IDL의 배열 생성 관련 팁

Intro

어떤 프로그래밍 언어에서든 마찬가지이겠지만, IDL에서도 배열의 생성 및 처리는 데이터 프로세싱의 기본입니다. 그리고 이와 관련하여 다양한 방법들이 존재합니다. 오늘은 IDL의 배열 생성에 있어서 비교적 덜 알려져 있지만 유용한 팁들을 소개해 보고자 합니다.

순차적 규칙의 값들로 구성된 배열

순차적으로 규칙성을 갖는 값들로 구성된 배열을 생성하는데 있어서 IDL에서는 전통적으로 □INDGEN이라는 이름의 내장함수를 주로 사용해왔습니다. 맨 앞글자인 □ 자리는 데이터 형을 의미하는 글자가 들어갑니다. 예를 들어 0부터 1까지 0.2 간격으로 증가하는 실수 값들로 구성된 배열을 만든다면, 전통적인 방법은 다음과 같습니다.

```
arr = FINDGEN(6)*0.2
```

그런데 IDL 8.3 버전부터는 다음과 같은 방법을 사용해도 동일한 형태의 배열을 만들 수 있게 되었습니다.

```
arr = [0 : 1 : 0.2]
```

이와 같이 배열을 구성할 시작값, 끝값, 간격의 순서로 : 으로 분리해서 [ ] 괄호안에 적어주는 방식의 문법입니다. 물론 이러한 문법은 IDL 8.3 이전 버전의 IDL에서는 지원되지 않는다는 점을 유의해야 합니다.

통일된 값들로 구성되는 배열

배열 내 값들이 하나로 통일된 형태의 배열을 생성하는 방법입니다. 원래 IDL에서 지원되는 □□□ARR이라는 이름의 배열생성 함수들을 사용하여 숫자들로 구성된 배열을 생성하면 배열 내 값들은 모두 0으로 통일됩니다. 그런데 0이 아닌 다른 값으로 통일된 배열을 만들고자 한다면 MAKE_ARRAY라는 함수를 사용하는 것이 좋습니다. 예를 들면 다음과 같습니다.

```
arr = MAKE_ARRAY(4, 3, VALUE=20)
```

이렇게 하면 4x3의 형태를 갖는 2차원의 정수형 배열이 생성되면서, 배열 내 값들이 모두 20으로 통일됩니다. 실제로 배열값들을 출력해보면 다음과 같습니다.

```
      20      20      20      20
      20      20      20      20
      20      20      20      20
```

MAKE_ARRAY 함수로 생성되는 배열의 자료형은 VALUE 키워드에 부여된 값에 따라 결정됩니다. 예를 들어, 만약 20 대신 20.0으로 적으면 실수형 배열로 생성됩니다.

Column 또는 Row 방향으로 통일

표현이 좀 애매할 것 같은데, 예를 들어 2차원 배열을 생성하고자 하는데 가로 방향으로는 [2, 5, 7, 8]로 구성되면서 이러한 형태가 세로 방향으로 6줄이 되는 배열을 생성하는 경우를 가정해봅시다. 이러한 2차원 배열을 직접 생성하려면 다음과 같이 가로 및 세로 방향의 형태를 정의하는 1차원 배열을 각각 생성하고, 이 두 배열에 대하여 행렬연산자 #를 적용하면 됩니다.

```
arr1 = [2, 5, 7, 8]
arr2 = MAKE_ARRAY(6, VALUE=1)
arr = arr1#arr2
HELP, arr
PRINT, arr
```

생성된 배열에 대한 정보 및 배열값들이 출력된 모습은 다음과 같습니다.

```
ARR             LONG      = Array[4, 6]
       2       5       7       8
       2       5       7       8
       2       5       7       8
       2       5       7       8
       2       5       7       8
       2       5       7       8
```

이번에는 세로 방향으로는 [2, 5, 7, 8]로 구성되면서 이러한 형태가 가로 방향으로 6줄이 되는 배열을 생성하는 경우를 가정해봅시다. 문법상으로는 위의 경우와 거의 유사하지만 다음과 같이 연산자가 # 대신 ##이 되어야 합니다. 나머지는 동일합니다.

```
arr = arr1##arr2
```

이렇게 생성된 배열에 대한 정보 및 배열값들이 출력된 모습은 다음과 같습니다.

```
ARR             LONG      = Array[6, 4]
       2       2       2       2       2       2
       5       5       5       5       5       5
       7       7       7       7       7       7
```

```
      8       8       8       8       8       8
```

즉 이와 같이 6x4의 형태를 갖는 배열로 생성됩니다.

