배열의 인덱스(Index)

IDL

배열의 각 요소를 가리키는 인덱스

거창하게 얘기할 필요 없습니다. 배열 A가 있고 이 중 3번에 들어 있는 값을 꺼내려면 A[3] 이라고 씁니다. 배열 번호가 0으로 시작한다는 것만 기억해 두세요. 즉, A[3]은 네 번째 값을 꺼내는 것입니다. 이럴 때 배열 번호 3을 인덱스(첨자)라고 합니다.

인덱스로 배열을 사용하면?

배열기반 언어로서 IDL의 특징은 인덱스로 배열을 사용할 때 두드러집니다.

```
IDL> a=[3.5, 4.7, 9.8, 2.2, 7.4]
IDL> print, a[[1, 0, 3, 2]]
       4.70000       3.50000       2.20000       9.80000
```

인덱스로 배열 [1,0,3,2]를 사용했고 예상대로 a[1], a[0], a[3], a[2] 의 순서로 값을 뽑아냈습니다. 인덱스가 꼭 1차원 배열일 필요는 없습니다.

```
IDL> index=[ [2, 1, 4], $
             [0, 3, 2] ]
IDL> print, a[index]
       9.80000       4.70000       7.40000
       3.50000       2.20000       9.80000
```

다음과 같이 정리할 수 있겠습니다.

1. 인덱스로 배열을 사용하는 것이 가능하다.
2. 출력되는 결과는 인덱스로 사용한 배열과 같은 모양(크기)이다.

인덱스로 배열을 사용하는 것은 반복문을 사용하지 않아 구문이 간결해질 뿐 아니라, IDL의 처리 속도 향상에도 직결되는 매우 중요한 기법입니다. 흔히 사용하는 WHERE 함수, SORT 함수 모두 이 기법을 적극 활용하고 있는 것을 보세요.

예제. SORT 함수의 결과

```
IDL> a=[3.5, 4.7, 9.8, 2.2, 7.4]
IDL> sort_index=sort(a)
IDL> print, sort_index
           3           0           1           4           2
```

SORT 함수는 정렬된 배열의 값을 바로 리턴하지 않습니다. 배열의 인덱스를 리턴합니다. 배열값이 가장 작은 인덱스부터 배열값이 가장 큰 인덱스까지 차례로 리턴하게 됩니다. 3, 0, 1, 4, 2가 그런 순서입니다. 인덱스가 있으니 원래 배열에서 순서대로 값을 뽑아내는 것은 어렵지 않습니다.

```
IDL> print, a[sort_index]
     2.20000       3.50000       4.70000       7.40000       9.80000
```

예제. WHERE 함수의 결과

배열에서 조건을 만족하는 값의 인덱스를 리턴하는 것이 WHERE 함수의 정의입니다.

```
IDL> a=[3.5, 4.7, 9.8, 2.2, 7.4]
IDL> ok=where(a lt 5) 5보다 작은 값들의 인덱스
IDL> print, ok
           0           1           3
IDL> print, a[ok]
       3.50000       4.70000       2.20000
```

예제. 인덱스컬러→ RGB트루컬러 변환

이 기법이 유용하게 활용되는 또 다른 상황은, 인덱스 컬러 영상을 트루컬러 영상으로 변환할 때입니다. 말 그대로 인덱스컬러 영상(배열)은 화소의 값이 인덱스입니다. 이 인덱스들이 가리키는 컬러를 뽑아 와서 영상을 만드는 작업이라고 할 수 있는데, "인덱스로 배열을 사용하면?"과 일치하는 상황입니다.

