

-Part2-

제4장 포인터와 배열

학습목차

4. 1 포인터와 1차원 배열

4. 2 포인터와 2차원 배열

4. 3 포인터 배열

4. 4 포인터와 문자 그리고 포인터와 문자열

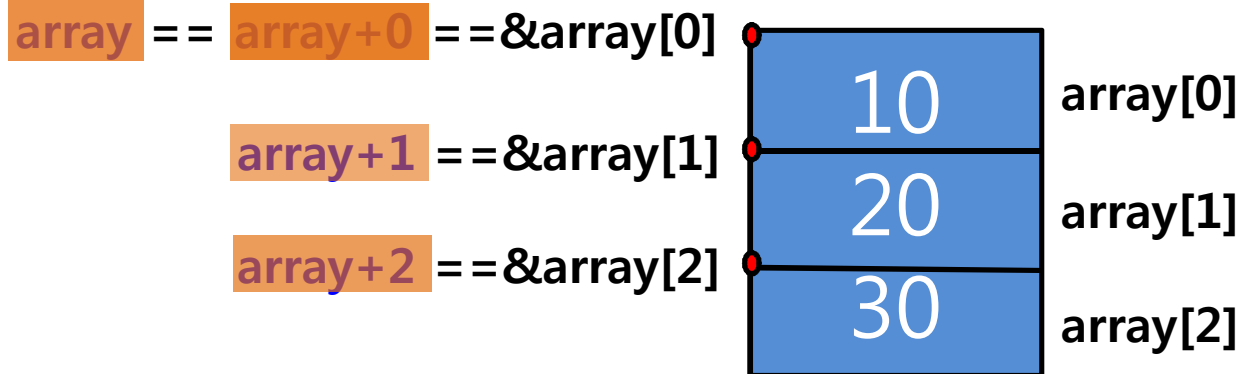
4.1 포인터와 1차원 배열

4.1 포인터와 1차원 배열 (1/16)---[4-1.c 실습]

```
#include <stdio.h>
int main(void)
{
    int array[3]={10, 20, 30};

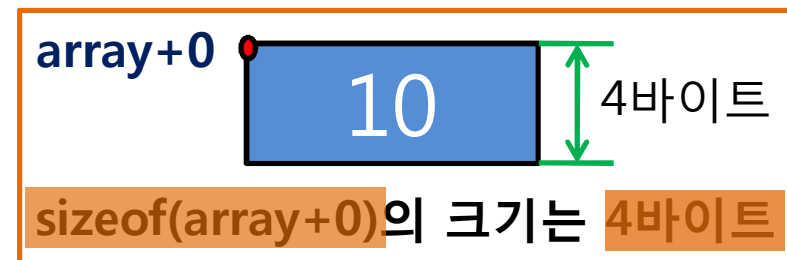
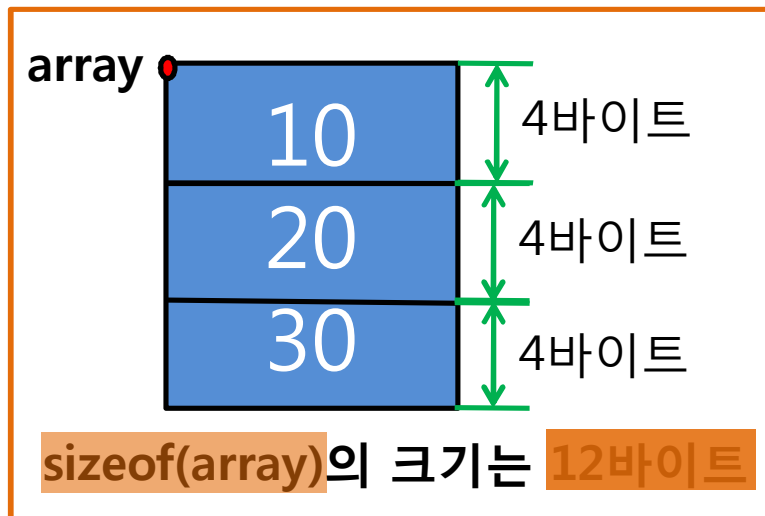
    printf("%x %x %x \n", array, array+0, &array[0]);
    printf("    %x %x \n",      array+1, &array[1]);
    printf("    %x %x \n",      array+2, &array[2]);
    printf("%d %d %d \n", sizeof(array), sizeof(array+0), sizeof(&array[0]));
    return 0;
}
```

배열의 이름은
'배열의 시작 주소'



4.1 포인터와 1차원 배열 (2/16)---[4-1.c 분석]

```
printf("%d %d %d \n", sizeof(array), sizeof(array+0), sizeof(&array[0]) );
```



4.1 포인터와 1차원 배열 (3/16)---[4-4.c 실습]

▶ 포인터 변수를 통한 1차원 배열 요소의 주소 접근

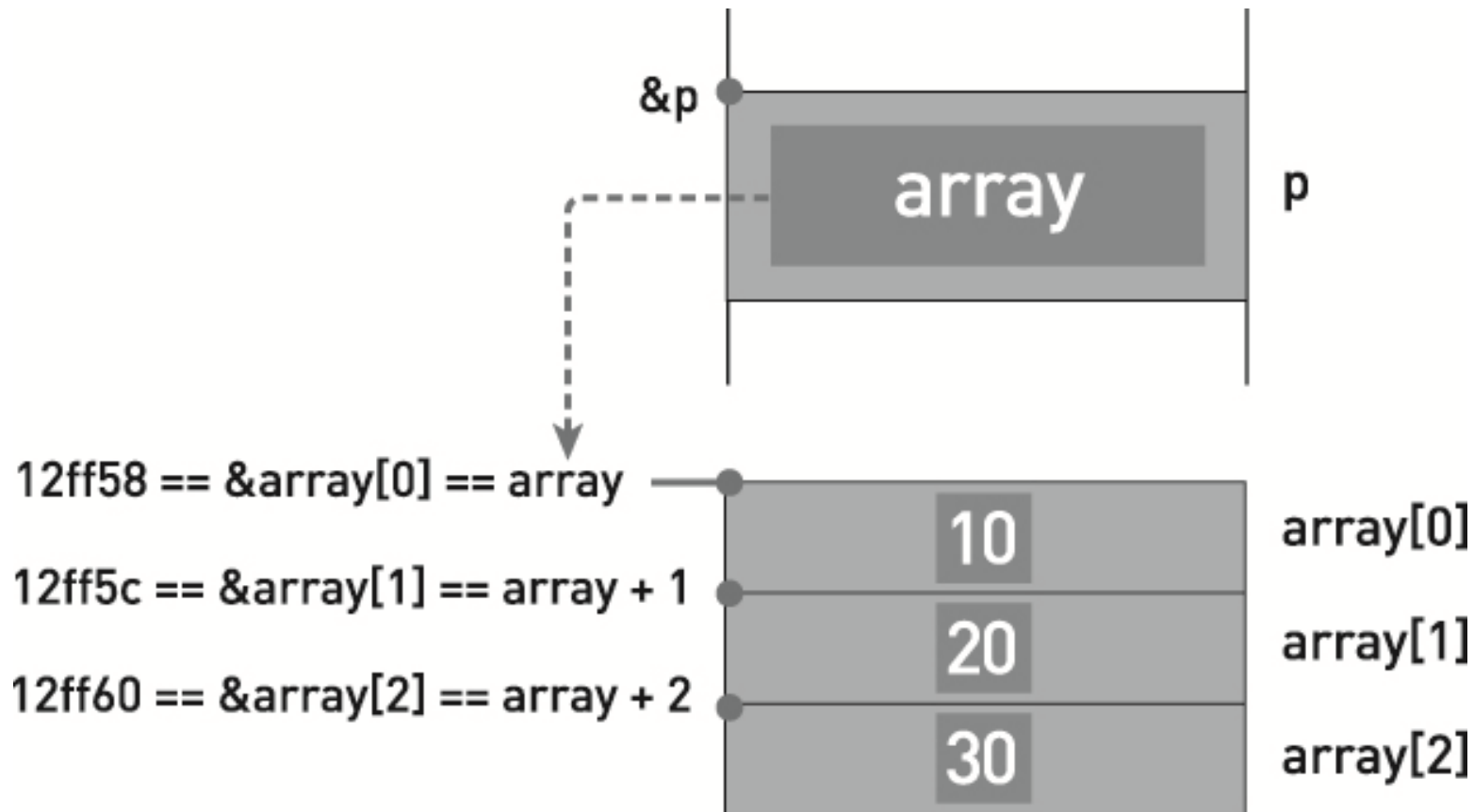
✓ 배열의 시작 주소를 저장

```
#include <stdio.h>
int main(void)
{
    int array[3]={10, 20, 30};
    int* p=NULL;

    p=array;      // p=&array[0];

    printf("%x %x %x \n", p, p+0, &p[0]);
    printf("%x %x \n",      p+1, &p[1]);
    printf("%x %x \n",      p+2, &p[2]);
    return 0;
}
```

4.1 포인터와 1차원 배열 (4/16)---[4-4.c 분석]



4.1 포인터와 1차원 배열 (5/16)---[4-5.c 실습]

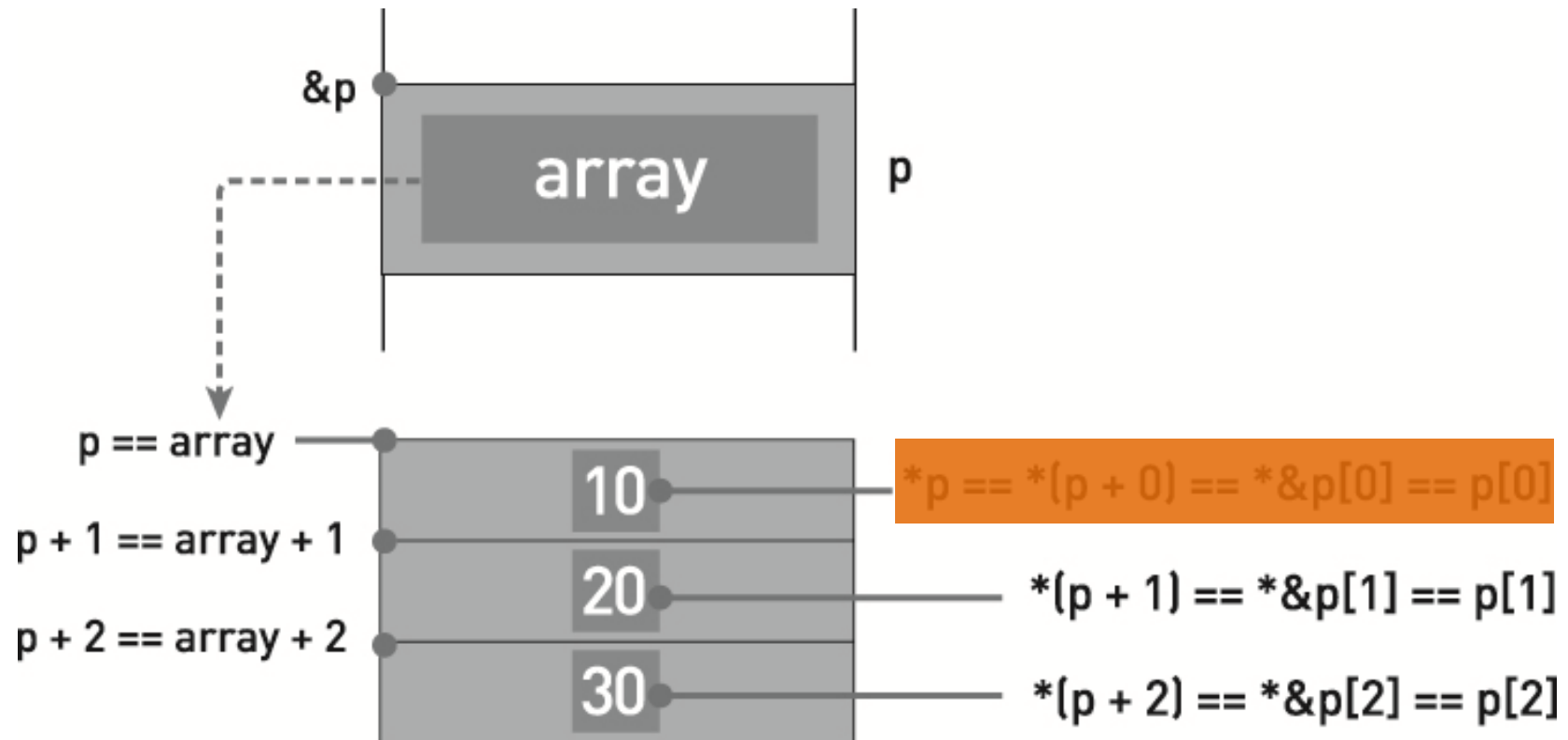
▶ 포인터 변수를 통한 1차원 배열 요소의 값 접근

✓ *연산자

```
#include <stdio.h>
int main(void)
{
    int array[3]={10, 20, 30};
    int* p=NULL;
    p=array;                // p=&array[0];

    // 주소에 *연산자를 붙임
    printf("%d %d %d \n", *p, *(p+0), *&p[0]);    // *&는 서로 상쇄
    printf("%d %d \n",      *(p+1), *&p[1]);        // *&는 서로 상쇄
    printf("%d %d \n",      *(p+2), *&p[2]);        // *&는 서로 상쇄
    return 0;
}
```


4.1 포인터와 1차원 배열 (6/16)---[4-5.c 분석]



4.1 포인터와 1차원 배열 (7/16)---[4-6.c 실습]

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int array[3]={10, 20, 30};
```

