정수 자료형의 크기 및 범위

자료형	크기	범위	비고
char signed char	1바이트, 8비트	-128~127	
unsigned char	1바이트, 8비트	0~255	
short short int	2바이트, 16비트	-32,768~32,767	int 생략 가능
unsigned short unsigned short int	2바이트, 16비트	0~65,535	int 생략 가능
int signed int	4바이트, 32비트	-2,147,483,648~ 2,147,483,647	
unsigned unsigned int	4바이트, 32비트	0~4,294,967,295	int 생략 가능
long long int signed long signed long int	4바이트, 32비트	-2,147,483,648~ 2,147,483,647	int 생략 가능
unsigned long unsigned long int	4바이트, 32비트	0~4,294,967,295	int 생략 가능
long long long long int signed long long signed long long int	8바이트, 64비트	-9,223,372,036,854,775,808~ 9,223,372,036,854,775,807	int 생략 가능
unsigned long long unsigned long long int	8바이트, 64비트	0~18,446,744,073,709,551,615	int 생략 가능

실수 자료형의 크기 및 범위(Windows)

자료형	크기	범위	유효자릿수
float	4바이트, 32비트	1.175494e-38~3.402823e+38	7
double	8바이트, 64비트	2.225074e-308~1.797693e+308	16
long double	8바이트, 64비트	2.225074e-308~1.797693e+308	16

자료형	크기	부호	지수	가수
float	32비트	1비트	8비트	23비트
double	64비트	1비트	11비트	52비트

자료형 크기 구하기** => `sizeof`

```
sizeof(int);     // 자료형 자체의 크기 구할 때는 반드시 ( )를 지정
int num1;
sizeof(num1);    // ( )를 지정하는 방법
sizeof num1;     // ( ) 없이 표현식으로 지정하는 방법
```

정수 자료형의 오버플로우와 언더플로우

- 부호 있는 정수: 오버플로우가 발생하면 최솟값(음수)부터 다시 시작, 언더플로우가 발생하면 최댓값(양수)부터 점점 작아집니다.
- 부호 없는 정수: 오버플로우가 발생하면 0부터 다시 시작, 언더플로우가 발생하면 최댓값부터 점점 작아집니다.

limits.h에 정의된 정수 자료형의 최솟값과 최댓값

자료형	최솟값	최댓값
char	CHAR_MIN	CHAR_MAX
short	SHRT_MIN	SHRT_MAX
int	INT_MIN	INT_MAX
long	LONG_MIN	LONG_MAX
long long	LLONG_MIN	LLONG_MAX
unsigned char	0	UCHAR_MAX
unsigned short	0	USHRT_MAX
unsigned int	0	UINT_MAX
unsigned long	0	ULONG_MAX
unsigned long long	0	ULLONG_MAX

stdint.h에 정의된 정수 자료형의 최솟값과 최댓값

자료형	최솟값	최댓값
int8_t	INT8_MIN	INT8_MAX
int16_t	INT16_MIN	INT16_MAX
int32_t	INT32_MIN	INT32_MAX
int64_t	INT64_MIN	INT64_MAX
uint8_t	0	UINT8_MAX
uint16_t	0	UINT16_MAX
uint32_t	0	UINT32_MAX
uint64_t	0	UINT64_MAX

실수 자료형은 오버플로우(inf(무한대)와 언더플로우(0)

float.h에 정의된 최솟값과 최댓값

float.h 헤더 파일에는 실수 자료형의 최솟값과 최댓값이 정의되어 있습니다.

▼ **표** 실수 자료형의 최솟값과 최댓값

자료형	크기	범위
float	FLT_MIN	FLT_MAX
double	DBL_MIN	DBL_MAX
long double	LDBL_MIN	LDBL_MAX

printf로 정수나 실수를 출력할 때 숫자 서식 지정자

서식 지정자	설명
%d	부호 있는 정수
%ld	부호 있는 long 정수
%lld	부호 있는 long long 정수
%u	부호 없는 정수
%lu	부호 없는 long 정수
%llu	부호 없는 long long 정수
%f	float, double형 실수
%Lf	long double형 실수
%e	float, double형 실수를 지수 표기법으로 출력
%Le	long double형 실수를 지수 표기법으로 출력