한장강의, "인덱스 컬러와 RGB 트루컬러의 상호전환" 편에서 두 번째 페이지의 "4. 인덱스 컬러영상 → RGB 트루 컬러 영상으로 변환" 부분을 보시면 일종의 패턴이라고 할 수 있는 구문이 등장합니다.

```
rgbimg=bytarr(3, xsize, ysize)
rgbimg[0, *, *] = R[kmaimg]
rgbimg[1, *, *] = G[kmaimg]
rgbimg[2, *, *] = B[kmaimg]
```

위 예제에서 kmaimg는 인덱스 이미지, R,G,B는 각 인덱스에 대응하는 컬러들의 테이블(배열)입니다.

다차원 배열의 인덱스

2차원 배열의 인덱스는 [X, Y] 의 형태입니다.

```
IDL> b=findgen(5,7)+0.1      ;연습용 2차원 배열 b
IDL> print, b
       0       1       2       3       4
0    0.1000   1.100   2.100   3.100   4.100
1    5.1000   6.100   7.100   8.100   9.100
2    10.1000  11.100  12.100  13.100  14.100
3    15.1000  16.100  17.100  18.100  19.100
4    20.1000  21.100  22.100  23.100  24.100
5    25.1000  26.100  27.100  28.100  29.100
6    30.1000  31.100  32.100  33.100  34.100
IDL> print, b[3, 2]
       13.1000
```

다차원에서 "인덱스로 배열을 사용"하는 방법은 다음과 같습니다.

```
IDL> print, b[[0, 1, 2], [3, 4, 5]]      ;b[0,3], b[1,4], b[2,5]
       15.1000       21.1000       27.1000
```

범위형 인덱스

배열의 인덱스를 범위로 지정할 수 있습니다.

[From : To : Step]

콜론(:)으로 구분하는 위와 같은 형식으로 사용하며 Step(건너뛰기)은 생략 가능합니다.

```
IDL> print, a[1:3]          ; a[[1,2,3]]과 같습니다.
       4.70000       9.80000       2.20000
```

범위형 인덱스에서 *는 배열의 끝을 의미합니다.

```
IDL> print, a[2:*]
      9.80000      2.20000      7.40000
```

짝수 배열 번호의 값을 추출할 때 다음과 같이 Step을 활용할 수 있습니다.

```
IDL> print, a[0:*:2]        ; a[[0,2,4]] 와 같습니다
      3.50000      9.80000      7.40000
```

마찬가지로, 홀수 배열 번호의 배열 값도 추출됩니다.

```
IDL> print, a[1:*:2]        ; a[[1,3]] 과 같습니다.
      4.70000      2.20000
```

배열의 크기를 예측하지 못할 경우(프로그램 실행 시에 배열의 크기가 결정될 경우) *기호는 매우 유용하게 사용될 것입니다.
범위형 인덱스는 다차원 배열에서도 똑같이 사용합니다. 다음과 같은 패턴은 이미지에서 일부 영역을 추출할 때 흔히 사용됩니다.

```
IDL> print, b[1:3 , 2:5]
      11.1000      12.1000      13.1000
      16.1000      17.1000      18.1000
      21.1000      22.1000      23.1000
      26.1000      27.1000      28.1000
```

2차원 배열의 행이나 열을 추출하는 데도 *를 사용할 수 있습니다.

```
IDL> print, b[*, 3]
      15.100      16.100      17.100      18.100      19.100
```

a[*]와 a의 차이

*가 단독으로 사용되면, 모든 배열요소를 가리킵니다.

```
IDL> print, a[*]            ; 이 경우라면 print, a 와 같습니다.
      3.5000      4.7000      9.8000      2.2000      7.4000
```

위와 같은 경우라면 print, a 로 쓰는 것과 차이가 없습니다. *를 쓰느냐 마느냐의 차이는 배열이 좌변으로 가게 될 때(즉, 배열이 값을 받는 상황) 차이가 납니다.

```
IDL> a[*]=0.5       ;a 배열의 모든 값을 0.5로 치환
IDL> print, a
     0.50000      0.50000      0.50000      0.50000      0.50000
IDL> a=0.5          ;a 변수 새로 선언. a변수는 스칼라 0.5
IDL> print, a
     0.500000
```

