2 장. 배열과 구조체

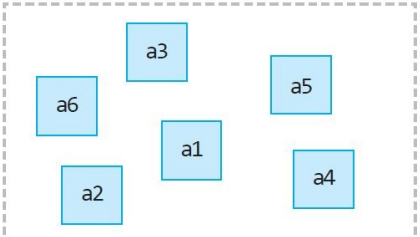
Jinseog Kim

dongguk UNIVERSITY WISE

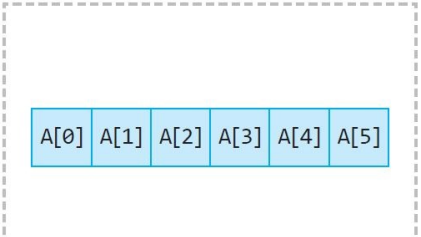
학습목표

1. 자료구조 배열을 이해하고 구현한다.
2. 구조체와 구조체 배열을 이해하고 구현한다.

- 변수와 배열 (Array, Vector)



(a) 6개의 변수 선언

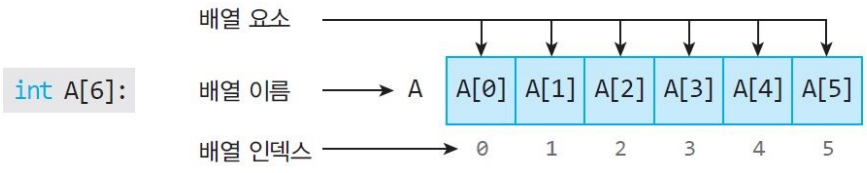


(b) 크기가 6인 하나의 배열 선언

- 같은 형의 변수를 여러 개를 합쳐 하나로 선언하는 경우 배열 (array) 을 사용

배열의 특징

- < 인덱스, 요소 > 쌍의 집합
- 인덱스를 주면 해당하는 값이 대응되는 구조



배열의 장단점

- 배열의 장점
  - 요소에 대한 임의 접근을 허용: 위치별로 요소에 빠른 접근 가능
  - 코드 최적화: 짧은 코드로 저장 및 접근 가능
  - 하나의 이름으로 동일한 유형의 여러 데이터 항목을 표현
  - 다른 데이터 구조 (연결 리스트, 스택, 큐, 트리, 그래프) 를 구현하는 데 사용
- 배열의 단점
  - 고정 크기 배열: 크기를 늘리거나 줄일 수 없음 (데이터 추가가 어려움)
  - 배열에 적은 메모리를 할당하면 데이터가 손실됨
  - 동질적 데이터만 저장 가능: 다른 데이터 유형의 값을 저장할 수 없음
  - 연속된 메모리 위치에 저장: 삭제/삽입이 비효율적임 (시간 복잡도 = O(N)).

예: 학생 성적 처리

```
int a1, a2, a3, a4, a5, a6, a7, a8,
    a9, a10, a11, a12, a13, a14, a15;
...
sum = a1 + a2 + a3 + a4 + a5 + a6
  + a7 + a8 + a9 + a10 + a11 + a12
  + a13 + a14 + a15;
```

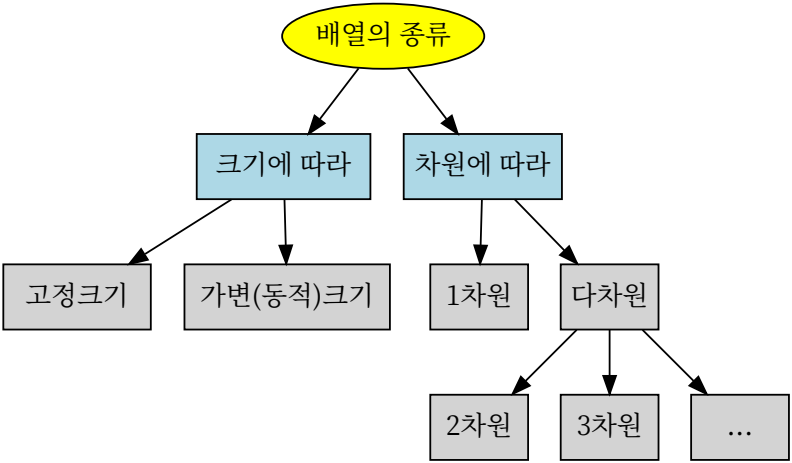
(a) 15개의 변수에 저장된 성적의 합 구하기

```
int A[15];
...
sum = 0
for (int i=0 ; i<15 ; i++)
    sum += A[i];
```

배열에서는 반복문을 사용할 수 있다.

