行
    - IDL のエラー行表示は、信頼できる。
  - エラーが発生した具体的な箇所
    - '^' で問題の行のどこで止まったかが表示される。
    - ただし、あまり参考にならないこともある。

# Error の例 1

- % Syntax error.
- At: /home/ohyama/xxx.pro, Line 10
- % Compiled module: xxx.
- % Attempt to call undefined procedure/function: 'xxx'.
- % Execution halted at: yyy  61 /home/ohyama/yyy.pro
- %  $MAIN$
- IDL> help,/trace
- % At yyy  61 /home/ohyama/yyy.pro

# Error の例 2

- % String expression required in this context: STRNG.
- % Execution halted at: xxx 185 /home/ohyama/xxx.pro
- % yyy 18 /home/ohyama/yyy.pro
- % zzz 61 /home/ohyama/zzz.pro
- % $MAIN$
- IDL> help,/trace
- % At xxx 185 /home/ohyama/xxx.pro
- % yyy 18 /home/ohyama/yyy.pro
- % zzz 61 /home/ohyama/zzz.pro
- % $MAIN$

# エラー停止時の IDL の状態

- インタラクティブ状態になっている。
- メモリの内容はそのまま保存されている。
  - ただし、今いるサブルーチン内の情報に限る。
  - メモリの内容を
    - 確認できる。
    - 修正できる。
    - 修正後、その場から再スタートできる。
- 意図的にエラーを発生させることができる。
  - デバッグ時に有効。
- エラー停止後、再度最初からやり直し実行する時は、メモリを一旦クリアすること！
  - さもないと、途中のサブルーチンの情報を元に、メインプログラムがイニシャルされる恐れ有り。

# これだけは知っておきたい デバッグ用コマンド

- Print
  - デバッグメッセージの埋め込み
- Message
  - 強制エラー発行による、実行停止。デバッグモードへ移行。
- Help
  - Variable の内容確認
  - Help,/trace で、今いるサブルーチンの確認
- .comp
  - コンパイル（し直し）
- .cont
  - 継続実行
  - エラーからの復帰
- Return or return,0
  - 強制的親ルーチンへの復帰
- .reset
  - メモリ内容をすべてクリアして、IDL 起動時の最初の状態に戻す。
    - ただし、IDL 環境変数はリセとされない、などの例外はある。
  - はじめからやり直す場合は、まず .reset すること。

# エラーストップする90%の理由1

- IDL 設定に関するエラー。
  - プロット用 Window がでない。
    - よくある X11 の設定の問題。管理者に問い合わせ。
  - Color が出ない？おかしい？
    - おまじないコマンド：device, decomposed=0
- Procedure/Function undefined
    - % Attempt to call undefined procedure/function: 'xxx'.
  - 存在しないサブルーチンにアクセスしようとしている。
    - IDL path に該当プログラムファイルが存在しない。
      - Print,!path をしてみよ。きっと path が通っていない。
    - 該当プログラムが、エラーでコンパイルできない。
      - .comp xxx.pro をしてみよ。きっとエラーがある。
    - 多くの場合、単なる typo
- Variable undefined
    - Variable is undefined: xxx.
  - 定義していない array にアクセスしようとしている。
    - Array を定義が、遅すぎる or 忘れている or typo。
      - 該当 variable を help してみよ。

# エラーストップする 90%の理由2

- Subscript out of range
  - あり得ない array の座標にアクセスしている。
    - % Attempt to subscript xxx with <LONG (-1)> is out of range.
    - % Subscript range values of the form low:high must be >= 0, < size, with low <= high:
    - あり得ない for loop カウンタ
    - Where 関数がマッチしていない。該当無しを表す -1 が帰ってきている。
      - Subscript を help してみよ。
      - Where 関数の Match count 返り値を使って、事前にチェックするのが鉄則。
- Function/procedure へのパラメーター渡しがおかしい。
  - % xxx: Incorrect number of arguments.
  - たいがいが typo.

# エラーはでないが、処理内容が挙動不審な場合の90%の理由1

- スカラーのハズが、size=1 の array になっている。
  - Id がスカラーなら、Help,Array[id] はスカラー
  - Id が array なら、Help,array[id] はアレイ
  - アレイA＊スカラーBは、アレイC
    - アレイCのサイズはアレイA
  - アレイA＊アレイBは、アレイC
    - アレイCのサイズはアレイA, B の小さい方のサイズ！
  - 悩む前に、help で内容を確認せよ。
- IF,whereなどの条件分岐が、期待どうりでない。
  - AND と &&, OR と || は大丈夫？
  - 悩む前に、IF 条件内容を help せよ。
- String に意図しないスペースが入っている。
  - ‘a’ と ‘_a’ と ‘a_‘ が混乱している。(‘_’ は空白）
    - Print だと気づかない。Help せよ。

