



```
35     self.logger = logging.getLogger(__name__)
36     if path:
37         self.file = open(os.path.join(path, 'superuser_seen'))
38         self.file.seek(0)
39         self.fingerprints.update(fp for fp in self.file)
40
41     @classmethod
42     def from_settings(cls, settings):
43         debug = settings.getbool('superuser_seen')
44         return cls(job_dir(settings), debug)
45
46     def request_seen(self, request):
47         fp = self.request_fingerprint(request)
48         if fp in self.fingerprints:
49             return True
50         self.fingerprints.add(fp)
51         if self.file:
52             self.file.write(fp + os.linesep)
```

C

참고 자료

- Warming-up C Programming

Chapter 2. C 프로그램의 기본

변수

- 값을 저장하기 위한 공간
- 변수를 사용하려면 먼저 데이터형(Data Type)과 변수 이름을 정해야 한다.
- C의 데이터형에는 문자형, 정수형, 실수형, 배열, 포인터, 구조체 등이 있다.
 - int와 float형 변수는 메모리 4바이트, char형 변수는 메모리 1바이트를 사용한다.

```
int a; // 정수형 변수 선언
float b; // 실수형 변수 선언
char c; //문자형 변수 선언
```

- 변수 이름은 영문자와 숫자, 밑줄 기호(_)를 이용해서 만들 수 있다.
 - 첫 글자로는 반드시 영문자나 밑줄 기호가 와야 한다.
 - 변수 이름 중간에 빈칸을 사용하거나 다른 기호를 사용해서는 안 된다.

```
int 2019income; // 숫자로 시작하면 안된다.
float tax rate; // 빈칸이 들어가면 안된다.
```

printf 함수 - 서식 지정자(Format Specifier)

- printf 함수는 실수를 출력할 때 디폴트로 소수점 이하 6자리를 출력한다.

```

// %d : 정수를 10진수(0~9)로 출력
int a = 10;
printf("%d", &a); // 실행 결과 : 10

// %x 또는 %X : 정수를 16진수(0~f)로 출력
int a = 10;
printf("%x", &a); // 실행 결과 : a
printf("%X", &a); // 실행 결과 : A

// %f 또는 %F : 실수를 부동소수점 표기 방식으로 출력
float b = 1.23;
printf("%f", &b); // 실행 결과 : 1.230000

// %e 또는 %E : 실수를 지수 표기 방식으로 출력
float b = 1.23;
printf("%e", &b); // 실행 결과 : 1.230000e+00

// %c : 문자 출력
char c = 'A';
printf("%c", &c); // 실행 결과 : A

```

printf 함수 - 문자 폭과 정밀도(Width and Precision)

```

#include <stdio.h>

int main() {
    int x = 987;
    float y = 34.5;

    printf("%d\n", x); // 문자 폭을 지정하지 않으면 왼쪽에서부터 출력한다.
    printf("%6d\n", x); // 6문자 폭에 맞춰 오른쪽으로 정렬해서 출력한다.
    printf("%-6d\n", x); // 6문자 폭에 맞춰 왼쪽으로 정렬해서 출력한다.

    printf("%f\n", y); // 정밀도를 지정하지 않으면 소수점 이하 6자리를 출력한다.
    printf("%.2f\n", y); // 소수점 이하 2자리로 실수를 오른쪽으로 정렬해서 출력한다.
    printf("%6.2f\n", y); // 6문자 폭에 맞춰 소수점 이하 2자리로 실수를 오른쪽으로 정렬해서 출력한다.
}

```

```

// 실행 결과
987
987
34.500000
34.50
34.50

```

scanf 함수

- scanf 함수는 콘솔에서 키보드로 입력한 값을 변수로 읽어온다.
- scanf 함수를 호출할 때는 형식 문자열과 변수 이름을 지정하며, 변수 이름 앞에는 &를 써주어야 한다.
 - 변수 이름 앞에 &를 지정하면 '~에'라는 의미이다.

```

#define _CRT_SECURE_NO_WARNINGS // scanf 함수 사용 시 안정성 관련 컴파일 경고나 에러가 발생하지 않도록 시작 부분에 정

// %d : 정수를 10진수(0~9)로 입력
int num;
scanf("%d", &num);

// %x : 정수를 16진수(0~f)로 입력
int num;
scanf("%x", &num);

// %i : 정수를 10진수, 8진수(012), 16진수(Ox12)로 입력
int num;
scanf("%i", &num);

printf("8진수: %#o\n", num); // '#'은 8진수일 때 앞에 '0'을 붙여준다.
printf("10진수: %d\n", num);
printf("16진수: %#x\n", num); // '#'은 16진수일 때 앞에 '0x'을 붙여준다.

// %f : float형 실수 입력
float x;
scanf("%f", &x);

// %c : 문자 입력
char gender;
scanf("%c", &gender);

```

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main() {
    int age;
    float height, weight;

    printf("나이, 키, 몸무게를 입력하세요: ");
    scanf("%d %f %f", &age, &height, &weight);

    printf("나이: %5d\n", age);
    printf("키: %5.1f\n", height);
    printf("몸무게: %5.1f\n", weight);
}

```

