<3장> 전처리기, 연산자, 표준 입출력 함수

학습 목표

- C 언어 전처리기의 기능 중 #include와 #define을 이해한다.
- C 언어가 제공하는 다양한 연산자들을 학습한다.
- printf()를 제외한 입출력 함수들을 사용한다.

목차

01

전처리기

1. 전처리기

- C 컴파일러의 전처리기
  - 소스 코드의 특정 문자열을 다른 문자열로 치환
  - 다른 파일 내용 삽입
  - 선택적 코드 삭제의 조건부 컴파일 기능
  - 궁극적으로 전처리기는 기존 소스 코드를 수정해서 컴파일러에 전달할 새로운 소스 코드를 생성

- 전처리기 지시자
  - 전처리기에서 인식하고 가공하기 위해 사용하는 키워드
  - # 기호로 시작

표 3-1 전처리기 기능 및 지시자

기능	기능 관련 지시자
매크로 전개(macro expansion)	#define, # 연산자 또는 ## 연산자
외부 파일 삽입(file inclusion)	#include
조건부 컴파일(conditional compilation)	#if, #ifdef, #ifndef, #elif, #endif, defined 연산자
기타	#error, #line, #undef, #pragma

1. 전처리기

전처리기 코드 작성 방법

- 공백 문자를 제외하고 '#' 으로 처음 시작하는 줄(line)은 전처리기 코드로 인식
- '#' 기호와 전처리기 지시자 사이에 공백 문자들이 포함됨
- 전처리기 코드는 일반적으로 한 줄에 끝남
- 여러 줄에 걸쳐서 전처리기 코드를 작성할 경우 줄 마지막에 '₩'를 작은 따옴표 없이 붙여 사용

1. 전처리기

전처리기 코드 작성 방법

- #include <stdio.h> 예시

```
#include <stdio.h>        // 띄어쓰기 없이 줄의 처음부터 시작
    #include <stdio.h>    // 띄어쓰기 있고 줄의 중간부터 시작
```

- '#' 기호 다음 공백

```
#include <stdio.h>
#   include <stdio.h>   // #과 include 사이에 공백이 있어도 무시되지만, 권하는 방법은 아님
```

- 여러 줄에 걸친 전처리기 코드

```
#define printErrMsg \
    printf("Error"); \
    printf(" Message");
```

1. 전처리기

외부 파일 삽입 기능 - #include

- < > 또는 " " 사이에 입력된 외부 파일을 열고 내용을 삽입하라는 코드

```
#include 〈외부_파일_이름〉  // C 언어의 헤더 파일
#include "외부_파일_이름"  // 프로그래머의 헤더 파일
```

- <...>와 "..."의 차이
  - 현재는 <...>는 컴파일러가 제공하는 파일, "..."는 본인이 작성한 파일로 가정

코드 3-1 Include.c

```
1   #include "src.txt"
```

코드 3-2 src.txt

```
1   #include 〈stdio.h〉
2
3   int main(void)
4   {
5       printf("included src.txt\n");
6       return 0;
7   }
```

1. 전처리기

외부 파일 삽입 기능 - #include

- 코드 3-1, 3-2 전처리가 실행된 후 생성되는 임시 코드

```
#include <stdio.h>

int main(void)
{
    printf("included src.txt\n");
    return 0;
}
```

- Include.c 파일을 컴파일하고 실행한 결과

```
<실행 결과>

included src.txt
```

1. 전처리기

매크로 확장 기능 - #define

- 해당 문자열을 다른 문자열로 치환하는 기능
- 매크로 상수와 매크로 함수로 나뉨

매크로 상수

- 단순하게 특정 문자열을 다른 문자열로 치환
- 매크로를 상수(숫자, 문자, 문자열 등)로 변환

```
#define 매크로_이름   [치환할_값]
```

1. 전처리기

매크로 확장 기능 - #define

매크로 상수 - 주의점

- **매크로_이름도** 식별자이므로 **영문자나 밑줄 문자로 시작**
- **두 번째 글자부터는 영문자, 숫자, 밑줄 문자만 사용 가능(공백 문자 포함X)**
- 영문자는 보통 대문자로 표현
- **매크로_이름과 치환할_값 사이에 공백 또는 탭으로 분리**
- 치환할_값은 생략 가능
- 치환할_값은 공백 문자 포함이 가능
- 치환할_값에는 또 다른 전처리기 지시자 사용 불가
- 치환할_값이 산술 연산식이라면 ( )로 감싸주어야 함
- 문자 상수나 문자열 상수에 있는 매크로_이름이 있다면 변환하지 않음
- #define 이후에 #include되는 파일은 해당 파일에서 매크로_이름을 치환할_값으로 변환

1. 전처리기

매크로 확장 기능 - #define

매크로 상수

- 전처리기를 이용해서 포함되는 src2.txt의 MSG 매크로 상수는 문자열 상수 "included src2.txt"로 치환

코드 3-3 Include2.c

```
1   #define MSG "included src2.txt"
2
3   #include "src2.txt"
```

1. 전처리기

매크로 확장 기능 - #define

매크로 상수

- 'src2.txt' 파일 내용

```
#include <stdio.h>

int main(void)
{
    printf(MSG);
    printf("MSG");
    return 0;
}
```

MSG는 #define의 "included src2.txt"로 대체됨. printf("included src2.txt");가 됨

MSG가 문자열 상수 안에 있으므로 변환되지 않음. printf("MSG");가 됨

<실행 결과>

```
included src2.txt
MSG
```

1. 전처리기

매크로 확장 기능 - #define

매크로 상수

- 동일한 매크로가 두 개 이상 선언되었을 때

코드 3-4 SameMacros.c

```c
#include <stdio.h>

#define A "32"

int main(void)
{
    printf("A = %s\n", A);

 #define A 64
    printf("A = %d\n", A);
 }
```

<실행 결과>

```
A = 32
A = 64
```

1. 전처리기

매크로 확장 기능 - #define

매크로 상수

- 매크로 상수의 이름을 제대로 사용한 예시

```
#define ERR  -1
#define MSG   "message"
#define PRINT_HELLO2 printf("hello\n"); printf("hi again\n");
#define NEW_LINE     '\n'
```

- 매크로 상수의 이름을 잘못 사용한 예시

```
#define 1ERR -1                      // 숫자로 시작
#define MSG NAME "message"           // MSG와 NAME 사이에 공백 존재
#define "ERR" "Error"                // 매크로_이름은 특수 기호로 시작 불가
#define ERR #define ERROR "Error"   // 같은 줄에 지시자는 두 개 이상 사용 불가
```

