

Week 3

# Chapter 2

# Types,

# Operators, and

# Expressions

CSE2018 시스템프로그래밍기초  
2016년 2학기

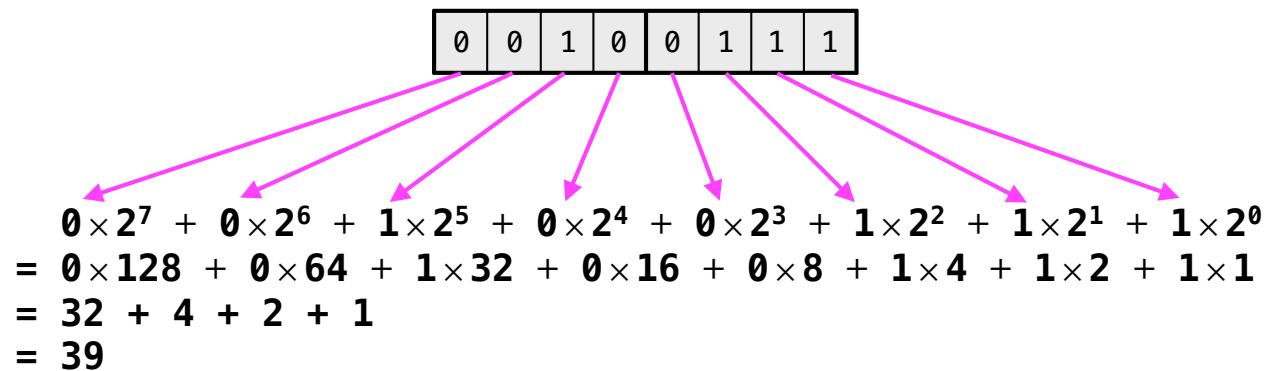
한양대학교 ERICA  
컴퓨터공학과 => 소프트웨어학부  
도경구

# 10진수

2016

$$\begin{aligned}
 & 2 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 6 \times 10^0 \\
 &= 2 \times 1000 + 0 \times 100 + 1 \times 10 + 6 \times 1 \\
 &= 2000 + 0 + 10 + 6 \\
 &= 2016
 \end{aligned}$$

# 2진수



$$\begin{aligned}
 & 0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
 &= 0 \times 128 + 0 \times 64 + 1 \times 32 + 0 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 \\
 &= 32 + 4 + 2 + 1 \\
 &= 39
 \end{aligned}$$

		bits	표현가능 수의 개수	수의 범위
unsigned	char	8	$2^8 = 256$	0 ~ 255
signed	short / int	16	$2^{15} = 32,768$	0 ~ 32,767
signed	int / long	32	$2^{31} = 2,147,483,648$	0 ~ 2,147,483,647

# 데이터 타입와 크기

## Data Types & Sizes

type	size	qualifiers
char	1 byte (8 bits)	signed : 0 ~ 255 ( $2^8$ ) unsigned : -128 ~ 127 ( $2^7$ )
int	an integer (16 or 32 bits)	short ( $\geq 16$ ) long ( $\geq 32$ ) signed unsigned
float	single-precision floating point	
double	double-precision floating point	long

정식 표현	약식 표현
char	
signed char	
unsigned char	
signed short int	short
signed int	int
signed long int	long
unsigned short int	unsigned short
unsigned int	unsigned int
unsigned long int	unsigned long
float	
double	
long double	

```
short int i;
short i;
long int counter;
long counter;
```

참고: C Standard Library  
<limits.h> and <float.h>

문자

Characters

정수로 인코딩

'x' \ : Escape 문자

ASCII Code Chart

Legend:

Alphabetic

Control character

Numeric digit

PunctuationExtended punctuationInternational

이름	특수문자	값
alert	\a	7
backslash	\\	92
backspace	\b	8
carriage return	\r	13
double quote	\"	34
formfeed	\f	12
horizontal tab	\t	9
newline	\n	10
null character	\0	0
single quote	\'	39
vertical tab	\v	11
question mark	\?	63

ASCII (1977/1986)																
	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	NUL 0000 0	SOH 0001 1	STX 0002 2	ETX 0003 3	EOT 0004 4	ENQ 0005 5	ACK 0006 6	BEL 0007 7	BS 0008 8	HT 0009 9	LF 000A 10	VT 000B 11	FF 000C 12	CR 000D 13	SO 000E 14	SI 000F 15
1_	DLE 0010 16	DC1 0011 17	DC2 0012 18	DC3 0013 19	DC4 0014 20	NAK 0015 21	SYN 0016 22	ETB 0017 23	CAN 0018 24	EM 0019 25	SUB 001A 26	ESC 001B 27	FS 001C 28	GS 001D 29	RS 001E 30	US 001F 31
2_	SP 0020 32	! 0021 33	" 0022 34	# 0023 35	\$ 0024 36	% 0025 37	& 0026 38	' 0027 39	( 0028 40	) 0029 41	* 002A 42	+ 002B 43	, 002C 44	- 002D 45	. 002E 46	/ 002F 47
3_	0 0030 48	1 0031 49	2 0032 50	3 0033 51	4 0034 52	5 0035 53	6 0036 54	7 0037 55	8 0038 56	9 0039 57	: 003A 58	; 003B 59	< 003C 60	= 003D 61	> 003E 62	? 003F 63
4_	@ 0040 64	A 0041 65	B 0042 66	C 0043 67	D 0044 68	E 0045 69	F 0046 70	G 0047 71	H 0048 72	I 0049 73	J 004A 74	K 004B 75	L 004C 76	M 004D 77	N 004E 78	O 004F 79
5_	P 0050 80	Q 0051 81	R 0052 82	S 0053 83	T 0054 84	U 0055 85	V 0056 86	W 0057 87	X 0058 88	Y 0059 89	Z 005A 90	[ 005B 91	\ 005C 92	] 005D 93	^ 005E 94	_ 005F 95
6_	` 0060 96	a 0061 97	b 0062 98	c 0063 99	d 0064 100	e 0065 101	f 0066 102	g 0067 103	h 0068 104	i 0069 105	j 006A 106	k 006B 107	l 006C 108	m 006D 109	n 006E 110	o 006F 111
7_	p 0070 112	q 0071 113	r 0072 114	s 0073 115	t 0074 116	u 0075 117	v 0076 118	w 0077 119	x 0078 120	y 0079 121	z 007A 122	{ 007B 123	 007C 124	} 007D 125	~ 007E 126	DEL 007F 127

# 문자

## Characters

정수로 인코딩

```
char c = 'a';

printf("%c", c);    /* prints a */
printf("%d", c);    /* prints 97 */

printf("%c%c%c", c, c + 1, c + 2);    /* prints abc */
```

```
char c;
int i;

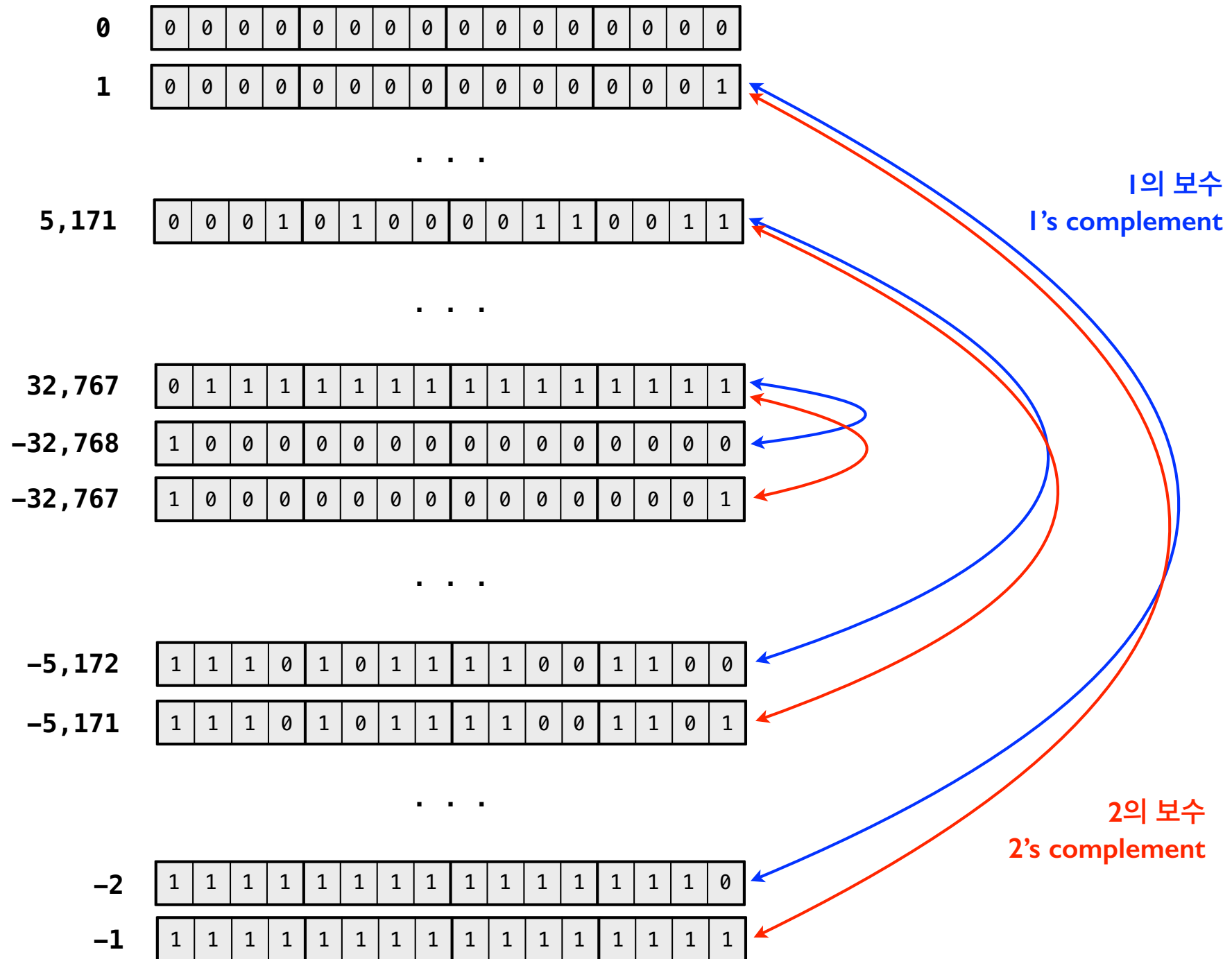
for (i = 'a'; i <= 'z'; ++i)
    printf("%c", i);    /* prints abcdefghijklmnopqrstuvwxyz */
for (c = 65; c <= 90; ++c)
    printf("%c", c);    /* prints ABCDEFGHIJKLMNOPQRSTUVWXYZ */
for (c = '0'; c <= '9'; ++c)
    printf("%d", c);    /* prints 48 49 50 51 52 53 54 55 56 57 */
```

수

Numerals

literal	type
1234	int
123456789 <del>l</del> , 123456789 <del>L</del>	long
1234 <del>u</del> , 1234 <del>U</del>	unsigned int
123456789 <del>ul</del> , 123456789 <del>UL</del>	unsigned long
123.4 <del>f</del> , 1e-2 <del>F</del>	float
123.4, 1e-2	double
123.4 <del>l</del> , 1e-2 <del>L</del>	long double

