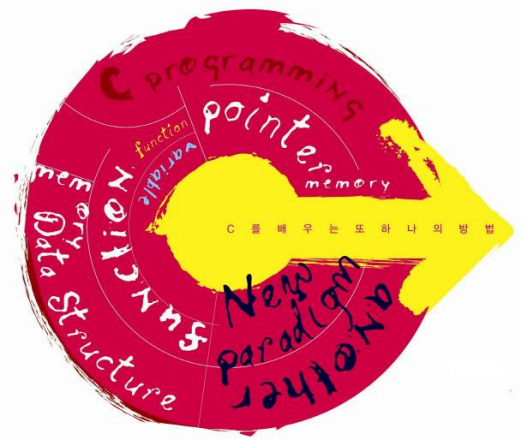
Clickseo.com

# C Programming

## 배열

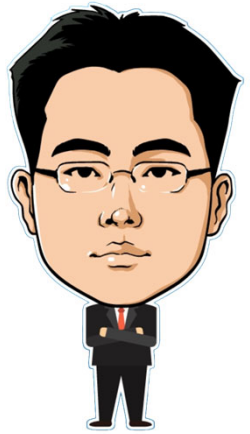
## (Arrays)




Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



CC BY NC ND

# 목 차

CC BY NC ND

백문이불여일타(百聞而不如一打)

- 배열의 이해: 1차원 배열
- 문자열의 이해
- 다차원 배열
- 정렬과 검색 알고리즘

# 배열의 이해: 1차원 배열

CC BY NC ND

- **배열의 이해:** 1차원 배열

백문이불여일타(百聞而不如一打)

  - 배열과 포인터
  - 배열과 함수
- 문자 배열(문자열)
- 다차원 배열
- 정렬과 검색 알고리즘

# 배열의 이해: 1차원 배열 (1/6)

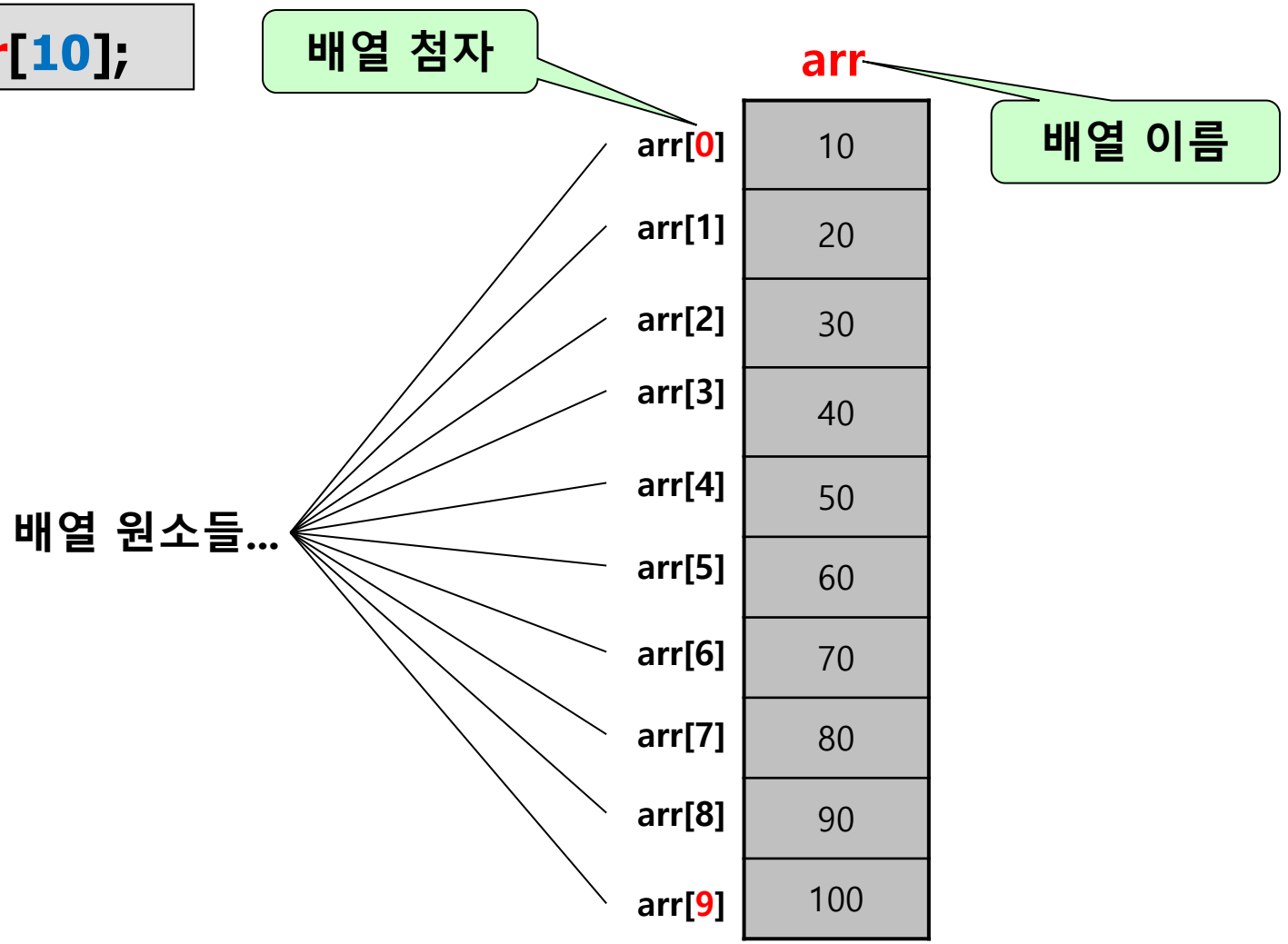
- **배열(Array)**
  - 동일한 데이터 형의 자료들을 모아 메모리에 연속적으로 저장하여 사용하는 변수들의 집합체를 말한다.
    - 배열은 단일 자료 형의 원소들로 구성되며, 메모리의 연속된 위치에 저장된다.
  - **배열 원소(Array Element)**
    - 배열의 원소는 원소의 번호를 **색인(index)**으로 사용하여 참조한다.
      - 배열의 첫 번째 원소에 접근하기 위해서는 색인을 **0** 으로 사용한다.

**N 개의 원소를 갖는 배열**

**arr[0] , arr[1] , ... , arr[N - 1]**

# 배열의 이해: 1차원 배열 (2/6)

```
int     arr[10];
```



# 배열의 이해: 1차원 배열 (3/6)

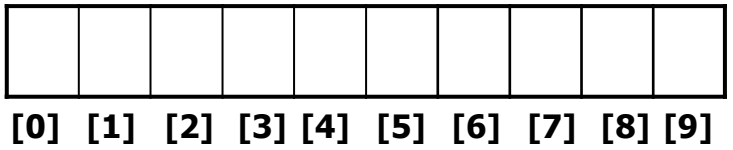
## ● 배열 선언 및 정의

❍ 배열의 선언 및 정의는 컴파일러에게 배열의 이름, 각 원소의 자료형 그리고 배열의 크기(원소의 개수)를 컴파일러에게 알려준다.

- 즉, 자료형과 배열 이름 그리고 요소의 수를 이용해 선언한다.

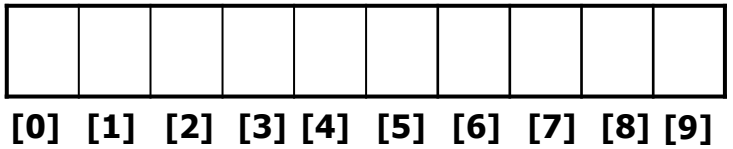
[기억 클래스] 자료형 배열이름[배열크기];

```
char    name[10];
```



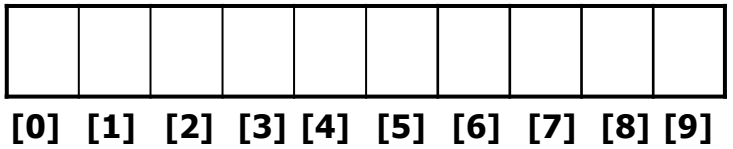

10 bytes

```
int     tot[10];
```




40 bytes

```
double  ave[10];
```




80 bytes

# 배열의 이해: 1차원 배열 (4/6)

## ● 배열 초기화

### ❍ 배열을 초기화할 때 콤마(,) 이후의 모든 요소는 0으로 초기화된다.

```
// tot[0] = 80, tot[1] = 90, tot[2] = 0, tot[3] = 0, tot[4] = 0
int         tot[5]  =  { 80,  90, };
```

- 배열 원소의 자료형과 초기화 값이 서로 다르면, 배열 원소의 자료형으로 (자동) 형 변환이 이루어진다.

```
double     ave[5]  =  { 70,  75, 80,  85, 90 };      // double으로 형 변환되어 저장
```

```
int  arr[5] = { 10, 20, 30, 40, 50 };
```

| 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|

```
int  arr[ ] = { 10, 20, 30, 40, 50 };
```

| 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|

```
int  arr[5] = { 10, 20 };
```

| 10 | 20 | 0 | 0 | 0 |
|---|---|---|---|---|

```
int  arr[1000] = { 0 };
```

| 0 | 0 | ... | 0 | 0 |
|---|---|---|---|---|

# 배열의 이해: 1차원 배열 (5/6)

## 예제 7-1: 배열의 논리적 순서와 물리적 순서

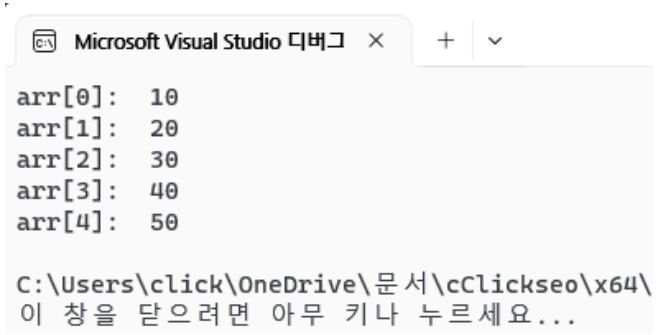
```
#include <stdio.h>
#define  arrMAXSIZE  5        // 배열의 크기(원소 개수)

int main(void)
{
        int     arr[arrMAXSIZE] = { 10, 20, 30, 40, 50 };

        printf("\n ### 1차원 배열의 논리적 순서와 물리적 순서 ### \n\n");
        for(int i=0; i<arrMAXSIZE; ++i)
                printf("&arr[%d] 주소 : %p, arr[%d] = %d\n",
                                                i, &arr[i], i, arr[i] );

        return 0;
}
```

```
Microsoft Visual Studio 디버그

 ### 1차원 배열의 논리적 순서와 물리적 순서 ###

&arr[0] 주소 : 0000000E11D3F858, arr[0] = 10
&arr[1] 주소 : 0000000E11D3F85C, arr[1] = 20
&arr[2] 주소 : 0000000E11D3F860, arr[2] = 30
&arr[3] 주소 : 0000000E11D3F864, arr[3] = 40
&arr[4] 주소 : 0000000E11D3F868, arr[4] = 50

C:\Users\click\OneDrive\문서\cClickseo\x64\Debug\
이 창을 닫으려면 아무 키나 누르세요...
```

