成都理工大学 | 地球物理学院

《地球与空间探测数据处理方法》

第三章 IDL程序设计

陶 丹

Email：adam.tao@hotmail.com

Address: 北翼楼（地物院）5814室

课程要求和考核方式

课程基础

高等数学/线性代数/概率论与数理统计
高级程序语言与程序设计(C)/Matlab 程序设计

学习目的

1. 掌握IDL可视化分析工具的基础内容，能够独立程序编写并处理地球空间数据。
2. 能够熟悉了解空间探测数据类型及数据文件的存储格式，并能够熟练利用IDL可视化工具来实现空间探测数据的读写、数据格式的转换以及相关可视化分析等。
3. 能够熟练掌握空间探测数据的坐标变化、数据的平滑处理、滤波、谱分析、相关性分析、拟合、平均值统计、中值及误差分析等方法。

考核方式

平时成绩（40%）+ 考试成绩（60%）

其中：

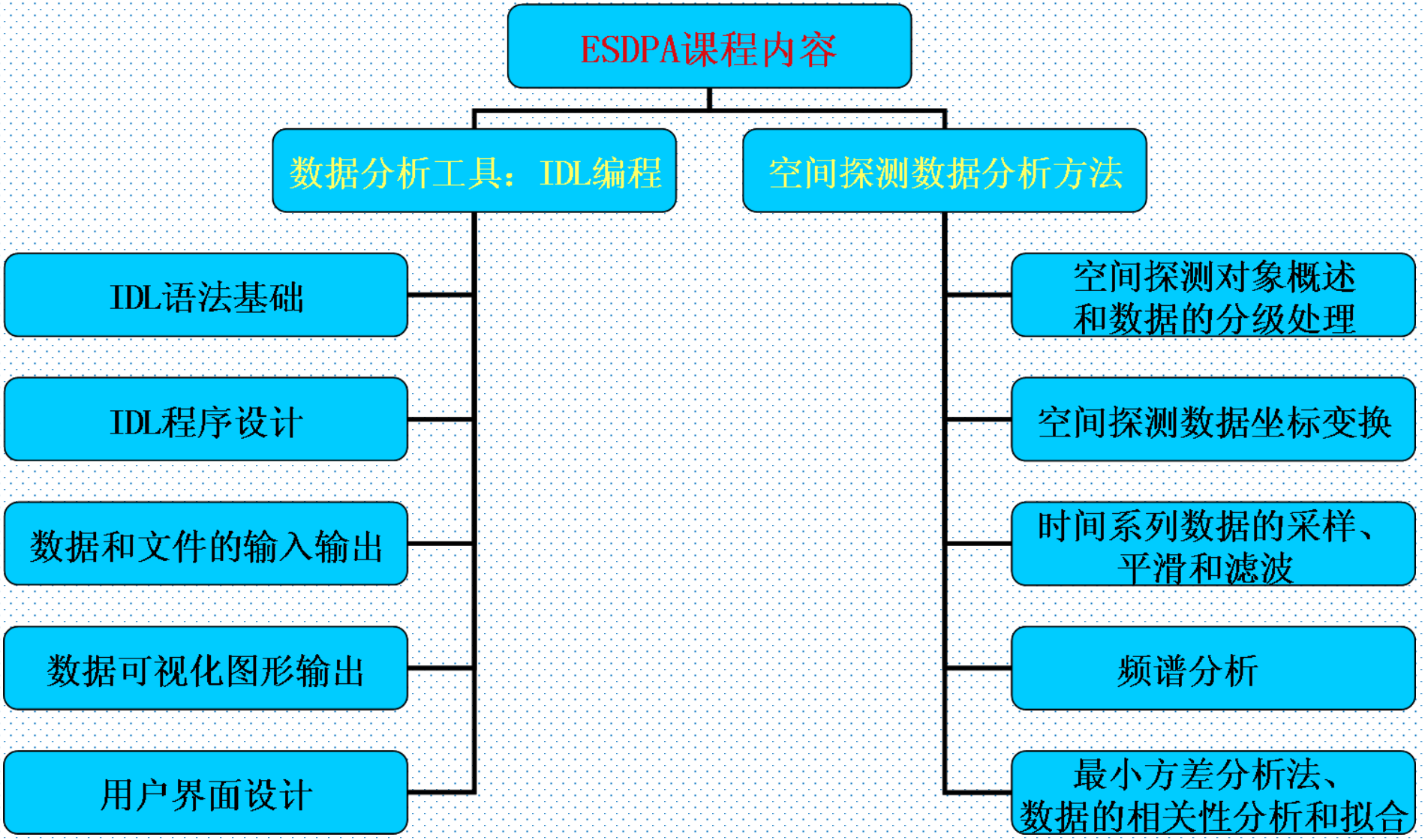
平时成绩（40%）：考勤（10%）+ 大作业/实验报告（30%）

考试成绩（60%）：期末考试（60%，闭卷）

参考书目

- 《IDL可视化工具入门与提高》，闫殿武编著，机械工业出版社，2003.
- 《IDL可视化分析与应用》，韩培友编著，西北工业大学出版社，2006.
- 《IDL程序设计—数据可视化与ENVI二次开发》，董彦卿编著，高等教育出版社，2012.

地球与空间探测数据处理方法课程架构



三、IDL程序设计：过程

3.1 IDL程序分为两种类型：过程和函数。

- 过程(PRO)： 过程一般将几个相关的操作加到一个程序模块中。

格式：PRO 过程名 [，参数1，参数2，…，参数N] [，关键字1，关键字2，…，关键字N]

命令系列（例如表达式、判断语句或循环语句等）

END

EXAMPLE-3.1

例如：*PRO READ_IMAGE,IMAGE,DATE,TIME,LANDSCAPE=LANDSCAPE,COLOR=COLOR*

;STATEMENTS…

END

EXAMPLE-3.2

过程可以被主程序或其他过程或函数来调用，其调用语法为：

@过程名 ； 调用前线编译需要调用的过程。

过程名，实参1，…，实参N[，关键字1，…，关键字N] ；实参就是已经赋值的参数。

注意事项：

- IDL的过程名命名规则与变量名的命名规则一样，且过程名不区分大小写。但是字符串里的字母是区分大小写的，譬如在设计软件进行加密时，设置的口令要区分大小写输入的。
