

第二讲 IDL语法基础 和控制语句

石雪

测绘地理信息学院

语法基础

- 1. 变量
- 2. 数组
- 3. 字符串
- 4. 结构体
- 5. 指针
- 6. 对象
- 7. 链表
- 8. 哈希表
- 9. 运算符
- 10. 其他符号

1 变量

- 局部变量
- 系统变量

1 变量

- 局部变量

- 命名规则

- 最大长度不超过255个字符
 - 变量的首位：字母，下划线 “_”
 - 变量的中后部：字母、数字、下划线 “_” 和连接符 “\$”

1 变量

- 局部变量

- 命名规则

- 例如:

- `abc_3$d` `ok_24_bit` `IDL_type` `variable`
 - `_day_month_year`
 - `4_line` `abc.cha` `one%file`

1 变量

- 局部变量

- 相关函数

- 变量名是否有效

- 格式: IDL_VALIDNAME (String [, /CONVERT_ALL] [, /CONVERT_SPACES]),

```
IDL控制台 - 历史命令 | 问题
IDL> idl_validname('result')
result
IDL> idl_validname('1result')
```

```
IDL控制台 - 历史命令 | 问题
IDL> idl_validname('%result')
IDL> idl_validname('a_result')
a_result
IDL> idl_validname('_result')
_result
```

1 变量

- 变量命名方法

- 匈牙利命名法：

- 开头字母用变量类型的缩写，其余部分用变量的英文或英文的缩写，单词第一个字母大写。

- iMyAge: “i” 是int类型的缩写；

- cMyName: “c” 是char类型的缩写；

- fManHeight: “f” 是float类型的缩写；

1 变量

- 变量命名方法

- 下划线命名法：

- 单词间用 “_” 分隔

- my_age

- my_name

- man_height

1 变量

- 变量命名方法

- 驼峰式命名法（小驼峰）：

- 第一个单词首字母**小写**，后面其他单词首字母大写。

- myAge

- myName

- manHeight

1 变量

- 变量命名方法

- 驼峰式命名法（大驼峰/帕斯卡命名法）

- 每一个单词的首字母都大写

- MyAge

- MyName

- ManHeight

1 变量

●数据类型

数据类型	字节数	范围	创建变量	类型转换
字节型	1	0 至 255	Var=0B	byte()
16 位有符号整型	2	-32768 至 32767	Var=0	fix()
32 位有符号长整型	4	-2^{31} 至 $2^{31}-1$	Var=0L	long()
64 位有符号整型	8	-2^{63} 至 $2^{63}-1$	Var=0LL	long64()
16 位无符号整型	2	0 至 65535	Var=0U	uint()
32 位无符号长整型	4	0 至 $2^{32}-1$	Var=0UL	ulong()
64 位无符号整型	8	0 至 $2^{64}-1$	Var=0ULL	ulong64()
浮点型	4	-10^{38} 至 $10^{38}-1$	Var=0.0	float()
双精度浮点型	8	-10^{308} 至 $10^{308}-1$	Var=0.0D	double()
复数	8	-10^{38} 至 $10^{38}-1$	Var=Complex(0.0, 0.0)	complex()
双精度复数	16	-10^{308} 至 $10^{308}-1$	Var=Dcomplex(0.0D, 0.0D)	dcomplex()
字符串	0-32767	None	Var="" 或 Var=""""	string()
指针	4	None	Var=Ptr_New()	
对象	4	None	Var=Obj_New()	

1 变量

● 类型转换

类型转换	函数名称	示例	
		操作	结果
字节型	BYTE	BYTE(1.2)	1B
整型	FIX	FIX(2.5)	2
无符号整型	UINT	UINT([5.5,-3])	5 65533
长整型	LONG	LONG(65538.5)	65538
无符号长整型	ULONG	ULONG([5.5,-3])	5 4294967293
64 位长整型	LONG64	LONG64([5.5,-3])	5 -3
无符号 64 位长整型	ULONG64	ULONG64([5.5,-3])	5 18446744073709551613
浮点型	FLOAT	FLOAT([5.5,-3])	5.50000 -3.00000
双精度类型	DOUBLE	DOUBLE([5.5,-3])	5.5000000 -3.0000000
复数类型	COMPLEX	COMPLEX(1, 2)	(1.00000, 2.00000)
双精度复数类型	DCOMPLEX	DCOMPLEX(1, 2)	(1.0000000, 2.0000000)