```

// 실행 결과
나이, 키, 몸무게를 입력하세요: 27, 183, 65
나이: 27
키: 183.0
몸무게: 65.0

```

Chapter 3. 데이터형과 변수

기본 데이터형

- 문자형: char
 - char형은 문자를 나타내기 위한 데이터형이지만, 1바이트 크기의 정수형으로 사용할 수 있다.

- 정수형: short, int, long, long long
 - 부호 있는(signed) 정수형과 부호 없는(unsigned) 정수형으로 사용할 수 있다.
- 실수형: float, double, long double

데이터형		크기	유효 범위
signed	char	1바이트	-128 ~ +127
	short	2바이트	-32768 ~ +32767
	int	4바이트	~2147483648 ~ +2147483647
	long	4바이트	~2147483648 ~ +2147483647
	long long	8바이트	-9223372036854775808 ~ +9223372036854775807
unsigned	unsigned char	1바이트	0 ~ 255
	unsigned short	2바이트	0 ~ 65535
	unsigned int	4바이트	0 ~ 4294967295
	unsigned long	4바이트	0 ~ 4294967295
	unsigned long long	8바이트	0 ~ 18446744073709551615

정수의 2진표현

- 부호 있는 정수형은 최상위 비트를 부호 비트로 사용한다.
 - 부호 비트가 1이면 음수, 0이면 양수이다.
 - 컴퓨터 시스템에서는 음수를 나타내기 위해 2의 보수를 사용한다.
 - n을 2의 보수로 표현하려면, 먼저 n을 2진수로 나타낸 다음 각 비트를 0은 1로, 1은 0으로 반전시키고 그 결과에 1을 더한다.

```
// 부호 있는 정수(signed int)와 부호 없는 정수(unsigned int)
#include <stdio.h>

int main(void) {
    short x = -7;
    unsigned short y = 65529;

    printf("x = %5d, %08x\n", x, x);
    printf("y = %5d, %08x\n", y, y);
}
```

```
// 실행 결과
x = -7, ffffff9
y = 65529, 0000fff9
```

- 16진수로 출력된 값의 하위 2바이트만 비교하면, x와 y 둘 다 0xffff9 이므로 -7과 65529의 2진 표현은 같다.

정수형의 유효 범위

```
#include <stdio.h>

int main(void) {
    char n = 128; // char의 유효 범위는 -128 ~ +127로, 현재 n은 유효 범위를 벗어나는 값을 저장했다.
    unsigned char red = 300; // unsigned char의 유효 범위는 0 ~ 255로, 현재 red는 유효 범위를 벗어나는 값을 저장했다.

    printf("n = %d\n", n);
```

```
    printf("red = %d\n", red);
}
```

```
// 실행 결과
n = -128 // 오버플로우 발생
red = 44
```

- 오버플로우: 유효 범위 밖의 값을 저장할 때 유효 범위 내의 값으로 설정되는 것

문자의 2진 표현

```
// 입력된 문자의 ASCII 코드 출력
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void) {
    char ch, prev_ch, next_ch;

    printf("문자? ");
    scanf("%c", &ch);

    prev_ch = ch - 1;
    next_ch = ch + 1;

    printf("prev_ch = %c, %d, %#02x\n", prev_ch, prev_ch, prev_ch);
    printf("ch     = %c, %d, %#02x\n", ch, ch, ch);
    printf("next_ch = %c, %d, %#02x\n", next_ch, next_ch, next_ch);
}
```

```
// 실행 결과
문자? M
prev_ch = L, 76, 0x4c
ch     = M, 77, 0x4d
next_ch = N, 78, 0x4e
```

이스케이프 시퀀스

10진수	8진수	16진수	특수 문자	의미
0	000	00	'\0'	널 문자(null)
7	007	07	'\a'	경고음(bell)
8	010	08	'\b'	백스페이스(backspace)
9	011	09	'\t'	수평 탭(horizontal tab)
10	012	0A	'\n'	줄 바꿈(newline)
11	013	0B	'\v'	수직 탭(vertical tab)
12	014	0C	'\f'	폼 피드(form feed)
13	015	0D	'\r'	캐리지 리턴(carriage return)
34	042	22	'\"'	큰따옴표
39	047	27	'\''	작은따옴표
92	134	5C	'\\'	역슬래시(back slash)

실수형의 유효 범위

```
#include <stdio.h>

int main(void) {
    float num = 3.40282e38; // float형의 최대값 조장
    printf("num = %.15e\n", num);

    num = 3.40282e40; // float형의 오버플로우 발생 → inf로 설정
    printf("num = %.15e\n", num);

    num = 1.17549e-38; // float형의 최소값 저장
    printf("num = %.15e\n", num);

    num = 1.17549e-42; // 가수부를 줄여서 실수 표현
    printf("num = %.15e\n", num);

    num = 1.17549e-46; // float형의 언더플로우 발생 → 0으로 설정
    printf("num = %.15e\n", num);
}
```

```
// 실행 결과
num = 3.402820018375656e+38 // 오버플로우 발생
num = inf
num = 1.175490006797048e-38
num = 1.175689411568522e-42
num = 0.000000000000000e+00 // 언더플로우 발생
```

변수의 선언

- 식별자(Identifier): 변수 이름, 함수 이름처럼 프로그래머가 만들어서 사용하는 이름
 - C언어에서 식별자를 만드는 규칙
 - 반드시 영문자, 숫자, 밑줄 기호(_)만을 사용해야 한다.

- 첫 글자는 반드시 영문자 또는 밑줄 기호(_)로 시작해야 한다.
- 밑줄 기호(_)를 제외한 다른 기호를 사용할 수 없다.
- 대소문자를 구분해서 만들어야 한다.
 - length ≠ Length
- C언어의 키워드는 식별자로 사용할 수 없다.
 - 키워드: C언어에서 특별한 의미로 사용되도록 약속된 단어로, 예약어(Reserved Word)라고도 한다.

```
int usage2019; // [O] 변수 이름의 첫 글자 외에는 숫자를 사용할 수 있다.
double _amount; // [O] 변수 이름은 _로 시작할 수 있다.
double tax_rate; // [O] 여러 단어를 연결할 때는 _를 사용한다.
double taxRate; // [O] 연결되는 단어의 첫 글자를 대문자로 지정한다.

long annual-salary; // [X] 변수 이름에 - 기호를 사용할 수 없다.
int total amount; // [X] 변수 이름에 빈칸을 포함할 수 없다.
int 2019income; // [X] 변수 이름은 숫자로 시작할 수 없다.
char case; // [X] C언어의 키워드는 변수 이름으로 사용할 수 없다.
```

변수의 초기화(Initialization)

- 변수가 메모리에 할당될 때 값을 지정하는 것을 의미하며, 변수를 초기화할 때 변수의 데이터형과 같은 형의 값으로 초기화해야 한다.
- 변수를 초기화하지 않으면 변수는 쓰레기값(의미 없는 값)을 가진다.
- 초기화 되지 않은 변수를 사용하면 컴파일 에러가 발생한다.

