제 3 부 — 배열

7장
배열 기초

Rank, Shape, and Bounds

계수 · 형상 · 경계 · 배열 생성자 · 슬라이스 · 열 우선 저장

THREE WORDS

```
scores(5)
   rank 1, shape [5]

grid(3, 4)
   rank 2, shape [3,4]

offset(-2:2)
   bounds -2:2
```

학습 목표

이 장에서 익히는 세 가지

01

배열 선언

계수(rank) · 형상(shape) · 경계(bounds)를 정해 배열을 선언한다.

02

배열 채우기

배열 생성자와 묵시적 do 로 값을 한 번에 채운다.

03

슬라이스 다루기

배열 슬라이스와 인덱싱으로 원하는 부분만 골라 다룬다.

동일한 자료형의 수많은 물리량을 하나의 대표 변수명과 인덱스(첨자)로 묶어 메모리에 연속 배치하는 자료 구조 — 그것이 배열이다.

도입

변수 하나에 값 하나로는 감당할 수 없다

1

연속 함수의 격자화

sin x 를 0 부터 2π 구간까지 200개의 격자점으로 쪼개어 계산하고 그 거동을 선 그래프로 시각화한다.

2

기상 관측 데이터 처리

0.1초마다 고속으로 측정한 초음파 온도 데이터 10,000개를 저장하고 평균 플럭스를 산출한다.

3

2차원 수치 모델링

가로·세로 각각 60칸의 격자 공간 위에서 열전달 방정식을 풀고 재료 내부의 온도 분포를 계산한다.

단일 변수로 처리한다면

t1, t2, t3, … , t1000 처럼 수백 또는 수천 개의 변수 이름을 코드에 일일이 선언해야 하므로 구현 자체가 불가능하다.

```
integer :: t(1000)          real(real64) :: field(60, 60)
```

알아두기

언어마다 배열의 성격이 다르다

언어	인덱스 시작	메모리 저장 순서	연산 방식의 특징
Fortran	1	**열 우선 (column-major)**	배열 전체에 수학 공식을 직접 적용하는 벡터화 연산을 기본 지원한다.
C / C++	0	**행 우선 (row-major)**	배열을 연속된 단일 메모리 블록(포인터)으로 취급한다. 전체 일괄 연산은 지원하지 않아 반복문이 필요하다.
Java	0	**행 우선 (배열 속 배열)**	모든 배열이 독립된 객체다. 실행 중 허용 범위를 넘으면 시스템이 자동으로 오류를 막아 준다.
Python	0	**NumPy 는 행 우선 기본**	대규모 수치 해석에서는 NumPy 로 Fortran 과 유사한 고속 벡터화 연산을 수행한다.
R	1	**열 우선 (Fortran 전통)**	벡터·행렬·다차원 배열이 핵심 타입이며, 루프 없이 배열 전체에 통계 수식을 곧바로 적용한다.

인덱스가 1부터 시작하고 열 우선으로 저장한다 — 이 두 가지가 다른 언어에서 온 사람이 가장 먼저 부딪히는 차이다.

7.1 배열 선언

계수 · 형상 · 경계, 세 가지 용어

계수

rank

차원의 수. 1차원이면 계수 1, 2차원이면 계수 2. Fortran 에서는 최대 15까지 허용된다.

형상

shape

각 차원의 길이를 모아 나타낸 것. (3, 4) 배열의 형상은 [3, 4] 이다.

경계

bounds

각 차원에서 인덱스가 가질 수 있는 최솟값(하한)과 최댓값(상한). 하한 기본값은 1 이다.

```
scores(5)      ! rank 1, shape [5],    bounds 1:5,      size 5
grid(3, 4)     ! rank 2, shape [3, 4], bounds 1:3,1:4,  size 12
offset(-2:2)   ! rank 1, shape [5],    bounds -2:2,     size 5
```

범위(extent) 상한 − 하한 + 1

크기(size) 모든 차원 범위의 곱

7.1 선언과 조회

두 가지 선언 방식, 다섯 개의 조회 함수

속성 방식

```
type, dimension(bounds-list) :: name-list
```

여러 변수에 동일한 크기와 차원을 한 번에 부여할 때 효율적이다.