문자는 printf함수에서 문자를 출력할 때 문자 서식 지정자 => `%c`

```
printf("%c\n", 'a');    // a
```

자주 사용하는 제어 문자

10진수	16진수	문자	표기법	설명
10	0x0A	LF	\n	개행, 라인 피드(Line Feed), 새 줄(newline), 줄바꿈
13	0x0D	CR	\r	복귀, 캐리지 리턴(Carriage Return), 줄의 끝에서 시작 위치로 되돌아감
9	0x09	TAB	\t	수평 탭(horizontal tab)

]리터럴과 상수

상수는 const를 붙여서 선언하며 리터럴이 저장된 공간(상수는 한 번 선언한 뒤에는 값을 변경할 수 x)

```c
// 리터럴
printf("%d\n", 10);
printf("%f\n", 0.1f);
printf("%c\n", 'a');

// 상수
const int con1 = 1;
const float con2 = 0.1f;
const char con3 = 'a';
```

정수 리터럴 접미사 => 정수의 크기와 부호 유무를 명시적으로 표기할 때 사용

접미사	자료형
생략	int
l, L	long
u, U	unsigned int
ul, UL	unsigned long
ll, LL	long long
ull, ULL	unsigned long long

실수 리터럴 접미사 => 실수 리터럴의 크기를 명시적으로 표기할 때 사용

접미사	자료형
f, F	float
생략	double
l, L	long double

입력 값을 변수에 저장 => `scanf 함수를 사용하며 변수 앞에는 &를 붙임`

- %d: 정수
- %f: 실수
- %c: 문자

```c
int num1;
float num2;
char c1;

scanf("%d %f %c", &num1, &num2, &c1);
```

getchar는 문자 하나를 입력받고, putchar는 문자 하나를 출력

```c
char c1 = getchar();    // 표준 입력에서 문자 하나를 입력받은 뒤 결과를 반환

putchar(c1);    // 문자 하나를 화면(표준 출력)에 출력
```

산술 연산자

연산자	설명	
+	덧셈	
-	뺄셈	
*	곱셈	
/	나눗셈	
%	나머지(정수를 나눈 뒤 나머지만 구함)	
+=	덧셈 후 할당	
-=	뺄셈 후 할당	
*=	곱셈 후 할당	
%=	나머지를 구한 후 할당	
전위	++a	1 증가, b = ++a; 증가 후 할당
--a	1 감소, b = --a; 감소 후 할당	
후위	a++	1 증가, b = a++; 할당 후 증가
a--	1 감소, b = a--; 할당 후 감소	

0으로 나누기 => `컴파일 에러발생`

논리연산자

&& = and

|| = or

! = NOT

불 자료형 => `include <stdool.h> 꼭 해줘야 함`

```
# include <stdool.h> // bool, true, false가 정의된 헤더 파일

int main ()
{
    bool b1;

    b1 = true;

    return 0 ;
}
```

비트연산자 => `이진수로 바꾼 후 계산`

연산자	설명	예시
&	비트 AND	`둘 다 1 이여야 1` ex) 8 & 5 = 1000 & 0111 = 0000
|	비트 OR	`하나면 1 이면 1` ex) 8 | 5 = 1000 | 0111 = 1111
^	비트 XOR (배타적 OR, Exclusive OR)	`숫자가 다르면 1 같으면 0` ex) 8 ^ 5 = 1000 ^ 0100 = 1100
~	비트 NOT	`~1100 이면 0011로 바꾼다는 뜻`
<<	비트를 왼쪽으로 시프트	가장 왼쪽에 있는 숫자는 부호비트로 음수(1) 양수(0)를 뜻한다 `왼쪽으로 숫자를 밀고 빈칸은 부호비트로 채우기`
>>	비트를 오른쪽으로 시프트	가장 왼쪽에 있는 숫자는 부호비트로 음수(1) 양수(0)를 뜻한다 `오른쪽으로 숫자를 밀고 빈칸은 부호비트로 채우기`
&=	비트 AND 연산 후 할당	`둘 다 1 이여야 1` ex) int a = 8 int b = 5 일 때 a &= b a = 0000 b = 0100
|=	비트 OR 연산 후 할당	`하나면 1 이면 1` ex) a = 1000 b = 0100 a |= b a = 1100 b = 0100
^=	비트 XOR 연산 후 할당	
<<=	비트를 왼쪽으로 시프트한 후 할당	a = 128 a <<= 3 하면 a = 1000 0000 a = 0000 0111
>>=	비트를 오른쪽으로 시프트한 후 할당	

