7장 포인터

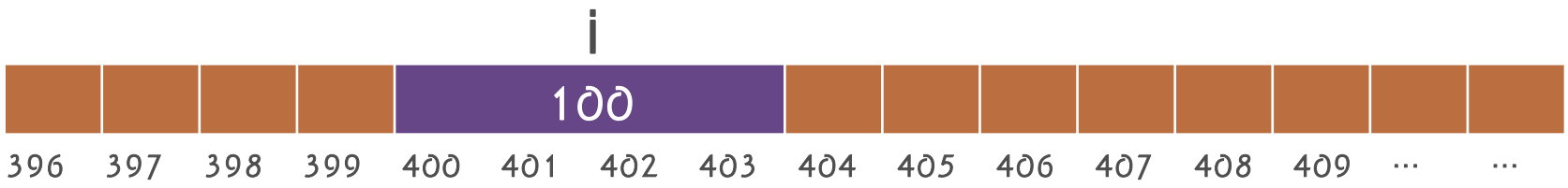
포인터

- 메모리는 각 바이트 별로 주소가 붙여진 1차원 배열
- 할당 받은 메모리 공간은 변수 이름으로 접근
  - 예 1

```
int a, b, c;          // a, b, c를 위한 메모리 할당
a = b + c;            // 변수 이름 a, b, c로 메모리 접근
```

  - 예 2

```
int  i;
i = 100;                    // 변수 i에 100 할당
```



포인터와 포인터 변수

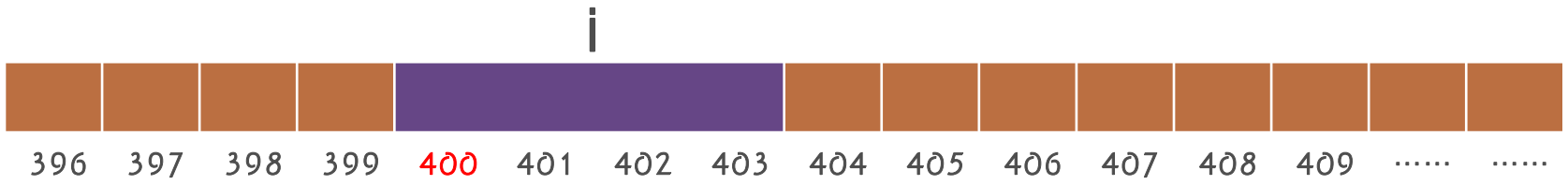
- **포인터**
  - 주소를 다루기 위한 자료형
  - 데이터를 가진 메모리 공간을 주소로 접근하기 위해 사용
- **포인터 변수**
  - 값으로 메모리 주소를 갖는 변수
  - 포인터 변수 선언 방법

```
int  *p;      // int 형 포인터 변수 p 선언
int  *a, b;   // int 형 포인터 변수 a 와 int 형 변수 b 선언
int  *a, *b;  // int 형 포인터 변수 a와 b 선언
```

주소연산자

- **주소연산자 &**
  - 메모리에 할당된 변수의 주소값을 알려주는 연산자

```
int  i;                          // i를 메모리에 할당
printf( "%u" , &i);              // 400 출력
```



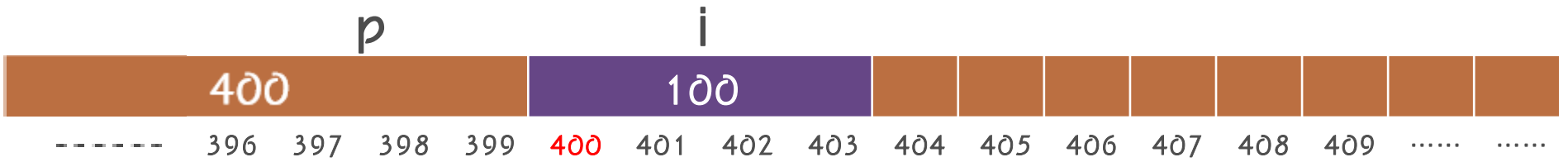
  - 상수나 수식 앞에는 주소연산자를 사용할 수 없음
    - 잘못 사용한 예

```
&3                  // 상수 앞에서 사용
&(i + 3)            // 수식 앞에서 사용
```

포인터 변수와 주소연산자

- **포인터 변수 사용**

```
int  i = 100, *p;
p = &i;                    // 변수 i 의 메모리 주소를 p에 할당
```



- p가 i를 가리킨다(포인트 한다)라고 표현

포인터 변수와 주소연산자

- **사용예제**

```
int i, *p;
register int v;

p = 0;                  // p = NULL;
p = &i;
p = 3000;               // 컴파일 시 경고, 상수에 사용
p = (int *)1776;        // int형 상수를 (int*)로 캐스트
p = &(i + 99);          // 컴파일 시 에러, 수식에 사용
p = &v                  // 오류, register 변수에 사용
```

역참조 연산자

- **역참조 연산자 ***
  - 포인터가 가리키는 메모리 공간에 접근하기 위한 연산자
  - 단항 연산자, 우에서 좌로의 결합순위
  - 사용 예

```
int  i = 100, *p;
p = &i;          // 변수 i 의 메모리 주소를 p에 할당
```


```
printf( "%d" , i);          // 100
printf( "%d" , *p );        // 100
```

예제 프로그램

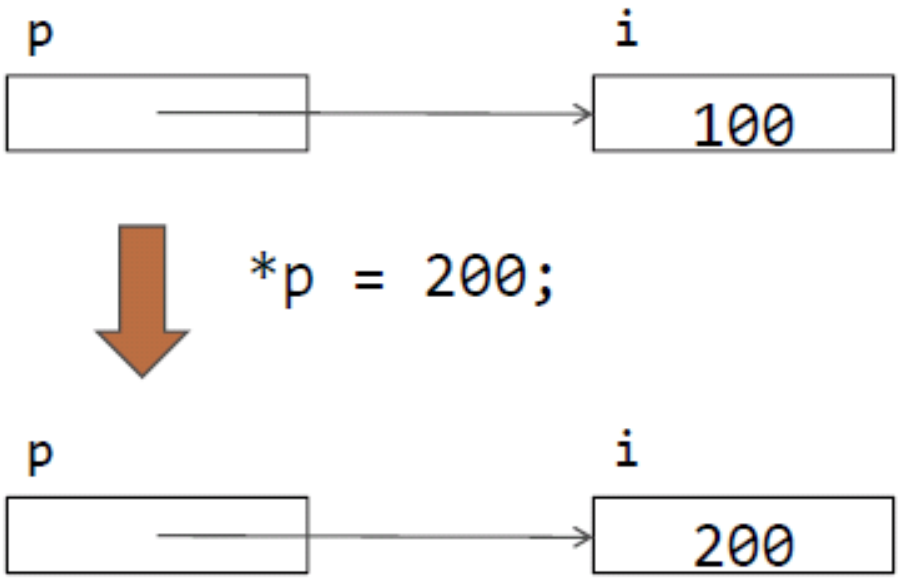
```c
#include <stdio.h>
int main(void){
    int i = 100;
    int *p;
    p = &i;
    printf("i 주소 : %p\n", &i);
    printf(" i 값  : %d\n", i);
    printf(" p 값  : %p\n", p);
    printf("*p 값  : %d\n", *p);
    return 0;
}
```

프로그램 결과

```
i 주소 : 0x23efc4  // 시스템 마다 다름
 i 값  : 100
 p 값  : 0x23efc4  // 시스템 마다 다름
*p 값  : 100
```

역참조 연산자

- 역참조 연산자가 붙은 포인터는 일반 변수와 같이 배정 연산자 좌측에 올 수 있음
  - 그 포인터가 가리키는 위치를 나타냄



예제 프로그램

```c
#include <stdio.h>
int main(void){
    int i, j = 5, *p;
    p = &i;
    i = 10;
    printf("(p = &i)i = %d, j = %d, *p = %d\n", i, j, *p);
    *p = *p * j;                    // i = i * j;
    printf("(p = &i)i = %d, j = %d, *p = %d\n", i, j, *p);
    p = &j;                         // *p == j
    printf("(p = &j)i = %d, j = %d, *p = %d\n", i, j, *p);
    return 0;
}
```

프로그램 결과

```
(p = &i)i = 10, j = 5, *p = 10
(p = &i)i = 50, j = 5, *p = 50
(p = &j)i = 50, j = 5, *p = 5
```

포인터 변수의 크기

- 시스템에 따라 상이함
  - sizeof 연산자로 계산
  - 포인터 변수가 가리키는 형과 관계없이 크기가 동일
  - 예

```
int i;
int *ip = &i;
double d;
double *dp = &d;
printf( "%ld %ld" , sizeof(i), sizeof(ip));   // 4 8
printf( "%ld %ld" , sizeof(d), sizeof(dp));   // 8 8
```

void 포인터 변수

- **void 형 포인터 변수**
  - 형이 없는 포인터

```
int *p;
float x;
void *v;
p = &x; // 오류, 두 형이 다름, 포인터는 자동 형 변환 안됨
v = &x;
p = v;
```

void 포인터 변수

- void 형 포인터 변수를 사용할 때에는 적절한 형 변환 필요

```
int i, j = 20;
void *v;

v = &j;
i = *((int *)v) + 10;
// v가 가리키는 곳의 데이터 형을 int 형으로 다룸
```