1. 전처리기

매크로 확장 기능 - #define

매크로 상수

- 치환할 값이 산술 연산식에 ()를 사용하지 않았을 때(오류 발생 가능)

```
#define SUM n1 + n2    // n1과 n2의 덧셈 연산으로 치환

int n1 = 2;
int n2 = 4;
int result = SUM * 2;  // n1 + n2 * 2로 치환됨. 원하는 (n1 + n2) * 2가 아님
```

- 치환할 값이 산술 연산식에 ()를 사용했을 때

```
#define SUM (n1 + n2)
```

- 치환할_값없이 사용했을 때

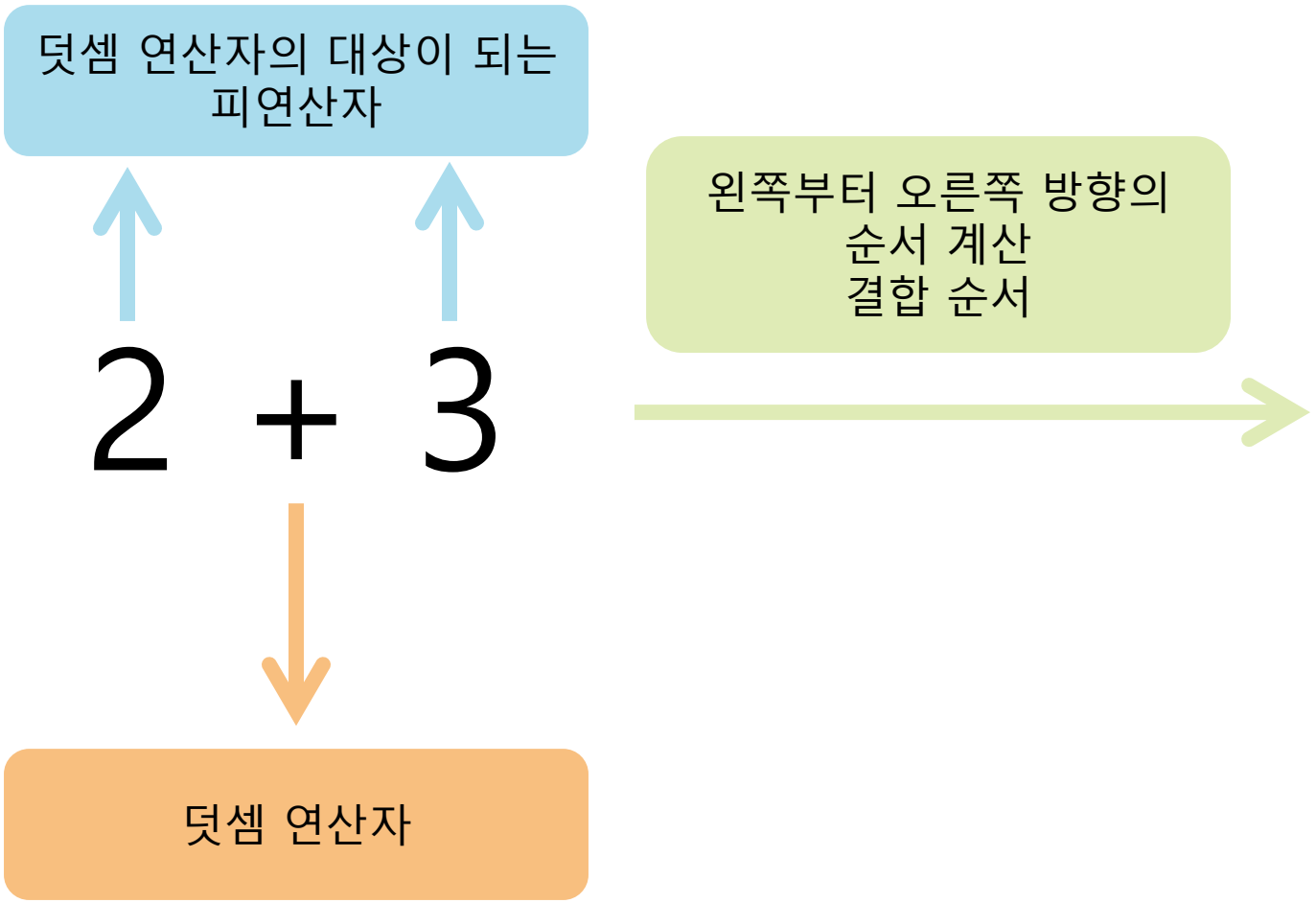
```
#define NO_VALUE
```

02

연산자

2. 연산자

■ 피연산자, 우선순위, 결합 순서



2. 연산자

■ 피연산자, 우선순위, 결합 순서

표 3-2 연산자 우선순위

순위	연산자	설명	결합 순서	책에서 다루는 장
1	( )	괄호(수학 연산에 쓰이는 괄호의 개념)	왼쪽 → 오른쪽	3장
1	[]	배열 인덱스	왼쪽 → 오른쪽	8장
1	.	구조체(공용체) 멤버 직접 선택 연산자	왼쪽 → 오른쪽	10장
1	−〉	구조체(공용체) 멤버 간접 선택 연산자	왼쪽 → 오른쪽	10장
2	++ −−	증감 연산자(후위 연산)	왼쪽 → 오른쪽	3장
3	++ −−	증감 연산자(전위 연산)	오른쪽 → 왼쪽	3장
3	+ −	부호 연산자	오른쪽 → 왼쪽	3장
3	! ~	논리(!)/비트(~) NOT 연산자	오른쪽 → 왼쪽	3장
3	*	간접 참조 연산자	오른쪽 → 왼쪽	8장
3	&	주소 연산자	오른쪽 → 왼쪽	2장
3	sizeof	sizeof 연산자	오른쪽 → 왼쪽	2장
4	(자료형)	형 변환 연산자	오른쪽 → 왼쪽	2장

2. 연산자

■ 피연산자, 우선순위, 결합 순서

5	* / %	산술 연산자	왼쪽 → 오른쪽	3장
6	+ −	산술 연산자	왼쪽 → 오른쪽	3장
7	〈〈 〉〉	비트 이동 연산자	왼쪽 → 오른쪽	3장
8	〈 〈= 〉 〉=	관계 연산자	왼쪽 → 오른쪽	3장
9	== !=	관계 연산자	왼쪽 → 오른쪽	3장
10	&	비트 AND 연산자	왼쪽 → 오른쪽	3장
11	^	비트 XOR	왼쪽 → 오른쪽	3장
12	|	비트 OR 연산자	왼쪽 → 오른쪽	3장
13	&&	논리 AND 연산자	왼쪽 → 오른쪽	3장
14	||	논리 OR 연산자	왼쪽 → 오른쪽	3장
15	? :	조건 연산자	오른쪽 → 왼쪽	4장
16	= += −= *= /= %= 〈〈= 〉〉= &= ^= |=	대입 연산자(=), 복합 대입 연산자	오른쪽 → 왼쪽	3장
17	,	콤마 연산자	왼쪽 → 오른쪽	5장

