



```
35     self.logger = logging.getLogger(__name__)
36     if path:
37         self.file = open(os.path.join(path, 'fingerprint.log'), 'a')
38         self.file.seek(0)
39         self.fingerprints.update(self._read())
40
41     @classmethod
42     def from_settings(cls, settings):
43         debug = settings.getbool('SUPERUSER_DEBUG')
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); // 6자리 폭에 맞춰 오른쪽으로 정렬해서 출력한다.
    printf("%-6d\n", x); // 6자리 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); // 6자리 폭에 맞춰 소수점 이하 2자리로 실수를 오른쪽으로 정렬해서 출력한다.
}

```

```

// 실행 결과
987
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 옵션

// %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.
  - $-n$ 을 2의 보수로 표현하려면, 먼저  $n$ 을 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, fffffff9
y = 65529, 0000ffff
```

- 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
    unsigned char red = 300; // unsigned char의 유효 범위는 0 ~ 255로, 현재 red는

    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
```

## 아스케이프 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 amount
```

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

- 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 기호상수(Symbolic Constant)가 있다.
    - 리터럴 상수: 소스 코드에서 직접 사용되는 값

상수의 종류	구분	예	데이터형
문자형	일반 문자	'x', 'y'	int
	이스케이프 시퀀스	'\wb', '\wt', '\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 2147483647l 처럼 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 표기하라
    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
```

```
// 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  
  
    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
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 오류)
printf("%d %% %d = %d\n", x, y, x % y); // % 문자를 출력하려면 문자열 내에 %% 사용
```

### 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

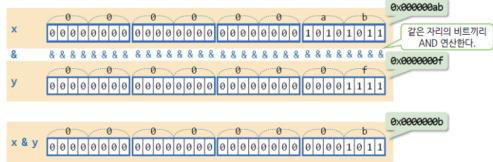
### ❖ 비트 AND 연산자(&)

- 각 비트 단위로 AND 연산을 수행

```
unsigned int x = 0xab;
unsigned int y = 0xaf;
x & y
```

수식의 값은  
0x000000af

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

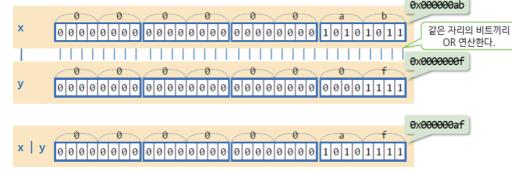
### ❖ 비트 OR 연산자(|)

- 피연산자의 같은 위치에 있는 비트에 대해서 비트 OR 연산을 수행

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

수식의 값은  
0x000000af

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



하나라도 1이면 1, 둘 다 0일때만 0

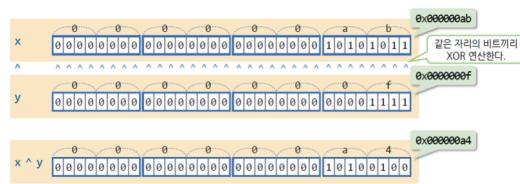
### ❖ 비트 XOR 연산자(^)

- 피연산자의 같은 위치에 있는 비트에 대해서 비트 XOR 연산을 수행

```
unsigned int x = 0xab;
unsigned int y = 0xaf;
x ^ y
```

수식의 값은  
0x000000a4

a의 비트	b의 비트	a의 비트 ^ b의 비트
0	0	0
0	1	1
1	0	1
1	1	0



둘 다 같으면 0, 서로 다르면 1

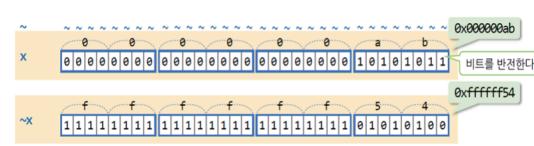
### ❖ 비트 NOT 연산자(~)

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

```
unsigned int x = 0xab;
~x
```

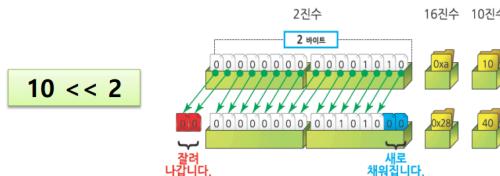
수식의 값은  
0xffffffff54

a의 비트	~a의 비트
0	1
1	0



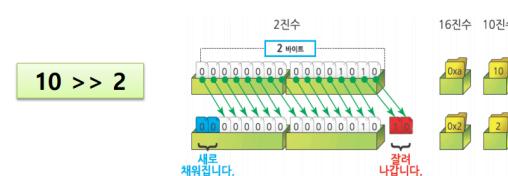
### ❖ 비트 왼쪽 이동 연산자 <<

- 비트들을 왼쪽으로 이동(<<)시킴
- 왼쪽으로 밀려난 비트는 사라지고, 오른쪽 빈자리는 0이 채워짐
- N비트 왼쪽 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

- $z = x >> 2; \rightarrow x / 2$ 의 제곱

#### ◦ 비트 왼쪽 이동 연산자

- $z = x << 2; \rightarrow x * 2$ 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 }
```

10진수  
171에 해당

### 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
```