

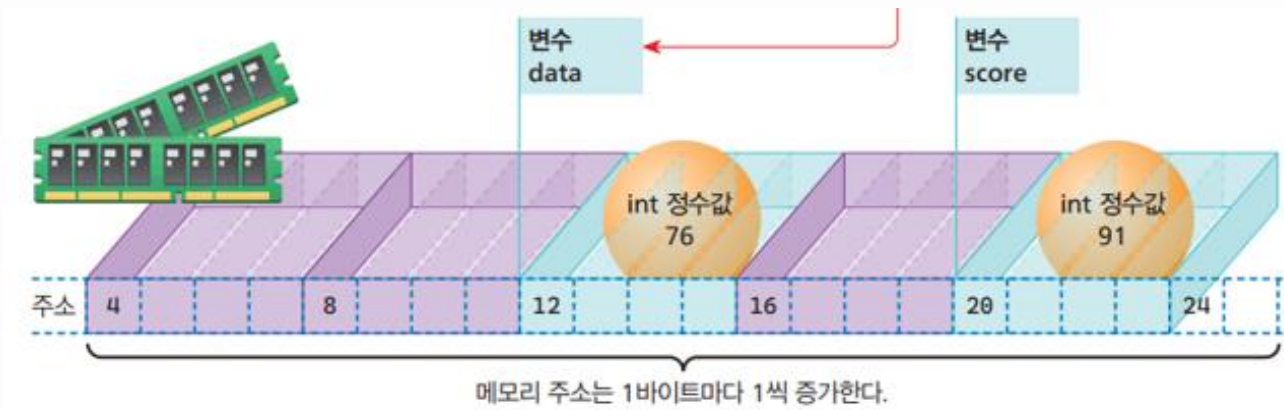
제 11 장 포인터 기초

- 01 포인터 변수와 선언
- 02 간접연산자 *와 포인터 연산
- 03 포인터 형변환과 다중 포인터
- 04 포인터를 사용한 배열 활용

주소 개념

- 고유한 주소(address)

- 메모리 주소는 저장 장소인 변수이름과 함께 기억장소를 참조하는 또 다른 방법



메모리 주소연산자와 주소 출력

정수 입력: 100

입력 값: 100

주소값: 000000B8DA52FC34(16진수)

주소값: 793936854068(10진수)

주소값 크기: 8

64비트 시스템에서 주소값은 8바이트(64비트)

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);	64비트 시스템에서 16개의 16진수 주소값이 출력	
	10	printf("입력 값: %d\n", input);		
	11	printf("주소값: %p(16진수)\n", &input);	주소값을 10진수로 출력하기 위해 uintptr_t로 변환해 출력	
	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	}		
결과		정수 입력: 100		
		입력 값: 100		
		주소값: 000000B8DA52FC34(16진수)		
		주소값: 793936854068(10진수)		
		주소값 크기: 8	%zu : "size of unsigned integer"	
			64비트 시스템에서 주소값은 8바이트(64비트)	

포인터 변수

- 포인터 변수 : ...
- 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* pptr;
06     pptr = &data;
07     printf("변수명   주소값            저장값\n");
08     printf("-----\n");
09     printf(" data   %p   %d\n", &data, data);
10     printf("pptr   %p   %p\n", &pptr, pptr);
11     printf("%zu\n", sizeof(pptr));
12     return 0;
13 }
```

결과	변수명	주소값	저장값
----	-----	-----	-----

다양한 자료형 포인터 변수

```
char c = '@';  
int m = 100;  
double x = 5.83;  
  
char *pc = &c;  
int *pm = &m;  
double *px = &x;  
  
printf("%3s %12p %9c\n", "c", pc, c);  
printf("%3s %12p %9d\n", "m", pm, m);  
printf("%3s %12p %9f\n", "x", px, x);
```



여러 포인터 변수 선언과 NULL 주소값 대입

```
int *ptr1, *ptr2, *ptr3; //ptr1, ptr2, ptr3 모두 int형 포인터임  
int *ptr1, ptr2, ptr3;  //ptr1은 int형 포인터이나 ptr2와 ptr3는 int형 변수임
```

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

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

