Power C++

제6-1장 문자열 String

- C++ 에서 문자열을 다루는 두 가지 방법
  - 문자 배열 (C-string)
  - string class : 8장

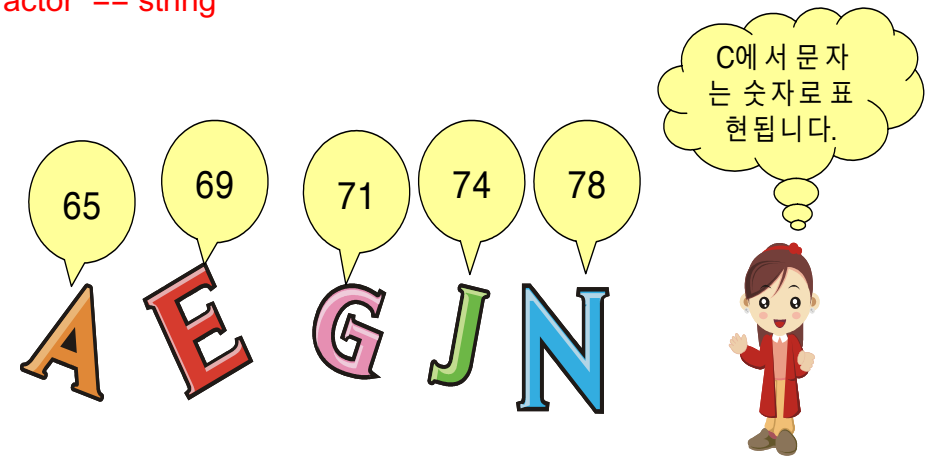
문자 : charactor
문자열 : string (charactor's sequence)

string type이 없어서 배열로 받아옴

문자표현방법

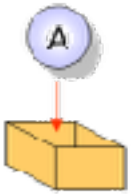
- 컴퓨터에서는 각각의 문자에 숫자코드를 붙여서 표시한다.
- 아스키코드(ASCII code): 표준적인 8비트 문자코드
  - 0에서 127까지의 숫자를 이용하여 문자표현
- 유니코드(unicode): 표준적인 16비트 문자코드
  - 전세계의 모든 문자를 일관되게 표현하고 다룰 수 있도록 설계

array of charactor == string



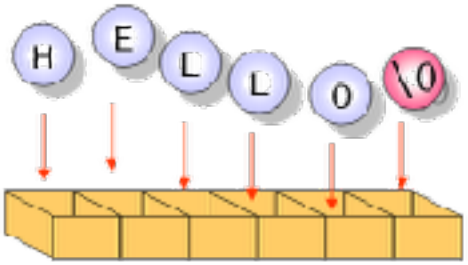
문자열 표현 방법

- 문자열*(string):* 문자들이 여러 개 모인것
  - "A"
  - "Hello World!"
  - "변수 score의 값은 %d입니다“
- 문자열 상수
  - "Hello World"
  - "Hong"
  - "string!#$"
  - "guest123"
  - "ascii code = %d“
- 문자열 변수
  - char형 배열

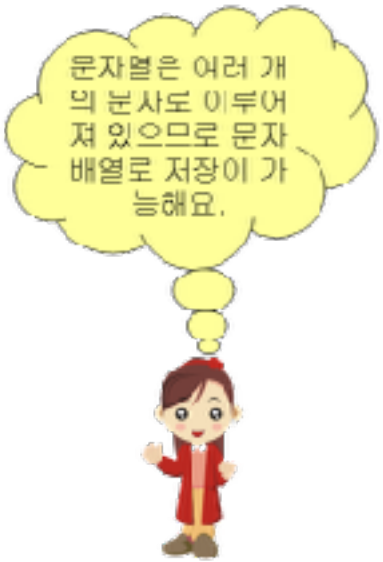


하나의 문자는 char형 변수로 저장

숫자 0

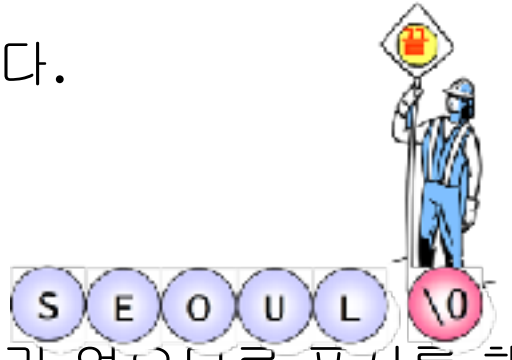


문자열은 char형 배열로 저장

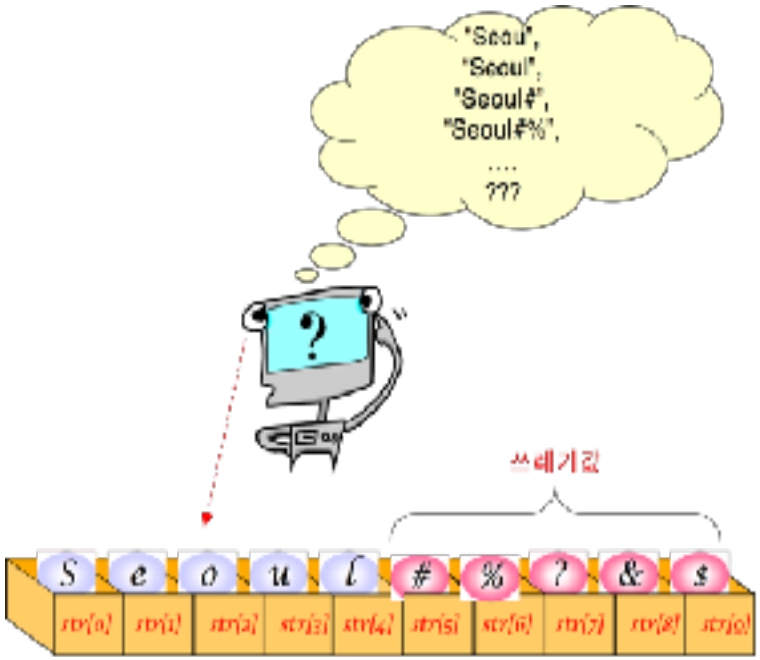


NULL 문자

- NULL 문자: 문자열의 끝을 나타낸다.