```
    int i=0;
```

```
    int* p=NULL;
```

```
    p=array;          // p=&array[0];
```

```
    for(i=0; i<3; i++)
```

```
        printf("%d %d %d \n", *(p+i), *&p[i], p[i] );
```

```
    printf("-----\n");
```

```
    for(i=0; i<3; i++)
```

```
        printf("%d %d %d \n", *(array+i), *&array[i], array[i]);
```

```
    return 0;
```

```
}
```

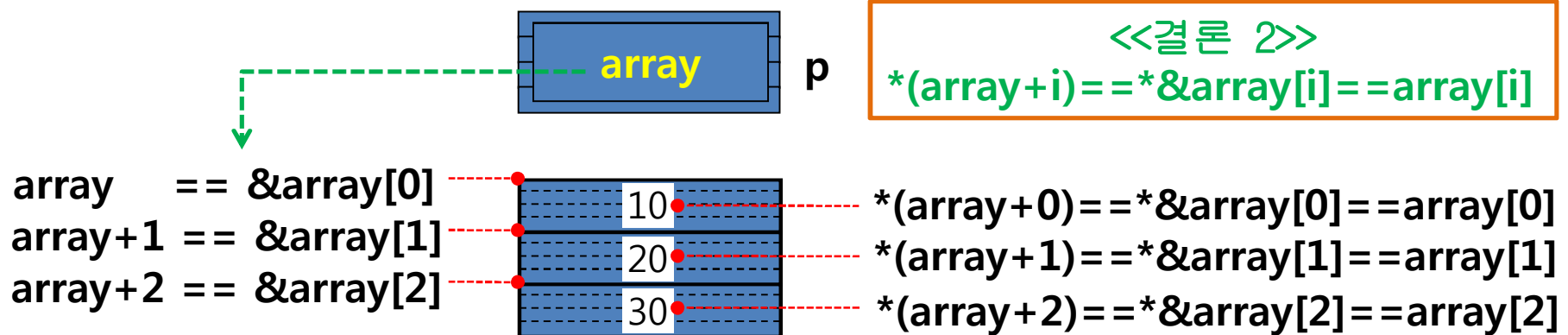
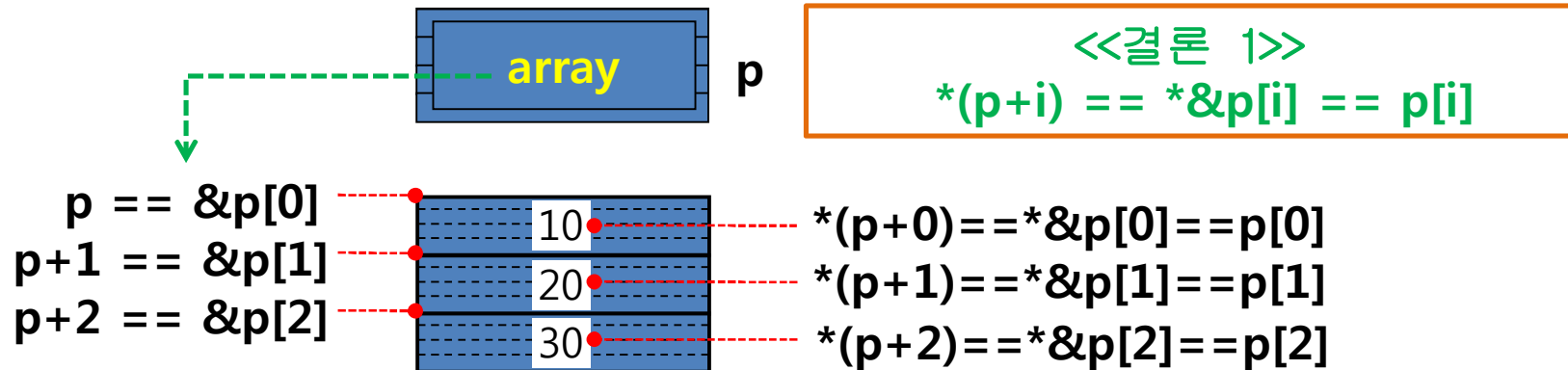
<<결론 1>>

$*(p+i) == *&p[i] == p[i]$

<<결론 2>>

$*(array+i) == *&array[i] == array[i]$

4.1 포인터와 1차원 배열 (8/16)---[4-6.c 분석]



4.1 포인터와 1차원 배열 (9/16)---[4-7.c 실습]

▶ 포인터 변수와 배열의 크기 차이

✓ 포인터 변수 - 4바이트로 고정, 배열 - 배열 길이에 따라 가변적

```
#include <stdio.h>
int main(void)
{
    int array[3]={10, 20, 30};
    int* p=NULL;

    p=array;    // 포인터 변수에 배열의 시작 주소 저장

    printf("%d %d %d \n",   array[0],   array[1],   array[2]);
    printf("%d %d %d \n", *(array+0), *(array+1), *(array+2));
    printf("%d %d %d \n",   p[0],   p[1],   p[2]);
    printf("%d %d %d \n", *(p+0), *(p+1), *(p+2));

    printf("배열의 크기 : %d 포인터의 크기 : %d \n", sizeof(array), sizeof(p));
    return 0;
}
```

12바이트

4바이트

4.1 포인터와 1차원 배열 (10/16)---[4-8.c 실습]

▶ 주소의 가감산을 이용한 배열의 접근

...

```
int array[3]={10, 20, 30};  
int* p=NULL;
```

7행

```
p=array;    // p=&array[0];  
printf("%d %d %d \n", p[0], p[1], p[2]);  
printf("%d %d %d \n", *p, *(p+1), *(p+2));
```

11행

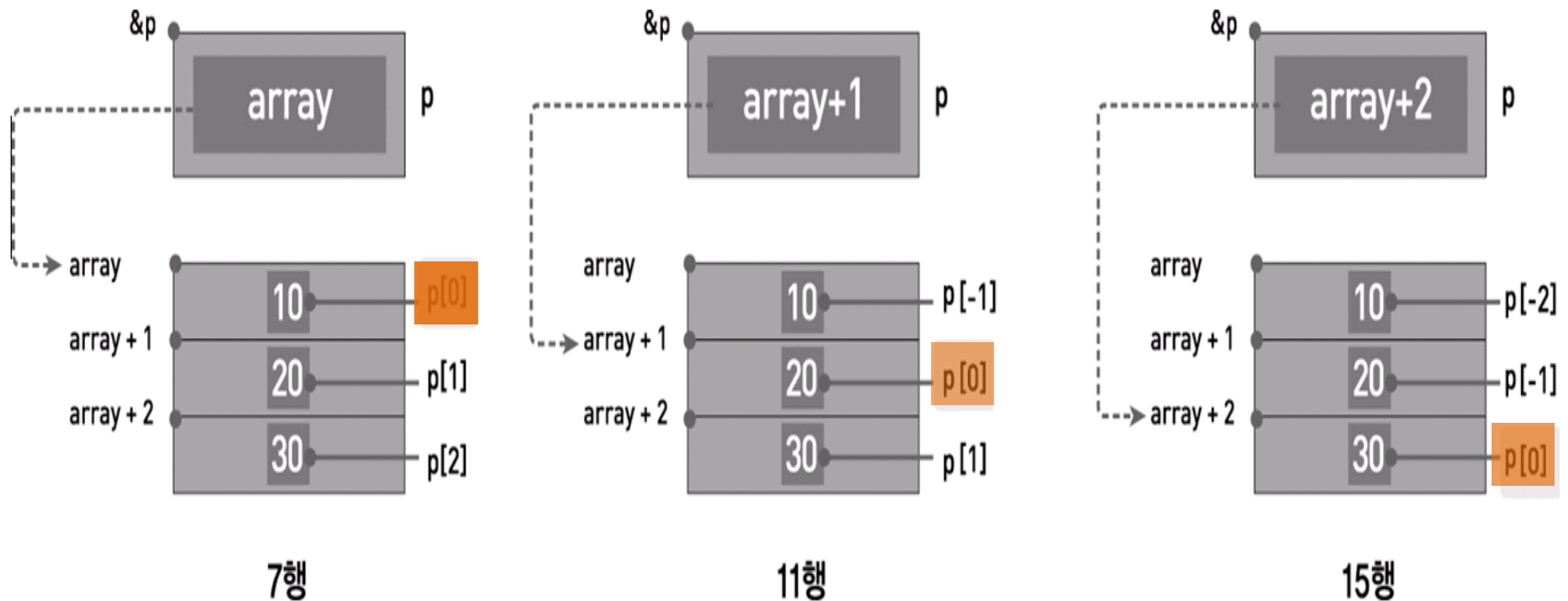
```
p=array+1;  // p=&array[1];  
printf("%d %d %d \n", p[-1], p[0], p[1]);  
printf("%d %d %d \n", *(p-1), *p, *(p+1));
```

15행

```
p=array+2;  // p=&array[2];  
printf("%d %d %d \n", p[-2], p[-1], p[0]);  
printf("%d %d %d \n", *(p-2), *(p-1), *p);
```

...

4.1 포인터와 1차원 배열 (11/16)---[4-8.c 분석]



4.1 포인터와 1차원 배열 (12/16)---[4-9.c 실습]

...

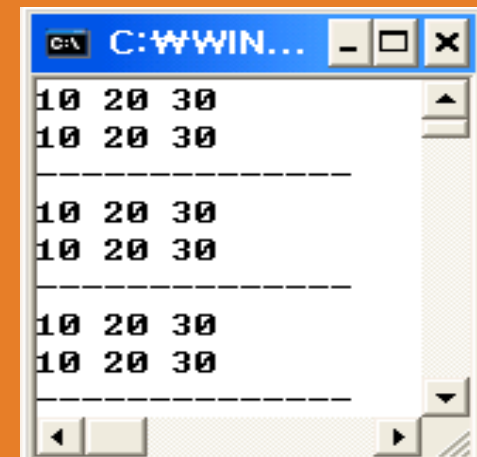
```
int array[3]={10, 20, 30};  
int* p=NULL;
```

```
p=array;           // p=&array[0];  
printf("%d %d %d \n", p[0], p[1], p[2]);  
printf("%d %d %d \n", *p, *(p+1), *(p+2));  
printf("-----\n");
```

```
p=p+1;           // p=&array[1];  
printf("%d %d %d \n", p[-1], p[0], p[1]);  
printf("%d %d %d \n", *(p-1), *p, *(p+1));  
printf("-----\n");
```

```
p=p+1;           // p=&array[2];  
printf("%d %d %d \n", p[-2], p[-1], p[0]);  
printf("%d %d %d \n", *(p-2), *(p-1), *p); 0)  
printf("-----\n");  
return 0;
```

...



```
C:\WIN...  
10 20 30  
10 20 30  
-----  
10 20 30  
10 20 30  
-----  
10 20 30  
10 20 30  
-----
```

4.1 포인터와 1차원 배열 (13/16)---[4-10.c 실습]

7행

```
...  
int array[3];  
int* p=NULL;  
p=array;
```

9행

```
*p = 10;  
printf("%d %d %d \n", p[0], p[1], p[2]);  
printf("-----\n");
```

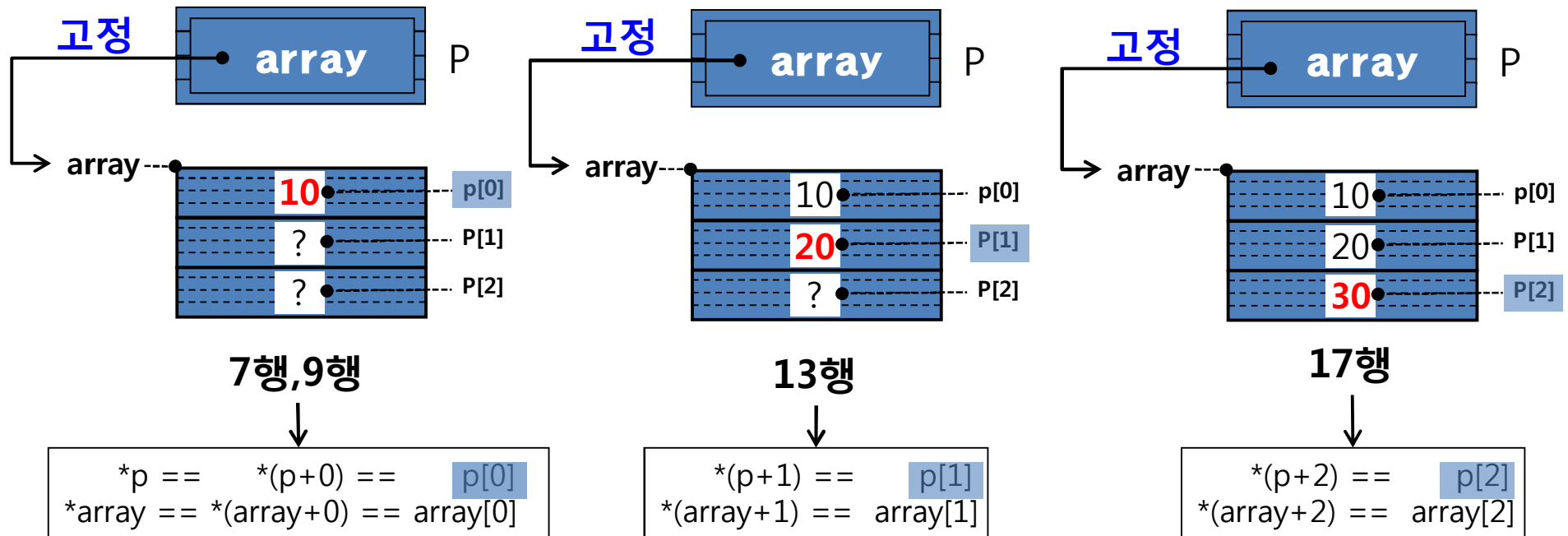
13행