포인터의 포인터 변수

- 포인터를 가리키는 포인터 변수

```
int i = 100, *p = &i, **q = &p;
```

- q는 포인터의 포인터 변수
- 선언문에서 **로 명시
- *q : p
- **q : i

예제 프로그램

```
#include <stdio.h>
int main(void){
    int i = 100, j = 200, *p = &i, **q = &p;
    printf("i = %d, j = %d, *p = %d, **q = %d\n", i, j, *p, **q);
    printf("&i = %p, &j = %p, p = %p, *q = %p\n", &i, &j, p, *q);
    printf("&p = %p, q = %p\n", &p, q);
    *q = &j;     // p = &j
    printf("i = %d, j = %d, *p = %d, **q = %d\n", i, j, *p, **q);
    printf("&i = %p, &j = %p, p = %p, *q = %p\n", &i, &j, p, *q);
    printf("&p = %p, q = %p\n", &p, q);
    return 0;
}
```

i, j, p, q 변수의 주소는 각각 300, 400, 500, 600이라고 가정

실행결과

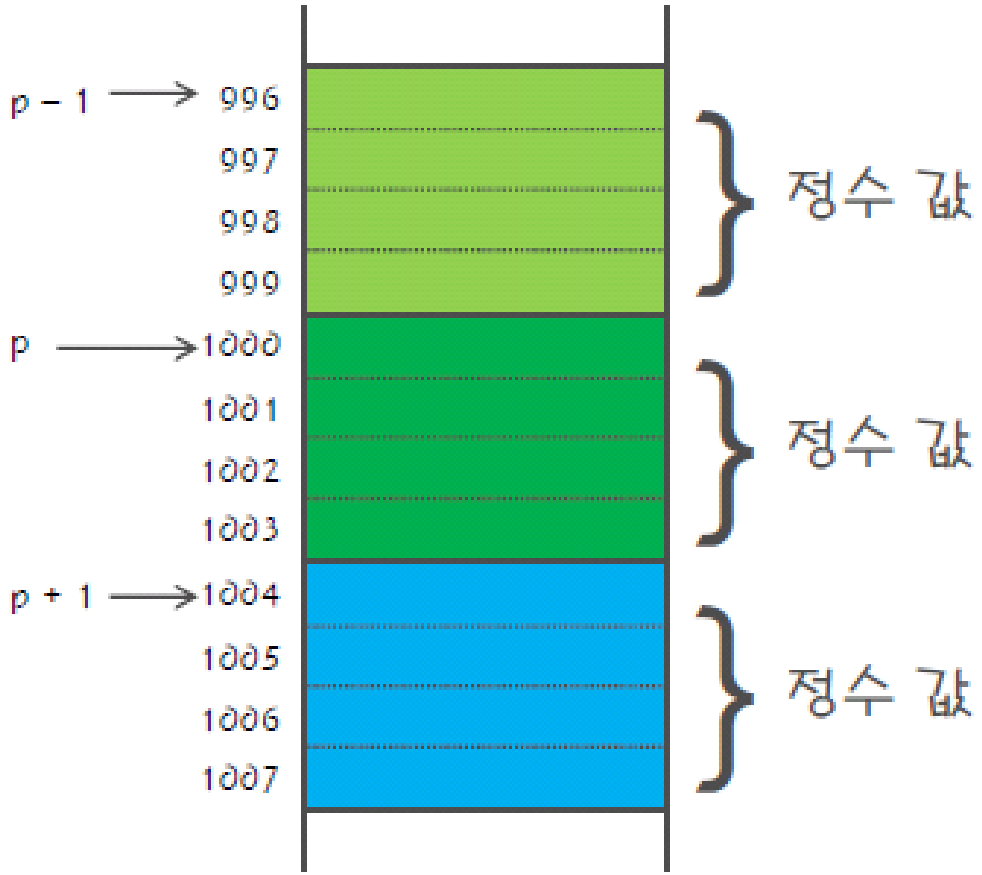
```
   i=100,     j=200,  *p=100,  **q=100
  &i=300,    &j=400,    p=300,   *q=300
 &p=500,     q=500
   i=100,     j=200,  *p=200,  **q=200
  &i=300,    &j=400,    p=400,   *q=400
 &p=500,     q=500
```

포인터 연산

- **p와 q가 포인터 변수라면**
  - p + i, p – i : p가 가리키고 있는 곳으로부터 i번째 앞 또는 뒤 원소
  - p – q : p와 q 사이의 원소 개수

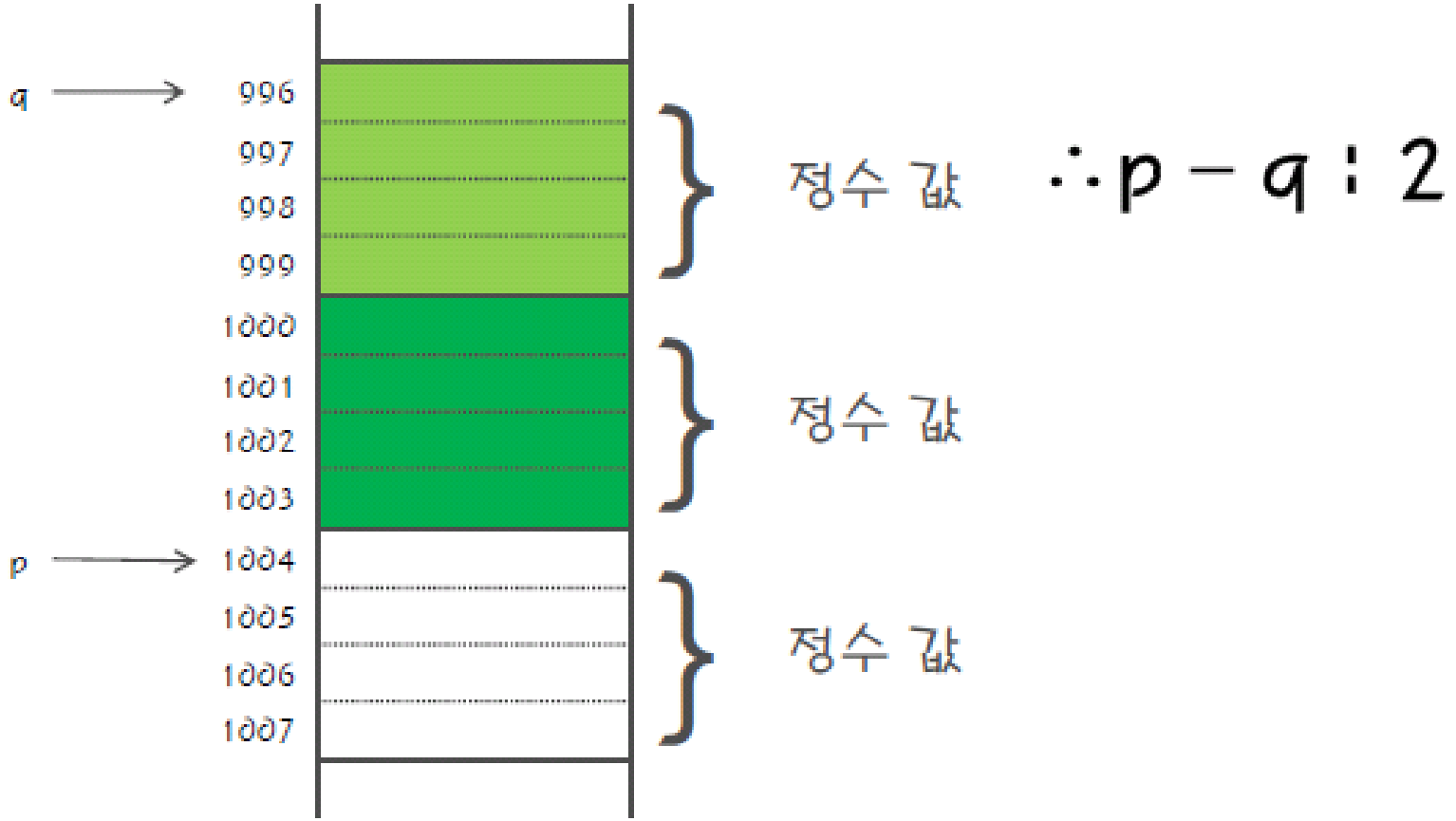
포인터 연산

- p, p + 1, p – 1
  - 가정 : p가 int 형 포인터, 1000번지를 가리킴



포인터 연산

- p – q
  - 가정 : p와 q가 int 형 포인터



예제 프로그램

```c
#include <stdio.h>
int main(void){
    double x[10], *p, *q;

    p = &x[2];
    q = p + 5;
    printf("q - p = %d\n", q - p);
    printf("(int) q - (int) p = %d\n", (int) q - (int) p);
    return 0;
}
```

프로그램 결과

```
q - p = 5

(int) q - (int) p = 40
```

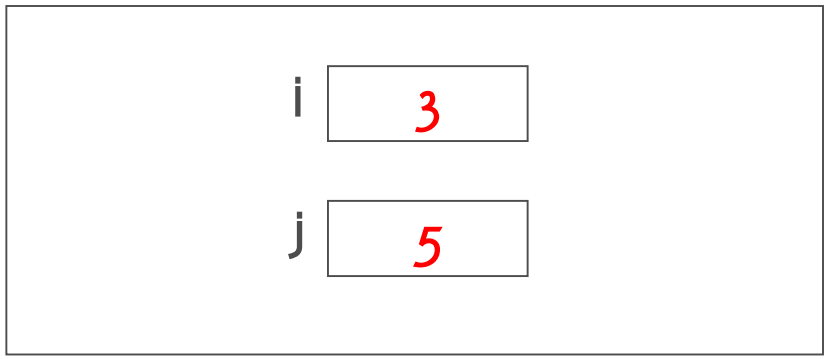
포인터와 함수

- C는 인자 전달 방법으로 "값에 의한 호출" 메커니즘 사용
  - 변수가 함수의 인자로 전달될 때, 변수의 복사본이 전달
  - 호출한 환경의 변수 자체는 변경되지 않음
- C에서 "주소에 의한 호출"을 통해 원본에 접근

값에 의한 호출

```
void swap(int , int);
int  main(void){
    int  i = 3, j = 5;
    swap(i,  j);
    printf("%d %d\n", i, j);
    /* 3 5 is printed */
    return 0;
}
```