(b) 배열에 저장된 성적의 합 구하기

배열의 종류

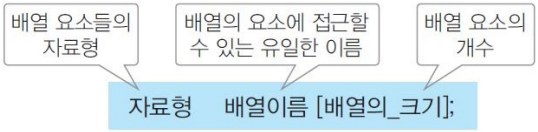


배열 (Array) 의 추상데이터형 (ADT) 예시

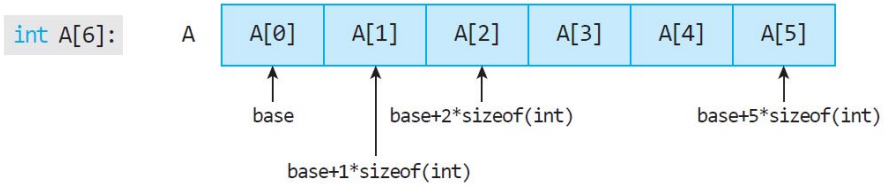
1. 데이터: < 인덱스, 값 > 으로 이루어진 집합, 인덱스는 1 차 또는 그이상의 차원을 가지는 유한 순서 집합
   - 1 차원 인덱스: {0, ..., n-1}
   - 2 차원 인덱스: {(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2))}
2. 연산 또는 함수:
   - Create(j, sizes): j-차원 배열 생성, sizes 는 각 차원별 크기
   - Retrieve(A, i): 배열 A 에서 인덱스 i 의 원소를 반환
   - Store(A, i, x): 배열 A 에서 인덱스 i 에 x 를 저장
   - Dimension(A): 배열 A 에서 차원 (dimension) 을 반환
   - SizeOf(A): 배열 A 에서 차원별 크기를 반환
   - ...

1 차원 배열

- 배열 선언



- 배열 선언 예시
  - 배열의 인덱스는 0 부터 시작
  - A 의 주소 = A[0] 의 주소 (그림에서 base)
  - sizeof(A): A 의 크기를 byte 단위로 계산



1 차원 배열

- 예: 변수와 배열의 크기 확인

```c
#include <stdio.h>
void main()
{
    char c, cA[10];
    int i, iA[10];

    printf("char 형 = %zd  c 의 크기 = %zd\n", sizeof(char), sizeof(c));
    printf("int 형 = %zd  i 의 크기 = %zd\n", sizeof(int), sizeof(i));

    printf("cA 의 크기 = %zd  cA[0] 의 크기 = %zd\n", sizeof(cA), sizeof(cA[0]));
    printf("iA 의 크기 = %zd  iA[0] 의 크기 = %zd\n", sizeof(iA), sizeof(iA[0]));

    printf("cA 요소의 수 = %zd 개\n", sizeof(cA) / sizeof(cA[0]));
    printf("iA 요소의 수 = %zd 개\n", sizeof(iA) / sizeof(iA[0]));
}
```

- 선언하면서 초기화

```
int A[6] = { 1, 2, 3, 4, 5, 6 };
```

```
int A[ ] = { 1, 2, 3, 4, 5, 6 };
```

크기를 생략하면 초깃값을 나열한 수 만큼의 메모리가 할당됨

동일

```
int A[6];
A[0] = 1;
A[1] = 2;
A[2] = 3;
A[3] = 4;
A[4] = 5;
A[5] = 6;
```

배열의 연산 (Operations)