```
*(p+1)=20;  
printf("%d %d %d \n", p[0], p[1], p[2]);  
printf("-----\n");
```

17행

```
*(p+2)=30;  
printf("%d %d %d \n", p[0], p[1], p[2]);  
printf("-----\n");  
  
printf("%d %d %d \n", *p, *(p+1), *(p+2));  
printf("%d %d %d \n", p[0], p[1], p[2]);  
printf("-----\n");
```


4.1 포인터와 1차원 배열 (14/16)---[4-10.c 분석]



4.1 포인터와 1차원 배열 (15/16)---[4-11.c 실습]

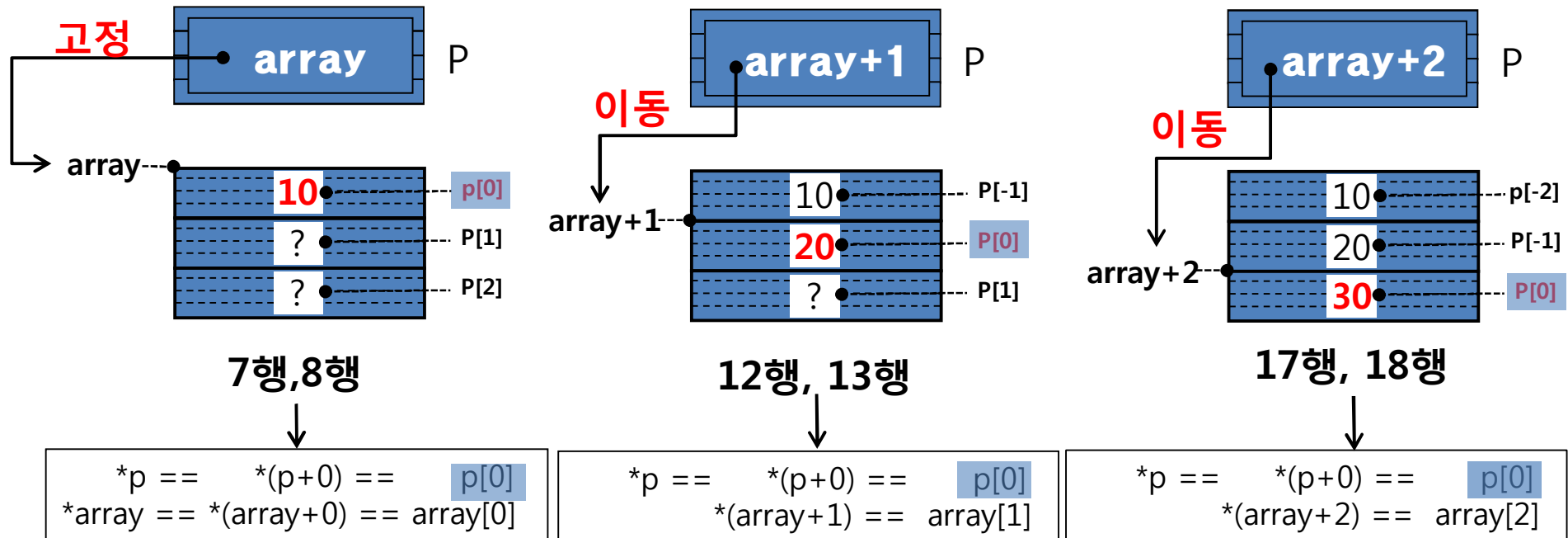
```
...  
int array[3];  
int* p=NULL;  
7행 p=array;  
8행 *p=10;  
printf("%d %d %d \n", p[0], p[1], p[2]);  
printf("-----\n");
```

```
12행 p=p+1;  
13행 *p=20; // p[0]=20;  
printf("%d %d %d \n", p[-1], p[0], p[1]);  
printf("-----\n");
```

```
17행 p=p+1;  
18행 *p=30; // p[0]=30;  
printf("%d %d %d \n", p[-2], p[-1], p[0]);  
printf("-----\n");
```

```
printf("%d %d %d \n", p[-2], p[-1], p[0]);  
printf("%d %d %d \n", *(p-2), *(p-1), *p);
```

4.1 포인터와 1차원 배열 (16/16)---[4-11.c 분석]



4.2 포인터와 2차원 배열

4.2 포인터와 2차원 배열 (1/17)

▶ '2차원 배열에서 `array[i] == *(array+i)`는 주소이다.'

✓ 복습(PART2 - 2장)

서로 상쇄
↓

`*(array+i) == array[i] == *&array[i]`

- 1차원 배열 : `*(array+i) == array[i] == *&array[i]`는 값
- 2차원 배열 : `*(array+i) == array[i] == *&array[i]`는 주소

4.2 포인터와 2차원 배열 (2/17)---[4-12.c 실습]

```
#include <stdio.h>
int main(void)
{
    int array[3][3]={10,20,30,40,50,60,70,80,90};

    printf("%x %x %x \n", &array[0][0], &array[0][1], &array[0][2]); // 주소 출력
    printf("%x %x %x \n", &array[1][0], &array[1][1], &array[1][2]); // 주소 출력
    printf("%x %x %x \n", &array[2][0], &array[2][1], &array[2][2]); // 주소 출력
    printf("-----\n");

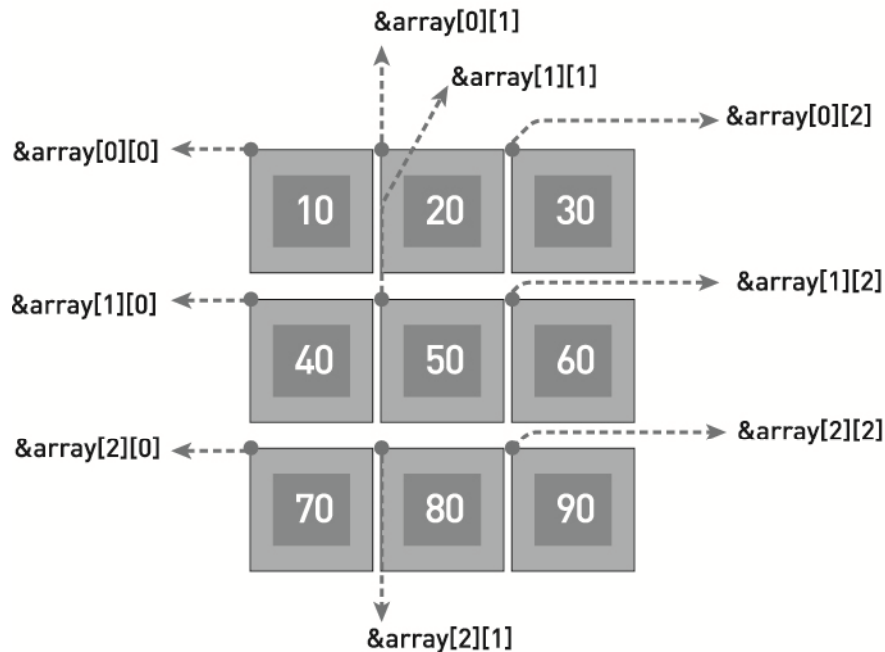
    printf("%d %d %d \n", *&array[0][0], *&array[0][1], *&array[0][2]); // 값 출력
    printf("%d %d %d \n", *&array[1][0], *&array[1][1], *&array[1][2]); // 값 출력
    printf("%d %d %d \n", *&array[2][0], *&array[2][1], *&array[2][2]); // 값 출력
    printf("-----\n");

    printf("%d %d %d \n", array[0][0], array[0][1], array[0][2]); // *& 서로 상쇄
    printf("%d %d %d \n", array[1][0], array[1][1], array[1][2]); // *& 서로 상쇄
    printf("%d %d %d \n", array[2][0], array[2][1], array[2][2]); // *& 서로 상쇄

    return 0;
}
```

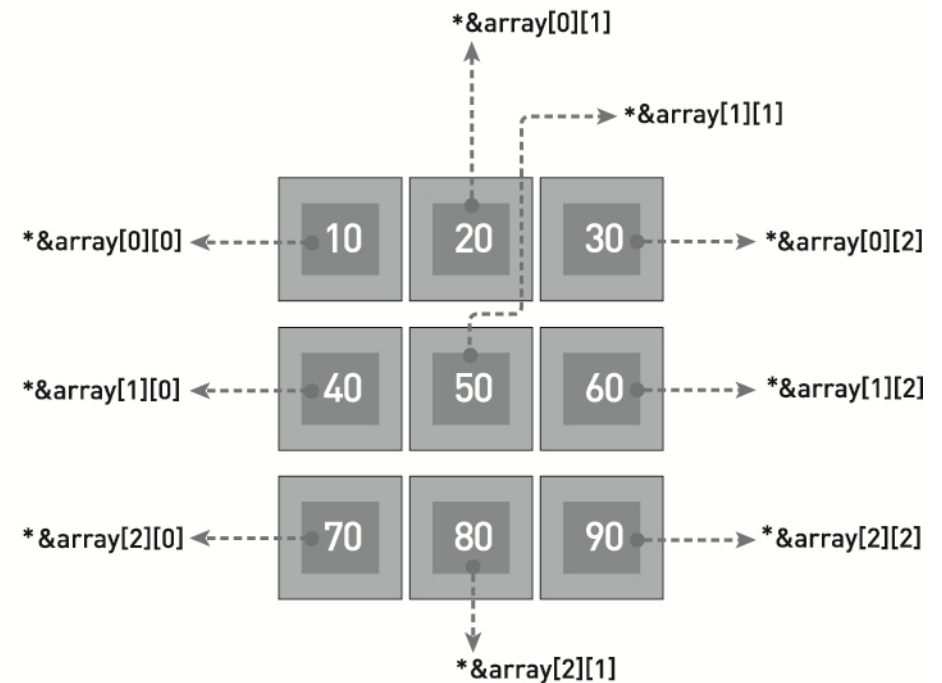
4.2 포인터와 2차원 배열 (3/17)---[4-12.c 분석]

```
int array[3][3]={10, 20, 30, 40, 50, 60, 70, 80, 90}
```



```
&array[0][0], &array[0][1], &array[0][2]
&array[1][0], &array[1][1], &array[1][2]
&array[2][0], &array[2][1], &array[2][2]
```

6행~9행 : 배열 요소들의 주소 표현



```
*&array[0][0], *&array[0][1], *&array[0][2]
*&array[1][0], *&array[1][1], *&array[1][2]
*&array[2][0], *&array[2][1], *&array[2][2]
```

11행~14행 : 배열에 저장된 값 표현

4.2 포인터와 2차원 배열 (4/17)---[4-13.c 실습]

```
...
int array[2][3] = {10,20,30,40,50,60};
printf("-----#1-----\n");
printf("%x %x %x \n", &array[0][0], &array[0][1], &array[0][2]);
printf("%x %x %x \n", &array[1][0], &array[1][1], &array[1][2]);

printf("-----#2-----\n");
printf("%x %x \n", array, array+1);
printf("%x %x \n", array[0], array[1]);
printf("%x %x \n", *(array+0), *(array+1));

printf("-----#3-----\n");
printf("%d %d %d \n", *(array[0]+0), *(array[0]+1), *(array[0]+2));
printf("%d %d %d \n", *(array[1]+0), *(array[1]+1), *(array[1]+2));

