IDL 일단 시작하기 # 2 : 데이터형과 데이터조직구조

Carl Heiles @ UC Berkeley 천문학과

2005. 1. 16

데이터형(data type)이라는 말은 예를 들자면, 정수형, 실수형 변수, 문자열, 복소수형 등을 의미하는 용어다. 조직구조(organizational structures)라는 말은 스칼라, 벡터, 배열, 구조체를 의미한다. 모든 데이터는 데이터형과 조직구조형을 가진다. 예를 들자면, 문자열의 배열이라든지, 복소수의 벡터와 같은 것이다.

제 1 절 데이터형

이 문서에서 우리는 단지 IDL의 기본적인 데이터형만 다루지만, 부호없는 정수형이라든지, 복소수형 등 여기에 다루지 않는 것들도 존재한다.

1.1 숫자, 비트, 바이트, 워드

우리는 이제 컴퓨터의 내부에서 일이 어떻게 처리되는지 조금 알아야만 될 단계에 이르렀다. 컴퓨터는 숫자나 문자를 어떤 방식으로 기억하는 것일까?

사람은 10의 승수로 자리수를 올려가는 방식으로 수를 다루는데, 이를 십진수라고 한다. 십진법 체계에는 10가지의 숫자(0→9)가 있으며 무언가를 셀 때 이 숫자들로부터 시작한다. 그러다가 한자리 숫자로 최대의 값에 도달했을 때에는 두개의 숫자를 사용하여 그 다음 단계의 연속되는 숫자인 10→99를 생성한다. 이 두 자리 수에서 앞에 나오는 숫자를 f 뒤에 나오는 숫자를 s라고 하면, 이 수가 의미하는 값은 $f \times 10^1 + s$가 된다. 더 많은 자리수에 대해서도 이와 같은 의미를 가진다.

기본적으로 모든 컴퓨터의 정보는 2의 승수로 자리수를 올려가는 이진수의 형태로 저장된다. 이 경우에는 몇 가지의 숫자를 가질까? 0과 1. 두 개 뿐이다. 그러므로 한자리 숫자로 표현할 수 있는 가장 큰 수는 1이다. 두자리 숫자의 범위는 10에서 11이 되고 그 값의 의미는 $f \times 2^1 + s$다. 더 많은 자리수도 이런 식의 의미를 가진다. 하지만, 잠깐 짚고 넘어갈 일이 있다. 2진 체계에서 "숫자(digit)"라고 하는 것은 사실 잘못된 용어다. 이 말은 10진수와 관련된 용어일 뿐이며, 제대로 된 용어는 **비트**(bit)라고 한다. 각각의 이진 "숫자"는 사실 **비트**다. 그러므로 이진수 1001은 4-비트 수다. 1001에 해당하는 10진수는 무엇일까?

편의를 위하여 컴퓨터와 컴퓨터 프로그래머들은 비트를 여덟개 씩 묶어서 취급한다. 이 여덟개의 비트 묶음을 **바이트**(byte)라고 한다. 이진수 11111111을 생각해 보라. 이것이 하나의 바이트가 표현할 수 있는 가장 큰 수다. 이 값은 10진수로 어떻게 되는가?

마지막으로, 컴퓨터는 바이트들을 묶어서 **워드**(word)라는 단위로 취급한다. 가장 오래된 퍼스널 컴퓨터에서는 8-비트(1-바이트)를 1-워드로 취급하였다. 펜티엄 컴퓨터나 스팍 웍스테이션은 32-비트(4-바이트)를 1-워드로 취급한다. 4-바이트의 워드로 저장할 수 있는 가장 큰 수는 무엇인가? 표현할 수 있는 가장 작은 음수는 무엇인가?