- 保存过程的文件名必须与PRO后的过程名相同。
- 过程调用前先编译，并给过程的每个参数赋值成为实参。然后通过实参来调用过程，且实参的排列顺序和数目和函数定义的参数一样，但其关键字的顺序和个数可以和过程定义的不一样。
- 过程关键字是在过程调用时进行传输和交换的数据或者进行运行环境设置。关键字语法：KEYWORDNAME = KEYWORDSYMBOL，在函数调用时关键字的顺序和个数可变。

三、IDL程序设计：函数

3.2 函数的特点和类型

IDL函数是实现某种运算过程的功能单元，它可以被过程或其他函数多次调用，从而使程序形成模块化结构，避免多次编写相同的任务程序。

函数一般将一个操作加载到一个程序模块中，并用RETURN返回结果。

函数分为两类：

标准库函数和用户自定义函数

- 标准库函数：IDL提供了大量的标准库函数，包括数学函数、字符串函数、数组函数、图形绘制函数等。

常用标准库函数表

函数名	功能	示例
Abs(X)	计算参数X的绝对值	Print，Abs(-3.14)
Ceil(X)	返回大于或者等X的最小整数值	Print，Ceil(3.14)
Floor(X)	返回小于或者等X的最小整数值	Print，Floor(3.14)
Cir_3pnt	给出一个三点，返回圆的的半径和圆心	x = [1.0, 2.0, 3.0] y = [1.0, 2.0, 1.0] Cir_3pnt, x, y, r, x0, y0
Dist (*N* [, *M*])	creates an array in which each array element value is Proportional to its frequency	Print, Dist(100)
Exp(*X*)	计算以*e=2.718*,,,,为底，以*X*为指数的指数函数的值	Print，Exp(2)
Round(X)	返回最接近X的整数值	Print, Round(3.1)
Sqrt(X)	计算X的平方根	Print, Sqrt(2.0)
Total(A)	计算矩阵元素之和	
Systime()	返回当前系统时间	Print,systime()
Strlen(expC)	计算字符串expC的长度	Print, Len(„I am a boy!“)

Y=ALOG(X)：计算X的值的自然对数，且X的值大于0.
Y=ALOG10(X)：计算X的值的常用对数，且X的值大于0.

