

# 컴퓨터 프로그래밍 및 실습

## 강의자료 3

배열 (Array)

# 배열의 필요성

- 학생이 10명이 있고 이들의 평균 성적을 계산한다고 가정하자.

방법 #1 : 개별 변수 사용

```
int s0;  
int s1;  
...  
int s9;
```

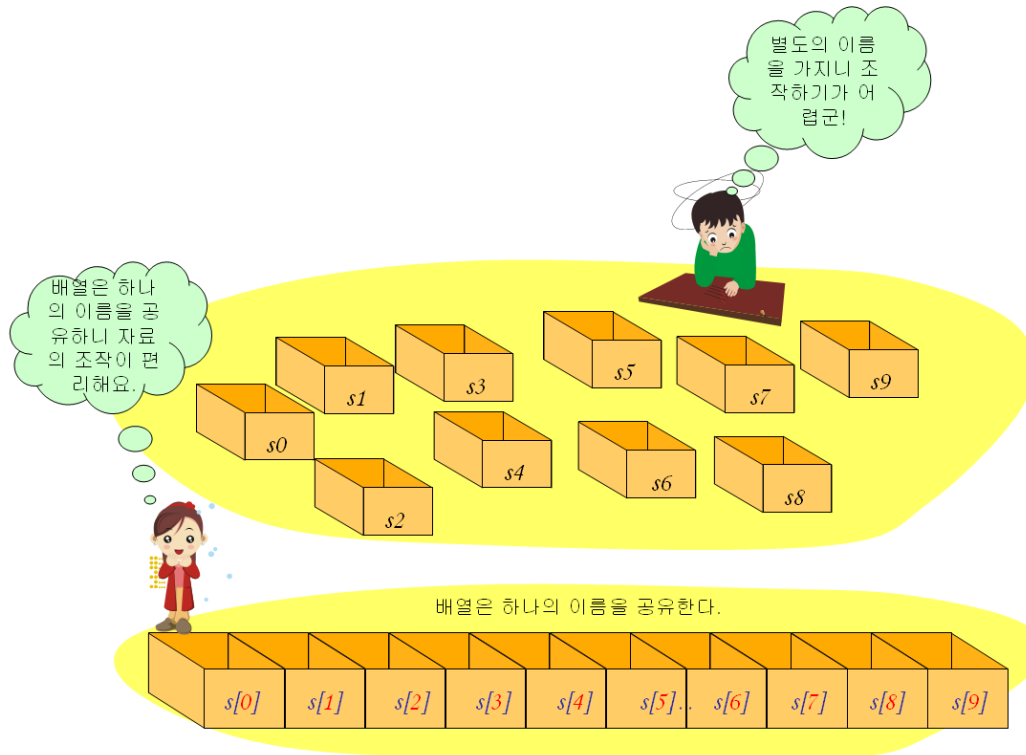
방법 #1 : 배열 사용

```
int[10];
```

개별변수를  
사용하는 방법은  
학생수가 많아지면  
번거로워집니다.

