

WEEK 10

포인터 기초

학습목표

- I. 포인터 변수 이해
- II. 다중 포인터와 배열 포인터 이해
- III. 배열과 포인터의 관계 이해

학습목차

- I. 제 1교시 포인터 변수와 선언
- II. 제 2교시 포인터 형변환과 다중 포인터
- III. 제 3교시 포인터를 사용한 배열 활용

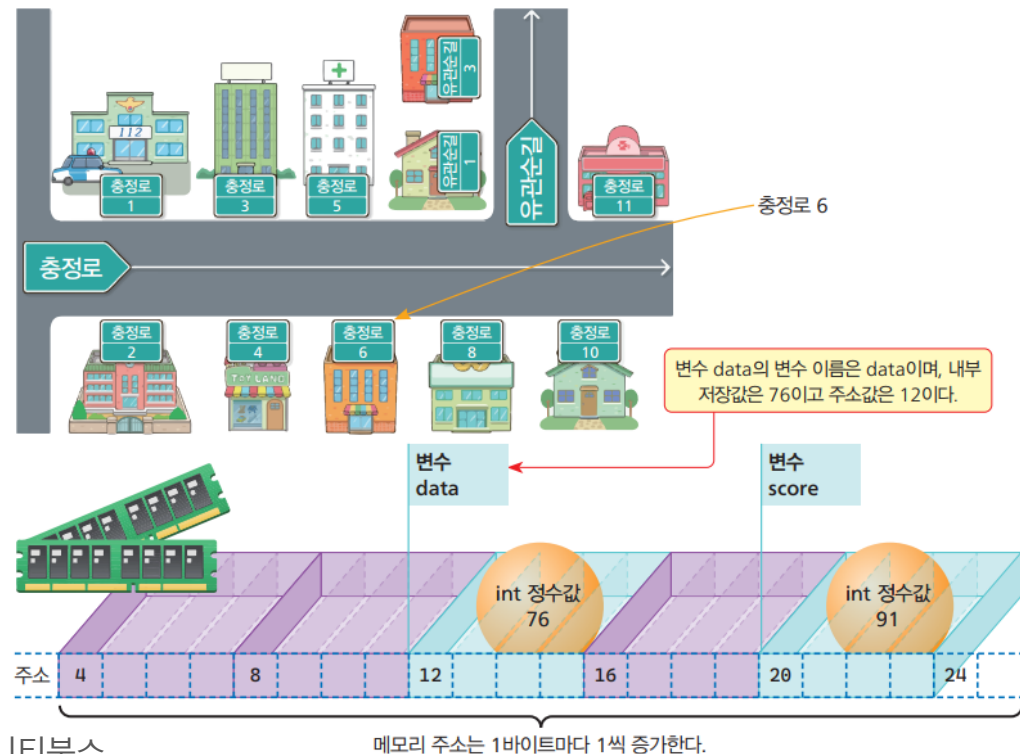
1. 포인터 변수와 선언

1. 메모리 주소와 주소연산자 &
2. 포인터 변수 개념과 선언
3. 간접연산자 *와 포인터 연산

1. 메모리 주소와 주소 연산자 &

◆ 고유한 주소(address)

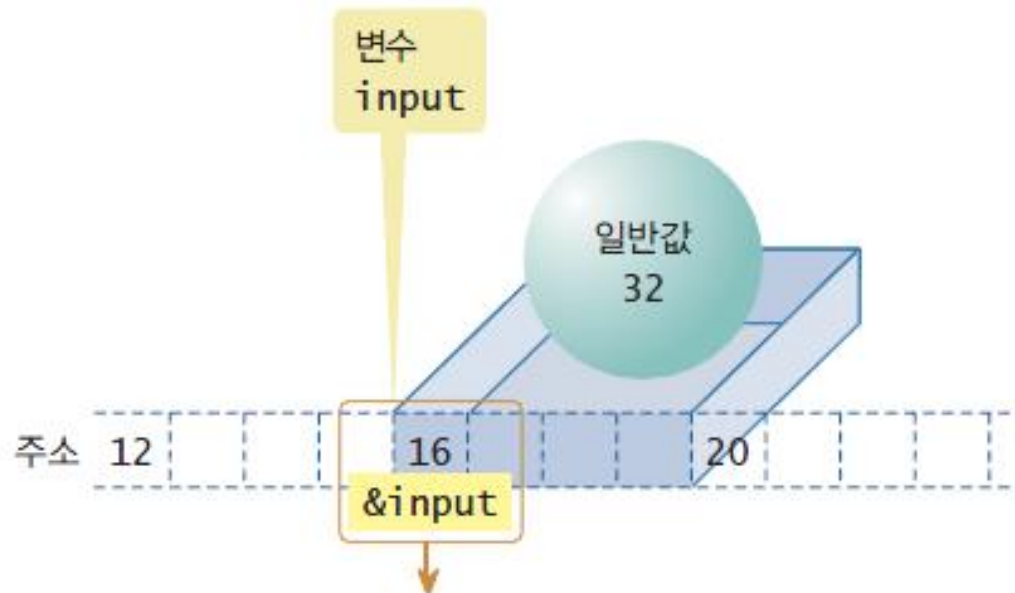
- ❖ 메모리 공간은 8비트인 1 바이트마다 고유한 숫자
- ❖ 0부터 바이트마다 1씩 증가
- ❖ 메모리 주소는 저장 장소인 변수이름과 함께 기억장소를 참조하는 또 다른 방법



1. 메모리 주소와 주소 연산자 &

◆ 주소 연산자 &

- ❖ &가 피연산자인 변수의 메모리 주소를 반환
- ❖ 변수의 주소값은 형식제어문자 %u 또는 %d로 출력
- ❖ 16진수로 출력하려면 형식제어문자 %p를 사용



&input은 변수 input의 주소값을 반환,
실제 주소값은 4바이트 크기의 양의 정수

1. 메모리 주소와 주소 연산자 &

❖ 변수의 주소 값

- 형식제어문자 %p로 직접 출력
- 윈도우 10의 64비트 시스템 주소 값
 - 8바이트(64비트)
 - 16진수의 16개 자릿수로 출력
- 형식제어문자 %llu로 출력
 - %llu는 long long unsigned를 의미
 - 64비트의 0과 양수의 정수형을 위한 형식제어문자
- 자료형 uintptr_t
 - typedef unsigned __int64 uintptr_t;
 - 헤더파일 vdefs.h에 unsigned __int64와 동일한 자료형으로 정의
 - __int64는 long long int와 같이 64비트 정수 자료형

1. 메모리 주소와 주소 연산자 &

- ❖ 연산자 sizeof(&input)의 반환 값
 - 자료형 size_t 유형의 주소의 크기, 형식제어문자 %zu로 출력
 - z는 size를 u는 unsigned를 의미
 - 자료형 size_t: unsigned long long
 - 연산자 sizeof의 반환 값
 - %zd로도 가능

실습예제 11-1

Prj01

01address.c

메모리 주소연산자와 주소 출력

난이도: ★

```

01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03
04 int main(void)
05 {
06     int input;
07
08     printf("정수 입력: ");
09     scanf("%d", &input);
10     printf("입력 값: %d\n", input);
11     printf("주소값: %p(16진수)\n", &input);
12     printf("주소값: %llu(10진수)\n", (uintptr_t)&input);
13
14     printf("주소값 크기: %zu\n", sizeof(&input)); // %zd도 가능
15
16     return 0;
17 }