三、IDL程序设计：函数

- 自定义函数(FUNCTION) 格式：

  FUNCTION 函数名 [，参数1，参数2，…，参数N] [, 关键字1，关键字2，…，关键字N]

  命令系列（例如表达式等）

  RETURN，*表达式计算结果*

  END

EXAMPLE-3.3

- 函数可以被主程序或其他过程或函数来调用，其调用语法为：

  @函数名 ；调用前先编译需要调用的函数。

  函数名（实参1，实参2，…，实参N [，关键字1，…，关键字N] ）；实参就是已经赋值的参数。

- 自定义函数和调用时的注意事项：
  - 保存函数的文件名必须与FUNCTION后的函数名相同。
  - 函数调用前先编译，并给函数的每个参数赋值成为实参。然后通过实参来调用函数，且实参的排列顺序和数目和函数定义的参数一样，但其关键字的顺序和个数可一和函数定义的不一样。
  - 函数关键字是在函数调用时进行相互传输和交换的数据或者进行运行环境设置。关键字语法：KEYWORDNAME = KEYWORDSYMBOL，关键字可有可无。

UNIX平台区分源文件名大小写，非UNIX平台不区分！
建议定义（过程和函数）源文件名时统一使用小写！
如使用calculating_magnetic_pressure.pro
不使用Calculating_Magnetic_Pressure.pro

三、IDL程序设计：函数

- 自定义函数和调用时的注意事项：
  - 函数拥有自己的名称，每个函数名应该清楚地表示出那个任务，可以使用合法的标识符来命名。但其名称不能与变量、常数和关键字等重复。
  - 函数内声明的变量属于局部变量，也就是说在不同函数内声明的变量彼此互不相关，其作用域局限在该函数内。所以在不同的函数内允许声明相同局部变量名称。
  - 函数有特定功能，程序代码简单明确，可读性高而且容易调试和维护。
  - 为了提高重用性，每个函数都只应该执行一个良好定义的任务。如果无法用一个简明扼要的名称来描述函数的功用，表明你的函数执行太多的分散的任务，通常，最好将这种函数分解为几个更小的函数。

3.3 过程或函数的两种编译调用模式

（1） ‘.COMPILE’或菜单命令进行手动编译；

（2）将过程或函数添加到IDL的PATH环境中、或通过“PROJECT->ADD/REMOVE FILES”将过程或函数增加到同一个PROJECT中、 或在主程序中使用“@过程名或函数名”自动编译。

注意：IDL主程序在运行时会自动编译所需过程或函数。如果IDL调用的过程或函数之前未被编译过，则运行程序会搜索路径下所有的文件夹以搜索源文件的名称。没有包括在IDL库函数、LIB、PATH环境变量或项目PROJECT中的过程或函数将不能被编译和调用。

例如：IDL> MAP_SET,/CONTINENTS

同时，为保证自动编译的方便性，可以使用以下方法之一：

- 保证每个过程和函数在不同的源文件中
- 如果相关函数和过程保存在同一个源文件中时，则将相对主要的过程或函数放在后面

例如 **hello2.pro**：

```
FUNCTION square,a
    return,a*a
END

PRO hello2
    print,'hello world!'
    print,'the square of 2 is', square(2)
END
```

3.4 IDL主程序的语法结构和过程一样，由PRO开头和END结束。

- 不过主程序名后面一般不需要设置参数或关键字，这一点和过程不一样。
- 另外，IDL主程序只能调用过程和函数，但是过程和函数可以相互调用。
- 主程序通过“@过程或函数名”来调用过程或函数，且@语句要放在主程序顶端，即在PRO之前使用。
- 主程序语法格式如下：

```
@MyFunctionFile
@MyProcedureFile
PRO MAIN_CODE_NAME
    MyFunctionFile(实参1，…,实参n [,关键字1，…,关键字n] )
    MyProcedureFile, 实参1，…,实参n [,关键字1，…,关键字n]
    命令系列（例如表达式、判断语句或循环语句等）
END
```