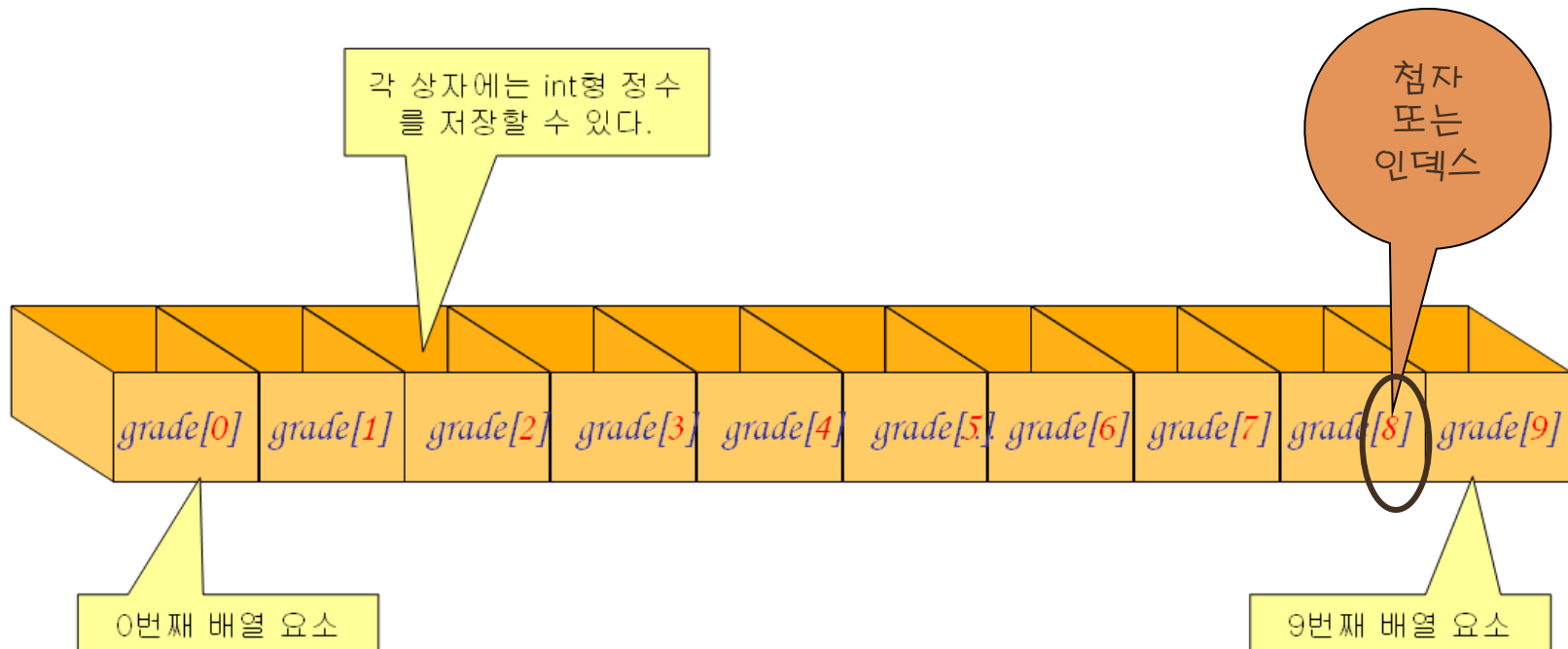
# 배열이란?

- 배열(array): 동일한 타입의 데이터가 여러 개 저장되어 있는 데이터 저장 장소
- 배열은 이름이 있고, 배열 안에 들어있는 각각의 데이터들은 정수로 되어 있는 번호(첨자)에 의하여 접근
- 배열을 이용하면 여러 개의 값을 하나의 이름으로 처리할 수 있다.

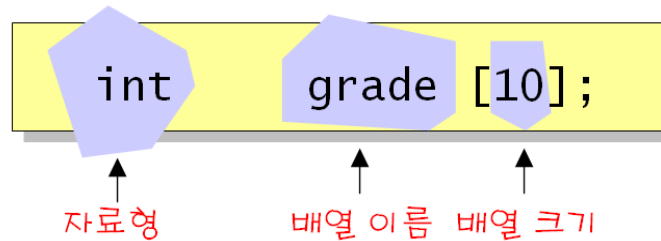


# 배열 원소와 인덱스

- *인덱스(index)* 혹은 *첨자*: 배열 원소의 번호



# 배열의 선언



- 자료형: 배열 원소들이 int형라는 것을 의미
- 배열 이름: 배열을 사용할 때 사용하는 이름이 grade
- 배열 크기: 배열 원소의 개수가 10개
- 인덱스(배열 번호)는 항상 0부터 시작한다.

# 배열 선언의 예

```
int score[60];
```

// 60개의 int형 값을 가지는 배열 grade

```
float cost[12];
```

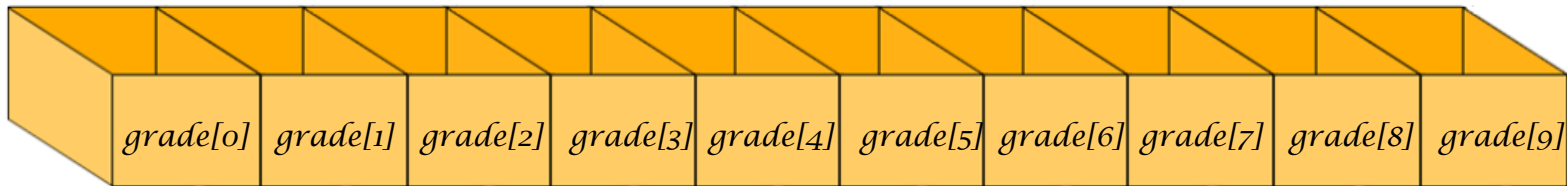
// 12개의 float형 값을 가지는 배열 cost

```
char name[50];
```