다음 부분에서는 IDL(다른 어떤 언어도 마찬가지다) 수 체계의 한계값이 어떻게 정해지는지에 대해 다룰 것이다. 컴퓨터 언어들은 여러가지의 테이터 형을 통해 각각 데이터값의 한계를 지정한다. 지금까지는 이런 세세한 내용은 별로 중요하지 않았다. 하지만 지금부터는... 이 문서는 모든 데이터형에 대해 다룰 계획은 아니다. 복소수(Complex; 그렇다, 복잡한 수다!), 16진수, 8진수 등은 여기서 다루지 않겠지만, 관심이 있다면 쉽게 찾아 볼 수 있을 것이다.

1.2 IDL에서 정수를 다루는 데이터형들

정수 데이터형은 여러분이 예상하는 바와 같은 데이터를 다룬다. IDL에서는 1, 2, 4, 8 바이트의 길이가 다른 네 가지 정수형을 지원한다. 짧은 것일 수록 메모리를 적게 차지하고, 긴 것일 수록 더 큰 수를 표현할 수 있다. 상황에 따라 이 두가지 요건을 적절히 절충하는 것이 기술이다.

1.2.1 1 바이트 : Byte 형

바이트형(byte)는 1바이트의 크기를 가지며 언제나 양의 수이다. 그러므로 이 값의 범위는 0→255가 된다. 이미지는 항상 바이트형으로 화면에 표현된다. 이미지 데이터 자체는 바이트형이 아니더라도, 컴퓨터가 비디오 프로세서에 보내는 데이터는 항상 바이트형이다. 비디오 화면은 큰 메모리와 빠른 처리 속도가 필요하기 때문에 바이트형이 이상적이다. **bindgen**을 이용하면 바이트형 배열을 생성할 수 있고, 바이트형 단일 변수는 **x=3b**와 같이 생성할 수 있다. 계산 중에 바이트형 데이터가 255를 넘어가게 된다면, 최소값으로 순환하게 될 것이다. 예를 들자면, 256은 다시 0으로, 257은 1로 순환하는 그런 식이다.

1.2.2 2 바이트 : Integer와 Unsigend Integer형

2바이트 크기로 항상 양의 값을 가지는 정수형은 **Unsigned Integer형**이라고 한다. 이 데이터형은 $0 \rightarrow 256^2 - 1$, 즉 $0 \rightarrow 65535$의 범위를 다룰 수 있다. **uindgen**으로 배열을 생성한다. Unsigned Integer형은 최대값을 초과했을 때 어떤 식으로 순환하게 될까?

일반적인 경우 여러분들은 음수의 영역을 다룰 수 있는 데이터형이 필요할 것이다. 이 때는 **Integer형**을 사용한다. 2바이트의 integer형이 다룰 수 있는 정수의 가짓수는 $256^2 = 65536$이다. 이 것을 공평하게 음수와 양수가 나누어 써야 되므로 각각 $65536/2 = 32768$개 씩 차지하면 된다. 그런데 0도 빼놓을 수 없는 중요한 정수다. 그래서 양수 또는 음수 중에 하나를 뺏어서 0의 자리로 주어야 하는데, 컴퓨터는 음수의 편을 들어 준다. 양의 값이 표현할 수 있는 크기를 하나 줄여 0에게 준 것이다. 결과적으로 integer형이 표현할 수 있는 정수의 범위는 $-32768 \rightarrow 32767$이 된다. **indgen**을 사용하여 integer형 배열을 생성한다.

Integer형이 최대값을 넘었을 때 순환은 어떻게 될까? **x=5 & y=30000**이라면 **z=x*y**의 결과는 어떻게 될까? 한번 확인해 보기 바란다.

1.2.3 4 바이트 : Long Integer와 Unsigned Long Integer형

이에 관련된 논의는 바로 위의 2 바이트 정수형들의 경우와 똑같다. 다만 256^2이 256^4로 대치될 뿐이다. 그렇다면 이 값의 범위는 어떻게 될까? IDL 온라인 도움말 문서의 "Data types"를 찾아 보라. 배열은 **ulindgen**과 **lindgen**으로 생성한다.

