

02강

배열

컴퓨터과학과 정광식 교수

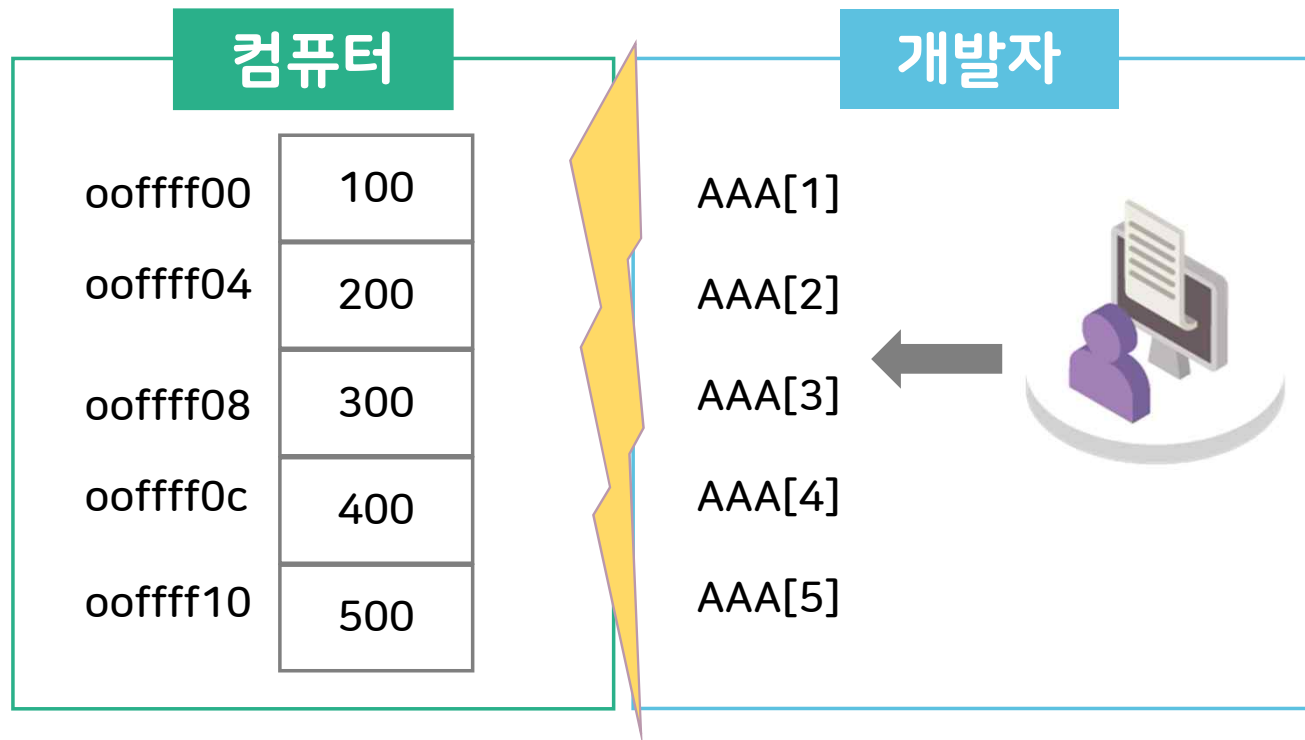
오늘의 학습목차

- 01 배열의 정의
- 02 배열의 추상 자료형
- 03 배열의 연산의 구현
- 04 1차원 배열 및 배열의 확장
- 05 희소행렬의 개념

01

배열의 정의

★ 배열의 정의



◆ 배열

★ 배열의 정의

- ◆ 일정한 차례나 간격에 따라 벌여 놓음 (사전적 정의)
- ◆ '차례'(순서)와 관련된 기본적인 자료구조

1

2

3

4

5

◆ 배열

★ 배열의 정의

- ◆ 인덱스와 원소값(<index, value>)의 쌍으로 구성된 집합

1

2

3

4

5

◆ 배열

★ 배열의 정의

- ◆ 원소의 메모리 공간(메인 메모리, DDR)의 물리적인 위치를 '순서'적으로 결정하는 특징
- ◆ 배열의 순서는 메모리 공간에서 저장되는 '원소값의 물리적 순서'