// 50개의 char형 값을 가지는 배열  
name

# 배열 원소 접근

```
int grade[10];
```



**grade[5] = 80**

인덱스 혹은 첨자

- 배열 원소는 배열이름[인덱스]로 접근한다.
  - 인덱스는 상수 혹은 정수형 식
  - 유효한 인덱스의 범위는 **0~(배열크기-1)**
  - 배열을 선언하는 경우, 대괄호 사이에 기술되는 수는 배열의 크기를 의미하지만, 각각의 배열 원소를 참조하는 대괄호 사이의 수는 배열 원소를 표현하는 첨자
  - 첫번째 배열원소는 첨자가 **0**이며, 다음 원소부터 첨자가 차례로 **1**씩 증가
    - 배열 첨자는 유효한 값의 범위를 벗어나는 경우, 문제 발생되므로 배열 첨자의 사용에 주의

# 배열 원소 접근 예

```
int grade[10];
```

```
grade[5] = 80;  
grade[1] = grade[0];  
grade[i] = 100;      // i는 정수 변수  
grade[i+2] = 100;    // 수식이 인덱스가 된다.  
grade[index[3]] = 100; // index[]는 정수 배열  
grade[10] = 100;    // 오류 : 인덱스가 배열 범위를 초과
```



# 배열 선언 예제

```
#include <stdio.h>

int main(void)
{
    int i;
    int grade[5];

    grade[0] = 10;
    grade[1] = 20;
    grade[2] = 30;
    grade[3] = 40;
    grade[4] = 50;

    for(i=0; i < 5; i++)
        printf("grade[%d]=%d\n", i, grade[i]);
    return 0;
}
```

```
grade[0]=10
grade[1]=20
grade[2]=30
grade[3]=40
grade[4]=50
```

# 배열과 반복문

- 배열의 가장 큰 장점은 반복문을 사용하여 배열의 원소를 간편하게 처리할 수 있다는 점

```
grade[0] = 0;  
grade[1] = 0;  
grade[2] = 0;  
grade[3] = 0;  
grade[4] = 0;
```

```
#define SIZE 5  
...  
for(i=0 ; i<SIZE ; i++)  
    grade[i] = 0;
```

# 배열 선언 예제

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5

int main(void)
{
    int i;
    int grade[SIZE];

    for(i = 0; i < SIZE; i++)
        grade[i] = rand() % 100;

    for(i = 0; i < SIZE; i++)
        printf("grade[%d]=%d\n", i, grade[i]);

    return 0;
}
```

```
grade[0]=41
grade[1]=67
grade[2]=34
grade[3]=0
grade[4]=69
```

# 배열 선언 예제

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5

int main(void)
{
    int i;
    int grade[SIZE];
    printf("5명의 점수를 입력하시오\n");

    for(i = 0; i < SIZE; i++)
        scanf("%d", &grade[i]);

    for(i = 0; i < SIZE; i++)
        printf("grade[%d]=%d\n", i, grade[i]);
    return 0;
}
```

5명의 점수를 입력하시오

23

35

67

45

21

grade[0]=23

grade[1]=35

grade[2]=67

grade[3]=45

grade[4]=21

# 배열 선언 예제

```
#include <stdio.h>
#define STUDENTS 5
int main(void)
{
    int grade[STUDENTS];
    int sum = 0;
    int i, average;
    for(i = 0; i < STUDENTS; i++)
    {
        printf("학생들의 성적을 입력하시오: ");
        scanf("%d", &grade[i]);
    }
    for(i = 0; i < STUDENTS; i++)
        sum += grade[i];
    average = sum / STUDENTS;
    printf("성적 평균= %d\n", average);

    return 0;
}
```

학생들의 성적을 입력하시오: 10  
학생들의 성적을 입력하시오: 20  
학생들의 성적을 입력하시오: 30  
학생들의 성적을 입력하시오: 40  
학생들의 성적을 입력하시오: 50  
성적 평균 = 30

# 배열의 초기화

## ○ 값의 초기화 방법

- 배열을 선언한 후 배열의 각 원소에 값을 저장하려면 인덱스를 이용하여 각 원소에 값을 대입 (그림1)
- 손쉽게 배열을 선언하면서 각 원소의 값을 지정하는 방법 : 배열 초기화(initialization) 구문 (그림2)

```
int a[4];  
a[0] = 10;  
a[1] = 30;  
a[2] = 40;  
a[3] = 50;
```