```
int *ptr1 = NULL, *ptr2 = NULL, *ptr3 = NULL;  
int *ptr4, ptr5, ptr6;
```

```
printf("ptr1: %p\n", (void*)ptr1);  
printf("ptr2: %p\n", (void*)ptr2);  
printf("ptr3: %p\n", (void*)ptr3);
```

포인터 기초

01 포인터 변수와 선언

02 간접연산자 *와 포인터 연산

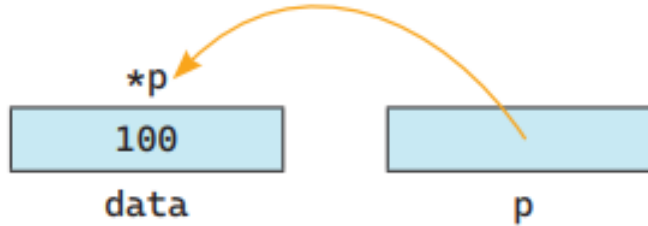
03 포인터 형변환과 다중 포인터

04 포인터를 사용한 배열 활용

간접연산자 *

- 간접연산자 (indirection operator) *를 사용한 역참조

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



- 포인터 p가 가리키는 변수가 data → *p은 변수 data 를 의미
 - *p로 data 변수 저장 장소인 l-value와 참조 값인 r-value로 참조 가능
 - 변수 data로 가능한 작업은 *p로도 가능
 - ex) *p = 200;

```
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
```

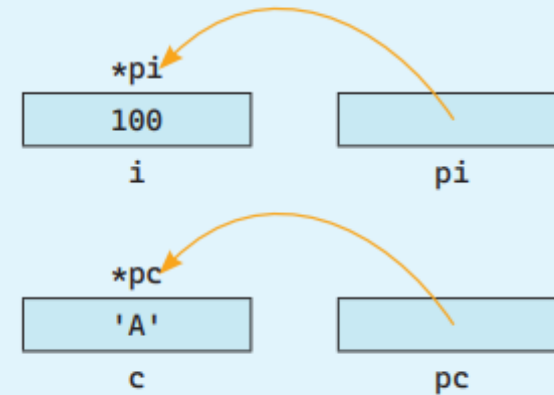
```
13     *pc = 'B'; //변수 c를 *pc로 간접참조하여 그 내용을 수정
```

```
14     printf("직접참조 출력: %d %c\n", i, c);
```

```
15
```

```
16     return 0;
```

```
17 }
```



**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.

포인터 변수의 연산

주소 연산

- 간단한 더하기와 뺄셈 연산으로 이웃한 변수의 주소 연산을 수행
 - 절대적인 주소의 계산이 아니며, 변수 자료형의 상대적인 위치에 대한 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														

```
int arr[] = { 1, 2, 3, 4, 5 };  
int length = sizeof(arr) / sizeof(arr[0]);  
int *p = arr;  
  
for (int i = 0; i < length; i++) {  
    printf("%p: %d\n", (void*)(p + i), *(p + i));  
}
```

```
int arr[] = { 10, 20, 30, 40, 50 };  
int *p1 = &arr[1];  
int *p2 = &arr[4];  
  
printf("p2 - p1: %ld\n", p2 - p1);
```

LAB 포인터를 이용하여 두 수의 값을 교환하는 프로그램

```
void swap(int* x, int* y)
{
    int t;
    t = *x;
    *x = *y;
    *y = t;
}

int main()
{
    int num1, num2;
    scanf_s("%d%d", &num1, &num2);

    swap(&num1, &num2);

    printf("%d,%d\n", num1, num2);
    return 0;
}
```

포인터 기초

01 포인터 변수와 선언

02 간접연산자 *와 포인터 연산

03 포인터 형변환과 다중 포인터

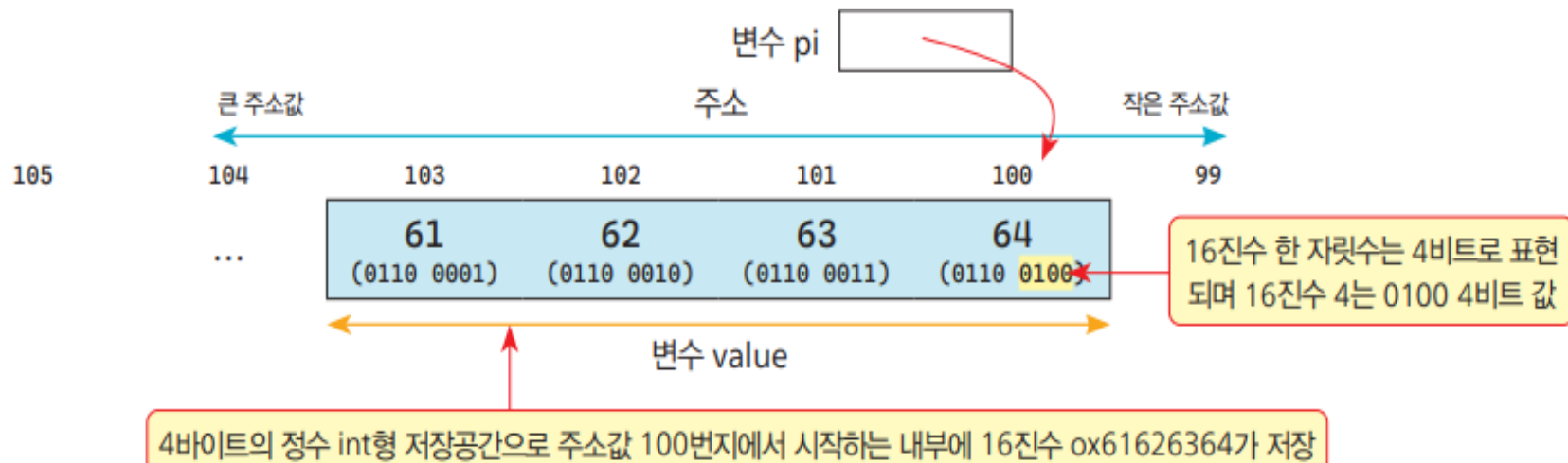
04 포인터를 사용한 배열 활용

변수의 내부 저장 표현