```

64비트 시스템에서 16개의 16진수 주소값이 출력

주소값을 10진수로 출력하기
위해 uintptr_t로 변환해 출력

결과

```

정수 입력: 100
입력 값: 100
주소값: 000000B8DA52FC34(16진수)
주소값: 793936854068(10진수)
주소값 크기: 8

```

64비트 시스템에서 주소값은 8바이트(64비트)

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 포인터 변수 개념과 선언

❖ 주소 저장 변수

- 주소연산식 &data는 포인터 변수에 저장하여 사용 가능
- 즉, 포인터 변수는 주소값을 저장하는 변수
- 포인터 변수는 변수 선언에서 자료형 뒤에 *를 사용
- 즉, 다음 변수 선언에서 ptrint, ptrshort, ptrchar, ptrdouble은 모두 포인터 변수이며 간단히 포인터라고도 부름
- 변수 자료형이 따라 그 변수의 주소를 저장하는 포인터의 자료형도 다름

포인터 변수 선언

자료형 *변수이름;

```
int *ptrint;  
short *ptrshort;  
char *ptrchar;  
double *ptrdouble;
```

```
int *ptrint; // 가장 선호  
int* ptrint;  
int * ptrint;  
int* ptrint;
```

2. 포인터 변수 개념과 선언

◆ 포인터에 주소값 저장

❖ `int *ptring = &data;`

- 포인터 변수 ptring은 변수 data 를 가리킨다" 또는 "참조(reference)한다"라고 표현
- 포인터 변수는 가리키는 변수의 종류에 관계없이 크기가 모두 8바이트

```
int data = 100;  
int *ptring;  
  
ptring = &data;
```

2. 포인터 변수 개념과 선언

실습예제 11-2

Prj02

02pointer.c

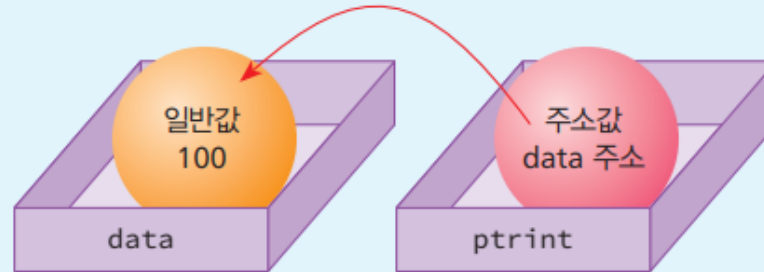
포인터 변수 선언과 주소값 대입

난이도: ★

```

01  #include <stdio.h>
02  int main(void)
03  {
04      int data = 100;
05      int* ptring;
06      ptring = &data;
07      printf("변수명  주소값      저장값\n");
08      printf("-----\n");
09      printf(" data  %p  %d\n", &data, data);
10      printf("ptring %p  %p\n", &ptring, ptring);
11      printf("%zu\n", sizeof(ptring));
12      return 0;
13  }

```



포인터 변수 ptring의 저장 값과
&data는 모두 data의 주소로 같은 값

&ptring도 포인터 변수의 ptring의 주소값

결과

변수명	주소값	저장값
data	0000001C091BF8F4	100
ptring	0000001C091BF918	0000001C091BF8F4
8		

주소값은 %p로 출력하면 16개의 16진수로 출력

3. 간접연산자 *와 포인터 연산

◆ 포인터 선언과 NULL 포인터

❖ 여러 포인터 선언

- 여러 개의 포인터 변수를 한 번에 선언하기 위해서는 다음과 같이 변수마다 *를 앞에 기술

```
int *ptr1, *ptr2, *ptr3;
```

❖ NULL 저장

- 특별한 초기값이 없는 경우에 NULL로 초기값을 저장

```
int *ptr = NULL;
```

❖ NULL

- 헤더 파일 stdio.h에 다음과 같이 정의되어 있는 포인터 상수로서 0번지의 주소값을 의미

```
#define NULL ((void *)0)
```

3. 간접연산자 *와 포인터 연산

◆ 포인터 변수도 다른 일반 변수와 같이 지역변수로 선언

- ❖ 초기 값을 대입하지 않으면 쓰레기 값이 들어가므로
- ❖ 포인터 변수에 지정할 특별한 초기 값이 없는 경우
 - 0번 주소 값인 NULL로 초기 값을 저장

```
int *p3, data2;  
printf("%p\n", p3); //int * 형의 쓰레기 값으로 오류  
printf("%d\n", data2); //int형의 쓰레기 값으로 오류
```

3. 간접연산자 *와 포인터 연산

Prj03

03nullptr.c

한 번에 여러 포인터 변수 선언과 NULL 주소값 대입

난이도: ★

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     int data = 10;
06     int *p1 = NULL, *p2 = &data;
07
08     printf("%d\n", data);
09     printf("%p %p\n", p1, p2);
10
11     //int *p3, data2;
12     //printf("%p\n", p3); //int * 형의 쓰레기 값으로 오류
13     //printf("%d\n", data2); //int형의 쓰레기 값으로 오류
14
15     return 0;
16 }
```

10

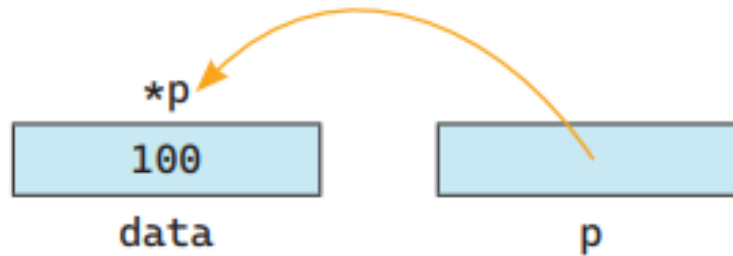
0000000000000000 0000008A77AFFBB4

3. 간접연산자 *와 포인터 연산

◆ 간접 연산자 *

- ❖ 포인터 변수가 가리키고 있는 변수를 참조하려면 간접연산자 *를 사용
 - 전위연산자로 피연산자는 포인터
 - *p는 피연산자인 p가 가리키는 변수 자체를 반환
- ❖ `int *p = &data;`
 - 위 선언으로 포인터 p가 가리키는 변수가 data라면 *p는 변수 data를 의미
 - 문장 `*p = 200;`으로 변수 data의 저장 값을 200으로 수정 가능

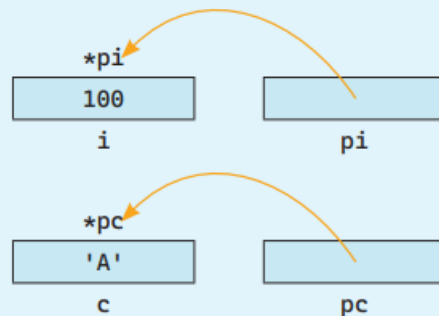
```
int data = 100;  
int *p = &data;  
printf("간접참조 출력: %d \n", *p);
```



```

01 #include <stdio.h>
02
03 int main(void)
04 {
05     int i = 100;
06     char c = 'A';
07
08     int *pi = &i;
09     char *pc = &c;
10     printf("간접참조 출력: %d %c\n", *pi, *pc);
11
12     *pi = 200; //변수 i를 *pi로 간접참조하여 그 내용을 수정
13     *pc = 'B'; //변수 c를 *pc로 간접참조하여 그 내용을 수정
14     printf("직접참조 출력: %d %c\n", i, c);
15
16     return 0;
17 }

```



결과
간접참조 출력: 100 A
직접참조 출력: 200 B



TIP

주소연산자 &와 간접연산자 *

주소연산자 &와 간접연산자 *, 모두 전위 연산자로 주소 연산 '&변수'는 변수의 주소값이 결과값이며, 간접 연산 '*포인터변수'는 포인터 변수가 가리키는 변수 자체가 결과값이다.