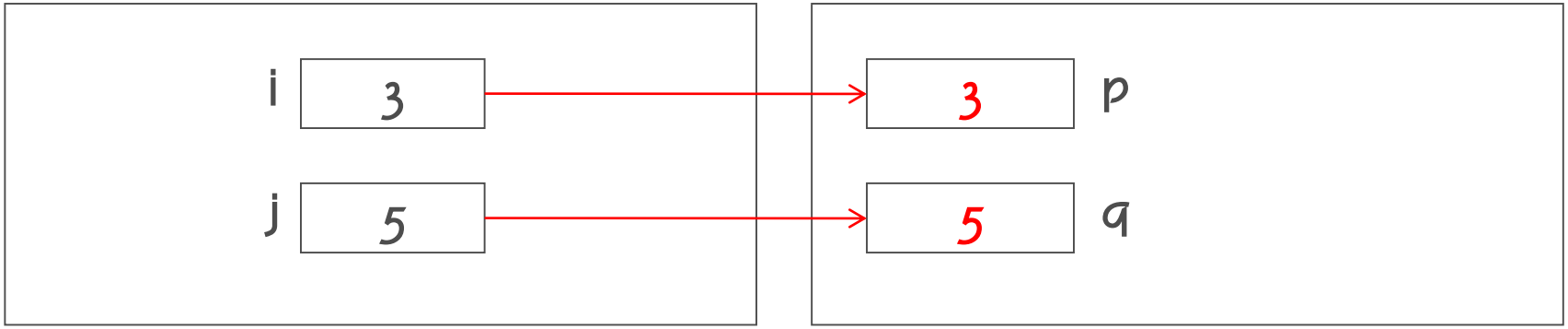
```
void swap(int  p, int  q){
    int  tmp;
    tmp = p;
    p = q;
    q = tmp;
}
```



값에 의한 호출

```
void swap(int , int);
int  main(void){
    int  i = 3, j = 5;
    swap(i,  j);
    printf("%d %d\n", i, j);
    /* 3 5 is printed */
    return 0;
}
```

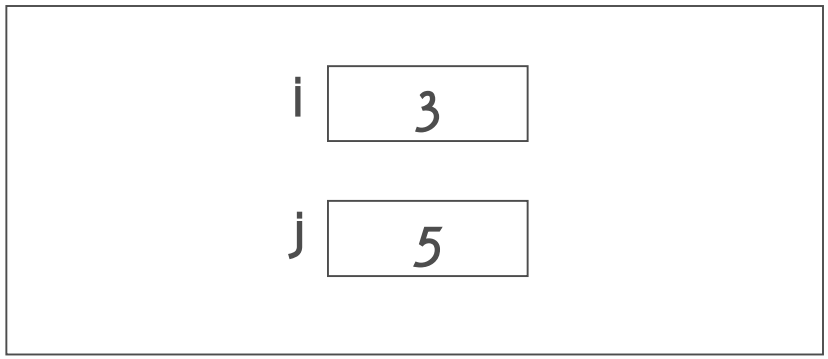
```
void swap(int  p, int  q){
    int  tmp;
    tmp = p;
    p = q;
    q = tmp;
}
```

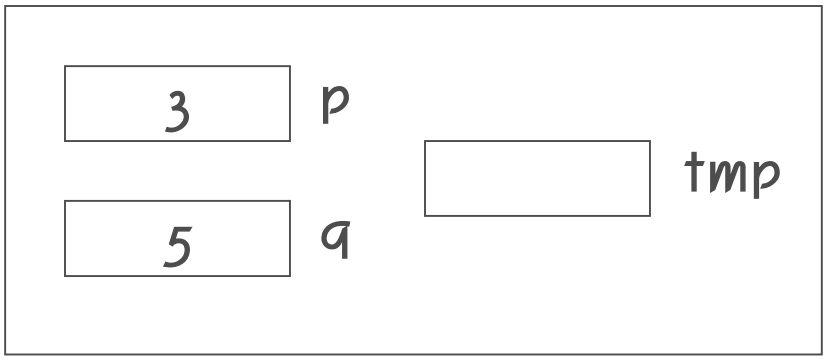


값에 의한 호출

```
void swap(int , int);
int  main(void){
    int  i = 3, j = 5;
    swap(i,  j);
    printf("%d %d\n", i, j);
    /* 3 5 is printed */
    return 0;
}
```

```
void swap(int  p, int  q){
    int  tmp;
    tmp = p;
    p = q;
    q = tmp;
}
```

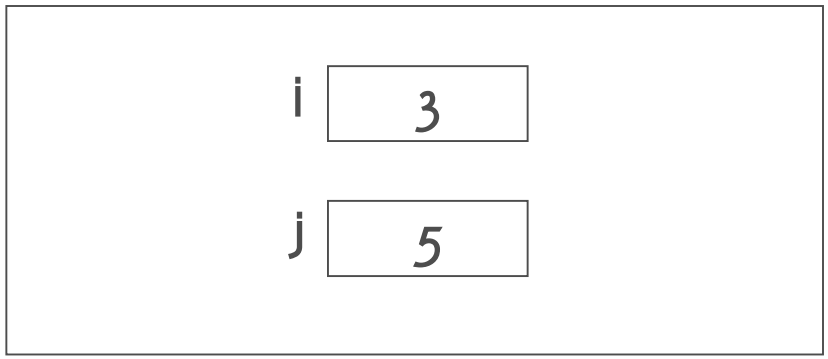


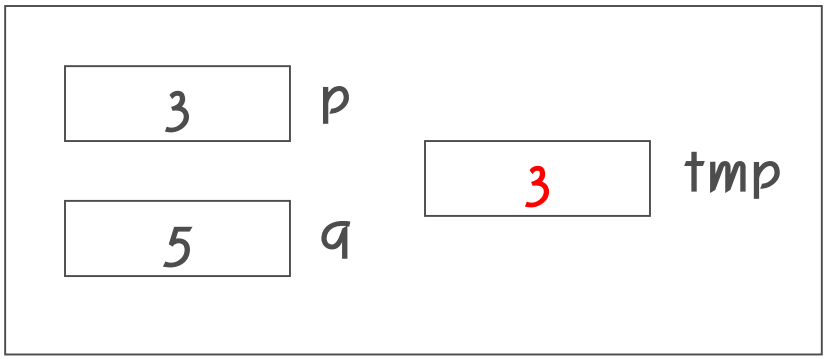


값에 의한 호출

```
void swap(int , int);
int  main(void){
    int  i = 3, j = 5;
    swap(i,  j);
    printf("%d %d\n", i, j);
    /* 3 5 is printed */
    return 0;
}
```

```
void swap(int  p, int  q){
    int  tmp;
    tmp = p;
    p = q;
    q = tmp;
}
```

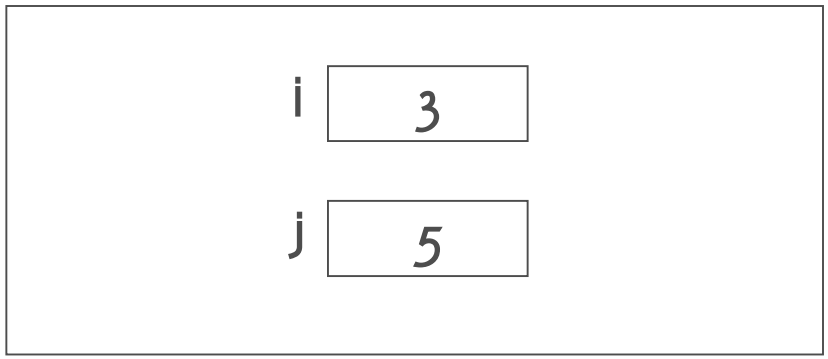


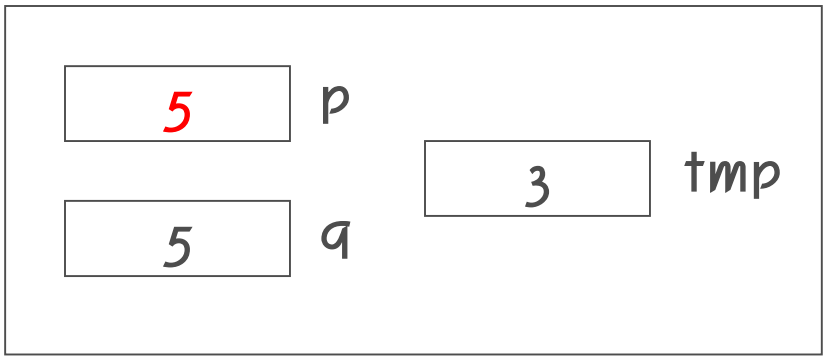


값에 의한 호출

```
void swap(int , int);
int  main(void){
    int  i = 3, j = 5;
    swap(i,  j);
    printf("%d %d\n", i, j);
    /* 3 5 is printed */
    return 0;
}
```

```
void swap(int  p, int  q){
    int  tmp;
    tmp = p;
    p = q;
    q = tmp;
}
```

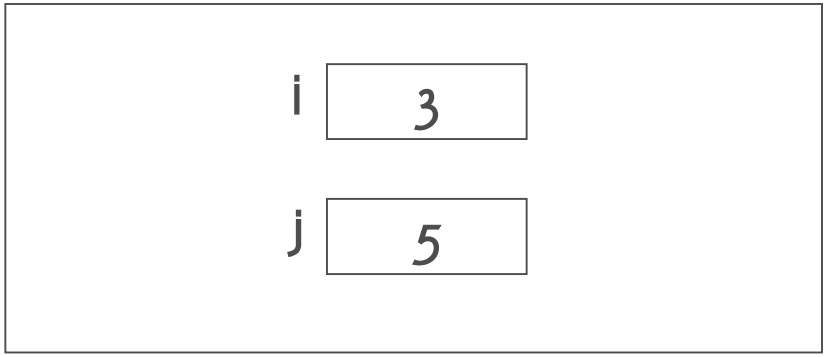


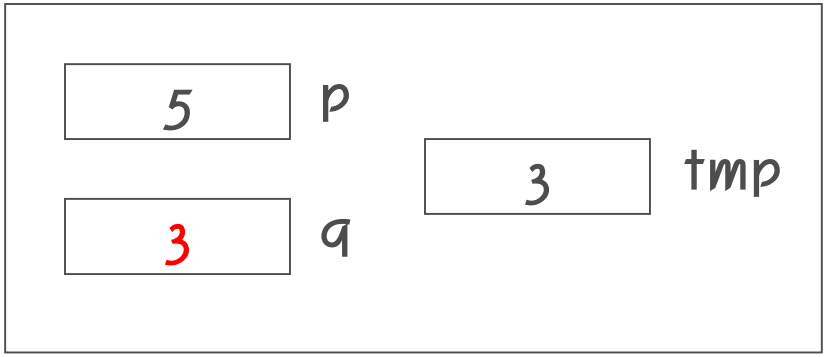


값에 의한 호출

```
void swap(int , int);
int  main(void){
    int  i = 3, j = 5;
    swap(i,  j);
    printf("%d %d\n", i, j);
    /* 3 5 is printed */
    return 0;
}
```

```
void swap(int  p, int  q){
    int  tmp;
    tmp = p;
    p = q;
    q = tmp;
}
```