2. 연산자

부호 연산자

- 단항 연산자이고, 표현식의 피연산자에 부호를 붙임

```
+표현식   -표현식
```

- 변수에 부호 연산자가 붙을 때

```
+3      +n      -n      40 + -n  // 3, n의 값, n * -1의 값, 40 - n의 값으로 계산됨
```

2. 연산자

산술 연산자

- 덧셈, 뺄셈, 곱셈, 나눗셈, 나머지 연산자

표 3-3 산술 연산자, 설명, 예시

연산자	피연산자 종류	설명	예시
+	정수, 실수	덧셈	2 + 3, 2.3 + 3.2f, n + 3
-	정수, 실수	뺄셈	2 - 3, 2.3 - 3.2, n - 2
*	정수, 실수	곱셈. 곱셈 기호가 알파벳 x와 혼동될 수 있어 *를 곱셈 기호로 사용한다. 포인터를 참조하는 간접 참조 연산자인 단항 연산자인 *와 구별해야 한다.	2 * 3, 3 - 2 * 4
/	정수, 실수	나눗셈. 왼쪽 피연산자를 오른쪽 피연산자로 나눈다. 피연산자 두 개 모두 정수이면 정수의 나눗셈을 실행(몫 반환)한다. 피연산자 중 한 개 이상이 실수이면 실수의 나눗셈을 실행한다.	3 / 2 (결과 1) 1 / 2 (결과 0) 3.0 / 2 (결과 1.5)
%	정수	나머지 연산. 왼쪽 피연산자를 오른쪽 피연산자로 나눈 나머지 값을 반환한다. 두 개 피연산자는 모두 정수형이어야 한다. 정수형이 아니면 컴파일 오류가 발생한다.	3 % 2 (결과 1) 1 % 2 (결과 1)

※ 산술 연산자 사용 코드

➔ 코드 3-5
➔ 실행 결과

2. 연산자

비트 연산자

- 메모리에 비트 단위로 저장되어 있는 값을 조작하거나 연산을 실행
- 비트 이동(bit shift) 연산자/비트 연산자(bitwise operation)로 나뉨

표 3-4 비트 이동 연산자

연산자	피연산자 종류	설명	예시
《	정수	n 《 b, n의 비트를 왼쪽으로 b개만큼 이동시키고 오른쪽 빈칸은 0으로 채운다.	2 《 3, n1 《 2
》	정수	n 》 b, n의 비트를 오른쪽으로 b개만큼 이동시키고 왼쪽 빈칸을 0으로 채운다.	2 》 3, n1 》 2