0 ~ 32,767



10진수 decimal	8진수 octal
-----------------	--------------

— 0 —

0 00

**1** **01**

2 02

3 03

4 04

5 05

6 06

**7 07**

8                      010

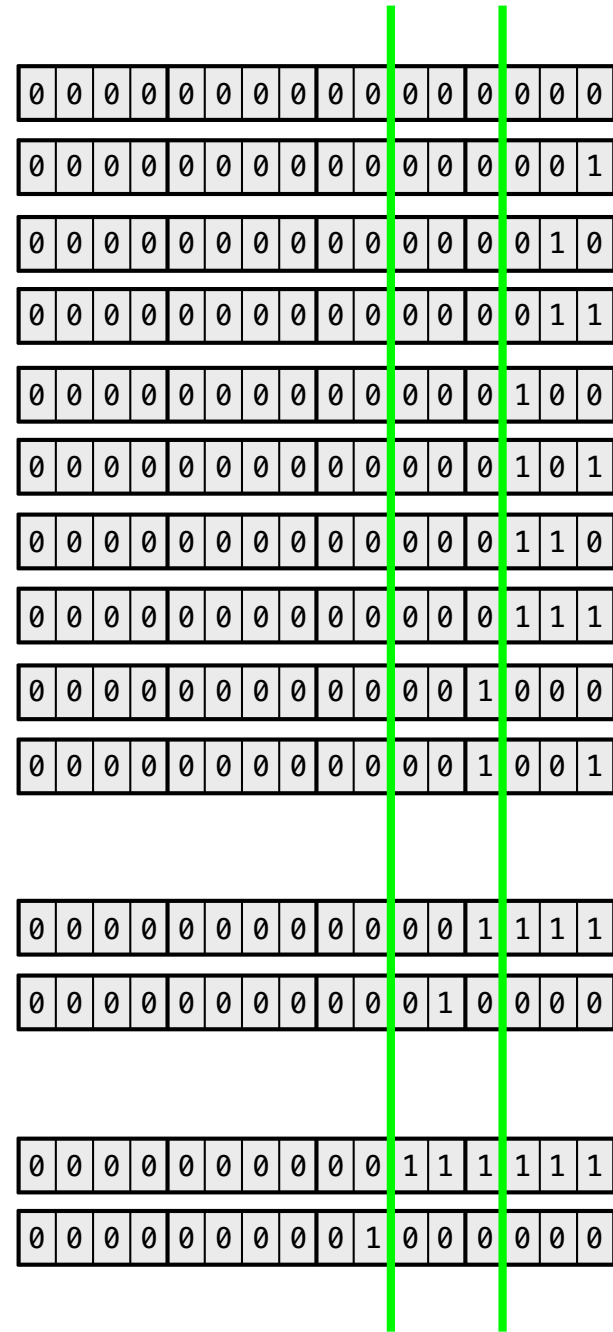
9                      011

15 017

**16** **020**

63 077

64                      0100





10진수  
decimal

16진수  
hexa-  
decimal

—	0X— 0x—	
0	0x0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1	0x1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
2	0x2	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
3	0x3	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
4	0x4	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
5	0x5	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
6	0x6	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
7	0x7	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
8	0x8	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
9	0x9	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1
10	0xA	0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0
11	0xB	0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0
12	0xC	0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0
13	0xD	0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1
14	0xE	0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0
15	0xF	0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0
16	0x10	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0

10진수  
decimal

16진수  
hexa-  
decimal

—	0X— 0x—	
17	0x11	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1
255	0xFF	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
256	0x100	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
4095	0xFFFF	0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
4096	0x1000	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

# 문자열 String

"hello, world"

0	1	2	3	4	5	6	7	8	9	10	11	12
'h'	'e'	'l'	'l'	'o'	','	' '	'w'	'o'	'r'	'l'	'd'	'\0'

....

0
'\0'

"hello," " world" => "hello, world"  
at compile-time

strlen("hello, world")

참고: C Standard Library <string.h>

## Question

'x' 와 "x" 는 같은가, 다른가?

## 열거형 상수 Enumeration Constant

```
enum boolean { NO, YES };  
              ↑   ↑  
              0   1
```

```
enum escapes { BELL = '\a', BACKSPACE = '\b', TAB = '\t',  
               NEWLINE = '\n', VTAB = '\v', RETURN = '\r' };
```

```
enum months { JAN = 1, FEB, MAR, APR, MAY, JUN,  
              JUL, AUG, SEP, OCT, NOV, DEC };  
              ↑   ↑   ↑   ↑   ↑   ↑  
              7   8   9  10  11  12  
              2   3   4   5   6  
              ↓   ↓   ↓   ↓   ↓
```

- 이름 중복은 허용하지 않음
- 여러 이름이 같은 값을 갖는 건 괜찮음

# 선언 Declarations

- 변수는 사용 전에 반드시 선언해야 함
- 타입을 반드시 명시해야 함

```
int lower, upper, step;
char c, line[1000];

int lower;
int upper;
int step;
char c;
char line[1000];
```

- 선언과 동시에 초기값 지정 가능

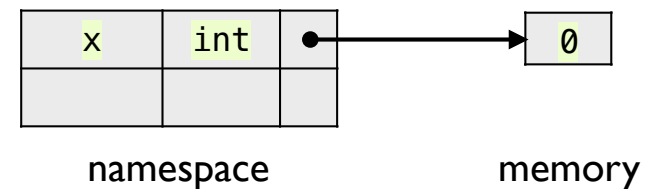
```
char esc = '\\';
int i = 0;
int limit = MAXLINE+1;
float eps = 1.0e-5;
```

- 수정불가 변수

```
const double e = 2.718281845905;
const char msg[] = "warning: ";
int strlen(const char);
```

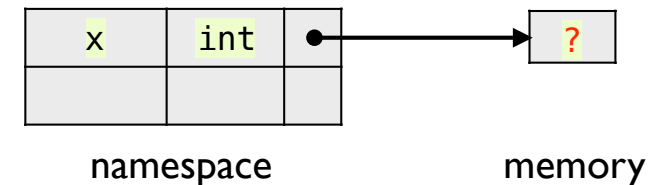
- 외부변수(전역변수) **External (Global) variables**
- 정적변수 **Static variables**

```
external int x;
```



- 자동변수(지역변수) **Automatic variables (Internal, Local)**

```
int x;
```



## 변수 작명규칙 Variable Names

문자 (문자 | 숫자 | '\_' )

변수이름은 모조리 소문자  
상수이름은 모조리 대문자

대소문자 구별  
앞 31개 문자만으로 구별  
외부변수 길이는 6이하로

if, else, int, float 등은 키워드로 예약되어 있어서 변수이름으로 쓸 수 없음  
키워드는 모두 소문자

변수이름은 용도에 맞게 잘 지어야 함

## 산술연산자 Arithmetic Operators

연산자	의미
+	더하기
-	빼기
*	곱하기
/	나누기
%	나머지     int only

```
if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
    printf("%d is a leap year\n", year);
else
    printf("%d is not a leap year\n", year);
```

## 비교연산자 Relational Operators

연산자	의미	우선순위
>	크다	높음
>=	크거나 같다	
<	작다	
<=	작거나 같다	낮음
==	같다	
!=	다르다	

- 산술연산자보다 우선순위가 낮음
- 단축연산

## 논리연산자 Logical Operators

연산자	의미	우선순위
!	논리역	가장높음
&&	논리곱	높음
	논리합	낮음

- 비교연산자보다 우선순위가 낮음
- 단축연산

```

for (i = 0; 1 i < lim - 1 && 2 (c = getchar()) != '\n' && 3 c != EOF; ++i)
    s[i] = c;

```

논리값	의미
1<	참
0	거짓

```
if (!valid)
```

```
if (valid == 0)
```



# 타입변환 Type Conversion

char

```
/* atoi: convert s to integer */
int atoi(char s[]) {
    int i, n;

    n = 0;
    for (i = 0; s[i] >= '0' && s[i] <= '9'; ++i)
        n = 10 * n + (s[i] - '0');
    return n;
}
```

```
/* lower: convert c to lower case (ASCII only) */
int lower(int c) {
    if (c >= 'A' && c <= 'Z')
        return c + 'a' - 'A';
    else
        return c;
}
```

# 타입변환 Type Conversion

char

C Standard Library <ctype.h>

```
/* atoi: convert s to integer */
int atoi(char s[]) {
    int i, n;

    n = 0;
    for (i = 0; s[i] >= '0' && s[i] <= '9'; ++i)
        n = 10 * n + (s[i] - '0');
    return n;
}
```

```
/* atoi: convert s to integer */
int atoi(char s[]) {
    int i, n;

    n = 0;
    for (i = 0; isdigit(c); ++i)
        n = 10 * n + (s[i] - '0');
    return n;
}
```

```
/* lower: convert c to lower case (ASCII only) */
int lower(int c) {
    if (c >= 'A' && c <= 'Z')
        return c + 'a' - 'A';
    else
        return c;
}
```

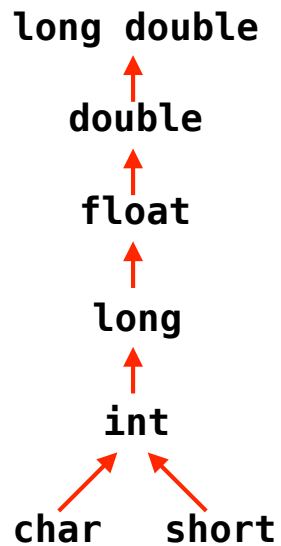
**tolower(c)**

# 타입변환 Type Conversion

## 자동 타입 변환 Implicit Type Conversion

- 산술연산 (arithmetic & relational operation)
- 지정문 (assignment statement)
- 함수호출 (function call)

- unsigned 가 없는 경우 변환 규칙



- 정보 손실이 없는 경우

```
float x;  
int i;  
  
x = i;  
i = x;
```

- 정보 손실이 있는 경우

```
float x;  
int i;  
  
i = x;  
x = i;
```

- unsigned 가 섞이면 타입변환 규칙이 매우 복잡해짐

## 타입변환 Type Conversion

강제 타입 변환  
Explicit Type Conversion  
(Type Casting)

*(type-name) expression*

```
int n;  
root = sqrt((double) n)
```

sqrt in <math.h>

```
double sqrt(double);  
root2 = sqrt(2)
```

```
unsigned long int next = 1;  
  
/* rand: return pseudo-random integer on 0..32767 */  
int rand(void) {  
    next = next * 1103515245 + 21345;  
    return (unsigned int)(next/65536) % 32768;  
}  
  
/* srand: set seed for rand() */  
void srand(unsigned int seed) {  
    next = seed;  
}
```

# 증가연산자 Increment Operator

# 감소연산자 Decrement Operator

++

n++;  
++n;

--

n--;  
--n;

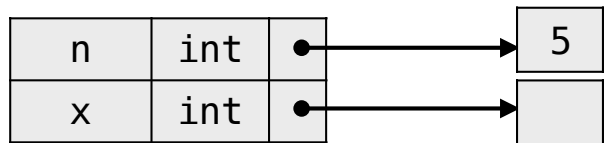
~~(i+j)++~~

program

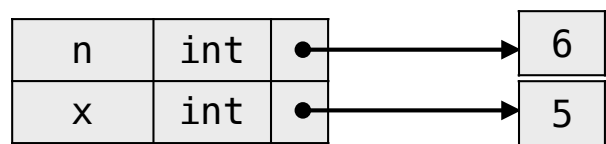
namespace

memory

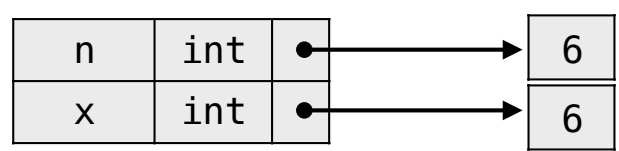
n = 5;



x = n++;



x = ++n;



## 증가연산자 Increment Operator

## 감소연산자 Decrement Operator

```
/* squeeze: delete all c from s */  
void squeeze(char s[], int c) {  
    int i, j;  
  
    for (i = j = 0; s[i] != '\0'; i++)  
        if (s[i] != c)  
            s[j++] = s[i];  
    s[j] = '\0';  
}
```

**s[j] = s[i];**  
**j++;**



## 증가연산자 Increment Operator

## 감소연산자 Decrement Operator

```
/* squeeze: delete all c from s */
void squeeze(char s[], int c) {
    int i, j;

    for (i = j = 0; s[i] != '\0'; i++)
        if (s[i] != c)
            s[j++] = s[i];
    s[j] = '\0';
}
```

**s[j] = s[i];**  
**j++;**

```
/* readline: read a line into s,
return length */
int readline(char s[], int lim) {
    int c, i;

    for (i = 0; i < lim - 1 && (c =
getchar()) != EOF && c != '\n'; ++i)
        s[i] = c;
    if (c == '\n') {
        s[i] = c;
        ++i;
    }
    s[i] = '\0';
    return i;
}
```