```

int n = 100;
int *p = &n; // 이제 *p와 n은 같은 변수
n = *p + 1; // n = n + 1;과 같음
*p = *p + 1; // *p는 l-value와 r-value 어느 위치에도 사용 가능
&n = 3; // 컴파일 오류 발생: &n은 l-value로는 사용할 수 없으므로

```

- '*포인터변수'는 l-value와 r-value로 모두 사용이 가능하나, 주소값인 '&변수'는 r-value로만 사용이 가능하다.
- '*포인터변수'와 같이 간접연산자는 포인터 변수에만 사용이 가능하나, 주소연산자는 '&변수'와 같이 모든 변수에 사용이 가능하다.

3. 간접연산자 *와 포인터 연산

◆ 주소 연산

- ❖ 간단한 더하기와 뺄셈 연산으로 이웃한 변수의 주소 연산을 수행
 - 절대적인 주소의 계산이 아니며, 변수 자료형의 상대적인 위치에 대한 연산
- ❖ $p+1$: p 가 가리키는 자료형의 다음 주소로 실제 주소 값
 - $p + [\text{자료형크기(바이트)}]$
- ❖ $p+2$: p 가 가리키는 자료형의 다음 다음 주소로 실제 주소 값
 - $p + [\text{자료형크기(바이트)}] * 2$
- ❖ $p+i$: p 가 가리키는 자료형의 다음 i 번째 주소로 실제 주소 값
 - $p + [\text{자료형크기(바이트)}] * i$

3. 간접연산자 *와 포인터 연산

◆ 주소 연산

절대 주소	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115
char형	*p	*(p+1)	*(p+2)	*(p+3)	*(p+4)	*(p+5)	*(p+6)	*(p+7)	*(p+8)	*(p+9)	*(p+10)	*(p+11)	*(p+12)	*(p+13)	*(p+14)	*(p+15)
	p	p+1	p+2	p+3	p+4	p+5	p+6	p+7	p+8	p+9	p+10	p+11	p+12	p+13	p+14	p+15
short형	*p		*(p+1)		*(p+2)		*(p+3)		*(p+4)		*(p+5)		*(p+6)		*(p+7)	
	p		p+1		p+2		p+3		p+4		p+5		p+6		p+7	
int형	*p				*(p+1)				*(p+2)				*(p+3)			
	p				p+1				p+2				p+3			
double형	*p								*(p+1)							
	p								p+1							

```

01  #include <stdio.h>
02
03  int main(void)
04  {
05      char* pc = (char*) 100;          // 100을 주소값으로 변환해 저장
06      int* pi = (int*) 100;            // 100을 주소값으로 변환해 저장
07      double* pd = (double*) 100;     // 100을 주소값으로 변환해 저장
08      //pd = 100;                      // double 포인터에 100으로 저장하면 경고 발생
09
10      printf("%lld %lld %lld\n", (long long)(pc - 1), (long long)pc,
11              (long long)(pc + 1));
12      printf("%llu %llu %llu\n", (unsigned __int64)(pi - 1),
13              (unsigned __int64)pi, (unsigned __int64)(pi + 1));
14      printf("%llu %llu %llu\n", (unsigned long long)(pd - 1),
15              (unsigned long long)pd, (unsigned long long)(pd + 1));
16
17      return 0;
18  }

```

double 형 포인터 변수 pd에 저장된 값 100과 비교하여 pd-1과 pd+1을 출력,
double의 크기가 8이므로 각각 8만큼의 차이가 나므로 92 100 108 출력

결과

```

99  100  101
96  100  104
92  100  108

```

char형 포인터 변수 pc에 저장된 값 100과 비교하여 pc-1과 pc+1을 출력,
char의 크기가 1이므로 각각 1만큼의 차이가 나므로 99 100 101 출력

I. 포인터 변수와 선언

1교시 수업을 마치겠습니다.





Ⅱ. 포인터 형변환과 다중 포인터

1. 변수의 내부 저장 표현
2. 다중 포인터와 증감연산자의 활용
3. 포인터 상수

1. 변수의 내부 저장 표현 ❖ 변수 value에 16진수 0x61626364를 저장

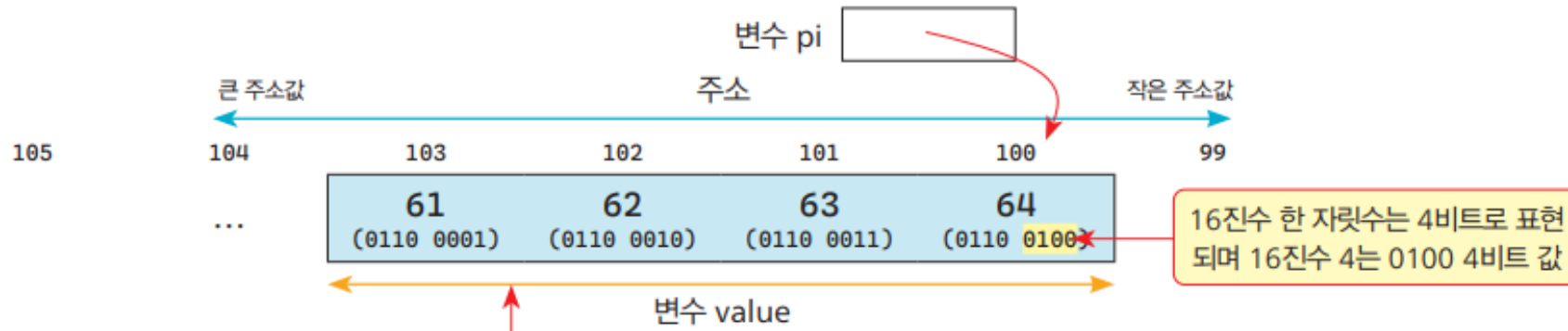
- 변수 value의 주소가 100번지
 - 100번지 1바이트 내부에 16진수 64가 저장
 - 다음 주소 101번지에는 63이 저장
 - 다음에 각각 62, 61이 저장
 - 즉 자연스럽게 0x61626364의 수가 큰 주소 값에서 작은 주소 값으로 저장
 - 반환 주소 값은 가장 작은 주소 값에 해당