그림 1

```
int a[4] = {10, 30, 40, 50};
```

그림 2

```
int a[4] = {0}; // 모든 원소를 0으로 초기화
```

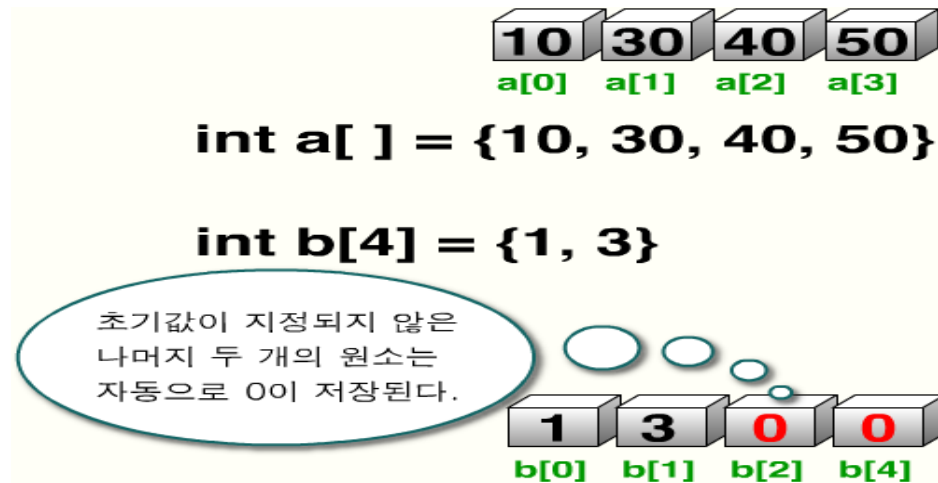
-배열 초기화(initialization)구문은 반드시 선언에서만 이용 가능

- 다음 문장은 잘못된 문장

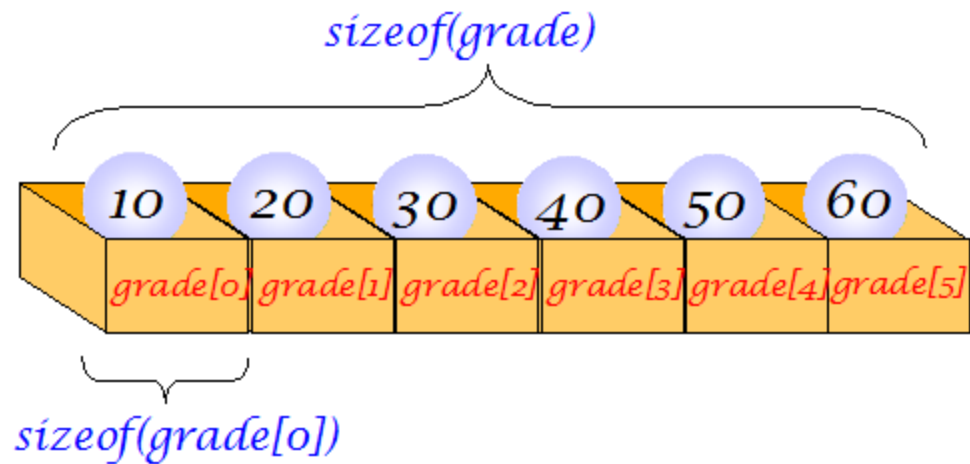
```
int a[4];  
a = {10, 30, 40, 50};
```

# 배열 초기화에서 배열 크기

- 배열의 초기화에서 배열의 크기는 생략 가능
  - 배열의 크기를 생략하는 경우, 초기값을 넣는 개수에 따라 배열의 크기가 결정
- 또한 배열의 크기를 지정하는 경우, 지정된 배열의 크기가 초기 값 개수보다 크면 나머지 지정되지 않은 원소의 초기 값은 자동으로 모두 0으로 저장



# 배열 원소의 개수 계산



```
int grade[] = { 1, 2, 3, 4, 5, 6 };  
int i, size;  
  
size = sizeof(grade) / sizeof(grade[0]);  
  
for(i = 0; i < size ; i++)  
    printf("%d ", grade[i]);
```



# 배열의 복사

- 어떤 배열을 다른 배열로 복사할 경우 원소 단위 복사를 해야한다.

```
int grade[SIZE];
```

```
int score[SIZE];
```

```
score = grade;
```

```
// 컴파일 오류!
```

잘못된 방법

```
int grade[SIZE];
```

```
int score[SIZE];
```

```
int i;
```

```
for(i = 0; i < SIZE; i++)
```

```
    score[i] = grade[i];
```

올바른 방법

# 두 배열의 내용이 같은지 비교

- 두 배열이 같은 값을 가지고 있는지를 알기 위해서는 원소 단위 비교를 해야 한다.

```
for(i = 0; i < SIZE ; i++)  
{  
    if ( a[i] != b[i] )  
    {  
        printf("a[]와 b[]는 같지 않습니다.\n");  
        return 0;  
    }  
}  
printf("a[]와 b[]는 같습니다.\n");  
return 0;  
}
```

