Summary of Pointers and Arrays

adopted from KNK C Programming : A Modern Approach

내용

- 배열 되짚어 보기
- 포인터 되짚어 보기
- 포인터와 배열
- 배열을 함수의 인자로 전달
- 다차원 배열
- 포인터와 가변 길이 배열

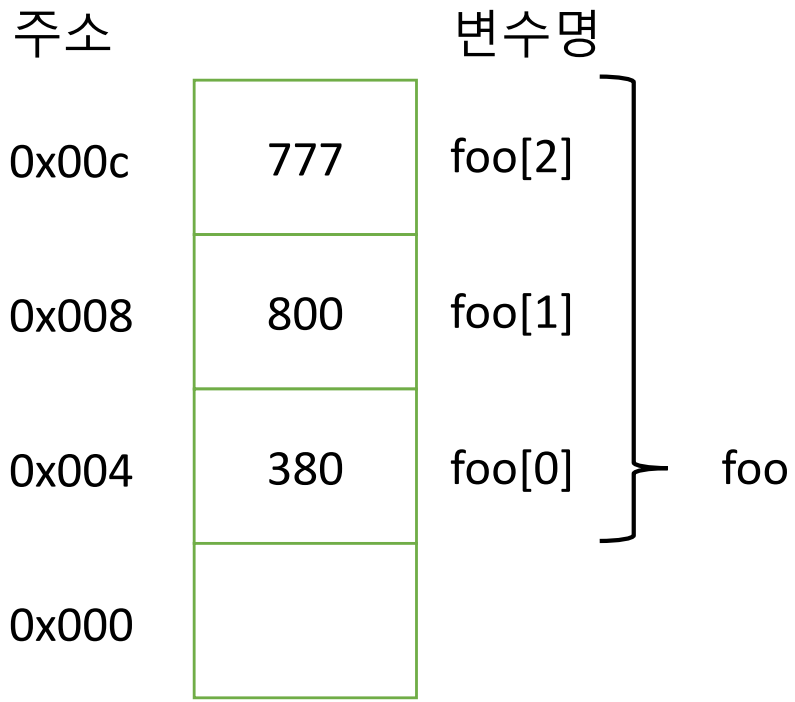
배열 되짚어 보기

- 배열: 같은 형의 값을 여러 개 저장할 수 있는 것과 그것의 시작 주소

배열 되짚어 보기

- 배열: 같은 형의 값을 여러 개 저장할 수 있는 것과 그것의 시작 주소

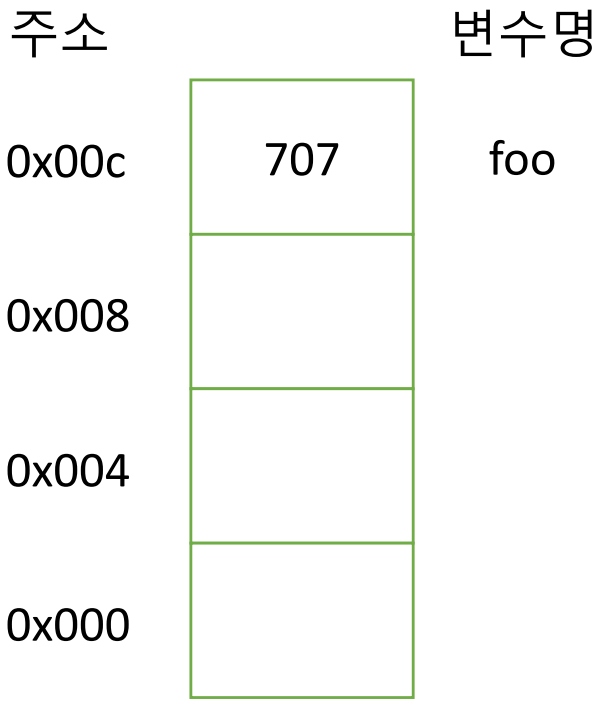
```
int foo[]= {380, 800, 777};
```



포인터 되짚어 보기

- Pointer
  - 가리키는 사람, 지적자, (무언가를)가리키는 막대기
- 포인터 변수
  - 일반적으로 값을 저장하는 변수와 달리 주소를 저장하고 다루는 특수한 변수

```
int foo = 707;
```