- 程序运行时遇到错误后返回主层次

程序出错导致运行中断，默认IDL在产生错误的过程或函数中间暂停。
例如**mod_test.pro**

```
FUNCTION mod_operator,x,y
    return,x mod y
END

PRO mod_test,a,b
    print,'mod test!'
    print,'a mod b =', mod_operator(a,b)
END
```

```
IDL> a=5
IDL> b=3
IDL> mod_test,a,b
       mod test!
       a mod b =       2
```

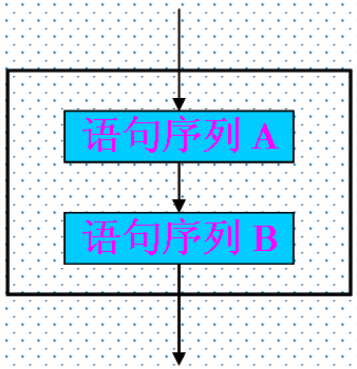
```
IDL> a=5
IDL> mod_test,a,c
mod test!
% Variable is undefined: Y.
% Execution halted at: MOD_OPERATOR       2  mod_test.pro
%                     MOD_TEST          7  mod_test.pro
%                     $MAIN$
```

;; 自变量c未定义时调用函数，程序暂停在此函数**mod_operator**中间位置，即第2行。
;; 函数**mod_operator**在源文件第7行被**mod_test**过程调用。
;; 最后显示“**mod_test**”从主层次（命令行层次）被调用。

三、IDL程序设计：程序结构

程序设计的基本结构

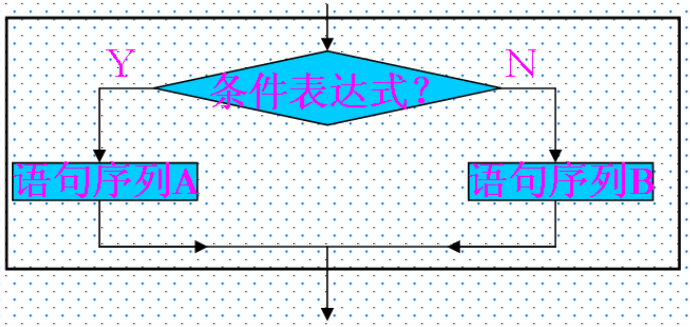
- 顺序结构



依据语句排列先后，依次执行

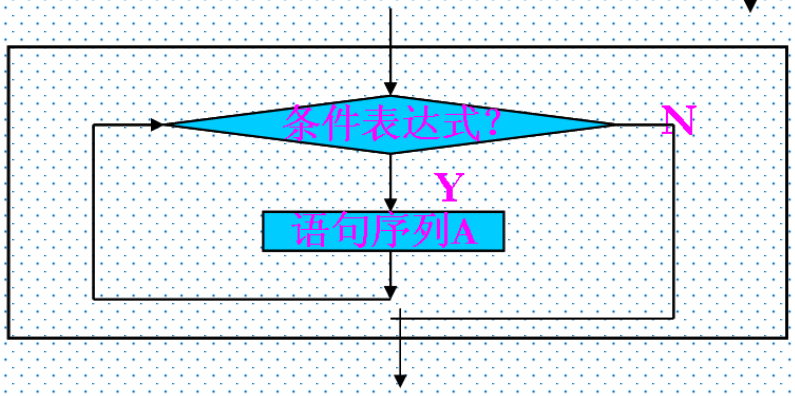
- 选择结构

依据不同条件执行不同语句



- 循环结构

循环执行多次



三、IDL程序设计：选择控制语句

3.5 IF语句

当特定的条件为真时，IF语句执行单个语句或一个语句快（多个语句）。

- 简单的IF…THEN…语句：

  格式1： if *condition* then *statement*

  格式2： if *condition* then begin
  statement
  endif

- IF…THEN…ELSE…语句串：

  格式1：if *condition* then *statement1* *else statement2* ；*statement2*执行condition不成立的情况