# 예제 - 성적입력 후 평균 구하기

```
#include <iostream>
using namespace std;
int main(void)
{
    const int STUDENTS=5;
    int grade[STUDENTS];
    int sum = 0;
    int i, average;
    for(i = 0; i < STUDENTS; i++)
    {
        cout << "학생들의 성적을 입력하시오: ";
        cin >> grade[i];
    }
    for(i = 0; i < STUDENTS; i++)
        sum += grade[i];
    average = sum / STUDENTS;
    cout << "성적 평균= " << average << endl;
    return 0;
}
```

```
학생들의 성적을 입력하시오: 10
학생들의 성적을 입력하시오: 20
학생들의 성적을 입력하시오: 30
학생들의 성적을 입력하시오: 40
학생들의 성적을 입력하시오: 50
성적 평균 = 30
```

# 예제 - 성적입력 후 최대값 구하기 1

```
#include <iostream>
using namespace std;
int main(void)
{
    const int STUDENTS=5;
    int grade[STUDENTS];
    int sum = 0;
    int i, largest;
    for(i = 0; i < STUDENTS; i++)
    {
        cout << "학생들의 성적을 입력하시오: ";
        cin >> grade[i];
    }
    largest = grade[0];
    for(i = 1; i < STUDENTS; i++)
        if (largest < grade[i])
            largest = grade[i];
    cout << " 최대 성적= " << largest << endl;
    return 0;
}
```

학생들의 성적을 입력하시오: 10  
학생들의 성적을 입력하시오: 40  
학생들의 성적을 입력하시오: 20  
학생들의 성적을 입력하시오: 50  
학생들의 성적을 입력하시오: 30  
최대 성적 = 50

# 예제 - 성적입력 후 최대값 구하기 2

```
#include <iostream>
using namespace std;
int main(void)
{
    const int STUDENTS=5;
    int grade[STUDENTS];
    int sum = 0;
    int i, index;
    for(i = 0; i < STUDENTS; i++)
    {
        cout << "학생들의 성적을 입력하시오: ";
        cin >> grade[i];
    }
    index = 0; // 최대 원소의 위치
    for(i = 1; i < STUDENTS; i++)
        if (grade[index] < grade[i])
            index = i;
    cout << " 최대 성적= " << grade[index] << endl;
    return 0;
}
```

학생들의 성적을 입력하시오: 10  
학생들의 성적을 입력하시오: 40  
학생들의 성적을 입력하시오: 20  
학생들의 성적을 입력하시오: 50  
학생들의 성적을 입력하시오: 30  
최대 성적 = 50

# 예제 - 성적입력 후 정렬하기

```
#include <iostream>
using namespace std;
int main(void)
{
    const int STUDENTS=5;
    int grade[STUDENTS];
    int sum = 0;
    int i, index, size, temp;
    for(i = 0; i < STUDENTS; i++)
    {
        cout << "학생들의 성적을 입력하시오: ";
        cin >> grade[i];
    }
    for (size = STUDENTS; size > 1; size--)
    {
        index = 0;
        for(i = 1; i < size; i++)
            if (grade[index] < grade[i])
                index = i;
        temp = grade[size]; grade[size] = grade[index]; grade[index] = temp;
    }
    ....
    return 0;
}
```

학생들의 성적을 입력하시오: 10  
학생들의 성적을 입력하시오: 40  
학생들의 성적을 입력하시오: 20  
학생들의 성적을 입력하시오: 50  
학생들의 성적을 입력하시오: 30