비트 XOR 연산자는 간단한 암호화에 사용

```
int plane = 100;
int key = 57;

int code = plane ^ key;
printf("%d\n", code); // 93

int newcode = code ^ key;
printf("%d\n", newcode); //100
```

- a ^ b = c
- c ^ b = a
- c ^ a = b

나누기 2를 하는 2가지 방법

```
a = a >> 1;
b = b / 2;
```

switch 분기문으로 다양한 조건 처리하기

- 정수 실수형과 char 만 쓸 수 있다
- break를 써야 뒷 코드가 실행 되지 않는다
- case 문에 안에서 변수를 선언해 주려면 {}를 사용해야 한다

```
# include <studio.h>
#define _CRT_SECURE_NO_WARNINGS    // scanf 보안 경고로 인한 컴파일 에러 방지

int main() {
    char c1;
    char num1;

    scanf("%c", &c1);

    switch (c1)
    {
        case 'a': // a 또는
        case '1': // 1일 때 아래 코드 실행
            {
                char c2 = c1;
                printf("a입니다.\n");
                break;
            }

        case 'b':
            printf("b입니다.\n");
            break;

        case 'c':
```

```
            printf ("c입니다.\n");
                break;

        default :
            printf("default\n");
            break;

    }

    return 0;
}
```

for 문으로 "Hello World"문 백번 출력하기

- for문과 if문 끝에는 세미콜론 붙이면 안된다

```
# include <studio.h>

int main()
{
    for (int i = 0, i < 100, i ++)
    {
        printf("Hello World!\n")
    }

    return 0;
}
```

- for 문 안에 반복하고 싶은 코드가 두개 이상이라면 반드시 중괄호로 묶어줘야 한다

```
# include <studio.h>

int main()
{
    for (int i = 0, i < 100, i ++) // 세미콜론 있으면 안되고
    { // 중괄호 있어야 하고
        printf("Hello World!\n"); // Hello World!도 100번
        printf("계속 반복\n"); // 계속 반복도 100번

    }

    for (int i = 0, i < 100, i ++)
    // 중괄호 없으면
        printf("Hello World!\n"); // Hello World!도 100번
        printf("계속 반복\n"); // 계속 반복도 1번


    return 0;
```

```
}
```

for 문으로 무한 루프 만들기

```
#include <stdio.h>

int main()
{
    for (;;)    // 초깃값, 조건식, 변화식을 모두 생략하면 무한 루프
    {
        printf("Hello, world!\n");
    }

    return 0;
}
```

while 문으로 "Hello World"문 백번 출력하기

```
#include <stdio.h>

int main()
{
    int i = 0;
    while (i < 100)    // i가 100보다 작을 때 반복. 0에서 99까지 증가하면서 100번 반복
    {
        printf("Hello, world!\n");
        i++;           // i를 1씩 증가시킴
    }

    return 0;
}
```

while 문으로 무작위 숫자 생성하기

```
#include <stdio.h>
#include <stdlib.h>    // srand, rand 함수가 선언된 헤더 파일
#include <time.h>      // time 함수가 선언된 헤더 파일

int main()
{
    srand(time(NULL));    // 현재 시간값으로 시드 설정

    int i = 0;
    while (i != 3)         // 3이 아닐 때 계속 반복
    {
```

```
        i = rand() % 10;   // rand 함수를 사용하여 무작위로 정수를 생성한 뒤 10 미만의 숫자로 만듦
        printf("%d\n", i);
    }

    return 0;
}
```

- srand: 난수를 발생시킬 초깃값인 시드(seed)를 설정합니다. 보통 현재 시간값을 사용합니다.
- rand: 난수를 발생시킵니다.
- time: 정수 형태로 된 현재 시간값을 반환합니다.

do while 문으로 무조건 한번 실행하고 반복문 실행하기

```
#define _CRT_SECURE_NO_WARNINGS    // scanf 보안 경고로 인한 컴파일 에러 방지
#include <stdio.h>

int main()
{
    int count;

    scanf("%d", &count);    // 값을 입력받음

    int i = 0;
    do // 처음 한 번은 아래 코드가 실행됨
    {
        printf("Hello, world! %d\n", i);    // Hello, world!와 i의 값을 함께 출력
        i++;                                // i를 1씩 증가시킴
    } while (i < count);                    // i가 count보다 작을 때 반복

    return 0;
}
```

break, continue로 반복문 제어하기

- break는 반복문을 아예 중단하고 빠져 나오지만
- continue는 반복문은 유지하되 continue의 하단 부분의 코드 실행만 건너뛰는 역할을 합니다.