1

2

3

4

5

◆ 배열

★ 배열의 의미

- ◆ '호수'(인덱스)로 표현되는 **순서**를 갖는 '아파트'(메모리 영역, 원소값을 위한 저장소)
- ◆ 원소들이 모두 **같은 자료형**과 **같은 크기의 기억 공간**을 가짐
- ◆ 배열의 인덱스값을 이용해서 배열의 원소값에 접근하기 때문에 직접 접근이 가능함

◆ 배열

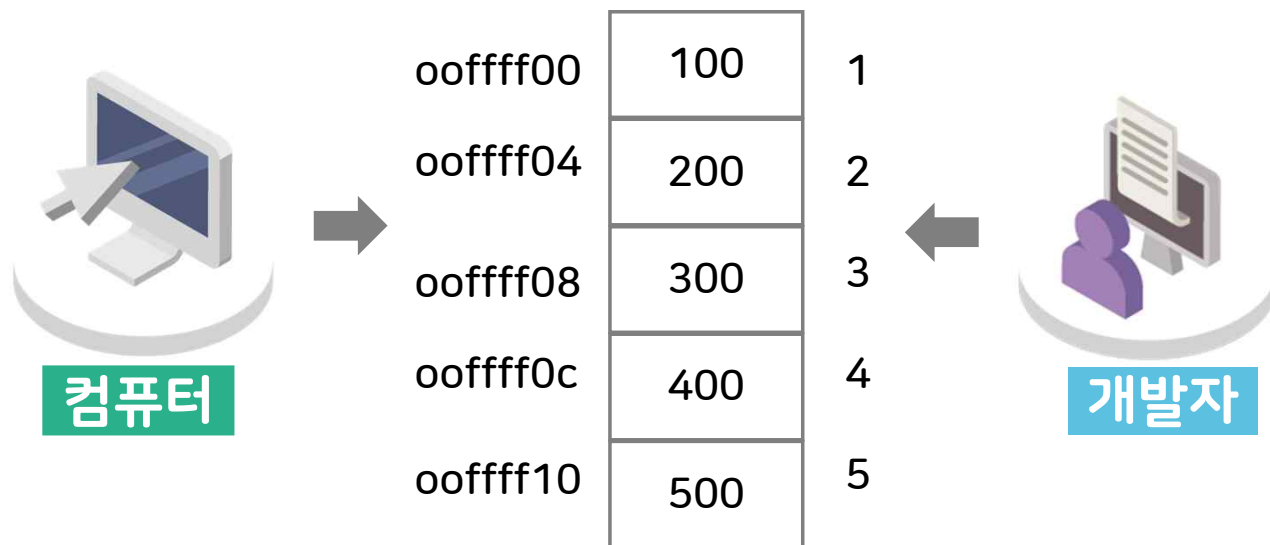
★ 배열의 의미

- ◆ 인덱스값은 추상화된 값 : 컴퓨터의 내부구조나 메모리 주소와 무관하게 개발자에게 개념적으로 정의됨
- ◆ 메모리 주소값은 실제 메모리의 물리적인 위치값(주소값)

◆ 배열

★ 배열의 의미

- ◆ **인덱스**와 **주소값**의 관계(보통 배열의 인덱스는 0부터 시작)