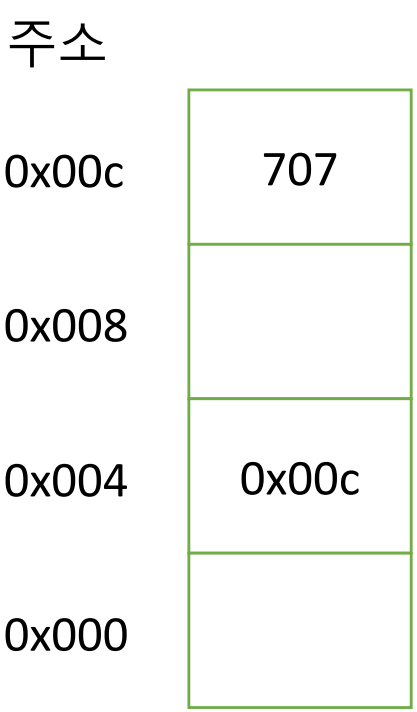


포인터 되짚어 보기

- Pointer
  - 가리키는 사람, 지적자, (무언가를)가리키는 막대기
- 포인터 변수
  - 일반적으로 값을 저장하는 변수와 달리 주소를 저장하고 다루는 특수한 변수

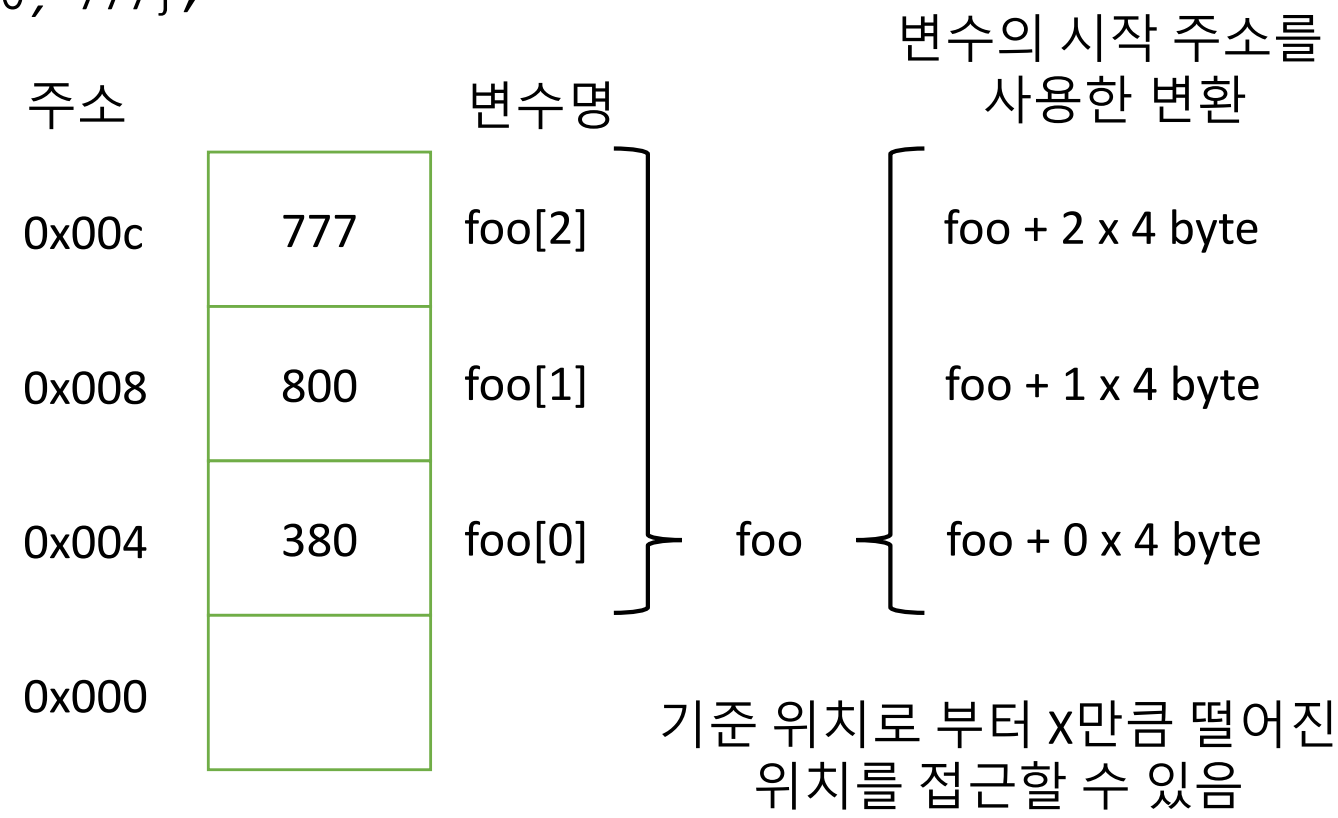
```
int foo = 707;

int *p;
p = &foo;
```