직접 선언 방식

```
type :: name(bounds-list)
```

변수마다 다른 형상을 부여할 때 코드가 간결해진다. bounds-list는 [lower:]upper 형태이며 lower 를 생략하면 1이다.

배열 조회 내장 함수

`size(a)`	배열 a 의 전체 요소 개수를 반환한다.
`size(a, dim)`	dim 번째 차원의 크기(요소 수)를 반환한다.
`shape(a)`	배열 a 의 형상을 담은 1차원 배열을 반환한다.
`lbound(a, dim)`	dim 번째 차원의 하한 인덱스를 반환한다.
`ubound(a, dim)`	dim 번째 차원의 상한 인덱스를 반환한다.

7.1 [예제]

계수 · 형상 · 경계를 직접 확인하기

```
integer :: scores(5)         ! rank 1, shape [5], bounds 1:5
integer :: grid(3, 4)        ! rank 2, shape [3, 4]
integer :: offset(-2:2)      ! explicit bounds: -2 to 2

do i = -2, 2
   offset(i) = i * i
end do

print '(a, i0)', "size(grid) = ", size(grid)
```

```
size(scores) = 5
size(grid)   = 12
offset bounds: -2 .. 2
offset(-2) = 4         offset( 2) = 4
```

2차원의 전체 크기

size 는 배열 전체의 원소 개수를 돌려준다. grid(3, 4) 는 각 차원 형상의 곱인 3 × 4 = 12 다.

임의의 경계 설정

offset(-2:2) 는 하한을 -2, 상한을 2로 명시했으므로 offset(-2) 부터 offset(2) 까지 총 5개의 공간이 할당된다.

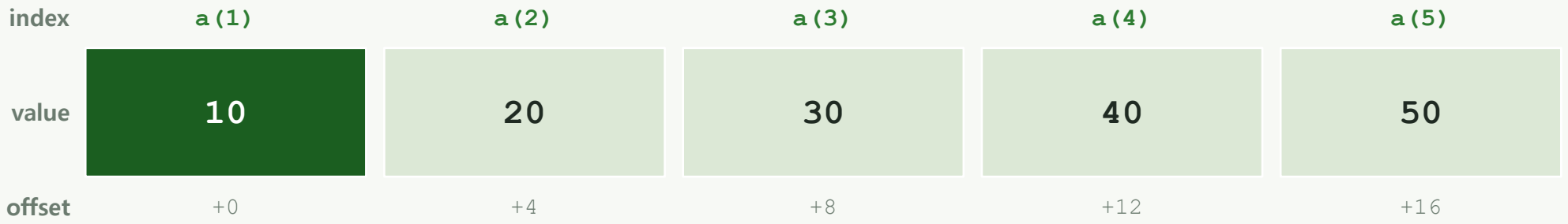
음수 인덱스의 활용

중심 격자를 기준으로 한 사분면 좌표계나, 시계열의 과거 시점을 참조하는 시간 지연 모델에서 물리적 직관과 인덱스를 일치시킬 수 있다.

7.1 알아두기

1차원 배열의 물리적 메모리 구조

integer :: a(5) 에 [10, 20, 30, 40, 50] 을 대입하면



하한을 바꿔도 메모리는 그대로

하한 인덱스를 변경해도 실제 메모리의 물리적 배치와 데이터의 연속성은 변하지 않는다. 각 칸을 가리키는 인덱스의 주소 매핑만 달라질 뿐이다.

[흔한 실수] 첫 요소는 a(1) 이다

C나 파이썬 습관으로 scores(0) 에 첫 값을 넣으면 안 된다. integer :: scores(5) 의 유효 첨자는 1부터 5까지다.

scores(0) 은 하한을 벗어난 접근이다 — 컴파일은 통과하지만 실행 중에 의도치 않은 메모리의 값을 불러온다.