배열의 크기 부풀리기

배열 내 값들을 그래도 유지하되 크기만 더 부풀리는 경우입니다. 예를 들면 원본 배열은 [2, 5, 7, 8]과 같이 4개의 값들로 구성되어 있는데, 개수를 12개로 늘리면서 값들 자체는 각각 3개씩으로 늘어난 형태로 만드는 경우입니다. 즉 [2, 2, 2, 5, 5, 5, 7, 7, 7, 8, 8, 8]과 같은 구성이 되도록 만들고자 한다면 어떻게 하면 될까요? 이런 경우에는 CONGRID 함수를 이용하면 됩니다. 정답은 다음과 같습니다.

```
arr0 = [2, 5, 7, 8]
arr = CONGRID(arr0, 12, /CENTER)
```

이렇게 하여 생성된 배열 arr의 값들을 출력해보면 확인이 가능합니다. 만약 4개씩으로 부풀려서 2, 2, 2, 2, 5, 5, 5, 5, ...와 같이 만들고자 한다면 위의 CONGRID 함수 내용에서 12를 16으로 바꿔주면 됩니다. 직접 확인해 보시기 바랍니다. 사실 IDL 유저들에게 CONGRID는 그리 낯선 함수는 아닙니다. 아마도 주로 2차원 이미지 배열의 크기를 변경할 때 많이들 사용하셨을 것 같습니다. 하지만 이런 방식으로도 이용이 가능하다는 점을 참조해 두시면 좋을 것 같습니다.

1차원 배열에 원소 추가

배열에 대하여 원소값을 추가적으로 계속 붙여나가면서 그 크기가 점점 늘어나도록 하는 경우입니다. 예를 들어서 처음에는 [6, 4, 7]로 3개의 값들로만 구성되었던 배열이 있을 때, 값을 추가적으로 붙여나가면서 4개, 5개 이런 식으로 배열 크기를 늘려나가는 것입니다. 이런 식의 처리를 위해서는 다음과 같이 [ ] 괄호 안에 원본 배열과 추가될 값을 함께 적어주면 됩니다.

```
arr = [6, 4, 7]
arr = [arr, 2]
HELP, arr
ARR             INT       = Array[4]
PRINT, arr
       6       4       7       2
arr = [arr, 5]
HELP, arr
ARR             INT       = Array[5]
PRINT, arr
       6       4       7       2       5
```

이와 같이 배열의 크기가 점점 커지는 것을 확인할 수 있습니다. 그리고 심지어는 초기 배열이 null이어도 됩니다. 즉 무에서 유를 창조하는 셈입니다.

```
arr = !null
arr = [arr, 2]
arr = [arr, 5]
arr = [arr, 4]
```

이렇게 하면 arr은 처음에는 아무것도 아닌 null이었지만 결국 [2, 5, 4]의 형태를 갖는 배열로 진화하게 됩니다. 이러한 기법은 반복형 구문에서 활용하면 편리합니다. 즉 처음에는 배열의 크기를 정할 수 없지만 계속 값들을 붙여나가면서 제대로 된 배열로 완성해가야 하는 작업에서 그 진가를 발휘합니다.

2차원 배열에 행 또는 열 추가

앞에, 1차원 배열에 원소를 추가하여 배열을 키우는 방법을 소개했는데, 2차원 배열에도 유사한 방법을 적용할 수 있습니다. 예를 들어, 3x2의 구조를 갖는 2차원 배열을 정의하고, 여기에 [20, 20, 20]과 같은 가로 방향의 행(Row)을 붙여서 3x3의 배열로 만들 수 있습니다.

```
arr = [[7, 6, 3], [9, 11, 8]]
arr = [[arr], [20, 20, 20]]
```

그 결과는 다음과 같은 3x3 배열이 됩니다.

```
      7       6       3
      9      11       8
     20      20      20
```

세로 방향의 열(Column)을 붙일 경우에는 다음과 같이 TRANSPOSE 함수를 이용해야 합니다.

```
arr = [[7, 6, 3], [9, 11, 8]]
arr = [[arr], TRANSPOSE([20, 20])]
```

그 결과는 다음과 같은 4x2 배열이 됩니다.

```
      7       6       3      20
      9      11       8      20
```

이런 개념은 다차원 배열에도 확장이 가능합니다. 예를 들면 3차원 배열에 채널을 추가하는 상황 같은 거죠.

그 외의 기법들

오늘 소개된 내용과 관련된 더 상세한 내용은 아래 링크를 통해서 보실 수 있습니다.
http://blog.daum.net/swrush/546