```
int value = 0x61626364; // 정수의 일부분인 코드 61은 문자 'a'
```

```
int *pi = &value;
```

```
printf("%#x %d\n", value, value);
```

다음이 출력으로 10진수 값은 16진수 0x61626364에 해당하는 10진수 0x61626364 1633837924



4바이트의 정수 int형 저장공간으로 주소값 100번지에서 시작하는 내부에 16진수 0x61626364가 저장

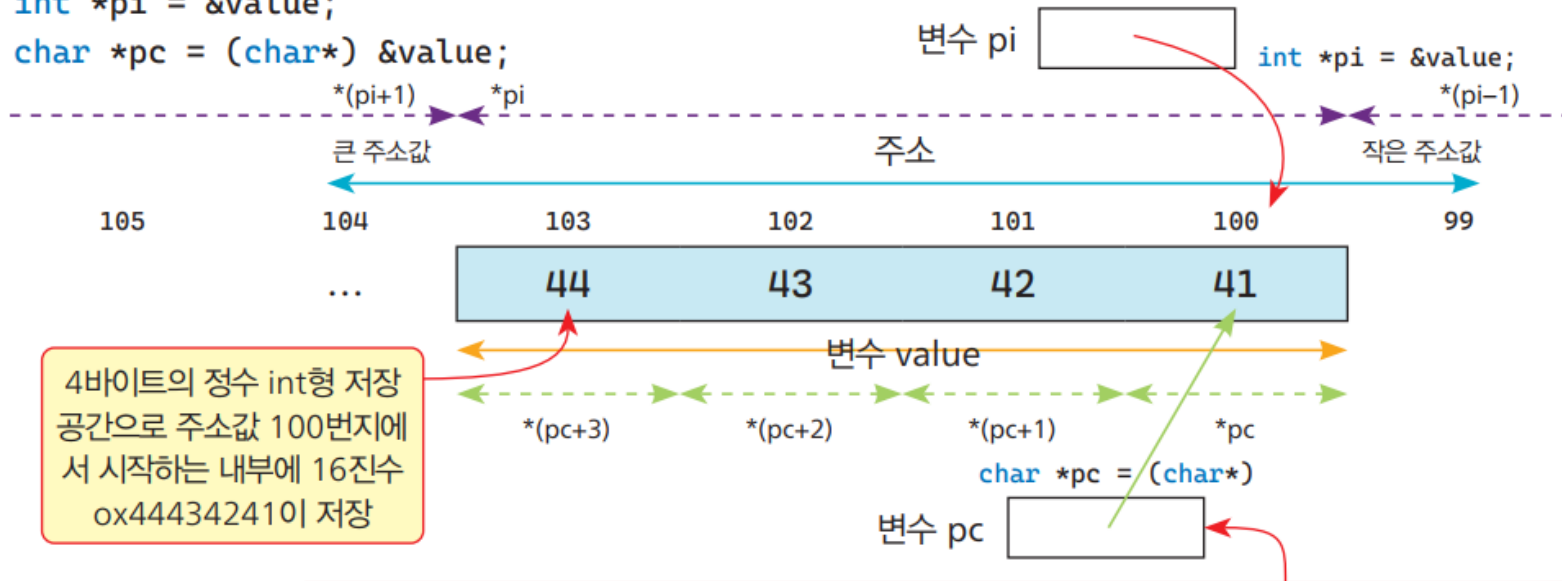
1. 변수의 내부 저장 표현

◆ 명시적 형변환

```
int value = 0x44434241; // 정수의 일부분인 코드 41은 문자 'A'
int *pi = &value;
char *pc = &value;
```

warning C4133: '초기화중': 'int *'과(와) 'char *' 사이의 형식이 호환되지 않습니다.

```
int value = 0x44434241; // 정수의 일부분인 코드 41은 문자 'A'
int *pi = &value;
char *pc = (char*) &value;
```



4바이트의 정수 `int`형 저장 공간으로 주소값 100번지에서 시작하는 내부에 16진수 `0x44434241`이 저장

`char*`인 `pc`는 주소값 100의 1바이트만 참조하며, `*(pc+1)`는 다음 `char`인 101번지 1바이트를 참조

1. 변수의 내부 저장 표현

Prj06 06ptypcast.c 포인터 자료형의 변환 난이도: ★

```

01 #include <stdio.h>
02
03 int main(void)
04 {
05     //int value = 0x61626364;    // 정수의 일부분인 코드 61은 문자 'a'
06     int value = 0x44434241;    // 정수의 일부분인 코드 41은 문자 'A'
07     printf("저장 값: %#x(16진수) %d(10진수)\n\n", value, value);
08
09     int *pi = &value;
10     char *pc = (char*) &value;
11
12     printf("변수명   저장값       주소값\n");
13     printf("-----\n");
14     printf(" value   %#x   %llu\n\n", value, (uintptr_t)pi); // 정수 int형 출력
15
16     printf("간접참조 코드 문자   주소값\n");
17     printf("-----\n");
18     //문자 포인터로 정수 내부의 문자 출력 모듈
19     for (int i = 0; i <= 3; i++)
20     {
21         char ch = *(pc + i);
22         printf(" *(pc+%d) %#x %3c %llu\n", i, ch, ch, (uintptr_t)(pc + i));
23     }
24
25     return 0;
26 }

```

4바이트인 10진수를 1바이트씩만 분리해서 출력한다면 16진수 44는 문자 'D'에 해당

char 포인터 pc를 선언하여 int 변수 value의 주소를 char 포인터로 형변환하여 저장, 이제 pc는 char 포인터이므로 1바이트씩 이동 가능

주소값을 10진수로 출력하려면 (uintptr_t)로 변환해 출력

char 변수 ch에 (pc+i)가 가리키는 문자를 저장하며, i가 0에서 3까지 반복되므로 pc가 가리키는 문자에서 부터 이웃한 3개, 총 4개 문자를 순서로 대입

저장 값: 0x44434241(16진수) 1145258561(10진수)

변수명	저장값	주소값
value	0x44434241	287588219092
간접참조	코드	문자
*(pc+0)	0x41	A
*(pc+1)	0x42	B
*(pc+2)	0x43	C
*(pc+3)	0x44	D

2. 다중 포인터와 증감연산자의 활용

◆ 이중 포인터

```
int i = 20;  
int *pi = &i;  
int **dpi = &pi;
```

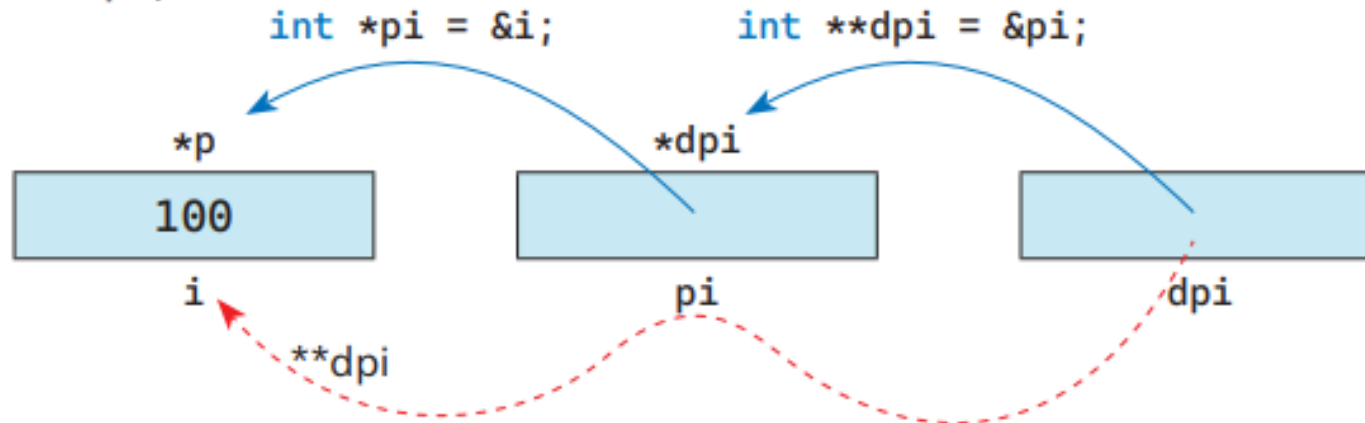


그림 11-15 이중 포인터의 메모리와 변수

2. 다중 포인터와 증감연산자의 활용

- ❖ 이중 포인터 변수 dpi
 - `**dpi`가 바로 변수 `i`
- ❖ 문장 `*pi = i + 2;`
 - 변수 `i`를 2 증가
- ❖ 문장 `**dpi = *pi + 2;`
 - 변수 `i`를 2 증가

실습예제 11-7
Prj07 07multptr.c 이중 포인터를 이용한 변수의 참조 난이도: ★