1 变量

- 系统变量

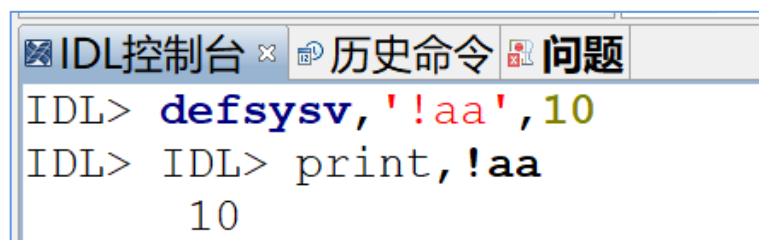
- 自定义系统变量

- 格式: `Defsysv, ' !变量名', 值`

- 系统变量使用

- 格式: `!变量名`

- 例如:



```
IDL控制台 x 历史命令 问题
IDL> defsysv, '!aa', 10
IDL> IDL> print, !aa
      10
```

1 变量

- 系统变量

- 常用的系统变量

```
IDL> !cpu
{
    "HW_VECTOR": 0,
    "VECTOR_ENABLE": 0,
    "HW_NCPU": 12,
    "TPOOL_NTHREADS": 12,
    "TPOOL_MIN_ELTS": 100000,
    "TPOOL_MAX_ELTS": 0
}
IDL> !dir
D:\Program Files\Exelis\IDL85
```

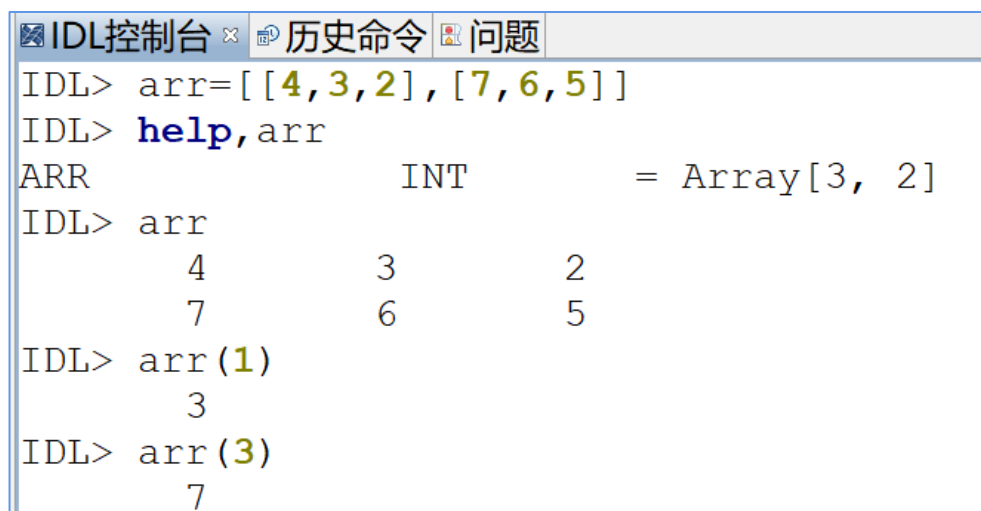
2 数组

- 定义： 把具有相同类型的若干变量按有序的形式组织起来, 这些按序排列的同类数据元素的集合称为数组。
- 特点：
 - IDL支持0→8维数组
 - 数组的下标是先列后行