# 배열 예제 - 그래프 출력

```
#include <iostream>
using namespace std;

const int STUDENTS = 5 ; // #define STUDENTS 5

int main(void)
{
    int grade[STUDENTS] = { 30, 20, 10, 40, 50 };
    int i, s;

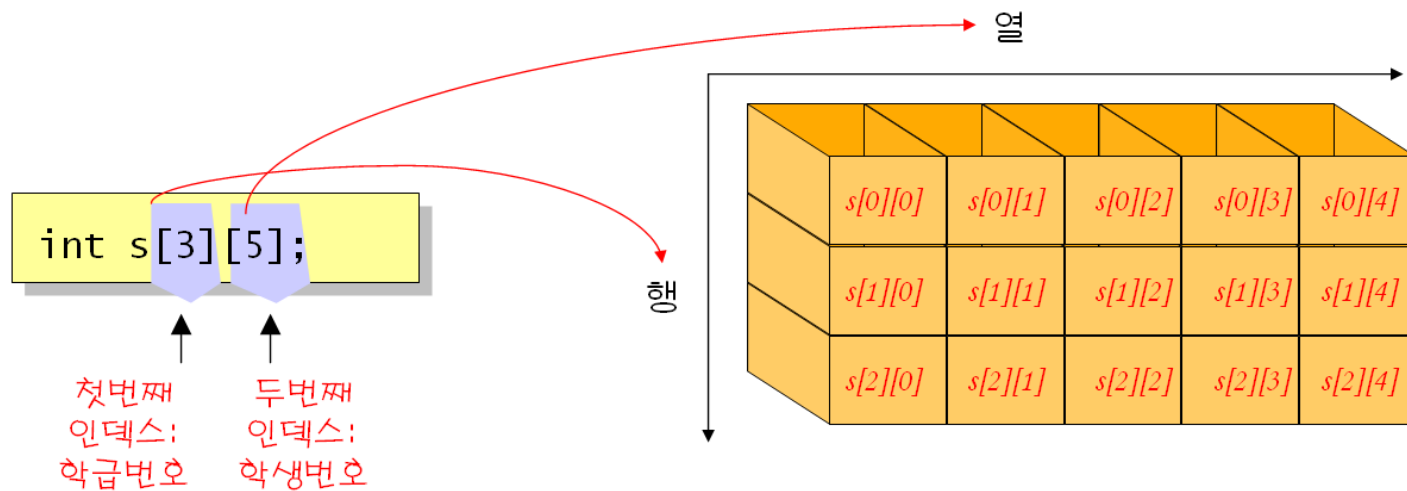
    for(i = 0; i < STUDENTS; i++)
    {
        cout << "번호 " << i;
        for(s = 0; s < grade[i]; s++)
            cout << "*";
        cout << "\n";
    }

    return 0;
}
```

```
번호 0: *****
번호 1: *****
번호 2: *****
번호 3: *****
번호 4: *****
```

# 2차원 배열

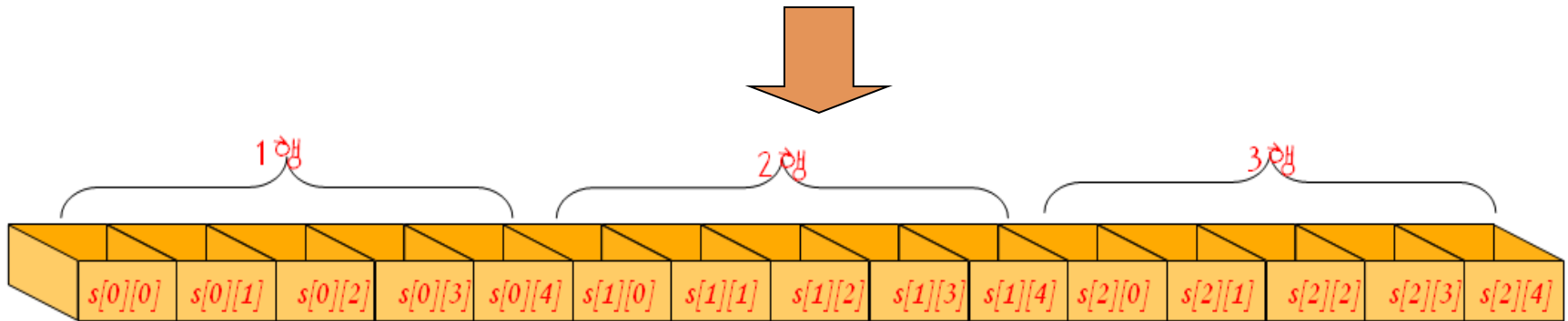
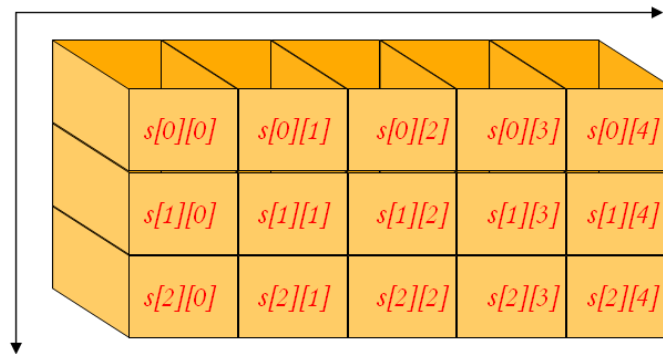
```
int s[10];    // 1차원 배열  
int s[3][10]; // 2차원 배열  
int s[5][3][10]; // 3차원 배열
```





# 2차원 배열의 구현

- 2차원 배열은 1차원적으로 구현된다.