```

01  #include <stdio.h>
02
03  int main(void)
04  {
05      int i = 100;
06      int *pi = &i;    // 포인터 선언
07      int **dpi = &pi; // 이중 포인터 선언
08
09      printf("%p %p %p\n", &i, pi, *dpi);
10
11      *pi = i + 30;    // i = i + 30;
12      printf("%d %d %d\n", i, *pi, **dpi);
13
14      **dpi = *pi + 30; // i = i + 30;
15      printf("%d %d %d\n", i, *pi, **dpi);
16
17      return 0;
18  }
```

모두 변수 i의 주소값을 참조하는 방식

모두 변수 i 자체를 참조하는 방식

결과

00000066683CF664 00000066683CF664 00000066683CF664
 130 130 130
 160 160 160

2. 다중 포인터와 증감연산자의 활용

❖ 연산자 우선순위

우선순위	단항연산자	설명	결합성(계산 방향)
1	a++ a--	후위 증가, 후위 감소	-> (좌에서 우로)
2	++a --a & *	전위 증가, 전위 감소 주소 간접 또는 역참조	<- (우에서 좌로)

❖ 여러 연산 방법

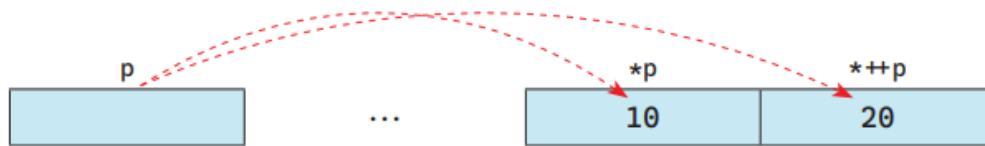


표 11-2 증가연산자 ++와 간접연산자 *의 사용 사례

연산식		결과값		연산 후	
				주소값 p 이동	*p
*p++	*(p++)	10	*p : p의 간접참조 값	p + 1 : p 다음 주소	20
*++p	*(++p)	20	*(p + 1) : (p + 1) 간접참조 값	p + 1 : p 다음 주소	20
(*p)++		10	*p : p의 간접참조 값	p : 변화 없음	11
++*p	++(*p)	11	*p + 1 : *p에 1 더하기	p : 변화 없음	11

2. 다중 포인터와 증감연산자의 활용

Prj08 08varptrop.c 포인터 변수에 대한 다양한 연산

```

01 #include <stdio.h>
02
03 int main(void)
04 {
05     int a[] = {10, 20};
06     int *p = &a[0]; // 배열의 첫 번째 원소 포인터 선언
07     printf("%p %d %p %d\n\n", p, *p, p+1, *(p+1));
08
09     printf("%d\n", *p++); // *(p++) 동일
10     printf("%p %d\n", p, *p);
11
12     p = &a[0]; // 다시 배열의 첫 번째 원소 주소값 대입
13     printf("%d\n", **p); // **(*p) 동일
14     printf("%p %d\n\n", p, *p);
15
16     p = &a[0]; // 다시 배열의 첫 번째 원소 주소값 대입
17     printf("%d\n", (*p)++);

```

```

18     printf("%p %d\n", p, *p);
19
20     a[0] = 10; // 다시 배열의 첫 번째 원소에 10 저장
21     p = &a[0]; // 다시 배열의 첫 번째 원소 주소값 대입
22     printf("%d\n", **p); // **(*p) 동일
23     printf("%p %d\n\n", p, *p);
24
25     return 0;
26 }

```

```
0000002F9CDBF6D8 10 0000002F9CDBF6DC 20
```

10

0000002F9CDBF6DC 20

20

0000002F9CDBF6DC 20

10

0000002F9CDBF6D8 11

11

0000002F9CDBF6D8 11

3. 포인터 상수

◆ 포인터 변수도 `const`를 사용해 포인터 상수로 생성

- ❖ 위치는 세가지(3) 종류가 있지만
- ❖ 첫 번째와 두 번째는 같은 의미이므로 두 가지 방식이 존재

```
int i = 10, j = 20;
```

① `const int *p = &i;`

`*p = 20; //오류 발생`

• (지역 변수) `const int *p`
`*p`가 상수로 `*p`로 수정할 수 없음
[온라인 검색](#)

② `int const *p = &i;`

`*p = 20; //오류 발생`

• (지역 변수) `const int *p`
`*p`가 상수로 `*p`로 수정할 수 없음
[온라인 검색](#)

③ `int* const p = &i;`

`p = &j; //오류 발생`

3. 포인터 상수

실습예제 11-9

Prj09

09constptr.c

포인터와 포인터 간접참조를 상수로 선언

난이도: ★★

```

01  #include <stdio.h>
02
03  int main()
04  {
05      int i = 10, j = 20;
06      const int* p = &i; /*p가 상수로 *p로 수정할 수 없음
07      /*p = 20; //오류 발생
08      printf("%d ", *p);
09      p = &j;
10      printf("%d\n", *p);
11
12      double d = 7.8, e = 2.7;
13      double* const pd = &d;
14      //pd = &e; //pd가 상수로 다른 주소값을 저장할 수 없음
15      printf("%f ", *pd);
16      *pd = 4.4;
17      printf("%f\n", *pd);
18
19      return 0;
20  }
```

결과

10 20

7.800000 4.400000

Ⅱ. 포인터 형변환과 다중 포인터

2교시 수업을 마치겠습니다.





Ⅲ. 포인터를 사용한 배열 활용

1. 1차원 배열과 포인터
2. 2차원 배열과 간접연산자, 배열 포인터
3. 포인터 배열

1. 1차원 배열과 포인터

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

	<code>&score[0]</code>	<code>&score[1]</code>	<code>&score[2]</code>
주소값 참조	<code>score</code>	<code>score+1</code>	<code>score+2</code>
배열 score	5	10	15
	<code>score[0]</code>	<code>score[1]</code>	<code>score[2]</code>
저장 값 참조	<code>*score</code>	<code>*(score+1)</code>	<code>*(score+2)</code>

- 배열의 주소값(배열 첫 번째 원소의 주소값): `score`, `&score[0]`
- 배열 첫 번째 원소 저장 값: `*score`, `score[0]`
- 배열 (i+1)번째 원소 주소값: `(score + i)`, `&score[i]`
- 배열 (i+1)번째 원소 저장 값: `*(score + i)`, `score[i]`

1. 1차원 배열과 포인터

배열 초기화 문장		int score[] = {10, 20, 30};		
원소 값		10	20	30
배열원소 접근 방법	score[i]	score[0]	score[1]	score[2]
	*(score+i)	*score	*(score+1)	*(score+2)
주소값(첫 주소 + 배열원소 크기*i)		100	104 (100 + 1*4)	108 (100 + 2*4)
주소값 접근 방법	&score[i]	&score[0]	&score[1]	&score[2]
	score+i	score	score+1	score+2