**if (c == '\n')**  
**s[i++] = c;**

## 증가연산자 Increment Operator

## 감소연산자 Decrement Operator

```
/* strcat: concatenate t to end of s;  
           s must be big enough to hold the two */  
void strcat(char s[], char t[]) {  
    int i, j;  
  
    i = j = 0;  
    while (s[i] != '\0') /* find end of s */  
        i++;  
    while ((s[i++] = t[j++]) != '\0') /* copy t */  
        ;  
}
```

```
char sw[10] = "ERI";  
strcat(sw, "CA");
```



## 증가연산자 Increment Operator

## 감소연산자 Decrement Operator

```
/* strcat: concatenate t to end of s;  
           s must be big enough to hold the two */  
void strcat(char s[], char t[]) {  
    int i, j;  
  
    i = j = 0;  
    while (s[i] != '\0') /* find end of s */  
        i++;  
    while ((s[i++] = t[j++]) != '\0') /* copy t */  
        ;  
}
```

```
char sw[10] = "ERI";  
strcat(sw, "CA");
```

## 증가연산자 Increment Operator

## 감소연산자 Decrement Operator

```
/* strcat: concatenate t to end of s;  
           s must be big enough to hold the two */  
void strcat(char s[], char t[]) {  
    int i, j;  
  
    i = j = 0;  
    while (s[i] != '\0') /* find end of s */  
        i++;  
    while ((s[i++] = t[j++]) != '\0') /* copy t */  
        ;  
}
```

```
char sw[10] = "ERI";  
strcat(sw, "CA");
```

sw	char[]	

0	1	2	3	4	5	6	7	8	9
'E'	'R'	'I'	'\0'						

0	1	2	3
'E'	'R'	'I'	'\0'

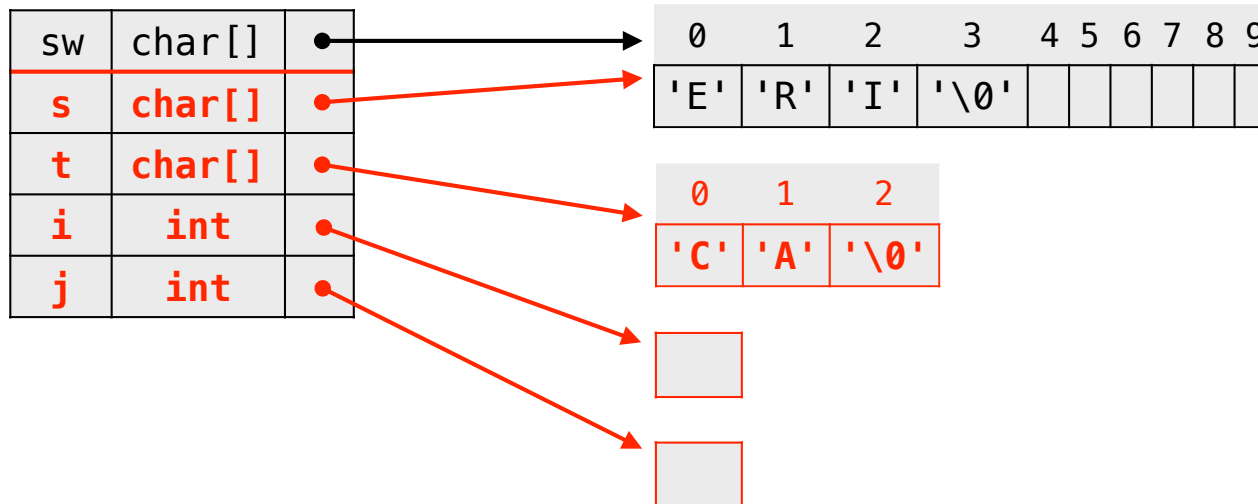
## 증가연산자 Increment Operator

## 감소연산자 Decrement Operator

```
/* strcat: concatenate t to end of s;
   s must be big enough to hold the two */
void strcat(char s[], char t[]) {
    int i, j;

    i = j = 0;
    while (s[i] != '\0') /* find end of s */
        i++;
    while ((s[i++] = t[j++]) != '\0') /* copy t */
        ;
}
```

```
char sw[10] = "ERI";
strcat(sw, "CA");
```



0	1	2	3
'E'	'R'	'I'	'\0'

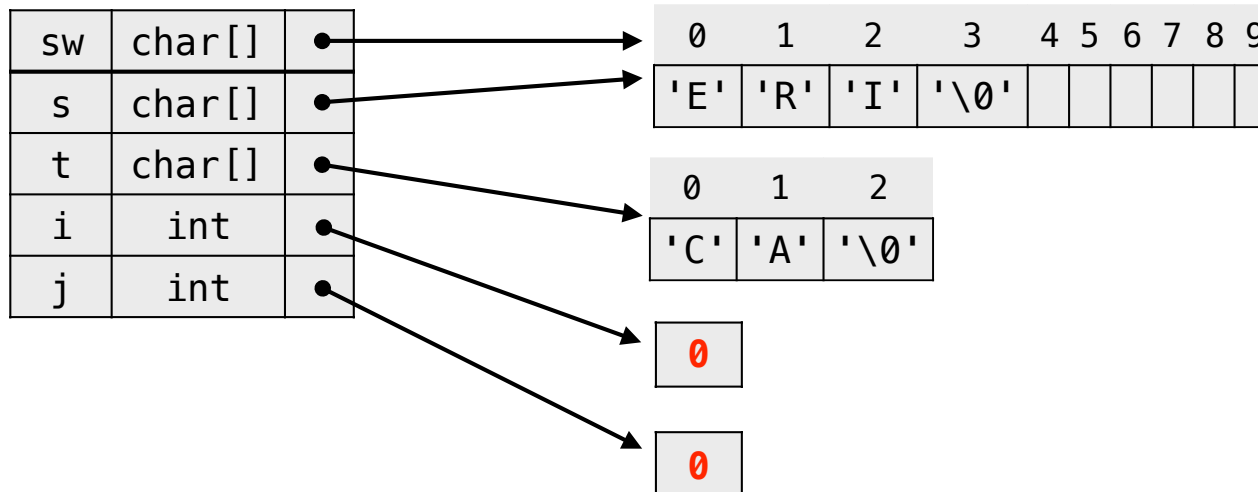
## 증가연산자 Increment Operator

## 감소연산자 Decrement Operator

```
/* strcat: concatenate t to end of s;
   s must be big enough to hold the two */
void strcat(char s[], char t[]) {
    int i, j;

    i = j = 0;
    while (s[i] != '\0') /* find end of s */
        i++;
    while ((s[i++] = t[j++]) != '\0') /* copy t */
        ;
}
```

```
char sw[10] = "ERI";
strcat(sw, "CA");
```



0	1	2	3
'E'	'R'	'I'	'\0'

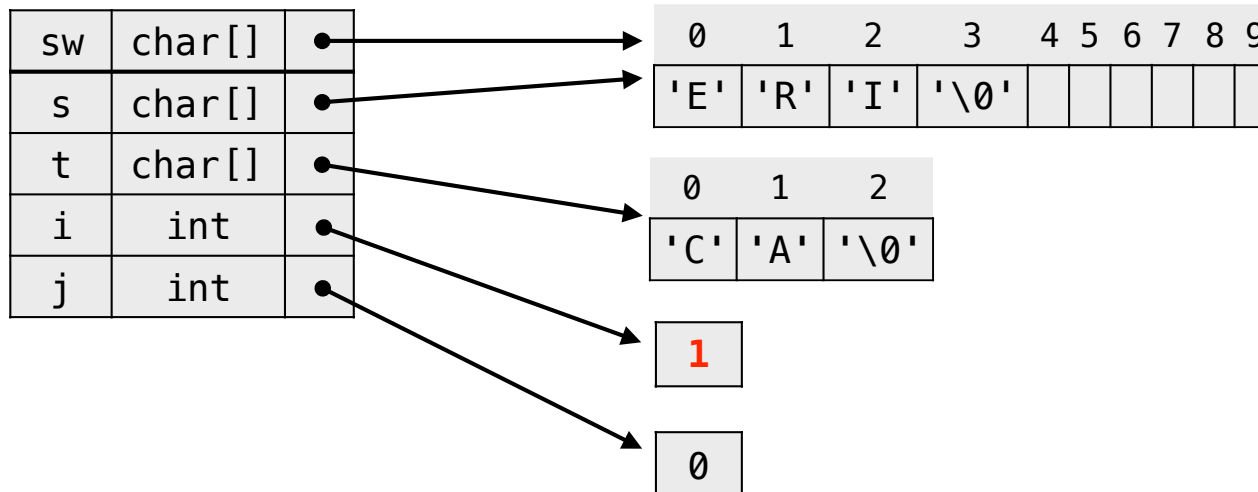
## 증가연산자 Increment Operator

## 감소연산자 Decrement Operator

```
/* strcat: concatenate t to end of s;
   s must be big enough to hold the two */
void strcat(char s[], char t[]) {
    int i, j;

    i = j = 0;
    while (s[i] != '\0') /* find end of s */
        i++;
    while ((s[i++] = t[j++]) != '\0') /* copy t */
        ;
}
```

```
char sw[10] = "ERI";
strcat(sw, "CA");
```



0	1	2	3
'E'	'R'	'I'	'\0'

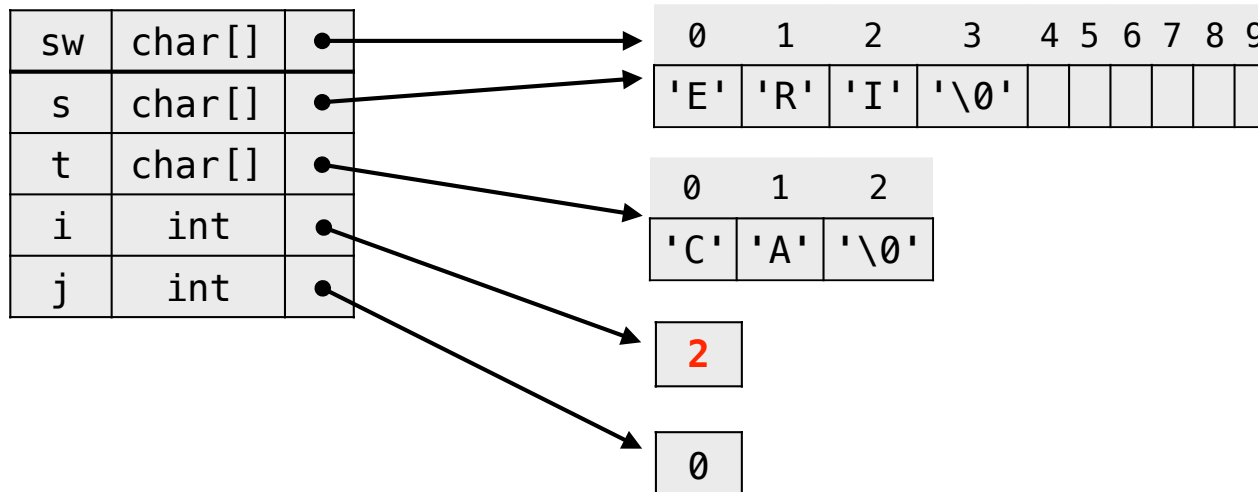
## 증가연산자 Increment Operator

## 감소연산자 Decrement Operator

```
/* strcat: concatenate t to end of s;
   s must be big enough to hold the two */
void strcat(char s[], char t[]) {
    int i, j;

    i = j = 0;
    while (s[i] != '\0') /* find end of s */
        i++;
    while ((s[i++] = t[j++]) != '\0') /* copy t */
        ;
}
```

```
char sw[10] = "ERI";
strcat(sw, "CA");
```



0	1	2	3
'E'	'R'	'I'	'\0'