7.2 배열 생성자

do 반복문 없이 한 번에 채운다

일반 do 로 채우면

```
integer :: arr1(3), arr2(5), arr3(3)

do i = 1, 3
   arr1(i) = i
end do
do i = 1, 5
   arr2(i) = i
end do
do i = 1, 3
   arr3(i) = i * 2
end do
```

배열 생성자로 채우면

```
integer :: arr1(3) = [1, 2, 3]

integer :: arr2(5) = [(i, i = 1, 5)]

integer :: arr3(3) = [(i*2, i = 1, 3)]
```

직접 나열

```
[ value1, value2, … ]
```

묵시적 do

```
[ ( expr, var = start, end [, step] ) ]
```

형 일관성

생성자 내의 모든 값은 반드시 같은 자료형과 종류여야 한다.

크기 일치

생성 결과의 크기는 나열된 값의 개수와 같다. 대입 시 좌변 배열의 크기와 반드시 일치해야 한다.

대괄호 […] 로 표기한다. 레거시의 (/ … /) 표기도 같은 의미지만, 현대 코드에서는 대괄호를 쓴다.

7.2 [예제]

배열을 채우는 세 가지 방식

```
! 값을 직접 나열
primes  = [2, 3, 5, 7, 11]

! 묵시적 do : i * i for i = 1..6
squares = [(i * i, i = 1, 6)]

! 실수 변환을 포함한 묵시적 do
ramp    = [(real(i, real64) * 0.5_real64, i = 1, 10)]
```

```
primes:  2 3 5 7 11
squares: 1 4 9 16 25 36
ramp:    0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
print '(a, *(i0, 1x))', "primes:  ", primes
```

전체 배열 대입

루프 없이 배열명에 데이터를 통째로 넘긴다. 좌변의 형상과 우변 생성자의 형상이 일치하면 원소가 순서대로 일대일 매칭되어 한꺼번에 대입된다.

묵시적 do 의 적용

제어 변수가 범위를 돌며 식을 평가하고, 그 결과값들이 차례대로 배열에 담긴다.

무한 반복 서식 *(...)

출력할 배열의 길이에 맞춰 괄호 안의 편집 기술자를 컴파일러가 자동으로 반복해 준다. 배열 크기를 몰라도 전체를 출력할 수 있다.

7.2 [예제]

중첩 묵시적 do — 안쪽 첨자가 먼저 변한다

```
integer :: v(6)

v = [((i + j, i = 1, 3), j = 1, 2)]
```

생성 과정

`j = 1`	안쪽 i 가 1 → 2 → 3 으로 변하며	→ `2, 3, 4`
`j = 2`	안쪽 i 가 1 → 2 → 3 으로 변하며	→ `3, 4, 5`

`v =`

2	3	4	3	4	5

결과는 1차원으로 평탄화된다

중첩 루프 구조를 거치더라도 최종 결과물은 다차원이 아니라 1차원 배열로 평탄화(flattening)되어 차례대로 메모리에 펼쳐진다.

열 우선 규칙과 같다

안쪽 첨자 i 가 바깥쪽 첨자 j 보다 더 빨리 변하며 데이터를 나열하는 것은, Fortran 의 열 우선(column-major) 저장 규칙과 정확히 일치한다.

7.2 [흔한 실수]

하나의 배열에 정수와 실수를 섞을 수 없다

```
a = [1, 2, 3.0]
```

```
Error: Element in INTEGER(4) array
  constructor at (1) is REAL(4)
```

Fortran 의 기본 배열은 태생적으로 동일한 자료형만 담도록 설계되어 있다. 정수 배열을 원하면 [1, 2, 3], 실수 배열을 원하면 [1.0_real64, 2.0_real64, 3.0_real64] 처럼 종류를 통일한다.

굳이 섞어야 한다면, 두 가지 길이 있다

1 형변환 함수로 강제 통일

```
real :: a(3)
a = [real(1), real(2), 3.0]
```

내장 함수 real() 로 정수를 실수로 명시적으로 변환한 뒤 대입한다.