# 배열의 이해: 1차원 배열 (6/6)

## 예제 7-2: 배열의 선언과 데이터 저장

```c
#include <stdio.h>
#define  arrMAXSIZE  5 // 배열의 크기(원소 개수)

int main(void)
{
        int     arr[arrMAXSIZE];

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

    for(int i=0; i<arrMAXSIZE; ++i) {
        printf("arr[%d] : ", i);
        scanf_s("%3d", &arr[i] );
        // scanf("%3d", &arr[i] );
    }

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



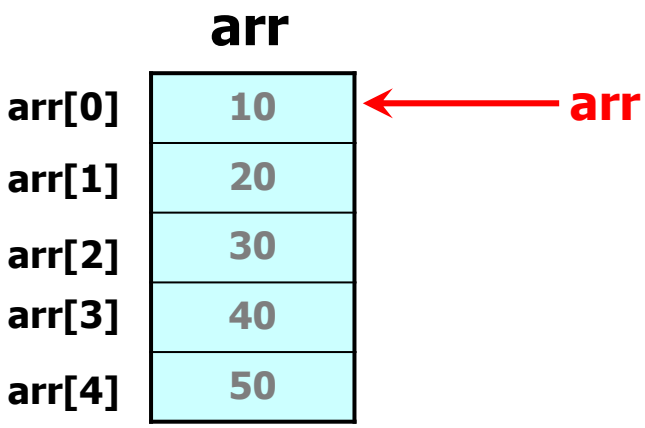
CC BY NC ND

# 배열의 이해: 1차원 배열

## 배열과 포인터

# 배열과 포인터 (1/6)

- **배열과 포인터: 배열 이름**
  - 배열 이름은 배열의 첫 요소를 위한 **포인터 상수** 이다.
    - 배열의 이름을 간접 참조할 때는 그 배열의 첫 요소를 간접 참조 하는 것




"배열 이름은...
첫 번째 배열 원소의 시작 주소 값을 갖는 포인터 상수"

**arr == &arr[0]**

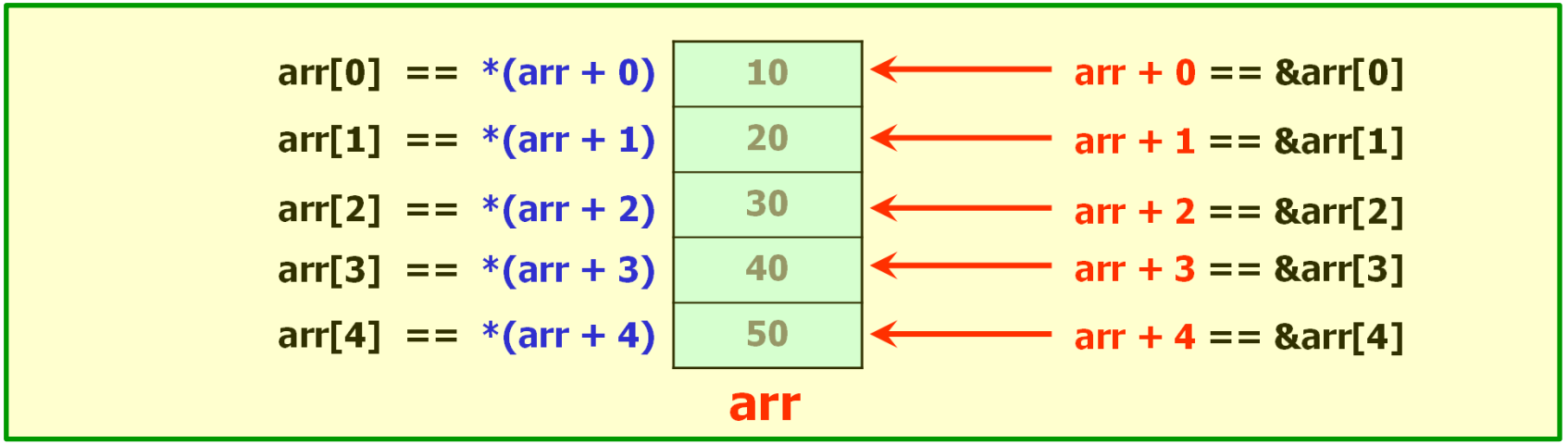
# 배열과 포인터 (2/6)

- **배열과 포인터:** 포인터 연산
  - 포인터 연산의 사용

```
int      *pArr = arr + 4;
```

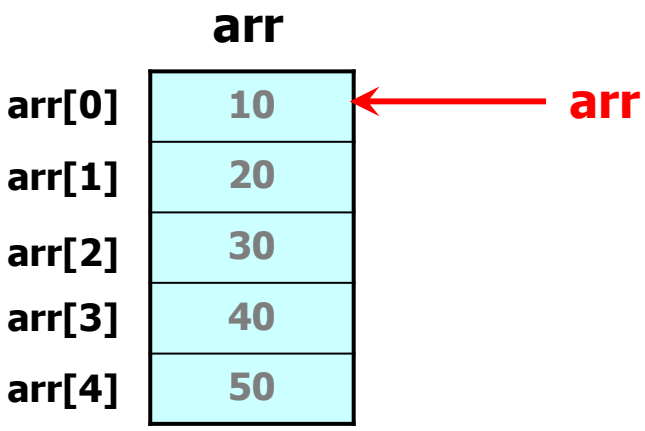
    - 다른 포인터에 할당
    - 간접연산자의 사용: arr 가 배열 이름이고 num 이 상수일 때, 다음 2 개의 표현은 완전히 동일하다.

```
arr [num]  ==  *(arr + num)
```



# 배열과 포인터 (3/6)

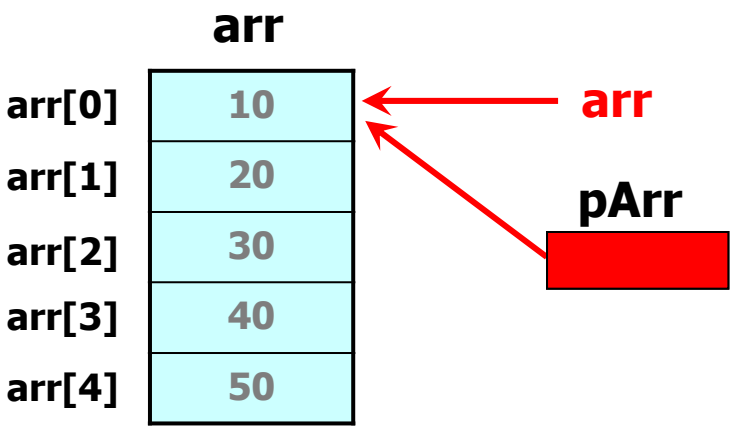
- **배열과 포인터:** 배열 이름과 간접 참조




```
#include <stdio.h>
int main(void)
{
        int         arr[5] = { 10, 20, 30, 40, 50 };

        // arr == &arr[0]
        // *arr == arr[0]
        printf("%d %d", *arr, arr[0] );

        return 0;
}
```

# 배열과 포인터 (4/6)

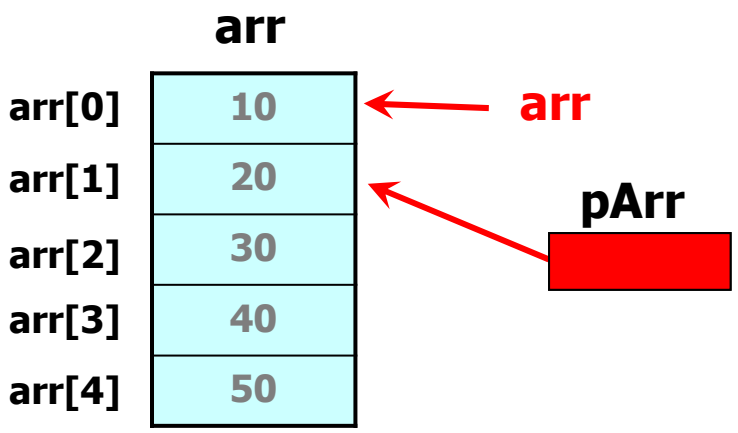
- **배열과 포인터:** 배열 이름과 포인터




```
#include <stdio.h>
int main(void)
{
        int     arr[5] = { 10, 20, 30, 40, 50 };

        // int  *pArr = &arr[0];
        int     *pArr = arr;

        printf("%d %d", arr[0], *pArr );

        return 0;
}
```

# 배열과 포인터 (5/6)

- **배열과 포인터:** 첨자 연산과 포인터

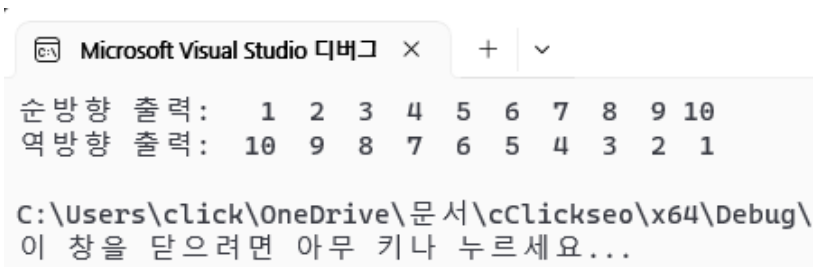



```
#include <stdio.h>
int main(void)
{
        int     arr[5] = { 10, 20, 30, 40, 50 };
        int     *pArr = &arr[1];

        printf("%d %d",  arr[0], pArr[-1] );
        printf("%d %d",  arr[1], pArr[0] );

        return 0;
}
```

# 배열과 포인터 (6/6)

## 예제 7-3: 배열과 포인터 -- 포인터 연산

```c
#include <stdio.h>
#define  arrMAXSIZE  10       // 배열의 크기(원소 개수)

int main(void)
{
        int     arr[arrMAXSIZE] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int     *pArr;

        pArr = arr;                          // pArr = &arr[0];
        printf("순방향 출력: ");
        for ( ; pArr < arr + arrMAXSIZE; pArr++ )
                printf("%3d", *pArr );
        printf("\n");

        pArr = arr + arrMAXSIZE - 1;         // pArr = &arr[9];
        printf("역방향 출력: ");
        for(; pArr >= arr; pArr-- )
                printf("%3d", *pArr );
        printf("\n");

        return 0;
}
```



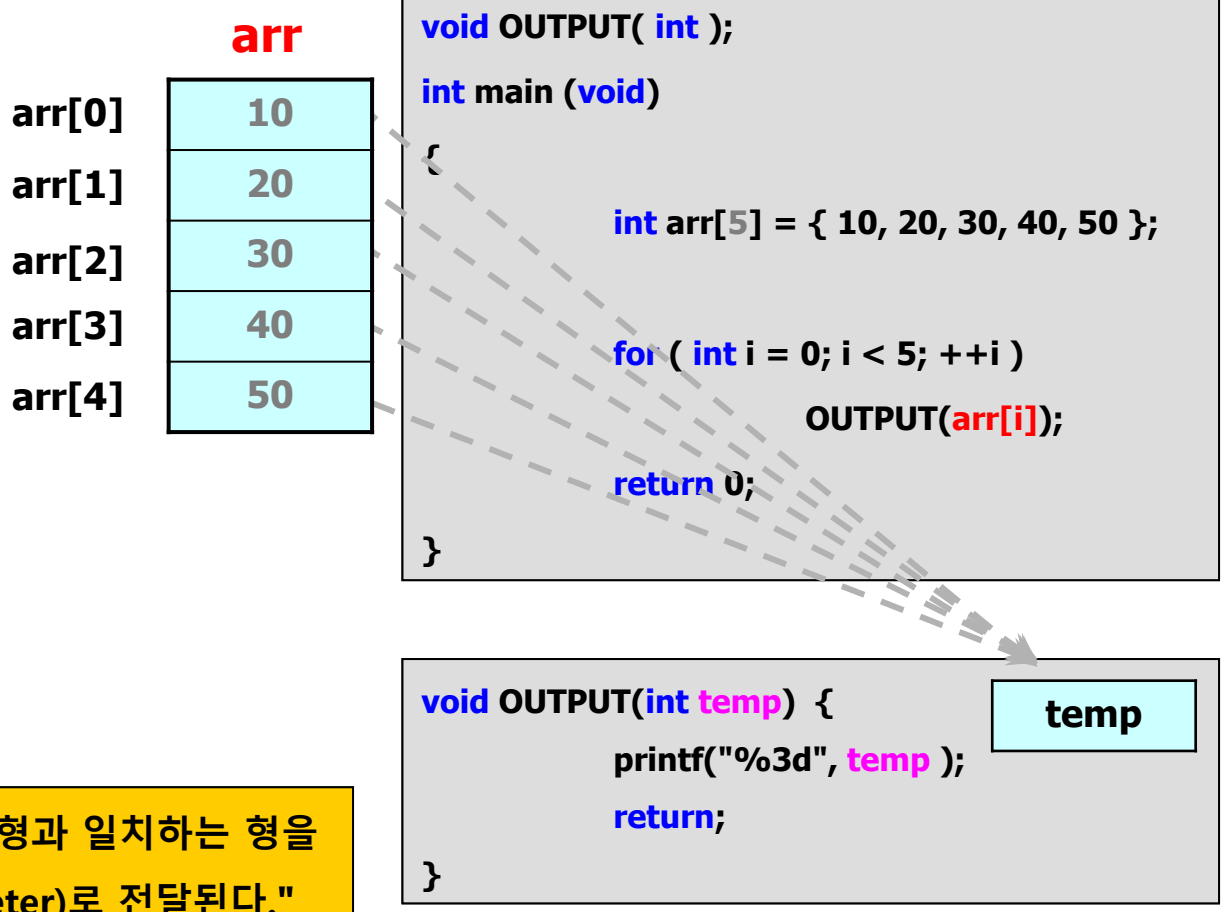
CC BY NC ND

# 배열의 이해: 1차원 배열

## 배열과 함수

# 배열과 함수 (1/3)

- 개별 원소 전달

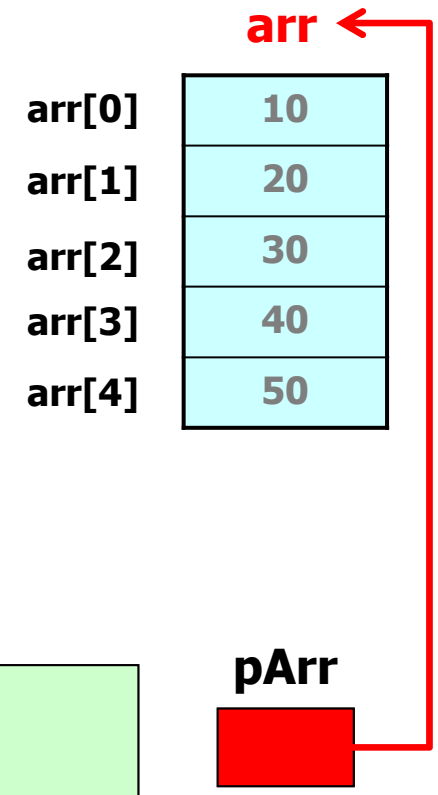



"함수가 요구하는 자료 형과 일치하는 형을 값-인수(value parameter)로 전달된다."

# 배열과 함수 (2/3)

- 전체 배열 전달

"배열 이름은
첫 번째 원소의 시작 주소"




```
#include <stdio.h>

// void  OUTPUT(int []);
void  OUTPUT(int *);

int main(void)
{
        int arr[5] = { 10, 20, 30, 40, 50 };
        OUTPUT(arr);
        return 0;
}
```

```
// void  OUTPUT(int pArr[])
void  OUTPUT(int *pArr)  {
        for ( int i = 0; i < 5; ++i )
                // printf("%3d", pArr[i] );
                 printf("%3d", *(pArr+i) );
        printf("₩n");
        return;
}
```

함수정의의 헤더부분의
다음 두 문장은 같다.