02

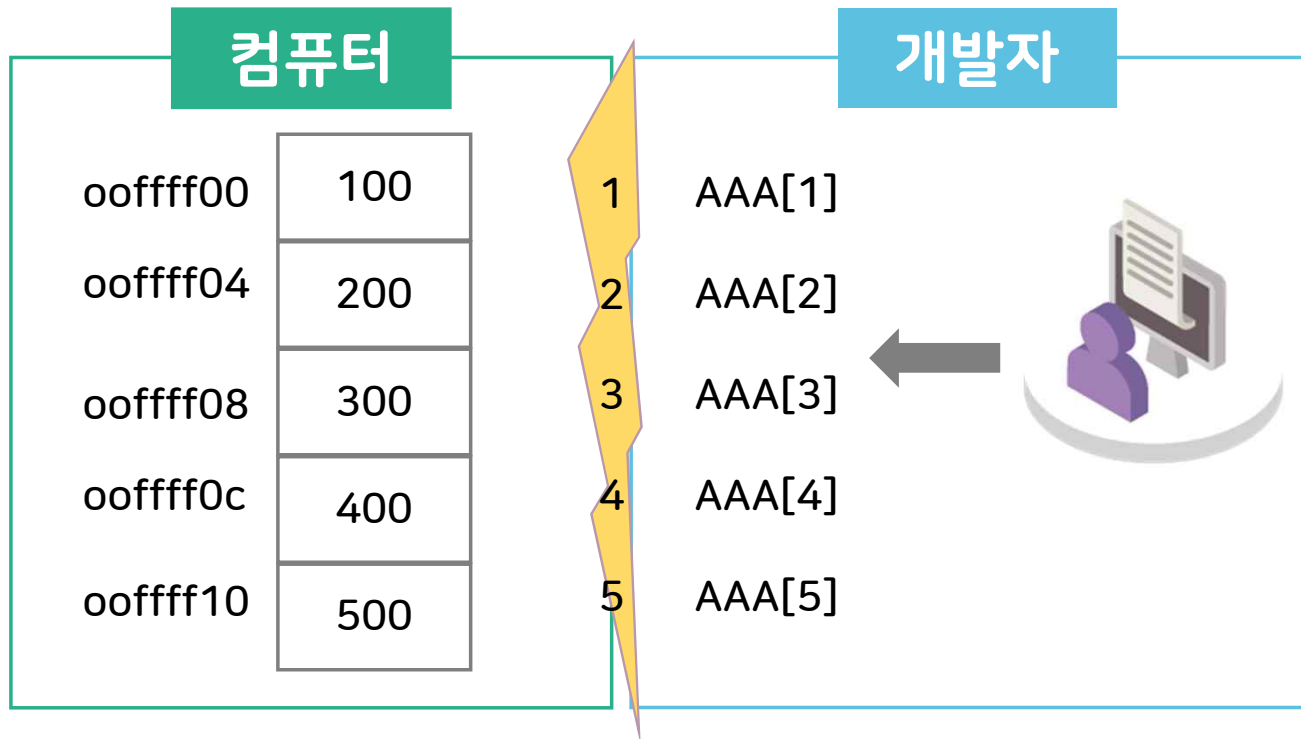
배열의 추상 자료형

◆ 배열의 추상 자료형

◆ 추상자료형 : 객체 및 관련된 연산의 정의

◆ 자료형 : 메모리 저장 할당을 위한 선언

★ 배열의 정의



◆ 배열의 추상 자료형

- ◆ ADT Array 객체 :
 $\langle i \in \text{Index}, e \in \text{Element} \rangle$ 쌍들의 집합
 - Index : 순서를 나타내는 원소의 유한집합
 - Element : 타입이 같은 원소의 집합

◆ 배열의 추상 자료형

- ◆ 연산 : $a \in \text{Array}$; $i \in \text{Index}$; $\text{item} \in \text{Element}$;
 $n \in \text{Integer}$ 인 모든 a , item , n 에 대하여
다음과 같은 연산이 정의됨

- a : 0개 이상의 원소를 갖는 배열
- item : 배열에 저장되는 원소
- n : 배열의 최대 크기를 정의하는 정수값

◆ 배열의 추상 자료형

- ① Array create(n) ::= 배열의 크기가 n 인 빈 배열을 생성하고 배열을 반환한다;
- ② Element retrieve(a, i) ::= if ($i \in \text{Index}$)
 then { 배열의 i 번째에 해당하는 원소값 'e'를
 반환한다; }
 else { 에러 메시지를 반환한다; }