# 2차원 배열의 활용

```
#include <iostream>
using namespace std;

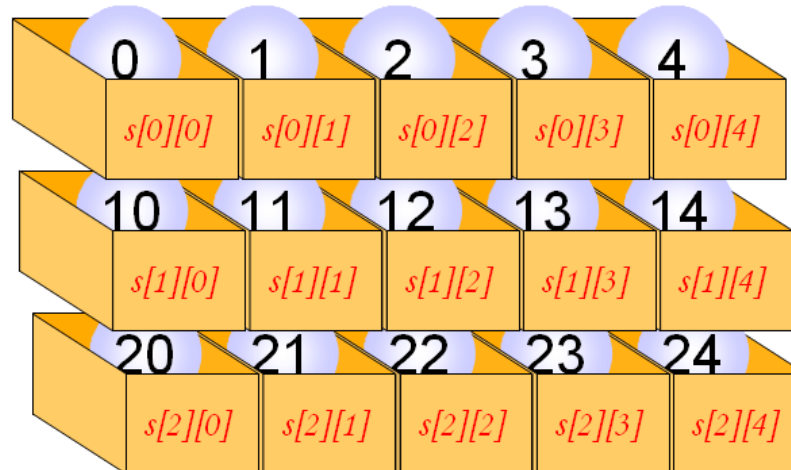
int main(void)
{
    int s[3][5];        // 2차원 배열 선언
    int i, j;           // 2개의 인덱스 변수
    int value = 0;      // 배열 원소에 저장되는 값

    for(i=0;i<3;i++)
        for(j=0;j<5;j++)
            s[i][j] = value++;

    for(i=0;i<3;i++)
    {
        for(j=0;j<5;j++)
            cout << s[i][j] << " ";
        cout << endl;
    }
    return 0;
}
```

# 2차원 배열의 초기화

```
int s[3][5] = {  
    { 0, 1, 2, 3, 4}, // 첫 번째 행의 원소들의 초기값  
    { 10, 11, 12, 13, 14}, // 두 번째 행의 원소들의 초기값  
    { 20, 21, 22, 23, 24}, // 세 번째 행의 원소들의 초기값  
};
```



# 이차원 배열을 이용한 행렬의 표현

```
#include <iostream>
using namespace std;
const int ROWS=3;
const int COLS=3;

int main()
{
    int A[ROWS][COLS] = {{ 2,3,0 }, { 8,9,1 }, { 7,0,5 } };
    int B[ROWS][COLS] = {{ 1,0,0 }, { 1,0,0 }, { 1,0,0 } };
    int C[ROWS][COLS];
    int r,c;
    for(r = 0;r < ROWS; r++)
        for(c = 0;c < COLS; c++)
            C[r][c] = A[r][c] + B[r][c];
    for(r = 0;r < ROWS; r++)
    {
        for(c = 0;c < COLS; c++)
            cout << C[r][c] << " ";
        cout << endl;
    }
    return 0;
}
```

2	3	0
8	9	1
7	0	5

1	0	0
1	0	0
1	0	0

3	3	0
9	9	1
8	0	5

# 3차원 배열

```
int s [6][3][5];
```

첫번째    두번째    세번째  
인덱스:    인덱스:    인덱스:  
학년번호    학급번호    학생번호

```
#include <stdio.h>
int main(void)
{
    int s[3][3][3];    // 3차원 배열 선언
    int x, y, z;        // 3개의 인덱스 변수
    int i = 1;          // 배열 원소에 저장되는 값

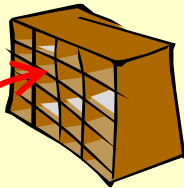
    for(z=0;z<3;z++)
        for(y=0;y<3;y++)
            for(x=0;x<3;x++)
                s[z][y][x] = i++;

    return 0;
}
```

# 배열과 함수

- 함수의 인자가 배열인 경우, 함수 호출시 형식인자는 실제인자 배열을 참조한다.

```
int main(void)
{
    ...

    get_average(  , int n);

    ...
}
```

- 실제인자 배열이 형식인자로 복사되는 것이 아님  
(즉, 실제인자 배열의 사본이 만들어지지 않음)

```
int get_average(int score[], int n)
{
    ...

    sum += score[i];

    ...
}
```

# 배열과 함수

```
#include <stdio.h>
#define STUDENTS 5
int get_average(int score[], int n); // ①
```

```
int main(void)
{
    int grade[STUDENTS] = { 1, 2, 3, 4, 5 };
    int avg;

    avg = get_average(grade, STUDENTS);
    printf("평균은 %d입니다.\n", avg);
    return 0;
}
```

배열이 인자인 경우,  
참조에 의한 호출

```
int get_average(int score[], int n) // ②
{
    int i;
    int sum = 0;

    for(i = 0; i < n; i++)
        sum += score[i];
    return sum / n;
}
```

실제인자 배열이  
**score**로 참조

# 배열이 함수의 인자인 경우

```
#include <stdio.h>
#define SIZE 7
```

```
void square_array(int a[], int size);
void print_array(int a[], int size);
```

```
int main(void)
```

```
{
    int list[SIZE] = { 1, 2, 3, 4, 5, 6, 7 };
```

```
    print_array(list, SIZE);
    square_array(list, SIZE);
    print_array(list, SIZE);
```

```
    return 0;
```

```
}
```

배열의 원본 list이 a로  
참조된다.