printf("-----#4-----\n");
printf("%d %d %d \n", (*(array+0)+0), (*(array+0)+1), (*(array+0)+2));
printf("%d %d %d \n", (*(array+1)+0), (*(array+1)+1), (*(array+1)+2));
...
```

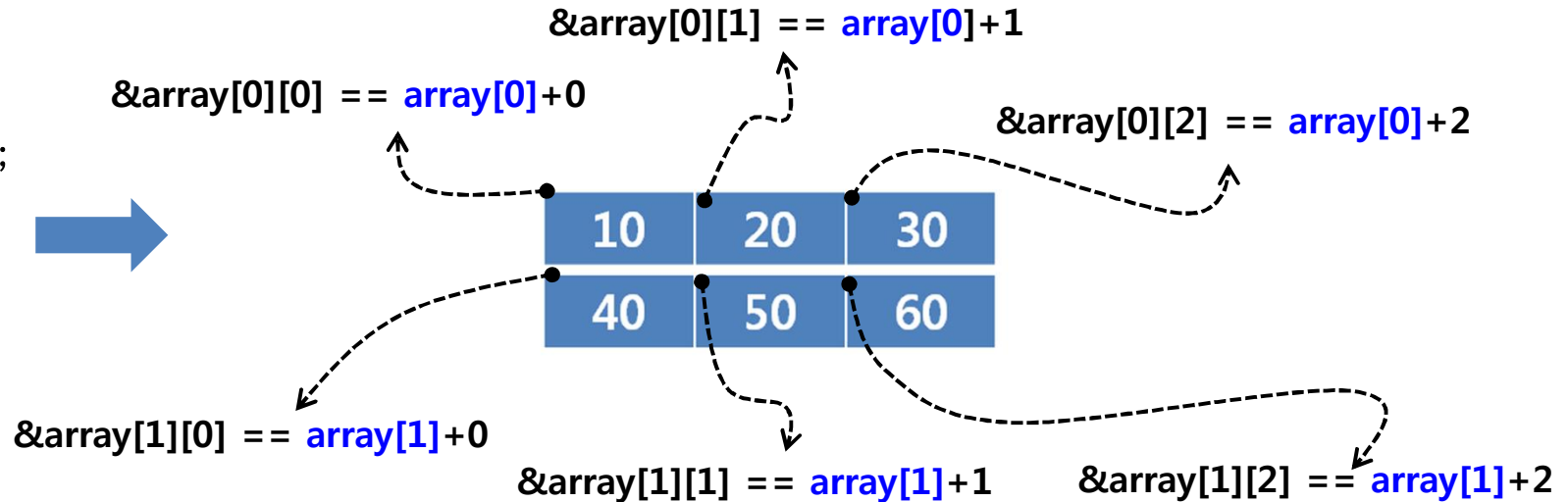
PART2-2장 (복습)

array[i] == *(array+i)

4.2 포인터와 2차원 배열 (5/17)---[4-13.c 분석]

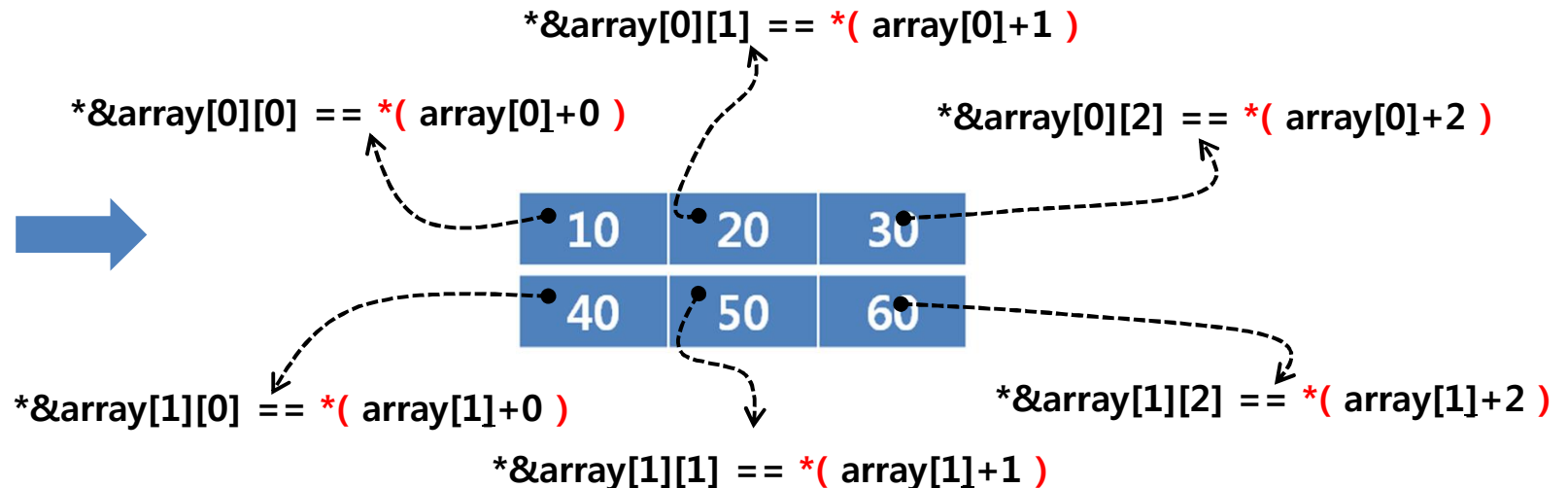
int array[2][3];

주소 표현



int array[2][3];

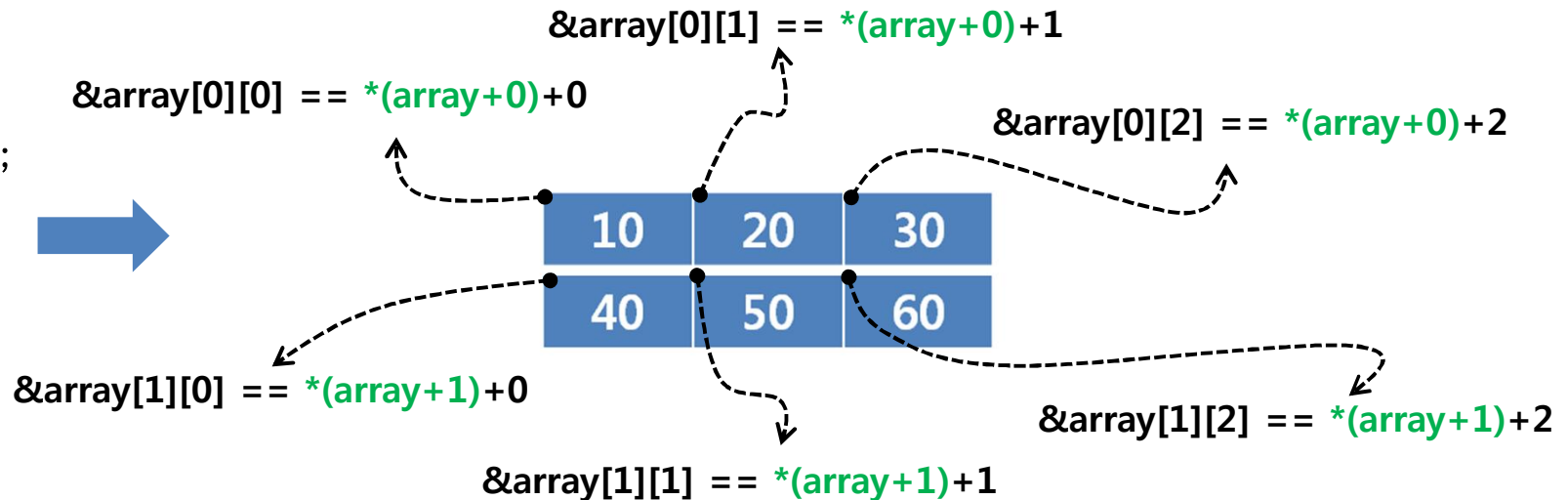
값 표현



4.2 포인터와 2차원 배열 (6/17)---[4-13.c 분석]

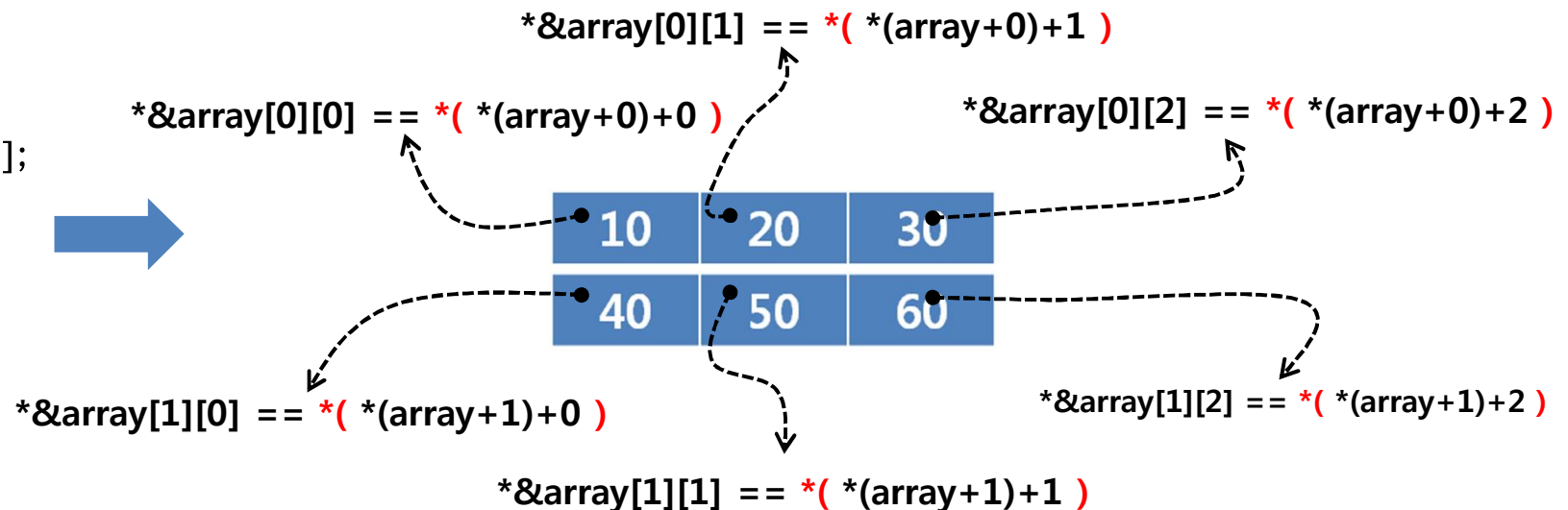
int array[2][3];

주소 표현



int array[2][3];

값 표현

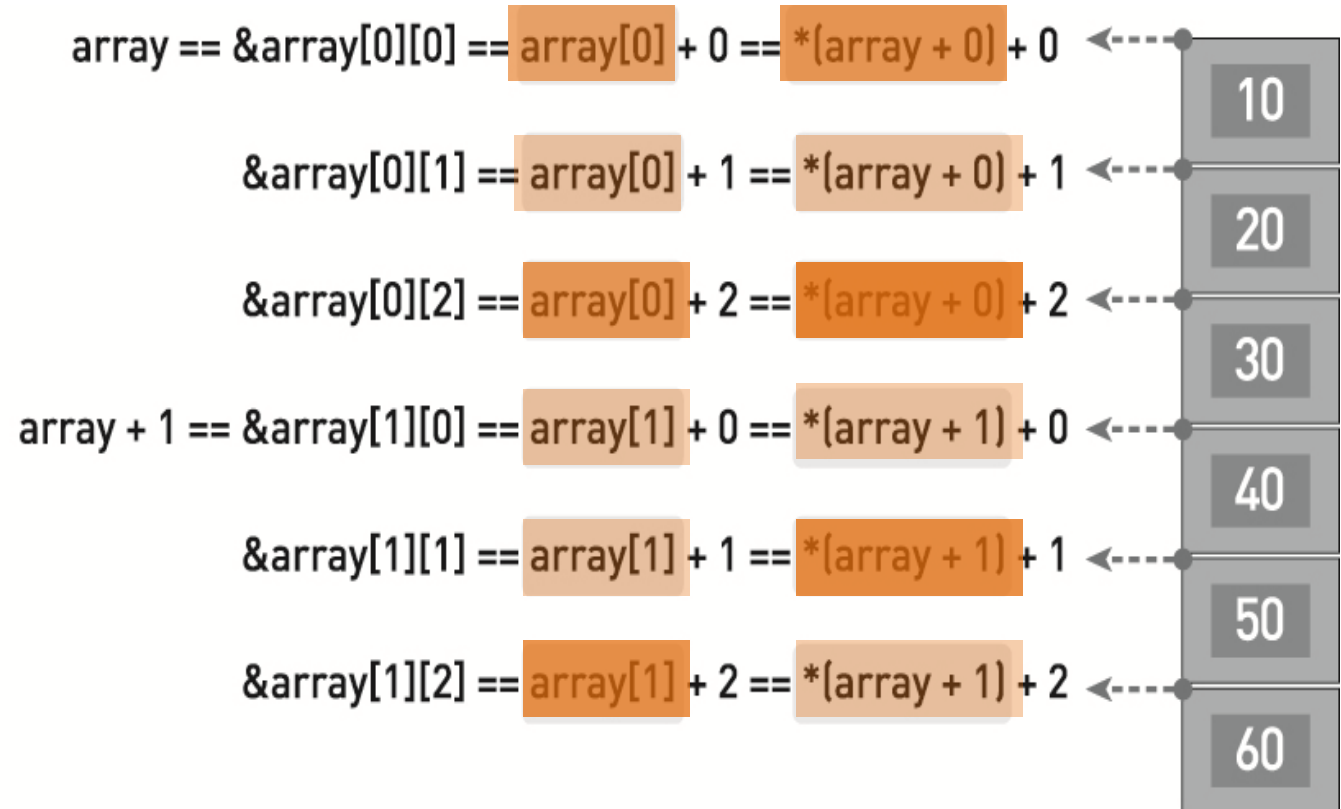


4.2 포인터와 2차원 배열 (7/17)---[4-13.c 분석]

▶ 2차원 배열의 물리적 메모리 구조

int array[2][3]

주소 표현



4.2 포인터와 2차원 배열 (8/17)---[4-14.c 실습]

▶ 포인터 변수를 통한 2차원 배열의 접근

✓ 2차원 배열의 시작 주소를 저장

```
#include <stdio.h>
int main(void)
{
    int array[2][3]={10,20,30,40,50,60};
    int* p=NULL;