◆ 배열의 추상 자료형

③ Array store(a, i, e) ::= if ($i \in \text{Index}$)

then { 배열 a 의 i 번째 위치에 원소값 ' e '를
저장하고 배열 a 를 반환한다; }

else { 인덱스 i 가 배열 a 의 크기를 벗어나면
에러 메시지를 반환한다; }

03

배열의 연산의 구현

◆ 배열의 연산의 구현

★ 배열의 생성

```
void create(int *a, int n) { // n=5
    int i;
    for(i=0, i<n, i++){
        a[i] = 0;
    }
}
```

1

2

3

4

5

◆ 배열의 연산의 구현

★ 배열의 생성결과

	a[0]	a[1]	a[2]	a[3]	a[4]
a	0	0	0	0	0

1

2

3

4

5

◆ 배열의 연산의 구현

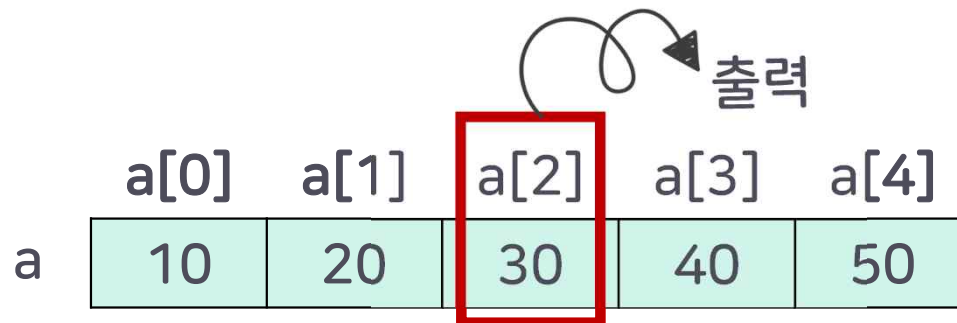
★ 배열값의 검색(retrieve 연산)

```
#define array_size 5
int retrieve(int *a, int i) { // i = 2
    if( i >= 0 && i < array_size )
        return a[i];
    else { printf("Error\n");
           return(-1); }
}
```

◆ 배열의 연산의 구현

★ 배열값의 검색 결과

- ◆ 다음과 같은 원소값이 저장되어있다고 가정하며, '30'이 출력됨



	a[0]	a[1]	a[2]	a[3]	a[4]
a	10	20	30	40	50

◆ 배열의 연산의 구현

★ 배열값의 저장(store 연산)

```
#define array_size 5
void store(int *a, int i, int e) { // i=3, e=35
    if( i >= 0 && i < array_size)
        a[i] = e;
    else printf("Error\n");
}
```