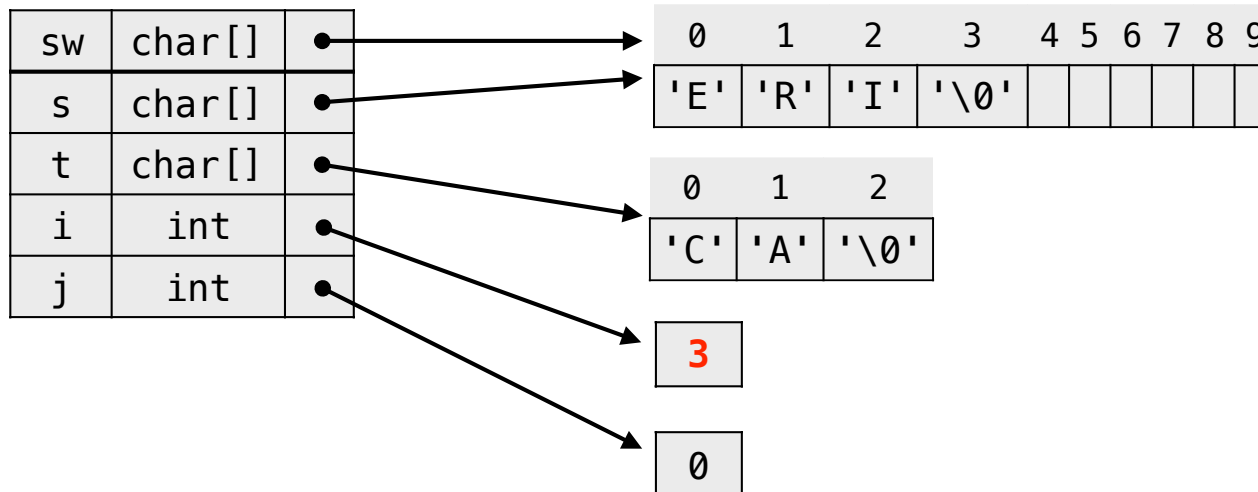
## 증가연산자 Increment Operator

## 감소연산자 Decrement Operator

```
/* strcat: concatenate t to end of s;
   s must be big enough to hold the two */
void strcat(char s[], char t[]) {
    int i, j;

    i = j = 0;
    while (s[i] != '\0') /* find end of s */
        i++;
    while ((s[i++] = t[j++]) != '\0') /* copy t */
        ;
}
```

```
char sw[10] = "ERI";
strcat(sw, "CA");
```



0	1	2	3
'E'	'R'	'I'	'\0'

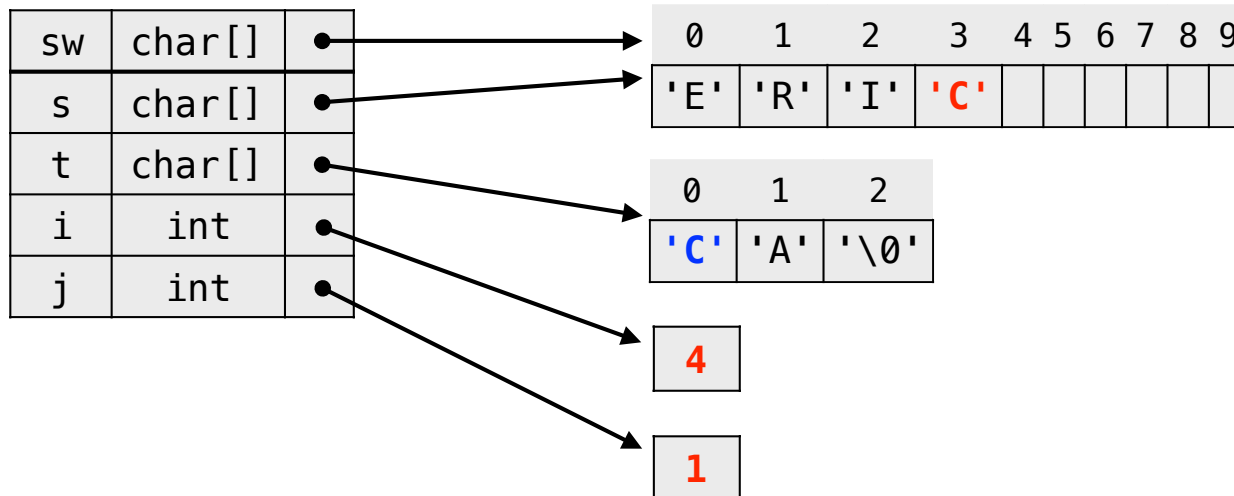
## 증가연산자 Increment Operator

## 감소연산자 Decrement Operator

```
/* strcat: concatenate t to end of s;
   s must be big enough to hold the two */
void strcat(char s[], char t[]) {
    int i, j;

    i = j = 0;
    while (s[i] != '\0') /* find end of s */
        i++;
    while ((s[i++] = t[j++]) != '\0') /* copy t */
        ;
}
```

```
char sw[10] = "ERI";
strcat(sw, "CA");
```



0	1	2	3
'E'	'R'	'I'	'\0'



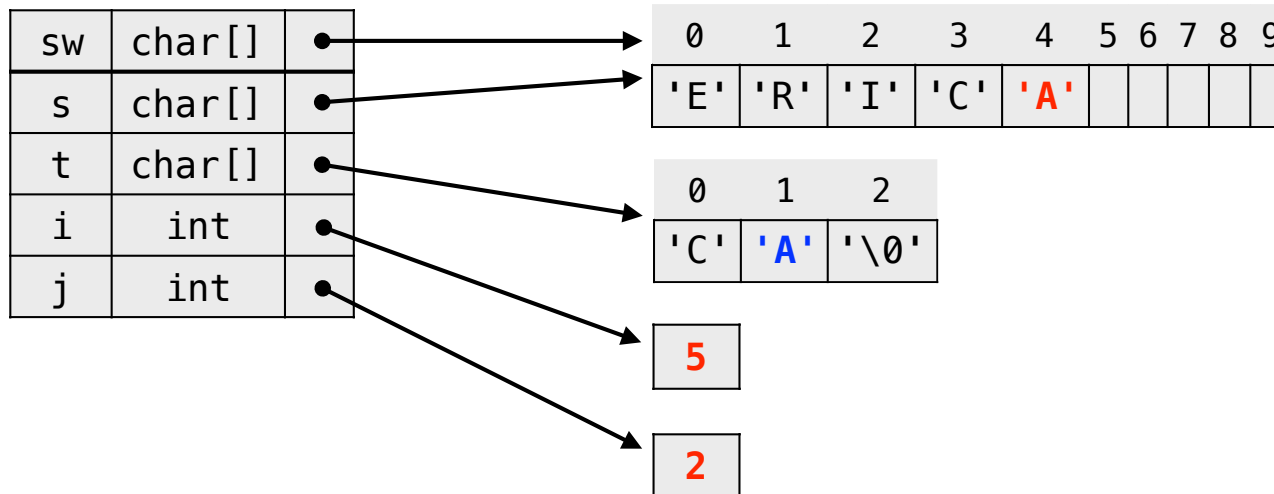
## 증가연산자 Increment Operator

## 감소연산자 Decrement Operator

```
/* strcat: concatenate t to end of s;
   s must be big enough to hold the two */
void strcat(char s[], char t[]) {
    int i, j;

    i = j = 0;
    while (s[i] != '\0') /* find end of s */
        i++;
    while ((s[i++] = t[j++]) != '\0') /* copy t */
        ;
}
```

```
char sw[10] = "ERI";
strcat(sw, "CA");
```



0	1	2	3
'E'	'R'	'I'	'\0'

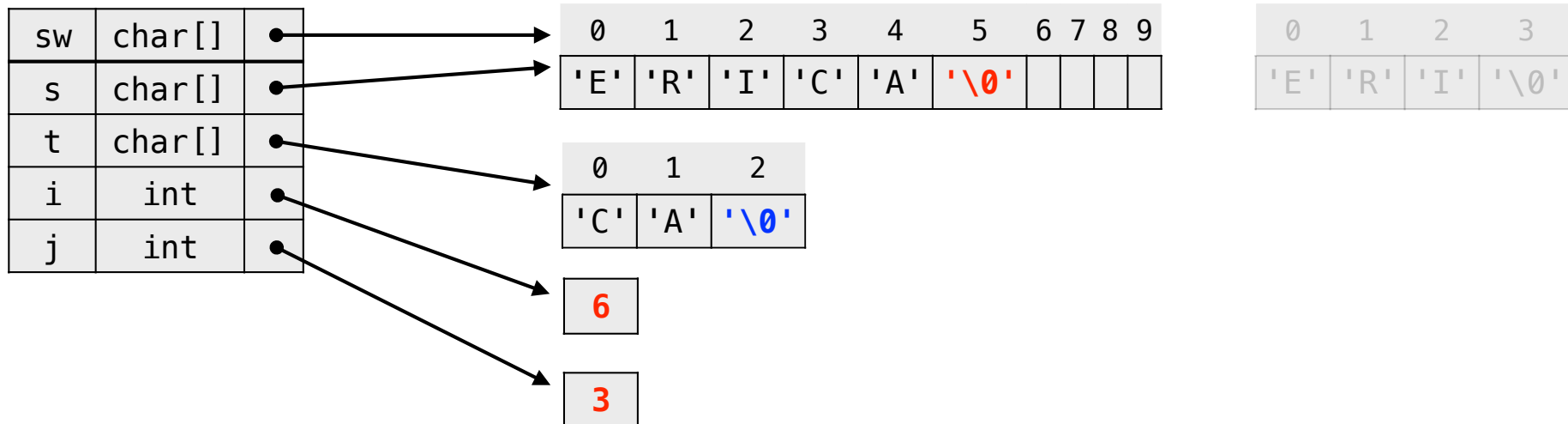
## 증가연산자 Increment Operator

## 감소연산자 Decrement Operator

```
/* strcat: concatenate t to end of s;
   s must be big enough to hold the two */
void strcat(char s[], char t[]) {
    int i, j;

    i = j = 0;
    while (s[i] != '\0') /* find end of s */
        i++;
    while ((s[i++] = t[j++]) != '\0') /* copy t */
        ;
}
```

```
char sw[10] = "ERI";
strcat(sw, "CA");
```



## 증가연산자 Increment Operator

## 감소연산자 Decrement Operator

```
/* strcat: concatenate t to end of s;
   s must be big enough to hold the two */
void strcat(char s[], char t[]) {
    int i, j;

    i = j = 0;
    while (s[i] != '\0') /* find end of s */
        i++;
    while ((s[i++] = t[j++]) != '\0') /* copy t */
        ;
}
```

```
char sw[10] = "ERI";
strcat(sw, "CA");
```

sw	char[]	

0	1	2	3	4	5	6	7	8	9
'E'	'R'	'I'	'C'	'A'	'\0'				

0	1	2	3
'E'	'R'	'I'	'\0'

0	1	2
'C'	'A'	'\0'

6

3

# 비트조작 연산자 Bitwise Operators

char  
short  
int  
long

## 논리연산자 Logical Operators

연산자	실행의미	-nary	-fix
&	bitwise AND	binary	infix
	bitwise inclusive OR	binary	infix
^	bitwise exclusive OR	binary	infix
~	one's complement (unary)	unary	prefix