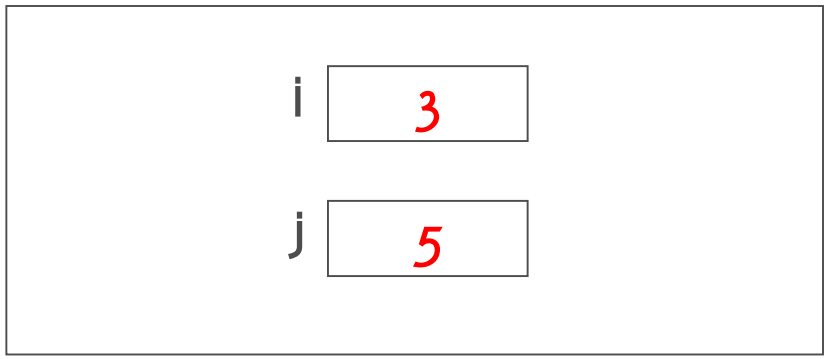




값에 의한 호출

```
void swap(int , int);
int  main(void){
    int  i = 3, j = 5;
    swap(i,  j);
    printf("%d %d\n", i, j);
    /* 3 5 is printed */
    return 0;
}
```

```
void swap(int  p, int  q){
    int  tmp;
    tmp = p;
    p = q;
    q = tmp;
}
```



주소에 의한 호출

1. 함수 매개변수를 포인터 형으로 선언

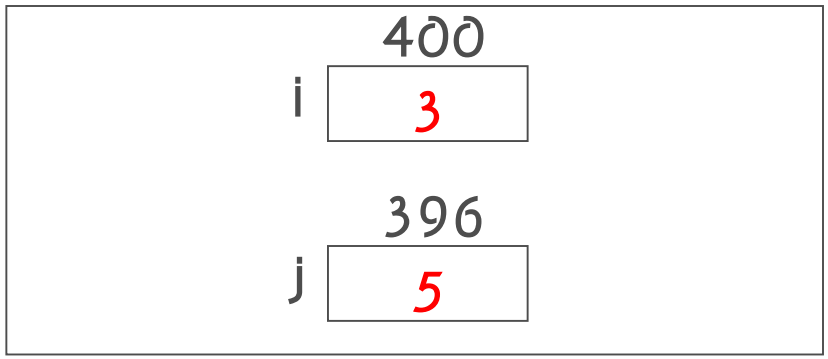
2. 함수 몸체에서 역참조 연산자 사용

3. 함수를 호출할 때 주소를 인자로 전달

주소에 의한 호출

```
void swap(int *, int *);
int main(void){
    int i = 3, j = 5;
    swap(&i, &j);
    printf("%d %d\n", i, j);
    /* 3 5 is printed */
    return 0;
}
```

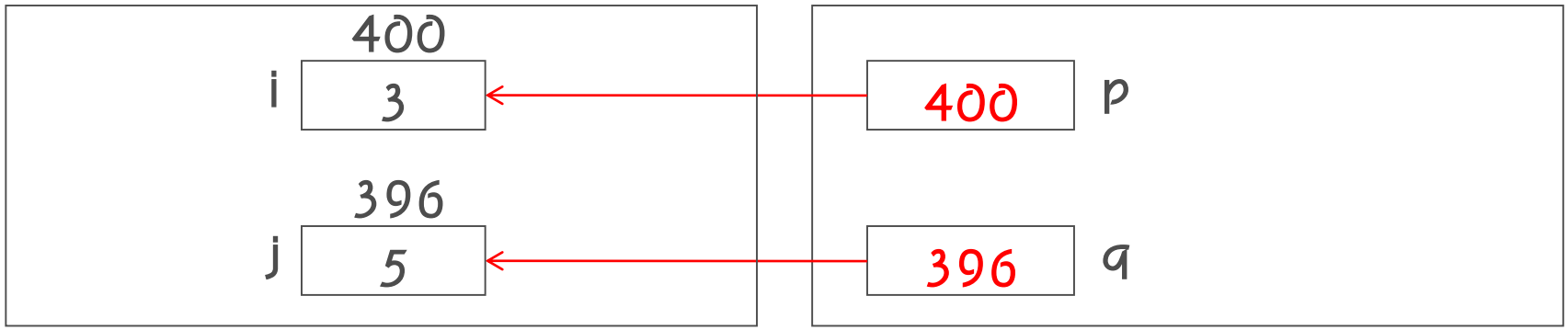
```
void swap(int *p, int *q){
    int tmp;
    tmp = *p;
    *p = *q;
    *q = tmp;
}
```



주소에 의한 호출

```
void swap(int *, int *);
int main(void){
    int i = 3, j = 5;
    swap(&i, &j);
    printf("%d %d\n", i, j);
    /* 3 5 is printed */
    return 0;
}
```

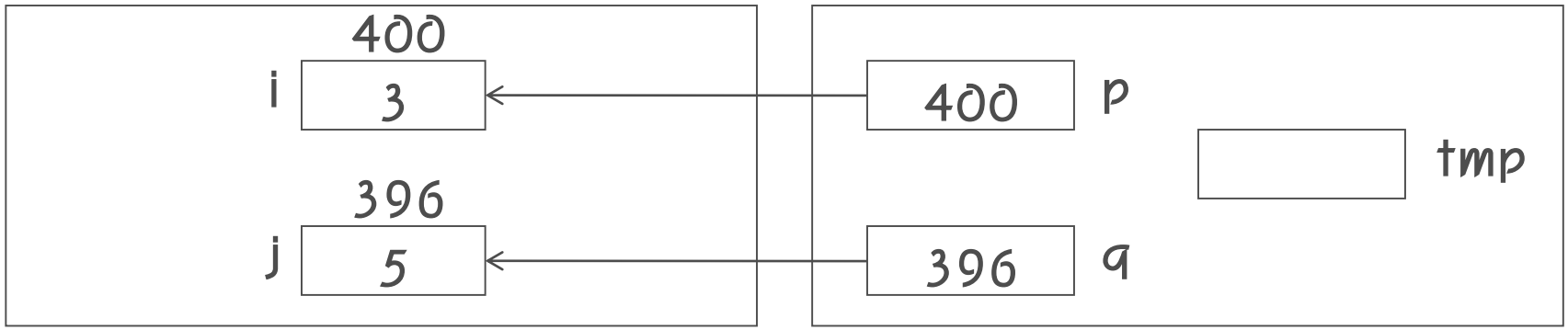
```
void swap(int *p, int *q){
    int tmp;
    tmp = *p;
    *p = *q;
    *q = tmp;
}
```



주소에 의한 호출

```
void swap(int *, int *);
int main(void){
    int i = 3, j = 5;
    swap(&i, &j);
    printf("%d %d\n", i, j);
    /* 3 5 is printed */
    return 0;
}
```

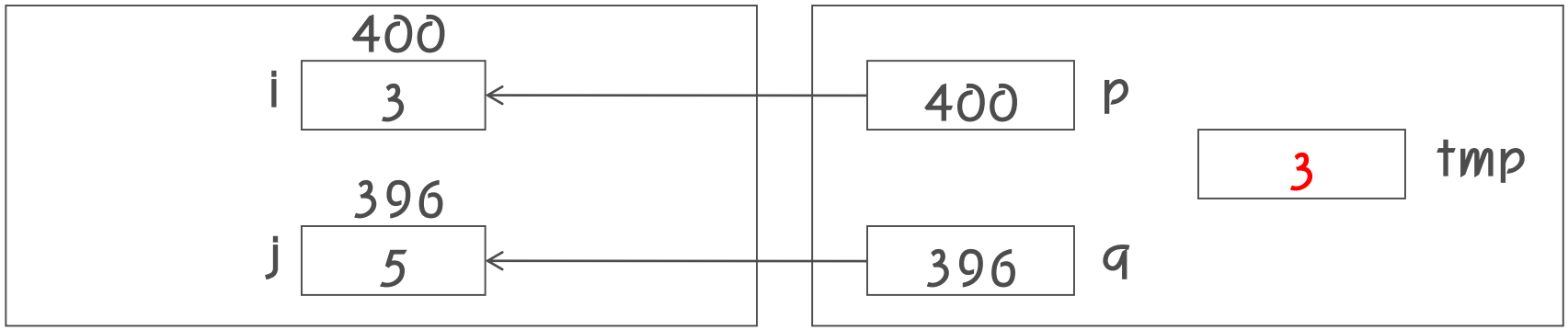
```
void swap(int *p, int *q){
    int tmp;
    tmp = *p;
    *p = *q;
    *q = tmp;
}
```



주소에 의한 호출

```
void swap(int *, int *);
int main(void){
    int i = 3, j = 5;
    swap(&i, &j);
    printf("%d %d\n", i, j);
    /* 3 5 is printed */
    return 0;
}
```