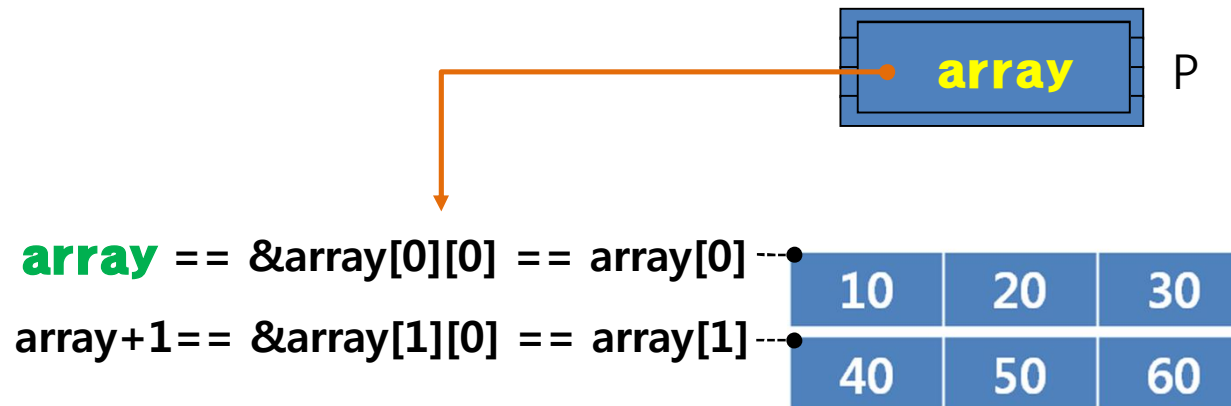
    p=array;           // p=&array[0][0]; // p=array[0]; // 포인터 변수에 시작 주소 저장

    printf("-----\n");
    printf("%x %x %x \n", &p[0], &p[1], &p[2]);
    printf("%x %x %x \n", &p[3], &p[4], &p[5]);

    printf("-----\n");
    printf("%d %d %d \n", p[0], p[1], p[2]); // printf("%d %d %d \n", *(p+0), *(p+1), *(p+2));
    printf("%d %d %d \n", p[3], p[4], p[5]); // printf("%d %d %d \n", *(p+3), *(p+4), *(p+5));
    return 0;
}
```

4.2 포인터와 2차원 배열 (9/17)---[4-14.c 분석 1]

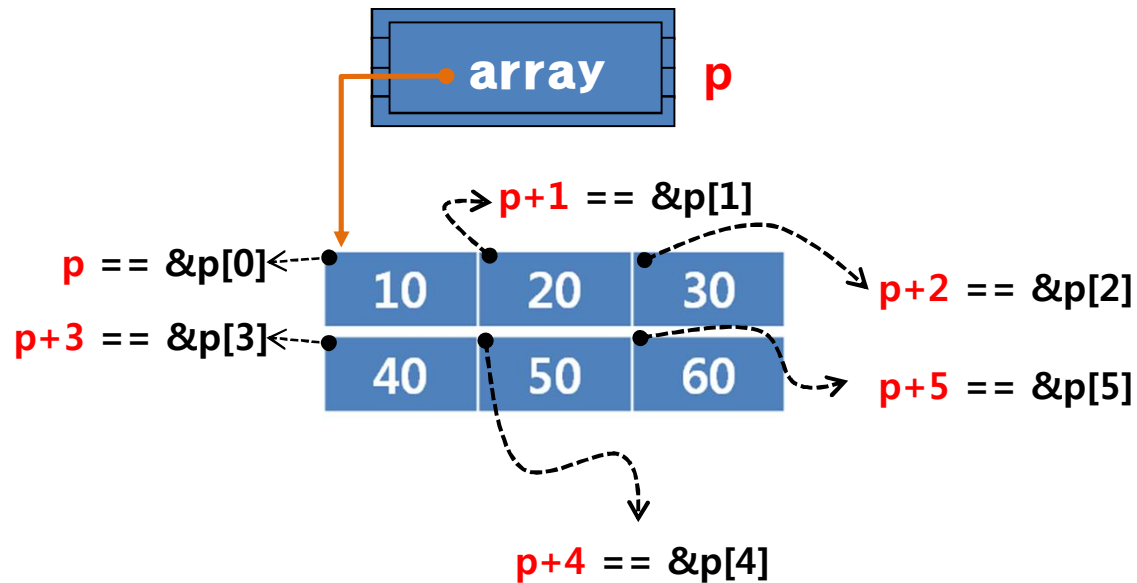
```
int array[2][3]={10,20,30,40,50,60};  
int* p=NULL;  
p=array;
```



4.2 포인터와 2차원 배열 (10/17)---[4-14.c 분석 2]

```
int array[2][3]={10,20,30,40,50,60};  
int* p=NULL;  
p=array;
```

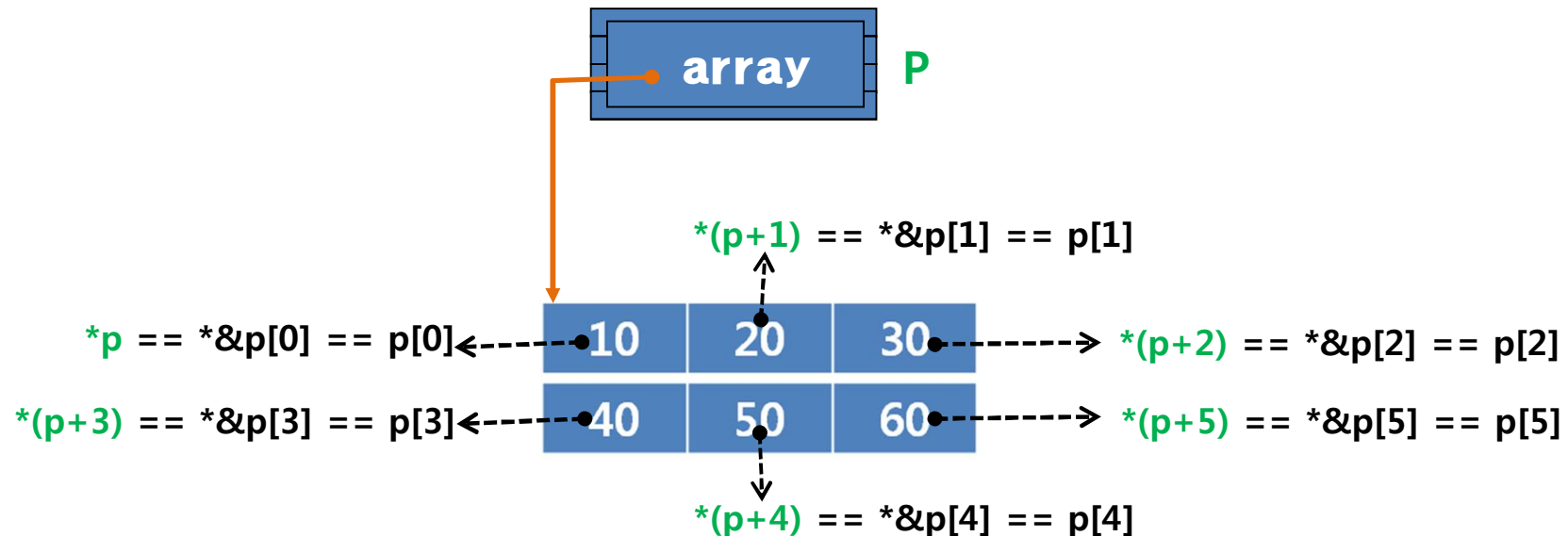
1차원 포인터 변수 **p**를 이용한 '주소' 표현



4.2 포인터와 2차원 배열 (11/17)---[4-14.c 분석 3]

```
int array[2][3]={10,20,30,40,50,60};  
int* p=NULL  
p=array;
```

1차원 포인터 변수 **p**를 이용한 '값' 표현



4.2 포인터와 2차원 배열 (12/17)---[4-15.c 실습]

```
#include <stdio.h>
int main(void)
{
    int array[2][3]={10,20,30,40,50,60};
    int* p=NULL;          // int** p;

    p=array;  // 1차원 포인터 변수에 2차원 배열의 시작 주소 저장

    printf("%d %d %d \n", p[0][0], p[0][1], p[0][2]); // 에러
    printf("%d %d %d \n", p[1][0], p[1][1], p[1][2]); // 에러
    return 0;
}
```

↓

'1차원 포인터 변수 p는 2차원 배열 array[2][3]을 1차원으로만 접근할 수 있다.'

↓

p[0], p[1], p[2], p[3], p[4], p[5], p[6]

4.2 포인터와 2차원 배열 (13/17)---[4-15.c 실습]

```
#include <stdio.h>
int main(void)
{
    int array[2][3]={10,20,30,40,50,60};
    int** p=NULL;

    p=array;    // 2차원 포인터 변수에 2차원 배열의 시작 주소 저장

    printf("%d %d %d \n", p[0][0], p[0][1], p[0][2]); // 에러
    printf("%d %d %d \n", p[1][0], p[1][1], p[1][2]); // 에러
    return 0;
}
```

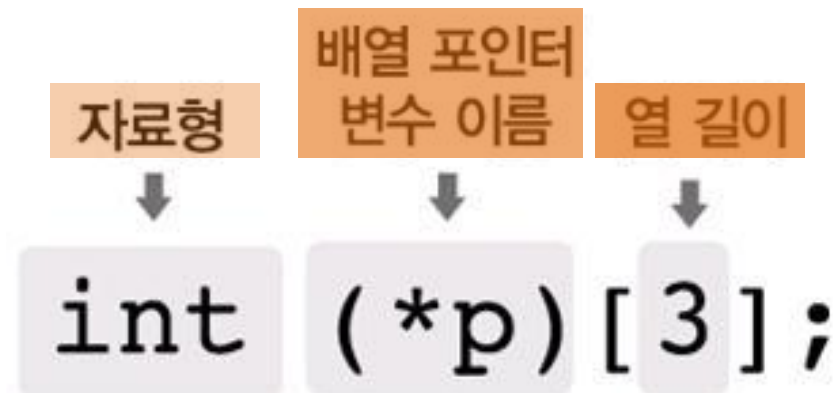


'2차원 포인터 변수 p는 1차원 포인터의 주소만 저장할 수 있다.'

4.2 포인터와 2차원 배열 (14/17)

▶ 배열 포인터

✓ 열을 지정 할 수 있는 포인터 (배열을 가리키는 포인터)



- **자료형**: 배열 포인터 변수가 저장하는 배열의 자료형
- **배열 포인터 변수 이름**: * 연산자와 배열 포인터 변수 이름을 함께 괄호로 묶음
- **열의 길이**: 배열 포인터 변수가 가리키는 배열의 열의 길이를 지정

4.2 포인터와 2차원 배열 (15/17)

▶ 배열 포인터 변수를 통한 2차원 배열의 접근

✓ 열을 지정

...

```
int array1[2][3];
```

```
int (*p1)[3]=NULL;    // 배열 포인터 변수 p1 선언
```

```
double array2[2][4];
```

```
double (*p2)[4]=NULL; // 배열 포인터 변수 p2 선언
```

```
p1=array1; // p1에 3열을 가지는 2차원 배열 array1의 시작 주소를 저장
```

```
p2=array2; // p2에 4열을 가지는 2차원 배열 array2의 시작 주소를 저장
```

```
return 0;
```

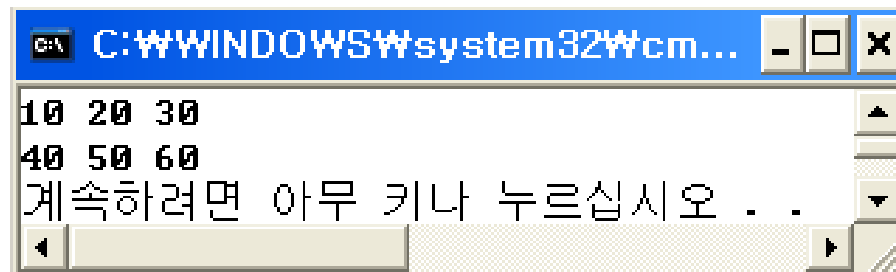
...

4.2 포인터와 2차원 배열 (16/17)---[4-16.c 실습]

```
#include <stdio.h>
int main(void)
{
    int array[2][3]={10,20,30,40,50,60};
    int (*p)[3]=NULL;      // 배열 포인터 변수 p 선언

    p=array;               // 포인터 변수에 배열의 시작 주소 저장

    printf("%d %d %d \n", p[0][0], p[0][1], p[0][2]);
    printf("%d %d %d \n", p[1][0], p[1][1], p[1][2]);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd...
10 20 30
40 50 60
계속하려면 아무 키나 누르십시오 . . .
```

4.2 포인터와 2차원 배열 (17/17)

▶ 스스로 해보기

✓ 교재 4-17.c

✓ 교재 4-18.c

코드를 분석하며 그림도 그려볼 것

4.3 포인터 배열

4.3 포인터 배열 (1/8)

▶ 포인터 배열

✓ 주소를 저장하는 배열

자료형	포인터 배열 이름	배열 길이
↓	↓	↓
<code>int*</code>	<code>pointer</code>	<code>[3]</code>
<code>pointer[3];</code>		

- **자료형**: 포인터 배열의 자료형을 지정, 자료형 다음에 * 연산자를 붙임
- **포인터 배열 이름**: 주소를 저장할 배열의 이름을 지정
- **배열 길이**: 주소를 저장할 배열의 전체 길이를 지정

4.3 포인터 배열 (2/8)

▶ 포인터 배열의 선언

```
int a=1, b=2, c=3;  
int* pointer[3]={NULL, NULL, NULL}; // 포인터 배열 선언  
  
pointer[0]=&a;  
pointer[1]=&b;  
pointer[2]=&c;  
return 0;
```

▶ 포인터 배열의 필요성

- ✓ 포인터 변수가 많아지는 단점을 보완

4.3 포인터 배열 (3/8)---[4-19.c 실습]

```
#include <stdio.h>
int main(void)
{
    int a=10, b=20, c=30;
    int* ap=NULL;
    int* bp=NULL;
    int* cp=NULL;
```

```
ap=&a;
bp=&b;
cp=&c;
```

‘포인터 변수가 많아지면
관리가 어려워 질 수 있다.’