2 파생 타입으로 그룹화

```
type :: record
   integer :: id
   real    :: value
end type record
```

수치 계산용 행렬이 아니라 서로 다른 속성의 데이터를 한 묶음으로 관리하고 싶을 때 쓴다.

start : end [: stride]

`a(3:7)`	3번째부터 7번째까지 선택한다. 양 끝 포함, 총 5개 요소.
`a(1:10:2)`	1부터 10까지 보폭 2만큼 건너뛴다 (1, 3, 5, 7, 9).
`a(10:1:-1)`	10부터 1까지 거꾸로 선택한다 (보폭 -1).
`a(:)`	해당 차원의 전체 요소를 선택한다.
`a(5:)` · `a(:5)`	시작이나 끝을 생략하면 그 차원의 경계값이 기본값이 된다.

국제 표준에서는 단면(section)이라 부르지만, 이 책에서는 관용적인 슬라이스(slice)로 쓴다.

대입의 좌변으로도 쓴다

읽기만이 아니라 대입문의 좌변으로도 쓸 수 있다. a(2:4) = [20, 30, 40] 은 2·3·4번째 요소만 바꾼다.

정합성

슬라이스를 이용한 대입 시 좌변 슬라이스의 크기와 우변의 크기가 반드시 같아야 한다.

보폭의 방향

보폭이 양수면 start ≤ end, 음수면 start ≥ end 여야 한다. 위반하면 빈 배열이 반환되거나 오류가 난다.

7.3 [예제]

1차원 슬라이스와 상한 포함 규칙

```
a = [(i, i = 1, 10)]

print '(a, *(i3))', "a(3:7)      : ", a(3:7)
a(2:4) = [20, 30, 40]
```

```
a          :   1  2  3  4  5  6  7  8  9 10
a(3:7)     :   3  4  5  6  7
a(1:10:2)  :   1  3  5  7  9
a(10:1:-1) :  10  9  8  7  6  5  4  3  2  1
after edit :   1 20 30 40  5  6  7  8  9 10
```

[흔한 실수] 파이썬과 다르다

파이썬의 a[3:7] 은 4개(상한 미포함)지만, Fortran 의 a(3:7) 은 3·4·5·6·7 을 모두 포함해 5개다.

슬라이스 크기 공식

```
end - start + 1
```

보폭 1일 때의 원소 개수다. 슬라이스를 좌변에 쓸 때는 우변의 크기를 이 값에 정확히 맞춘다.

슬라이스를 쓰면 루프 없이도 배열의 특정 부분만 제어하거나 데이터를 복사·이동할 수 있다.

7.3 알아두기

왜 열 우선(column-major)인가

메모리는 2차원 평면이 아니라 선형 구조다. 2차원 배열도 실제로는 일렬로 펼쳐 담아야 한다.

m(3, 4) — 수학적 행렬 구조

row 1	11	12	13	14
row 2	21	22	23	24
row 3	31	32	33	34

메모리 순서 — 첫 첨자가 가장 빨리 변한다

선형 인덱스 = `(j − 1) × nrows + (i − 1)`

m(3,4) 는 (4−1)×3 + (3−1) = 11 — 마지막 칸이다. C·NumPy 는 반대로 행 우선이다.

가장 안쪽 반복문의 제어 변수를 첫 첨자 i 로 두면 메모리 접근이 물리적 주소 증가 방향과 일치해 캐시 적중률이 극대화된다.

7.3 [예제]

2차원 인덱싱과 행 · 열 슬라이스

```
! column-major friendly: 안쪽 루프를 첫 첨자로
do j = 1, 4
   do i = 1, 3
      m(i, j) = i * 10 + j
   end do
end do

print '(a, *(i4))', "column 2  m(:,2): ", m(:, 2)
```

```
 matrix (row by row):
   11   12   13   14
   21   22   23   24
   31   32   33   34
 column 2  m(:,2):    12   22   32
 row 1     m(1,:):    11   12   13   14
```

m(i, j)

관례적으로 첫 첨자 i 는 행(row), 둘째 첨자 j 는 열(column)을 가리킨다.

콜론으로 줄을 통째로

특정 차원에 콜론(:)을 두면 그 줄 전체가 1차원 부분 배열이 된다. m(:, 2) 는 2번째 열, m(1, :) 는 1번째 행.