  格式2：if *condition* then begin
  statement1
  endif else begin
  statement2
  endelse

CONDITION 通常是一个逻辑（布尔）判断表达式，而*STATEMENT(S)*是IDL命令（如数学表达式、输入或输出等）。

三、IDL程序设计：选择控制语句

- IF语句的嵌套

```
if condition1 then begin
    if condition2 then begin
        statement1
    endif else begin
        statement2
    endelse
endif else begin
    if condition3 then begin
        statement3
    endif else begin
        statement4
    endelse
endelse
```

EXAMPLE-3.4

3.6 CASE语句

有时会遇到一个判断条件，其判断结果可能有一些不同的值，无法用真和假来表示，这时就应该使用CASE语句。CASE语句根据一个标量的表达式来选择某个语句或语句块运行。CASE语句常用于响应程序的不同的事件从而执行相应的代码。

格式1：
```
CASE condition of
    condition 1：statement 1
    ……
    condition n：statement n
ENDCASE
```

格式2：
```
CASE condition of
    condition 1：begin
                    statement(s) 1
                end
    ……
    condition n：begin
                    statement(s) n
                end
    else ：begin
                statement(s) n+1
           end
ENDCASE
```

EXAMPLE-3.5

当表达式和其中的某个情况匹配，相应的语句被执行， CASE语句结束。如果没有匹配的情况，那么执行ELSE下的语句，如果没有ELSE语句，将会发生错误，建议在CASE 语句中都加上ELSE.

3.7 FOR语句

FOR循环运用计数器来多次执行一个或多个语句。

格式1： for *i=v1, v2* do *statement* ;步长默认为1

for *i=v1,v2, inc* do *statement* ;可以将步长*inc* 为设定任何整数值

适用于执行单个语句

格式2： for *i=v1, v2, inc* do begin

statements

endfor

适用于执行一个语句块

*inc*为正：最后依次循环操作发生在*i*小于或等于*v2*时；
*inc*为负：最后依次循环操作发生在*i*大于或等于*v2*时。

功能：用变量*i*来控制循环次数，使序列*statements*循环执行指定的次数。

对于循环次数大于32767的，需要使用长整形控制变量。

错： for *i=0,40000,10000* do *print,i*

对： for *i=0L ,40000L,10000L* do *print,i*

当然，*i* 也可以时浮点型循环变量

EXAMPLE-3.6

3.8 WHILE 循环控制语句

在判断条件为真时，循环不断地执行一条或多条语句。

格式1：**while** *condition* **do** *statement*

格式2：**while** *condition* **do begin**

statement(s)

endwhile

在循环开始进行条件判断

EXAMPLE-3.7

3.9 REPEAT...UNTIL循环语句

与WHILE 循环相似，不同之处在于，REPEAT...UNTIL循环语句在循环末尾进行条件判断，而不是在循环的开始。

格式1：**repeat** *statement* **until** *condition*

格式2：**repeat begin**

statement(s)

endpep until *condition*

在循环末尾进行条件判断

例如：**repeat begin**

*index =amount * 10*

number =number +index

endpep until number Gt 100

三、 IDL程序设计：跳转控制语句

3.10 CONTINUE语句

使得FOR、WHILE或REPEAT循环得以连续进行。

适用于当需要结束当前循环而进入下一个循环时

例：计算并输出2 + 4 + … + 10000

EXAMPLE-3.8

3.11 BREAK语句

使执行从FOR、WHILE、REPEAT等循环语句或CASE、SWITCH等选择语句中及时跳出。同时控制将转到循环结束后的语句或CASE、SWITCH语句的下一种情况。