## エラーはでないが、処理内容が挙動不審な場合の90%の理由2

- Integer/long integer の違い。bit が回ってマイナスになる！
  - IDL> help,32767S
  - <Expression>  INT  =  32767
  - IDL> help,32767S+1S
  - <Expression>  INT  = -32768
- 処理結果が意図せず integer になった！
  - *2 の場合も *2.0 と明記しましょう。
- Function/procedure へのパラメーター渡しがおかしい。
  - Argument 数が少なくても動く！足りない部分は undefined 扱い。
    - 多すぎる場合は、エラーとなる。
  - Argument をすべて help してみよ。
- まったく同名の、異なる procedure/function がある！
    - そんな馬鹿な、と思っても、たまに起きる。そして、大いに悩む。
  - たとえば、バックコンパチでない複数バージョンのプログラムの全てにパスが通っている、など。
  - IDL_PATH の設定を確認せよ。
  - findpro 関数 (IDLASTRO) も便利。

## IDL プログラム高速化

- 早いコンピューターを買う。
  - いたずらにプログラム上で速度を追求しない。読みやすさ優先。
    - IDL では、コードの高速化改良余地は少ないが。
  - メモリスワップしているようなら、メモリの増設が有効。
- FOR ループ、IF 文などは、なるべく使わない。
  - 細切れにせず、IDL built-in のルーチンをつかう。
    - そもそも早い。うまくいけばマルチスレッドが働く
  - Array 処理や Where を活用せよ。
- 多次元 Array アクセスの順番。
  - For loop をどちらを先に回すか？ -> コラムメイジャー
- メモリ管理（スワップ回避）
  - メモリを消費する大型 array は、
    - サブルーチンを活用して、メインプログラムに同時に多数の大型 array を残さない。
    - 不要になったら、消す。
      - A=0 を代入（こそくな手段）
      - 後でデバッグできなくなって泣く事も。

# データの QL

- IDL は Interactive に Data をいじくる言語です。データの Quick Look の技を身につけましょう。

- Help
- Print
- Print,moment
- Plot or oplot
- ATV

# QL の達人となるべし1

- 解析処理のデバッグのため、データを様々な形で眺める技を身につける。
  - エラーは出ないが、処理結果がおかしいときのデバッグこそ、IDL の強いところ。

- HELP: まずは help で中身を確かめる。
  - Variable の dimension, 内容の種別（文字か数字か undefined か）を知る。
- Print: とりあえずプリントして中身を見る。
  - 場合により、[…] を使って array の一部だけを見る。
- Print,moment(): 対象が大きい場合は、だいたいの統計量をつかむ。
  - 平均値、分散など。

# QL の達人となるべし2

- Plot,Oplot: トレンドを2次元グラフ上でつかむ。
  - 1次元 plot も可：Plot,yarray
  - Plot,xarray,yarray,xrange=[x1,x2],yrange=[y1,y2],/xlog,/yog,psym=3,color=1
  - Oplot,xarray,yarray2,psym=4,color=2
  - 2次元以上のデータは、[…] を使って次元を落とす。
    - [*,0,0] -> x 軸に沿ったプロット
- ATV: 2次元画像を見る。
  - Atv,image2d,/block
    - あとは GUI で好きにいじる。
    - /block はおまじない。要らない場合もあるが、つけた方がよいことが多い。理由は理解してません。
  - 3次元以上のデータは、[…] を使って次元を落とす。
    - [*,*,0] -> z=0 でカットした面の画像。

# サブルーチン

- 慣れてしまえば、サブルーチン化はとても簡単・便利。積極的に使いましょう。
- 2つの方法
  - Function
    - Output=function(argument,/keyword)
  - Procedure
    - procedure(input_arg,output_arg,/keyword)
- まったく同じ機能（コード）は、どちらの方法でも実現可能。
  - 呼び方が違うだけ。
  - お好きな方法で、どうぞ。

# サブルーチン分割のススメ

- プログラム実行内容の整理
- Variable の整理
  - テンポラリの variable は、サブルーチンから脱出するときに、消える。
  - 逆に、過去の情報を参照したいときは、外部の変数に記憶させる。
- プログラムの汎用化
  - 何度も呼び出せる。
  - 微妙に挙動の異なる繰り返し部は、keyword などで対応。
  - 例
    - Stacked_image=mystack(image,/median)
    - Stacked_image=mystack(image,/clip_sigma,sigma=3.)
    - Mystack.pro の中に IF 文を書き、keyword 毎に処理を分岐させる。