계단식 별 모양 그리기

```
*
**
***
****
*****
```

```
#include <stdio.h>

int main()
{
    for (int i = 0; i < 5; i++)    // 5번 반복. 세로 방향
    {
        for (int j = 0; j < 5; j++)   // 5번 반복. 가로 방향
        {
            if (j <= i)               // 세로 방향 변수만큼
                printf("*");          // 별 출력
        }
        printf("\n");              // 가로 방향으로 별을 다 그린 뒤 다음 줄로 넘어감
    }

    return 0;
}
```

대각선 별 모양 그리기

```
*
 *
  *
   *
    *
```

```
#include <stdio.h>

int main()
{
    for (int i = 0; i < 5; i++)    // 5번 반복. 세로 방향
    {
        for (int j = 0; j < 5; j++)    // 5번 반복. 가로 방향
        {
            if (j == i)               // 세로 방향 변수와 같을 때
                printf("*");          // 별 출력
            else                      // 세로 방향 변수와 다를 때
                printf(" ");          // 공백 출력
        }
        printf("\n");             // 가로 방향으로 별을 다 그린 뒤 다음 줄로 넘어감
    }

    return 0;
}
```

go to 레이블

- 많이 쓰면 꼬이니까 보기 안 좋음

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

int main()
{
    int num1;

    scanf("%d", &num1);

    if (num1 == 1)          // num1이 1이면
        goto ONE;           // 레이블 ONE으로 즉시 이동
    else if (num1 == 2)     // num1이 2이면
        goto TWO;           // 레이블 TWO로 즉시 이동
    else                    // 1도 아니고 2도 아니면
        goto EXIT;          // 레이블 EXIT로 즉시 이동

ONE:    // 레이블 ONE
    printf("1입니다.\n");
    goto EXIT; // 레이블 EXIT로 즉시 이동

TWO:    // 레이블 TWO
    printf("2입니다.\n");
    goto EXIT; // 레이블 EXIT로 즉시 이동

EXIT:    // 레이블 EXIT
    return 0;
}
```

삼항 연산자 => `조건을 기준으로 ? 참 일때 코드 : 거짓일 때 코드`

```c
printf(i % 3 ? i % 5 ? "%d\n" : "Buzz\n" : i % 5 ? "Fizz\n" : "FizzBuzz\n", i);
```

포인터 출력하기 => `printf("%p\n", &num1)`

- 출력할 때 %p와 변수앞에 &(주소연산자)를 붙인다

```c
#include <stdio.h>

int main()
{
    int num1 = 10;

    printf("%p\n", &num1);    // 008AF7FC: num1의 메모리 주소를 출력
                              // 컴퓨터마다, 실행할 때마다 달라짐

    return 0;
}
```

포인터 변수

포인터 변수는 *를 사용하여 선언합니다(포인터 변수는 포인터로 줄여서 부르기도 합니다).

- **int(자료형) * 포인터이름;**
- **포인터 = &변수;**

역 참조하면 다시 int 출력 가능

```
#include <stdio.h>

int main()
{
    int *numPtr;      // 포인터 변수 선언

    int num1 = 10;    // int형 변수를 선언하고 10 저장

    numPtr = &num1;   // num1의 메모리 주소를 포인터 변수에 저장

    printf("%p\n", numPtr);    // 0055FC24: 포인터 변수 numPtr의 값 출력
                               // 컴퓨터마다, 실행할 때마다 달라짐
    printf("%p\n", &num1);     // 0055FC24: 변수 num1의 메모리 주소 출력
                               // 컴퓨터마다, 실행할 때마다 달라짐
    printf("%d\n", *numPtr);    // 10: 역참조 연산자로 num1의 메모리 주소에 접근하여 값을 가져옴

    return 0;