2. 연산자

■ 비트 연산자

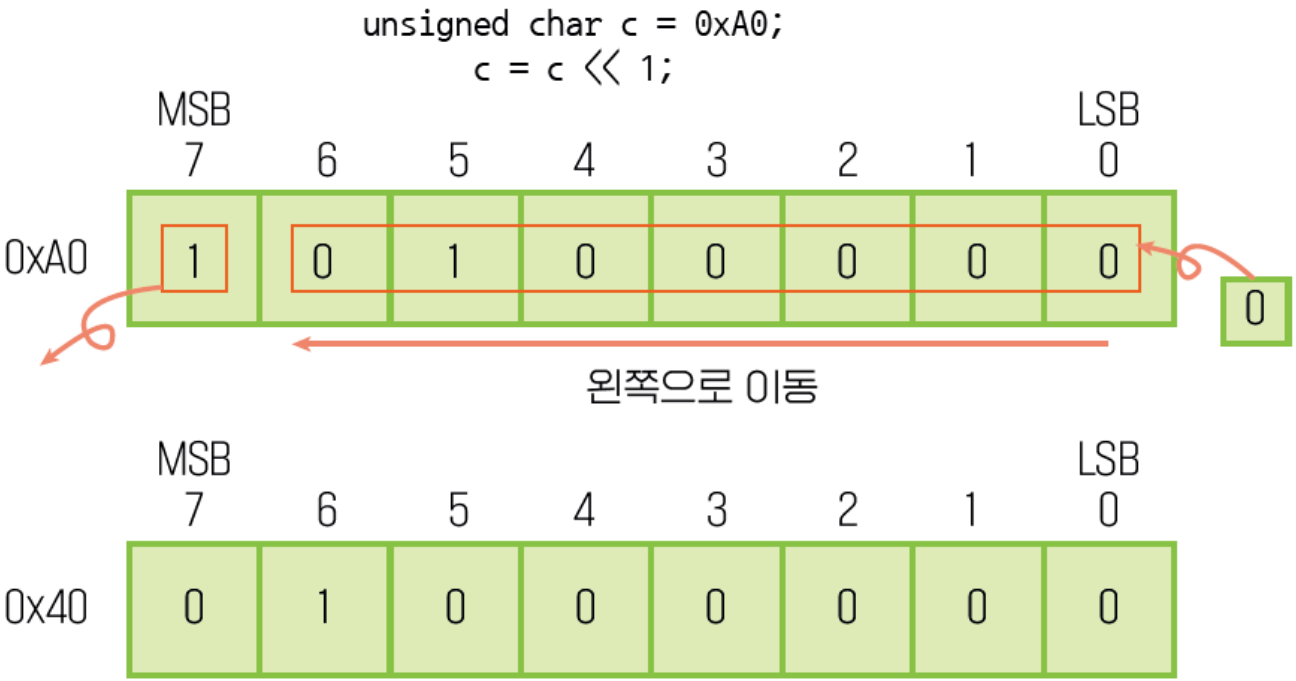


그림 3-1 0xA0을 왼쪽으로 1비트 이동

2. 연산자

■ 비트 연산자

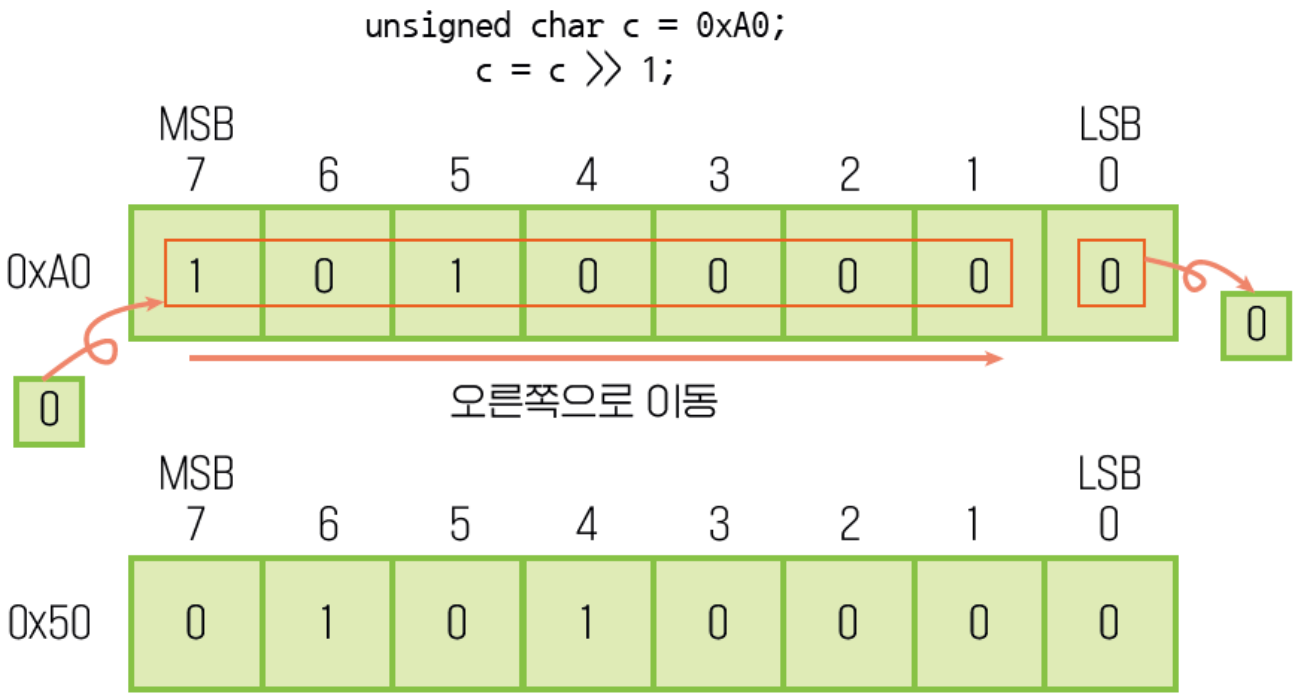


그림 3-2 0xA0을 오른쪽으로 1비트 이동

※ 비트 이동 코드

➔ 코드 3-6
➔ 실행 결과

2. 연산자

산술 연산자

- 비트 연산자

표 3-5 비트 연산자(AND(&), OR(|), XOR(^), NOT(~))

b1	b2	b1 & b2(비트 AND)	b1 | b2(비트 OR)	b1 ^ b2(비트 XOR)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

※ AND, OR, XOR, NOT 연산을 하는 코드

➔ 코드 3-7
➔ 실행 결과

2. 연산자

증감 연산자

- 정수 변수와 함께 사용
- 변숫값을 1만큼 증가시키거나 감소
- 변수 앞 또는 뒤에 위치

```
int n = 3;
n++;          // n = n + 1; n에 1을 더해서 4가 됨
n--;          // n = n - 1; n에서 1을 빼서 3이 됨
++n;          // n = n + 1; n에 1을 더해 4가 됨
--n;          // n = n - 1; n에서 1을 빼서 3이 됨
```

증감 연산자

증감 연산자의 위치

- 증감 연산자가 뒤에 있을 때

```
result = n + 4;
n++;
```

- 증감 연산자가 앞에 있을 때

```
++n;
result = n + 4;
```

※ 증감 연산자 코드 - 결과 예측해 보기

➔ 코드 3-8
➔ 실행 결과

2. 연산자

복합 대입 연산자

- 연산과 대입을 한 번에 쓰는 것
- op는 이항 연산자(+, -, /, *, %, <<, >>, &, ^, |) 중 한 가지 사용 가능

```
변수 op = 표현식
```

- 복합 대입 연산자는 다음 코드를 줄여 작성하는 방법

```
변수 = 변수 op 표현식
```

2. 연산자

복합 대입 연산자

- 예시 1

```
int n = 3;
n += 4;      // n = n + 4
```

- 예시 2

```
n += 4 + 4;
```

- 예시 3

```
n2 = n3;
n1 += n2;
```

2. 연산자

순차 연산자

- 콤마(,)로 표시해서 콤마 연산자라고도 함
- for문에서 사용하는 경우를 제외하면, 많이 사용 안됨
- 왼쪽에서 오른쪽으로 순서대로 표현식의 값을 평가(evaluate)
- 순차 연산자의 결괏값은 가장 오른쪽에 위치한 표현식의 결괏값

```
표현식1, 표현식2, 표현식3, ..., 표현식n
```

순차 연산자

- 여러 개 대입 연산 코드를 한 줄로 압축해서 표현

코드 3-9 Comma0p1.c

```c
#include <stdio.h>

int main(void)
{
    int n1, n2, n3;
    n1 = (n2 = 4, n3 = 5); // 괄호가 있어야 함
    printf("n1 = %d, n2 = %d, n3 = %d\n", n1, n2, n3);
    return 0;
}
```

대입 연산자는 오른쪽부터 평가 실행
(n2 = 4, n3 = 5)가 먼저 실행되는데 n2 = 4부터 실행됨
n2에 4가 저장되고, 순차적으로 n3에 5가 저장됨
순차 연산자의 결괏값은 n3 = 5의 결괏값, 즉 n3의 값이 됨

<실행 결과>

```
n1 = 5, n2 = 4, n3 = 5
```

순차 연산자

- 여러 개 대입 연산 코드를 한 줄로 압축해서 표현
  - 코드 3-9의 순차 연산식은 다음 세 줄의 코드를 한 줄로 줄여 작성한 것

```
n2 = 4;
n3 = 5;
n1 = n3;
```

  - 우선순위가 가장 낮으므로, 괄호 없이 사용하면 다른 결과가 출력됨

```
n1 = n2 = 4, n3 = 5;
```

2. 연산자

순차 연산자

- for 반복문에서 두 개 이상의 변수를 사용 (주 사용처)

코드 3-10 CommaOp2.c

```c
#include <stdio.h>

int main(void)
{
    for (int first = 1, second = 9; i < 10; first++, second--) {
        printf("%d%d ", first, second);
    }
    return 0;
}
```

<실행 결과>

```
19 28 37 46 55 64 73 82 91
```

03

표준 입출력 함수

3. 표준 입출력 함수

사용자로부터 입력받기

- 표준 입력 장치의 입력을 C 언어 프로그램에서 처리할 수 있는 자료형의 값으로 변환하는 것
- 표준 입력 장치 : 키보드, 마우스 등
- 표준 출력 장치 : 터미널 프로그램의 콘솔(console) 화면

서식 지정자를 이용해서 입력받기

scanf( ) 함수

- scanf( ) 함수를 사용하려면 stdio.h 헤더 파일을 포함
- C 언어의 표준 입출력 함수들은 모두 stdio.h에 존재
- 사용자가 입력한 문자열을 코드에 지정된 자료형으로 변환해서 변수에 저장
- 자료형을 지정하는 것은 printf( )에서 사용한 비슷한 서식 지정자를 이용