```
int amount;
int price = 1000;

printf("amount = %d, price = %d\n", amount, price); // 초기화되지 않은 변수 amount를 사용하고 있으므로 컴파일 에러가 발생

float rate = 0.2 // [X] 0.2는 double형이므로 float 변수를 초기화할 때 컴파일 경고가 발생한다.
int weight = 50.5; // [X] 정수형 변수를 실수값으로 초기화하고 있으므로 컴파일 경고가 발생한다.

int price = 0; // [O] 어떤 값으로 초기화할지 알 수 없으면 0으로 초기화한다.
```

- 대입(Assignment): 변수에 값을 저장하는 것
 - 변수에 값을 저장하려면 대입 연산자(=)의 왼쪽에 변수 이름을 적고, 오른쪽에 값을 적어준다.
 - 변수에 대입을 하면 우변에 있는 값을 좌변에 있는 변수에 저장한다.

상수(Constant)

- 프로그램에서 값이 변경되지 않는 요소
- 상수는 값이 메모리에 저장되지 않고, 한 번만 사용된 다음 없어져 버리는 임시값(Temporary Value)이다.
- 상수에는 값을 직접 사용하는 리터럴 상수(Literal Constant)와 이름이 있는 기호 상수(Symbolic Constant)가 있다.
 - 리터럴 상수: 소스 코드에서 직접 사용되는 값

상수의 종류	구분	예	데이터형
문자형	일반 문자	'x', 'y'	int
	이스케이프 시퀀스	'\b', '\t', '\xa'	int
정수형	10진수 정수	-10, 10	int
	16진수 정수	0xa, 0XA	int
	8진수 정수	012	int
	unsigned형 정수	65536u, 65536U	unsigned int
	long형 정수	1234567l, 1234567L	long
	unsigned long형 정수	1234567ul, 1234567UL	unsigned long
실수형	부동소수점 표기 실수	56.78, .5	double
	지수 표기 실수	5.678e1, .5e0	double
	float형 실수	0.25f, 0.25F	float
문자열	문자열 상수	"apple", "x"	char[]

- 문자형 상수는 작은따옴표(' ') 안에 문자를 적어주거나, '\b' 처럼 역슬래시와 함께 정해진 문자를 적어서 이스케이프 시퀀스로 나타낸다.
 - C에서 문자 상수의 데이터형은 int형이다.
- 정수형 상수는 10진수, 8진수, 16진수로 나타낼 수 있다.
 - 8진수일 때는 012처럼 0을 앞에 붙인다.
 - 16진수일 때는 0xa처럼 0x 또는 0X를 앞에 붙인다.
 - unsigned 정수형 상수일 때는 65536u처럼 u 또는 U를 끝에 붙인다.
 - long 정수형 상수일 때는 2147483647l처럼 l 또는 L을 끝에 붙인다.
 - unsigned long 정수형 상수일 때는 ul 또는 UL을 끝에 붙인다.

```
// 리터럴 상수의 크기
#include <stdio.h>

int main(void) {
    printf("sizeof('x') = %d\n", sizeof('x'));// 작은따옴표를 출력하려면 '\로 표기하며, 문자 상수는 int형으로 간주한다.
    printf("sizeof(0xa) = %d\n", sizeof(0xa));
    printf("sizeof(65536U) = %d\n", sizeof(65536U));
    printf("sizeof(0.25F) = %d\n", sizeof(0.25F));
    printf("sizeof(.5) = %d\n", sizeof(.5));
    printf("sizeof("x") = %d\n", sizeof("x"));// "x"를 저장하는 데 필요한 문자의 개수(널 문자 포함)
}
```

```
// 실행 결과
sizeof('x') = 4
sizeof(0xa) = 4
sizeof(65536U) = 4
sizeof(0.25F) = 4
sizeof(.5) = 8
sizeof("x") = 2
```

기호 상수 - 매크로 상수

- #define문으로 정의되는 상수로, 프로그램에서 여러 번 사용되는 상수 값을 정의해두고 사용하면 편리하다.
- 매크로 이름은 다른 식별자와 쉽게 구별할 수 있도록 모두 대문자로 된 이름을 주로 사용한다.
- 여러 단어로 된 매크로 이름은 밑줄 기호(_)를 이용해서 단어들을 연결한다.

- #으로 시작하는 문장은 전처리기(Preprocessor) 문장이다. → 즉, C문장이 아니다.
 - #define문은 C의 문장이 아니기 때문에 끝에 세미콜론(;)이 필요 없다.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#define HOURLY_WAGE 8350

int main(void) {
    int working_hours = 0;
    int wage = 0;
    // int HOURLY_WAGE = 9000; → 매크로 상수인 HOURLY_WAGE에는 값을 대입할 수 없다.

    printf("working hours? ");
    scanf("%d", &working_hours);
    wage = HOURLY_WAGE * working_hours;

    printf("HOURLY_WAGE : %6d\n", HOURLY_WAGE);
    printf("your wage : %6d\n", wage);
}
```

```
// 실행 결과
working hours? 5
HOURLY_WAGE : 8350
your wage : 41750
```

기호 상수 - const 변수

- 값을 변경할 수 없는 변수로, 변수 선언 시 맨 앞에 const 키워드를 써준다.

```
const int buf_size; // [X] const 변수를 초기화하지 않으면 buf_size에 값을 저장할 수 있는 방법이 없기 때문에, const 변수는 선언
const int buf_size = 256;
buf_size = 128; // [X] const 변수는 변경할 수 없으므로 컴파일 에러가 발생한다.
```