}
```

다양한 변수에 포인터 할당하기

```
#include <stdio.h>

int main()
{
    long long *numPtr1;    // long long형 포인터 선언
    float *numPtr2;        // float형 포인터 선언
    char *cPtr1;           // char형 포인터 선언

    long long num1 = 10;
    float num2 = 3.5f;
    char c1 = 'a';

    numPtr1 = &num1;    // num1의 메모리 주소 저장
    numPtr2 = &num2;    // num2의 메모리 주소 저장
    cPtr1 = &c1;        // c1의 메모리 주소 저장

    printf("%lld\n", *numPtr1);    // 10
```

```
    printf("%f\n", *numPtr2);      // 3.500000
    printf("%c\n", *cPtr1);        // a

    return 0;
}
```

void 포인터 = `void * ptr`

자료형이 정해지지 않은 포인터

역 참조는 불가능

```
#include <stdio.h>

int main()
{
    int num1 = 10;
    char c1 = 'a';
    int *numPtr1 = &num1;
    char *cPtr1 = &c1;

    void *ptr;        // void 포인터 선언

    // 포인터 자료형이 달라도 컴파일 경고가 발생하지 않음
    ptr = numPtr1;    // void 포인터에 int 포인터 저장
    ptr = cPtr1;      // void 포인터에 char 포인터 저장

    // 포인터 자료형이 달라도 컴파일 경고가 발생하지 않음
    numPtr1 = ptr;    // int 포인터에 void 포인터 저장
    cPtr1 = ptr;      // char 포인터에 void 포인터 저장


        printf("%d", *ptr);   // void 포인터는 역참조할 수 없음. 컴파일 에러
        printf("%c", *ptr);   // void 포인터는 역참조할 수 없음. 컴파일 에러

    return 0;
}
```

이중 포인터 = `int(자료형) \**numPtr2(포인터이름);`

- 포인터의 포인터 선언하기

```
#include <stdio.h>

int main()
{
    int *numPtr1;     // 단일 포인터 선언
    int **numPtr2;    // 이중 포인터 선언
    int num1 = 10;
```

```
    numPtr1 = &num1;    // num1의 메모리 주소 저장

    numPtr2 = &numPtr1; // numPtr1의 메모리 주소 저장

    printf("%d\n", **numPtr2);    // 20: 포인터를 두 번 역참조하여 num1의 메모리 주소에 접근

    return 0;
}
```

메모리 할당하기 => `numPtr2 = malloc(sizeof(int));`

#include <stdlib.h> 선언 해야함

메모리를 사용하려면 malloc 함수로 사용할 메모리 공간을 확보

메모리 크기는 바이트 단위로 지정합니다

- **포인터 = malloc(크기);**
- ◦ 성공하면 메모리 주소를 반환, 실패하면 NULL을 반환

```
#include <stdio.h>
#include <stdlib.h>    // malloc, free 함수가 선언된 헤더 파일

int main()
{
    int num1 = 20;    // int형 변수 선언
    int *numPtr1;     // int형 포인터 선언

    numPtr1 = &num1;  // num1의 메모리 주소를 구하여 numPtr에 할당

    int *numPtr2;     // int형 포인터 선언

    numPtr2 = malloc(sizeof(int));    // int의 크기 4바이트만큼 동적 메모리 할당

    printf("%p\n", numPtr1);    // 006BFA60: 변수 num1의 메모리 주소 출력
                                // 컴퓨터마다, 실행할 때마다 달라짐

    printf("%p\n", numPtr2);     // 009659F0: 새로 할당된 메모리의 주소 출력
                                // 컴퓨터마다, 실행할 때마다 달라짐

    free(numPtr2);    // 동적으로 할당한 메모리 해제

    return 0;
}
```

한꺼번에 메모리 할당하기 => `memset(numPtr, 0, 8);`

string.h 헤더 파일에 선언

- **memset(포인터, 설정할값, 크기);**
- - **void *memset(void *_Dst, int _Val, size_t _Size);**
  - 값 설정이 끝난 포인터를 반환

```
#include <stdio.h>
#include <stdlib.h>    // malloc, free 함수가 선언된 헤더 파일
#include <string.h>    // memset 함수가 선언된 헤더 파일

int main()
{
    long long *numPtr = malloc(sizeof(long long));  // long long의 크기 8바이트만큼 동적 메모리 할당

    memset(numPtr, 0x27, 8);    // numPtr이 가리키는 메모리를 8바이트만큼 0x27로 설정

    printf("0x%llx\n", *numPtr);    // 0x2727272727272727: 27이 8개 들어가 있음

    free(numPtr);    // 동적으로 할당한 메모리 해제

    return 0;
}
```

자료형의 크기와 포인터의 크기

- memset 함수에 설정할 크기를 지정할 때 보통 숫자 대신 sizeof를 사용

```
char *cPtr = malloc(sizeof(char));    // char의 크기 1바이트만큼 동적 메모리 할당

memset(cPtr, 0, sizeof(char));    // char의 크기 1바이트만큼 0으로 설정(올바른 방법)
memset(cPtr, 0, sizeof(char *));  // 32비트: char 포인터의 크기 4바이트만큼 0으로 설정(잘못된 방법)
                                  // 64비트: char 포인터의 크기 8바이트만큼 0으로 설정(잘못된 방법)
free(cPtr);
```

- memset 함수에서 sizeof를 사용할 때는 이러한 부분을 주의해야 합니다.

null 포인터

- 메모리가 할당 되지 않은 포인터에 메모리 할당하기
- 아무것도 가리키지 않는 상태를 뜻합니다. 따라서 역참조는 할 수 없습니다.