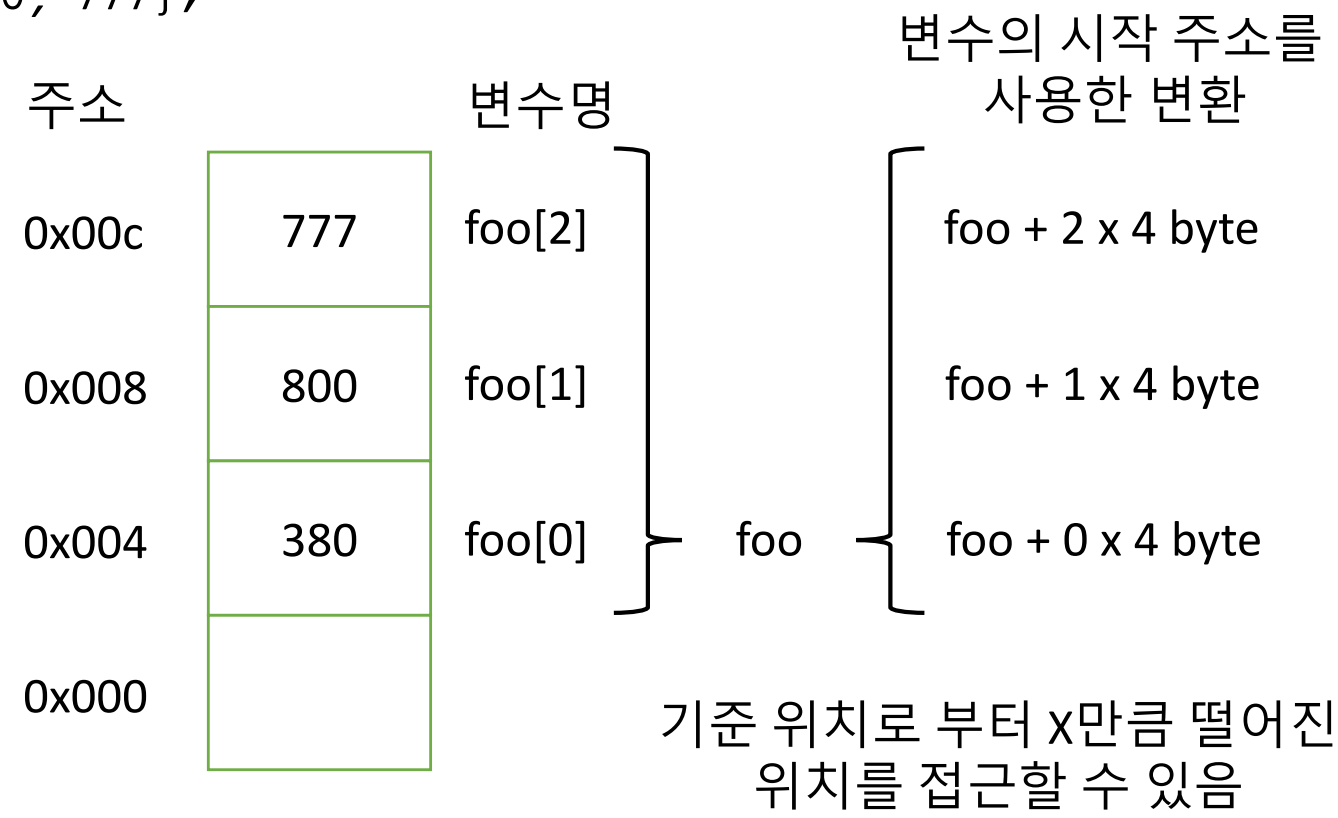
배열: 포인터와의 관계

```
int foo[]= {380, 800, 777};
```