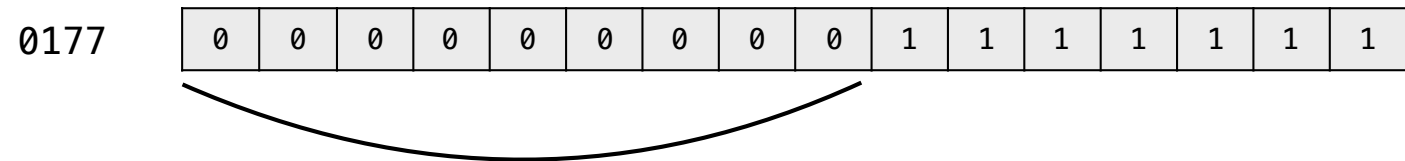
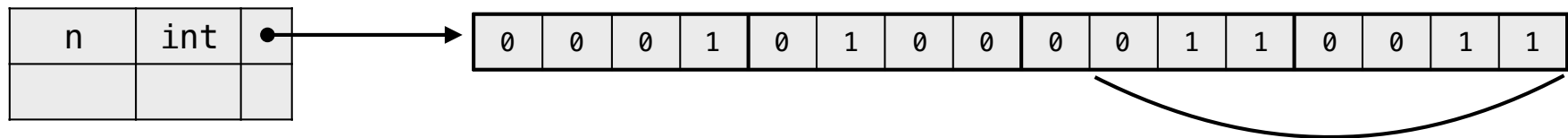
a	b	a & b	a   b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

a	~a
0	1
1	0

# 비트조작 연산자 Bitwise Operators

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

연산자	실행의미	-nary	-fix
&	bitwise AND	binary	infix

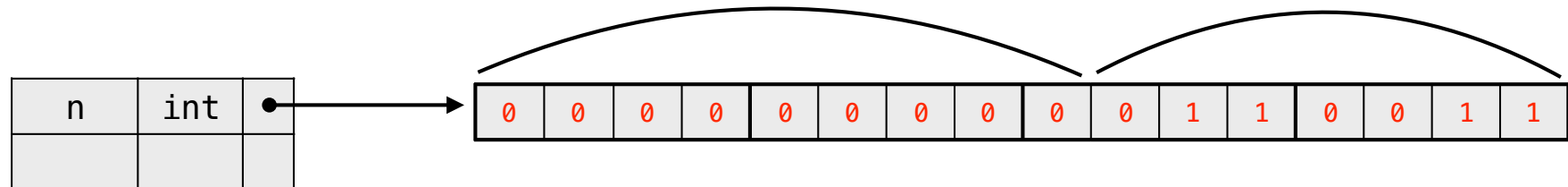


**n = n & 0177**



실행전

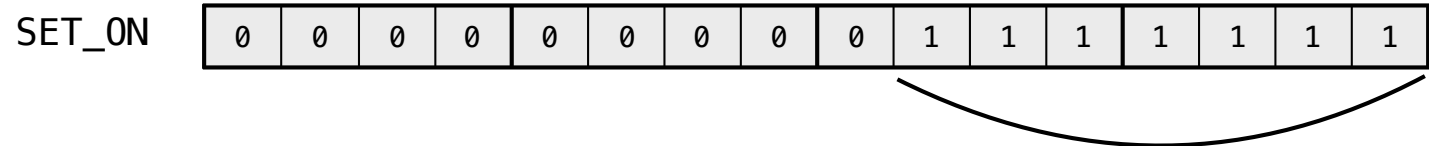
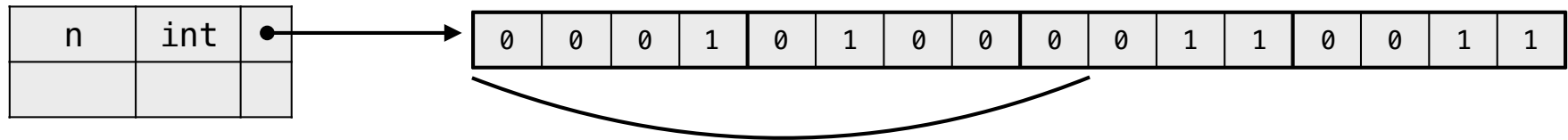
실행후



# 비트조작 연산자 Bitwise Operators

a	b	a   b
0	0	0
0	1	1
1	0	1
1	1	1

연산자	실행의미	-nary	-fix
	bitwise inclusive OR	binary	infix

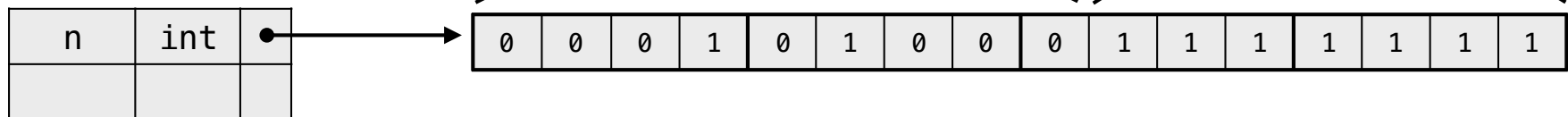


`n = n | SET_ON`



실행전

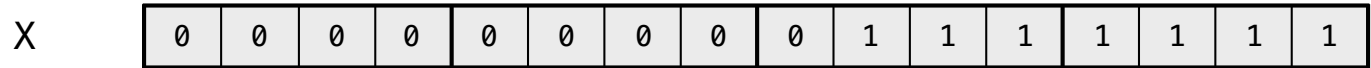
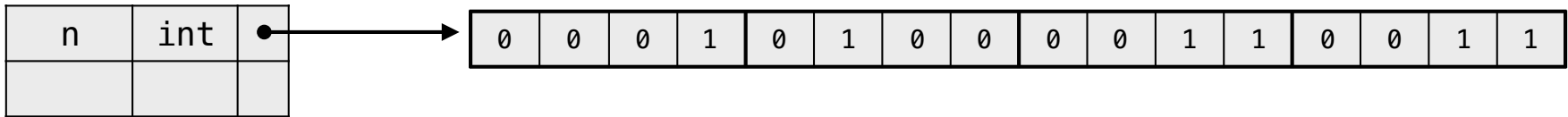
실행후



## 비트조작 연산자 Bitwise Operators

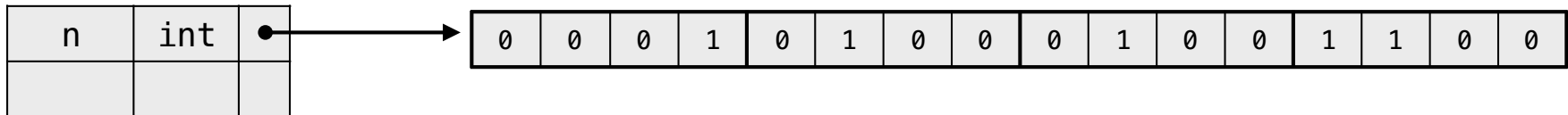
a	b	$a \wedge b$
0	0	0
0	1	1
1	0	1
1	1	0

연산자	실행의미	-nary	-fix
$\wedge$	bitwise exclusive OR	binary	infix


$$n = n^X$$

## 실행전

실행후



# Bitwise Complement

연산자	실행의미	-nary	-fix
~	one's complement (unary)	unary	prefix

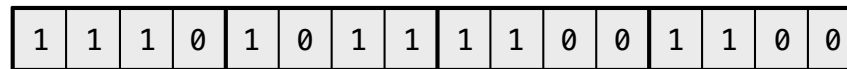
`int n = 5171`



`= 5171`

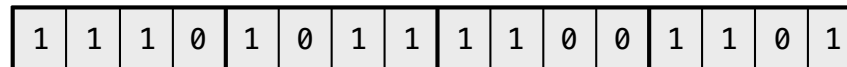
one's complement

`~n`



`= -5172`

two's complement



`= -5171`

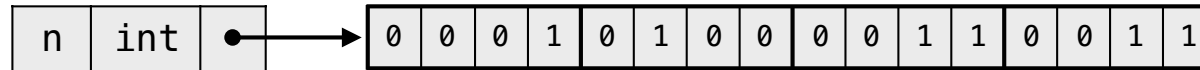


# 비트조작 연산자 Bitwise Operators

char  
short  
int  
long

## Shift Operators

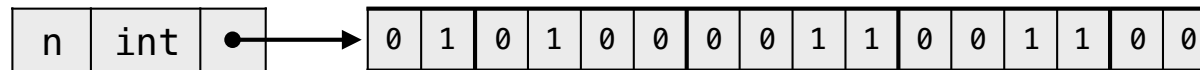
연산자	실행의미	-nary	-fix
<<	left shift	binary	infix
>>	right shift	binary	infix



n = n << 2

실행전

실행후



n = n >> 1

실행전

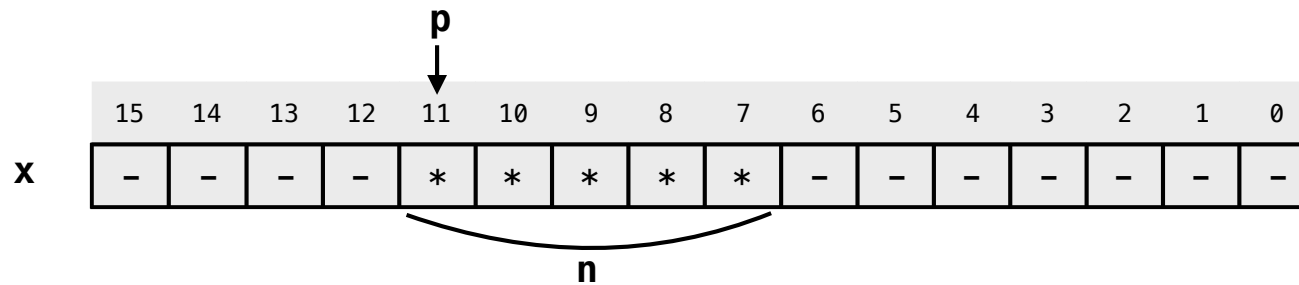
실행후



# 비트조작 연산자

## Bitwise Operators

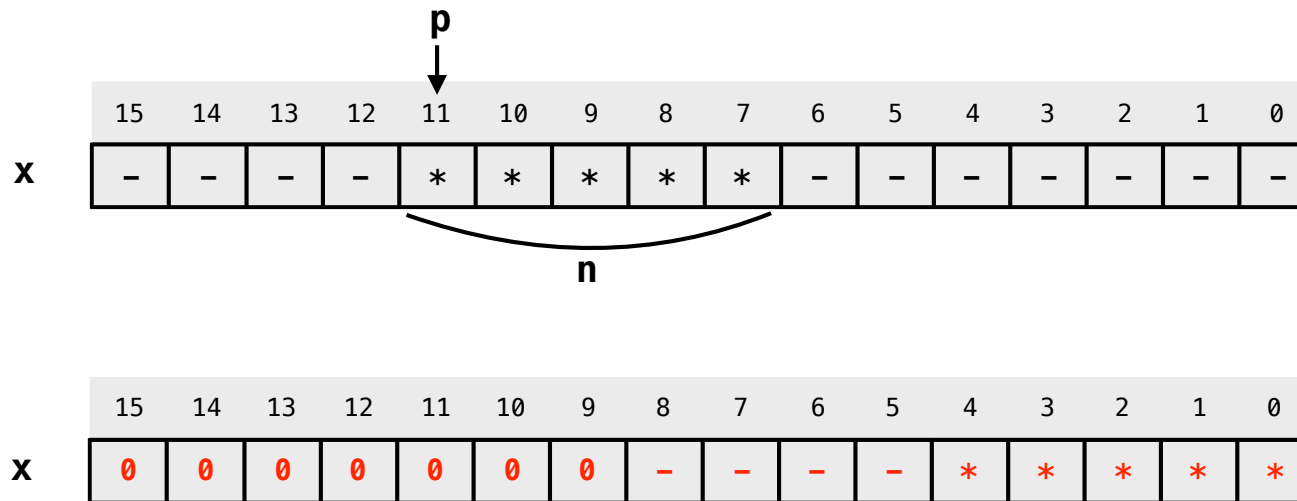
```
/* getbits: get n bits from position p */  
unsigned getbits(unsigned x, int p, int n) {  
    return (x >> (p+1-n)) & ~(~0 << n);  
}
```