```
int  pArr[];    int *pArr;
```

# 배열과 함수 (3/3)

## 예제 7-4: 배열과 함수

```
#include <stdio.h>
#define  arrMAXSIZE  10        // 배열의 크기(원소 개수)

void OUTPUT(int *pArr, int num);

int main(void)
{
        int     arr[arrMAXSIZE] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

    OUTPUT(arr, sizeof(arr) / sizeof(int));

        return 0;
}
```

배열 크기(bytes) = sizeof(배열 한 원소의 자료형) * 원소 개수

```
void OUTPUT(int *pArr, int num) {
        printf("배열의 원소들... \n");
        for(int i=0; i<num; ++i )
                printf("%3d", *(pArr + i) ) ;
        printf("\n");
        return;
}
```

```
Microsoft Visual Studio 디버그
배열의 원소들...
  1  2  3  4  5  6  7  8  9 10

C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

# 문자 배열

CC BY NC ND

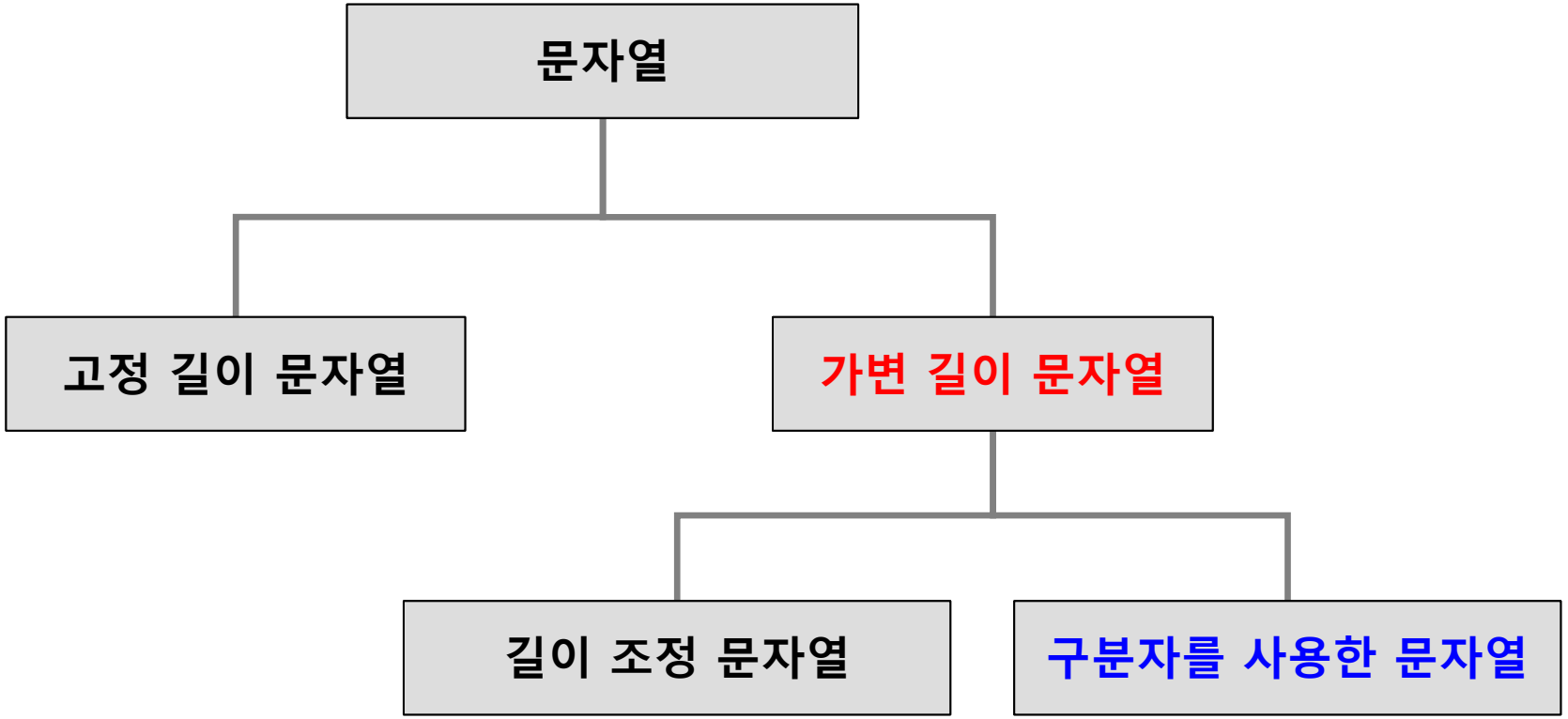
- 배열의 이해: 1차원 배열

백문이불여일타(百聞而不如一打)

- **문자 배열(문자열)**
  - 문자열 입출력
  - 문자열과 포인터
  - 포인터 배열
- 다차원 배열
- 정렬과 검색 알고리즘

# 문자 배열 (1/4)

- **문자열**(String)
  - 문자열은 연속된 문자들이며, 한 단위로 취급된다.

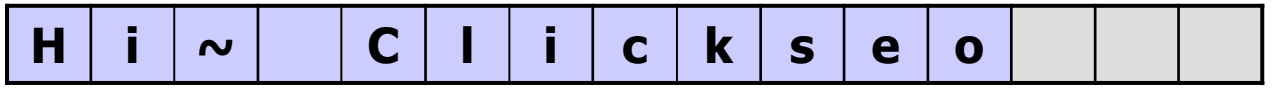


# 문자 배열 (2/4)

## ● 문자열 분류

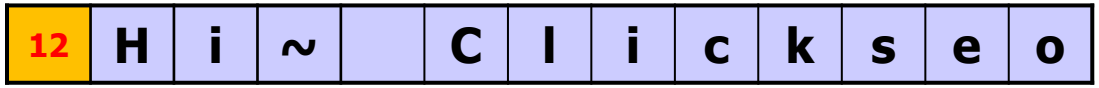
### ❍ 고정 길이 문자열

- 가장 먼저 변수의 크기를 결정한다.
- 빈 데이터의 표현: 공백같은 빈 데이터 문자를 마지막에 추가

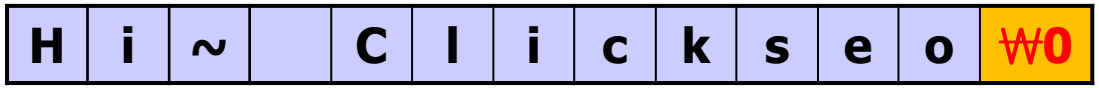



### ❍ 가변 길이 문자열

- 길이 조정 문자열: 문자열 내에 문자의 개수를 명시




- 구분자를 사용한 문자열: 문자열의 끝을 표시(널 문자, null character)

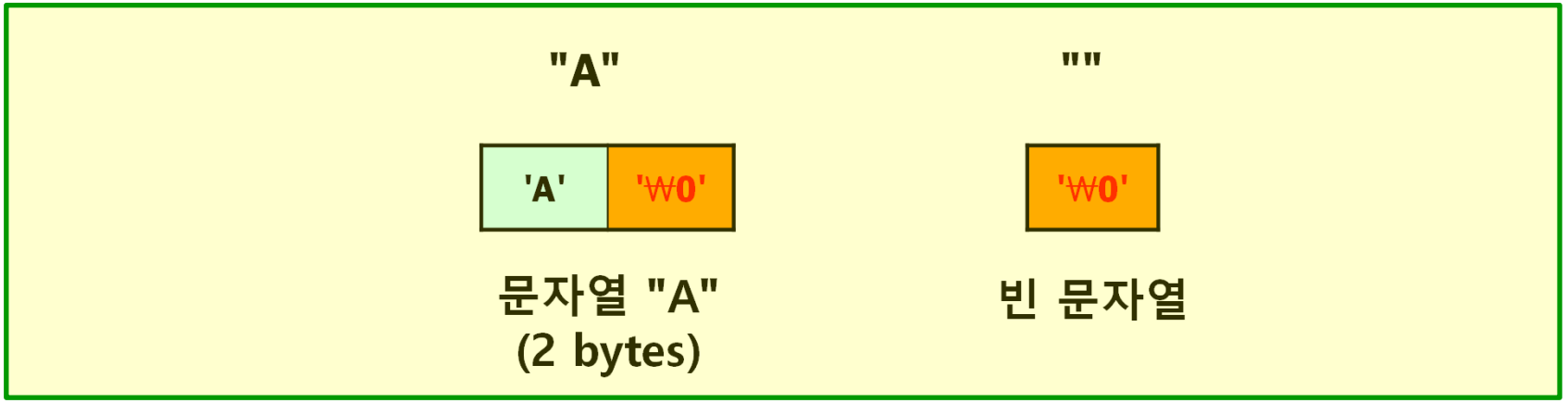


# 문자 배열 (3/4)

- **문자열 저장**
  - C 언어의 문자열은 널 문자로 구분되는 가변 길이 문자 배열
    - 문자열을 문자 배열에 저장할 때는 끝을 **널 문자 ('₩0')** 로 표시한다.

| H | i | ~ |  | C | l | i | c | k | s | e | o | ₩0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

```
char  str[13] = "Hi~ Clickseo";
```



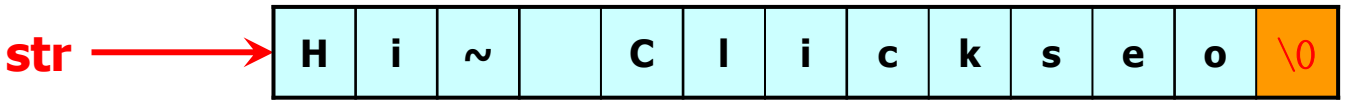
# 문자 배열 (4/4)

- **문자열 초기화**
  - 다른 초기화와 마찬가지로 정의할 때 값을 할당

```
char str[13] = "Hi~ Clickseo";        // 13 bytes 를 값과 구분자로 채운다.
char str[] = "Hi~ Clickseo";          // 컴파일러가 13 bytes 배열 생성
```

  - 문자 배열을 초기화: 코딩이 길어지므로 자주 쓰이지 않는다.

```
char  str[13]  =  {'H', 'i', '~', '', 'C', 'l' , 'i' , 'c' , 'k' , 's' , 'e' , 'o' , '₩0' };
```



CC BY NC ND

# 문자 배열

## 문자열 입출력

# 문자열 입출력 (1/4)

- **서식화 된 문자열 입출력: scanf, printf 계열 함수**
  - **문자열 변환 코드: %s**
  - **scanf 계열 함수**
    - 앞에 나오는 공백(whitespace) 문자를 모두 무시
    - 첫 번째 문자가 나오면 그때부터 공백 문자가 나올 때까지 문자를 읽으며 순서대로 배열에 저장 한다.
      - 공백 문자가 나오면 맨 마지막에 널 문자('₩0') 를 저장하고 문자열을 끝낸다.
  - **printf 계열 함수**
    - 문자열을 출력할 때 왼쪽 정렬 플래그, 너비, 한도 세 가지 옵션이 있다.
      - 정렬 플래그(-)는 왼쪽 정렬을 위해 쓴다.
      - 출력할 문자의 최대 개수를 지정하는 한도도 지정할 수 있다.