2 数组

●数组创建

●赋值创建 使用方括号[]



```
IDL控制台 x 历史命令 问题
IDL> arr=[[4,3,2],[7,6,5]]
IDL> help,arr
ARR          INT          = Array[3, 2]
IDL> arr
      4      3      2
      7      6      5
IDL> arr(1)
      3
IDL> arr(3)
      7
```

```
IDL> arr = [1,2,3]
IDL> help, arr
ARR          INT          = Array[3]
IDL> arr = [1,2,3B]
IDL> help, arr
ARR          INT          = Array[3]
IDL> arr = [1,2,3D]
IDL> help, arr
ARR          DOUBLE       = Array[3]
```


2 数组

- 数组创建
 - 函数创建
 - 全零数组
 - 索引数组

数据类型	创建全 0 数组	创建索引数组
字节型	<code>bytArr()</code>	<code>bindgen()</code>
16 位有符号整型	<code>intarr()</code>	<code>indgen()</code>
32 位有符号长整型	<code>lonarr()</code>	<code>lindgen()</code>
64 位有符号整型	<code>lon64arr()</code>	<code>l64indgen()</code>
16 位无符号整型	<code>uintarr()</code>	<code>uindgen()</code>
32 位无符号长整型	<code>ulongarr()</code>	<code>ulindgen()</code>
64 位无符号整型	<code>ulon64arr()</code>	<code>ul64indgen()</code>
浮点型	<code>fltarr()</code>	<code>findgen()</code>
双精度浮点型	<code>dblarr()</code>	<code>dindgen()</code>
复数	<code>complexarr()</code>	<code>cindgen()</code>
双精度复数	<code>dcomplexarr()</code>	<code>dcindgen()</code>
字符串	<code>strarr()</code>	<code>sindgen()</code>
指针	<code>ptrarr()</code>	
对象	<code>objarr()</code>	

2 数组

- 数组创建

- 函数创建

- 全零数组:

- 索引数组:

```
IDL控制台 x 历史命令 问题
IDL> arr=bytarr(4,3)
IDL> arr
  0  0  0  0
  0  0  0  0
  0  0  0  0
IDL> help,arr
ARR          BYTE          = Array[4, 3]
IDL> arr=intarr(4,3)
IDL> help,arr
ARR          INT           = Array[4, 3]
```

```
IDL控制台 x 历史命令 问题
IDL> arr=bindgen(4,3)
IDL> arr
  0  1  2  3
  4  5  6  7
  8  9 10 11
IDL> help,arr
ARR          BYTE          = Array[4, 3]
IDL> arr=indgen(4,3)
IDL> help,arr
ARR          INT           = Array[4, 3]
IDL> arr=findgen(4,3)
IDL> help,arr
ARR          FLOAT         = Array[4, 3]
```

2 数组

●数组创建

●函数创建

●函数：

●MAKE_ARRAY(列，行，/数据类型， value=数值)

Syntax

```
Result = MAKE_ARRAY ( [D1[, ..., D8]], DIMENSION=vector, INCREMENT=value, /INDEX, /NOZERO, SIZE=vector, START=value,  
TYPE=type_code, VALUE=value, /BOOLEAN, /BYTE, /COMPLEX, /DCOMPLEX, /DOUBLE, /FLOAT, /INTEGER, /L64, /LONG, /OBJ,  
/PTR, /STRING, /UINT, /UL64, /ULONG )
```

Type Code	Type Name	Data Type
0	UNDEFINED	Undefined
1	BYTE	Byte
2	INT	Integer
3	LONG	Longword integer
4	FLOAT	Floating point
5	DOUBLE	Double-precision floating
6	COMPLEX	Complex floating
7	STRING	String
8	STRUCT	Structure
9	DCOMPLEX	Double-precision complex
10	POINTER	Pointer
11	OBJREF	Object reference
12	UINT	Unsigned Integer
13	ULONG	Unsigned Longword Integer
14	LONG64	64-bit Integer
15	ULONG64	Unsigned 64-bit Integer

2 数组

●数组创建

●函数创建

```
IDL> arr=make_array(4,3,/int,value=10)
IDL> help,arr
ARR          INT          = Array[4, 3]
IDL> arr
    10      10      10      10
    10      10      10      10
    10      10      10      10
IDL> arr=make_array(4,3,/float,value=10)
IDL> help,arr
ARR          FLOAT        = Array[4, 3]
IDL> arr
    10.000000    10.000000    10.000000    10.000000
    10.000000    10.000000    10.000000    10.000000
    10.000000    10.000000    10.000000    10.000000
IDL> arr=make_array(4,3,type=4,value=10)
IDL> help,arr
ARR          FLOAT        = Array[4, 3]
```

