

C 프로그래밍 #3

배열, 포인터

1차원 배열

■ 비교

```
int grade0, grade1, grade2, grade3, grade4;  
int grade[5];
```

grade0
grade1
grade2
grade3
grade4

grade[0]
grade[1]
grade[2]
grade[3]
grade[4]

배열 선언과 첨자

- 선언 형식

type array_name[size];

- 선언 예

int grade[5];

- 첨자(index) : 문장에서 쓰이는 대괄호 안의 수

- 첨자 사용 예

grade[2] = 10;

- 첨자의 범위

- 배열의 크기가 *size*인 경우 첨자는 0부터 *size* - 1까지

배열의 초기화

- 전통적인 C에서는 외부와 정적 배열만 배열 초기화 이용 가능했으나 ANSI C 부터 자동 배열도 초기화 가능
 - 외부와 정적 배열은 명시적으로 초기화하지 않아도 디폴트로 모두 0으로 초기화 [사용자가 직접 초기화 하는 습관을 가지는 것이 안전]
 - 자동 배열은 반드시 초기화를 시켜야 함

■ 사용 예

```
double    real[5] = {0., 1., 2., 3., 4.};
```

- 초기화 목록의 개수가 배열 원소 개수보다 작은 경우 나머지는 모두 0으로 초기화

```
int    a[100] = {0};
```

- 배열의 크기를 생략한 경우 초기자의 개수가 크기가 된다

```
int a[] = {2, 3, 4, -7};    /* a[]의 크기는 4 */
```

- 문자열의 경우 마지막의 `\0` 유의

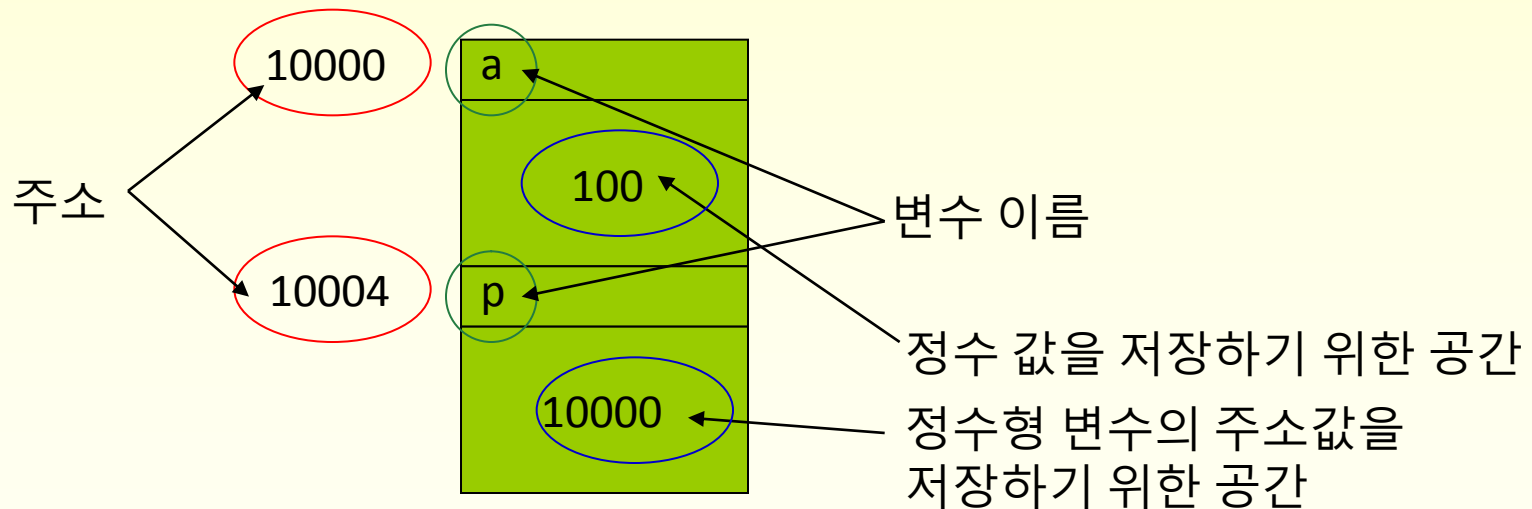
```
char s[] = "abc"; /* 다음과 동일 */
```

```
char s[4] = {'a','b','c','\0'};
```

포인터

- 변수의 3 요소 : 이름, 주소, 공간
- 포인터 : 주소 값을 저장하기 위한 변수

```
int a, *p;  
a = 100;  
p = &a;
```