배열의 원본 list이 a로  
참조된다.

```
void print_array(int a[], int size)
{
    int i;
```

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

```
void square_array(int a[], int size)
{
```

```
    int i;
    for(i = 0; i < size; i++)
        a[i] = a[i] * a[i];
}
```

1	2	3	4	5	6	7
1	4	9	16	25	36	49



# 원본 배열의 변경을 금지하는 방법

```
void print_array(const int a[], int size)
```

```
{
```

```
...
```

```
    a[0] = 100;
```

```
}
```

함수 안에서 a[]는 변경할 수 없다.

**// 컴파일 오류!**

# 재귀 (recursion)

- 문제의 해를 구하는 알고리즘 설계 기법
- 주어진 입력에 대한 해결방법을 작은 입력에 대한 자신의 해결방법으로 기술
- 두 부분으로 이루어짐

Base case와 general case (recursive case)

(1) Base case

직접 해를 구하는 부분

(2) General case (recursive case)

주어진 입력에 대한 알고리즘을 작은 입력에 대한 자기자신의 방법으로 기술

# 재귀

- 문제의 해를 구하는 알고리즘 설계 기법
- 주어진 입력에 대한 해결방법을 작은 입력에 대한 자신의 해결방법으로 기술
- 두 부분으로 이루어짐

Base case와 general case (recursive case)

(1) Base case

직접 해를 구하는 부분

(2) General case (recursive case)

주어진 입력에 대한 알고리즘을 작은 입력에 대한 자기자신의 방법으로 기술

# 예 1

- Factorial

$n! = 1$  if  $n = 0$  // base case  
 $= n * (n-1)!$  If  $n > 0$  // recursive case

```
int fact(int n)
{
    if (n == 0)
        return 1;
    else
        return fact(n-1);
}
```

```
int main()
{
    int n;
    int result;
    scanf("%d", &n);

    result = fact(n);
    printf(" %d ! = %d □n", n,
        result);
    return 0;
}
```

## 예 2

- GCD (Greatest Common Divisor)

- 양의 정수  $m, n$  ( $m \geq n$ )의 최대공약수

$\text{gcd}(m, n) = n$  if  $m \% n == 0$  // base case  
 $= \text{gcd}(n, m \% n)$  if  $m \% n \neq 0$  // recursive case

```
int gcd(int m, int n)
{
    if (m % n == 0)
        return n;
    else
        return gcd(n, m%n);
}
```

```
int main()
{
    int a, b;
    int result;
    scanf("%d %d", &a, &b);

    result = gcd(a, b);
    printf("gcd of %d and %d= %d □n", a, b,
        result);
    return 0;
}
```

## 예 3

- 정수의 각 자리의 숫자 합 구하기

$\text{digits\_sum}(n) = n$  if  $n == 0$  // base case  
 $= \text{digits\_sum}(n/10) + n \% 10$  if  $n > 0$  // recursive case

```
int digits_sum(int n)
{
    if (n == 0)
        return 0;
    else
        return digits_sum(n/10)
        + n%10;
}
```

```
int main()
{
    int n;
    int result;
    scanf("%d", &n);

    result = digits_sum(n);
    printf(" %d 의 각 자리의 숫자합 = %d □n", n,
        result);
    return 0;
}
```

## 예 4

- 배열  $a$ 의  $n$ 개 원소들의 합  $a[0] + a[1] + \cdots a[n-1]$  구하기

`array_sum(int a[], n)`

`= a[0]`

if `n == 1` // base case

`= array_sum(a, n-1) + a[n-1]`

if `n > 1` // recursive case

```
int array_sum(int a[], int n)
{
    if (n == 1)
        return a[0];
    else
        return array_sum(a, n-1)
            + a[n-1];
}
```

```
int main()
{
    int score[100], n;
    int i;
    float avr;
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        scanf("%d", &score[i]);
    avr = array_sum(score, n)/ float(n);
    printf("평균 = %f □n", avr);
    return 0;
}
```