Type Code	Type Name	Data Type
0	UNDEFINED	Undefined
1	BYTE	Byte
2	INT	Integer
3	LONG	Longword integer
4	FLOAT	Floating point
5	DOUBLE	Double-precision floating
6	COMPLEX	Complex floating
7	STRING	String
8	STRUCT	Structure
9	DCOMPLEX	Double-precision complex
10	POINTER	Pointer
11	OBJREF	Object reference
12	UINT	Unsigned Integer
13	ULONG	Unsigned Longword Integer
14	LONG64	64-bit Integer
15	ULONG64	Unsigned 64-bit Integer

2 数组

- 数组使用 使用方括号 []
 - 下标方式
 - 使用下标来读取数组中的元素。
 - 向量方式
 - 通过向量下标获得数组中的系列元素。
 - 子数组选取
 - 下标的方式

```
IDL> arr=4*indgen(9)
IDL> arr
      0      4      8     12     16     20     24     28     32
IDL> a=[1,3,5,7]
IDL> arr(a)
      4     12     20     28
```

2 数组

●数组使用

```
IDL> arr=bindgen(4,3)*2
IDL> print, arr
```

0	2	4	6
8	10	12	14
16	18	20	22

```
IDL> arr[2,1]
12
IDL> arr[1,2]
18
IDL> arr[1:2,1:*]
10 12
18 20
IDL> arr[[1,3],1:*]
10 14
18 22
```

```
IDL> arr = indgen(10)
IDL> print, arr
```

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

```
IDL> print, arr[3]
3
IDL> indices = [0,1,3,7]
IDL> print, arr[indices]
0 1 3 7
IDL> print, arr[4:8]
4 5 6 7 8
IDL> print, arr[4:*]
4 5 6 7 8 9
```

2 数组

- 数组存储
 - 数组按行存储
 - 一维数组
 - 二维数组

$\text{arr}[0] \rightarrow \text{arr}[1] \rightarrow \dots \rightarrow \text{arr}[m-1]$

$\text{arr}[0,0] \rightarrow \text{arr}[1,0] \rightarrow \text{arr}[2,0] \dots \text{arr}[m-1,0] \rightarrow$

$\text{arr}[0,1] \rightarrow \text{arr}[1,1] \rightarrow \text{arr}[2,1] \dots \text{arr}[m-1,1] \rightarrow$

...

$\text{arr}[0,n-1] \rightarrow \text{arr}[1, n-1] \rightarrow \text{arr}[2, n-1] \dots \text{arr}[m-1, n-1]$

3 字符串

- 创建字符串及字符串数组 使用引号 ‘ ’ 或” ”

- 直接赋值

```
IDL> str='abc'  
IDL> str  
abc  
IDL> str="a'b'c"  
IDL> str  
a'b'c
```

- 创建字符串数组

- StrArr(维数)

```
IDL> arr=strarr(6)  
IDL> help,arr  
ARR          STRING      = Array[6]  
IDL> arr=strarr(3,2)  
IDL> help,arr  
ARR          STRING      = Array[3, 2]
```

```
IDL> arr=strarr(6)  
IDL> arr(0)='name'  
IDL> arr(1)='age'  
IDL> arr(2)='city'  
IDL> arr(3)='work'  
IDL> arr(4)='address'  
IDL> arr(5)='tele'  
IDL> print,arr  
name age city work adress tele
```


3 字符串

●字符操作函数

STRCMP(str1,str2,N,/FOLD_CASE)	对两个字符串进行比较,如果 N 存在只对前 N 个进行比较, /FOLD_CASE 表示模糊比较
STRCOMPRESS(str1)	删除字符串 str1 中的空格
STREGEX()	正则表达式
STRJOIN()	字符串相连接;
STRLEN()	返回字符串的长度
STRLOWCASE()	将所有的大写字母改写成小写字母
STRTRIM(str,Flag)	移去字符串中的空格 Flag: 0(移去右边空格), 1(移去左边空格), 2 (移去两边的空格)
STRUPCASE()	将所有的小写字母改写成大写字母