배열: 포인터와의 관계

```
int foo[]= {380, 800, 777};
```

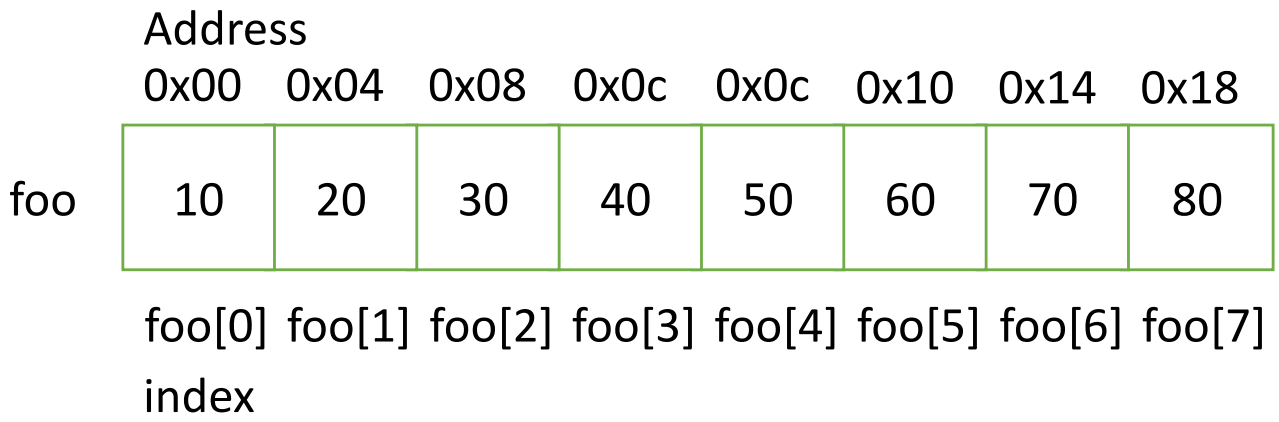


공식

포인터변수명 + 인덱스 x 타입의크기

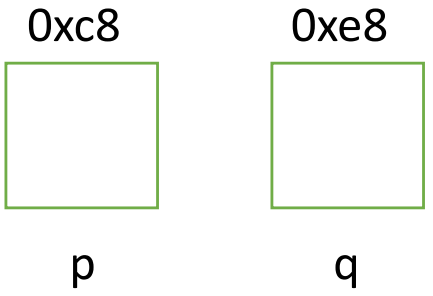
포인터의 덧셈과 뺄셈

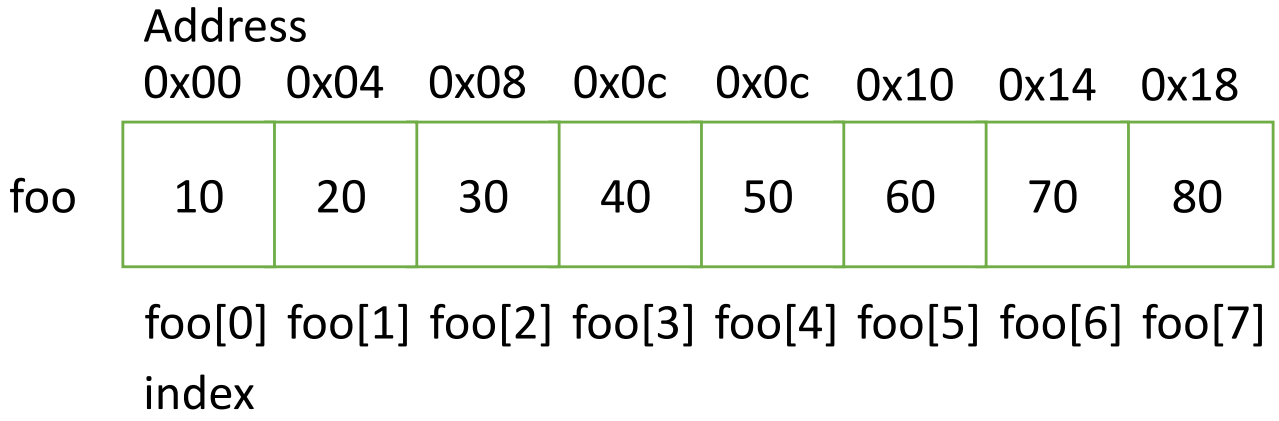
```
int foo[]= {10, 20, 30, 40, 50, 70, 80};
```