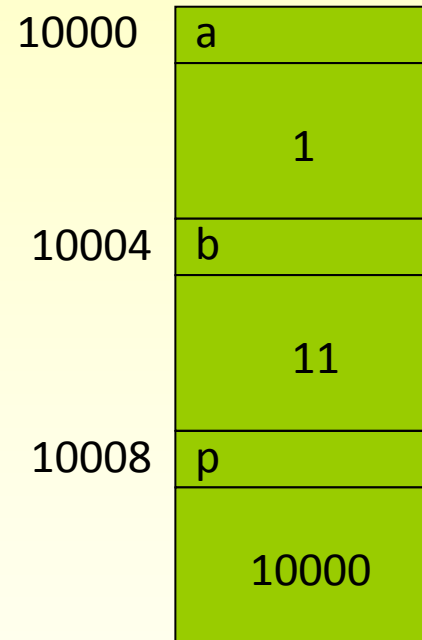
& 연산자와 * 연산자

- & 연산자 : 객체의 주소
- * 연산자 : 저장되어 있는 주소로 객체를 찾아가도록 하는 연산자

```
int a = 1, b = 2, *p;
```

```
p = &a;  
b = *p + 10;
```

[Quiz] 그림에서 &a, &b,
&p, *p, p ?



& 연산자와 * 연산자 예제

```
int      a, b, *p, *q, **r, **s;
a = 100;
p = &a;
q = &b;
*q = 300;

printf("address of a : %p\t value of a : %d\n", &a, a);
printf("address of b : %p\t value of b : %d\n", &b, b);
printf("pointer operator - : p - q = %d \n", p - q);
printf("int operator - : p - q = %d \n", (int)p - (int)q);

r = &p;
s = &q;
**r = 1000;
**s = *p + 400;
printf("address of r : %p\t value of r : %d\n", &r, r);
printf("address of s : %p\t value of s : %d\n", &s, s);
```

10000	a
	1000
10004	b
	1400
10008	p
	10000
10012	q
	10004
10016	r
	10008
10020	s
	10012

주소 값 출력 예

■ %p 형식

```
/* Printing an address, or location. */
#include <stdio.h>
int main(void)
{
    int i = 7, *p = &i;

    printf("%s%d\n%s%p\n",
           "  Value of i: ", *p,
           " Location of i: ", p);
    return 0;
}
```


참조에 의한 호출

■ 값에 의한 호출

- 매개변수에서 변수 이름을 쓰면 변수에 저장된 값이 복사되어 전달
- 호출한 함수의 변수는 변화가 없음
- 매개변수로 쓰이는 변수가 함수의 지역 변수가 되므로 호출된 함수에서 값을 변화시켜도 그 함수 내부에서만 유효

■ 함수를 호출한 함수에서도 값이 변하도록 하기 위해서는 반드시 주소가 전달되어야 함

■ 참조에 의한 호출

- 매개변수로 변수 값이 아닌 주소 값을 넘겨주는 방법
- 둘 이상의 값을 반환(return)할 수 없으므로 주소를 직접 넘겨줌

참조에 의한 호출 예제

- swap()에서 주소를 전달 받았기에 main()안의 i와 j 값이 수정됨
- swap 함수의 정의 부분과 호출 부분의 형태를 주목할 것

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

```
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;
}
```

배열과 포인터의 관계(1)

- 배열의 이름은 **고정된 주소 또는 포인터 값**
- 배열과 포인터 모두 **참자 이용 가능**
- C에서는 배열을 포인터를 이용 구현

```
int a[100], *p;
```

```
a[0] = 100;  
a[10] = 400;
```

```
p = a;  
printf("address of p = %p, value of p = %p, value of *p = %d\n", &p, p, *p);  
printf("address of a[0] = %p, value of a[0]= %d\n", &a[0], a[0]);
```

```
printf("value of *(p+10) = %d, value of a[10] = %d\n", *(p+10), a[10]);  
printf("value of p[10] = %d, value of *(a+10) = %d\n", p[10], *(a+10));
```

배열과 포인터의 관계(2)

```
#define N      100
int          * p, a[N], sum ;
```

Version 1

```
for (i = 0, sum = 0; i < N; ++i)
    sum += a[i];      /* 또는 sum += *(a + i); */
```

Version 2

```
for (p = a, sum = 0; p < &a[N]; ++p)
    sum += *p ;
```

Version 3

```
for (p = a, i = 0, sum = 0; i < N; ++i)
    sum += p[i] ;
```

함수 인자로서의 배열

- 다음 두 선언은 동일

```
double sum(double a[], int n) /* n is the size a[] */  
{
```

.....

```
double sum(double *a, int n) /* n is the size a[] */  
{
```

.....