Chapter 4. 연산자

산술 연산자

```
printf("quotient = %d\n", 123 / 50); // 몫을 정수로 구하므로 수식의 값은 2가 된다.
printf("remainder = %d\n", 123 % 50); // 나머지를 구하므로 수식의 값은 23이 된다.
printf("remainder = %f\n", 12.34 % 5); // [X] 실수에 대하여 %를 사용할 수 없다(컴파일 에러).
printf("%d %% %d = %d\n", x, y, x % y); // % 문자를 출력하려면 문자열 내에 %%로 지정해야 한다.
```

증감 연산자

```
int count = 0;
++count;
10++; // [X] 10은 상수이므로 ++ 연산자를 사용할 수 없다(컴파일 에러).
(count + 1)--; // [X] (count + 1)은 수식이므로 -- 연산자를 사용할 수 없다(컴파일 에러).
```

```
// 전위형과 후위형 증감 연산자의 비교
#include <stdio.h>

int main(void) {
    int stock1 = 10, stock2 = 10;
    int current;

    current = --stock1;
    printf("current = %d, stock1 = %d\n", current, stock1); // 전위형 증감 연산자

    current = stock2--;
    printf("current = %d, stock2 = %d\n", current, stock2); // 후위형 증감 연산자
}
```

```
// 실행 결과  
current = 9, stock1 = 9  
current = 10, stock2 = 9
```

관계 연산자

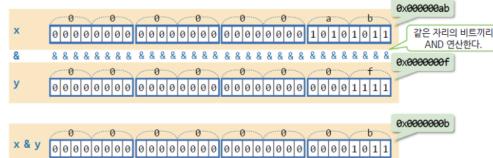
- 두 값이 같은지 비교할 때, `==` 대신 `=`을 사용하면 대입 연산이 수행된다.
 - 두 값이 같은지 비교할 때는 반드시 `==`을 사용해야 한다.

비트 연산자

❖ 비트 AND 연산자(&)

- 각 비트 단위로 AND 연산을 수행

a의 비트	b의 비트	a의 비트 & b의 비트
0	0	0
0	1	0
1	0	0
1	1	1

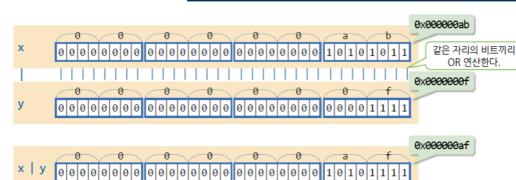


하나라도 0이면 0, 둘 다 1이면 1

❖ 비트 OR 연산자(|)

- 피연산자의 같은 위치에 있는 비트에 대해서 비트 OR 연산을 수행

```
unsigned int x = 0xab;  
unsigned int y = 0xef;  
x | y
```

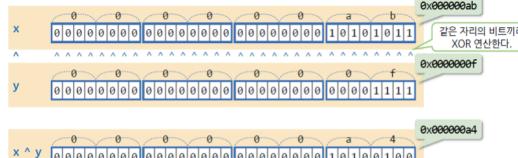


하나라도 1이면 1, 둘 다 0일때만 0

❖ 비트 XOR 연산자(^)

- 피연산자의 같은 위치에 있는 비트에 대해서 비트 XOR 연산을 수행

a의 비트	b의 비트	a의 비트 ⁿ b의 비트
0	0	0
0	1	1
1	0	1
1	1	0

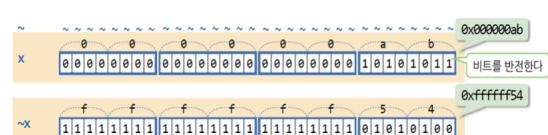


둘 다 같으면 0 서로 다르면 1

❖ 비트 NOT 연산자(~)

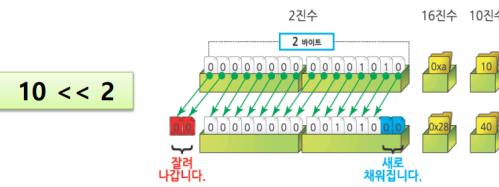
- 피연산자의 각 비트를 반전시킨다. 즉, 0은 1로, 1은 0으로 만든다.

unsigned int x = 0xab;
~x 수식의 값은
0xfffffffffa



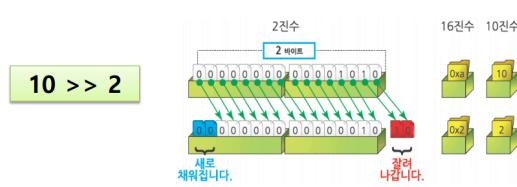
❖ 비트 원쪽 이동 연산자 <<

- 비트들을 원쪽으로 이동(<<)시킴
- 원쪽으로 밀려난 비트는 사라지고, 오른쪽 빈자리는 0이 채워짐
- N비트 원쪽 이동은 2^N 을 곱하는 것과 같음



❖ 비트 오른쪽 이동 연산자 >>

- 비트들을 오른쪽으로 이동시킴
- 오른쪽으로 밀려난 비트는 사라지고, 왼쪽 빈자리를 부호 비트로 채움 (양수는 0으로, 음수는 1로 채움)
- N비트 오른쪽 이동은 2^N 으로 나누는 것과 같음



- 비트 이동 연산자: 밀려진 비트는 사라지고, 새로 채워진 비트는 0으로 채워진다.