포인터의 덧셈과 뺄셈

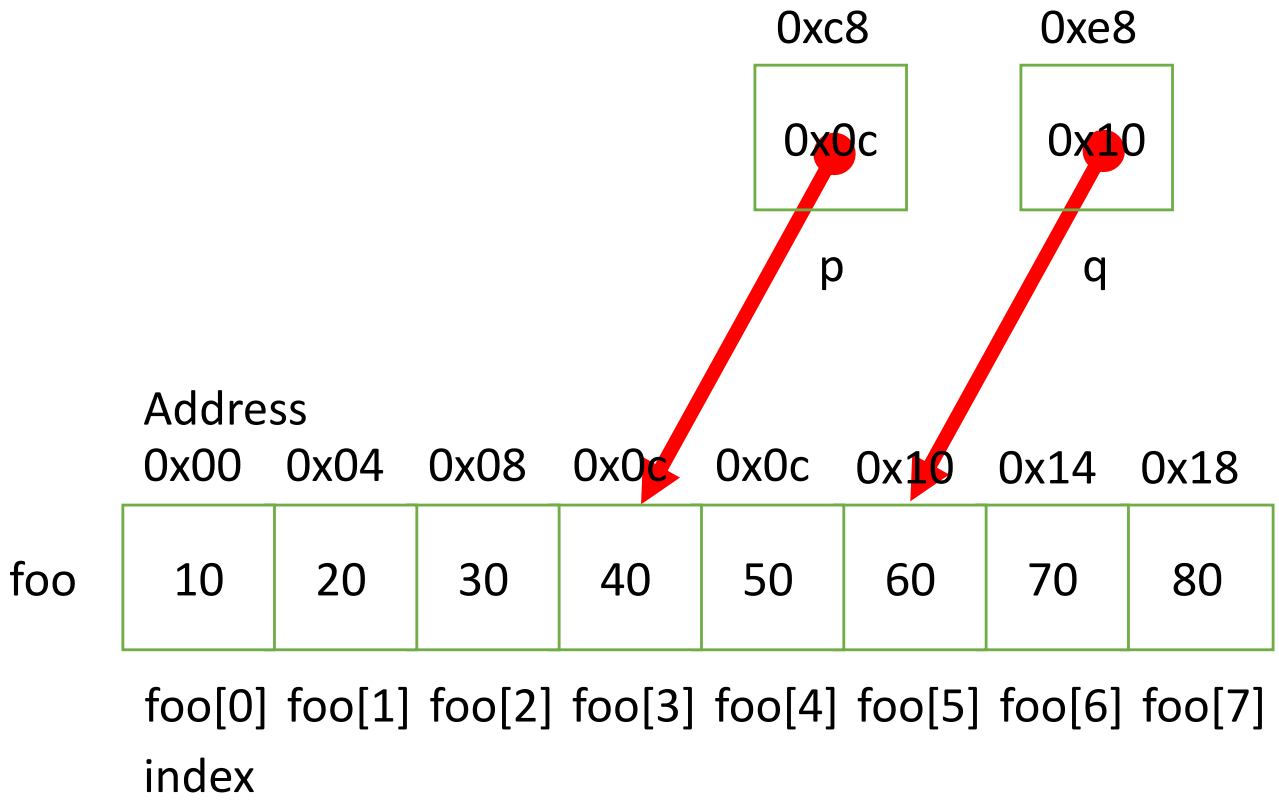
```
int foo[]= {10, 20, 30, 40, 50, 70, 80};
int *p, *q;
```