1.2.4 8 바이트 : 64-bit Long Integer와 Unsigned 64-bit Long Integer형

이에 관련된 논의도 2바이트 정수형과 다르지 않다. 다만 256^2을 256^8로 대치하여 생각하면 된다. 이 데이터 형이 다룰 수 있는 수의 범위는 어떨까? IDL 온라인 도움말 문서의 "Data Types"를 찾아 보기 바란다. 배열 생성은 **ul64indgen**과 **l64indgen**을 사용한다.

1.3 IDL에서 실수를 다루는 데이터형들

정수 계열의 데이터형들이 가지는 문제는 말 그대로 정수만 다룰 수 있다는 것이다 - 분수를 다룰 수 없다! 만약에, 정수를 다른 정수로 나누고자 한다면, 그 결과는 분수꼴이 되어야 하는데, 정수 계열의 데이터형은 소수점 아래를 잘라버릴 것이다(예를 들어 $\frac{5}{3}$의 결과가 1이다). 이 문제를 해결하기 위해, 실수 계열의 데이터형에서는 몇개의 비트를 지수(exponent)값을 저장하는 데에 사용한다. 이 지수값은 음수가 될 수도 있고 양수가 될 수도 있다.[1] 정수의 표현 정밀도에서 약간의 손해를 보는 대신 우리는 훨씬 넓은 영역의 수를 다룰 수 있게된다.

1.3.1 4 바이트 : Float형

"부동소수(Floating Point)"는 소수점이 떠 다니는 것이다. Float형에서 지수의 범위는 약 $-38 \rightarrow +38$이고 대략 유효숫자 6자리 정도의 정밀도를 가진다. 배열의 생성은 **findgen**을 사용하고, **x=3.**과 같이 소수점을 포함하는 수를 쓰거나 **x=3e5**와 같이 지수 표현을 써서 단일 변수를 생성할 수 있다.

1.3.2 8 바이트 : Double형

Double형(Double Precesion; 정밀도 두 배)은 Float형과 같은 방식인데 데이터의 크기가 두 배이다. 지수의 범위는 약 $-307 \rightarrow +307$이고 유효숫자 16자리 정도의 정밀도를 지닌다. **bindgen**으로 배열을 생성할 수 있고 **x=3d**나 **x=3d5**와 같이 써서 Double형 단일 변수를 생성할 수 있다.

1.4 IDL의 문자열 데이터

문자열형은 문자들을 저장한다. 여기서 문자는 알파벳과 같은 글자, 기호, 그리고 숫자(물론 문자로서의 숫자다. 계산에 이용될 수 없다) 등이 될 수 있다. 문

[1] 역자 보충: 지수는 소수점의 위치를 의미한다고 생각할 수 있다. 예를 들어 4×10^{-3}이라는 수를 보자면, -3이 지수다. 이를 4.0에서 소수점을 앞쪽으로 세칸 움직이라는 의미로 이해한다면 0.004가 될 것이다. 컴퓨터 내부적으로는 정수와 같은 형태로 숫자를 기억하고 단지 몇개의 비트를 지수 저장소로 이용하여 이 수의 소수점이 어디에 있는지 기억한다. 이런 방식을 소수점이 떠다닌다고 표현한 것이 부동소수(Float)라는 말의 의미다.

자열 상수 **hello**는 다섯개의 문자로 이루어져있다. 이 데이터를 저장하는 데에는 한 문자당 한 바이트씩 5 바이트가 필요하다. 하나의 바이트가 표현할 수 있는 가지수는 256가지인데, 2*26개의 알파벳 문자(대문자와 소문자), 10개의 숫자에다가 세미콜론(;)이나 마침표(.) 등의 기호들이 포함된다.[2] **strarr**을 이용하여 배열을 생성하며, **x='Hi there!!!'**와 같이 써서 단일 문자열을 생성한다.