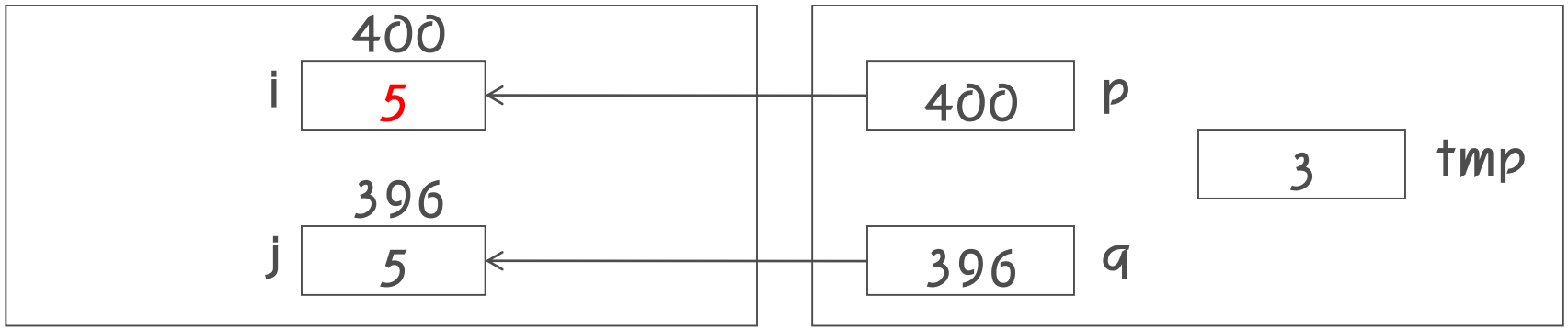
```
void swap(int *p, int *q){
    int tmp;
    tmp = *p;
    *p = *q;
    *q = tmp;
}
```



주소에 의한 호출

```
void swap(int *, int *);
int main(void){
    int i = 3, j = 5;
    swap(&i, &j);
    printf("%d %d\n", i, j);
    /* 3 5 is printed */
    return 0;
}
```

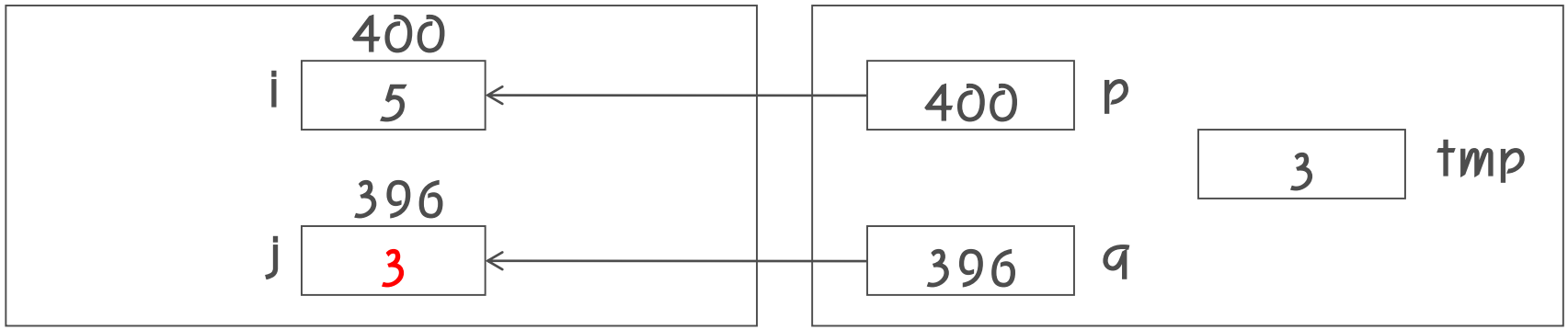
```
void swap(int *p, int *q){
    int tmp;
    tmp = *p;
    *p = *q;
    *q = tmp;
}
```



주소에 의한 호출

```
void swap(int *, int *);
int main(void){
    int i = 3, j = 5;
    swap(&i, &j);
    printf("%d %d\n", i, j);
    /* 3 5 is printed */
    return 0;
}
```

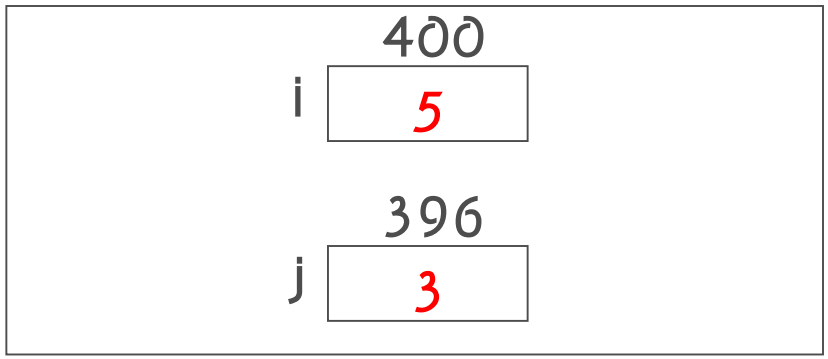
```
void swap(int *p, int *q){
    int tmp;
    tmp = *p;
    *p = *q;
    *q = tmp;
}
```



주소에 의한 호출

```
void swap(int *, int *);
int main(void){
    int i = 3, j = 5;
    swap(&i, &j);
    printf("%d %d\n", i, j);
    /* 5 3 is printed */
    return 0;
}
```

```
void swap(int *p, int *q){
    int tmp;
    tmp = *p;
    *p = *q;
    *q = tmp;
}
```



예제 프로그램

```
int divide_p(int, int, int *, int *);
int main(void){
    int i, j, q, r;
    printf("피제수를 입력하세요 : ");
    scanf("%d", &i);
    printf("제수를 입력하세요 : ");
    scanf("%d", &j);
    if (divide_p(i, j, &q, &r))
        printf("0으로 나눌 수 없습니다.\n");
    else
        printf("%d / %d : 몫은 %d이고 나머지는 %d입니다.\n",
                i, j, q, r);
    return 0;
}
```

예제 프로그램

```
int divide_p(int dividend, int divisor,
                           int * quotient, int * rem)
{
    if (divisor == 0)
        return -1;
    *quotient = dividend / divisor;
    *rem = dividend % divisor;
    return 0;
}
```

포인터와 배열

- 배열 이름은 포인터이고, 그 값은 배열의 첫 번째 원소의 주소값
- 포인터에 배열의 원소를 지정하는 첨자 사용 가능
- 배열 이름은 고정된 주소를 갖는 상수 포인터
  - 예

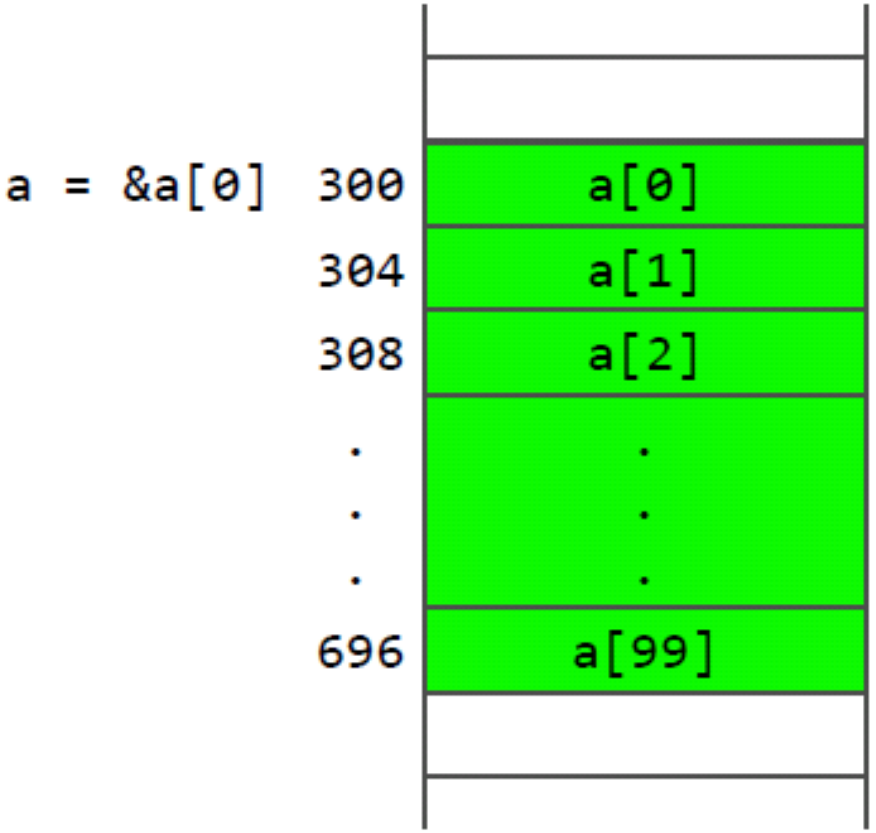
```
int array[3] = {1, 2, 3};
array = array + 1;  // 오류
array++;            // 오류
```

포인터와 배열

```
#define   N   100
int  a[N], i, *p;  // a : int 형 포인터, a[0]의 주소
p = a;             // p = &a[0];
p = a + i;
    // a + i : a[0]에서 i번째 떨어진 원소 위치(주소)
*(a + i) = 10;
p[i] = 10;         // p[i] == *(p + i)
*(p + i) = 10;
a = &i;            // 오류
```