# 비트조작 연산자

## Bitwise Operators

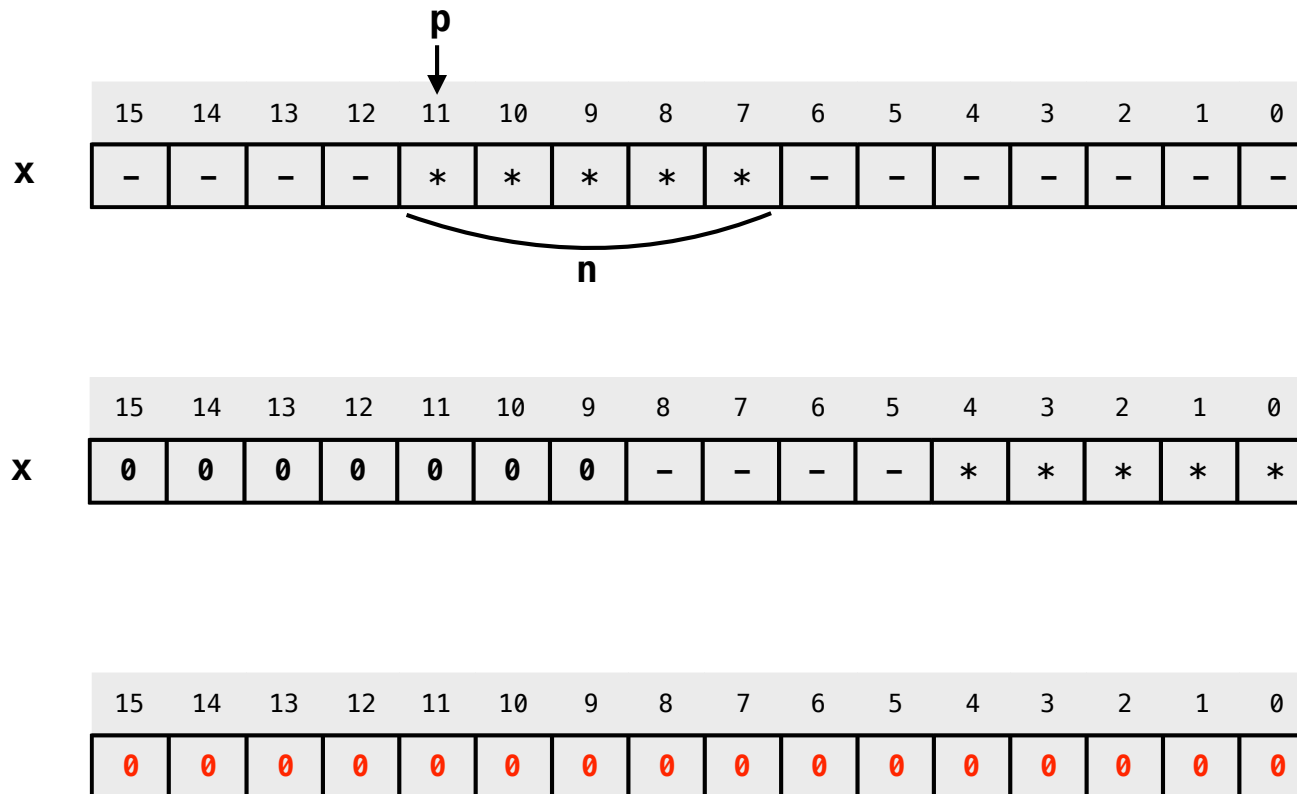
```
/* getbits: get n bits from position p */  
unsigned getbits(unsigned x, int p, int n) {  
    return (x >> (p+1-n)) & ~(~0 << n);  
}
```



# 비트조작 연산자

## Bitwise Operators

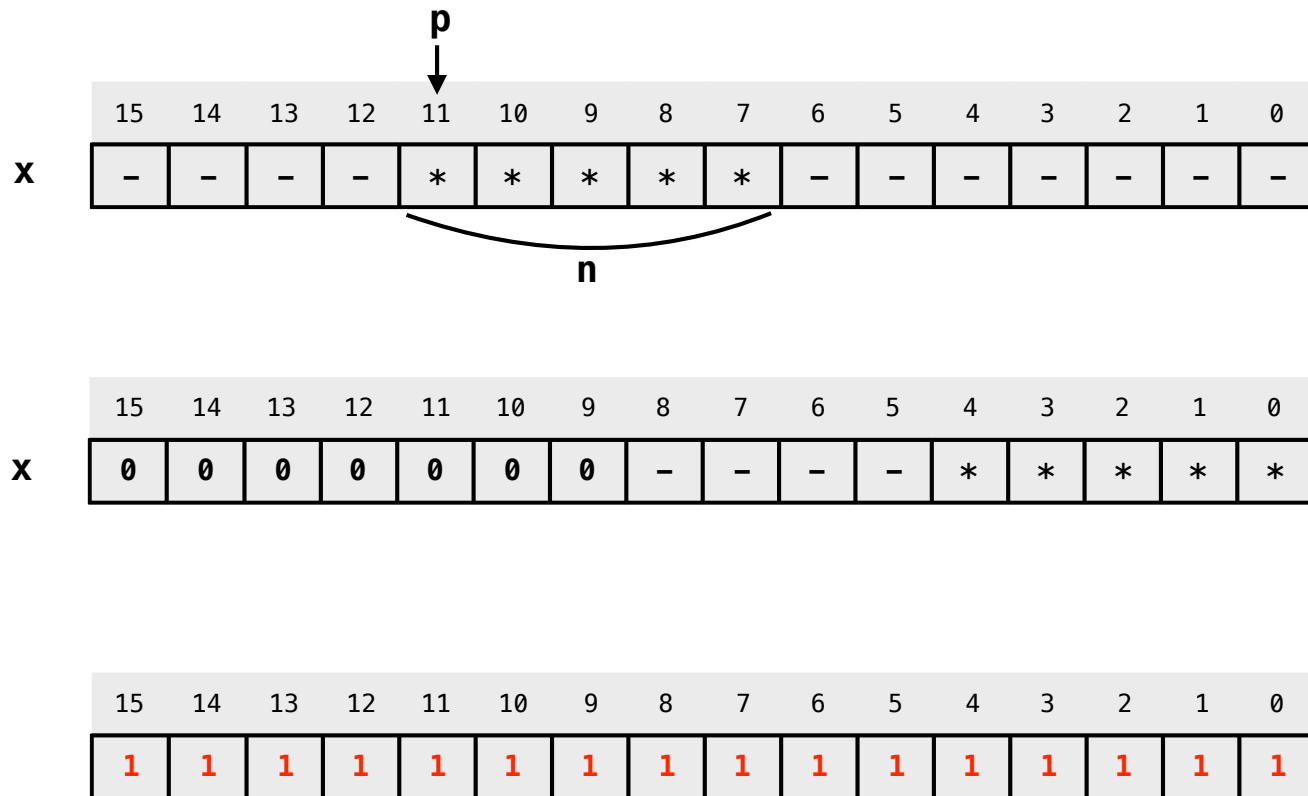
```
/* getbits: get n bits from position p */  
unsigned getbits(unsigned x, int p, int n) {  
    return (x >> (p+1-n)) & ~(~0 << n);  
}
```



# 비트조작 연산자

## Bitwise Operators

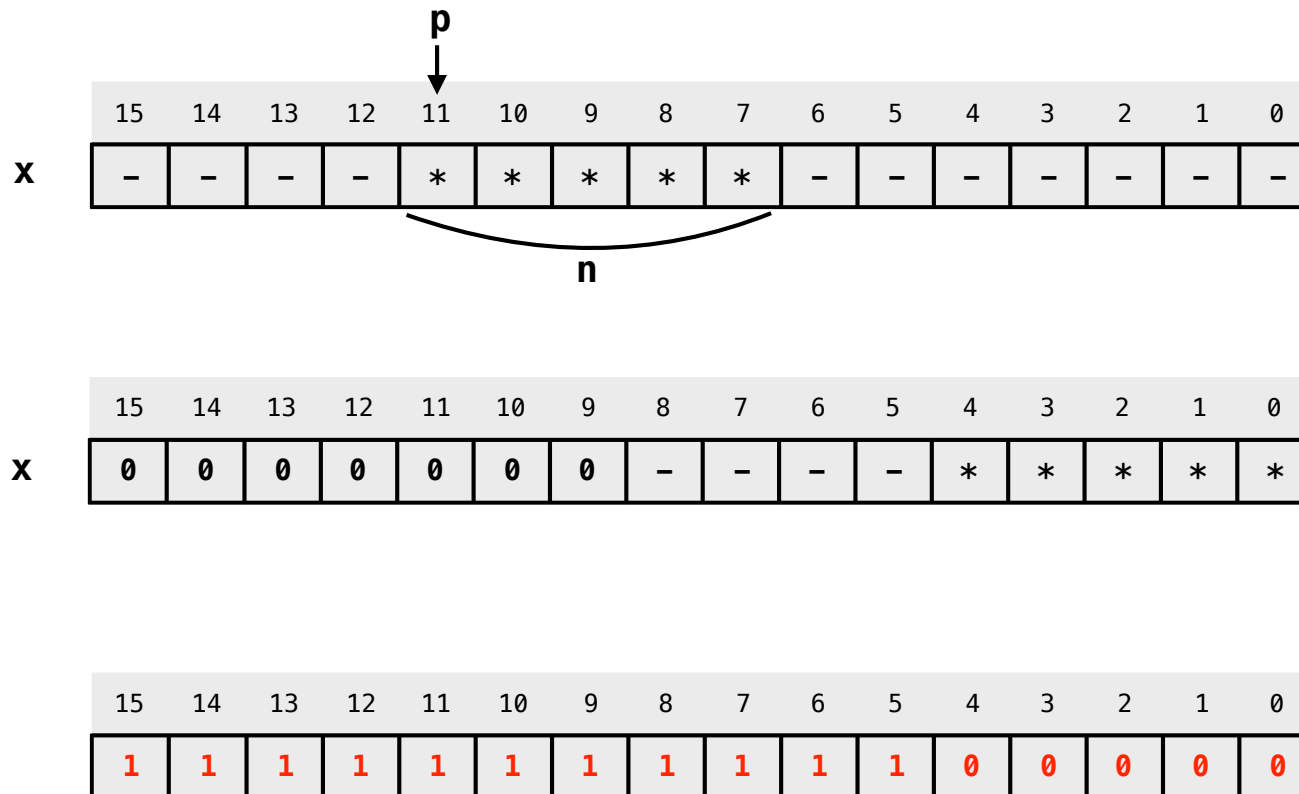
```
/* getbits: get n bits from position p */  
unsigned getbits(unsigned x, int p, int n) {  
    return (x >> (p+1-n)) & ~(~0 << n);  
}
```



# 비트조작 연산자

## Bitwise Operators

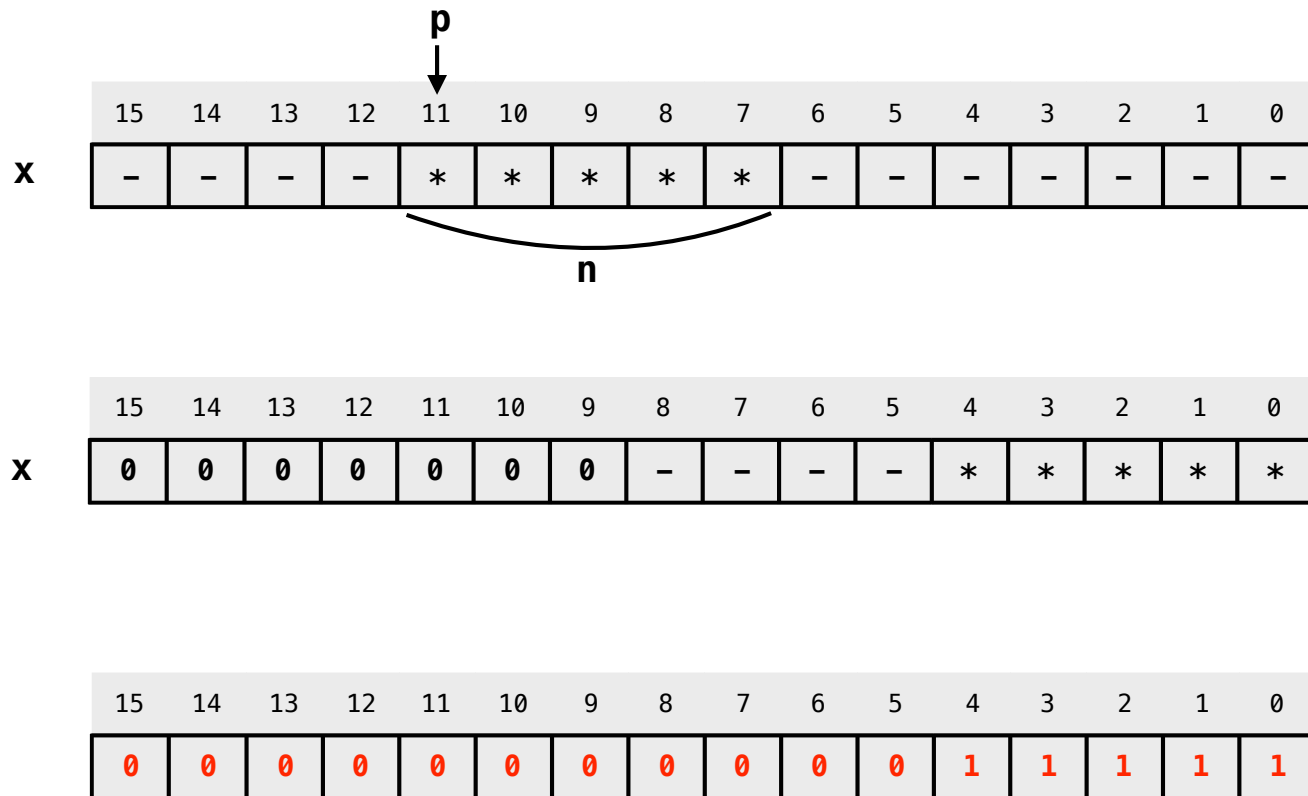
```
/* getbits: get n bits from position p */  
unsigned getbits(unsigned x, int p, int n) {  
    return (x >> (p+1-n)) & ~(~0 << n);  
}
```