제 2 절 데이터 조직구조

2.1 스칼라; Scalars

스칼라는 단 하나의 데이터다. 예를 들어, 문자열 스칼라는 **joename='joe'**와 같은 것이다.

2.2 벡터; Vectors

벡터란 1차원 배열이다. 예를 들어, 이름 세 개의 요소를 가지는 벡터, **threenames=['joe', 'ivan', 'mark']**와 같은 것이다.

2.3 배열; Arrays

IDL은 8차원 배열까지 지원한다(8개의 첨자를 가진다). 두개의 첨자를 가지는 배열은 수학적으로 행렬로 이해될 수 있으며 또는 연산자나 많은 행렬 처리용 루틴들이 존재한다. IDL의 온라인 도움말에서 **matrices**나 **matrix operators**로 검색해 보라. 벡터나 배열의 생성은 float형의 경우 **fltarr**이나 **findgen**으로 생성하는데 다른 모든 데이터형에 대해서도 이에 상응하는 함수가 존재한다. 배열 데이터를 다루는 데 있어서 여러분이 편리한 방법을 사용하는 것은 좋지만, 반복문은 가능한 피하는 것이 좋다. 대신 **where**함수나 * 연산자 등을 적절히 활용하라.

IDL은 배열을 다루는 데 매우 유연한 방법들을 제공하고, 이런 유연성이 IDL을 이용할 때 누릴 수 있는 강점이다. 예를 들어 **a=findgen(100,100)**과 같은 2차원 배열을 생각해 보자. 그러면 다음과 같은 문장으로 2차원 배열 안의 지정된 일부 영역을 추출하여 3×3의 2차원 배열 **b**를 만들 수 있다.

```
b=a[23:25, 67:69]
```

다음과 같은 문장의 조합을 이용하면, 배열 **a**의 요소 중에서 10보다 큰 것들을 추려낸 1차원 배열 **b**를 생성할 수 있다.

```
indx=where(a gt 10.)
b=a[indx]
```

다음 문장의 조합은 배열 **a**에서 10보다 크고 100보다 작거나 같은 요소를 뽑아 1차원 배열 **b**를 만드는 예인데, 배열 요소의 접근을 인덱스의 배열로 할 수도 있음을 보여 주는 예이다.[3]

```
indx=where(a gt 10.)
jndx=where(a[indx] le 100.)
b=a[indx[jndx]]
```

[2] 역자 보충 : 백스페이스, 탭, 줄바꿈, 경고음 등의 특수문자도 있다

[3] 역자 보충 :이는 단지 저자가 인덱스의 배열을 다른 배열의 첨자로 이용할 수 있음을 이해시키기 위해 사용한 예다. 실제로는 훨씬 간단하고 명확한 방법을 써야 한다.
indx=where(a gt 10. and a le 100.) & b=a[indx]

2.4 행렬; Matrices

행렬은 2차원 배열이다. IDL은 행렬 연산과 처리를 위한 많은 연산자와 다양한 루틴들을 제공하고 있다. 두 개의 행렬을 곱할 때, #연산자를 이용하라. 두 **A**와 **B**의 곱을 **C**라 하면 $C = A\#\#B$ 또는 $C = A\#B$가 된다. 이 둘 중에 어떤 쪽을 이용해야할지는 행렬의 형태가 행-주축인가 열-주축인가에 따라 달라지는데, 약간의 신중을 기해야하는 점이다.