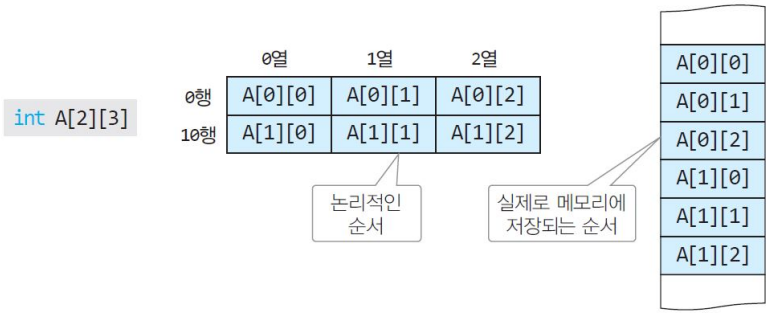
1. 순회 (탐색): 배열의 모든 (또는 특정 원소) 를 탐색
2. 삽입: 특정 위치에 요소를 삽입
3. 삭제: 특정 요소 (또는 특정 위치) 의 요소를 삭제
4. ...

배열을 관리하는 C 함수

- `void *malloc(int n)`: n 개의 정수를 메모리에 할당
- `void free(void *p)`: 주소 p 에 할당된 메모리를 해제
- `void *memmove(void *d, void *s, size_t n)`: 주소 s 에서 주소 d 로 n byte 를 이동
- `void *memcpy(void *d, void *s,size_t n)`: 주소 s 에서 주소 d 로 n byte 를 복사
- `int sizeof()`: 주어진 객체의 크기를 반환 (byte)
- `void *realloc(void *src, int n)`: 배열 src 의 크기를 n 으로 조정

- 2 차원 배열 선언

- 요소들의 위치



- 선언하면서 초기화

```
int A[2][3] = {{1,2,3},{4,5,6}};
```

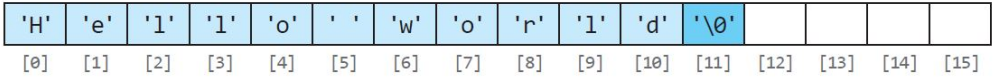
```
int A[2][3] = {1,2,3,4,5,6};
```

첫 행부터 순서대로 요소의
초깃값으로 저장됨

	0열	1열	2열
0행	1	2	3
1행	4	5	6

문자열

- 문자의 배열 문자열의 끝을 나타내는 '\0'(또는 0, NULL) 이 마지막에 추가됨



- 아래 코드의 차이?

```
1  char s1[16] = "Hello World";
2  char s2[] = "Hello World";
3  char s3[16] = { 'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '\0' };
4  char s4[] = { 'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd' };
```

예제코드 테스트 (string_test.c)

```c
#include <stdio.h>
#include <string.h>           // 문자열 복사, 길이 계산 등의 함수 사용을 위해
void main()
{
    char s1[16] = "Hello World";
    char s2[] = "Hello World";
    char s3[16] = { 'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '\0' };
    char s4[] = { 'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd' };

    printf("s1: %s\n", s1);
    printf("s2: %s\n", s2);
    printf("s3: %s\n", s3);
    printf("s4: %s\n", s4);
    printf(" 문자열 s1 의 길이: %zd\n", strlen(s1));
    printf(" 문자열 s2 의 길이: %zd\n", strlen(s2));
    printf(" 문자열 s3 의 길이: %zd\n", strlen(s3));
}
```

배열을 함수로 전달

- 배열 이름 (주소) 가 전달: 배열이름은 배열의 첫 요소의 주소
- 배열의 크기 함께 전달

```c
#include <stdio.h>
void reset_array(int a[], int len) {
    for (int i = 0; i < len; i++) a[i] = 0;
}
void main()
{
  int A[3] = { 10, 20, 30 };
  reset_array(A, 3);      // 길이가 3 인 배열 A 를 0 으로 초기화 -> 성공

  for (int i=0; i < 3 ; i++)
    printf("A[%d]=%d ", i, A[i]);
  printf("\n");
}
```