# 문자열 입출력 (2/4)

- **표준 입출력:** 표준 입출력 함수
  - 형식화되지 않은 입출력: 문자열 입출력

| 함수의 선언 | 기 능 | 헤더 파일 |
|---|---|---|
| ~~char *gets (char *str);~~ | 새로운 줄이 나오거나 파일 끝(EOF)을 만날 때까지 stdin 으로부터 byte 문자열을 읽어 들인다. (C99에서 사용하지 않고, C11에서 제거됨) | stdio.h |
| char *gets_s(char *str, rsize_t num); | // since C11 | |
| int puts(const char *str); | 주어진 문자열을 출력한다. | |

```
#include <stdio.h>
int main(void)
{
        char    str[12];

        gets(str);              // gets_s (str, sizeof(str));
        puts(str);

        return 0;
}
```

```
// gets 함수 사용 시 Visual Studio 에서 경고(warning) 메시지
warning C4013: 'gets'이(가) 정의되지 않았습니다.
extern은 int형을 반환하는 것으로 간주합니다.
```

# 문자열 입출력 (3/4)

## 예제 7-5: 문자열 입출력 -- gets과 pus 계열 함수

```c
#include <stdio.h>

int main(void)
{
        char    addr[1024];

        printf("주소 입력: ");
        gets_s(addr, sizeof(addr));          // gets(addr);

        puts("입력된 주소: ");
        puts(addr);

        return 0;
}
```

```
Microsoft Visual Studio 디버그
주소 입력: 서울시 무슨구 무슨동
입력된 주소:
서울시 무슨구 무슨동

C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

gets / gets_s : “test” 입력 시...

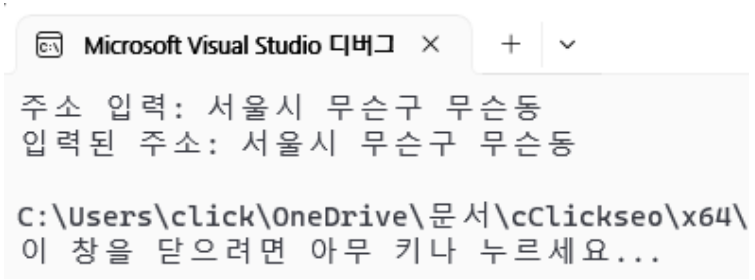
로컬

| 이름 | 값 |
|---|---|
| addr | 0x006ff888 "test" |
| [0] | 116 't' |
| [1] | 101 'e' |
| [2] | 115 's' |
| [3] | 116 't' |
| [4] | 0 '\0' |

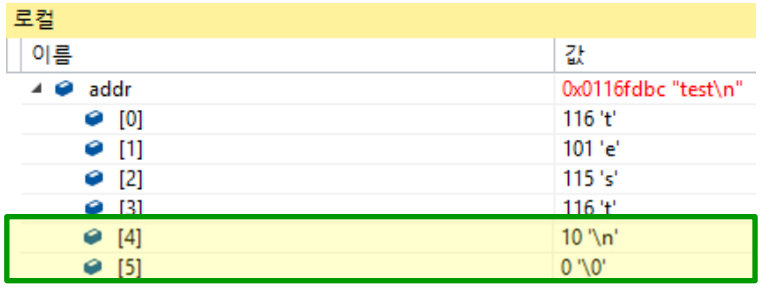
# 문자열 입출력 (4/4)

## 예제 7-6: 문자열 입출력 -- fgets과 fpus 함수

```c
#include <stdio.h>

int main(void)
{
        char    addr[1024];

        printf("주소 입력: ");
        fgets(addr, sizeof(addr), stdin);

        fputs("입력된 주소: ", stdout);
        fputs(addr, stdout);

        return 0;
}
```




gets / gets_s : “test” 입력 시...



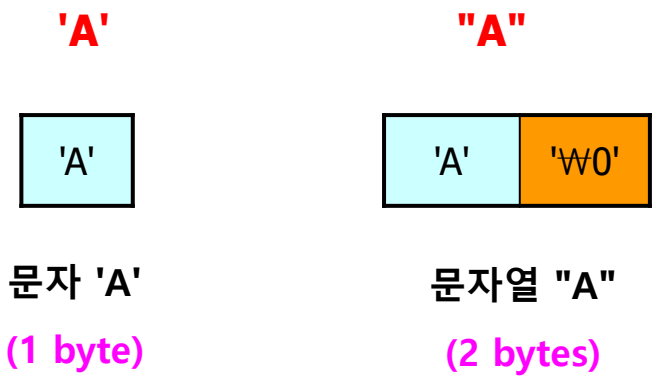
CC BY NC ND

# 문자 배열

## 문자 배열과 포인터

# 문자 배열과 포인터 (1/5)

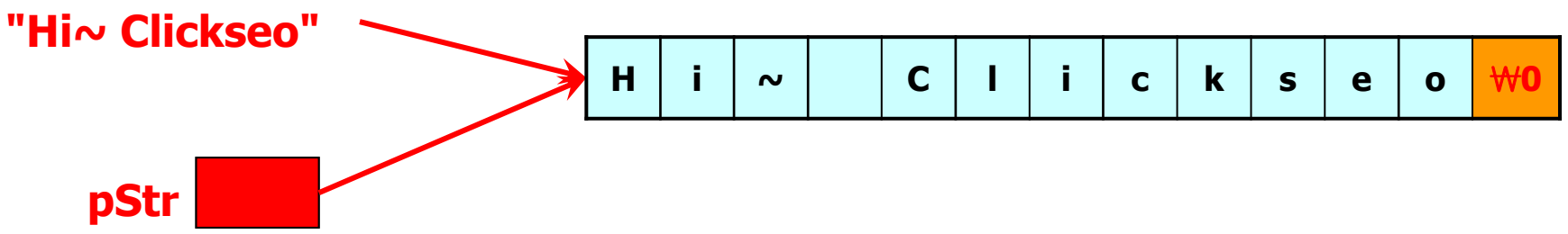
- **문자와 문자열 상수**
  - **문자 상수:** 문자 상수를 쓰려면 **작은 따옴표(' ')** 를 사용
  - **문자열 상수:** 문자열 상수를 쓰려면 **큰 따옴표(" ")** 를 사용
    - 문자열은 데이터 자체는 하나지만 그 뒤에 구분자를 쓸 공간이 필요하다.



# 문자 배열과 포인터 (2/5)

- **문자열 초기화**
  - 포인터형 변수에 문자열의 시작 주소를 할당
    - 문자열을 생성한 후, 포인터형 변수 pStr에 주소를 저장

```
char    *pStr  =  "Hi~ Clickseo";
```



# 문자 배열과 포인터 (3/5)

- **포인터형 변수와 상수**
  - char* pStr 과 char arr[ ] 의 차이

```
#include <stdio.h>
int main(void)
{
        char    arr[] = "Hi~ Clickseo";
        char    *pStr = "Hi~ Clickseo";

        pStr++;   // pStr = pStr + 1

        // error C2105: '++' needs l-value
        arr++;    // arr = arr + 1

        return 0;
}
```

"배열 이름 arr 는 변수가 아니라 포인터 상수이다."

# 문자 배열과 포인터 (4/5)

- **포인터형 변수와 상수:** 문자열 상수
  - 문자열 상수 변경

```
#include <stdio.h>

int main(void)
{
        char    *pStr = "Hi~ Clickseo";

        // 'A' 라는 문자를 pStr 가 가리키고 있는 메모리 주소 복사를 시도!!!
        *pStr = 'A';            // Segmentation Fault(memory dump)

        return 0;
}
```

"Hi~ Clickseo"는 문자열 상수이다.

"Hi~ Clickseo"라는 문자열 상수가 다른 문자열로 대체 될 수 있는가?

# 문자 배열과 포인터 (5/5)

- **문자열 출력**
  - 포인터형 변수를 이용한 문자열 출력

```
#include <stdio.h>
int main(void)
{
        char    str[] = "Hi~ Clickseo";
        char    *pStr;

        pStr = str;
        while (*pStr)                           // while(*pStr != '₩0')
                putchar(*pStr++);
        putchar('\n');

        return 0;
}
```

CC BY NC ND

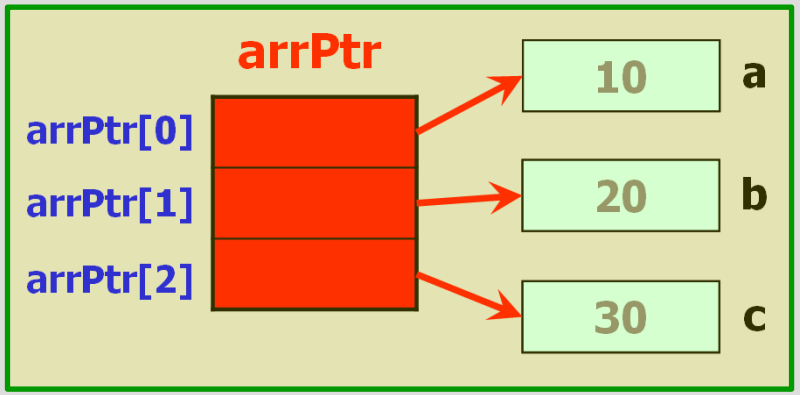
# 문자 배열

## 포인터 배열

# 포인터 배열 (1/3)

- **포인터 배열:** 정수형 변수

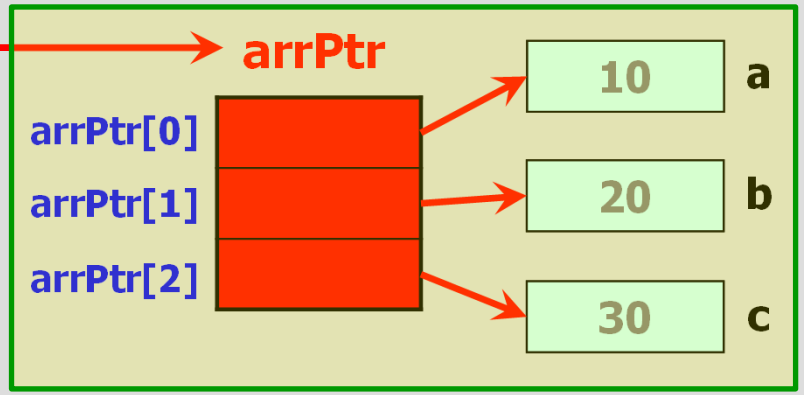
```
#include <stdio.h>
int main(void)
{
        int  a = 10, b = 20, c = 30;
        int *arrPtr[3];

        arrPtr[0]  =  &a;
        arrPtr[1]  =  &b;
        arrPtr[2]  =  &c;

        printf("a: %d, b: %d, c: %d ₩n", *arrPtr[0] , *arrPtr[1] , *arrPtr[2] );
        printf("a: %d, b: %d, c: %d ₩n", **arrPtr, **(arrPtr+1), **(arrPtr+2) );

        return 0;
}
```



# 포인터 배열 (2/3)

- **포인터 배열:** 다중 포인터

```
#include <stdio.h>
int main(void)
{
        int  a = 10, b = 20, c = 30;
        int *arrPtr[3];

        arrPtr[0] = &a;
        arrPtr[1] = &b;
        arrPtr[2] = &c;

        int **dArrPtr = arrPtr;
        for (int i = 0; i < 3; i++)
                printf("%3d", **dArrPtr++);
        printf("₩n");
        return 0;
}
```



# 포인터 배열 (3/3)

- **포인터 배열:** 문자열, 다중 포인터

```
#include <stdio.h>
int main(void)
{
        char *arrPtr[] =
                { "Hello C", "Hello C++", "Hi~ Clickseo", "Clickseo.com" };
        char **dArrPtr = arrPtr;

        printf(" [ %s ] ₩n", arrPtr[0]);
        printf(" [ %s ] ₩n", arrPtr[1]);
        printf(" [ %s ] ₩n", arrPtr[2]);
        printf(" [ %s ] ₩n", arrPtr[3]);

        for( int i = 0; i < 4; i++ )
                printf(" [ %s ] ₩n", *dArrPtr++);
        return 0;
}
```

# 다차원 배열

CC BY NC ND

- 배열의 이해: 1차원 배열

백문이불여일타(百聞而不如一打)

- 문자 배열(문자열)
- **다차원 배열**
  - 2차원 배열
  - 2차원 배열과 포인터
  - 2차원 배열과 함수
- 정렬과 검색 알고리즘

# 2차원 배열 (1/5)

- **2차원 배열:** 논리적 구조
  - 2차원 배열의 논리적 구조

2차원 배열
"1차원 배열의 배열"

```
int     table[3][5];
```

table

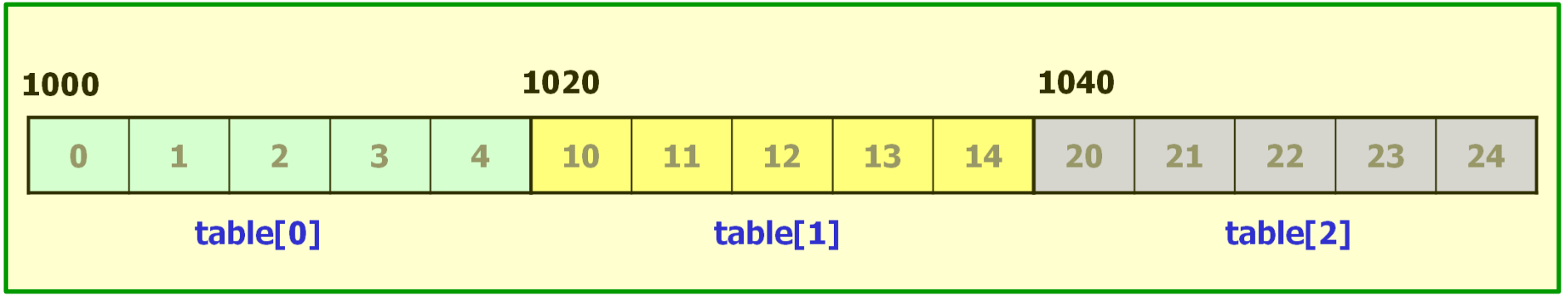
| | | | | | |
|---|---|---|---|---|---|
| table[0] | [0] | 0 | 1 | 2 | 3 | 4 |
| table[1] | [1] | 10 | 11 | 12 | 13 | 14 |
| table[2] | [2] | 20 | 21 | 22 | 23 | 24 |
| | | [0] | [1] | [2] | [3] | [4] |