3 字符串

● 字符操作函数

STRMATCH(Str1,str2)	字符串 Str1 中是否存在 Str2，可以使用通配符；
STRMID(Str1,pol,Len,REVERSE_OFFSET)	从字符串 pol 开始取出 Len 个字符。字符串的第一个字符的位置为 0
STRPOS(Exp_Str1,Sea_Str2,Pos,REVERSE_OFFSET/REVERSE_SEARCH]	从一个字符串中查找与另外一个字符串完全匹配的起始点所在的位置 Pos 查找点的起始位置，默认值为 0，1（如果指定 /REVERSE_SEARCH]）指定时，则表示从开始的 Pos 起，或者从末尾开始的 Pos 其（如果指定 REVERSE_OFFSET）
STRPUT,Des_Var_str,Sou_str,Pos	将 Sou_str 字符串插入到变量 Des_Var_str 之中 POS 插入点的默认值为 0Sou_str 不为字符串，则按默认格式自动转化从 POS 处开始插入 Des_Var_str，如果插入值的位置超过了 Des_Var 的最大长度，则自然截断
STRSPLIT(Str1)	根据特定的要求拆分字符串 str1；

3 字符串

- 字符串操作函数

- `strmid` (字符串, 位置, 长度)

- `strpos` (字符串1, 字符串2, 起始位置)

```
IDL> str='abc2021def.img'  
IDL> a=strpos(str, '2')  
IDL> year=strmid(str, a, 4)  
IDL> print, year  
2021  
IDL> b=strpos(str, '.')  
IDL> str1=strmid(str, b+1, 3)  
IDL> print, str1  
img
```

4 结构体

- 结构体 使用花括号 { }
- 标量、数组、字符的集合，是复合变量
- 创建结构体
 - 结构体 = { 成员名1: 成员值1, 成员名2: 成员值2, ... }

4 结构体

- 结构体 使用花括号 { }

- 创建结构体

- 命名结构体

- 匿名结构体

```
IDL> str={two,v1:0,v2:'123',v3:intarr(3)}  
IDL> help,str  
** Structure TWO, 3 tags, length=32, data length=24:  
V1          INT          0  
V2          STRING      '123'  
V3          INT          Array[3]
```

```
IDL> str={name:'tom',age:18,height:180.0,city:'guilin'}  
IDL> help,str  
** Structure <1882e3e0>, 4 tags, length=40, data length=38, refs=1:  
NAME        STRING      'tom'  
AGE          INT          18  
HEIGHT       FLOAT       180.000  
CITY         STRING      'guilin'
```

4 结构体

- 访问结构体

- StructName.VarName

- StructName.(idx)

- 修改结构体

- 创建了结构体后，其域的类型和大小就将不能再改变了，但可以进行类型转换。

```
IDL> str.name  
tom  
IDL> str.age  
18  
IDL> str.height  
180.00000  
IDL> str.CITY  
guilin
```

```
IDL> str.(0)  
tom  
IDL> str.(2)  
180.00000
```

```
IDL> str.name=100  
IDL> str  
{  
    "NAME": "100",  
    "AGE": 18,  
    "HEIGHT": 180.00000,  
    "CITY": "guilin"  
}
```

4 结构体

●结构体相关函数

函数名	用途
CREATE_STRUCT ()	根据给定的名字和值创建结构体，并能连接结构体；
Help, **/, /Struct	返回输入结构体的相关信息；
N_TAGS()	返回结构体中的成员个数；
TAG_NAMES()	返回结构体成员的名字；

```
IDL> help, str, /struct
** Structure <1882e3e0>, 4 tags, length=40, data length=38, refs=1:
  NAME          STRING      '    100'
  AGE           INT         18
  HEIGHT        FLOAT       180.000
  CITY          STRING      'guilin'
IDL> n_tags(str)
      4
IDL> tag_names(str)
NAME
AGE
HEIGHT
CITY
```