3. 표준 입출력 함수

서식 지정자를 이용해서 입력받기

- scanf() 함수 사용법

코드 3-11

```
int scanf(서식_문자열, 값을_저장할_변수의_주소1, 값을_저장할_변수의_주소2, ..., 값을_저장할_변수의_주소n)
```

- scanf() 함수 사용법 - 간단

코드 3-12

```
int scanf(서식_문자열, &값을_저장할_변수1, &값을_저장할_변수2, ..., &값을_저장할_변수n)
```

3. 표준 입출력 함수

서식 지정자를 이용해서 입력받기

- 정수 한 개를 입력받고 화면에 출력하는 프로그램

코드 3-13 Readint.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3
4   int main(void)
5   {
6       int n;
7       scanf("%d", &n);
8       printf("n = %d\n", n);
9       return 0;
10  }
```

scanf() 함수를 이용해서 정수를 입력받음
printf()에서 사용한 정수 서식형과 동일한 %d를 지정
입력을 저장할 변수 n의 주소를 전달
scanf()가 실행되면 n은 입력한 정숫값을 가짐

<실행 결과>

```
3
n = 3
```

3. 표준 입출력 함수

■ 서식 지정자를 이용해서 입력받기

- scanf() 함수에서 자주 사용하는 서식 지정자를 구성하는 방법

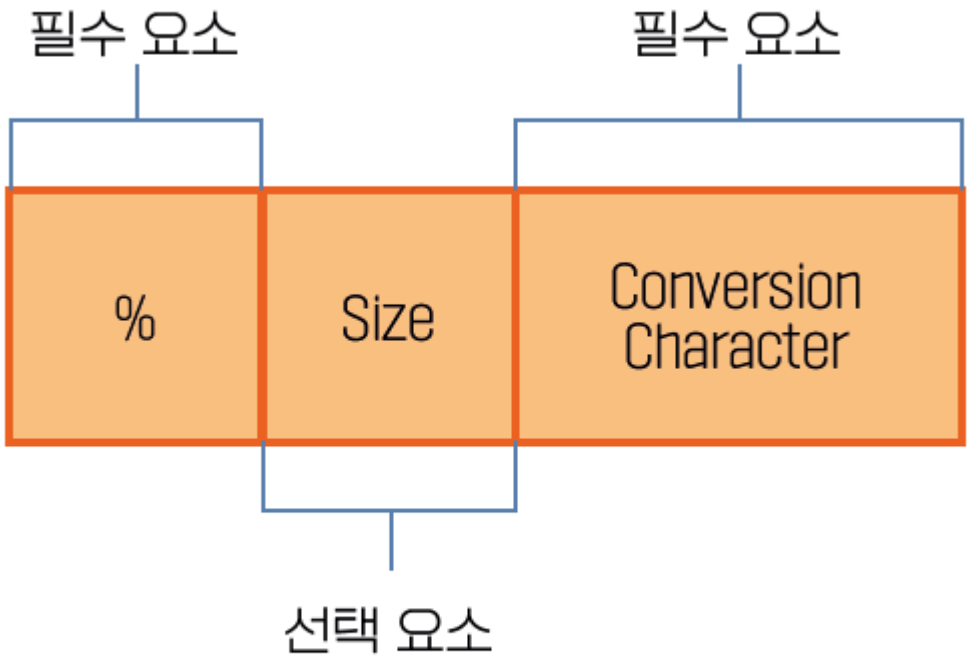


그림 3-3 서식 지정자 구성 요소

3. 표준 입출력 함수

■ 서식 지정자를 이용해서 입력받기

- scanf() 함수에서 자주 사용하는 변환 문자

표 3-6 scanf( )에서 자주 사용하는 서식 지정자

서식 지정자	값의 자료형	설명	입력 예시
%d	int	입력 내용을 부호 있는 10진수 정수로 변환 후 변수에 저장한다.	+23 −23 23
%i	int	입력 내용을 정수로 변환 후 변수에 저장한다. 입력 정수 앞에 0x가 있으면 16진수로 변환하고, 아니면 10진수로 변환한다.	+23, −23, 23, −0x1A, 0x1A
%u	unsigned	입력 내용을 양의 10진수 정수로 변환 후 변수에 저장한다.	23, +23
%x	unsigned	입력 내용을 16진수로 인식해서 양의 정수로 변환 후 변수에 저장한다. 입력할 때 0x는 가능하고 0~9, A~F, a~f를 모두 인식한다. 컴파일러에 따라 %X를 쓸 수 있다.	163f, 0x163f, 163F
%c	char	문자 한 개를 입력받아 변수에 저장한다. %c는 앞에 있는 공백 문자, 탭, 줄바꿈 문자를 제거하지 않는다.	a c 1
%s	char*	공백 문자가 없는 문자열을 입력한다. 〈9장〉에서 학습한다.	hello, world, language
%f, %e, %g	float	%f, %e, %g는 동일하게 동일하게 동작하고, 입력 내용을 실수로 인식해서 변환 후 변수에 저장한다. 정수도 입력 가능하다. 입력할 때 숫자E승수 또는 숫자e승수 형태로 입력하는 것도 가능하다(예 : 2.3E2는 2.3×10^2을 나타낸다). 입력할 때 E(또는 e) 앞 또는 뒤에 값이 없으면 0이 있는 것으로 가정한다. 일부 컴파일러는 %e와 %E를 모두 허용한다.	2.3, 1.5323, 1.3E10, E1, 1.3E

3. 표준 입출력 함수

■ 서식 지정자를 이용해서 입력받기

- size 요소

표 3-7 Size 요소

Size	함께 사용 가능 변환 문자	설명
l	d, i, u, x	long, unsigned long을 입력받을 때 사용한다.
ll	d, i, u, x	long long int, unsigned long long을 입력받을 때 사용한다.
l	f, e, g	double을 입력받을 때 사용한다.
L	f, e, g	long double을 입력받을 때 사용한다.

※ scanf() 함수 코드

➔ 코드 3-14
➔ 실행 결과

3. 표준 입출력 함수

입력 버퍼

- 입력 버퍼 코드

코드 3-15 ReadInts.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3
4   int main(void)
5   {
6       int n1;
7       int n2;
8       printf("정수 한 개를 입력하세요: ");
9       scanf("%d", &n1);
10      printf("정수 한 개를 입력하세요: ");
11      scanf("%d", &n2);
12      printf("n1 = %d, n2 = %d\n", n1, n2);
13      return 0;
14  }
```