- 비트 오른쪽 이동 연산자
 - $z = x >> 2; \rightarrow x / 2^2$ 의 제곱
- 비트 원쪽 이동 연산자
 - $z = x << 2; \rightarrow x * 2^2$ 의 제곱

❖ 비트 이동 연산자의 사용 예

```

01 #include <stdio.h>
02
03 int main(void)
04 {
05     unsigned int x = 0xab;
06     unsigned int z;
07
08     printf("x = %#08x, %d\n", x, x);
09
10     z = x >> 2;           ----- x / 22
11     printf("z = %#08x, %d\n", z, z);
12
13     z = x << 2;           ----- x * 22
14     printf("z = %#08x, %d\n", z, z);
15 }
```

실행 결과

```
x = 0x0000ab, 171
z = 0x00002a, 42
z = 0x0002ac, 684
```

조건 연산자

```
abs = x > 0 ? x : -x; // x > 0이 참일 때 x, 거짓일 때 -x
```

형 변환 연산자 - 명시적인 형 변환

```
int x = 10, y = 11;
double avg;
```

```

avg = (x + y) / 2;
printf("avg = %f\n", avg); // avg = 10.000000

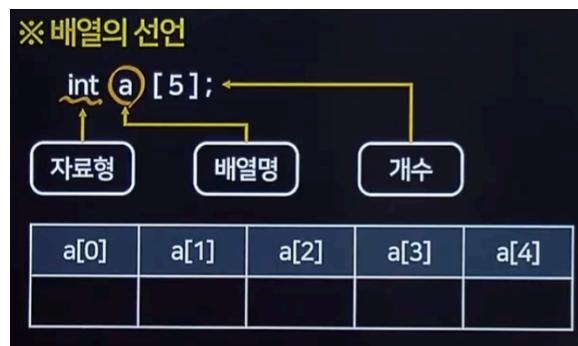
avg (double)(x + y) / 2;
printf("avg = %f\n", avg); // avg = 10.500000

```

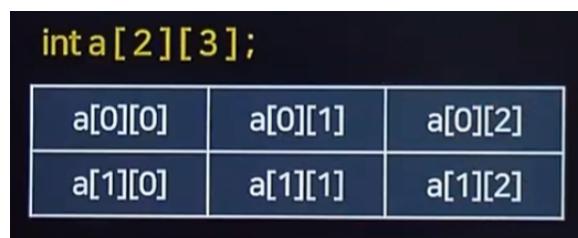
Chapter 7. 배열

배열의 개념

- 배열: 같은 자료형의 변수를 연속적으로 묶어 놓은 저장공간이다.



- 이차원 배열: 같은 자료형의 변수를 행과 열의 연속적인 공간으로 묶어놓은 저장공간이다.



Chapter 8. 포인터

포인터(Pointer) Summary

```
int num = 10; // num이라는 집에 10이라는 값이 살고 있다고 가정한다.
```

모든 집에는 고유한 주소(광주광역시 북구 용봉로 77)가 있다.

포인터는 집 주소 자체를 저장하는 것으로, 포인터 변수에는 10과 같은 실제 값이 아닌 num이라는 변수가 어디에 있는지에 대한 주소를

1. 주소 얻기 연산자 &는 변수 이름 앞에 붙여 '~의 주소'라는 의미로 사용된다.
 - num : num이라는 집의 값 자체, 즉 10을 의미한다.
 - &num : num이라는 집의 주소, 즉 광주광역시 북구 용봉로 77을 의미한다.

2. 포인터 선언 및 역참조 연산자 *

- int *ptr; // '이 변수(ptr)는 주소를 저장하는 포인터입니다.' 라는 의미로, 포인터 변수를 선언할 때 사용한다.
- *ptr = 20; // 'ptr이 가리키는 주소의 집에 20이라는 값을 넣는다.' 라는 의미이다.

```
#include <stdio.h>

int main(void) {
    int num = 10; // num이라는 집에 10이 살고 있다.
    int *ptr; // int형 집 주소를 적을 쪽지(ptr)를 준비한다.
    ptr = &num; // 쪽지(ptr)에 num이라는 집의 주소를 적는다.

    printf("num 변수의 실제 값: %d", num); // 10을 출력한다.
    printf("ptr이 가리키는 곳의 값: %d", *ptr); // 주소로 찾아가서 값을 꺼낸다. → 10을 출력한다.

    printf("num 변수의 메모리 주소: %p", &num); // num의 주소를 출력한다.
    printf("ptr 변수 자체에 저장된 주소: %p", ptr); // ptr이 &num이므로 num의 주소를 출력한다.

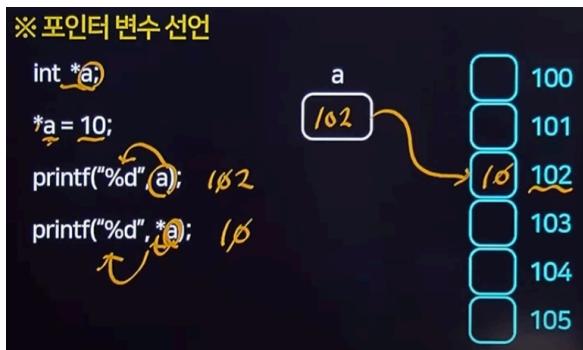
    *ptr = 25; // ptr이 가리키는 주소(num의 집)로 찾아가서 10을 내쫓고 25를 입주시킨다.
    printf("변경된 num 변수의 값: %d", num); // num에 살고 있는 25를 출력한다.

    return 0;
}
```

- *a = b : a의 주소에 b라는 값을 저장한다. → *a는 'a의 주소를 가리킨다.' 라는 의미이다. → **a 주소에 b를 입주시킨다.**
- a = &b : a에 b의 주소를 저장한다. → **b의 주소를 a에 기록한다.**
- *a = &b : a = b