```
printf("%d %d %d \n", a, b, c);
printf("%d %d %d \n", *ap, *bp, *cp);

printf("%x %x %x \n", &a, &b, &c);
printf("%x %x %x \n", ap, bp, cp);
printf("%x %x %x \n", &ap, &bp, &cp);
return 0;
```

4.3 포인터 배열 (4/8)---[4-20.c 실습]

```
#include <stdio.h>
int main(void)
{
    int a=10, b=20, c=30;
    int* ap[3]={NULL, NULL, NULL}; //포인터 배열
```

```
    ap[0]=&a;
    ap[1]=&b;
    ap[2]=&c;
```

‘포인터 배열’의 요소로
주소를 체계적으로 관리를 할 수 있다!

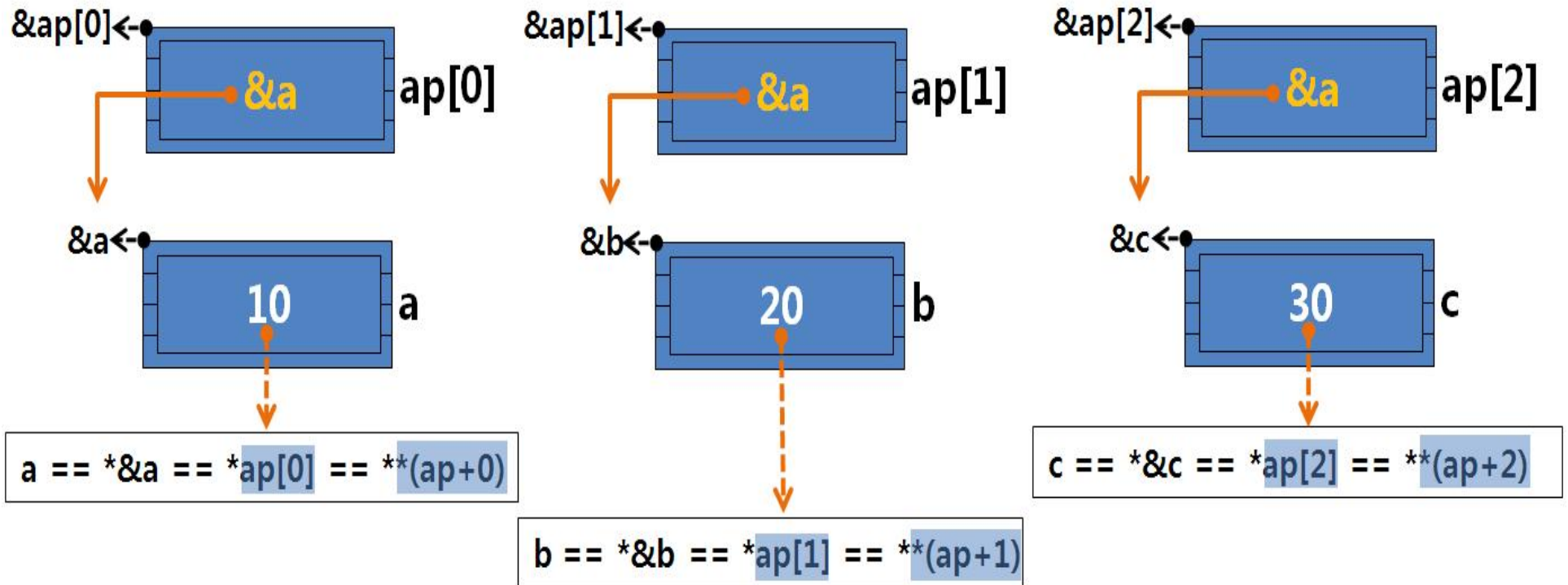
```
    printf("%x %x %x \n", &a, &b, &c);
    printf("%x %x %x \n", ap[0], ap[1], ap[2]);
    printf("%x %x %x \n", *(ap+0), *(ap+1), *(ap+2));
    printf("-----\n");
```

```
    printf("%d %d %d \n", *&a, *&b, *&c);
    printf("%d %d %d \n", *ap[0], *ap[1], *ap[2] );
    printf("%d %d %d \n", ***(ap+0), ***(ap+1), ***(ap+2) );
    return 0;
```

$ap[i] == *(ap+i)$

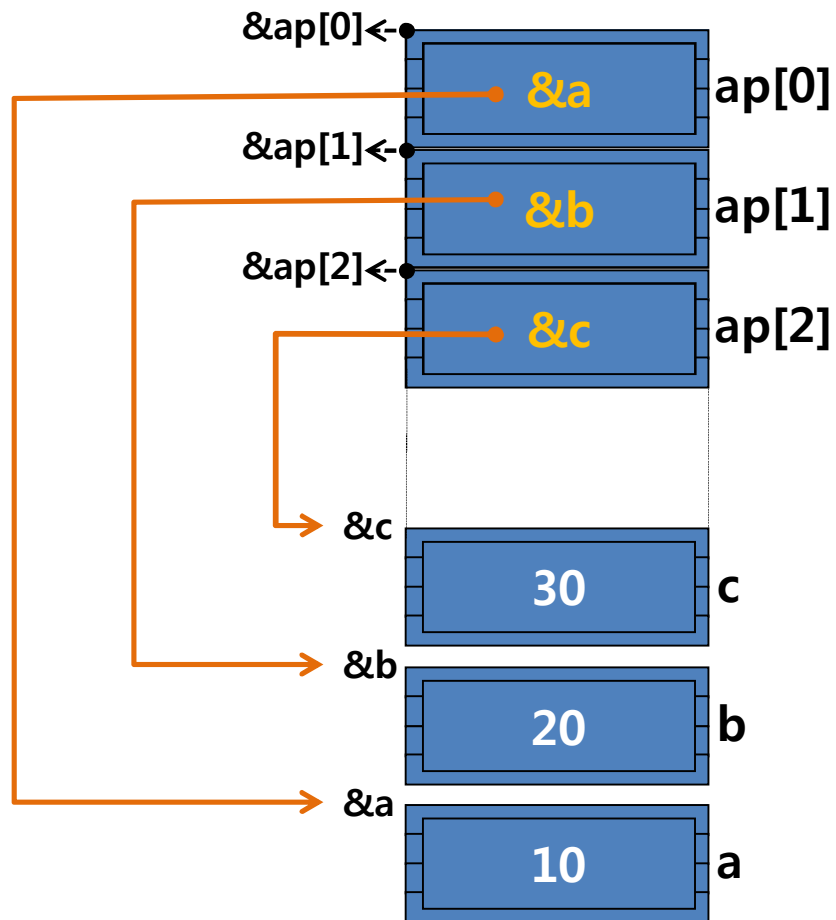
4.3 포인터 배열 (5/8)---[4-20.c 분석]

```
int a=10, b=20, c=30;  
int* ap[3]={NULL, NULL, NULL};  
ap[0]=&a;  
ap[1]=&b;  
ap[2]=&c;
```



4.3 포인터 배열 (6/8)---[4-20.c 분석]

보다 정확한 구조



```
int a=10, b=20, c=30;
int* ap[3]={NULL, NULL, NULL};
ap[0]=&a;
ap[1]=&b;
ap[2]=&c;
```

4.3 포인터 배열 (7/8)

▶ '포인터 배열' 과 '배열 포인터' 의 차이

✓ 배열 포인터 변수

- 3열 가진 2차원 배열의 시작 주소를 저장

```
int (*p)[3]=NULL;
```

✓ 포인터 배열 변수

- 괄호가 생략되어 있으며, 주소를 저장할 수 있는 배열

```
int* p[3]={NULL, NULL, NULL};
```

4.3 포인터 배열 (8/8)---[4-21.c 실습]

```
...  
int a=10, b=20, c=30;  
int* ap[3];           //포인터 배열  
  
int array[2][3]={10,20,30,40,50,60};  
int (*p)[3];          // 배열 포인터  
  
ap[0]=&a;  
ap[1]=&b;  
ap[2]=&c;  
...  
  
p=array;
```

4.4 포인터와 문자 그리고 포인터와 문자열

4.4 포인터와 문자 그리고 포인터와 문자열 (1/21)

▶ 문자 배열과 포인터

✓ 문자 상수

- 작은따옴표(' ') 내에 포함된 하나의 문자
- 키보드로 표현할 수 있는 영문자와 숫자, 특수 기호
- 문자 그 자체

✓ 문자 배열

- 문자 상수를 저장하고 있는 배열
- 배열에 저장된 문자 변경 가능

✓ 문자 배열과 포인터

- 배열에 저장된 문자를 포인터를 통해 접근

4.4 포인터와 문자 그리고 포인터와 문자열 (2/21)---[4-22.c 실습]

```
...
char array[ ]={'A', 'B', 'C', 'D'};      // 문자 배열 선언

// 문자 상수
printf("문자 상수 : %c %c %c %c \n", 'A', 'B', 'C', 'D');

// 문자 배열
printf("문자 배열 변경 전 : %c %c %c %c \n", array[0], array[1], array[2], array[3]);

array[0]='D';
array[1]='C';
array[2]='B';
array[3]='A';

printf("문자 배열 변경 후 : %c %c %c %c \n", array[0], array[1], array[2], array[3]);
printf("문자 배열 array의 크기 : %d \n", sizeof(array) );    // 문자 배열의 크기 4
...
```

4.4 포인터와 문자 그리고 포인터와 문자열 (3/21)---[4-23.c 실습]

```
#include <stdio.h>
int main(void)
{
    char array1[ ]={'A', 'B', 'C', 'D'};
    char* p=NULL;
```

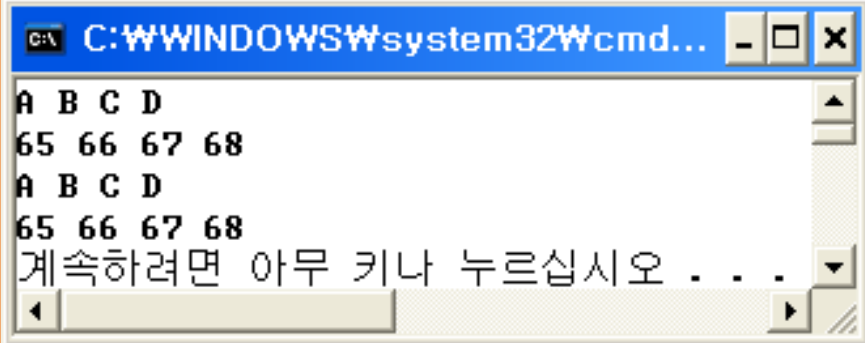
```
p=array1;
```

```
printf("%c %c %c %c \n", p[0], p[1], p[2], p[3]);
printf("%d %d %d %d \n", p[0], p[1], p[2], p[3]);
```

```
printf("%c %c %c %c \n", *(p+0), *(p+1), *(p+2), *(p+3) );
printf("%d %d %d %d \n", *(p+0), *(p+1), *(p+2), *(p+3) );
```

```
return 0;
```

```
}
```



```
C:\WINDOWS\system32\cmd...
A B C D
65 66 67 68
A B C D
65 66 67 68
계속하려면 아무 키나 누르십시오 . . .
```

4.4 포인터와 문자 그리고 포인터와 문자열 (4/21)

▶ 문자열과 널(Null)문자

✓ 문자열의 특징

- ""(큰따옴표) 내에 포함된 하나 이상의 문자
- 문자열의 끝에는 문자열의 끝을 알리는 널(Null) 문자 즉, 종료 문자(W0) 삽입
- 문자열의 시작 주소를 알면 저장된 문자들에 접근 가능
- 문자열은 문자열 상수와 문자열 변수로 구분
- 문자열을 입력하고 출력할 때 서식문자 %s를 사용

```
#include <stdio.h>
int main(void)
{
    printf("ABCD");           // 문자열 상수
    return 0;
}
```

4.4 포인터와 문자 그리고 포인터와 문자열 (5/21)

▶ 널(Null) 문자와 널(NULL) 포인터

✓ 널(Null) 문자

- 종료문자 'w0'
- ASCII 코드 정수 0(10진수)
- 문자열 끝에 저장

✓ 널(NULL) 포인터

- 주소로 0을 의미
- 포인터 변수에 아무 주소도 저장하지 않겠다는 의미
- 널 포인터를 사용할 때는 반드시 대문자를 사용

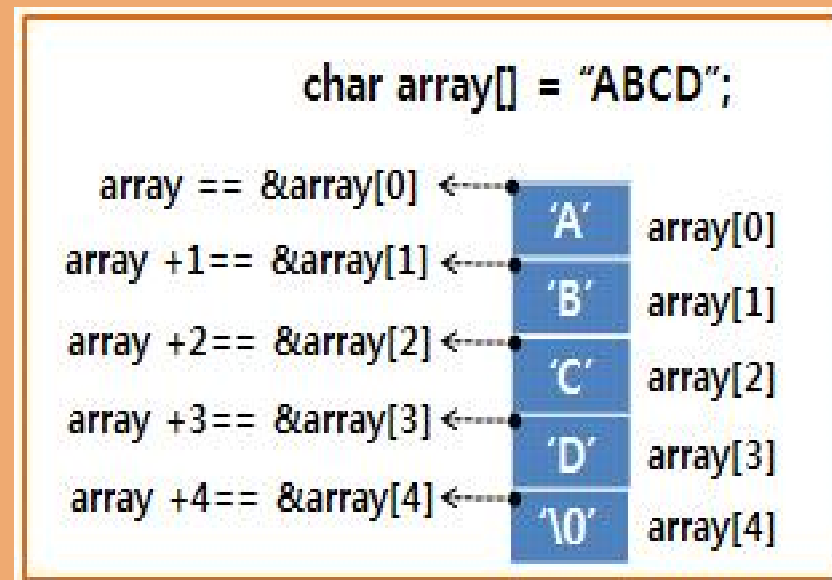
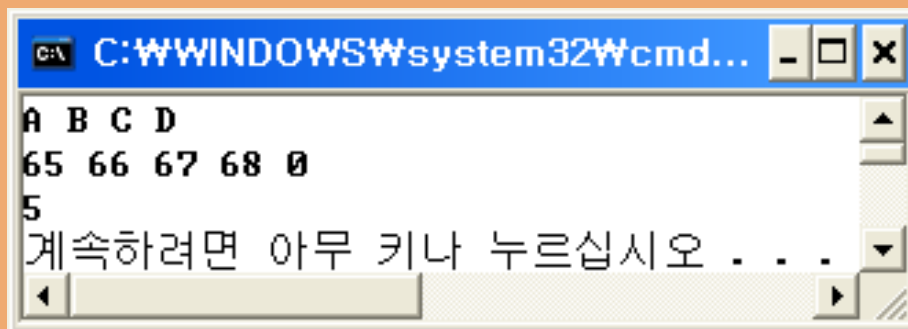
`int* p=NULL;`

4.4 포인터와 문자 그리고 포인터와 문자열 (6/21)---[4-24.c 실습]