用于无条件退出包含该BREAK语句的最内层循环，转向执行该层循环语句

例：编程序sum.Pro，要求从键盘任意输入两个数，计算并输出这两个数的和；然后询问是否重复执行上述操作，直到不愿意执行为止。

BREAK语句只离开当前循环的层次！！！

EXAMPLE-3.9

3.12 GOTO语句

指定一个程序标识符，当程序执行到GOTO语句时，程序就跳转到程序标识符所在的程序行。

格式：

goto , 标号

标号表示为：

label：

尽量少用GOTO语句！
若需使用，尽量让GOTO语句往前跳，且使用次数不超过一次！

例如：可以用GOTO语句来打断一个FOR循环

```
for j =0, n do begin
    index= j * ! pi * 5.165 * thistestvalue
    if index gt 3000.0 then goto, jump out   ;jump out 是标号
endfor
jump out: print, index          ;程序标识符后面要有一个冒号。
```

EXAMPLE-3.10

3.13 SWITCH语句

将执行转到‘of’后面一系列情况中的第一种匹配情况（即表达式的值与它的条件值相符合的情况），就执行与它相应语句序列和它后面的所有语句序列。

格式：

```
switch expression of                          ;expression=case1 or case2 … ?
    case 1: begin
                statement 1
            end
    case 2: begin
                statement 2
            end
    ……
    case n: begin
                statement n
            end
  [ else begin
                statement n+1
            end]
endswitch
```

EXAMPLE-3.11

3.14 RETURN语句

在当前程序单元中产生一个即时出口，并返回控制。

格式1：return, 结果

返回一个变量“结果”。其在函数中具有强制性，因为函数必须返回一个变量值。

格式2：return

在过程中使用。

每个函数中RETURN语句使用次数尽量为1次；
否则，则要确定哪一个先返回。

3.15 参数和关键字

过程和函数可以接受两种类型的数据，即参数和关键字。

■ 参数

函数和过程相互调用时，进行相互传输和交换的数据，实现数据在两者之间的相互传递。

格式：**PRO** 过程名 [，参数1，参数2，…，参数N]

命令系列

END

FUNCTION 函数名 [，参数1，参数2，…，参数N]

命令系列

RETURN，表达式

END

注意事项：

- 参数定义时要有确定的顺序及类型；在使用时为必选项。
- 参数定义时可以没有确定的值（形参），只要确定其位置和类型，可以为变量、数组、结构体、指针或对象；在使用时要有确定值（实参）。
- 参数使用时，要先定义再使用。
- 参数使用时，其数量、类型和顺序要和定义时保持一致。

3.15 参数和关键字

过程和函数可以接受两种类型的数据，即参数和关键字。

■ 关键字

函数和过程相互调用时，进行相互传输和交换的数据或者进行运行环境的设置。

格式：**PRO** 过程名 [，参数] [，关键字1=关键字变量1，…，关键字N=关键字变量N]

命令系列　　关键字名　关键字值

END

FUNCTION 函数名 [，参数] [，关键字1=关键字变量1，…，关键字N=关键字变量N]

命令系列

RETURN，表达式

END

注意事项：

- 关键字定义时顺序任意；在使用时为可选项。
- 关键字定义时可以没有确定的值（形字），只确定类型，可以为变量、数组、结构体、指针或对象；在使用时要有确定值（实字）。
- 关键字使用时，要先定义再使用，且类型要和定义时一致。
- 关键字使用时，顺序可以定义时不一致，且顺序任意；数量也可以比定义时少。

```
HELP, ISTRUCT, /STRUCTURE
PRINT, IARRAY, FORMAT='(7F20)'
```

三、IDL程序设计：变量的传递机制

- IDL变量的传递方式分为按值传递和按地址传递：

- 按值传递：
  仅仅把参数和关键字的值传递给相应的函数和过程的形参。在函数或过程运行过程中所有对参数的和关键字的操作，在函数和过程调用结束后不影响参数和关键字的原值。

  以值传递的自变量，其类型、大小和值在被调用的过程和函数中都可以被改变，但改变后的自变量不传递到调用函数。

  按值传递的数据有常量、数组元素、结构体成员、系统变量和表达式（不包含变量）。

- 按地址传递：
  把参数和关键字的地址传递给相应的函数和过程的形参。在函数或过程运行过程中所有对参数的和关键字的操作，在函数和过程调用结束后直接影响参数和关键字的原值。