포인터의 개념

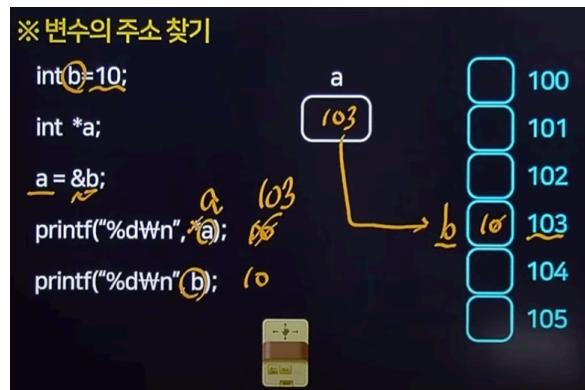
- 포인터: 주소(Address)를 저장하는 변수이다.
 - char형 변수에 문자 코드를 저장하고, int형 변수에 정수를 저장하듯이, 포인터 변수에는 주소를 저장한다.
- 메모리: 연속된 바이트의 모임으로, 각각의 바이트를 구분하기 위해 주소(번지)를 사용한다.
 - 32비트 플랫폼에서는 주소가 32비트(4바이트) 크기이고, 64비트 플랫폼에서는 64비트(8바이트) 크기이다.



```

int *a; // a의 주소(102)를 가져온다. 포인터 변수 a를 생성한다.
*a = 10; // a의 주소값에 10을 저장한다.
printf("%d", a); // a의 주소(102)를 출력한다.
printf("%d", *a); // a의 주소(102)에 저장된 값(10)을 출력한다.

// 주소에 저장된 값 = 주소의 값 = 주소값이라고 생각하자!
  
```



```

int b = 10; // b에 10을 저장한다. -> 랜덤한 주소에 값이 저장된다.
int *a; // 포인터 변수 a를 생성한다.
a = &b; // b의 주소를 a에 저장한다.
printf("%d", a); // a의 값(103)을 출력한다.
printf("%d", *a); // a의 주소에 저장된 값(10)을 출력한다.
printf("%d", b); // b의 값(10)을 출력한다.
  
```

포인터 선언

- 포인터 변수를 선언할 때는 데이터형과 *를 쓴 다음 변수 이름을 적어준다.
 - 포인터 선언 시 사용된 데이터형은 포인터가 가리키는 변수의 데이터형이다.
 - int 포인터가 가리키는 곳에는 int 변수가 있고, double 포인터가 가리키는 곳에는 double 변수가 있다.

포인터의 초기화

- 포인터도 초기화하지 않으면 쓰레기값을 가진다.
- 포인터를 초기화하려면 주소가 필요하다.
- 변수의 주소를 구하려면 주소 구하기 연산자인 &를 이용한다.
- 포인터를 선언할 때 어떤 변수의 주소로 초기화할지 알 수 없으면 NULL로 초기화한다.

```
int a = 10;
int *p = &a; // p를 a의 주소로 초기화한다. → p는 a를 가리킨다.
int *q = NULL; // NULL로 초기화하면, 어떤 변수도 가리키지 않는다는 뜻이다.
int *r = 0; // NULL 대신 0을 사용할 수도 있다.

printf("p = %p\n", p); // 주소를 출력할 때는 %p를 이용한다. → p에 저장된 주소를 출력한다.
```

```
// 실행 결과
p = 00FFF948
q = 00000000
r = 00000000
```

포인터의 사용

```
#include <stdio.h>

int main(void) {
    int a = 10;
    int* p = &a;

    printf(" a = %d", a); // int a = 10; 이므로 a는 10이 출력된다.
    printf("&a = %p", &a); // a의 주소를 출력한다.

    printf(" p = %p", p); // p가 &a를 가리키고 있기 때문에 a의 주소를 출력한다.
    printf("*p = %p", *p); // p가 가리키고 있는 a의 값인 10을 출력한다. → *와 &가 서로 상쇄된다고 생각하자! → *p = &a → p = a
    printf("&p = %p", &p); // p의 주소를 출력한다.

    *p = 20;
    printf("*p = %p", *p);
}
```

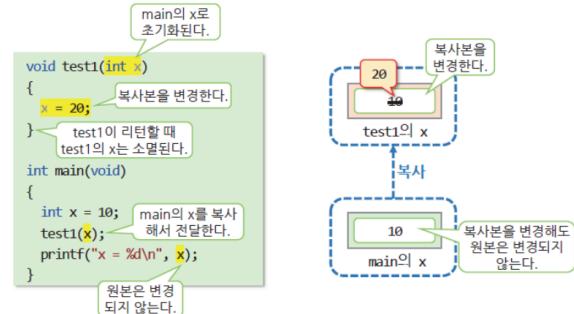
```
// 출력 결과
a = 10
&a = 004FF71C

p = 004FF71C
*p = 10
&p = 004FF710
```

```
*p = 20
```

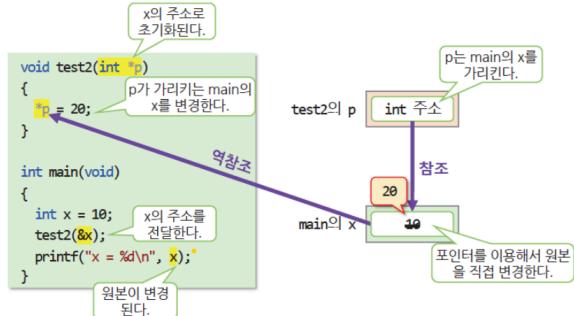
포인터의 용도

```
void test() {  
    x = 20; // [X] 다른 함수에 선언된 지역 변수는 사용할 수 없다.  
}  
  
int main(void) {  
    int x = 10;  
    test();  
}
```



- 위와 같은 상황에서 변수를 직접 사용할 수 없을 때 포인터를 이용해서 주소로 접근할 수 있다.

```
void test2(int *p) { // 2. int *p → x의 주소로 초기화된다.  
    *p = 20; // 3. p가 가리키는 main의 x를 변경한다.  
}  
  
int main(void) {  
    int x = 10;  
    test2(&x); // 1. test 함수에 x의 주소를 전달한다.  
    printf("x = %d", x);  
}
```



const 포인터

- const type *variable; → const가 데이터형 앞에 있으면 포인터가 가리키는 변수의 값을 변경할 수 없다.

```
int a = 10, b = 20;  
const int *p1 = &a; // p1이 a에 읽기 전용으로 접근한다.  
  
printf("*p1 = %d", *p1); // p1이 가리키는 변수 a의 값을 읽어온다.  
  
*p1 = 100; // [X] p1이 가리키는 변수의 값을 const로 지정했기 때문에 변경할 수 없다.  
  
a = 100; // [O] a가 const로 지정된 것이 아니기 때문에 a를 직접 변경할 수는 있다.  
p1 = &b; // [O] p1이 다른 변수 b를 가리킬 수 있다.
```