참고: call-by-value, by-reference, by-address(pointer)

```
#include <stdio.h>
void swap_r(int& x, int& y){
  int z = x;
  x = y;
  y = z;
}

void swap_p(int *x, int *y){
  int z = *x;
  *x = *y;
  *y = z;
}
void swap_v(int x, int y, int *out){
  out[0] = y;
  out[1] = x;
}
```

```
int main()
{
  int a = 45, b = 35;
  int o[2];
  printf("Before Swap\n");
  printf("a = %d, b=%d\n", a, b);
  printf("After Swap\n");
  swap_r(a, b);
  printf("1. reference: a = %d, b=%d\n", a, b);
  swap_p(&a, &b);
  printf("2. pointer: a = %d, b=%d\n", a, b);
  swap_v(a, b, o);
  printf("3. value: o[0] = %d, o[1]=%d\n",
         o[0], o[1]);
}
```

결과

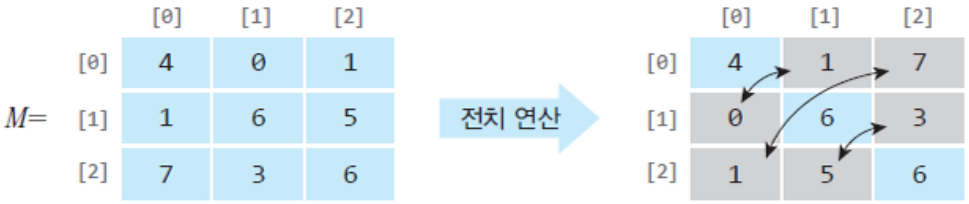
```
Before Swap
a = 45, b=35
After Swap
1. call by reference: a = 35, b=45
2. call by pointer: a = 45, b=35
3. call by-value: o[0] = 35, o[1]=45
```

Lab: 행렬 표현하기

- 2 차원 배열로 행렬 표현 및 전치행렬 (transpose matrix) 구하기

$$M = \begin{pmatrix} 4 & 0 & 1 \\ 1 & 6 & 5 \\ 7 & 3 & 6 \end{pmatrix}, \quad M^T = \begin{pmatrix} 4 & 1 & 7 \\ 0 & 6 & 3 \\ 1 & 5 & 6 \end{pmatrix}$$

- 행렬의 2 차원 배열 표현과 전치행렬 연산



C 코드 구현

```c
#include <stdio.h>
#define ROWS 3
#define COLS 3
void print_mat(int m[ROWS][COLS], char* str)
{
    printf("%s\n", str);
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            printf(" %3d", m[i][j]);
        }
        printf("\n");
    }
}
```

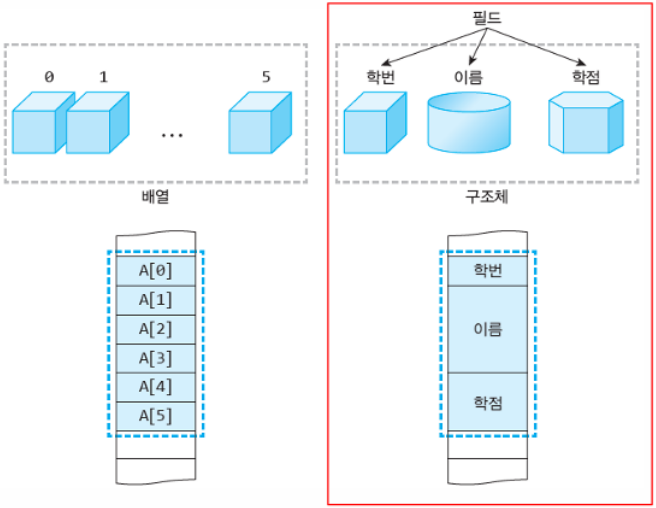
C 코드 구현

```c
void transpose_mat(int m[ROWS][COLS])
{
    for (int i = 0; i < ROWS; i++) {
        for (int j = i+1; j < COLS; j++) {
            int tmp = m[i][j];
            m[i][j] = m[j][i];
            m[j][i] = tmp;
        }
    }
}

void main()
{
    int mat[ROWS][COLS] = { 4, 0, 1, 1, 6, 5, 7, 3, 6 };
    print_mat(mat, " 원래 행렬");
    transpose_mat(mat);
    print_mat(mat, " 전치 행렬");
}
```