3. 표준 입출력 함수

입력 버퍼

- 정수 <Enter> 정수 <Enter> 입력 결과

〈실행 결과〉

```
정수 한 개를 입력하세요: 130 〈Enter〉
정수 한 개를 입력하세요: 150 〈Enter〉
n1 = 130, n2 = 150
```

- 정수 정수 <Enter> 입력 결과

〈실행 결과〉

```
정수 한 개를 입력하세요: 130 150
정수 한 개를 입력하세요: n1 = 130, n2 = 150
```

3. 표준 입출력 함수

입력 버퍼

- 코드 3-15를 실행하고 사용자가 130 엔터를 입력했 때

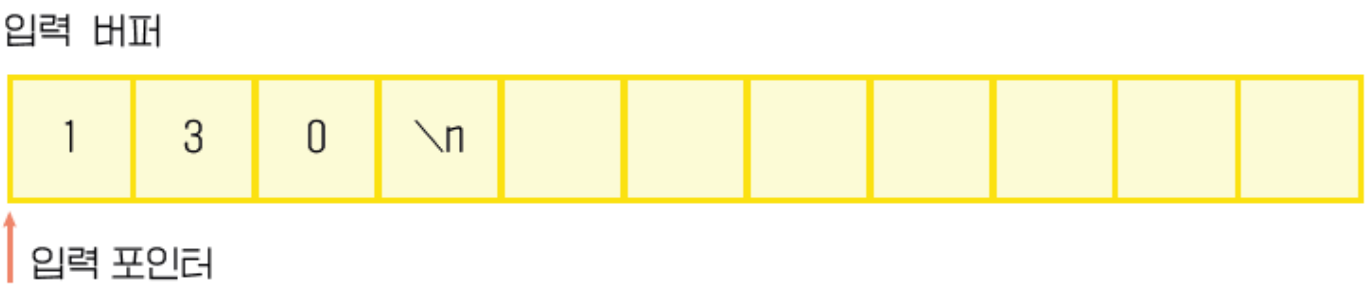


그림 3-4 130 〈Enter〉를 입력했을 때의 입력 버퍼

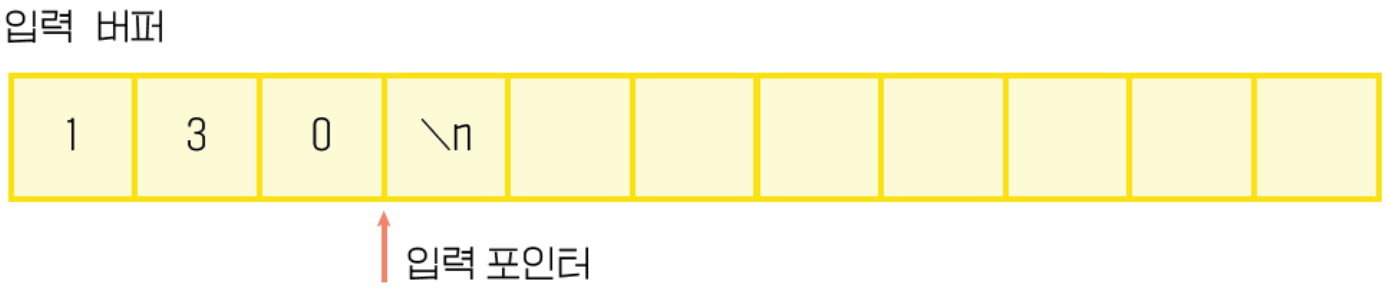


그림 3-5 130을 입력받은 후 입력 버퍼

3. 표준 입출력 함수

입력 버퍼

- 코드 3-15의 두 번째 scanf() 함수 동작

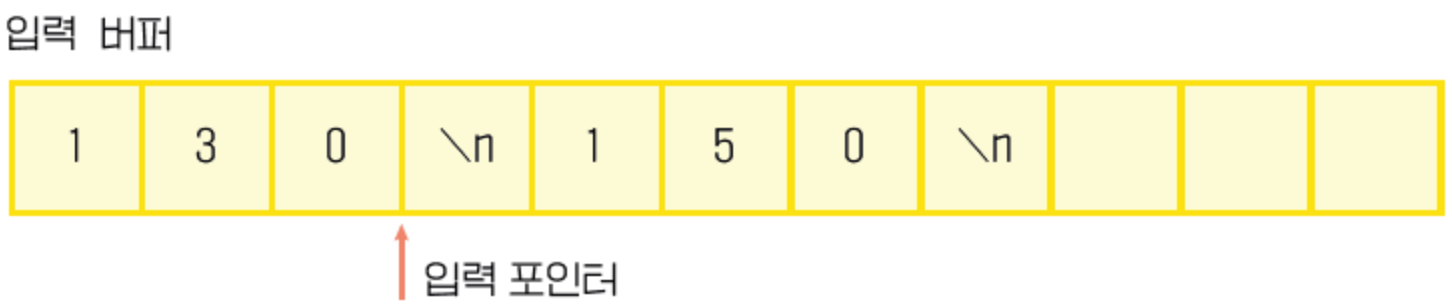


그림 3-6 150 〈Enter〉를 입력한 후 입력 버퍼

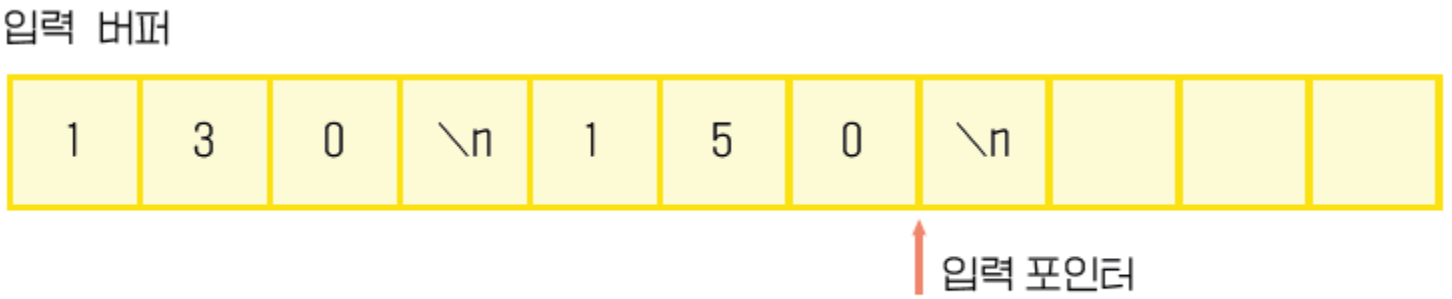


그림 3-7 150을 변수에 저장한 후 입력 버퍼

3. 표준 입출력 함수

입력 버퍼

- 130 150 <Enter> 입력 경우

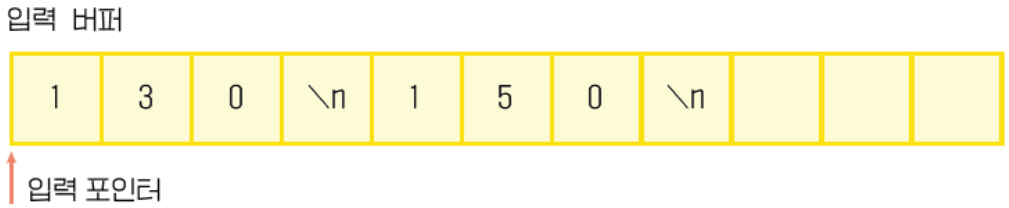


그림 3-8 130 150 〈Enter〉를 입력한 후 입력 버퍼

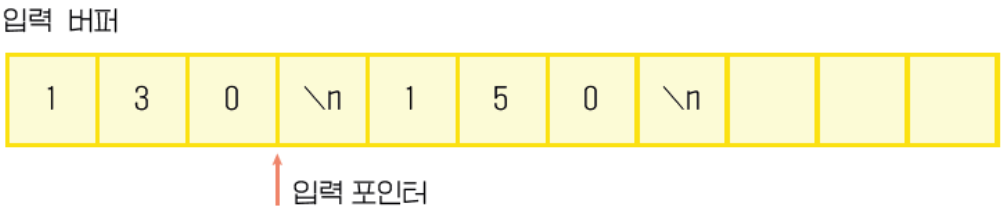


그림 3-9 130이 변수에 저장된 후 입력 버퍼

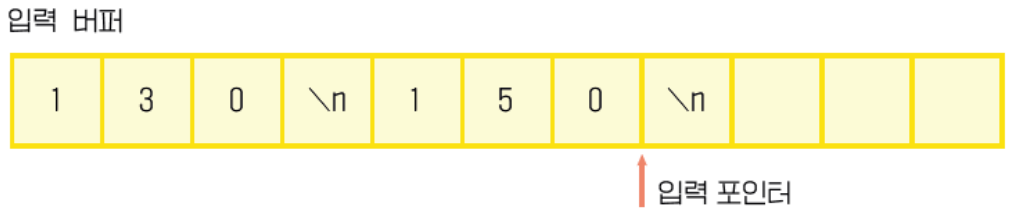


그림 3-10 150이 변수에 저장된 후 입력 버퍼

3. 표준 입출력 함수

입력 버퍼

- 코드 3-14를 실행했을 때 나타났던 현상 다시 분석

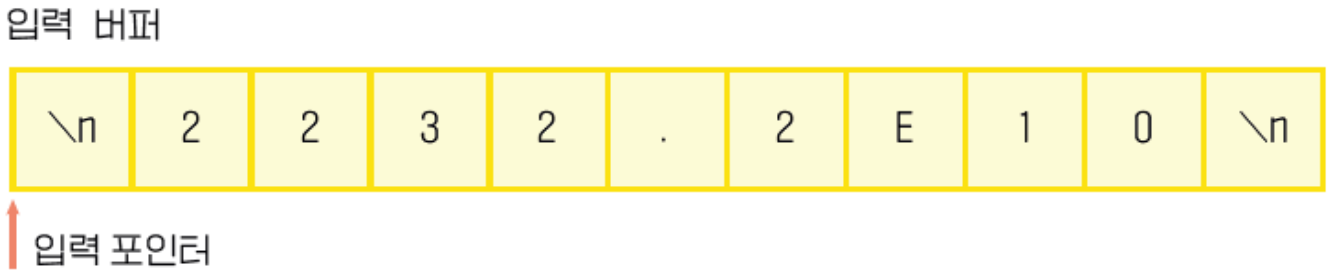


그림 3-11 사용자가 실숫값을 입력한 후의 입력 버퍼