```
table[i][j];
```

# 2차원 배열 (2/5)

- **2차원 배열:** 물리적 구조
  - 2차원 배열의 물리적 구조

```
int     table[3][5];
```




```
sizeof (table)    = 60 Bytes
sizeof (table[0]) = 20 Bytes
```

# 2차원 배열 (3/5)

- **2차원 배열:** 초기화

```
int table[3][5] = { 0, 1, 2, 3, 4, 10, 11, 12, 13, 14, 20, 21, 22, 23, 24 };
```

```
int table[3][5]  = {
                      {  0,   1,   2,   3,   4 },
                      { 10, 11, 12, 13, 14 },
                      { 20, 21, 22, 23, 24 },
                    };
```

# 2차원 배열 (4/5)

## 예제 7-7: 2차원 배열 -- 논리적 순서와 물리적 순서

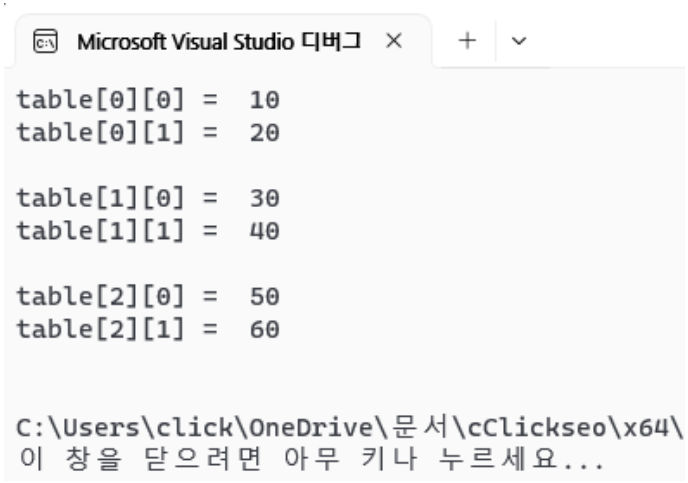
```c
#include <stdio.h>

#define ROW 3           // 행
#define COL  5          // 열

int main(void)
{
        int     table[ROW][COL] = {
                                    { 0,  1,  2,  3,  4 },
                                    { 10, 11, 12, 13, 14 },
                                    { 20, 21, 22, 23, 24 }
                                  };

        printf("### 2차원 배열의 논리적 순서와 물리적 순서 ### \n\n");
        for( int i=0; i<ROW; ++i ) {
                for( int j=0; j<COL; ++j )
                        printf("&table[%d][%d]: %p, table[%d][%d] = %d \n",
                                i, j, &table[i][j], i, j, table[i][j] );
                printf("\n");
        }
        return 0;
}
```

```
Microsoft Visual Studio 디버그
### 2차원 배열의 논리적 순서와 물리적 순서 ###

&table[0][0]: 0000009688BEF4B8, table[0][0] = 0
&table[0][1]: 0000009688BEF4BC, table[0][1] = 1
&table[0][2]: 0000009688BEF4C0, table[0][2] = 2
&table[0][3]: 0000009688BEF4C4, table[0][3] = 3
&table[0][4]: 0000009688BEF4C8, table[0][4] = 4

&table[1][0]: 0000009688BEF4CC, table[1][0] = 10
&table[1][1]: 0000009688BEF4D0, table[1][1] = 11
&table[1][2]: 0000009688BEF4D4, table[1][2] = 12
&table[1][3]: 0000009688BEF4D8, table[1][3] = 13
&table[1][4]: 0000009688BEF4DC, table[1][4] = 14

&table[2][0]: 0000009688BEF4E0, table[2][0] = 20
&table[2][1]: 0000009688BEF4E4, table[2][1] = 21
&table[2][2]: 0000009688BEF4E8, table[2][2] = 22
&table[2][3]: 0000009688BEF4EC, table[2][3] = 23
&table[2][4]: 0000009688BEF4F0, table[2][4] = 24


C:\Users\click\OneDrive\문서\cClickseo\x64\Debug\
이 창을 닫으려면 아무 키나 누르세요...
```

# 2차원 배열 (5/5)

## 예제 7-8: 2차원 배열 -- 배열 선언 및 데이터 입력

```c
#include <stdio.h>

#define ROW 3           // 행
#define COL  2          // 열

int main(void)
{
        int     table[ROW][COL];

        table[0][0] = 10;
        table[0][1] = 20;

        table[1][0] = 30;
        table[1][1] = 40;

        table[2][0] = 50;
        table[2][1] = 60;

        for( int i=0; i<ROW; ++i ) {
                for( int j=0; j<COL; ++j )
                        printf("table[%d][%d] = %3d \n", i, j, table[i][j] );
                printf("\n");
        }
        return 0;
}
```



CC BY NC ND

# 다차원 배열

## 2차원 배열과 포인터

# 2차원 배열과 포인터 (1/2)

- 2차원 배열과 포인터

table[i][j] == *(*(table + i) + j)

table + 0 == &table[0]
table + 1 == &table[1]
table + 2 == &table[2]

table[0]
table[1]
table[2]

table

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |
| 10 | 11 | 12 | 13 | 14 |
| 20 | 21 | 22 | 23 | 24 |

"배열 이름은...
첫 번째 배열 원소의 시작주소 값을 갖는 포인터 상수"

table == &table[0]
table[0] == &table[0][0]

# 2차원 배열과 포인터 (2/2)

- **2차원 배열과 포인터:** 포인터 변수

```c
#include <stdio.h>

#define ROW 3                   // 행
#define COL  5                  // 열

int main(void)
{
        int     table[ROW][COL] = { 0 };
        int     (*pTable)[COL];

        pTable = table;         // pTable = &table[0];
        for( int i=0; i<ROW; ++i ) {
                for( int j=0; j<COL; ++j )
                        printf("table[%d][%d] = %d ₩n", i, j, *(*(pTable + i) + j) );
                printf("₩n");
        }
        return 0;
}
```

Microsoft Visual Studio 디버그

```
table[0][0] = 0
table[0][1] = 0
table[0][2] = 0
table[0][3] = 0
table[0][4] = 0

table[1][0] = 0
table[1][1] = 0
table[1][2] = 0
table[1][3] = 0
table[1][4] = 0

table[2][0] = 0
table[2][1] = 0
table[2][2] = 0
table[2][3] = 0
table[2][4] = 0


C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

CC BY NC ND

# 다차원 배열

## 2차원 배열과 함수

# 2차원 배열과 함수 (1/3)

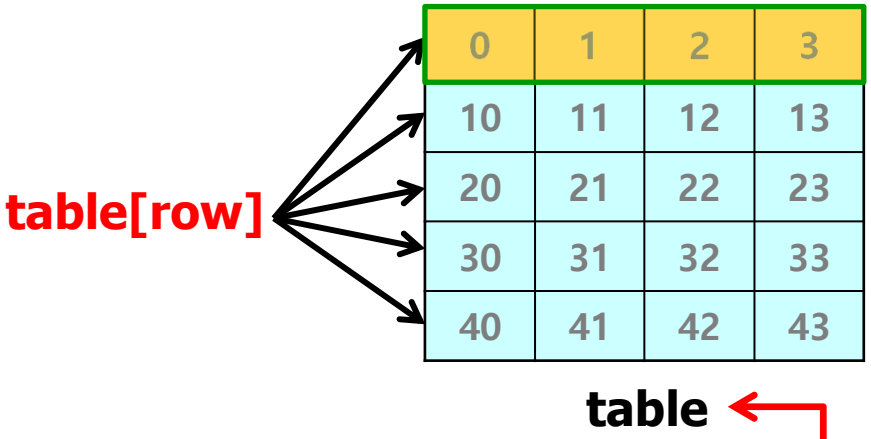
- **2차원 배열: 함수 전달**

  1. 각각의 원소들을 전달: table[0][0]
  2. 배열의 한 행을 전달: table[0]
     - 배열 이름을 한 행 번호로만 색인 함으로써 한 행 전체를 전달한다.
  3. 전체 배열을 전달: table
     - 배열 이름을 실 인수(actual parameter)로 사용한다.
     - 단, 두 번째 차원의 크기는 반드시 정의되어야 한다.

```
void    OUTPUT(int  table[ ][COL])
void    OUTPUT(int  (*table)[COL])
```

# 2차원 배열과 함수 (2/3)

- 한 행씩 전달

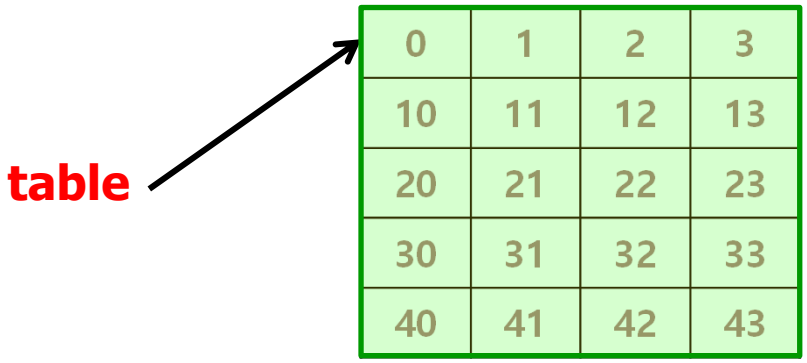



```
// void  OUTPUT(int pArr[])
void  OUTPUT(int *pArr)
{
        for ( int i = 0; i < COL; ++i)
                printf("%3d", pArr[i] );
        printf("₩n");
        return;
}
```

```
#include <stdio.h>
#define  ROW  5          // 행
#define  COL   4          // 열

// void  OUTPUT( int [] );
void OUTPUT( int * );
int main(void)
{
        int  table[ROW][COL] =
                {
                   {  0,  1,   2,   3 },
                   { 10, 11, 12,  13 },
                   { 20, 21, 22,  23 },
                   { 30, 31, 32,  33 },
                   { 40, 41, 42,  43 }
                };

        for(int i = 0; i < ROW; ++i)
            OUTPUT(table[i]);
        return 0;
}
```

# 2차원 배열과 함수 (3/3)

- 전체 배열 전달




```
// void  OUTPUT(int pTable[][COL])
void  OUTPUT(int(*pTable)[COL])
{
        for ( int i = 0; i < ROW; ++i) {
                for ( int j = 0; j < COL; ++j)
                    printf("%3d",  pTable[i][j] );
                printf("₩n");
        }
        return;
}
```

pTable

```
#include <stdio.h>

#define  ROW  5          // 행
#define  COL   4         // 열

// void  OUTPUT(int [][COL] ;
void OUTPUT(int (*)[COL]);

int main(void)
{
        int table[ROW][COL] =
                {
                  {  0,  1,  2,  3 },
                  { 10, 11, 12, 13 },
                  { 20, 21, 22, 23 },
                  { 30, 31, 32, 33 },
                  { 40, 41, 42, 43 }
                };

        OUTPUT( table );
        return 0;
}
```

# 정렬과 검색 알고리즘

CC BY NC ND

- 배열의 이해: 1차원 배열

백문이불여일타(百聞而不如一打)