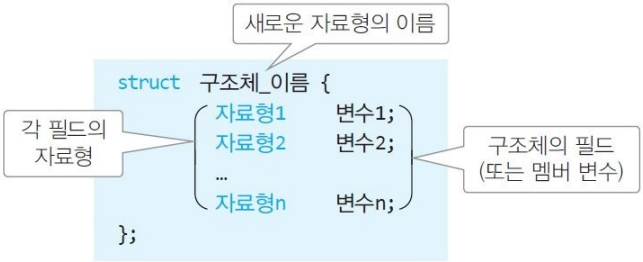
구조체 (structure)

- 구조체 (structure) 는 다양한 자료형의 데이터 모임
- 배열: 같은 자료형의 데이터 모임



구조체의 정의와 선언

- 구조체의 정의: 사용하기 전 반드시 정의해야 함




```c
struct Student{
  int id;
  char name[100];
  double score;
};
struct Student a;
```

```c
// typedef : 구조체에 대한 별칭 정의
typedef struct student_t{
  int id;
  char name[100];
  double score;
}Student;
Student a;
```

구조체의 정의와 선언

- 선언과 동시에 초기화

```
Student a = {20240101, "Jinseog Kim", 99.5};
```

- 멤버 접근: 항목 연산자 (membership operator) “.” 이용

```
a.id = 20200101;
a.name = 90.5;
```

구조체의 연산

- 구조체는기본적으로 대입 연산자 (=) 만 지원

```
int x = 10, y;
struct Student a = {20240101, "Jinseog Kim", 99.5};
struct Student b;

y = x; // int 형 변수 복사
b = a; // 구조체의 복사
```

- 사용자가 필요한 연산이 있으면 정의하여 사용

구조체를 포함하는 구조체, 구조체 배열

- 구조체를 포함하는 구조체

```c
typedef struct{
  int year;
  int month;
  int date;
}Birthday;
```

```c
typedef struct {
  char name[20];
  Birthday birthday; //구조체가 포함
}Friend;
```

- 구조체 배열선언 및 초기화

```c
Friend f_list[100]; // 구조체 배열 선언
// 구조체 배열 초기화
f_list[0].name = 'Jinseog';
f_list[0].birthday.year = 2000;
f_list[0].birthday.month = 8;
f_list[0].birthday.date = 10;
```

구조체와 함수

- 구조체의 값을 매개변수로 전달 (Call-by-value)
- 구조체의 주소를 매개변수로 전달 (Call-by-address)
- 아래의 Lab 에서 설명

Lab: 희소 행렬 표현하기

- 희소 행렬 (sparse matrix) : 행렬의 원소에 0 이 아닌 값이 매우 적은 행렬

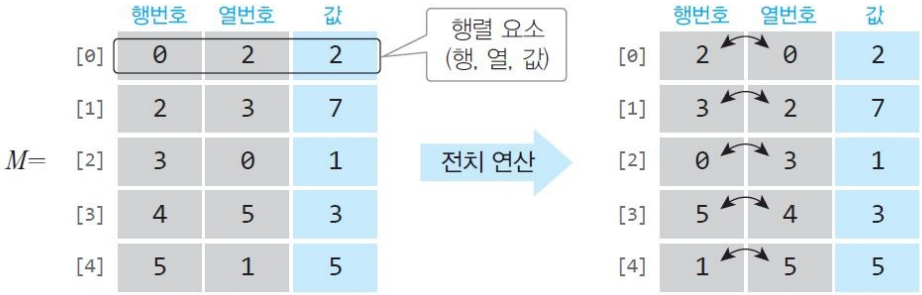
$$M = \begin{bmatrix} 002000 \\ 000000 \\ 000700 \\ 100000 \\ 000003 \\ 050000 \end{bmatrix}$$