컴퓨터에서 다차원배열을 인덱스하는 방식은 열-주축 형태와 행-주축 형태 두 가지이다. IDL은 행-주축 형태를 사용하는데 이는 Fortran 언어와 같은 방법이다. C 언어 계열에서는 이와 반대로 열-주축 형태를 사용한다. 2×2 행렬 **A**가 있다면, IDL은 행-주축의 형태를 지원하므로 `print, A`로 내용을 확인하면,

$$\begin{bmatrix} A_{0,0} & A_{1,0} \\ A_{0,1} & A_{1,1} \end{bmatrix}$$

의 형태로 출력할 것이다. 이는 열-주축의 형태인 표준적인 행렬의 표기법과는 차이가 있다(표준 행렬 표기법의 전치 형태다). 표준적인 행렬 표기법에 의하면 다음과 같다.[4]

$$\begin{bmatrix} A_{0,0} & A_{0,1} \\ A_{1,0} & A_{1,1} \end{bmatrix}$$

우리는 앞으로 IDL의 출력 결과와 같은 행-주축 형태를 따를 것이다. 다시 말해, 우리의 행렬 정의는 표준적인 행렬 정의의 전치 형태이다.

이러한 차이가 중요하게 작용하는 몇가지 행렬 연산자가 있다. 행렬의 곱 뿐만 아니라 **invert**나 **svsol**와 같은 행렬 함수도 마찬가지이다. IDL은 거의 항상, 입력되는 행렬이 행-주축의 형태를 따른다고 가정한다.

만일 여러분이 결벽주의자가 되고자하여, 표준적인 행렬 표기법인 열-주축의 방법을 고수해야겠다면, 그렇게 해도 좋다. 다음 세가지만 지켜라. 첫째, 행렬을 출력할 때는, 다음과 같이 전치된 형태를 출력해야 한다.

```
print, transpose(A)
```

둘째, 모든 IDL의 행렬식에서 ##은 #으로 대체하라. 셋째, IDL의 루틴들이 입력되는 행렬을 어떤 형태로 받아들이는지 확인하라. 기본적으로 IDL은 입력되는 행렬이 행-주축의 형태라고 가정한다.

구체적인 예를 들어 보자. 만일 여러분이 IDL의 고유한 형태인 행-주축의 특성을 따르고 있다면, 즉 표준적 표기법의 전치 형태를 따르고 있다면, 두 행렬의 곱은 다음과 같이 쓰일 수 있다.

$$C = A\#\#B$$

반면에, 여러분이 구지 표준적인 행렬 표기법을 따르고 있다면, 다음과 같이 써야 한다.

$$C = A\#B$$

[4] 역자 보충 : 표준적인 행렬 표기법의 인덱스는 (행,열)의 순서쌍을 따른다. IDL의 배열 인텍스 표기는 (x, y)의 순서쌍을 따르기 때문에 행렬 표기와 차이가 나는 것이다

왜 도대체 IDL은 이런 비표준적인 방법을 고수하는 것일까? 이미지 프로세싱에서는 이쪽이 훨씬 직관적이기 때문이다. 이미지 데이터는 전통적으로 열 단위가 아닌 행 단위로 처리된다(텔레비젼 주사방식도 마찬가지). IDL은 행렬의 연산이 아닌 이미지 프로세싱에 근간을 두고 있다. 어쩌면, 행렬 연산을 함에 있어서 IDL의 전통이 약간 불편하다고 느낄 때도 있을 것이다. 하지만, IDL로 이미지 프로세싱을 할 때에 느끼게 될 편리함은, 이정도 불편에 대에 충분한 보상이 되고도 남을 것이다.

2.5 구조체; Structure

구조체는 여러가지 다른 종류의 데이터형이 연관되어 있는 데이터를 다둘 때 매우 유용하다. 예를 들어 별들 위치와 적색화량 등의 내용이 포함된 목록을 처리할 경우를 생각해 보자. 여러분은 이 목록의 내용 전체를 구조체의 배열에 담을 수 있다. 이 배열 요소 각각은 별의 이름, 별의 위치, 별의 밝기 등 다양한 물리량을 포함하고 있는 구조체가 될 수 있을 것이다. 구조체는 우리가 데이터 베이스를 구축하는 데 편리한 방법을 제공한다. 이 데이터 베이스에서 필요한 자료만 뽑아내는 것도 명령 입력 창에서 **where** 함수를 사용하면 간단히 처리할 수 있다.