  以地址传递的自变量，其类型、大小和值在被调用的过程和函数中也都可以被改变，但改变后的自变量传递到调用函数。

  按地址传递的数据有变量名、数组、结构名和指针。

EXAMPLE-3.12

三、IDL程序设计：参数和关键字

特殊关键字

当关键字被传递到过程或函数时，有时需要将这些关键字从调用程序传递到另外一个调用函数。

两个特殊关键字：**_EXTRA和_REF_EXTRA**

_EXTRA=EXTRA_KEYWORDS（按值传递）

_REF_EXTRA=EXTRA_KEYWORDS（按地址传递）

用于在过程或函数设计中，要调用一个已经创建好的程序，且要保留其原有的关键字仍然生效。

```
;extra_plot.pro
PRO EXTRA_PLOT,X,Y,NAME=NAME,_EXTRA=EXTRA_KEYWORDS
if (n_params() ne 2) then message,'usage: extra_plot,x,y'
if (n_elements(x) eq 0) then message,'argument x is undefined!'
if (n_elements(y) eq 0) then message,'argument y is undefined!'
if (n_elements(name) eq 0) then name='Adam'

;plot the data
plot,x,y,_extra=extra_keywords

;print name and date on the plot
date=systime()
xyouts,.0,.0,name,align=.0,/normal
xyouts,1.,.0,date,align=1.,/normal
END
```

```
IDL>X=FINDGEN(170)*.1
IDL>EXTRA_PLOT,X,COS(X),XRANGE=[.0,17.],XTITLE='X',YTITLE='COS(X)'
```

参数和关键字检测

过程或函数被调用时，IDL自动进行以下检测：

- 传递的参数是否比定义时多
- 是否传递了未知关键字，且_EXTRA关键字是否包含在定义中

一旦上述检测为真，程序将显示错误信息并中断。

表 3.3 用于检测自变量的程序

名　　称	功　　能
n_params( )	返回传递的参数数目（不包括关键字）
n_elements( )	返回一个变量中元素的数目（零表示没有定义）
size( )	返回一个变量的类型和大小信息
arg_present( )	如果自变量存在，并且由地址传递，则返回真
message	显示一个信息，并且中断执行

```
if (size(x, /n_dimensions) ne 1) then message, 'X must be a 1D array!' [, /continue]
```

- 检测布尔关键字，使用**KEYWORD_SET（关键字名）**函数，若变量已定义，则返回真值。
- 若参数或关键字在调用过程中存在，且通过地址传递（如果输出参数在调用过程中创建、更改，并被返回，则必须通过地址传递），则使用**ARG_PRESENT**函数返回真值。

例如：*if arg_present(xrange) then xrange=[min(x)-1.,max(x)+1.]*

三、IDL程序设计：变量的范围

3.16 全局变量和局部变量

IDL程序中所有调用的参数或关键字都可以是变量，根据其使用范围分为局部变量和全局变量两种。

- 局部变量：是指整个程序运行过程中，仅在程序中局部有效的内存变量。

  任何程序中的局部变量，不管其名称是否相同，均只在本程序内部有效，即进入程序生效，退出程序取消，并自动释放其所占有的内存空间。

```
pro main
    var = 10
    print,'var =',var
end
```

- 全局变量：是指整个程序运行过程中，始终有效的变量。

  · 系统变量：系统变量的名称均由感叹号“ ！”开头。

  （1）内部系统变量（包括只读和可写的两种）。

  （2）自定义系统变量：使用 **DEFSYSV**.