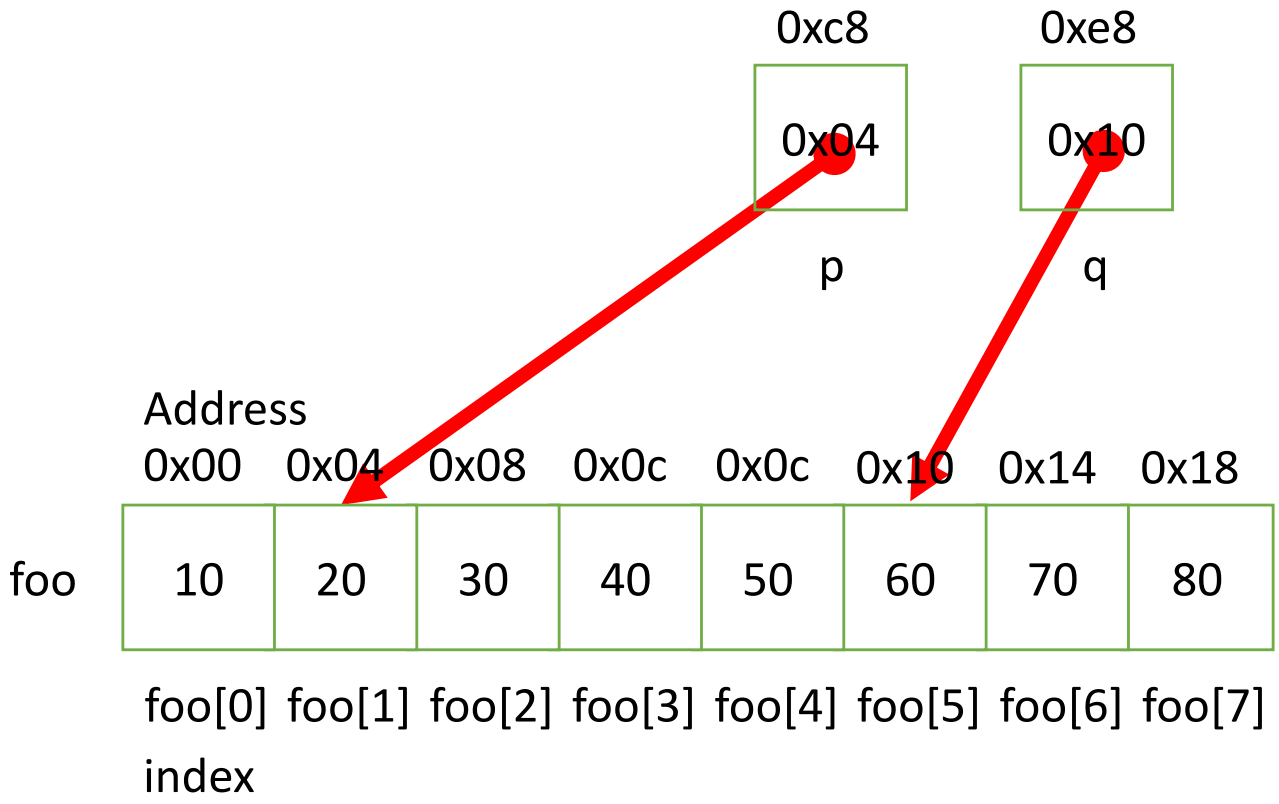
포인터의 덧셈과 뺄셈

```
int foo[]= {10, 20, 30, 40, 50, 70, 80};
int *p, *q;
p = &foo[3];
q = p + 2;
```



포인터의 덧셈과 뺄셈

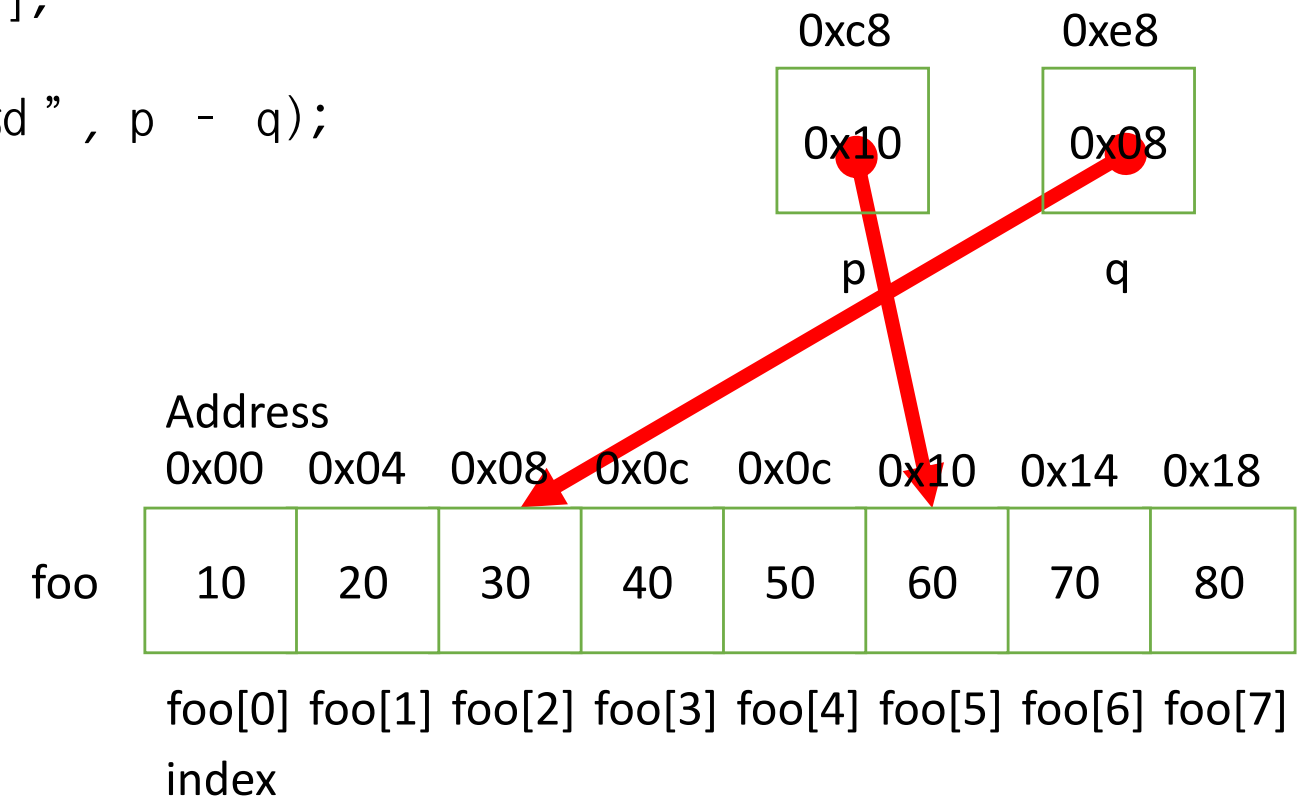
```
int foo[]= {10, 20, 30, 40, 50, 70, 80};
int *p, *q;
p = &foo[3];
q = p + 2;
p -= 2;
```