반복 순서가 속도를 가른다

안쪽 루프를 i 로 두면 CPU 가 연속된 주소를 순차 접근한다. 대형 격자 시뮬레이션에서는 이 순서 하나가 실제 연산 속도를 결정한다.

7.3 [예제]

2차원 배열을 csv 행렬로 내보내기

```
real(real64) :: field(nx, ny)      ! nx = ny = 60

do j = 1, ny
   do i = 1, nx
      x = -3.0_real64 + real(i - 1, real64) * dx
      y = -3.0_real64 + real(j - 1, real64) * dy
      field(i, j) = exp(-(x*x + y*y)) * cos(3.0_real64 * x)
   end do
end do

! 한 줄에 행렬 한 행씩 — ":" 가 끝의 쉼표를 막는다
do i = 1, nx
   write(u, '(*(f0.6, :, ","))') field(i, :)
end do
```

행 슬라이스를 그대로 넘긴다

출력 목록에 field(i, :) 를 전달하면 그 행 전체가 한 줄에 기록된다.

콜론(:) 편집 기술자

출력할 원소가 남아 있지 않으면 즉시 서식 적용을 종료한다. 덕분에 줄 끝에 불필요한 쉼표가 남지 않는다.

csv 한 줄이 행렬 한 행이므로, 읽어 들인 data[i, j] 는 Fortran 의 field(i+1, j+1) 에 대응한다.

로 감쇠하는 가우시안 성분 $\exp(-(x^2+y^2))$ 이 곱해진 형태다.

Python 은 np.loadtxt("field.csv", delimiter=",") 로 행렬을 통째로 읽는다.

7.3 [예제]

히트맵으로 본 2차원 격자

$$f(x, y) = \exp(-(x^2 + y^2)) \cdot \cos(3x)$$

두 성분이 겹친 파동 강도 분포

원점으로부터의 거리에 따라 감쇠하는 가우시안 성분과, x 축 방향으로 진동하는 주기적 성분이 곱해져 복합적인 강도 분포를 만든다. 중앙의 진한 봉우리 좌우로 음의 골(붉은 띠)이 나타난다.

왜 data.T 로 전치하는가

Fortran 의 A(i, j) 를 텍스트로 저장한 뒤 파이썬 imshow 로 읽으면, 첫 인덱스 i 를 화면의 수직축(y)으로, 둘째 인덱스 j 를 수평축(x)으로 인식해 그림이 90도 돌아간다. 이 예제는 i 를 x, j 를 y 로 정했으므로 관례대로 x 를 가로에 두려면 전치해야 한다.

배열-그림의 첨자 대응을 의식하는 습관이 2차원 데이터에서 방향 실수를 막아 준다.

7.3 [흔한 실수]

경계를 벗어난 접근은 조용히 지나간다

```
integer :: a(10)

print *, a(11)      ! 범위 밖
```

컴파일은 통과한다

배열 범위 밖을 읽으려 해도 컴파일 단계에서는 넘어간다. 실행 단계에서 엉뚱한 값을 주거나 프로그램이 강제 종료된다.

경계 검사 옵션을 켜면

```
gfortran -std=f2018 -fcheck=bounds bug.f90
```

```
Fortran runtime error: Index '11' of
  dimension 1 of array 'a' above upper bound of 10
```

개발 중에는 항상 켜 둔다

개발 단계에서는 -fcheck=all 을 늘 켜 두고, 검증이 끝난 뒤 실행 성능을 위해 끄고 다시 컴파일한다.

컴파일러가 잡아 주지 않는 오류도 있다 — a(1:3) = b(5:7) 처럼 크기는 우연히 같지만 인덱스를 잘못 지정한 경우다. 형상이 같은지 뿐 아니라 어떤 요소를 가리키는지도 함께 확인해야 한다.