- 문자 배열(문자열)
- 다차원 배열
- **정렬과 검색 알고리즘**
  - 선택.버블.삽입 정렬
  - 순차.이진 검색

# 정렬 (1/3)

## 정렬(Sort)

### 순서 없이 배열되어 있는 자료들을 재배열 하는 것

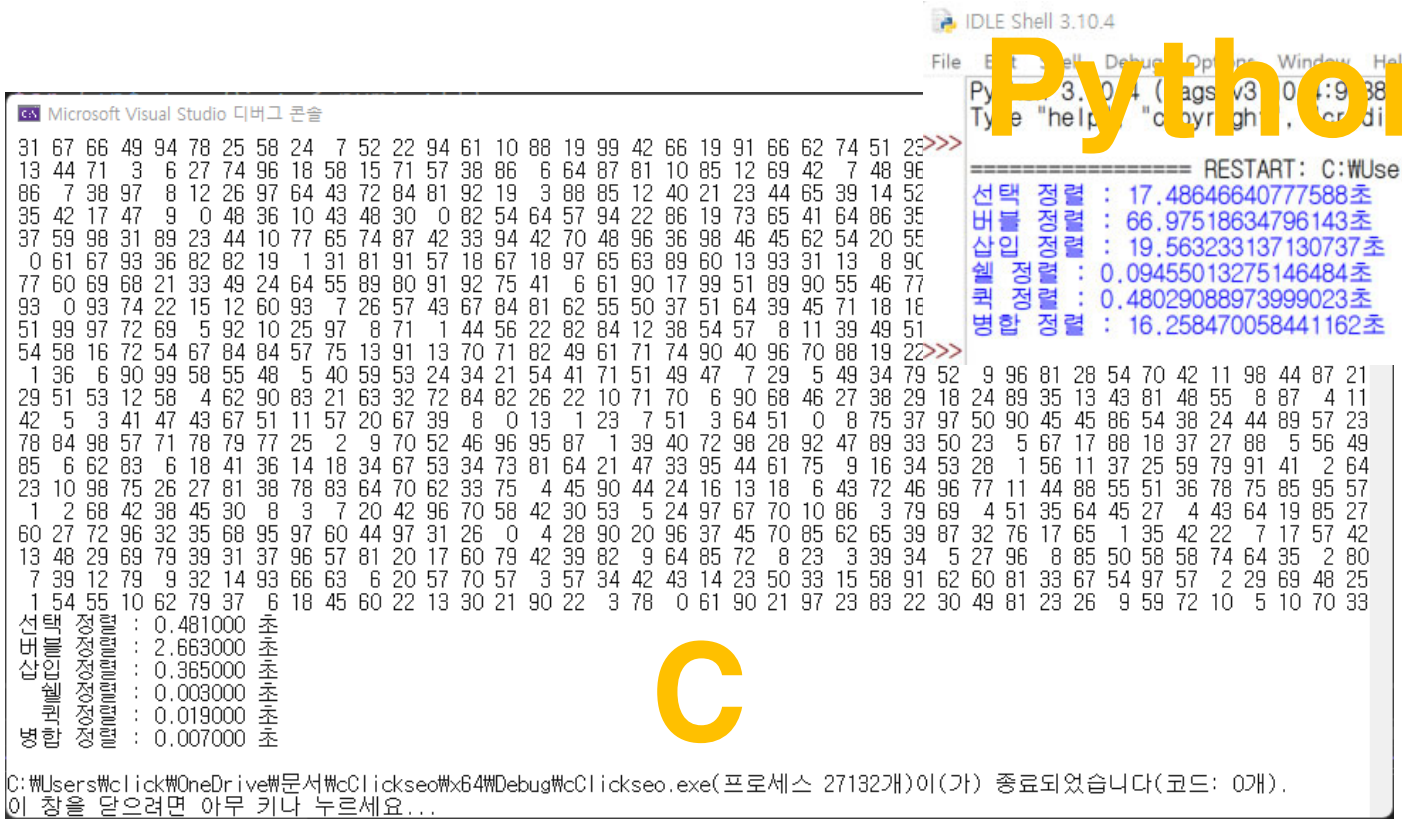
- **정렬의 대상:** 레코드
- **정렬의 기준:** 정렬 키(sort key) 필드

### 정렬 방법의 분류

- **실행 방법에 따른 분류:** 비교식 정렬, 분산식 정렬
- **정렬 장소에 따른 분류**
  - **내부 정렬:** 컴퓨터 메모리 내부에서 정렬
    - **정렬 속도는 빠르지만 자료의 양이 메인 메모리의 용량에 따라 제한된다.**
    - 교환방식, 삽입 방식, 병합 방식, 분배 방식, 선택 방식
  - **외부 정렬:** 메모리의 외부인 보조 기억 장치에서 정렬
    - **내부 정렬로 처리할 수 없는 대용량의 자료를 정렬**
    - 병합 방식: 2-way 병합, n-way 병합

# 정렬 (2/3)

- **정렬:** 알고리즘 성능 비교
  - 경험적 분석
    - 알고리즘을 프로그래밍 언어로 구현 후에 실행 시간을 비교해 보는 것



# 정렬 (3/3)

## 연습문제 7-9: 정렬 -- qsort | C

```c
#include <stdio.h>
#include <string.h>     // strcmp
#include <stdlib.h>     // qsort

int compare(const void *pa, const void *pb);

int main(void)
{
        int     nArr[] = { 5, 2, 7, 10, 1, 6, 3, 8, 9, 4 };
        int     arrSize = sizeof(nArr) / sizeof(*nArr);

        // qsort : 퀵 정렬 기반으로 구현, 시간 복잡도: O(NlogN)
        // void qsort(void* base, size_t num, size_t width,
        //                      int(__cdecl* compare)(const void*, const void*));
        qsort(nArr, arrSize, sizeof(nArr[0]), compare);

        // 정렬된 데이터 출력
        for (int i = 0; i < arrSize; ++i)
                printf("%3d", nArr[i]);
        printf("\n");

        return 0;
}

int compare(const void *pa, const void *pb) {
        return strcmp((char *)pa, (char*)pb);
}
```

```
Microsoft Visual Studio 디버그
  1  2  3  4  5  6  7  8  9 10

C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

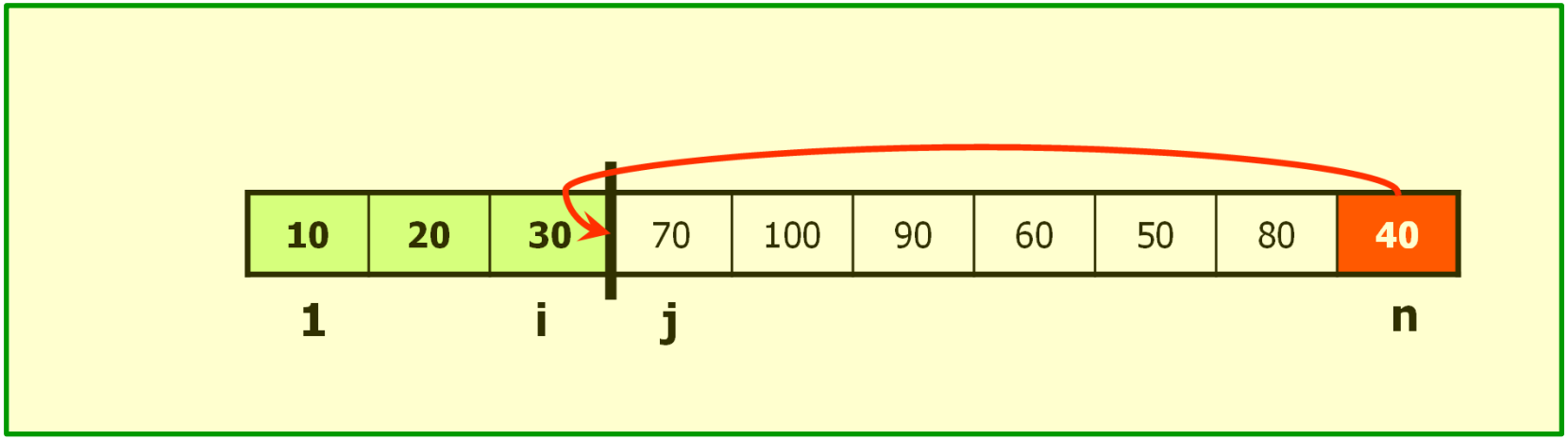
CC BY NC ND

# 기초적인 정렬과 검색 알고리즘

## 선택 정렬

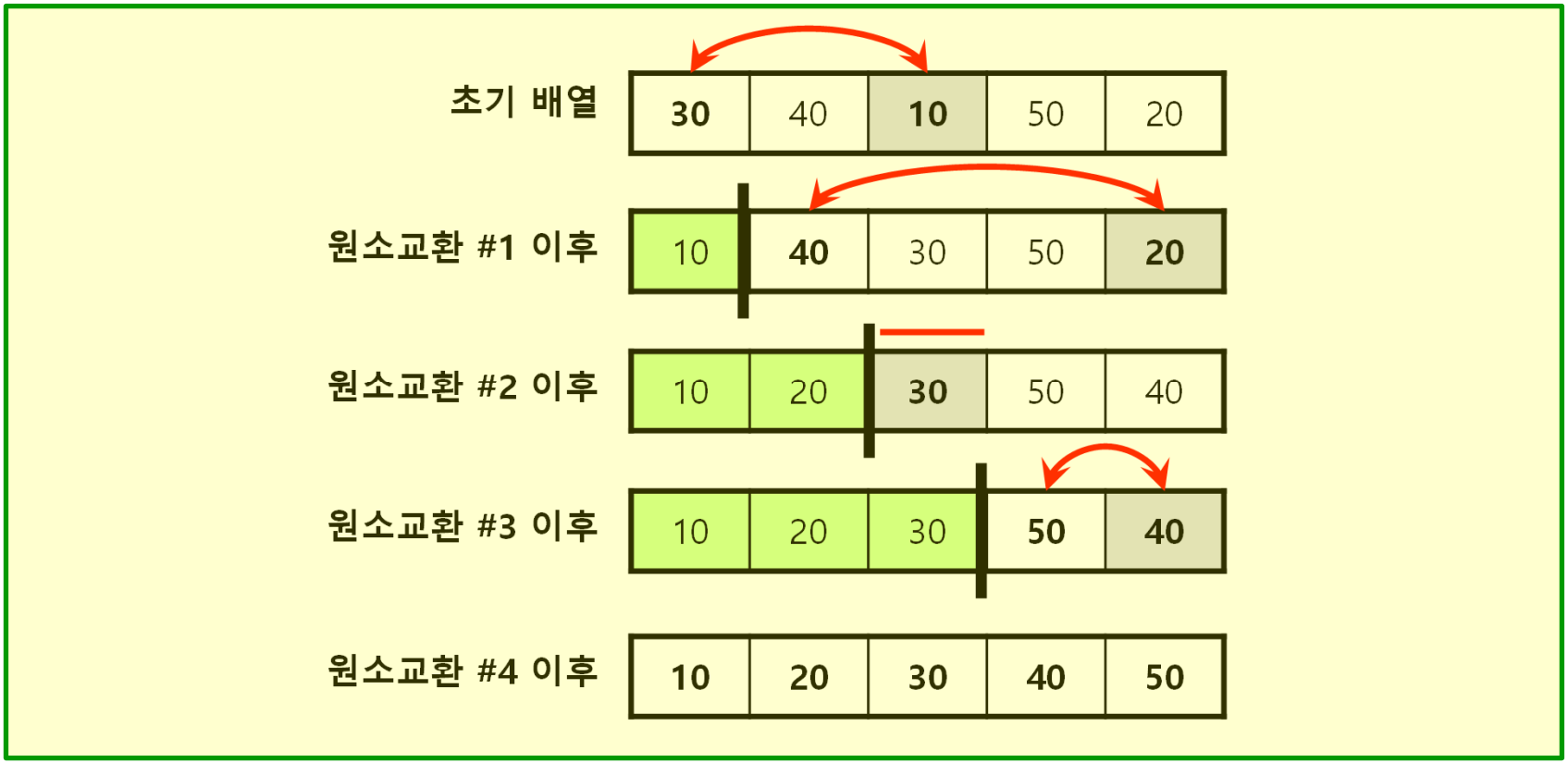
# 선택 정렬 (1/4)

- **선택 정렬(Selection Sort)**
  - **배열 원소에 대한 선택 정렬 과정**
    1. 먼저 정렬되지 않은 리스트에서 가장 작은 원소의 **위치 탐색**한다.
    2. 정렬되지 않은 리스트의 시작 위치에 있는 **원소와 교환**한다.
    3. 각각의 비교 및 교환 후에, 리스트의 경계를 한 개의 원소만큼 **이동한다.**



# 선택 정렬 (2/4)

## 선택 정렬 동작 과정




수행시간: $(n - 1) + (n - 2) + \ldots + 2 + 1 = O(N^2)$

Worst case

Average case

# 선택 정렬 (3/4)

- **선택 정렬:** 알고리즘