```
int a[3] = {5, 10, 15};
int* p = a; //a = &a[0]

//포인터 변수 p 사용, 배열 원소 값 참조
printf("%d %d %d\n", *(p), *(p + 1), *(p + 2));
//위 포인터 변수 p에서 배열처럼 첨자를 사용 가능
printf("%d %d %d\n", p[0], p[1], p[2]);
```

- 포인터 p에 &a[0]를 저장하면 연산식 *(p+i)로 배열원소를 참조할 수 있다.
- 특히 포인터 p로도 배열처럼 첨자를 이용해 p[i]로 배열원소를 참조할 수 있다.

1. 1차원 배열과 포인터

```
int a[3] = {5, 10, 15};  
//포인터 변수 p를 선언해 배열 a의 주소를 저장  
int* p = a; //a == &a[0]  
  
//a[0]을 출력 후, p 다음 주소로 증가  
printf("%d ", *p++);  /*(p++), 5 출력 후, p 다음 주소로 증가  
//a[1]을 출력  
printf("%d\n\n", *p); //10 출력
```

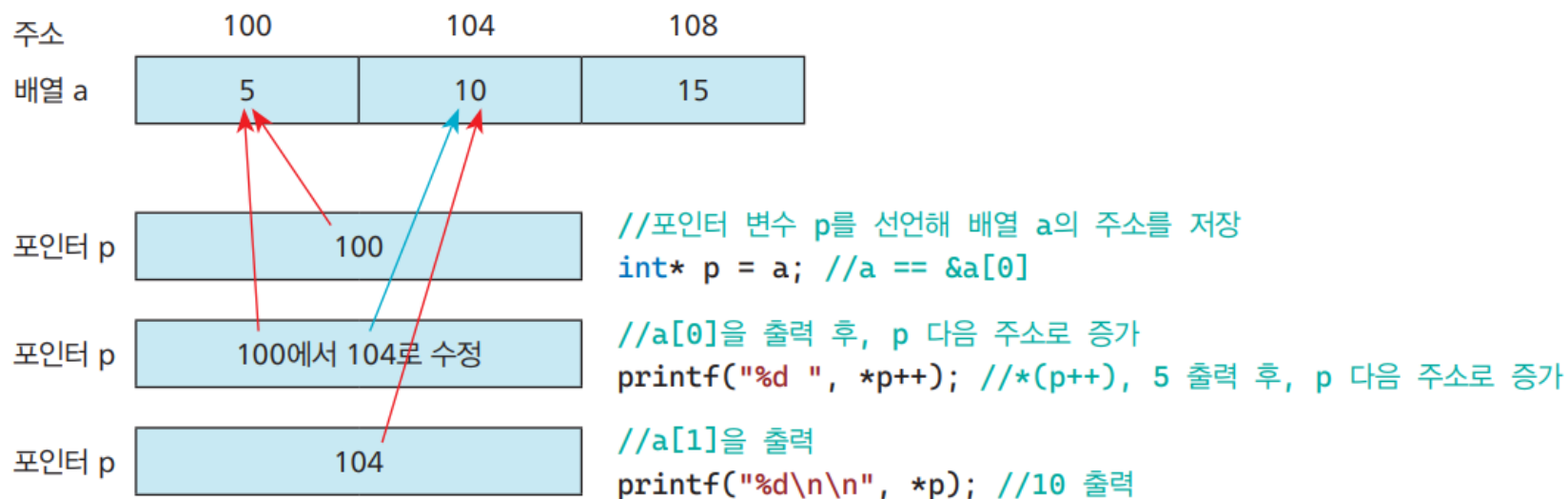


그림 11-16 연산식 *p++의 이해

1. 1차원 배열과 포인터

Prj10 10ptarray.c 1차원 배열에서 배열이름과 포인터를 사용해 원소와 주소값의 참조

난이도: ★★

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     int score[] = {10, 20, 30};
06     printf("%p %p\n", score, (score + 1));
07     printf("%d %d\n\n", *score, *(score + 1));
08
09     int a[3] = {5, 10, 15};
10     //포인터 변수 p를 선언해 배열 a의 주소를 저장
11     int* p = a; //a == &a[0]
12
13     //포인터 변수 p 사용, 배열 원소 값 참조
14     printf("%d %d %d\n", *(p), *(p + 1), *(p + 2));
15     //포인터 변수 p에서 배열처럼 첨자를 사용 가능
16     printf("%d %d %d\n", p[0], p[1], p[2]);
17     //a[0]을 출력 후, p 다음 주소로 증가
18     printf("%d ", *p++); //*(p++), 5 출력 후, p 다음 주소로 증가
19     //a[1]을 출력
20     printf("%d\n\n", *p); //10 출력
21
22     p = &a[2]; // &a[2] == a + 2
```

```
23 //a[2]를 출력 후, p 이전 주소로 감소
24 printf("%d ", *p--); // *(p--), 15 출력 후, p 이전 주소로 감소
25 //a[1]을 출력하고 하나 감소
26 printf("%d\n", (*p)--); // 10 출력 후, 1 감소해 9 저장
27 //현재 포인터 변수 p는 a[1]을 가리키고 있으며 다음으로 배열 모두 출력
28 printf("%d %d %d\n", *(p - 1), *p, *(p + 1)); // 5 9 15
29 //p와 첨자를 사용 가능, 상대적인 음수도 가능
30 printf("%d %d %d\n", p[-1], p[0], p[1]); // 5 9 15
31
32 return 0;
33 }
```

*p--는 다르며, 현재 p가 가리키는 저장 값을 참조하고 p의 주소를 이전 주소로 수정

000000247F2FF688 000000247F2FF68C

10 20

5 10 15

5 10 15

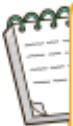
5 10

15 10

5 9 15

5 9 15

1. 1차원 배열과 포인터

**NOTE:** 버전 C99 추가 기능

포인터 변수에 형변환 연산자를 사용해 직접 배열 초기화 형태의 원소 초기값을 저장할 수 있다.

```
int *p = (int[]) { 3, 0, 3, 1, 4 };
```

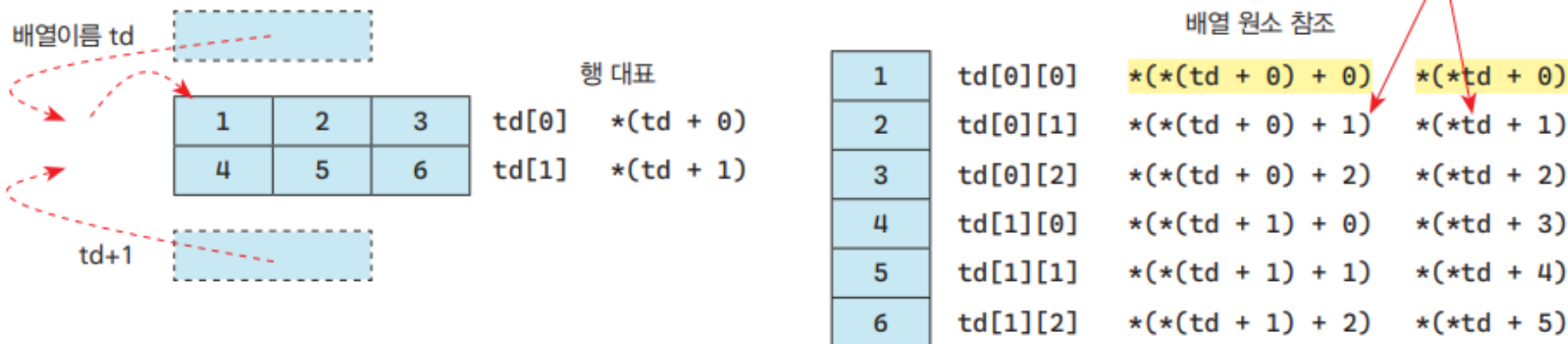
포인터 변수 p를 사용해 배열 형태의 저장 값을 모두 출력할 수 있다.

```
for (int i = 0; i < 5; i++) {  
    printf("%d ", *p++);  
}
```