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	2	0	0	0
[1]	0	0	0	0	0	0
[2]	0	0	0	7	0	0
[3]	1	0	0	0	0	0
[4]	0	0	0	0	0	3
[5]	0	5	0	0	0	0

- 희소행렬을 2 차원 배열로 저장하면 **메모리 낭비**가 심해짐

Lab: 희소 행렬 표현하기

- 희소행렬에서 0 이 아닌 값만 저장: (행 인덱스, 열 인덱스, 값)
- 효율적인 표현과 전치 연산



- 2 차원 배열을 이용할 수 있지만, 배열의 값이 정수가 아닌 경우 불가

Lab: 희소 행렬 표현하기

```
// 희소행렬의 각 요소를 위한 구조체
typedef struct{
  int row; // 행 인덱스
  int col; // 열 인덱스
  double val; // 해당 셀의 값
}Elem;
Elem x[5] = {{0,2,2}, {2,3,7},
             {3,0,1}, {4,5,3},
             {5,1,5}};
// 전치행렬 연산: 행번호 (row) 와 열번호 (col) 를 바꿔줌
void transpose(Elem x[], int len){
  for(i = 0; i<len; i++){
    int tmp = m[i].row;
    m[i].row = m[i].col;
    m[i].col = tmp;
  }
}
```

Lab: 다항식 프로그램

- 미지수 (x) 가 하나인 n 차 다항식의 일반적인 형태

$$f_n(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + ... + a_n x^n$$

  - 여기서 n 을 차수 (degree),
  - $a_i x^i$ 를 항 (term), a_i 를 계수 (coefficient), x^i 의 i 를 지수 (exponent)

- 위 다항식을 표현: 차수 (n) 과 각 항의 계수 (a_i)

- 구조체에 저장

```
1  typedef struct {
2    int degree; //다항식의 차수
3    float coef[100]; //계수를 저장 (여기서는 0 차부터 최고차항 순으로)
4  }Polynomial;
```

Lab: 다항식 프로그램 구현 & 테스트

```c
#include <stdio.h>
typedef struct {
  int degree; //다항식의 차수
  float coef[100]; //계수를 저장 (여기서는 0 차부터 최고차항 순으로)
}Polynomial;

// 미지수 x 를 대입한 결과를 반환
float evaluate(Polynomial p, float x)
{
    float result = p.coef[0];   // 상수항부터 시작
    float mul = 1;              // x^i 를 구하기 위함
    for (int i = 1; i <= p.degree; i++) {
        mul *= x;
        result += p.coef[i] * mul;
    }
    return result;
}
```

Lab: 다항식 프로그램 구현 & 테스트

```
// 다항식 출력
void print_poly(Polynomial p)
{
    for (int i = p.degree; i > 0; i--)
        printf("%5.1f x^%d + ", p.coef[i], i);
    printf("%4.1f\n", p.coef[0]);
}
void main()
{
    Polynomial a = { 5, { 3, 6, 0, 0, 0, 10} };
    print_poly(a);
    printf("a(1)= %f\n", evaluate(a, 1.0f));
}
```

희소 다항식의 표현 (연습)

- 희소 다항식: 계수가 0 인 항이 많은 (0 이 아닌 항이 매우 적은) 다항식
- 희소 다항식의 예: 차수는 100 이고 0 이 아닌 계수가 있는 항은 3 개

$$f(x) = 2 + 3x - x^{100}.$$

- 희소 다항식을 위한 구조체: 구조체의 배열을 사용하여 효율적 저장

```c
typedef struct {
  int exponent; // 지수
  float coeff;  // 계수를 저장
}Term;
typedef struct {
  int non_zeros; //다항식에서 0 이 아닌 계수의 수
  Term term[100]; //항과 계수를 저장
}Polynomial;
```

연습문제와 참고 문제

- 연습문제 (P. 71): 2.7, 2.8, 2.9, 2.10, P2.12
- 참고문제: Top 50 Array Coding Problems for Interviews