4 结构体

- 结构体相关函数

- 创建匿名结构体:

```
IDL> str1=create_struct('name','tom','age',18)
IDL> help,str1
** Structure <1702ed70>, 2 tags, length=24, data length=18, refs=1:
    NAME                STRING      'tom'
    AGE                  INT          18
```

- 创建命名结构体:

```
IDL> str2=create_struct(name='two',str1,'height',180.0)
IDL> help,str2
** Structure TWO, 3 tags, length=24, data length=22:
    NAME                STRING      'tom'
    AGE                  INT          18
    HEIGHT               FLOAT       180.000
```


5 指针

- 指针

- 创建指针时，其数据存储在堆变量中。堆变量在程序运行期间是全局变量，而且只有通过指针名才能访问。
- 堆变量是可以动态分配内存的全局变量。

5 指针

- 指针创建：函数Ptr_New()

- 指针访问：*指针名

- 赋值

- 空指针

```
IDL> pempty=ptr_new(/allocate_heap)
IDL> help,pempty
PEMPTY          POINTER    = <PtrHeapVar6>
IDL> help,* pempty
<PtrHeapVar6>   UNDEFINED = <Undefined>
IDL> print,ptr_valid(pempty)
1
```

```
IDL> a=ptr_new(2.0d)
IDL> help,a
A               POINTER    = <PtrHeapVar1>
IDL> *a
2.000000000000000000
```

```
IDL> b=a
IDL> *b
2.000000000000000000
IDL> *b=5
IDL> *b
5
```

5 指针

- 内存控制
 - 内存分配
 - Ptr_New(data, /No_Copy)
 - 内存释放
 - Ptr_Free, point

```
IDL> data=indgen(20,20)
IDL> pd=ptr_new(data)
IDL> help,data,pd
DATA                INT                = Array[20, 20]
PD                  POINTER            = <PtrHeapVar2>
IDL> help,pd,*pd
PD                  POINTER            = <PtrHeapVar2>
<PtrHeapVar2>      INT                = Array[20, 20]
```

```
IDL> pd=ptr_new(data,/no_copy)
IDL> help,data,pd,*pd
DATA                UNDEFINED          = <Undefined>
PD                  POINTER            = <PtrHeapVar3>
<PtrHeapVar3>      INT                = Array[20, 20]
```

5 指针

●指针相关函数

名称	作用
Ptr_New()	创建新指针;
Ptr_Free	释放指针;
Ptr_Valid	检测指针是否存在;
PtrArr	建立指针数组;

6 对象

- 对象是数据（属性）和程序（方法）封装在一起的实体。
- 对象的功能操作或接收外界信息后的处理操作称为对象方法。

6 对象

- 创建对象

- Obj_new() 函数：用来创建某一特定类的对象

- 调用格式：

- Result=Obj_new([ObjClassName [, Arg1...Argn]])

- Objarr() 函数

6 对象

- 调用对象

- 过程方法

```
IDL> var.ToBinary()  
1010
```

- 调用格式:

- Obj.Procedure_Name, Argument, [Optional_Argument]

- 函数方法

- 调用格式:

- Result=Obj.Function_Name(Argument, [Optional_Argument])

7 链表

- 链表是一个复合的数据类型，可以包含变量、数组、结构体、指针、对象、链表或哈希表等数据类型。
- 链表中的元素是**有次序**的，可以通过**索引**进行编辑。

7 链表

- 链表创建:

- Result = **LIST**([Value1, Value2, ... Valuen] [, /EXTRACT] [, LENGTH=value] [, /NO_COPY])

```
IDL> list1=LIST('one', 2.0, 3, 41,  
PTR_NEW(5), {n:6})  
IDL> help,list1  
LIST1          LIST    <ID=32  
NELEMENTS=6>
```

```
IDL> list_ex = list('a',1,ptr_new(5),{n:6})  
IDL> help,list_ex  
LIST_EX        LIST    <ID=23  NELEMENTS=4>
```