포인터의 덧셈과 뺄셈

```
int foo[]= {10, 20, 30, 40, 50, 70, 80};
int *p, *q;
p = &foo[5];
q = &foo[2];

printf( " %d " , p  -  q);
```



Example of Array Processing with Pointers

```
#define N 10
…
int a[N], sum, *p;
…
sum = 0;
for (p = &a[0]; p < &a[N]; p++)
    sum += *p;
```

연산자의 결합

- Translate the following sentence to a pointer form

```
p = &foo[i] // p is pointer variable
foo[i++] = j;
```

- 해석:
  - foo 배열의 i 번째 인덱스의 값을 j로 갱신
  - i의 인덱스 값을 1 증가
- 변환:
  1. foo[i]는 p가 가리킴 (p = &foo[i];)
  2. 현재 인덱스에 p를 저장 (*p = j;)
  3. p가 다음 인덱스를 가리키도록 증가 (p = p + 1)
  4. 2와 3을 결합 (*p++ = j, ++가 * 보다 우선순위가 높음)

연산자의 결합 (exercise)

Expression	*Meaning*
`*p++` or `*(p++)`	
`(*p)++`	
`*++p` or `*(++p)`	
`++*p` or `++(*p)`	

배열을 함수의 인자로 전달

배열을 함수의 인자로 전달하기

```
int find_largest(int a[], int n)
{
  int i, max;

  max = a[0];
  for (i = 1; i < n; i++)
    if (a[i] > max)
      max = a[i];
  return max;
}
```

함수의 원형에는 배열이라는 것을 명시
주소라는 정보만으로는 배열인지 알 수 없음
마찬가지로 배열의 길이도 알 수 없음

```
int main(void)
{
   …
   largest = find_largest(b, N);
   …
}
```