# 비트조작 연산자

## Bitwise Operators

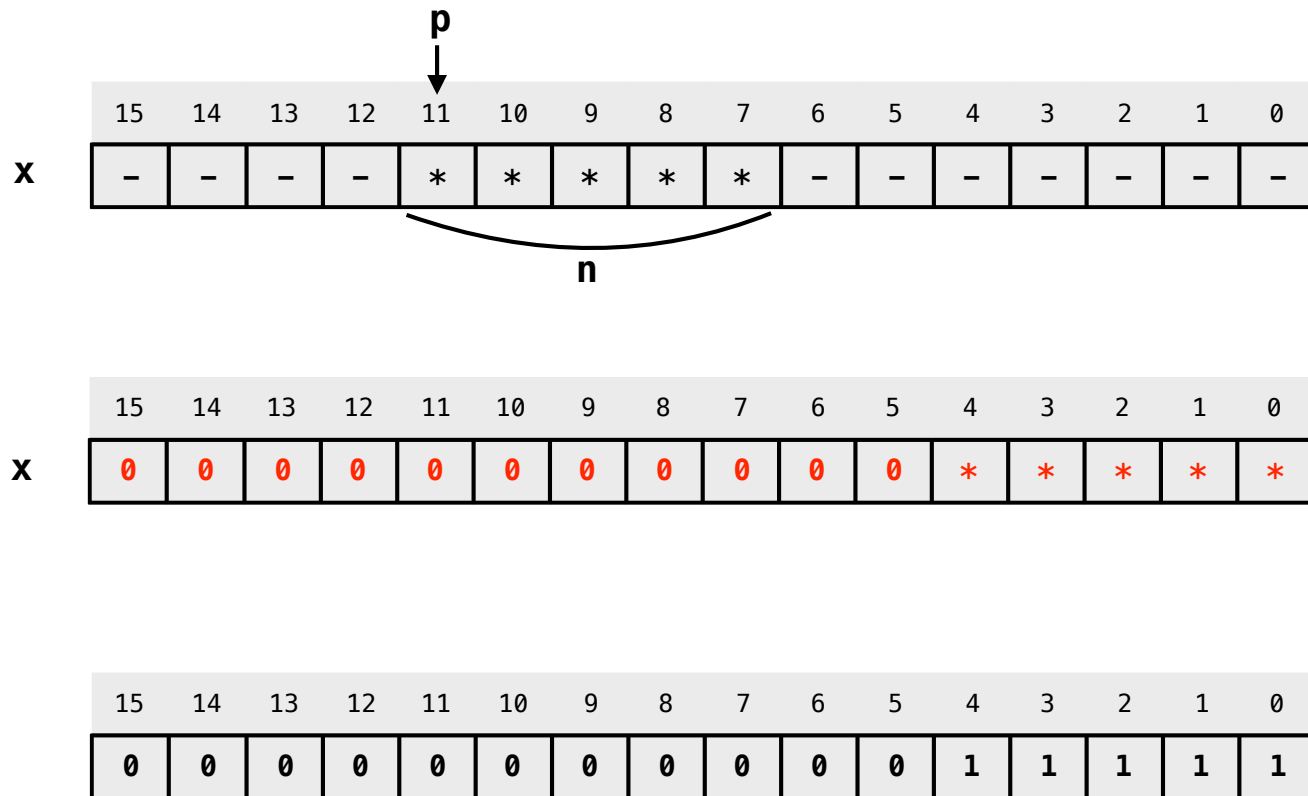
```
/* getbits: get n bits from position p */
unsigned getbits(unsigned x, int p, int n) {
    return (x >> (p+1-n)) & ~(~0 << n);
}
```



# 비트조작 연산자

## Bitwise Operators

```
/* getbits: get n bits from position p */  
unsigned getbits(unsigned x, int p, int n) {  
    return (x >> (p+1-n)) & ~(~0 << n);  
}
```





# 지정연산자 및 지정식

## Assignment Operators & Expressions

*op*

*expr<sub>1</sub> op = expr<sub>2</sub>*

+= <<=  
 -= >>=  
 \*= &=  
 /= ^=  
 %= |=



*expr<sub>1</sub> = (expr<sub>1</sub>) op (expr<sub>2</sub>)*

**i += 2**



**i = i + 2**

**x \*= y + 1**



**x = x \* (y + 1)**

~~**x = x \* y + 1**~~

## 지정연산자 및 지정식

### Assignment Operators & Expressions

```
/* bit count: count 1 bits in x */  
int bitcount(unsigned x) {  
    int b;  
  
    for (b = 0; x != 0; x >>= 1)  
        if (x & 01)  
            b++;  
    return b;  
}
```

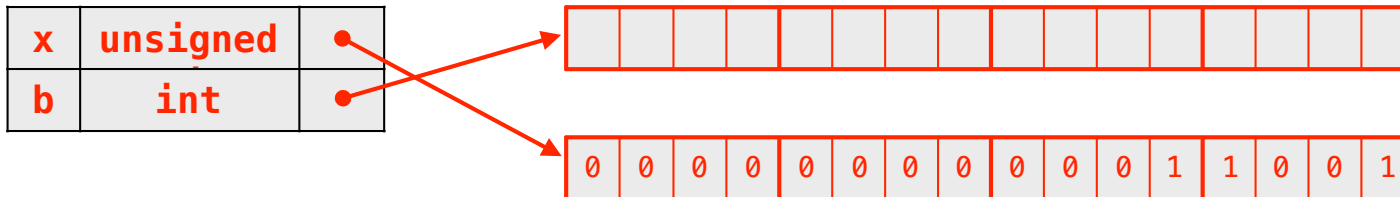
→ `printf(bitcount(25));`

## 지정연산자 및 지정식 Assignment Operators & Expressions

→

```
/* bit count: count 1 bits in x */  
int bitcount(unsigned x) {  
    int b;  
  
    for (b = 0; x != 0; x >>= 1)  
        if (x & 01)  
            b++;  
    return b;  
}
```

`printf(bitcount(25));`



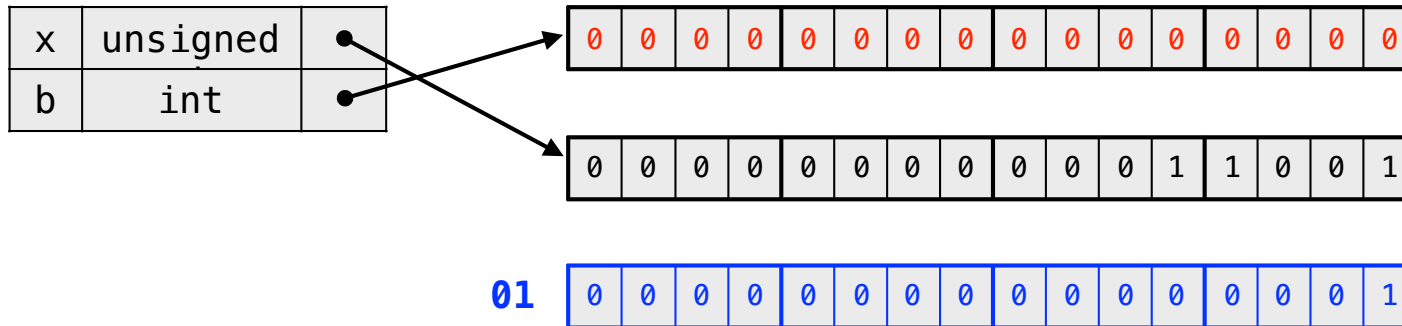
# 지정연산자 및 지정식

## Assignment Operators & Expressions

```
/* bit count: count 1 bits in x */
int bitcount(unsigned x) {
    int b;

    for (b = 0; x != 0; x >>= 1)
        if (x & 01)
            b++;
    return b;
}
```

`printf(bitcount(25));`



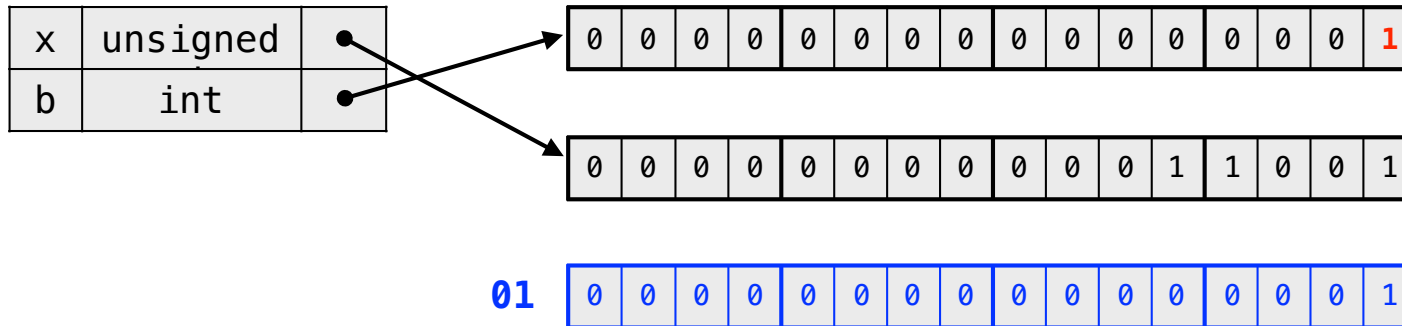
# 지정연산자 및 지정식

## Assignment Operators & Expressions

```
/* bit count: count 1 bits in x */
int bitcount(unsigned x) {
    int b;

    for (b = 0; x != 0; x >>= 1)
        if (x & 01)
            b++;
    return b;
}
```

`printf(bitcount(25));`



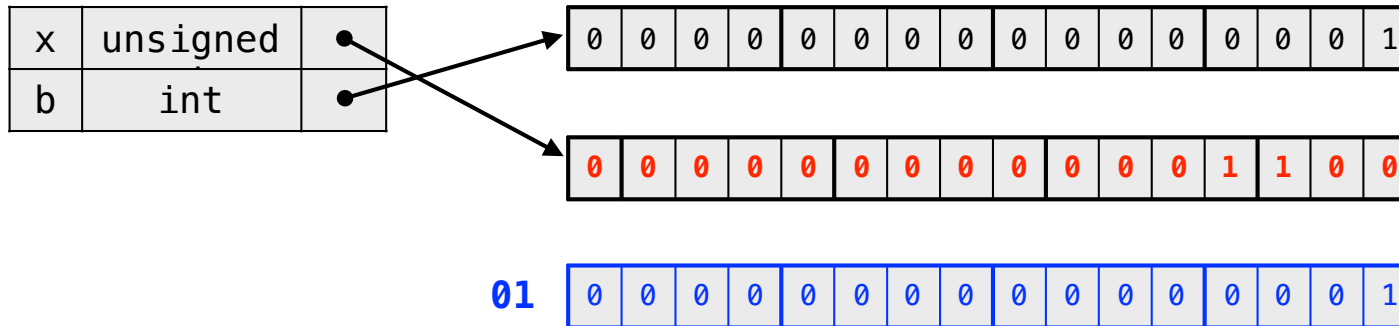
# 지정연산자 및 지정식

## Assignment Operators & Expressions

```
/* bit count: count 1 bits in x */
int bitcount(unsigned x) {
    int b;

    for (b = 0; x != 0; x >>= 1)
        if (x & 01)
            b++;
    return b;
}
```