```
selectionSort( A[], N )              // 배열 A[ 1, ... , N ]을 정렬
{
    for i ← 1 to N – 1                                          ①
    {
            smallest ← i;
            for j ← i + 1 to N
                    if (A[j] < A[smallest]) then smallest ← j;  ②
            A[i] ↔ A[smallest];                                 ③
     }
}
```

**수행시간: 1 + 2 + ... + (N – 1) + N = $O(N^2)$**

① 의 **for** 루프는 **N - 1** 번 반복
② 에서 가장 작은 수를 찾기 위한 비교 횟수: **1, 2, ... , N - 1**
③ 의 교환은 상수 시간 작업

# 선택 정렬 (4/4)

- **선택 정렬: 알고리즘 분석**
  - **메모리 사용공간: N 개의 원소에 대하여 N 개의 메모리 사용**
  - **원소 비교 횟수**
    - **1 단계:** 첫 번째 원소를 기준으로 **N 개의 원소 비교**
    - **2 단계:** 두 번째 원소를 기준으로 마지막 원소까지 **N - 1 개의 원소 비교**
    - **3 단계:** 세 번째 원소를 기준으로 마지막 원소까지 **N - 2 개의 원소 비교**
    - **(생략)**
    - **i 단계: i** 번째 원소를 기준으로 **N - i 개의 원소 비교**

$$C_{min} = C_{ave} = C_{max} = (n-1) + (n-2) + \ldots + 1 = \sum_{i=1}^{n-1} i$$

$$\sum_{i=1}^{n-1} i = \frac{(n-1)+1}{2}(n-1) = \frac{1}{2}n(n-1) = \frac{1}{2}(n^2-n)$$

**어떤 경우에서나 원소 비교 횟수가 같기 때문에...**

**시간 복잡도는 O(N²)**

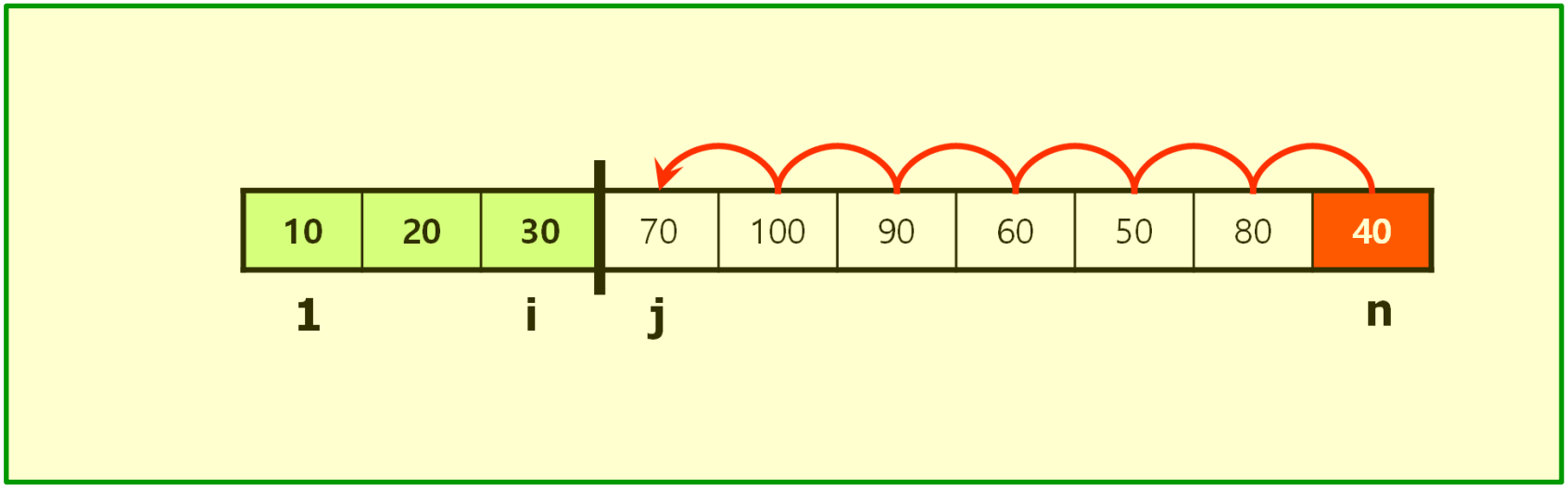
CC BY NC ND

# 기초적인 정렬과 검색 알고리즘

## 버블 정렬

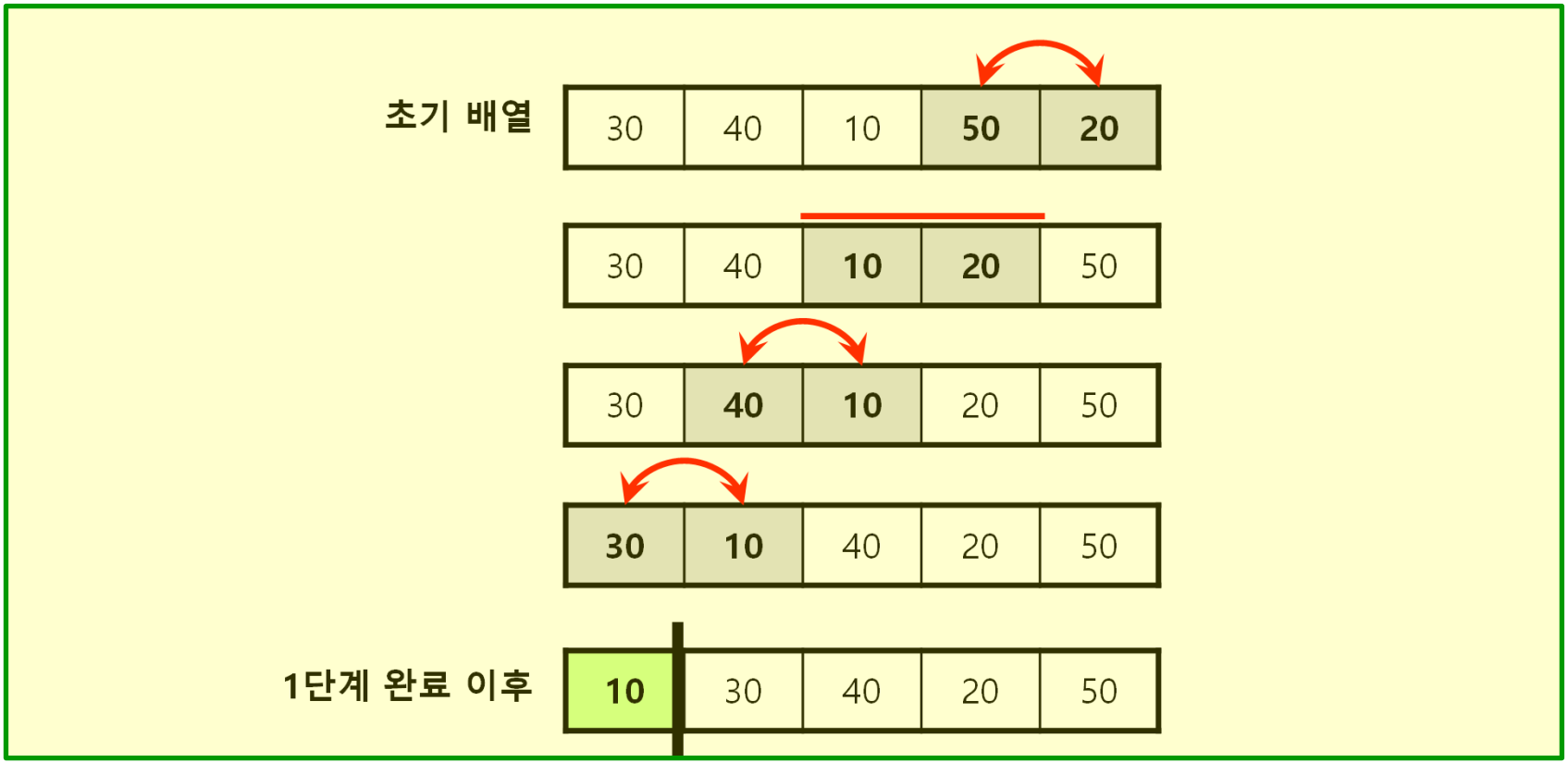
# 버블 정렬 (1/5)

- **버블 정렬(Bubble Sort)**
  - **배열 원소에 대한 버블 정렬 과정**
    1. 정렬되지 않은 리스트의 가장 작은 원소가 정렬된 서브 리스트로 이동한다.
    2. 각각의 비교 및 교환 후에 리스트의 경계를 한 개의 원소만큼 이동한다.



# 버블 정렬 (2/5)

## 버블 정렬 동작 과정




수행시간: 1 + 2 + ... + (N - 1) + N = O($N^2$)

Worst case

Average case

# 버블 정렬 (3/5)

- **버블 정렬:** 알고리즘

```
bubbleSort ( A[], N )                    // A[ 1 ... N ]을 정렬
{
    for i ← 1 to N - 1                                          ①
        for j ← N downto i - 1                                  ②
            if ( A[j] < A[j-1] ) then A[j] ↔ A[j-1];            ③
}
```

**수행시간: (N - 1) + (N - 2) + ⋯ + 2 + 1 = $O(N^2)$**

① 의 **for** 루프는 **N - 1** 번 반복

② 에서 가장 큰 수를 찾기 위한 비교 횟수: **N - 1, N - 2, ... , 2, 1**

③ 의 교환은 상수 시간 작업

# 버블 정렬 (4/5)

- **버블 정렬:** 변형(향상)된 알고리즘

```
bubbleSort ( A[], N )                    // A[ 1, ... , N ] 을 정렬
{
    for i ← 1 to N - 1
    {
            state ← TRUE;
            for j ← N downto i – 1
            {
                    if (A[j] < A[j-1]) then A[j] ↔ A[j-1];
                            state ← FALSE;
            }
             if (state = TRUE) then return;
    }
}
```

# 버블 정렬 (5/5)

- **버블 정렬:** 알고리즘 분석
  - 메모리 사용공간: **N 개의 원소에 대하여 N 개의 메모리 사용**
  - 연산 시간
    - **최선의 경우:** 자료가 이미 정렬되어 있는 경우
      - **원소 비교 횟수: i** 번째 원소를 **(N - i)** 번 비교하기 때문에 **N(N - 1)/2**
      - **원소 교환 횟수:** 자리교환이 발생하지 않는다.
    - **최악의 경우:** 자료가 역순으로 정렬되어 있는 경우
      - **원소 비교 횟수: i** 번째 원소를 **(N - i)** 번 비교하기 때문에 **N(N - 1)/2**
      - **원소 교환 횟수: i** 번째 원소를 **(N - i)** 번 교환하기 때문에 **N(N - 1)/2**

**평균 시간 복잡도는 $O(N^2)$**

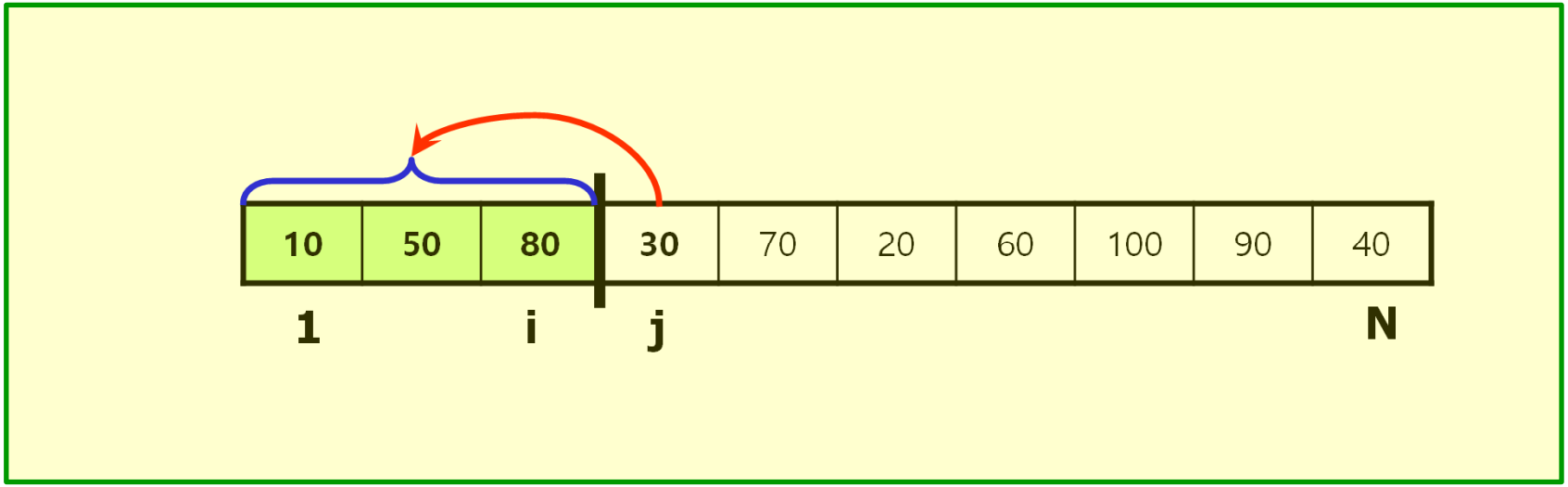
CC BY NC ND

# 기초적인 정렬과 검색 알고리즘

## 삽입 정렬

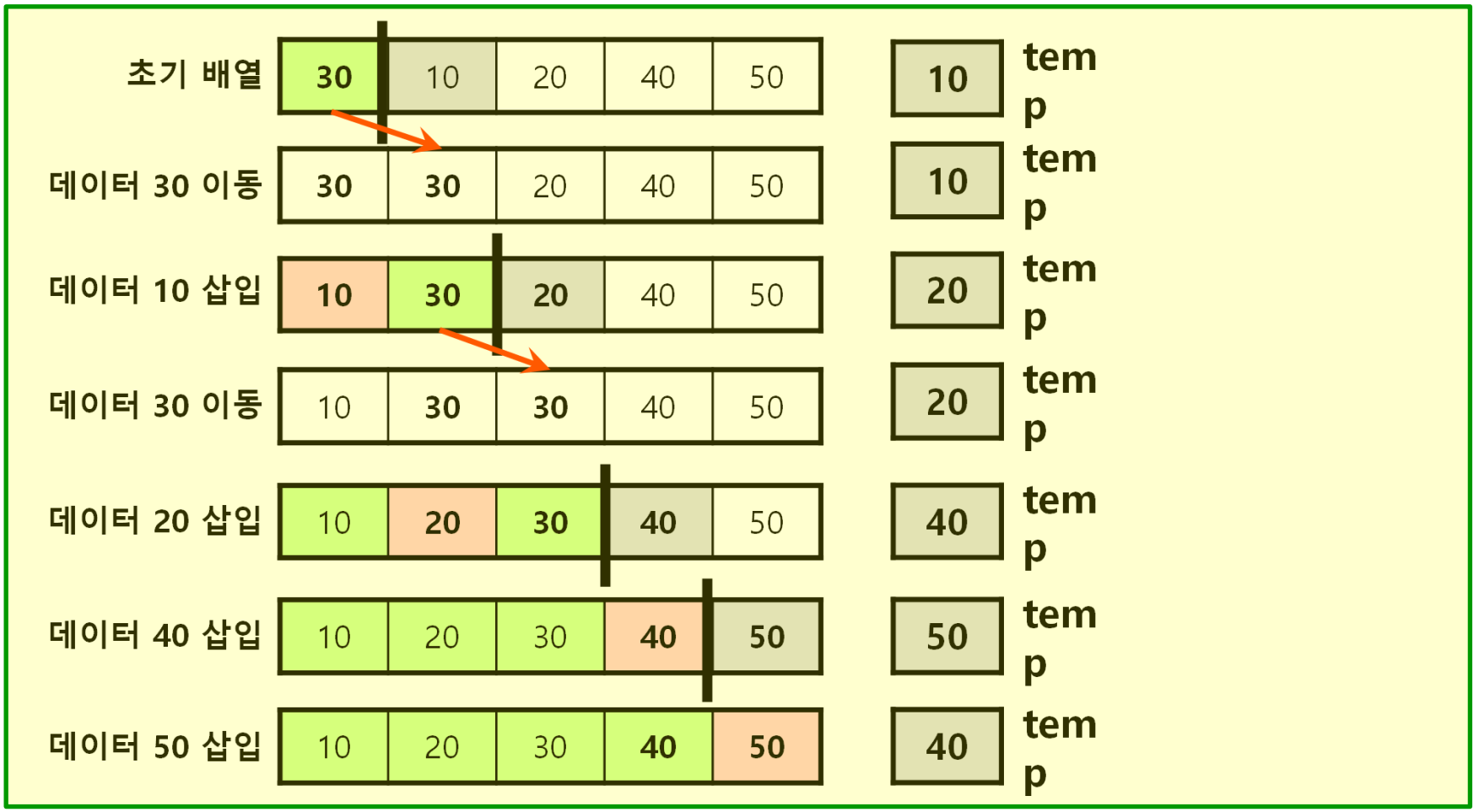
# 삽입 정렬 (1/4)

- **삽입 정렬(Insertion Sort)**
  - **배열 원소에 대한 삽입 정렬 과정**
    1. 각 단계에서 정렬되지 않은 리스트의 첫 번째 원소를 **선택**한다.
    2. 선택된 원소를 정렬된 리스트의 적절한 위치로 **삽입**한다.



삽입 정렬 동작 과정

# 삽입 정렬 (2/4)




수행시간: $O(N^2)$

Worst case: $1 + 2 + \cdots + (N-2) + (N-1)$

Average case: $\frac{1}{2}(1 + 2 + \cdots + (N-2) + (N-1))$

# 삽입 정렬 (3/4)

- **삽입 정렬**: 알고리즘