구조체에 대한 자세한 논의는 IDL의 표준 문서인 *Building IDL Applications*의 **9장. Structures** 부분을 참고하면 된다. 여기서는 간단한 예제만을 다루도록 하겠다. 우리의 별 목록을 다룰 구조체를 정의해 보자.

```
A={star, name:'alpha ori', ra:5.3345, dec:-7.6568, $
        reddening:fltarr(12)}
```

이제, **help, /struct, a**를 실행하면 다음과 같은 결과를 볼 수 있을 것이다.

```
** Structure STAR, 4 tags, length=68, data length=68:
   NAME            STRING    'alpha ori'
   RA              FLOAT           5.33450
   DEC             FLOAT          -7.65680
   REDDENING       FLOAT     Array[12]
```

이것이 의미하는 바는 어렵지 않다. 구조체 A는 *star*라는 이름으로 정의되어 있고, 네개의 필드를 가진다. 각각의 필드는 별의 이름(name), 별의 위치(RA, DEC), 그리고 12가지 방법으로 측정된 성간적색화량(reddening)을 의미한다. 여러분은 이제 다음과 같이 12가지 적색화량을 입력할 수 있다.

```
a.reddening=[1.2, 1.4, 1.3, 1.6, 1.3, 1.4, $
             1.3, 1.3, 1.6, 1.3, 1.4, 1.3]
```

이 별의 적위(dec)을 알아내기 위해서는 다음과 같이 입력하면 된다.

```
print, a.dec
```

이 별의 물리량을 수정하기 위해서는 다음과 같은 방법을 이용하면 된다.

```
a.dec = 5.5
a.reddening[3] = 0.7
```

위 예에서 정의된 구조체는 **이름 있는 구조체; named structure**인데, 이것은 한번 정의되면 새로운 필드를 추가하거나 삭제할 수 없는 특성을 가지고 있다. 구조체에는 이 외에도 다음과 같이 **이름 없는 구조체; anonymous structure**가 있다.

```
b={name:'alpha ori', ra:5.3345, dec:-7.6568, reddening:fltarr(12)}
```

이제, **help, b, /str**를 실행하면 다음과 같은 결과를 볼 수 있다.

```
** Structure <3d5d50>, 4 tags, length=68, data length=68, refs=1:
   NAME            STRING    'alpha ori'
   RA              FLOAT           5.33450
   DEC             FLOAT          -7.65680
   REDDENING       FLOAT     Array[12]
```

이처럼 이름 없는 구조체는 구조체가 정의된 후에도 필드를 추가하거나 제거할 수 있는 특성이 있다.

만일 이게 전부라면, 구조체도 우리에게 별 매력이 없어 보인다. 왜냐하면 우리는 585개의 별을 처리해야 하는데, 585개의 구조체를 하나하나 선언하는 것은 편한 점이 하나도 없지 않은가? 그렇지만, 다행스럽게도 구조체는 배열을 만들 수 있다.

```
cataloga=replicate({star}, 585)
; 또는 cataloga=replicate(a, 585)

catalogb=replicate(b, 585)
```

위와 같이 하면, 구조체 **a**와 **b**를 585개 복제하여 배열을 생성한다. 배열과 똑같은 방법으로 구조체의 배열에도 접근할 수 있다. 다음 문장은 구조체 배열의 3번 요소를 **a**와 같게 설정하는 것이다.

```
cataloga[3] = a
```

아니면, 구조체 배열 요소의 특정 필드에도 접근할 수 있다.

```
cataloga[3].name='alpha ori'
```

이제, 3번 별의 이름을 출력하려면 다음과 같이 하면 된다.

```
print, cataloga[3].name
```

일반적인 방법은 아니지만 다음과 같이 써도 옳은 표현이다.

```
print, (cataloga.name)[3]
```

여러분의 실험 데이터를 다룰 때 구조체를 써 보기를 권한다. 구조체가 점점 매력적으로 느껴질 것이다.