배열로 선언된 변수는 주소를 다룸
함수에 인자로 주소 값만 전달하면 됨

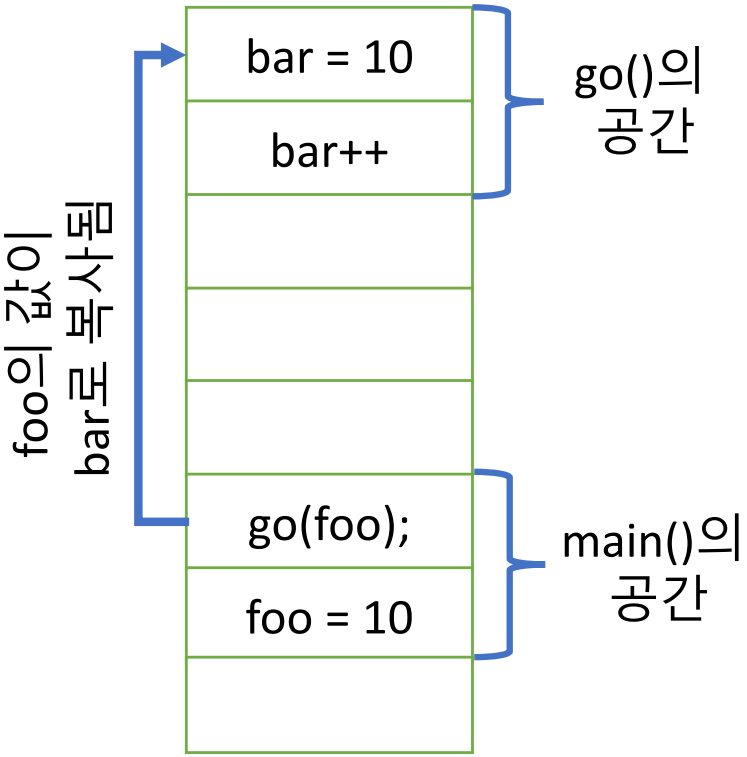
함수 인자로 배열은 포인터처럼 동작

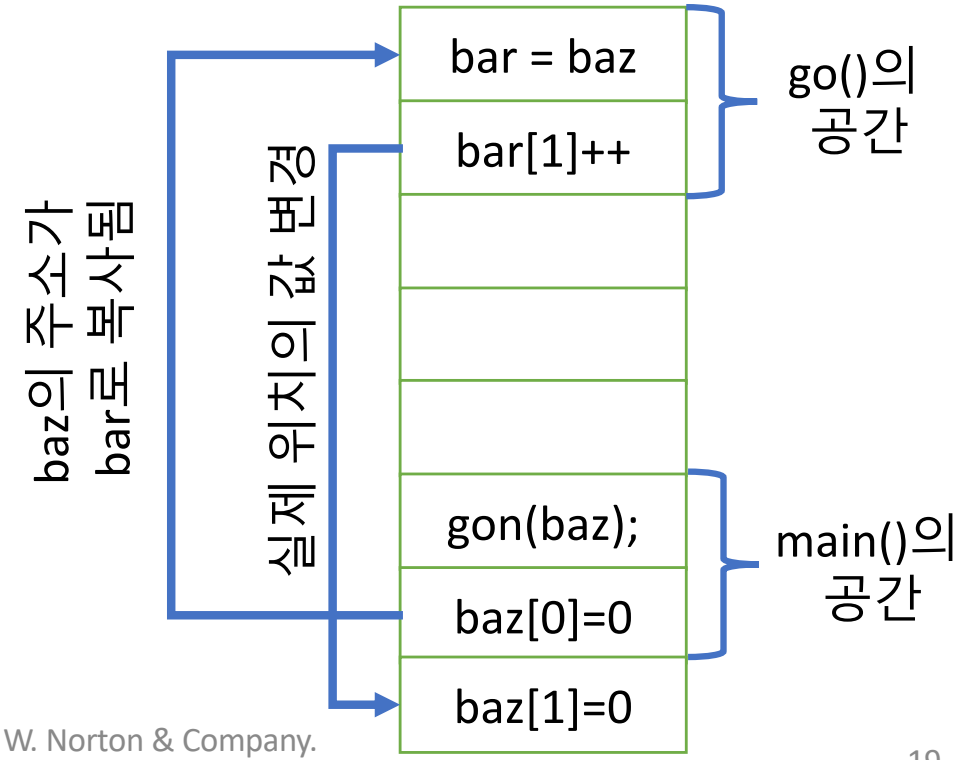
- 의미 1: 배열은 주소이기 때문에 함수가 쓰는 매개변수에 의해 배열의 값이 변경됨

```
int go(int bar){
    bar++;
}
```

```
int gon(int bar[]){
    bar[1]++;
}
```

```
int main(void){
    int foo = 10;
    int baz[2] = {0};
    go(foo);
    gon(baz);
}
```





함수 인자로 배열은 포인터처럼 동작

- 의미 2:
  - 주소만 전달되기 때문에 배열의 크기와 속도는 관계 없음

- 의미 3:
  - 매개변수를 포인터로 선언 할 수 있음

```
int gon(int bar[], int n){
    bar[1]++;
}

int main(void){
    int foo = 10;
    int baz[2] = {0};
    gon(baz, 3);
}
```

```
int gon(int* bar, int n){
    bar[1]++;
}

int main(void){
    int foo = 10;
    int baz[2] = {0};
    gon(baz, 3);
}
```

함수 인자로 배열은 포인터처럼 동작

- 의미 4:
  - 배열의 일부만 전달 할 수 있음

```
int gon(int bar[], int n){
    bar[1]++;
}

int main(void){
    int foo = 10;
    int baz[10] = {0};
    gon(&baz[5], 3);
}
```