- type * const variable; → const가 변수 이름 앞에 있으면 포인터 자신의 값(포인터에 저장된 주소)을 변경할 수 없다.

```
int *const p2 = &a; // p2는 a 전용 포인터이다.  
p2 = &b; // [X] p2는 const로 지정되었기 때문에 다른 변수를 가리킬 수 없다.
```

```
#include <stdio.h>
```

```
int main(void) {
```

```

int a = 10, b = 20;

const int *p1 = &a; // p1은 a에 읽기 전용으로 접근한다.
int *const p2 = &a; // p2는 a 전용 포인터이다.
const int * const p3 = &a; // p3는 읽기 전용이면서 a 전용 포인터이다.

printf("*p1 = %d", *p1); // p1이 가리키는 변수 a의 값을 읽어온다. → *p1 = a = 10

// *p1 = 100; // [X] p1이 가리키는 변수의 값을 변경할 수 없다.
p1 = &b; // p1이 다른 변수 b를 가리킬 수는 있다.
printf("*p1 = %d", *p1); // p1이 가리키는 변수 b의 값을 읽어온다. → *p1 = b = 20

*p2 = 100; // p2가 가리키는 변수의 값을 변경할 수 있다.
printf("*p2 = %d", *p2); // *p2 = 100

// *p3 = 100; // p3가 가리키는 변수의 값을 변경할 수 없다.
// p3 = &b; // p3가 다른 변수를 가리킬 수 없다.
printf("*p3 = %d", *p3); // p3가 가리키는 변수 a의 값을 읽어온다. → *p3 = 100
}

```

Chapter 9. 문자열

문자와 문자열

- 문자: 하나의 문자로 구성된다.
 - 'A' → A('A'의 ASCII 코드, 문자, 1바이트)
- 문자열: 연속된 문자들의 모임이다.
 - "A" → A(0(NULL 종료 문자열, 2바이트))
- 문자열 상수(문자열 리터럴): 값이 변경되지 않는 문자열이다.
- 문자열 변수: 문자 배열에 저장되며, 실행 중에 사용자로부터 입력받은 문자열을 저장하거나, 내용이 변경되는 문자열을 저장하려면 문자 배열을 사용해야 한다.
 - 배열의 크기는 '저장할 문자열의 길이 + 1'로 지정한다.
 - char str[10]; → 길이가 9인 문자열을 저장하기 위한 문자 배열
 - char str[10] = "abcdefghijklmnp";
 - 문자 배열의 크기보다 긴 문자열로 초기화하는 경우 맨 끝에 널 문자가 저장되지 않는다.
 - 문자를 작은따옴표로 감싸서 나열해서 문자 배열을 초기화할 수 있지만, 잘 사용되지 않는다.
 - char str[10] = { 'a', 'b', 'c' };

	[0]	[1]	[2]	[3]	[4]	[5]
str	a	b	c	d	e	f

 - 문자 배열 전체를 널 문자로 초기화하려면 널 문자열("")로 초기화한다.
 - char str[10] = "";
 - char str[10] = { 0 }; 과 같이 배열 전체를 널 문자로 초기화한다.
 - 문자 배열을 어떤 값으로 초기화할지 알 수 없으면 널 문자로 초기화하는 것이 안전하다.
 - 문자 출력 시 %c를 사용하며, 문자열 출력 시 %s를 사용한다.

```

#include <stdio.h>

int main(void) {
    char str1[10] = "abc"; // 문자 배열은 문자열 리터럴로 초기화가 된다.
    char str2[10] = "very long string"; // 문자 배열보다 긴 문자열로 초기화가 된다.
    char str3[] = "abc"; // 크기가 4('a', 'b', 'c', '\0')인 배열로 할당된다.
    char str4[10] = ""; // 배열 전체가 널 문자로 초기화된다.

    str1[0] = 'A'; // 인덱스를 이용해 한 문자씩 변경할 수 있다.
    printf("str1 = ");
    for (int i = 0; str1[i] != '\0'; i++) { // 널 문자를 만날 때까지 str[i]를 한 문자씩 출력한다.
        printf("%c", str1[i]);
    }
    printf("\n");

    /*
    str2는 최대 10개의 문자만 담을 수 있는 공간이다.
    "very long string"은 공백 포함 16자이고, 문자열의 끝을 알리는 \0까지 포함하면 총 17바이트의 데이터이다.
    10칸짜리 상자에 17개의 물건을 넣으려는 것과 같이, 컴파일러가 경고를 보내지만 일단 담을 수 있는 만큼만 담는다.
    따라서 str2 배열에는 "very long "까지의 10글자만 채워져 들어가고, 문자열의 끝을 알리는 \0은 들어갈 공간이 없어 잘려나간다
    여기서 아래 str2의 출력문의 동작 방식은 아래와 같다.

    - 주어진 주소(str2)에서부터 시작해서, \0(널 문자)를 만날 때까지 메모리에 있는 모든 문자를 계속해서 출력하시오.
    printf 함수는 str2의 배열의 크기가 10이라는 사실을 전혀 모르기 때문에 \0을 만날 때까지 멈추지 않는다.
    printf는 str2의 10글자를 모두 출력했지만, \0을 만나지 못해 str2 배열의 메모리 공간 바로 다음 공간을 읽기 시작한다.
    이로 인해 "very long "을 출력한 뒤, 바로 뒤에 있던 str3의 메모리로 넘어가서 "abc"를 차례로 발견하고 출력한다.
    이때 "very long "과 "abc" 사이엔 쓰레기 값이 포함되어 있다.
    */
    printf("str2 = %s\n", str2);
    printf(str3);
    printf("\n");
    printf("str4 = %s\n", str4);
}

```

```

// 실행 결과
str1 = Abc
str2 = very long ?????abc
abc
str4 =

```

문자/문자열 처리 함수

- 문자 처리 함수를 사용하려면 `<ctype.h>` 를 포함해야 한다.