오류 학습

Different shape / Incompatible ranks

형상 불일치 — 개수가 맞지 않는다

```
integer :: a(10)

a = [(i, i = 1, 10)]
a(2:4) = [20, 30]     ! 좌 3개, 우 2개
```

```
Error: Different shape for array assignment
  at (1) on dimension 1 (3 and 2)
```

(3 and 2) 가 곧 원인

좌변 슬라이스는 3개, 우변 생성자는 2개라 매핑할 수 없다는 뜻이다. 슬라이스 요소 수는 4 − 2 + 1 = 3 이므로, a(2:4) = [20, 30, 40] 으로 고친다.

계수 불일치 — 차원이 맞지 않는다

```
integer :: m(3, 4)

m = [(i, i = 1, 12)]
! 1차원 생성자를 2차원 배열에
```

```
Error: Incompatible ranks 2 and 1
  in assignment at (1)
```

값의 개수가 맞아도 안 된다

12개로 개수는 같지만 좌변은 2차원, 우변은 1차원이라 계수가 어긋난다. 1차원 값들을 2차원 모양으로 바꾸려면 reshape 내장 함수를 쓴다 (8장).

두 오류 모두 컴파일 단계에서 잡힌다 — 메시지를 정확히 읽으면 원인이 곧바로 보인다.

7장 정리

요약

선언과 생성

- 선언: integer :: a(5) 또는 integer, dimension(5) :: a. 2차원은 m(3, 4). 하한을 바꾸려면 offset(-2:2).
- 계수는 차원 수, 형상은 차원별 길이, 경계는 하한:상한. 하한 기본값은 1.
- 조회 함수: size · shape · lbound · ubound 로 배열 정보를 묻는다.
- 배열 생성자: [2, 3, 5, 7, 11], 묵시적 do [(i*i, i = 1, 6)], 중첩 [((i+j, i=1,3), j=1,2)].

슬라이스와 저장

- 슬라이스: a(3:7)(양 끝 포함) · a(1:10:2) · a(10:1:-1) · a(:). 좌변으로도 쓸 수 있다.
- 2차원 인덱싱: m(i, j), 행 슬라이스 m(i, :), 열 슬라이스 m(:, j).
- 저장은 열 우선이라, 안쪽 반복을 첫 첨자로 두면 캐시 적중률이 올라 빠르다.
- 자주 하는 실수: 하한 0 가정 · 슬라이스 끝 미포함 착각 · 생성자 자료형 혼합 · 경계 초과.

한 줄 요약: 첨자는 1부터, 슬라이스는 양 끝 포함, 저장은 열 우선 — 개발 중에는 -fcheck=all 을 켜 둔다.

연습 문제

1. 크기 7의 정수 배열을 선언하고 do 반복으로 각 요소에 첨자의 제곱을 넣은 뒤 전부 출력하라.

2. 배정밀도 실수 배열을 생성자로 채우고, size 로 크기를, 세 번째 요소 값을 함께 출력하라.

3. 경계가 0:9 인 정수 배열에 첨자 × 2 를 채우고 lbound 와 ubound 로 하한·상한을 출력하라.

4. 크기 12의 배열을 1~12로 채운 뒤 슬라이스 a(3:9) 와 a(12:1:-2) 를 각각 출력하라.

5. 4 × 3 정수 배열을 m(i, j) = i + j 로 채우고 한 행씩 줄 바꿔 출력하라.

6. 묵시적 do 생성자로 [1, 4, 9, ..., 100] 을 만들어 출력하라.

7. 크기 8의 실수 배열을 생성자로 채우고 do 반복으로 최댓값과 최솟값을 직접 찾아 출력하라.

8. 보폭 −1 슬라이스로 뒤집힌 순서를 출력한 뒤, do 반복으로 원본 배열 자체를 제자리에서 뒤집어라.

9. 5 × 5 정수 배열에서 대각선은 1, 나머지는 0으로 채우고(중첩 반복과 if) 행 단위로 출력하라.

10. y(i) = i × i − 10 × i 를 0~20 에서 계산해 배열에 담고 csv 로 내보낸 뒤 선그래프를 그려라.

다음 장에서는 배열 연산과 내장 함수 — 전체 배열 연산, reshape, where, 축약 함수를 다룬다.