7 链表

- 链表访问：

- 链表名[索引] 与数组访问一样，通过下标索引实现

- 链表销毁

- obj_destroy, 链表名

7 链表

●链表方法：链表名. 方法

- Add 添加成员
- Count 成员个数查询
- IsEmpty 是否为空
- Remove 移除成员
- Reverse 逆转顺序
- ToArray 转换为数组
- Where 查询

```
IDL> list_a=list(1,2,3)
IDL> list_a.add, 'abc'
IDL> help, list_a
LIST_A          LIST  <ID=59  NELEMENTS=4>
IDL> print, list_a
      1
      2
      3
abc

IDL> list_a.add, 10, 0
IDL> print, list_a
     10
      1
      2
      3
abc

IDL> list_a.remove, 3
IDL> print, list_a
      1
      2
      3
```

8 哈希表

- 哈希表（Hash）类型是一个高效的复合数据类型，可以包含变量、数组、结构体、指针、对象、链表或哈希表等数据类型。
- 哈希表的特点是**关键字（Keys）**和**值对应**，并可以通过关键字快速访问，通过哈希表函数进行处理。

8 哈希表

- 哈希表创建:

- `Result = HASH([Key1, Value1, Key2, Value2, ...
Keyn, Valuen] [, /EXTRACT] [, /NO_COPY])`

```
IDL>  
ha=hash('name','tony','age',21,'city','guilin')  
IDL> help, ha  
HA          HASH  <ID=42  NELEMENTS=3>  
IDL> print, ha  
city: guilin  
name: tony  
age:      21
```

- 哈希表访问:

- `result=哈希表名[关键词]`

```
IDL> ha['name']  
tony  
IDL> ha['age']  
21
```

8 哈希表

- 成员增加:

- 表名[关键词]=数值

- 或 表名+= HASH(关键词, 数值)

```
IDL> ha['work']='student'  
IDL> print, ha  
city: guilin  
name: tony  
work: student  
age: 21
```

- 哈希表销毁:

- obj_destroy, 表名

8 哈希表

● 哈希表方法：

- Count 成员个数
- HasKey 是否含有成员
- IsEmpty 是否为空
- Keys 成员名
- Remove 移除成员
- ToStruct 转换为结构体
- Values 成员值
- Where 查询

9 运算符

●数学运算符

- 加(+)
- 增运算(++)
- 减(-)
- 减运算(--)
- 乘(*)
- 除(/)
- 幂(^)

```
IDL> var=20
IDL> print, var++
      20
IDL> print, var++
      21
IDL> print, var++
      22
IDL> print, var++
      23
```

```
IDL> print, var--
      23
IDL> print, var--
      22
IDL> print, var--
      21
IDL> print, var--
      20
```

●数学运算符

- 取余mod
- 取大 >
- 取小 <

```
IDL> print, 32 mod 4
      0
IDL> print, 32 mod 5
      2
IDL> print, 32 > 29
      32
IDL> print, 32 < 29
      29
```


9 运算符

●逻辑运算符

如果表达式为真或非零, 则返回 1, 否则返回 0

●与运算(&&)

●或运算(||)

●非运算(~)

```
IDL> PRINT, 5 && 2
```

1

```
IDL> PRINT, 5 && 0
```

0

```
IDL> PRINT, "sd" && "d"
```

1

```
IDL> PRINT, "sd" && ""
```

0

```
IDL> PRINT, 5 || 2
```

1

```
IDL> PRINT, 5 || 0
```

1

```
IDL> PRINT, 0 || 0
```

0

```
IDL> print,~3
```

0

```
IDL> print,~0
```

1

9 运算符

●位运算

- 位加 (AND)
- 位取反 (NOT)
- 位或 (OR)
- 位与或 (XOR)