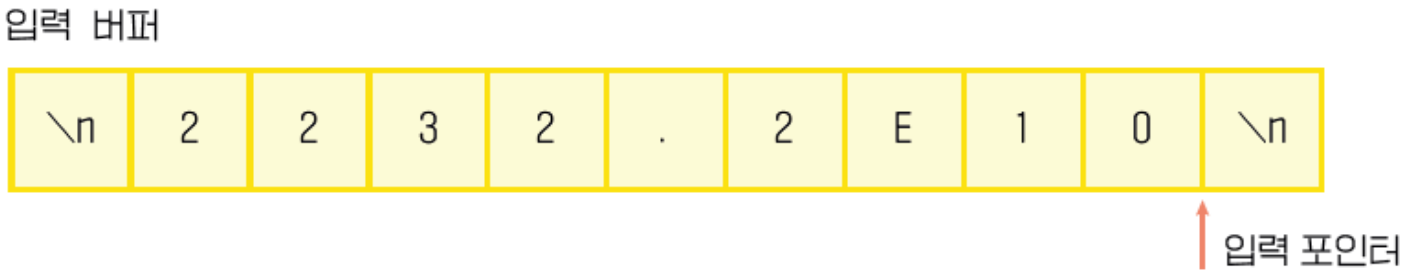


그림 3-12 실수가 변수에 저장된 후 입력 버퍼

3. 표준 입출력 함수

scanf() 함수로 두 개 이상 값을 입력받는 프로그램

- scanf() 함수를 한 번만 호출하는 것으로 코드 3-15 다시 작성

코드 3-16 Readints2.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3
4   int main(void)
5   {
6       int n1;
7       int n2;
8       printf("정수 두 개를 입력하세요: ");
9       scanf("%d %d", &n1, &n2);
10      printf("n1 = %d, n2 = %d\n", n1, n2);
11      return 0;
12  }
```

3. 표준 입출력 함수

scanf() 함수로 두 개 이상 값을 입력받는 프로그램

- 130 150을 입력했을 때

〈실행 결과〉

```
정수 2개를 입력하세요: 130 150
n1 = 130, n2 = 150
```

- 130과 150 사이에 공백을 더 추가해서 입력했을 때

〈실행 결과〉

```
정수 2개를 입력하세요: 130        150
n1 = 130, n2 = 150
```

3. 표준 입출력 함수

"%c"와 " %c"의 차이 - 서식_문자열의 공백 문자

- 정수 두 개와 문자 한 개를 입력받는 코드

코드 3-17 ReadIntsAndChar.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3
4   int main(void)
5   {
6       int n1;
7       int n2;
8       char ch;
9       printf("정수 두 개와 문자 한 개를 입력하세요: ");
10      scanf("%d%d%c", &n1, &n2, &ch);
11      printf("n1 = %d, n2 = %d, ch = %c\n", n1, n2, ch);
12      return 0;
13  }
```