`printf(bitcount(25));`



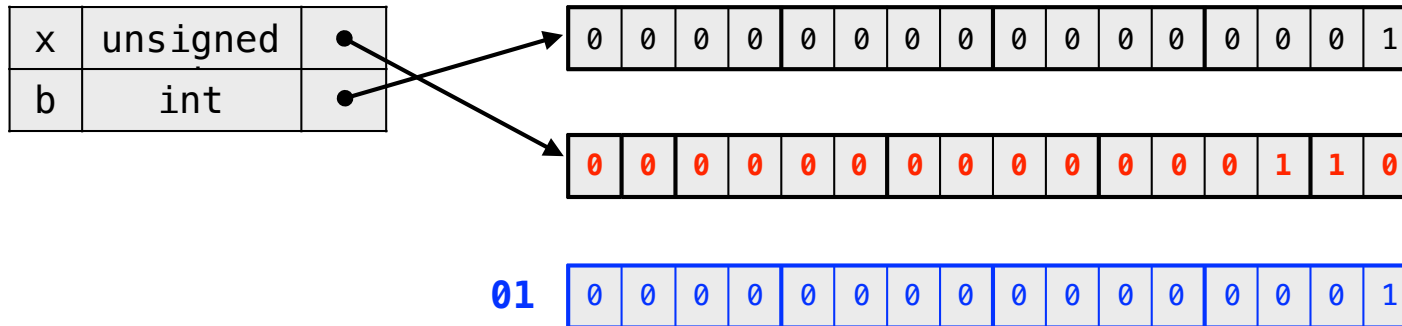
# 지정연산자 및 지정식

## Assignment Operators & Expressions

```
/* bit count: count 1 bits in x */
int bitcount(unsigned x) {
    int b;

    for (b = 0; x != 0; x >>= 1)
        if (x & 01)
            b++;
    return b;
}
```

`printf(bitcount(25));`



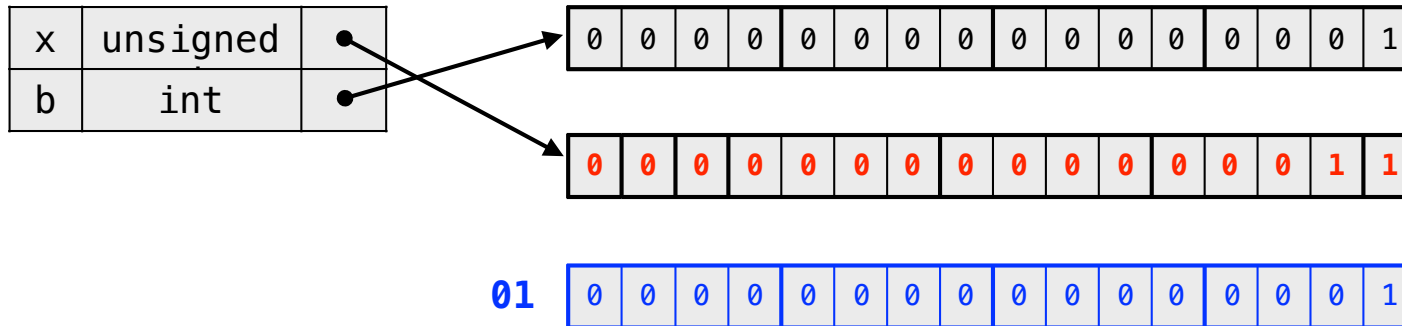
# 지정연산자 및 지정식

## Assignment Operators & Expressions

```
/* bit count: count 1 bits in x */
int bitcount(unsigned x) {
    int b;

    for (b = 0; x != 0; x >>= 1)
        if (x & 01)
            b++;
    return b;
}
```

`printf(bitcount(25));`





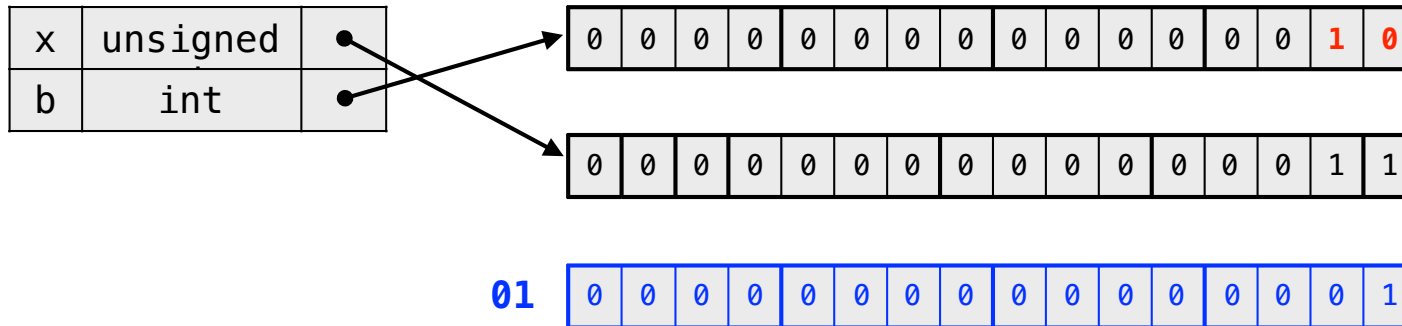
# 지정연산자 및 지정식

## Assignment Operators & Expressions

```
/* bit count: count 1 bits in x */
int bitcount(unsigned x) {
    int b;

    for (b = 0; x != 0; x >>= 1)
        if (x & 01)
            b++;
    return b;
}
```

`printf(bitcount(25));`



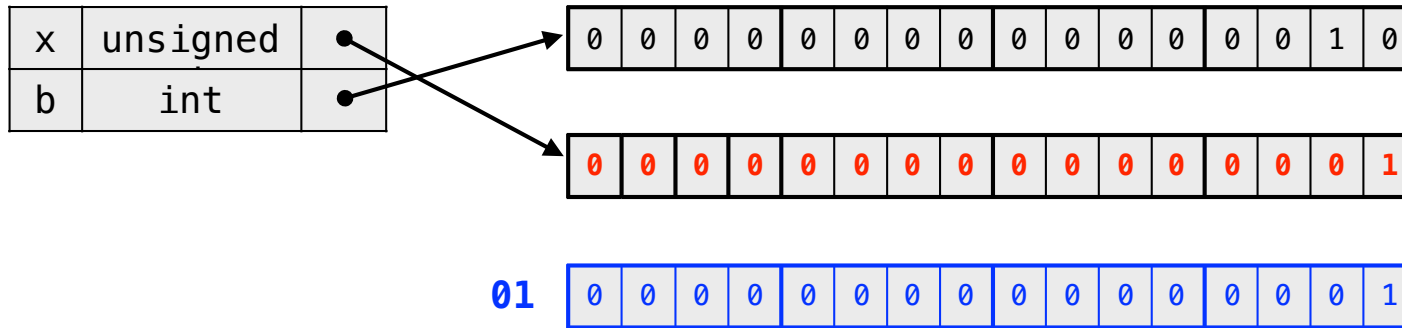
# 지정연산자 및 지정식

## Assignment Operators & Expressions

```
/* bit count: count 1 bits in x */
int bitcount(unsigned x) {
    int b;

    for (b = 0; x != 0; x >>= 1)
        if (x & 01)
            b++;
    return b;
}
```

`printf(bitcount(25));`



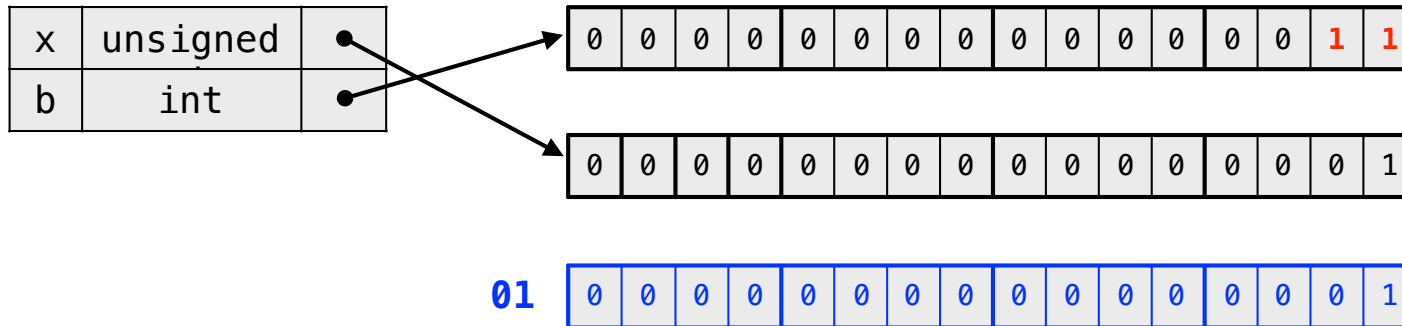
# 지정연산자 및 지정식

## Assignment Operators & Expressions

```
/* bit count: count 1 bits in x */
int bitcount(unsigned x) {
    int b;

    for (b = 0; x != 0; x >>= 1)
        if (x & 01)
            b++;
    return b;
}
```

`printf(bitcount(25));`



# 지정연산자 및 지정식

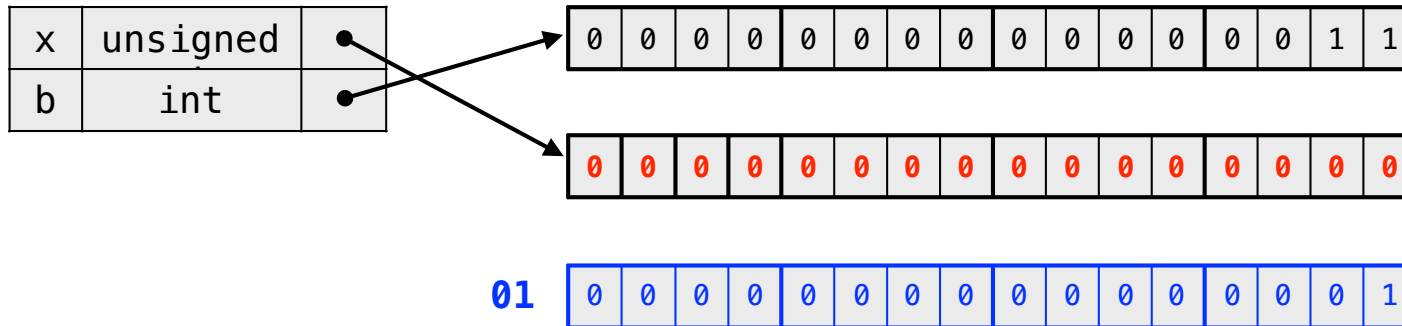
## Assignment Operators & Expressions

```
/* bit count: count 1 bits in x */
int bitcount(unsigned x) {
    int b;

    for (b = 0; x != 0; x >>= 1)
        if (x & 01)
            b++;

    return b;
}
```

`printf(bitcount(25));`



## 지정연산자 및 지정식

### Assignment Operators & Expressions

```

/* bit count: count 1 bits in x */
int bitcount(unsigned x) {
    int b;

    for (b = 0; x != 0; x >>= 1)
        if (x & 01)
            b++;
    return b;
}

```

→ `printf(bitcount(25));`

[illegible]

## 조건식 Conditional Expressions

*expr<sub>1</sub> ? expr<sub>2</sub> : expr<sub>3</sub>*

1

```
/* z = max(a, b) */
```

```
z = (a > b) ? a : b;
```

```
if (a > b)  
    z = a;  
else  
    z = b;
```

2

```
for (i = 0; i < n; i++)  
    printf("%6d%c", a[i], (i%10==9 || i==n-1) ? '\n' : ' ');
```

3

```
printf("You have %d item%s.\n", n, n==1 ? "" : "s");
```

# 연산자 우선순위와 결합순서

## Precedence and Associativity of Operators

연산자	결합순서
( ) [ ] -> .	left to right
! ~ ++ - + - * & (type) sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
	left to right
&&	left to right
	left to right
?:	right to left
= += -= *= /= %= &= ^=  = <<= >>=	right to left
,	left to right

- +, -, \* 단항연산자는 동일 이항연산자보다 우선순위가 높음  
unary operator    binary operator