```
#include <stdio.h>
int main(void)
{
    char array[ ]="ABCD";    // 문자열 배열 선언

    // 문자 출력
    printf("%c %c %c %c %c  \n", array[0], array[1], array[2], array[3], array[4]);
    printf("%d %d %d %d %d  \n", array[0], array[1], array[2], array[3], array[4]);

    // 문자열 크기 출력
    printf("%d \n", sizeof(array));
    return 0;
}
```



4.4 포인터와 문자 그리고 포인터와 문자열 (7/21)---[4-25.c 실습]

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    char array[ ]={'A', 'B', 'C', 'D', 'W0' };    // 문자열 배열 선언
```

종료문자 삽입

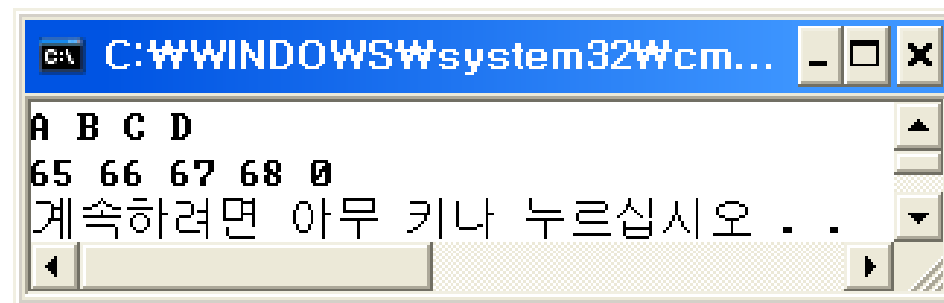
// 문자 출력

```
    printf("%c %c %c %c %c  \n", array[0], array[1], array[2], array[3], array[4]);
```

```
    printf("%d %d %d %d %d  \n", array[0], array[1], array[2], array[3], array[4]);
```

```
    return 0;
```

```
}
```

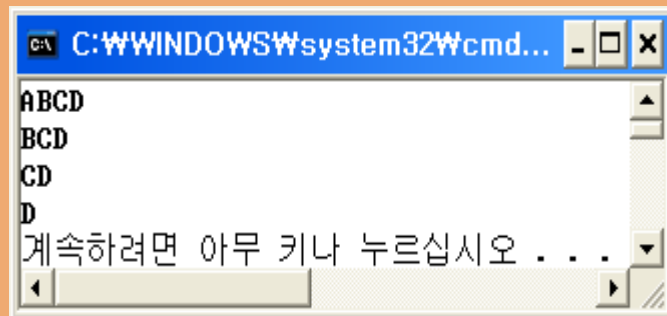


4.4 포인터와 문자 그리고 포인터와 문자열 (8/21)---[4-26.c 실습]

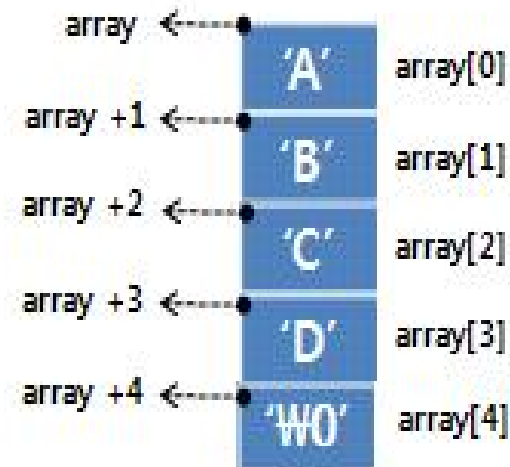
```
#include <stdio.h>
int main(void)
{
    char array[ ]="ABCD"; // 종료문자 '\0' 자동 삽입

    // 문자열 출력
    printf("%s\n", array);
    printf("%s\n", array+1);
    printf("%s\n", array+2);
    printf("%s\n", array+3);

    return 0;
}
```



char array[] = "ABCD";



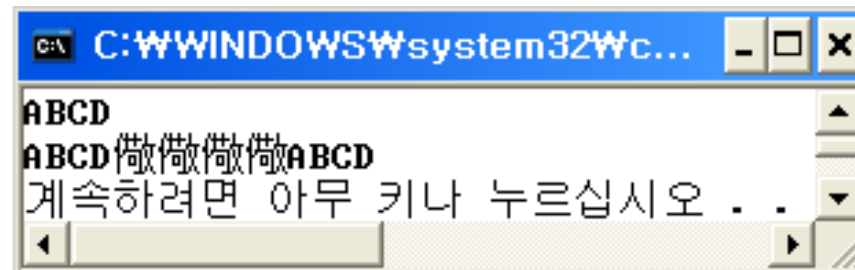
%s 는 문자열의 시작 주소를 통해 문자열을 출력한다.

4.4 포인터와 문자 그리고 포인터와 문자열 (9/21)---[4-27.c 실습]

```
#include <stdio.h>
int main(void)
{
    char array1[ ]={'A', 'B', 'C', 'D', '\0'}; // 문자열 배열 선언
    char array2[ ]={'A', 'B', 'C', 'D'};      // 문자 배열 선언

    // 문자열 출력
    printf("%s\n", array1);
    printf("%s\n", array2);

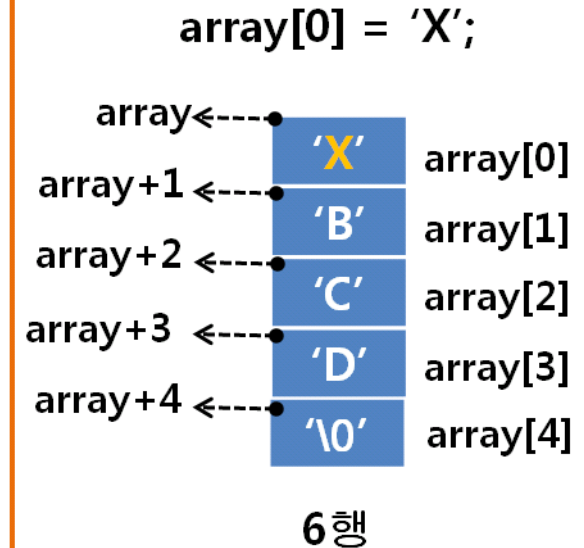
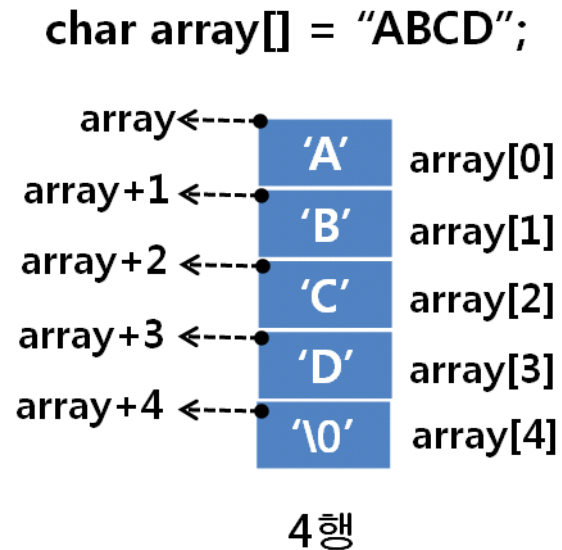
    return 0;
}
```



4.4 포인터와 문자 그리고 포인터와 문자열 (10/21)---[4-28.c 실습]

```
#include <stdio.h>
int main(void)
{
    char array[ ]="ABCD"; // 문자열 배열 선언

    array[0]='X';
    printf("%s\n", array);
    return 0;
}
```



4.4 포인터와 문자 그리고 포인터와 문자열 (11/21)

▶ 포인터와 문자열

✓ 포인터

- '문자열의 시작 주소를 저장한다.'
- '문자열의 특정 문자 위치를 저장한다.'

✓ 문자열의 특징

- '메모리 공간에 **연속**으로 저장되어 있어 **주소가 연속적이다.**'
- '**문자열의 시작 주소**를 알면 모든 문자들에 접근 가능하다.'
- '**서식문자 %s**로 문자열을 일괄 출력할 수 있다.'
- **%s**는 문자열의 **시작 주소**부터 종료 문자(₩0)를 만날 때 까지 문자열을 출력

4.4 포인터와 문자 그리고 포인터와 문자열 (12/21)---[4-29.c 실습]

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    char* p="ABCD";
```

// 문자열 상수 ABCD의 시작 주소를 p에 저장

```
    printf("%s\n", p);
```

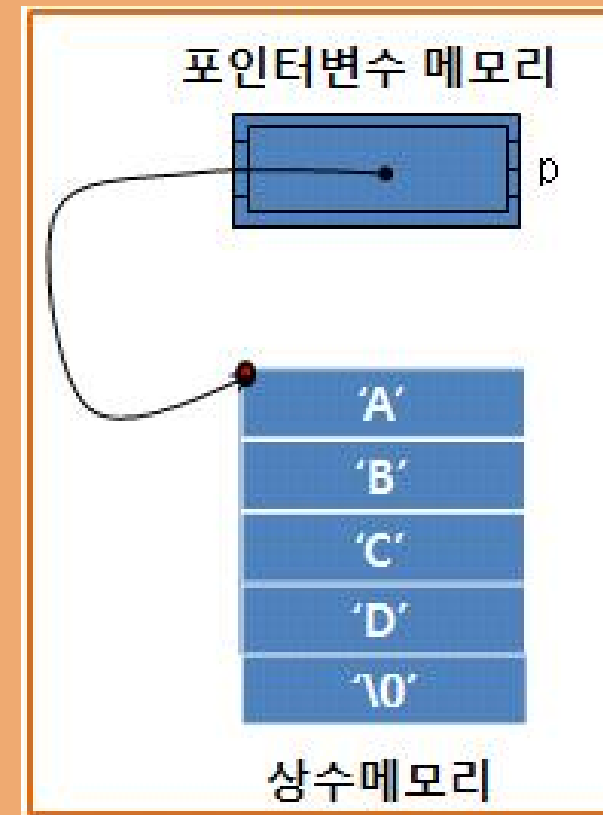
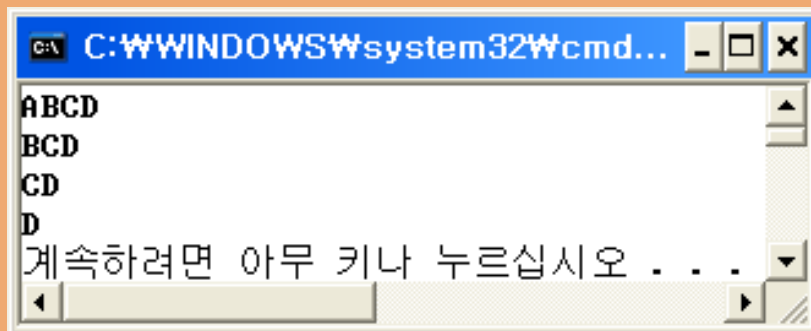
```
    printf("%s\n", p+1);
```

```
    printf("%s\n", p+2);
```

```
    printf("%s\n", p+3);
```

```
    return 0;
```

```
}
```



4.4 포인터와 문자 그리고 포인터와 문자열 (13/21)---[4-30.c 실습]

```
#include <stdio.h>
int main(void)
{
    char array[ ]="ABCD";
    char* p="ABCD"; // 문자열 상수의 시작 주소를 p에 저장
```

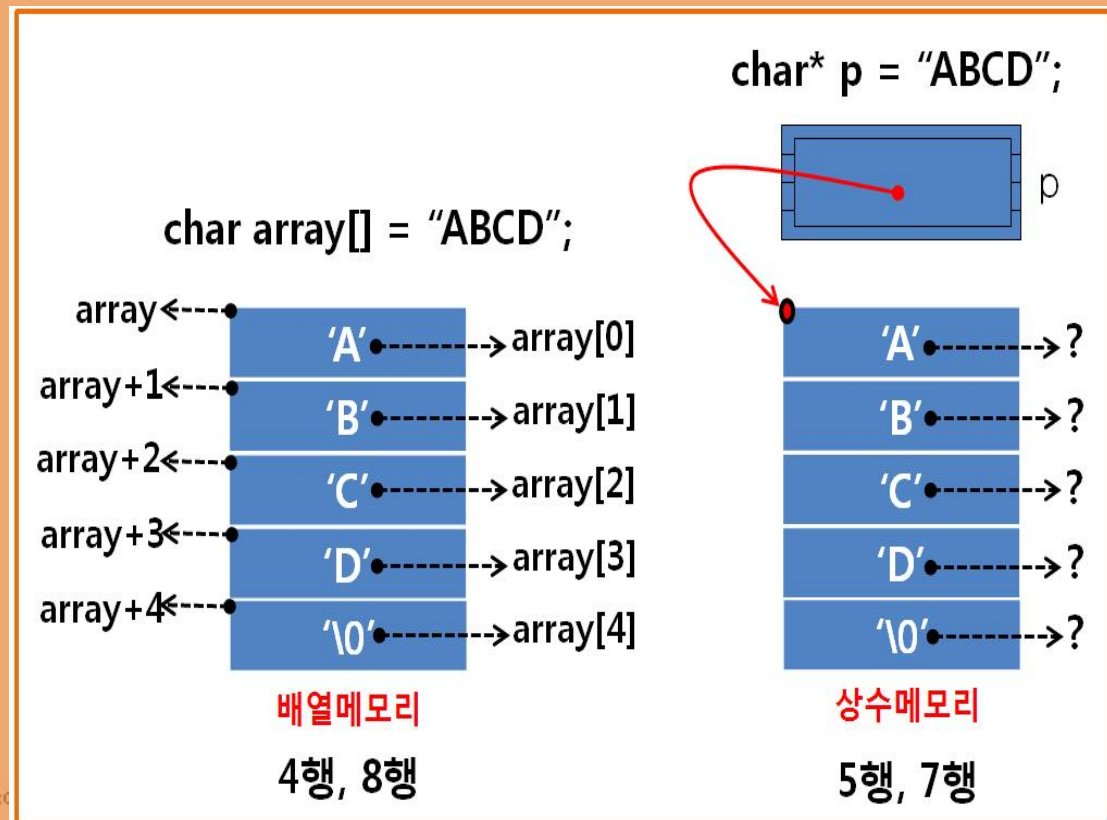
```
p[0]='X';           // 에러
array[0]='X';        // 변경 가능
```