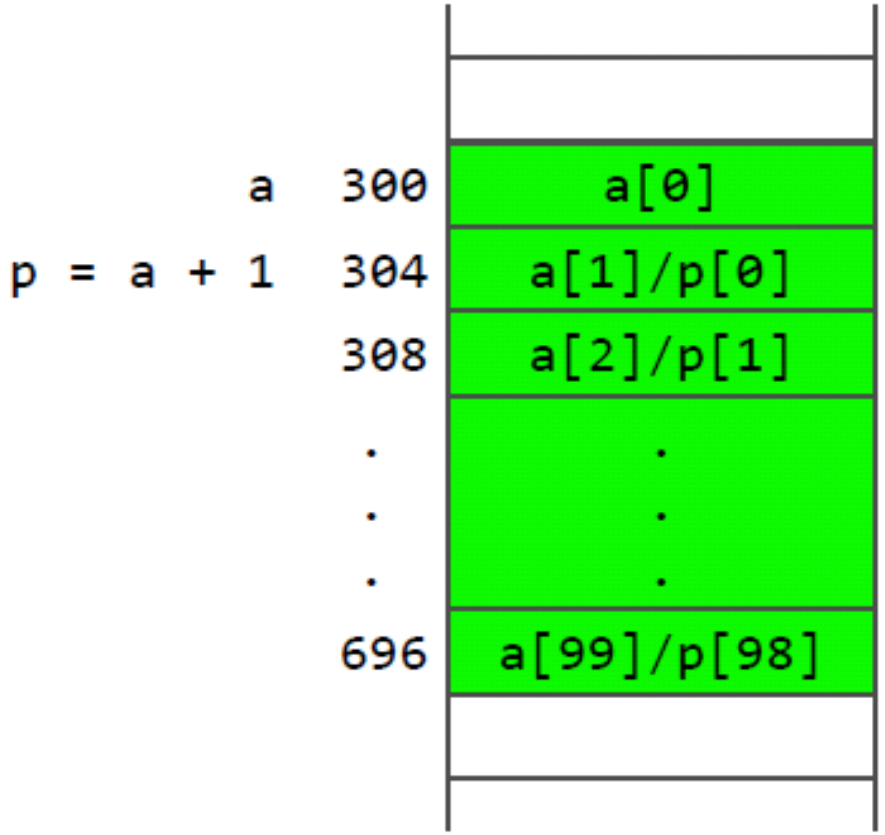
포인터와 배열

```
#define   N   100
int  a[N], i, *p;          // &a[0] : 300 번지
```



포인터와 배열

```
int  a[N], i, *p;          // &a[0] : 300 번지
p = a + 1;                 // p = &a[1];
```



포인터와 배열

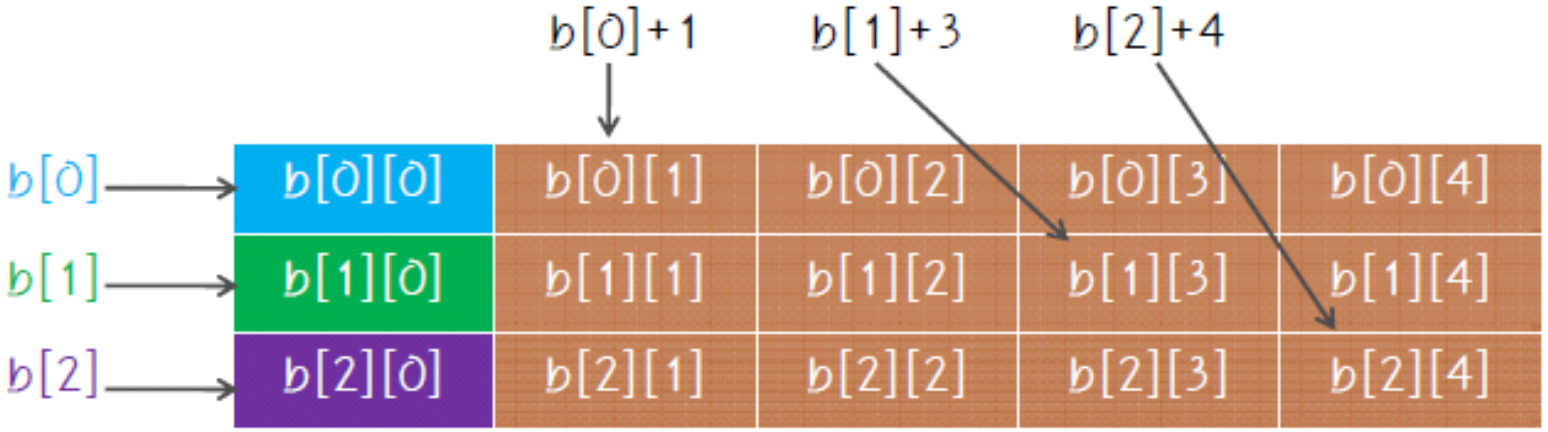
```
int ary[3] = {1, 2, 3};
int *p = ary;

// ary[0], *(ary + 0), p[0], *(p + 0)은 모두 동일
// ary[1], *(ary + 1), p[1], *(p + 1)은 모두 동일
// ary[2], *(ary + 2), p[2], *(p + 2)은 모두 동일
```

포인터와 배열

- 2차원 배열도 인덱스를 하나 제거하면 포인터가 됨

```
int  b[3][5];
```

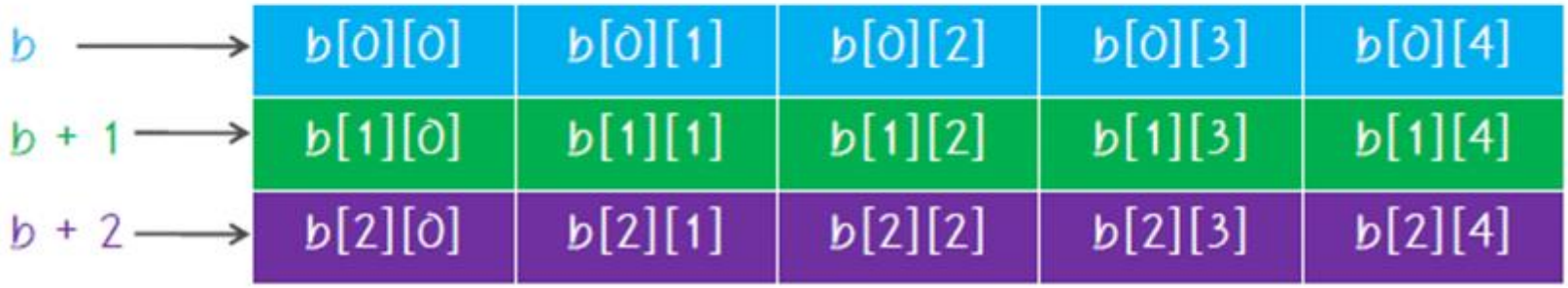
  - b[i] : int 형 포인터, &b[i][0]
  - b[i] + j : &b[i][j]



포인터와 배열

- 2차원 배열에서 인덱스를 두 개 제거하면 배열 포인터가 됨

```
int  b[3][5];
- b : int [5] 형 포인터
- b + i : &b[i][0]부터 int형 원소 5개 포인트
```



포인터와 배열

- 이차원 배열 원소인 **b[i][j]**의 다양한 표기 방법

```
#define   N   100
int  b[N][N];
```

  - b[i][j]
  - *(b[i] + j)
  - *(*(b + i) + j)
  - (*(b + i))[j]

배열과 함수

- 배열을 매개변수로 갖는 함수
  - 배열 매개변수는 포인터임

```
int grade_sum2(int gr[], int size){
    int sum, i;
    for (sum = 0, i = 0; i < size; i++)
        sum += gr[i];
    return sum;
}
```

  - gr : 포인터

배열과 함수

```
int new_grade_sum2(int *gr, int size){
    int sum, i;
    for (sum = 0, i = 0; i < size; i++)
          sum += *(gr + i);
    return sum;
}
```

배열과 함수

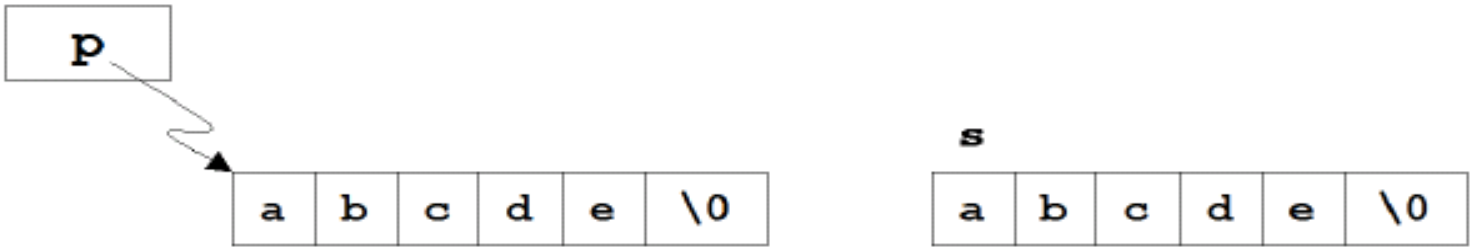
- 배열을 매개변수로 갖는 함수 호출
  - 대응 인자는 주소 값이면 됨

```
#define N 100
int i, a[N], sum;
. . .
sum = grade_sum2(a, N);
      // sum = a[0] + a[1] + ... + a[99]
sum = grade_sum2(&a[5], 10);
      // sum = a[5] + a[6] + ... + a[14]
sum = grade_sum2(&i, 1);
      // sum = i
```

문자열과 포인터

- 문자 배열과 문자열 포인터 차이

```
char *p = "abcde";
char s[] = "abcde";
```

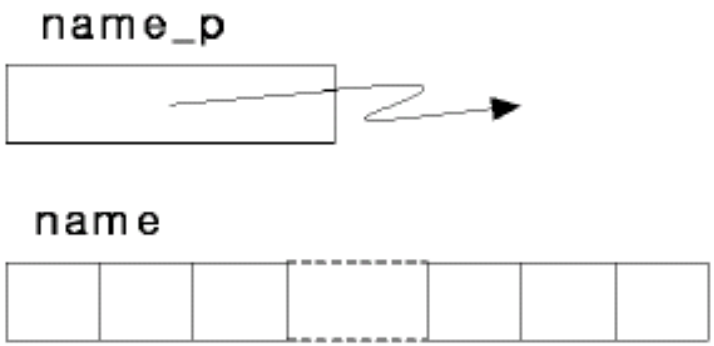



```
printf( "%c %c %c %c" , p[0], *p, s[0], *s); // a a a a
printf( "%s %s" , p, s);          // abcde  abcde
printf( "%s %s" , p+1, s+1); // bcde  bcde
```