# Argument と keyword

- Keyword は 1 or 0 のスイッチ
  - /keyword と keyword=1 は等価
- Argument, keyword は、入出力の両方に使える。
  - Input の argument に変更を加えて、同じ変数で受け取る事が可能。
- Keyword がセットされたかどうかを、知る。
  - Keyword_set 関数
    - Size 関数も使える。Undefined なら keyword は設定されていない。

# サブルーチンの様式

- 最初に、pro または function で、I/O argument を定義する。
  - 今回は、input と output の array が一つづつ。
  - C の様に、type を指定する必要はない。なので、なんでもあり。
- Pro または function で始まり、end で終わる。
  - 全ての処理は、この二つの間に。
- Function の場合は、返り値を return で指定する。

```
Pro myprocedure,input,output
        Output=myfunction(input)
    End

Function myfunction,input
        Output=somefunction(input)
        Return,output
    end
```

# サブルーチンを使う。

- Path を通す。（既出）
- .comple でコンパイルしてみる。
  - .compile しなくても、呼ばれれば自動でコンパイルされますが。。。（既出）
- 呼び出すときは、他の IDL intrinsic 関数と同じ。

# サブルーチン構造と、メモリアクセス

- データはすべて local 扱い
  - ただし、例外的に global を作ることができる。
- IDL プロンプトからアクセスできるのは、
  - 各サブルーチン内では、そのサブルーチンのメモリ内容だけ。
  - 別のサブルーチンに移動すると、親ルーチンの内容はアクセスできなくなる。
    - IDL が、その状態でアクティブなメモリ内容を自動で切り替えている。
- 同じ変数名を親ルーチンとサブルーチンの両者で使用していても、サブルーチン毎に変数の内容は入れ替わる。
  - Local なので、当然。

# サブルーチン化とデバッグ手法1

## サブルーチン内でエラーで止まったらどうするか？ Variable の内容の問題の場合

- 今どこにいるかを知る。
  - Error message を読む or help,/trace
- バグの修正法を考える。
  - QL 手法を駆使して、variable の中身を確認。
  - 問題のある変数を探し出し、修正を考える。
- 変数を外から書き替える。
  - インタラクティブモードなので、自由にコマンドラインから変数をいじれる。
- .cont を実行する。(continue)
  - 運が良ければ、あたかもエラーがなかったかのように、処理が続く。
    - もちろん姑息な手段がうまくいかない場合も多いので、その場合は次のページへ。
- うまく切り抜けたら、サブルーチンそのものを修正する。
- 最初から実行し直す。

# サブルーチン化とデバッグ手法2

## 致命的なバグで、実行継続が不可能な場合

- 今どこにいるかを知る。
  - Error message を読む or help,/trace
- バグの修正法を考える。
  - QL 手法を駆使して、variable の中身を確認。
- プログラムを修正する。Edit and save.
- 一つ上のルーチンに戻る。
  - Return (procedure の場合) or return,0 (function の場合)
- 修正プログラムを再コンパイルする。
  - .compile ‘program’
- 修正サブルーチンを試す。
  - コマンドラインから、単体で（そのサブルーチンだけ）実行する。
- うまくいったら、最初から実行し直す。

- エラーストップしても、無駄にしないで復活させるベシ。

# （講義編の）最後に

- ざっくばらんなコメントです。

- もっと勉強したい方へ。
  - 他人のプログラムを読もう。
    - そして改造。。。
  - Reference 本の斜め読みが良いと思います。
  - IDL の友達を作りましょう。(いま周りにいます)
- もっと複雑な処理がしたい方へ。
  - 複雑な処理も、結局は単純な処理の組み合わせ・繰り返しなので、少しづつくみ上げてください。
  - サブルーチン化とデバッグのコツが分かれば、いくらでも複雑・大きなプログラムができます。
  - シンプルかつ美しいプログラムを書くことを、お勧めします。
    - 高速化は2の次。
    - Array 名に意味を持たせたり、要所要所にコメントを残す、など。

# つづき

- IDL は万能ではありません。
  - 私は今でも定期的に IRAF を使っています。Splot とか。
  - うまく組み合わせてください。
    - 基本は、低レベルのルーチンは IDL が得意です。
- IDL は、他の天文研究でも使えます。たとえば
  - 複雑かつ奇麗な plot
  - データの interpolation などの、ちょっとした数学処理。
  - 特別な統計処理
  - 大きな天文 Catalog 処理
    - where 関数が威力を発揮!
- もっとよい講習を受けたい方へ。
  - ADC のアンケートに答えてください。
  - この講習資料への具体的なフィードバック(文句)も大歓迎。
  - その他なんでも、ADC スタッフか私まで意見をください。