IDL> print,5 AND 6

4

0101 ← 5

0110 ← 6

0100 ← 4

9 运算符

- 关系运算符： 返回值是‘真’为1，‘假’为0

- EQ 等于

```
IDL> print,2 EQ 2.0
```

1

- NE 不等于

```
IDL> print,2 NE 2.0
```

0

- GT 大于

```
IDL> print,2 NE 1
```

1

- LE 小于等于

```
IDL> print,2 GE 1
```

1

- LT 小于

```
IDL> print,2 GE 12
```

0

10 其他运算符

- 小括号 `()`：运算优先级
- 中括号 `[]`：数组索引
- 条件表达式 `?:` IDL> `print, (4 gt 3)?1:0`
1
- 对象方法调用符 `->`
- 指针引用符 `*`

控制语句

- 1. 循环语句
- 2. 条件语句
- 3. 跳转语句

1 循环语句

- FOR语句
- WHILE语句
- REPEAT语句
- FOREACH语句

1 循环语句

●FOR语句

●方式一: **for** v1,v2 **do** 语句

●方式二: **for** v1,v2 **do begin**
语句
endfor

```
1 PRO test_for
2   n=10
3   FOR i=0,n-1 DO BEGIN
4     val=i+1
5     print,val
6   ENDFOR
7
8 END
```

1 循环语句

●WHILE语句

●方式一： **while** 条件 **do** 语句

●方式二： **while** 条件 **do begin**

语句

endwhile

```
i=0  
n=10  
while i le n-1 do begin  
    val=i+1  
    i++  
    print, val  
endwhile  
print, 'endwhile'
```


1 循环语句

●REPEAT语句

●方式一： repeat 语句 until 条件

●方式二： repeat begin

语句

endrep until 条件

```
i=0
repeat begin
    val=i+1
    i++
    print, val
endrep until (i ge 10)
print, 'endrep'
```

2 条件语句

●IF语句

●方式一：if 条件 then 语句

●方式二：if 条件 then begin

语句

endif

```
1 PRO test_if
2   var=10
3   IF var EQ 10 THEN print, 'ture'
4   IF var EQ 10 THEN print, 'yes' ELSE print, 'no'
5   IF var EQ 10 THEN BEGIN
6     var=var*2
7     print, var
8   ENDIF
9   IF var EQ 10 THEN BEGIN
10    var=var*2
11    print, var
12  ENDIF ELSE BEGIN
13    var=0
14    print, var
15  ENDELSE
```

2 条件语句

●CASE语句

- 方式：
 - case 表达式 of
 - 情况 1: 语句
 - 情况 2: 语句
 - 情况 3: begin
语句
end
 - else: 语句
 - endcase

```
PRO test_case
val=5
case val of
  0:print,'black'
  1:print,'white'
  2:print,'gray'
  else:print,'color'
endcase
END
```

```
1 PRO test_case
2 val=3
3 case val of
4   0:print,'black'
5   1:print,'white'
6   2:print,'gray'
7   3:begin
8     print,'three'
9   end
10  else:print,'color'
11 endcase
12
13 END
```

2 条件语句

● SWITCH语句

- 方式： `switch` 表达式 `of`
 情况 1: 语句
 情况 2: 语句
 情况 3: 语句
 `endswitch`

```
num=3
switch num of
    1:print, 'one'
    2:print, 'two'
    3:print, 'three'
    4:print, 'four'
endswitch
print, 'end'
```

遇到条件一致的情况时，依次**执行**
后面各个情况直至**endswitch**

3 跳转语句

●BREAK语句

- break 提供了一个从循环中（FOR，WHILE，REPEAT）或CASE、SWITCH 等状态中快速退出的方法。

```
n=10
FOR i=0,n-1 DO BEGIN
    val=i+1
    IF val EQ 5 THEN BREAK
    print, val
ENDFOR
print, 'endbreak'
```

3 跳转语句

- CONTINUE语句

- continue 提供了一个从循环中（FOR, WHILE, 和 REPEAT）中进入下一步循环的方法。

```
n=10
FOR i=0,n-1 DO BEGIN
    val=i+1
    IF val EQ 5 THEN CONTINUE
    print, val
ENDFOR
print, 'endcon'
```

错误检测

- Catch语句
- (5.6节P76)