<실행 결과>

```
정수 2개와 문자 한 개를 입력하세요: 130 150 a
n1 = 130, n2 = 150, ch =
```

3. 표준 입출력 함수

"%c"와 " %c"의 차이 - 서식_문자열의 공백 문자

- 정수 두 개와 문자 한 개를 입력받는 코드 - %d%d %₩c로 서식 지정

코드 3-18 ReadIntsAndChar2.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3
4   int main(void)
5   {
6       int n1;
7       int n2;
8       char ch;
9       printf("정수 두 개와 문자 한 개를 입력하세요: ");
10      scanf("%d%d %c", &n1, &n2, &ch);
11      printf("n1 = %d, n2 = %d, ch = %c\n", n1, n2, ch);
12      return 0;
13  }
```

3. 표준 입출력 함수

"%c"와 " %c"의 차이 - 서식_문자열의 공백 문자

- 130 150 a를 입력했을 때

<실행 결과>

```
정수 2개와 문자 한 개를 입력하세요: 130 150 a
n1 = 130, n2 = 150, ch = a
```

- 150과 a 사이에 공백을 더 추가해서 입력했을 때

<실행 결과>

```
정수 2개와 문자 한 개를 입력하세요: 130 150     a
n1 = 130, n2 = 150, ch = a
```

- 130 150 줄바꿈 a를 입력했을 때

<실행 결과>

```
정수 2개와 문자 한 개를 입력하세요: 130 150
a
n1 = 130, n2 = 150, ch = a
```

3. 표준 입출력 함수

서식을 지정해서 값 읽기

- xxx-xxxx-xxxx 형태의 전화번호를 읽어서 세 개의 번호로 분리해서 저장하는 프로그램

코드 3-19 PhoneNum.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3
4   int main(void)
5   {
6       int n1, n2, n3;
7       scanf("%d-%d-%d", &n1, &n2, &n3);
8       printf("n1 = %03d, n2 = %d, n3 = %d\n", n1, n2, n3);
9       return 0;
10  }
```

n1을 출력할 때 세 자릿수로 맞추고 앞에 빈 자리는 0으로 채움

3. 표준 입출력 함수

서식을 지정해서 값 읽기

- '010-1111-2222'를 입력했을 때

```
<실행 결과>

010-1111-2222
n1 = 010, n2 = 1111, n3 = 2222
```

- '010- 1111- 2222'를 입력했을 때

```
<실행 결과>

010-     1111-     2222
n1 = 010, n2 = 1111, n3 = 2222
```

3. 표준 입출력 함수

■ 서식을 지정해서 값 읽기

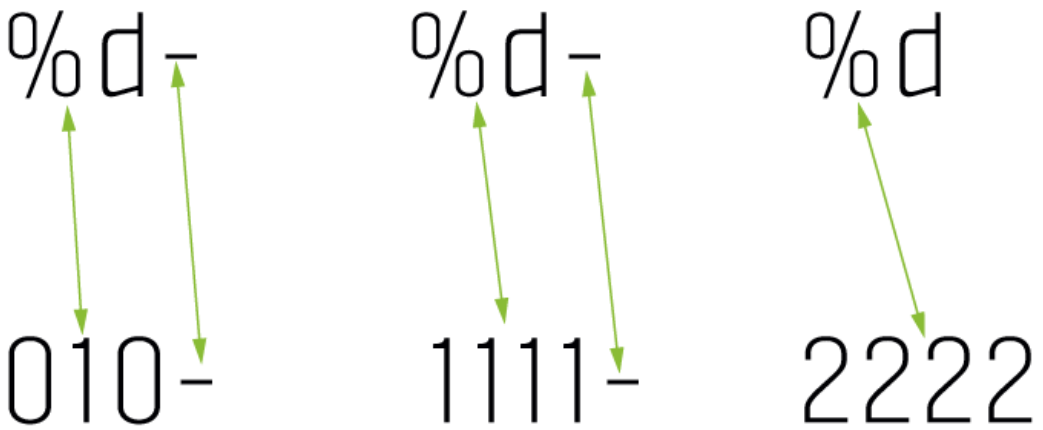


그림 3-13 "%d-%d-%d"와 "010- 1111- 2222"이 어떻게 사상(macthing)되는지 확인

- 숫자와 '-'의 간격이 벌어지게 입력했을 때

<실행 결과>

```
010    -1111    -2222
n1 = 010, n2 = 16, n3 = 673
```

04

다른 표준 입출력 함수

4. 다른 표준 입출력 함수

문자 단위 입출력 함수

- getchar(), getc(), putchar() 등
- 사용하려면 stdio.h를 포함해야 함

getchar() 함수

- 표준 입력 장치로부터 글자 한 개를 읽고 반환하는 함수

```
int getchar(void);
```

- int 형이 32 비트일 때, -1과 255의 비트 표현

```
-1의 비트 표현은 1111 1111 1111 1111 1111 1111 11111 1111
255의 비트 표현은 0000 0000 0000 0000 0000 0000 11111 1111
```

4. 다른 표준 입출력 함수

문자 단위 입출력 함수

getchar() 함수

- 문자 두 개를 입력받고 출력

코드 3-20 Getchar.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3
4   int main(void)
5   {
6       int ch = getchar( );
7       printf("ch = %c\n", ch);
8       ch = getchar( );
9       printf("ch = %c\n", ch);
10      return 0;
11  }
```

<실행 결과>

```
a
ch = a
ch =
```

4. 다른 표준 입출력 함수

문자 단위 입출력 함수

getc() 함수

코드 3-21 Getc.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3
4   int main(void)
5   {
6       int ch = getc(stdin);
7       printf("ch = %c\n", ch);
8       ch = getc(stdin);
9       printf("ch = %c\n", ch);
10      return 0;
11  }
```

4. 다른 표준 입출력 함수

문자 출력 함수

putchar() 함수

- 영문 알파벳, 숫자, 특수 문자 출력

코드 3-22 Putchar.c

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

int main(void)
{
    putchar('a');
    putchar('1');
    putchar('?');
    return 0;
}
```

<실행 결과>

```
a1?
```

4. 다른 표준 입출력 함수

■ 문자열 출력 함수

▪ putchar() 함수

코드 3-23 Puts.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3
4   int main(void)
5   {
6       puts("My first C program"); // 문자열 출력하는 puts() 함수 호출
7       return 0;
8   }
```

<실행 결과>

```
My first C program
```

4. 다른 표준 입출력 함수

문자열 출력 함수

putchar() 함수

- 어떤 문자열이든 화면에 출력한 후 무조건 줄바꿈

코드 3-24 Puts2.c

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

int main(void)
{
    puts("Hello World\n");
    puts("PI is 3.141519");
    return 0;
}
```

<실행 결과>

```
Hello World

PI is 3.141519
```