문자열과 포인터

- 문자 배열과 문자열 포인터 차이

```
#define  N  20
char name[N] = "";
char *name_p;
scanf("%s", name)
scanf("%s", name_p);          // 오류
```



동적 메모리 할당

- 효율적인 메모리 사용이 가능
- 메모리 할당 함수: calloc(), malloc()

```
<stdlib.h>
void *calloc(size_t N, size_t el_size); // 0으로 초기화
void *malloc(size_t N_bytes);           // 초기화 안함
```

  - 두 함수 모두 할당된 메모리 주소를 리턴
- calloc()과 malloc()을 사용하여 배열, 구조체, 공용체를 위한 공간을 동적으로 생성
- 사용 후 free()를 사용하여 메모리 해제

동적 메모리 할당

```
#include <stdlib.h>
...
int  *grade, N;
...
scanf("%d", &N);
...
grade = (int *)calloc(N, sizeof(int));
// 또는 grade = (int *)malloc(N * sizeof(int));
...
for (i= 0; i< N; i++)
   scanf("%d", &grade[i]);
...
free(grade);
grade = NULL;
```

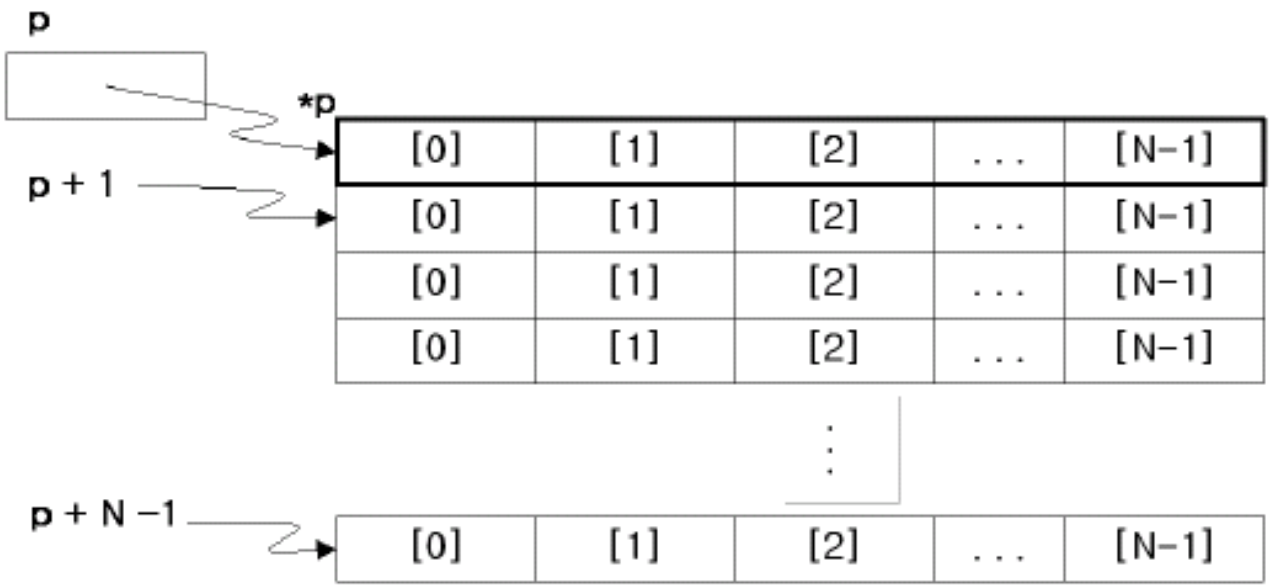
동적 메모리 할당

```
int(*p)[N], *q; // p : 2차원 배열을 포인트하기 위해
...
p = (int (*)[N])calloc(N * N, sizeof(int));
q = (int *) p;
for (i= 0; i< N * N; i++)
   q[i] = i;
for (i= 0; i< N; i++){
   for (j = 0; j < N; j++)
     printf("%3d ", p[i][j]);
   putchar('\n');
}
free(p);
p = NULL;
q = NULL;
```

동적 메모리 할당

- 예제 코드

```
int (*p)[N], *q;          // p : 2차원 배열을 포인트 하기 위해
. . .
p = (int (*)[N])calloc(N * N, sizeof(int));
q = (int *) p;
```



포인터 배열

- 2차원 배열

```
char words[3][9] = { "cat" , "dog" , "elephant" };
```

c	a	t	\0					
d	o	g	\0					
e	l	e	p	h	a	n	t	\0

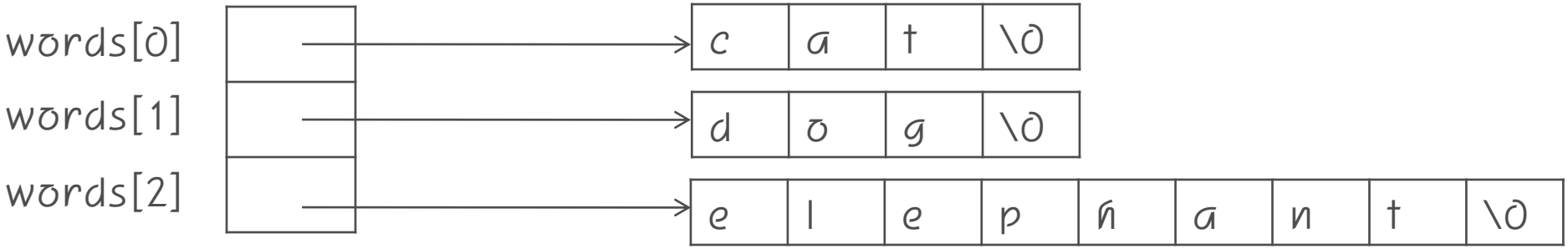
- 사용하지 않는 메모리 공간 낭비

```
printf( "%s %s" , words[0], words[1]);  // cat dog
printf( "%c %c" , words[0][0], words[1][1]); // c o
printf( "%s" , &words[0][1]);  // at
```

포인터 배열

- **포인터 배열**
  - 포인터를 배열원소로 갖는 배열

```
char *words[3] = { "cat" , "dog" , "elephant" };
```




```
printf( "%s %s" , words[0], words[1]);  // cat dog
printf( "%c %c" , *(words[0]), *(words[1]+1)); // c o
printf( "%s" , *(words+1)+1); // og
```

포인터 배열

- 단어 정렬 프로그램

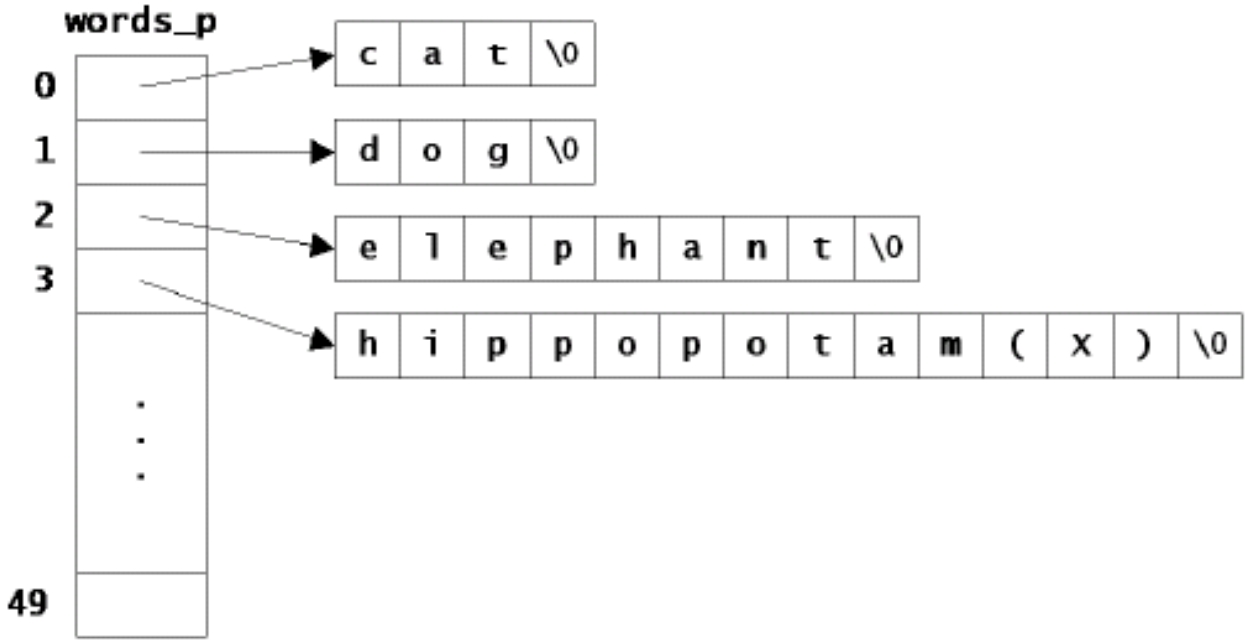
```
char words[N][M];        // N = 50, M = 14
```

words	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]
words[0]	c	a	t	\0										
words[1]	d	o	g	\0										
words[2]	e	l	e	p	h	a	n	t	\0					
words[3]	h	i	p	p	o	p	o	t	a	m	(	X	)	\0
words[4]	s	e	a		h	o	r	s	e	(	X	)	\0	
words[5]	w	h	a	l	e	\0								
.														
.														
words[49]	.	.	.											