```
p=array;            // 변경 가능
array=array+1;       // 에러
```

```
printf("%s\n", p);
printf("%s\n", array);
```

```
return 0;
```

```
}
```



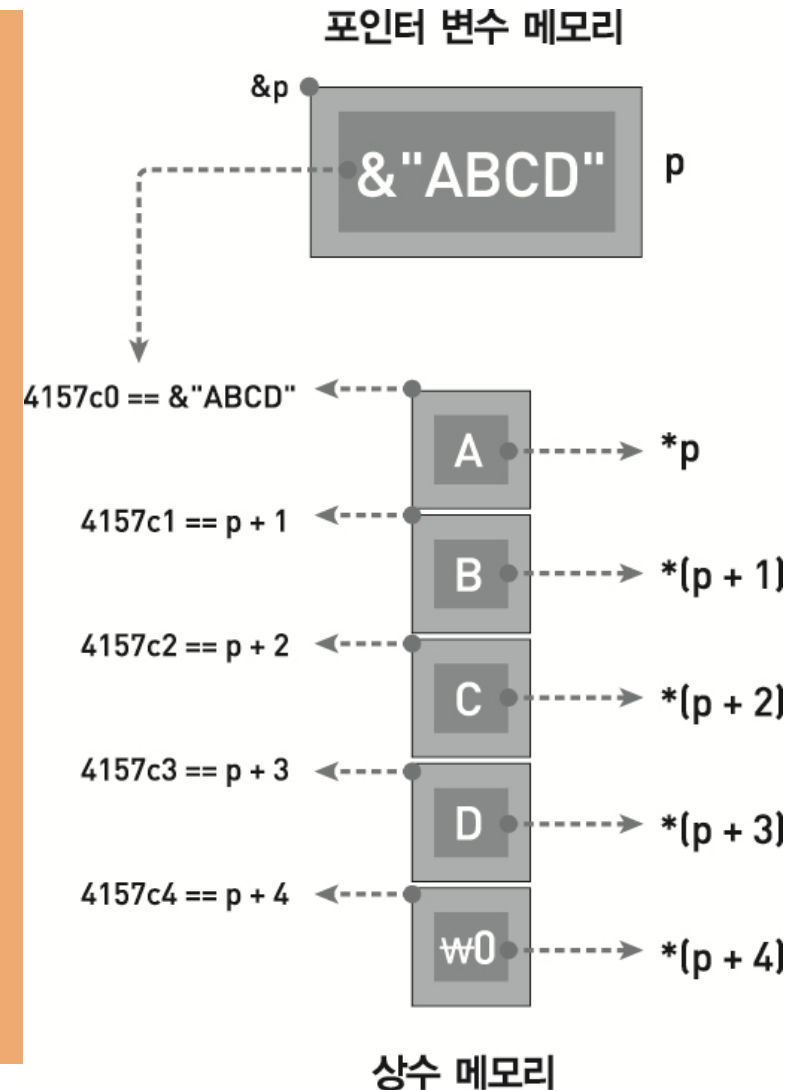
4.4 포인터와 문자 그리고 포인터와 문자열 (14/21)---[4-31.c 실습]

```
#include <stdio.h>
int main(void)
{
    char* p= &"ABCD"; // char* p="ABCD";

    printf("%x \n", p);
    printf("%x \n", p+1);
    printf("%x \n", p+2);
    printf("%x \n", p+3);
    printf("%x \n", p+4);
    printf("-----\n");

    printf("%x %x \n", &"ABCD", p);

    return 0;
}
```



4.4 포인터와 문자 그리고 포인터와 문자열 (15/21)---[4-32.c 실습]

...

```
char* p="Good morning";  
char* q="C-language";  
char* array[2]={"Good morning", "C-language"}; // 포인터 배열 선언
```

```
printf("%s \n", p);  
printf("%s \n", q);  
printf("-----\n");
```

```
printf("%s \n", array[0]);  
printf("%s \n", array[1]);  
printf("-----\n");
```

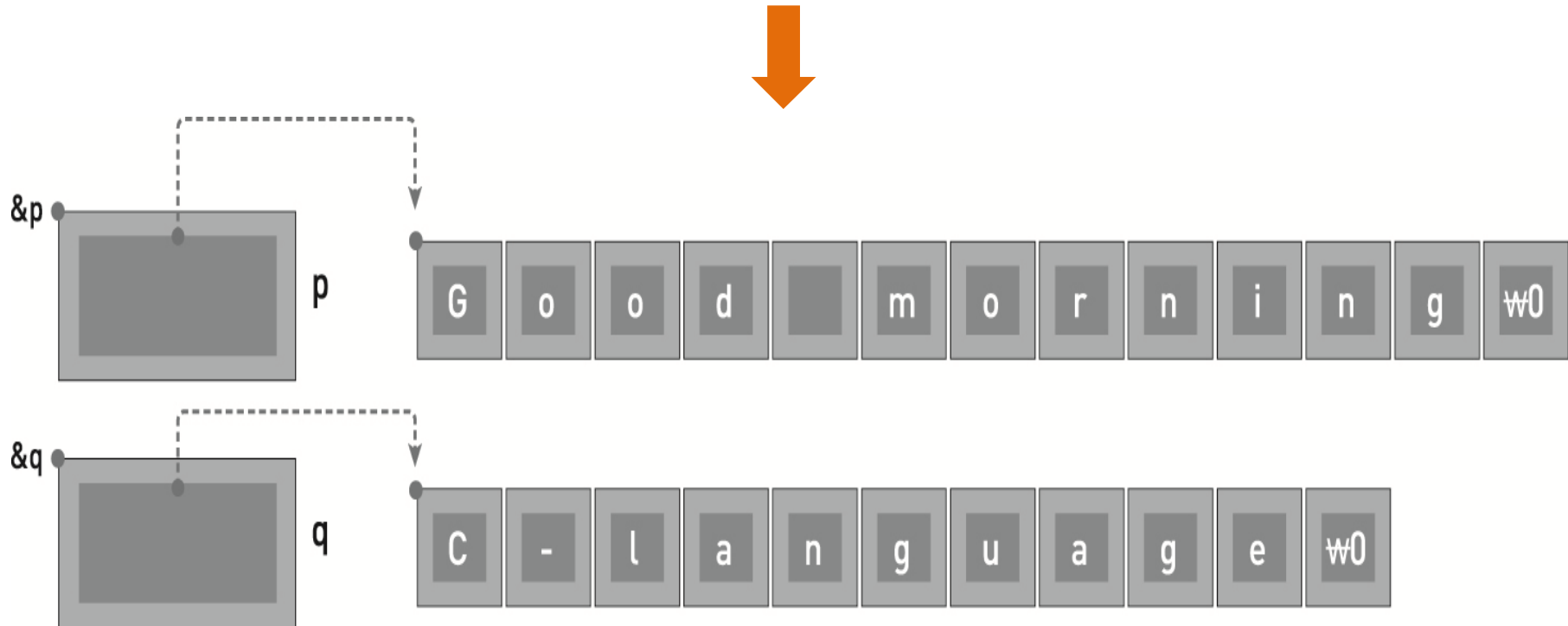
```
printf("%s \n", p+5);  
printf("%s \n", q+2);  
printf("-----\n");
```

```
printf("%s \n", array[0]+5);  
printf("%s \n", array[1]+2);
```

...

4.4 포인터와 문자 그리고 포인터와 문자열 (16/21)---[4-32.c 분석]

```
char* p="Good morning";  
char* q="C-language";
```

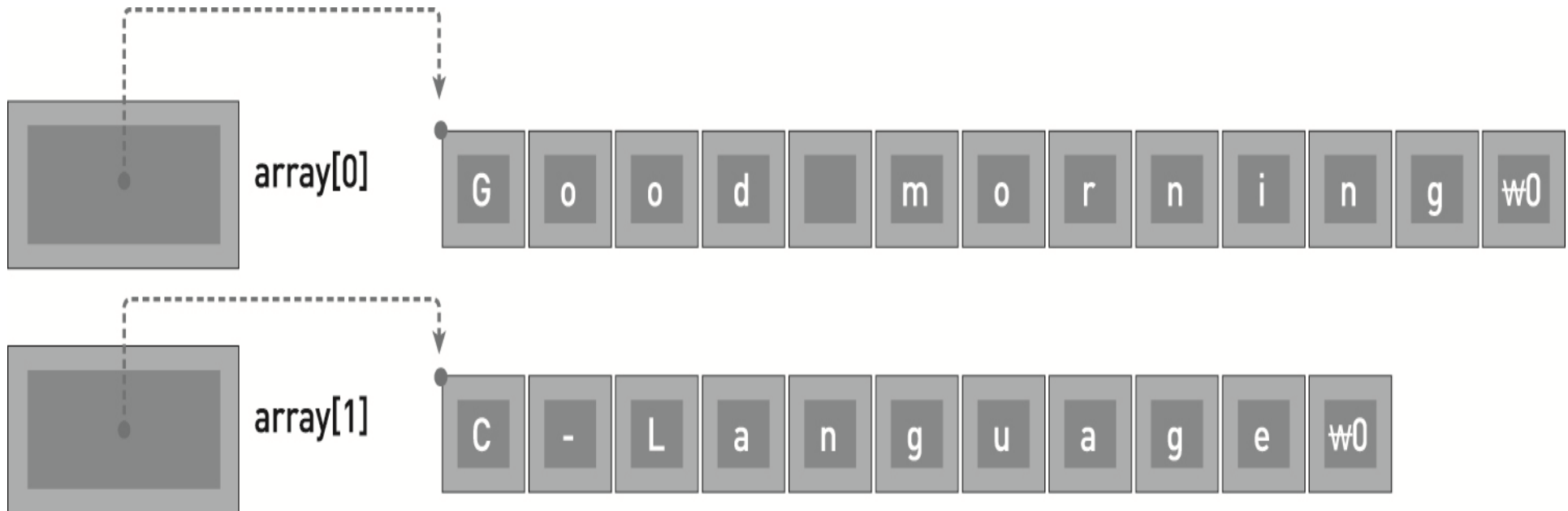


4.4 포인터와 문자 그리고 포인터와 문자열 (17/21)---[4-32.c 분석]

```
char* array[2]={"Good morning", "C-language"}; // 포인터 배열 선언
```

==

```
char* array[2]={&"Good morning", &"C-language"}; // 같은 표현
```



4.4 포인터와 문자 그리고 포인터와 문자열 (18/21)

▶ 포인터 변수의 상수화

✓ **const 키워드**를 이용해 포인터 변수를 상수화

▶ 포인터 변수의 상수화의 의미

- ① '포인터 변수에 **다른 주소**를 저장하지 못하게 한다.'
- ② '포인터 변수를 통해 메모리 공간의 **값**을 변경하지 못하게 한다.'
- ③ '① 과 ② 둘 다 못하게 한다.'

4.4 포인터와 문자 그리고 포인터와 문자열 (19/21)---[4-33.c 실습]

① '포인터 변수에 다른 주소를 저장하지 못하게 한다.'

```
#include <stdio.h>
int main(void)
{
    char a='A';
    char b='B';

    char* const p=&a;    // p=&a 상수화

    *p='C';              // 변경 가능
    printf("%c \n", *p);
    printf("%c \n", a);

    p=&b;                // 에러
    return 0;
}
```

4.4 포인터와 문자 그리고 포인터와 문자열 (20/21)---[4-34.c 실습]

② '포인터 변수를 통해 메모리 공간의 값을 변경하지 못하게 한다.'

```
#include <stdio.h>
int main(void)
{
    char a='A';
    char b='B';
    const char* p=&a;           // *p를 상수화

    printf("%c \n", *p);
    printf("%c \n", a);

    p=&b;    // 변경 가능
    printf("%c \n", *p);
    printf("%c \n", b);

    a='X';
    b='C';
    *p='D';    // 에러
    return 0;
}
```

4.4 포인터와 문자 그리고 포인터와 문자열 (21/21)---[4-35.c 실습]

③ '포인터 변수를 통해 메모리 공간의 주소와 값을 모두 변경 못하게 한다.'

```
...  
char a='A';  
char b='B';  
const char* const p=&a;  
  
printf("%c \n", *p);  
printf("%c \n", a);  
  
a='X';           // 변경 가능  
b='C';           // 변경 가능  
  
printf("%c \n", a);  
printf("%c \n", b);  
  
p=&b;           // 에러  
*p='D';        // 에러  
...
```

공부한 내용 떠올리기

- ▶ 포인터를 통해 1차원 배열에 접근하는 방법
- ▶ 포인터를 통해 2차원 배열에 접근하는 방법
- ▶ 포인터와 배열의 관계
- ▶ 포인터와 문자 배열
- ▶ 포인터와 문자열 배열
- ▶ 포인터를 상수화하는 방법

진실된 마음 (출처: 사랑과 지혜의 탈무드)