```
insertionSort ( A[], N )                    // A[ 1, ... , N ] 을 정렬
{
    for i ← 2 to N          ①
    {
        j ← i -1;
        temp ← A[i];
        while ( j ≥ 1 and temp < A[j] )    ②
        {
            A[j + 1] ← A[j];
            j--;
        }
        A[j + 1] ← temp;
    }
}
```

"배열이 거의 정렬되어 있는 상태 일 때 가장 매력적인 알고리즘"

**수행시간:**

① 의 **for** 루프는 **N - 1** 번 반복

② 의 삽입은 최악의 경우 **i - 1** 회 비교

**Worst case**: 1 + 2 + ... + (N - 2) + (N - 1) = **$O(N^2)$**

**Average case**: ½(1 + 2 + ... + (N - 2) + (N - 1)) = **$O(N^2)$**

# 삽입 정렬 (4/4)

- **삽입 정렬:** 알고리즘 분석
  - 메모리 사용 공간: **N 개의 원소에 대하여 N 개의 메모리 사용**
  - 연산 시간
    - **최선의 경우:** 원소들이 이미 정렬되어 있을 때 원소 비교 횟수가 최소
      - 이미 정렬되어 있는 경우에는 바로 앞자리 원소와 한번만 비교
      - **전체 원소 비교 횟수 = N - 1**
      - **시간 복잡도: O(N)**
    - **최악의 경우:** 모든 원소가 역순으로 되어있을 경우 원소 비교 횟수가 최대
      - **전체 원소 비교 횟수** = 1 + 2 + 3 + ⋯ + (N - 1) = **N(N - 1)/2**
      - **시간 복잡도: $O(N^2)$**
    - **삽입 정렬의 평균 원소 비교 횟수 = N(N - 1) / 4**

**평균 시간 복잡도는 $O(N^2)$**

CC BY NC ND

# 기초적인 정렬과 검색 알고리즘

**검색:** 순차.이진 검색

# 검색

## 검색(Search)

- 레코드의 집합에서 주어진 키를 지닌 레코드를 찾는 작업 탐색
  - 주어진 키 값: 목표 키(target key) 또는 검색 키(search key)

- 수행되는 위치에 따른 분류: 내부 검색, 외부 검색

- 탐색 방법에 따른 분류
  - **비교 검색:** 검색 대상의 키를 비교하여 검색
    - 순차 검색, 이진 검색, 트리 검색
  - **계산 검색:** 계수적 성질을 이용한 계산으로 검색
    - 해싱

# 순차 검색 (1/3)

- **순차 검색(Sequential Search): 선형 검색(Linear Search)**
  - **순차 검색 알고리즘**
    - 목표치를 찾기 위해 리스트의 처음부터 탐색을 시작해서, 목표치를 찾거나 리스트에 목표치가 없다는 것이 밝혀질 때까지 검색을 계속한다.
      - **순차 검색은 순서가 없는 리스트일 때 사용한다.**
      - 순차 검색은 리스트가 작거나, 가끔 한번씩 검색할 경우에만 사용한다.
    - **시간 복잡도: O(N)**

```
sequentialSearch( A[ ],  N,  key)
{
    i ← 0;
    while ( i < N )
    {
            if ( A[i] = key ) then
                    return  i;
            i ← i + 1;
    }

    return -1;
}
```

# 순차 검색 (2/3)

- **순차 검색:** 동작 과정 -- 검색 성공

목표 데이터: 73

- 순서 없는 리스트에 위치한 데이터

# 순차 검색 (3/3)

- **순차 검색:** 동작 과정 -- 검색 실패

목표 데이터: 90

  - 순서 없는 리스트에 검색 실패

# 이진 검색 (1/4)

- **이진 검색(Binary Search)**
  - 이진 검색은 배열이 정렬되어 있을 때 효율적인 알고리즘
  - 이진 검색 알고리즘의 조건
    - 검색할 데이터들은 정렬된 상태 이다.
    - 주어진 데이터들은 유일한 키 값을 가지고 있다.
  - 시간 복잡도: O(logN)

```
// 이진 검색: 재귀적 용법
binarySearch(A[ ],  first,  last,  key)
{
    if (first > last) then
            return -1;

    // 검색 범위의 중간 원소의 위치 값 계산
    mid ← (first + last) / 2;

    if (key = A[mid])        return mid;
    else if (key < A[mid])  then index ← binarySearch(A[], first, mid-1, key);
    else if (key > A[mid])  then index ← binarySearch(A[], mid+1, last, key);
    return index;
}
```

# 이진 검색 (2/4)

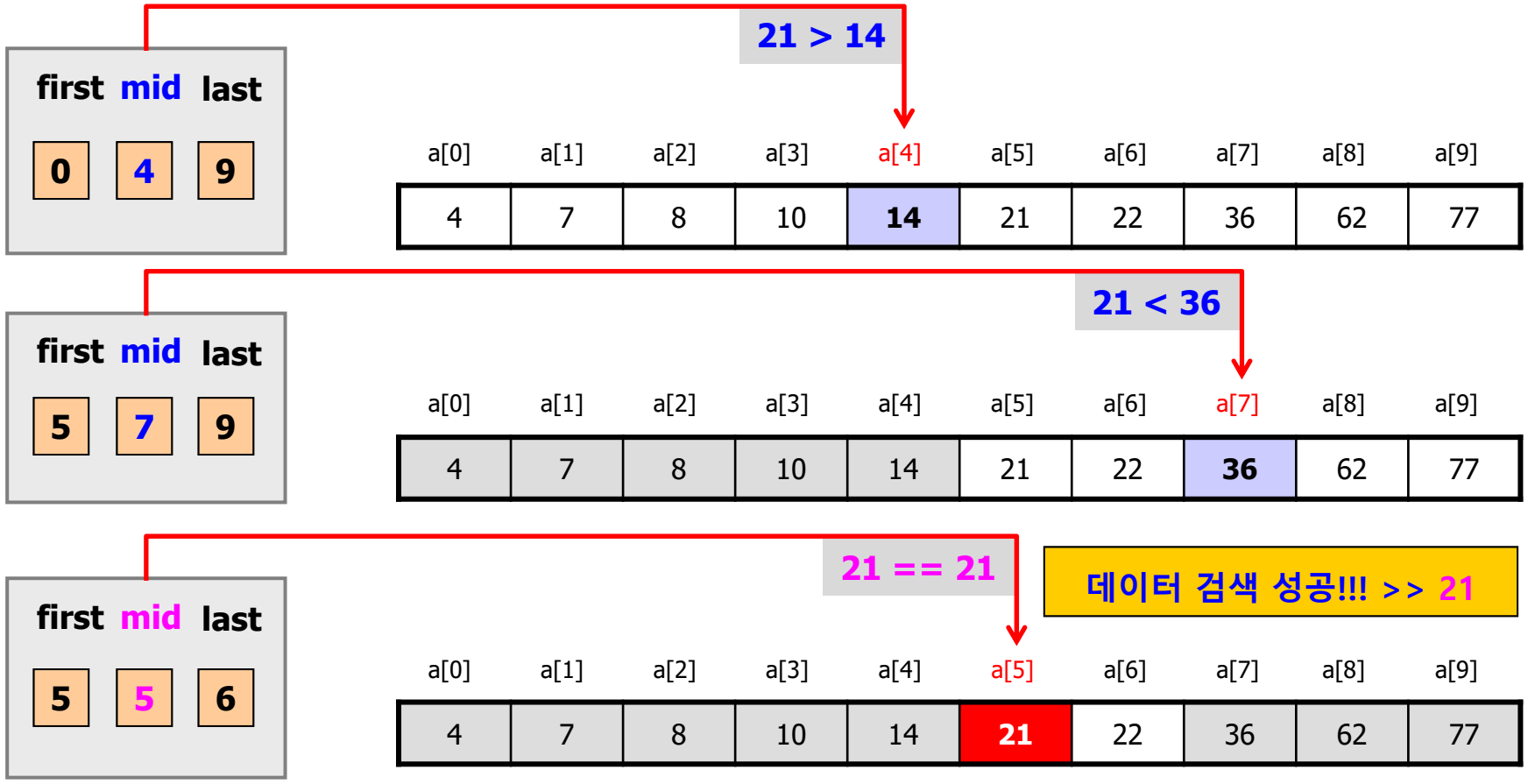
- **이진 검색(Binary Search): 비재귀적 용법**

```
// 이진 검색: 비재귀적 용법
binarySearch(A[ ], first, last, key)
{
    while (first <= last) then
    {
        // 검색 범위의 중간 원소의 위치 값 계산
        mid ← (first + last) / 2;

        if (key = A[mid])       return mid;
        else if (key < A[mid]) then  last ← mid-1;
        else if (key > A[mid]) then  first ← mid+1);
    }
    return -1;
}
```

# 이진 검색 (3/4)

- **이진 검색:** 동작 과정 -- 검색 성공

검색 데이터: 21



# 이진 검색 (4/4)

- **이진 검색: 동작 과정 -- 검색 실패**

검색 데이터: 11

Clickseo.com

# 참고문헌


[1] 서현우, “혼자 공부하는 C 언어: 1:1 과외 하듯 배우는 프로그래밍 자습서”, 한빛미디어, 2023.

[2] Paul Deitel, Harvey Deitel, "C How to Program", Global Edition, 8/E, Pearson, 2016.

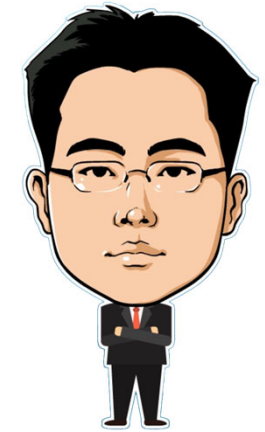
[3] Kamran Amini, 박지윤 번역, “전문가를 위한 C : 동시성, OOP부터 최신 C, 고급 기능까지!”, 한빛미디어, 2022.

[4] 서두옥, “(열혈강의) 또 하나의 C : 프로그래밍은 셀프입니다”, 프리렉, 2012.

[5] Behrouz A. Forouzan, Richard F. Gilberg, 김진 외 7인 공역, “구조적 프로그래밍 기법을 위한 C”, 도서출판 인터비젼, 2004.

[6] Brian W. Kernighan, Dennis M. Ritchie, 김석환 외 2인 공역, “The C Programming Language”, 2/E, 대영사, 2004.

[7] "C reference", cppreference.com, 2024 of viewing the site, https://en.cppreference.com/w/c.




이 강의자료는 저작권법에 따라 보호받는 저작물이므로 무단 전제와 무단 복제를 금지하며,
내용의 전부 또는 일부를 이용하려면 반드시 저작권자의 서면 동의를 받아야 합니다.

Copyright © Clickseo.com. All rights reserved.