* 회색 부분 : 메모리 낭비

포인터 배열

- 다음과 같이 저장하면 메모리를 효율적으로 사용할 수 있음

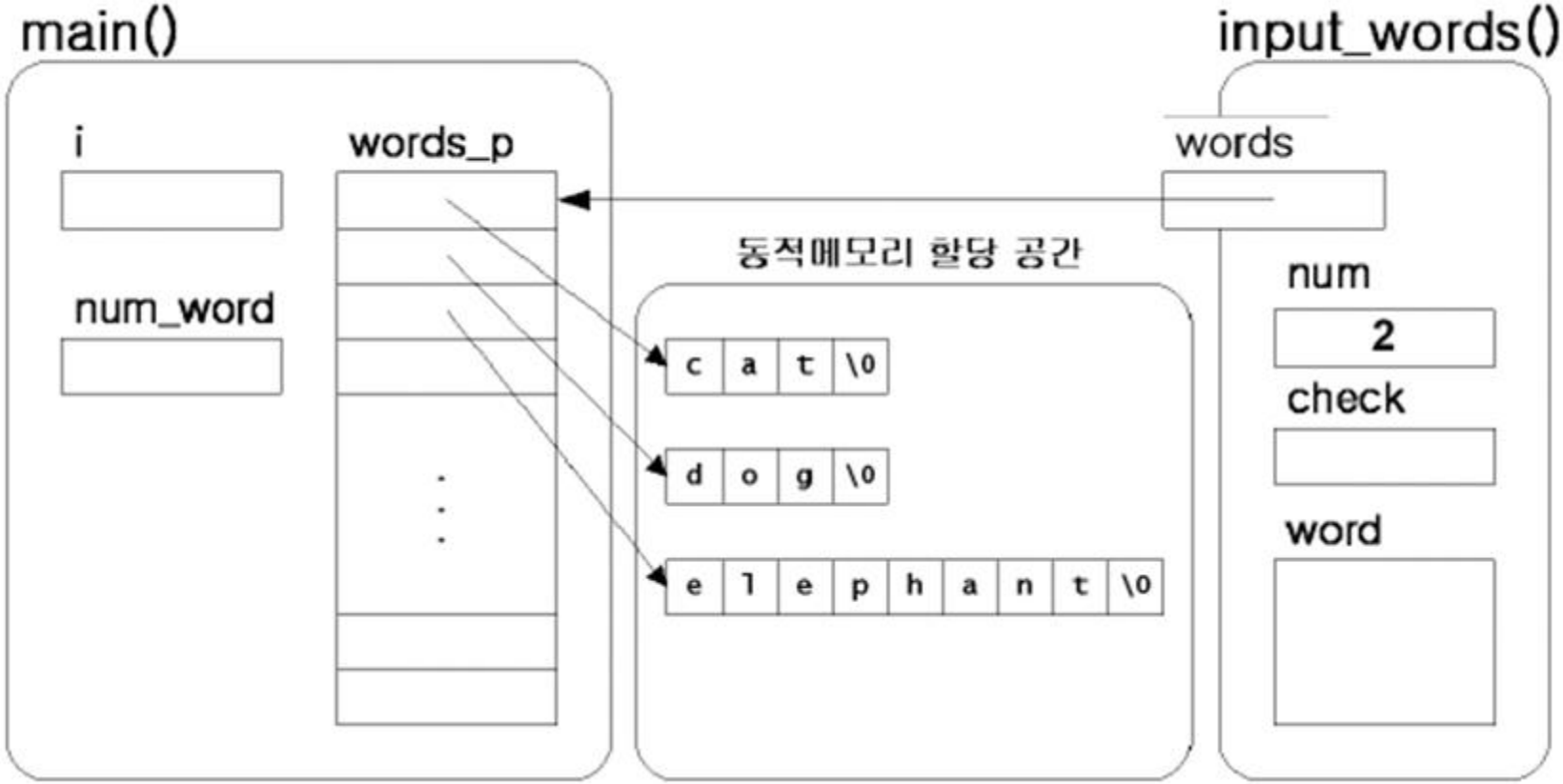



```
char * words_p[N]
```

포인터 배열

```
main(void):
    char *words_p[N];     // char * 형 배열
    input_words(words_p);
    for (i = 0; i < num_word; i++)
        free(words_p[i]);
input_words(char *words[]):
    char word[11];
    input_a_word(word);
    words[num] = (char *)calloc(strlen(word)+1, sizeof(char));
    strcpy(words[num], word);
```

포인터 배열


main()
i
words_p
num_word
input_words()
words
동적메모리 할당 공간
num
2
check
word
c a t \0
d o g \0
e l e p h a n t \0

main ( ) 함수의 인자

- main () 함수는 프로그램 실행 시 명령어 라인으로부터 인자를 전달 받을 수 있음

```
int main(int argc, char *argv[ ])
```

  - argc : 명령어 라인에서 전달된 인자 개수
  - argv : 명령어 라인에서 입력된 문자열들에 대한 포인터

main ( ) 함수의 인자

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int i;
    printf("총 인자 개수 : %d\n", argc);
    for (i = 0; i < argc; ++i)
        printf("%d 번째 인자 : %s\n", i, argv[i]);
    return 0;
}
```

프로그램 결과

```
$ prog hello main
총 인자 개수 : 3
0 번째 인자 : prog
1 번째 인자 : hello
2 번째 인자 : main
```

형 한정자

- 변수의 사용 제한 설정
  - const
  - restrict : C99에서 추가
- 기억영역 클래스 뒤와 형 앞에 지정

const

- const 변수는 초기화될 수는 있지만, 그 후에 배정되거나, 증가, 감소, 또는 수정될 수 없음
- 예

```
const float pi = 3.14;
                 // pi에는 다른 값을 배정할 수 없음
pi = 3.141592;   // 오류 (에러)
```

- 배열 예

```
const char months[][10] =
    {"January", "February", "March", "April",
     "May", "June", "July", "August", "September",
     "October", "November", "December"};
```

const

- const 변수를 포인트 할 때 주의

```
const int   a = 7;
int         *p = &a;    // 오류 (경고)
```

- p는 int를 포인트 하는 보통의 포인터이기 때문에, 나중에 ++*p와 같은 수식을 사용하여 a에 저장되어 있는 값을 변경할 수 있기 때문

const

- const 변수를 포인트 해야 할 경우

```
const  int   a = 7;
const  int  *p = &a;
```

  - 여기서 p 자체는 상수가 아님
  - p에 다른 주소를 배정할 수 있지만, *p에 값을 배정할 수는 없음

const

- 상수 포인터

```
int                 a;
int * const         p = &a;
```

  - p는 int에 대한 상수 포인터임
  - p에 값을 배정할 수는 없지만, *p에는 가능함
  - 즉, ++*p, *p = 10 등과 같은 수식은 가능

const

- const 변수에 대한 상수 포인터 선언

```
const int              a = 7;
const int *const       p = &a;
```

  - p는 const int를 포인트하는 상수 포인터임
  - p와 *p 값은 변경이 안됨

restrict

- C99에서 추가
- 포인터 변수에 적용되며, 현재 포인트 되는 객체는 다른 포인터에 의해서는 포인트 안 됨을 명시하기 위해 사용
- 컴파일러가 코드 최적화를 수행

restrict

```
a = b + c;
i = j - a;
k = l / m;
```

- 컴파일 시 세 번째 문장이 처음 두 문장보다 먼저 실행되거나 동시에 실행되도록 실행코드를 생성할 수 있음

```
*a = *b + *c;
*i = *j - *a;
*k = *l / *m;        // a와 k가 가리키는 곳이 같다면?
```

- 포인터들이 어떤 메모리 공간을 가리키는지 알 수 없으므로 실행코드가 최적화 될 수 없음

함수 포인터

- 하나의 함수를 여러 목적으로 유연하게 사용하고자 할 때 유용
- 유지보수가 용이
- 함수 명 자체가 함수 포인터
  - 선언 방법

    형 (*변수명) (매개변수_목록)

    - 형 : 함수 포인터 변수가 가리키는 함수의 리턴형
    - 변수명 : 함수 포인터 명
    - 매개변수_목록 : 함수 포인터 변수가 가리키는 함수의 매개변수 목록

함수 포인터

- 함수 포인터 선언

```
int (*fp) (int, int); // 올바른 선언
int *fp (int, int);   // 틀린 선언(함수원형)
```

예제 프로그램

```c
#include <stdio.h>
int sum(int, int);
int main()
{
    int (*fp)(int, int);
    int res;
    fp = sum;
    res = fp(10, 20);
    printf( "result : %d\n" , res);
    return 0;
}
```

```c
int sum(int a, int b)
{
    return a+b;
}
```

예제 프로그램

```c
#include <stdio.h>
void func(int (*)(int, int));
int sum(int, int);
int mul(int, int);
int main()
{
   int num;
   printf("1. 두 정수의 합   2. 두 정수의 곱 \n");
   printf("숫자를 입력하세요 : ");
   scanf("%d", &num);
   switch(num){
   case 1: func(sum); break;
   case 2: func(mul); break;
   }
   return 0;
}
```

```c
void func(int (*fp)(int, int))
{
   int a, b;
   int res;
   printf( "두 정수값을 입력하세요 : " );
   scanf( "%d%d" , &a, &b);
   res = fp(a, b);
   printf( "결과 : %d\n" , res);
}

int sum(int a, int b)
{
   return a+b;
}

int mul(int a, int b)
{
   return a*b;
}
```

qsort()

- 예

```
void qsort(void *array, size_t n_els,
        size_t el_size,
        int compare(const void *, const void *));
```

  - qsort()는 다양한 형의 배열을 퀵 정렬로 정렬할 수 있게 함
  - el_size 크기의 원소가 n_els개 있는 array 배열을 정렬함
  - 마지막 매개변수인 compare는 함수 포인터임
  - compare는 const void * 형 매개변수를 두 개 갖고 int 형을 리턴 하는 함수를 포인터 할 수 있음

qsort()

- int 형을 위한 compare 함수 예

```
int compare_int(const void *p, const void *q){
    if (*(int *)p > *(int *)q)
        return 1;
    else if (*(int *)p < *(int *)q)
        return -1;
    return 0;
}
```

- void * 형 포인터인 p와 q를 통해 값을 비교할 때 먼저 (int *)로 캐스트 해야 함

qsort()

- qsort() 사용 예

```
int compare_word(const void *p, const void *q){
    return strcmp(*(char **)p, *(char **)q);
}
. . .
qsort(words_p, num_word, sizeof(char *),
      compare_word);
```

  - compare_word 뒤에 괄호가 없으므로 함수 포인터 임