```
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

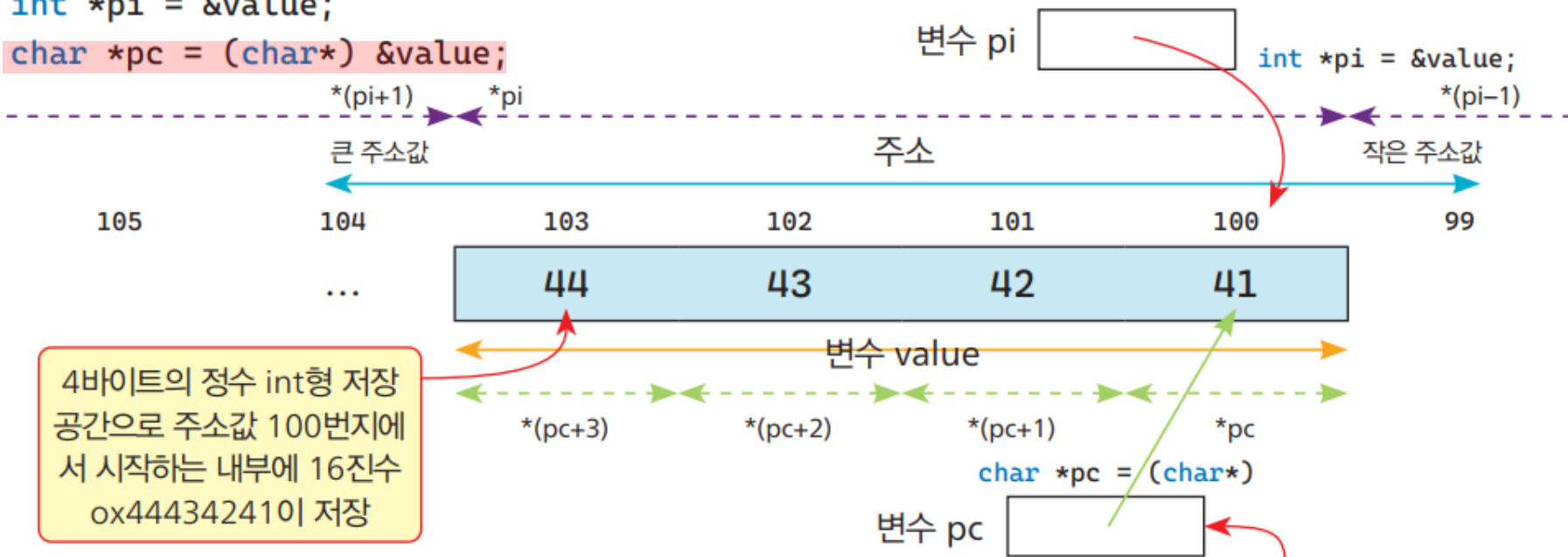


명시적 형변환

```
int value = 0x44434241; // 정수의 일부분인 코드 41은 문자 'A'
```

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

```
char *pc = (char*) &value;
```



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

포인터 자료형의 변환

```
int value = 0x44434241; // 41은 문자 'A'

char* pc = (char*)&value;

for (int i = 0; i <= 3; i++) {
    char ch = *(pc + i);
    printf(" *(pc+%d) %#x %3c %p\n", i, ch, ch, (void*)(pc + i));
}
```

```
*(pc+0) 0x41  A 0x7ffee9f7b44c
*(pc+1) 0x42  B 0x7ffee9f7b44d
*(pc+2) 0x43  C 0x7ffee9f7b44e
*(pc+3) 0x44  D 0x7ffee9f7b44f
```

포인터 기초 (I)

01 포인터 변수와 선언

02 간접연산자 *와 포인터 연산

03 포인터 형변환