◆ 배열의 연산의 구현

★ a[3]의 값이 35로 변경되어 저장된 모습

	a[0]	a[1]	a[2]	a[3]	a[4]
a	10	20	30	40	50

↓

	a[0]	a[1]	a[2]	a[3]	a[4]
a	10	20	30	35	50

1

2

3

4

5

04

1차원 배열 및 배열의 확장

◆ 1차원 배열 및 배열의 확장

★ 1차원 배열의 정의

- ◆ 한 줄짜리 배열을 의미하며, 하나의 인덱스로 구분됨

1

2

3

4

5

◆ 1차원 배열

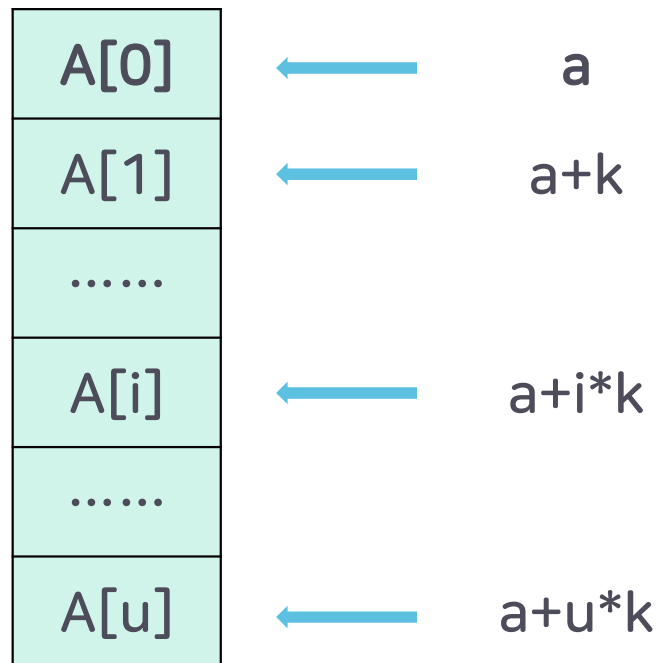
★ 1차원 배열의 정의

- ◆ $A[i]$ 는 배열의 첫 번째 원소 $A[0]$ 이 저장된 주소인 a 로부터 시작하여, $A[0]$ 부터 $A[i-1]$ 개까지 i 개의 배열 $A[]$ 를 지나서 저장됨
- ◆ 따라서, $A[]$ 의 시작주소를 a 라고 가정하면, $A[i]$ 저장 주소는 $[a + i * k]$ 가 됨

$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$
$A(L)$	$A(L+1)$	$A(L+2)$	$A(L+3)$	$A(U)$

◆ 1차원 배열

★ 1차원 배열에서의 주소 계산



◆ 배열의 확장

★ 행렬의 배열 표현

- ◆ 행렬을 컴퓨터에서 표현하기에는 2차원 배열이 적합함

$$\begin{bmatrix} 5 & 2 & 6 & 2 \\ 7 & 2 & 0 & 0 \\ 0 & 1 & 1 & 9 \end{bmatrix}$$

5	2	6	2
7	2	0	0
0	1	1	9

◆ 배열의 확장

★ 행렬의 2차원 배열 표현

열(Column)



행(row)



A[0][0]	A[0][1]	A[0][2]	A[0][m-1]
A[1][0]
A[2][0]
...
A[n-1][0]	A[n-1][m-1]

1

2

3

4

5

◆ 배열의 확장

★ 행 우선 배열

- ◆ 1차원 배열을 여러 개 쌓아 놓은 것이 2차원 배열

A[1]	A[2]	A[3]	A[4]	A[5]	
					B[1]
					B[2]
					B[3]

◆ 배열의 확장

★ 열 우선 배열

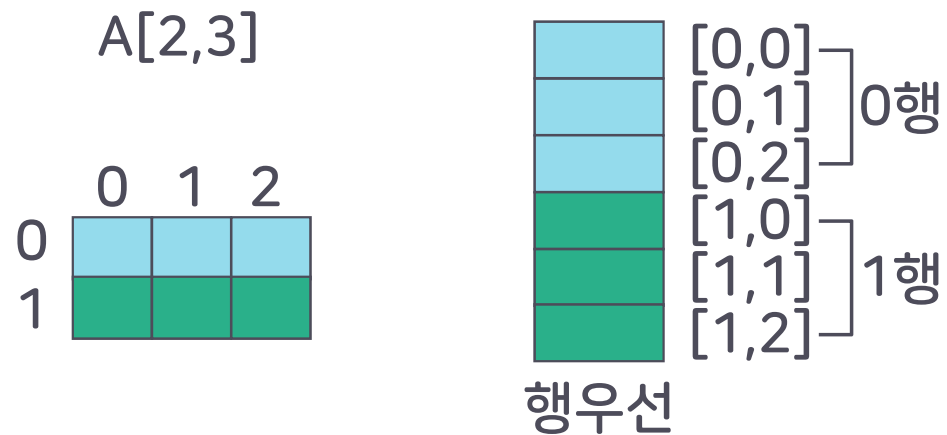
- ◆ 1차원 배열을 여러 개 세워 놓은 것이 2차원 배열

A[1]	A[2]	A[3]	A[4]	A[5]	
					B[1]
					B[2]
					B[3]