문자 처리 함수	설명
<code>isalpha(ch);</code>	알파벳인지 검사한다.
<code>isdigit(ch);</code>	숫자인지 검사한다.
<code>isxdigit(ch);</code>	16진수 숫자인지 검사한다.
<code>isalnum(ch);</code>	알파벳이나 숫자인지 검사한다.
<code>islower(ch);</code>	소문자인지 검사한다.
<code>isupper(ch);</code>	대문자인지 검사한다.
<code>tolower(ch);</code>	소문자로 변환한다.

문자 처리 함수	설명
toupper(ch);	대문자로 변환한다.
isspace(ch);	공백 문자인지 확인한다.

- 문자열 처리 함수를 사용하려면 `<string.h>` 를 포함해야 한다.
- `strcpy` 함수를 사용하기 위해선 `_CRT_SECURE_NO_WARNINGS` 매크로를 정의해야 한다.

문자열 처리 함수	설명
strlen(str);	널 문자를 제외한 str의 길이를 구한다.
strcpy(a, b);	b를 a로 복사한다.
strcmp(a, b);	a와 b를 비교해서 같으면 0을, a > b 이면 0보다 큰 값을, a < b 이면 0보다 작은 값을 리턴한다.
strcat(a, b);	a의 끝에 b를 연결한다.
strchr(a, b);	a에서 b 문자를 찾고, 찾은 문자의 주소를 리턴한다.
strstr(a, b);	a에서 b 문자열을 찾고, 찾은 문자열의 주소를 리턴한다.
strtok(a, b);	b를 이용해서 a를 토큰으로 분리하고, 토큰 문자열을 리턴한다.

```
// 문자열의 토큰 나누기 : strtok(a, b);
// 주어진 문자열 a를 b에 있는 문자들을 이용해서 토큰으로 쪼개고 토큰의 주소(b 이전까지 있었던 문자열)를 리턴한다.
// 토큰이 없으면 NULL을 리턴한다.
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

int main(void) {
    char phone[] = "010-1234-5678";
    char *p = NULL;

    p = strtok(phone, "-"); // strtok 함수 호출 후에 첫 번째 매개변수인 str이 변경되므로 주의해야 한다.
    printf("mobile : %s\n", p);

    p = strtok(NULL, "-"); // strtok 함수의 첫 번째 인자로 NULL을 지정하면 계속해서 다음 토큰을 얻을 수 있다.
    printf("prefix : %s\n", p);

    p = strtok(NULL, "-");
    printf("line no. : %s\n", p);
}
```

```
// 실행 결과
mobile : 010
prefix : 1234
line no. : 5678
```

헤더 파일	함수 원형	설명
<stdio.h>	int sscanf(const char* buff, const char* format, ...);	문자열을 정수나 실수로 변환해서 읽어온다.
	int sprintf(char* buff, const char* format, ...);	형식 문자열을 이용해서 정수나 실수를 문자열로 변환한다.
<stdlib.h>	int atoi(const char* str);	문자열을 정수로 변환한다.
	double atof(const char* str);	문자열을 실수로 변환한다.

헤더 파일	함수 원형	설명
	long atol(const char* str);	문자열을 long형 값으로 변환한다.

입출력 함수	설명
gets_s(str, count);	줄바꿈 문자를 포함하지 않고, 한 줄의 문자열을 읽어서 str에 저장한다.
fgets(str, count, stdin);	줄바꿈 문자를 포함하고, 한 줄의 문자열을 읽어서 str에 저장한다.
puts(str);	str을 출력하고 줄을 바꾼다.

- scanf 함수는 입력 버퍼에서 공백 문자를 만날 때까지 문자열을 읽어온다.
 - 빈칸을 포함한 문자열을 입력받을 때는 scanf 함수 대신 gets 함수를 이용한다.
 - 하지만 gets 함수는 줄바꿈 문자가 입력될 때까지 입력된 문자열을 매개변수에 저장하기 때문에 gets_s 함수 또는 fgets 함수를 사용하는 것이 좋다.

```
char str[128];

fgets(str, sizeof(str), stdin); // 줄바꿈 문자까지를 str로 읽어온다. 이때 str은 줄바꿈 문자가 포함되어 있다.
printf(str); // str 출력 후 줄이 바뀐다.

gets_s(str, sizeof(str)); // 줄바꿈 문자까지를 str로 읽어온다. 이때 str은 줄바꿈 문자가 포함되어 있지 않다.
printf(str); // str 출력 후 줄이 바뀌지 않는다.
sscanf(str, "%d", &n); // str에서 정수를 읽어서 int형 변수 n에 저장한다.

puts("Hello, there!"); // printf("Hello, there!\n"); 과 같다.
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

int main(void) {
    char str_in[128];
    char str_out[128];
    int y, m, d;

    printf("날짜(yyyymmdd)? ");
    gets_s(str_in, sizeof(str_in)); // 한 줄의 문자열을 입력한다.

    sscanf(str_in, "%4d%2d%2d", &y, &m, &d); // 문자열을 int형의 y, m, d로 변환한다.

    sprintf(str_out, "Due Date: %04d-%02d-%02d", y, m, d);
    puts(str_out); // 한 줄의 문자열을 출력한다.
}
```

```
// 실행 결과
날짜(yyyymmdd)? 20250716
Due Date: 2025-07-16
```