음수형 인덱스

IDL 8.0 부터는 인덱스로 음수를 사용할 수 있습니다. 배열의 끝번호부터 카운트합니다.

```
IDL> a=[3.5, 4.7, 9.8, 2.2, 7.4]
IDL> print, a[-1]
      7.40000
```

다음과 같이 배열을 뒤집을 수도 있습니다(배열 뒤집기는 REVERSE 함수를 사용하는 것이 일반적입니다).

```
IDL> print, a[-1:0:-1]
      7.4000      2.2000      9.8000      4.7000      3.5000
```

다차원 배열의 1차원적 방식 접근

다차원 배열도 시스템 내부적으로는 1차원배열처럼 한 줄로 쭉 늘어선 형태의 배열로 저장됩니다. 앞 페이지의 배열 b를 이용하여 예를 들어 보겠습니다.

```
IDL> print, b[3,1]
      8.10000
IDL> print, b[8]
      8.10000
```

2차원 배열 b가 줄 바꿈을 하지 않고 한 줄로 늘어서게 된다면(1차원배열이 된다면), 배열의 8번값은 8.1이 됩니다. 3차원 배열도 모두 마찬가지로 1차원 배열인 양 접근할 수 있습니다. 이 방법은 매우 유용합니다.
어떤 배열에서 값이 평균 이상인 곳을 모두 999로 대치하는 프로그램을 만든다고 생각해 보세요. 다음과 같이 하면 됩니다.

```
IDL> ok=where(b ge mean(b))
IDL> b[ok]=999
IDL> print, b          ;b는 여전히 2차원 배열
IDL> print, ok         ;WHERE함수의 리턴값은 1차원 배열형 인덱스
```

b는 2차원 배열이지만, where함수의 리턴값은 1차원배열 방식의 인덱스를 리턴합니다. 그래서 두 번째 줄과 같이 b[ok] 라고 간단히 인덱스를 쓸 수 있게 됩니다. 만일 where 함수의 리턴값이 검사 대상 배열의 차원에 맞추어 2차원형 인덱스로 주어진다면, 위 예제의 두 번째 줄은 좀 더 복잡해지게 됩니다.
더 큰 문제는, 1차원 배열을 처리하는 프로그램, 2차원 배열을 처리하는 프로그램, 3차원, ...을 처리하는 프로그램을 모두 따로 만들어야 하는 번거로움입니다. 하지만, where 함수는 이러한 리턴값을 1차원 방식으로 통일시켜 버렸습니다. 이것은 프로그래머에게 매우 합리적인 접근 방식입니다.

ARRAY_INDICES

다차원 배열에 대해서 1차원 방식의 인덱스를 사용하는 것은 연산 속도도 더 빠르며 프로그램은 더 간결해지는 효과가 있습니다(가능하면 반드시 이렇게 프로그래밍하세요). 그렇다고는 하지만 상황에 따라, 1차원 배열 스타일의 인덱스를 원래 배열의 차원 형태로 변환해 보는 것이 필요할 때가 있는데, "**배열의 차원과 크기를 알면**" 간단한 산수로 누구나 변환할 수는 있을 것입니다. 하지만 이런 일을 하는 함수가 이미 있으므로 굳이 직접 계산할 필요는 없습니다.

```
IDL> print, array_indices(b, ok)
           2           3       ;2차원배열의 인덱스로는 [2,3]
           3           3       ;2차원배열의 인덱스로는 [3,3]
     ... 이하 줄임...
```

ok 라는 1차원배열 형식의 인덱스가 b 배열에서 사실상 어떤 인덱스인지(위 결과와 같이 (2,3), (3,3) 등) 계산하는 함수입니다. 사실 b 배열이 필요한 것이 아니라 b 배열의 크기만 알면 되므로 다음 사용법도 있습니다.

```
IDL> print, array_indices([5,7], ok, /dimensions)
```