  DEFSYSV,‘变量名’，表达式 [， READ_ONLY][EXISTS =VARIABLE]

  · 公共变量：指由COMMON创建的全局变量。

  自定义公共变量格式：COMMON 公共变量模块名称，变量1，…，变量N

  一旦定义了公共模块，其公共变量个数不能改变，但其变量类型和数值可以改变。同时，任何指向该模块的程序单元都可以获取模块的变量，但需要先声明。

三、IDL程序设计：变量的范围

- 公共变量：指由COMMON创建的全局变量。

自定义公共变量格式：COMMON 公共变量模块名，变量1，…，变量N

一旦定义了公共模块，其公共变量个数不能改变，但其变量类型和数值可以改变。同时，任何指向该模块的程序单元都可以获取模块的变量，但需要先声明。

如果没有必要，尽量不使用公共模块！！！

例如：以下过程使用公共模块存储一个长整型值及该值设置的时间。

```
PRO SET_NUMBER
common set_number_info,number,time
number=0L
read,prompt='Enter a number:',number
time=systime()
END

PRO GET_NUMBER
common set_number_info,number,time (or common set_number_info)
print,'The last number set was',number,'at',time
END
```

if common set_number_info,number **then** an error will occur!

- 不提倡使用公共模块在单个程序或相关程序间传递变量，其让程序维持变得困难：如在已定义公共模块中增加变量，则必须离开IDL或者重新启用一个IDL时段。
  .reset_session命令可重设IDL时段而不退出IDL。
- 提倡使用指针和结构体来传递，尽量不使用公共模块。

3.17 关系和布尔运算符的使用

在IDL的条件控制语句中（IF) 经常将关系运算符和布尔运算符联合起来组成负复杂的逻辑结构。

通常将关系运算符(EQ, NE, LE, LT, GE, GT)的判断条件用（）号括起来，然后使用布尔运算符（NOT, AND, OR, XOR)将两个或两个以上的关系判断语句连接起来使用，中间空格。

EXAMPLE-3.13

3.18 错误处理

- 输入变量（文件）检测

N_ELEMENTS( )函数可以返回一个变量（例如数组或字符串）中元素的数目（零表示没有定义）。该函数常常用于检测自定义函数或过程所调用的数据文件是否存在，若存在就执行函数或过程的命令语句。若不存在就通过MESSAGE输出一个信息提示错误，并中断运行程序。

EXAMPLE-3.14

- 运行错误处理

当IDL过程或程序中遇到重大错误时，在默认情况下可，程序将在出错处中断运行。也可以使用ON_ERROR的赋值来调试程序.

```
ON_ERROR,0          ; 默认自动中断处理：在错误处立即停止
ON_ERROR,1          ; 返回主程序（主层次）
ON_ERROR,2          ; 返回调用出错程序的上一级程序
ON_ERROR,3          ; 返回出错的程序
```

- 数学错误提示

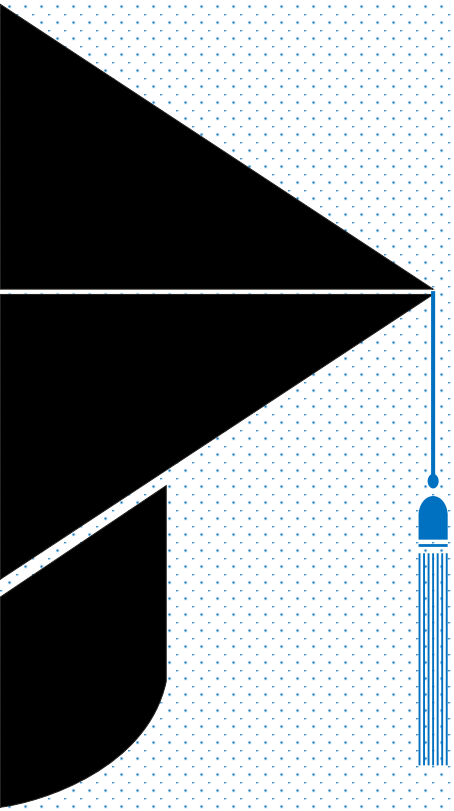
（1）输出-NAN（不是一个数字）表示负数被开方

```
IDL> PRINT,SQRT(-2.0)
      -NAN
% PROGRAM CAUSED ARITHMETIC ERROR: FLOATING ILLEGAL OPERAND
```

（2）输出INF（无限）表示浮点数被零除

```
IDL> PRINT,2.0/0.0
       INF
% PROGRAM CAUSED ARITHMETIC ERROR: FLOATING DIVIDE BY 0
```

成都理工大學
CHENGDU UNIVERSITY OF TECHNOLOGY



演示完毕

Thank You