2. 2차원 배열과 간접연산자, 배열포인터

- 포인터 상수 `td[0]`
 - 배열의 첫 행을 대표
 - 첫 번째 원소 `td[0][0]`의 주소 값 `&td[0][0]`을 갖는 포인터 상수
- 그러므로 배열이름인 `td`
 - 포인터의 포인터인 이중 포인터
- `td+1`은 두 번째 행을 대표하는 주소 값
 - `sizeof(td[0]), sizeof(td[1])`
 - 각각 첫 번째 행과 두 번째 행의 바이트 크기를 반환

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



- `td[1]`
 - 두 번째 행의 첫 원소의 주소
 - `*td[1]`로 `td[1][0]`를 참조
- `*td`
 - `td[0]`로 첫 번째 행의 일차원 배열을 의미
- `*td+1`
 - 그 다음 원소의 주소 값
- 그러므로 `*td+i`
 - 배열의 (i+1)번째 원소의 주소 값
 - `*(td + i)`
 - 원소의 저장 값
- `**td = 10;`
 - 첫번째 원소 `td[0][0]`의 값을 10으로 수정
 - `td`가 이중 포인터
 - `*(td + 0)`으로 간접연산자 `*`이 2개 필요

```

Prj11      11tdary.c      2차원 배열에서 배열이름과 포인터를 사용한 원소와 주소값의 참조
01  #include <stdio.h>
02
03  #define ROW 2
04  #define COL 3
05
06  int main(void)
07  {
08      int td[][COL] = { { 1, 2, 3 }, { 4, 5, 6 } };
09      printf("%d\n", sizeof(td));
10      printf("%d %d\n", sizeof(td[0]), sizeof(td[1]));
11      printf("%d %d\n", sizeof(*td), sizeof(*(td+1)));
12      printf("%p %p\n", td, td + 1);
13      printf("%p %p\n", *td, *(td + 1));
14
15      for (int i = 0, cnt = 0; i < ROW; i++)
16      {
17          printf("%p %p ", td[i], *(td + i));
18          for (int j = 0; j < COL; j++, cnt++)
19              printf("%d %d %d ", *(td + cnt), *(td[i] + j), (*(td + i) + j));
20          printf("\n");
21      }
22
23      **td = 10; //td[0][0] = 10;
24      *(*td + 4) = 20; //td[1][1] = 20;
25      (*(td + 1) + 2) = 30; //td[1][2] = 30;
26      printf("%d %d %d\n", td[0][0], td[1][1], td[1][2]);
27
28      return 0;
29  }
  
```

(`*td + 4`)는 `&td[1][1]`로 다섯 번째 원소의 주소값

(`*(td + 1) + 2`)는 `&td[1][2]`로 여섯 번째 원소의 주소값

```

24
12 12
12 12
0000007CE3CFFC38 0000007CE3CFFC44
0000007CE3CFFC38 0000007CE3CFFC44
0000007CE3CFFC38 0000007CE3CFFC38 1 1 1 2 2 2 3 3 3
0000007CE3CFFC44 0000007CE3CFFC44 4 4 4 5 5 5 6 6 6
10 20 30
  
```

배열의 첫 번째의 주소와 두 번째의 주소

2. 2차원 배열과 간접연산자, 배열포인터

- 일차원 배열 `int a[]`의 주소
 - `(int *)`인 포인터 변수에 저장 가능
- 문장 `int (*ptr)[4];`로 선언
 - 배열 포인터(pointer to array) 변수 `ptr`
 - 괄호 `(*ptr)`은 반드시 필요
 - 열이 4인 이차원 배열 `ary[][4]`의 주소 저장 가능
 - 대괄호 사이의 4는 포인터가 가리키는 이차원 배열에서의 열 크기
 - 이차원 배열의 주소를 저장하는 포인터 변수는 열 크기에 따라 변수 선언이 달라짐

원소자료형 *변수이름;
 변수이름 = 배열이름;
 또는
 원소자료형 *변수이름 = 배열이름;

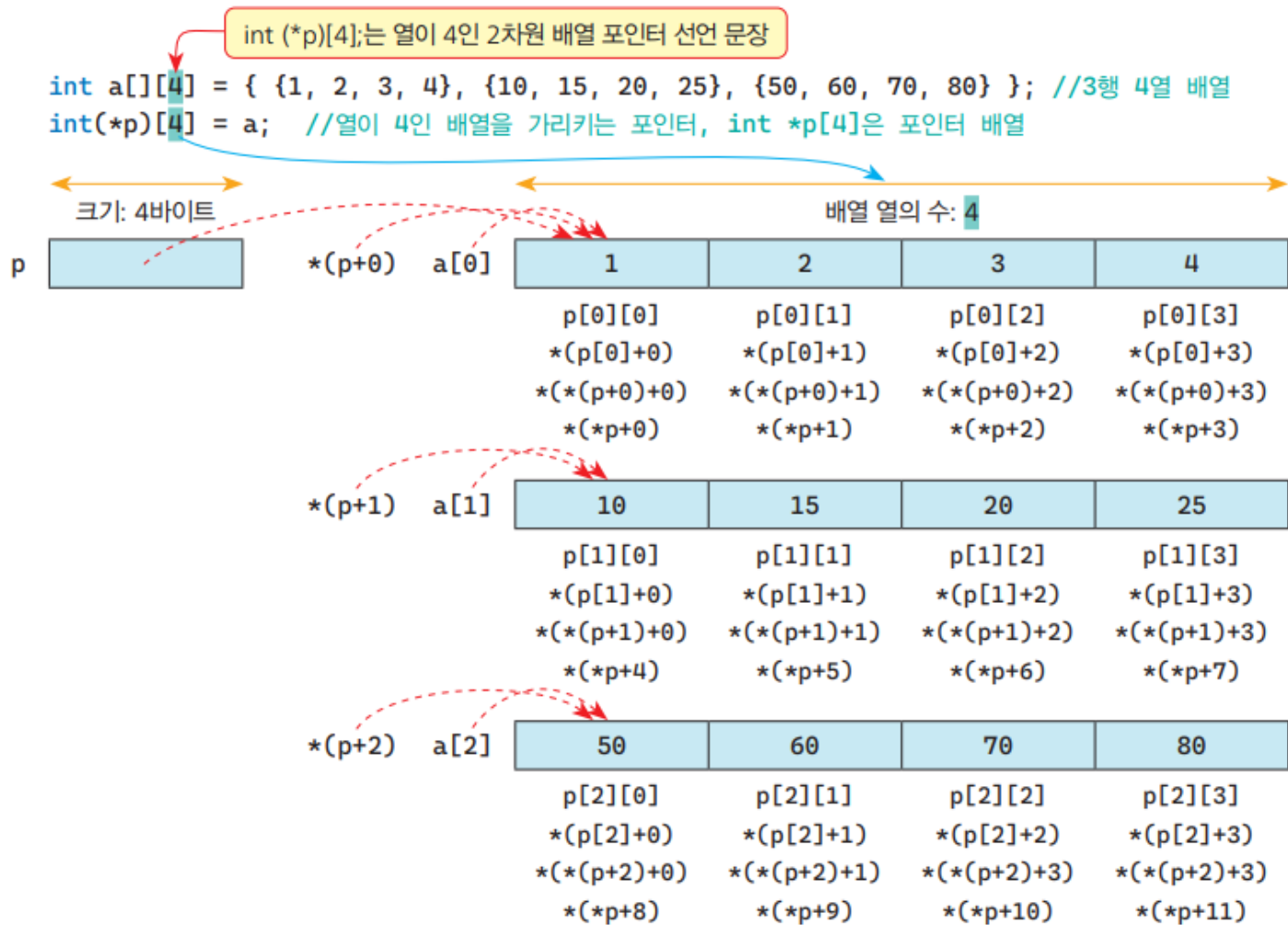
```
int a[] = {8, 2, 8, 1, 3};
int *p = a;
```