```c
#include <stdio.h>

int main()
{
    int *numPtr1 = NULL;    // 포인터에 NULL 저장

    printf("%p\n", numPtr1);    // 00000000

    return 0;
}
```

실무에서는 다음과 같이 포인터가 NULL인지 확인한 뒤 NULL이면 메모리를 할당하는 패턴을 주로 사용합니다.

```c
if (ptr == NULL)         // ptr이 널 포인터라면
{
    ptr = malloc(1024);   // 1024바이트만큼 메모리 할당
}
```

핵심 정리

문자열

문자열은 " "(큰따옴표)로 묶으며 char 포인터나 배열에 저장하여 사용합니다.

```c
char *s1 = "Hello, world!";       // char 포인터에 문자열의 메모리 주소를 저장
char s2[20] = "Hello, world!";    // char 배열에 문자열 저장
char s3[] = "Hello, world!";      // char 배열에 문자열 저장(배열의 크기 생략)
```

문자열을 인덱스로 접근하면 문자를 출력할 수 있습니다.

```c
// 문자열을 인덱스로 접근
printf("%c\n", s1[0]);    // H
printf("%c\n", s2[4]);    // o
```

이미 선언된 배열에는 문자열을 할당할 수 없습니다.

```c
char s1[20];

s1 = "Hello, world";    // 컴파일 에러
```

문자열 포인터와 문자 배열의 요소 변경

문자열 포인터에 문자열 리터럴을 할당한 뒤에는 인덱스로 접근하여 문자(요소)를 변경할 수 없습니다(문자열 리터럴이 있는 메모리 주소는 읽기 전용). 하지만 문자 배열은 인덱스로 접근하여 문자를 변경할 수 있습니다(배열에 문자열이 복사됨).

```
char *s1 = "Hello";       // 문자열 포인터에 문자열 리터럴 할당(읽기 전용 메모리를 가리킴)
s1[0] = 'A';              // 실행 에러: 문자를 변경할 수 없음

char s2[10] = "Hello";    // 문자 배열에 문자열 리터럴 할당(배열에 문자열이 복사됨)
s1[0] = 'A';              // 문자를 변경할 수 있음
```

입력된 문자열을 배열 또는 포인터에 저장하기

입력된 문자열을 배열에 저장할 때는 scanf 함수를 사용하며 배열 앞에는 &를 붙이지 않습니다.

```
char s1[10];        // 크기가 10인 char형 배열을 선언

scanf("%s", s1);    // 표준 입력을 받아서 배열 형태의 문자열에 저장
```

입력된 문자열을 포인터에 저장할 때도 포인터 앞에는 &를 붙이지 않습니다.

```
char *s1 = malloc(sizeof(char) * 10);    // char 10개 크기만큼 동적 메모리 할당

scanf("%s", s1);    // 표준 입력을 받아서 배열 형태의 문자열에 저장

free(s1);    // 동적 메모리 해제
```

문자열과 서식 지정자

문자열은 서식 지정자 %s를 사용합니다.

```
char s1[20];
scanf("%s", s1);     // %s로 문자열을 입력받음
printf("%s", s1);    // %s로 문자열을 출력
```

scanf로 공백까지 포함하여 문자열을 입력받으려면 서식 지정자로 "%[^\n]s"를 사용합니다.

```
char s1[30];
scanf("%[^\n]s", s1);    // 공백까지 포함하여 문자열 입력받기
```

문자열과 NULL

문자열은 문자열의 끝나는 지점을 알려주기 위해 마지막에 NULL(\0)이 붙습니다. 만약 배열이라면 NULL까지 들어가야 하므로 배열의 크기는 문자열의 길이보다 1이 더 커야 합니다.

```
char s1[6] = "Hello";    // 5글자 + NULL이므로 배열의 최소 크기는 6이상
```