◆ 배열의 확장

★ 행 우선 할당

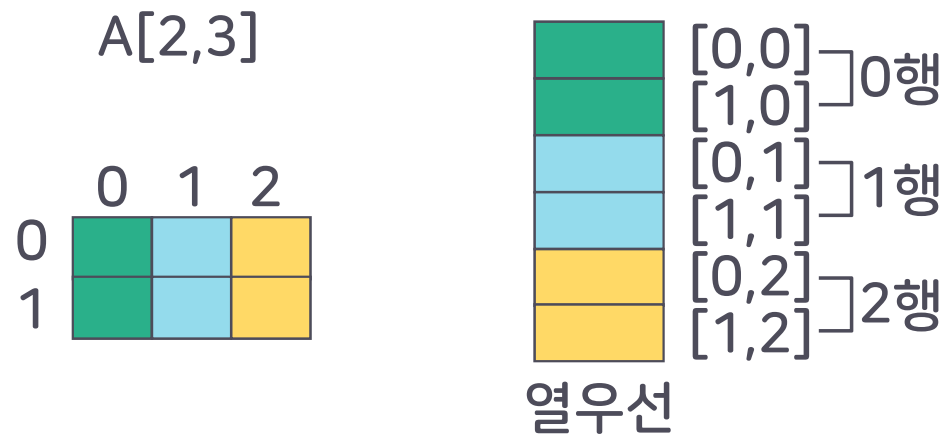
- ◆ 가로로 1차원 배열 단위로 메모리 영역을 우선 할당함



◆ 배열의 확장

★ 열 우선 할당

- ◆ 세로의 1차원 배열 단위로 메모리 영역을 우선 할당함



◆ 배열의 확장

★ C언어 에서의 2차원 배열(행 우선 순서 저장)

- ◆ C 언어에서 A[3][5]을 선언하면 다음과 같은 배열이 생성됨

0, 0	0, 1	0, 2	0, 3	0, 4
1, 0	1, 1	1, 2	1, 3	1, 4
2, 0	2, 1	2, 2	2, 3	2, 4

05

회소 행렬의 개념

◆ 희소행렬

★ 희소행렬

- ◆ 원소값이 0인 원소가 그렇지 않은 원소보다 상대적으로 많음

$$A = \begin{pmatrix} 0 & 20 & 0 & 0 & 9 & 0 & 0 & 11 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 78 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 67 & 0 & 0 & 0 & 0 \\ 0 & 31 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 91 & 0 & 0 & 44 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 19 & 0 & 0 & 27 & 0 \end{pmatrix}$$

◆ 희소행렬

★ 희소행렬의 일반적 배열표현

- ◆ 메모리 낭비를 막고 효율성을 높이기 위해서 0인 원소는 저장하지 않고 0이 아닌 값만을 따로 모아서 저장하는 방법이 필요함

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
[0]	0	20	0	0	9	0	0	11	0
[1]	0	0	0	0	0	0	0	0	0
[2]	78	0	0	0	0	0	0	0	0
[3]	0	0	0	0	67	0	0	0	0
[4]	0	31	0	0	0	0	0	0	0
[5]	0	0	0	91	0	0	44	0	0
[6]	0	0	0	0	0	0	0	0	0
[7]	0	0	0	0	19	0	0	27	0

◆ 희소행렬

★ 희소행렬의 일반적 배열표현

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]		행	열	값
[0]	0	20	0	0	9	0	0	11	0	0	8	9	10
[1]	0	0	0	0	0	0	0	0	0	1	0	1	20
[2]	78	0	0	0	0	0	0	0	0	2	0	4	9
[3]	0	0	0	0	67	0	0	0	0	3	0	7	11
[4]	0	31	0	0	0	0	0	0	0	4	2	0	78
[5]	0	0	0	91	0	0	44	0	0	5	3	4	67
[6]	0	0	0	0	0	0	0	0	0	6	4	1	31
[7]	0	0	0	0	19	0	0	27	0	7	5	3	91
										8	5	6	44
										9	7	4	19
										10	7	7	27

1

2

3

4

5

다음 시간 안내

03강

수고하셨습니다.

스택