원소자료형 (*변수이름)[배열열크기];
 변수이름 = 배열이름;
 또는
 원소자료형 (*변수이름)[배열열크기] = 배열이름;

```
int ary[][4] = {5, 7, 6, 2, 7, 8, 1, 3};
int (*p)[4] = ary;    //열이 4인 배열을 가리키는 포인터
//int *p[4] = ary;    //포인터 배열
```


2. 2차원 배열과 간접연산자, 배열포인터

- 괄호가 없는 `int *p[4];`은 다음에 배울 `int`형 포인터 변수 4개를 선언하는 포인터 배열 선언 문장이다.



- 이차원 배열 배열이름 ary와 배열 포인터 ptr
 - 연산자 sizeof 결과 값은 서로 다름
 - 즉 sizeof(a)는 배열의 총 크기인 $48 = (4 \times 3 \times 4)$
 - sizeof(p)는 단순히 포인터의 크기인 8(64비트 시스템인 경우)
- 연산식 $*(p[i] + j)$
 - (i+1) 행, (j+1)열 원소 참조
- 연산식 $*(*(p + i) + j)$
 - (i+1) 행, (j+1)열 원소 참조
- 연산식 $**p++$
 - 연산 우선순위에 따라 $**(p++)$
 - 현재 포인터가 가리키는 원소를 참조하고
 - p를 하나 증가시켜
 - 다음 행의 첫 원소를 가리키게 하는 연산식

```

01  #include <stdio.h>
02
03  int main(void)
04  {
05      int ary[][4] = { {10, 20, 30, 40}, {50, 60, 70, 80} }; //2행 4열 배열
06      int(*ptr)[4] = ary;    //열이 4인 배열을 가리키는 포인터, int *ptr[4]은 포인터 배열
07
08      printf("%zd %zd\n", sizeof(ary), sizeof(ptr));
09      printf("%zd %zd\n\n", sizeof(ary[0]), sizeof(ptr[0]));
10
11      printf("%2d %2d\n", **ary, **ptr); //첫 번째 원소, 10
12      printf("%2d %2d\n", **(ary + 1), *ary[1]); //두 번째 행의 첫 원소, 50
13      printf("%2d %2d\n", *(ptr + 1), *ptr[1]); //두 번째 행의 첫 원소, 50
14      printf("%2d %2d\n", *(ary[1] + 1), *(ptr[1] + 1)); //2행 2열, 60
15      printf("%2d %2d\n\n", (*(ary + 1) + 3), (*(ptr + 1) + 3)); //2행 4열, 80
16
17      printf("%2d ", **ptr++); //배열의 첫 원소 10 참조 후, ptr의 다음 행으로 주소 수정
18      printf("%2d\n", **ptr); //두 번째 행의 첫 원소 50 참조
19
20      return 0;
21  }

```

포인터의 크기는 8바이트

```

32 8
16 16

10 10
50 50
50 50
60 60
80 80

10 50

```

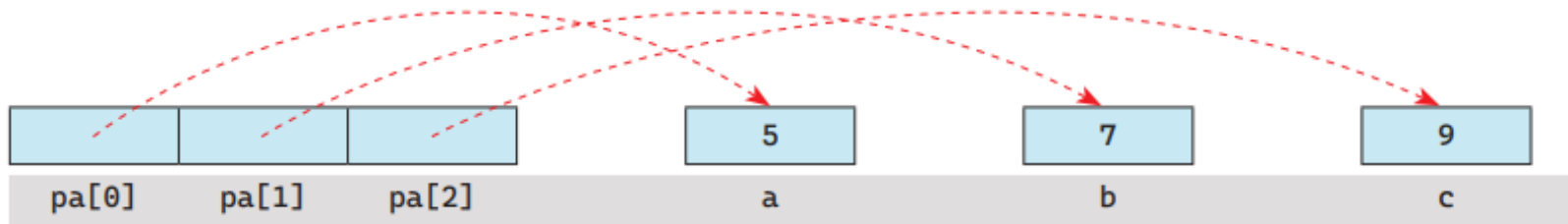
3. 포인터 배열

❖ 포인터를 배열 원소로 하는 배열

```
int a = 5, b = 7, c = 9;
```

```
int *pa[3];
```

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



포인터 배열 변수 선언

자료형 *변수이름[배열크기] ;

```
int *pary[5];
char *ptr[4];
float a, b, c;
double *dary[5] = {NULL};
float *ptr[3] = {&a, &b, &c};
```

3. 포인터 배열

- 포인터 배열 pary
 - 표준입력을 받아
 - 포인터 배열 원소들이 가리키는 변수 a, b, c로 출력
- 반복문 내부 scanf()
 - pary[i]가 저장할 주소
 - 그대로 사용

실습예제 11-13

Prj13

13aryptr.c

2차원 배열을 가리키는 배열 포인터의 선언과 이용

난이도: ★★

```

01 #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
02 #include <stdio.h>
03
04 #define SIZE 3
05
06 int main(void)
07 {
08     //포인터 배열 변수 선언
09     int* pary[SIZE] = { NULL };
10     int a = 10, b = 20, c = 30;
11
12     pary[0] = &a;
13     pary[1] = &b;
14     pary[2] = &c;
15     for (int i = 0; i < SIZE; i++)
16         printf("*pary[%d] = %d\n", i, *pary[i]);
17
18     for (int i = 0; i < SIZE; i++)
19     {
20         scanf("%d", pary[i]);
21         printf("%d, %d, %d\n", a, b, c);
22     }
23
24     return 0;
25 }

```

정수를 입력하고 엔터 키를 누르면 진행

결과

```

*pary[0] = 10
*pary[1] = 20
*pary[2] = 30

```

```

1
1, 20, 30

```

```

2
1, 2, 30

```

```

3
1, 2, 3

```

정수를 입력하고 엔터 키를 누르면 진행

Ⅲ. 포인터를 사용한 배열 활용

3교시 수업을 마치겠습니다.