- 문자열은 어디서 종료되는지 알수가 없으므로 표시를 해주어야 한다.



문자 배열의 초기화

1. 문자 배열 원소들을 중괄호 안에 넣어주는 방법

   - `char str[6] = { 'H', 'e', 'l', 'l', 'o', '\0' };`

2. 문자열 상수를 사용하여 초기화하는 방법

   - `char str[6] = "Hello";`· 단지 컴파일러가 편한 표기법을 제공해 주는것이지, string 타입은 원래 존재하지 않음

3. 만약 배열을 크기를 지정하지 않으면 컴파일러가 자동으로 배열의 크기를 초기화값에 맞추어 설정

   - `char str[] = "C Bible";` `// 배열의 크기는 7이 된다.`

문자 배열에 문자를 저장

1. 각각의 문자 배열 원소에 원하는 문자를 개별적으로 대입하는 방법이다.
   - str[0] = 'W';
   - str[1] = 'o';
   - str[2] = 'r';
   - str[3] = 'l';
   - str[4] = 'd';
   - str[5] = '\0';

2. strcpy()를 사용하여 문자열을 문자 배열에 복사
   - strcpy(str, "World");

예제 #1

```
#include <iostream>
using namespace std;

int main()
{
	char str1[7] = "Seoul ";
	char str2[3] = { 'i', 's' };
	char str3[] = " the capital city of Korea.";

	cout << str1 << str2 << str3 << endl;
	return 0;
}
```

```
Seoul is the capital city of Korea.
```

예제 #2

```
#include <iostream>
using namespace std;

int main()
{
	char str[] = "komputer";
	int i;

	for(i=0;i<8;i++)
		cout << str[i] << " ";

	str[0] = 'c';
	cout << endl;

	for(i=0;i<8;i++)
		cout << str[i] << " ";
	cout << endl;
	return 0;
}
```


```
k o m p u t e r
c o m p u t e r
```

문자열 길이 계산 예제

```
// 문자열의 길이를 구하는 프로그램
#include <iostream>
using namespace std;

int main()
{
	char str[30] = "C++ language is easy";
	int i = 0;

	while(str[i] != 0)
		i++;
	cout << "문자열 "<< str << "의 길이는 " << i << "입니다." << endl;

	return 0;
}
```

```
문자열 C++ language is easy의 길이는 20입니다.
계속하려면 아무 키나 누르십시오 . . .
```

문자 처리 함수들 (#include <cctype>

- 문자를 검사하거나 문자를 변환한다.

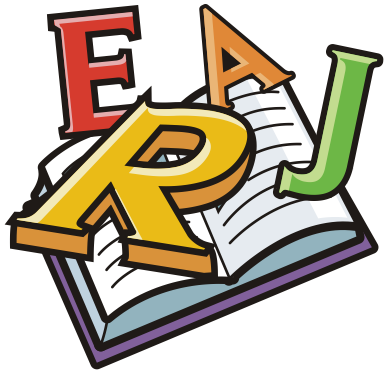
c++charactor type

코드로 짜보기!

함수	설명
isalpha(c)	c가 영문자인가?(a-z, A-Z)
isupper(c)	c가 대문자인가?(A-Z)
islower(c)	c가 소문자인가?(a-z)
isdigit(c)	c가 숫자인가?(0-9)
isalnum(c)	c가 영문자이나 숫자인가?(a-z, A-Z, 0-9)
isxdigit(c)	c가 16진수의 숫자인가?(0-9, A-F, a-f)
isspace(c)	c가 공백문자인가?(‘ ’, ‘\n', '\t', '\v', '\r')
ispunct(c)	c가 구두점 문자인가?
isprint(c)	C가 출력가능한 문자인가?
iscntrl(c)	c가 제어 문자인가?

함수	설명
toupper(c)	c를 대문자로 바꾼다.
tolower(c)	c를 소문자로 바꾼다.
toascii(c)	c를 아스키 코드로 바꾼다.

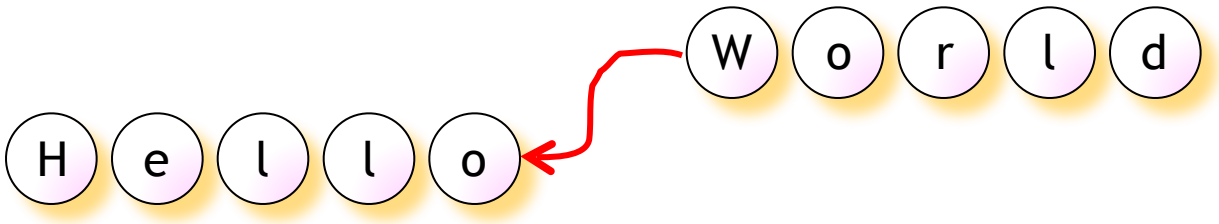
```
bool is_lower(char c) {
    if (c < 'a' || c > 'z') {
        return false;
    } return true;
}
```




```
bool is_alpha(char c) {
    return (is_upper(c) == true ||
is_lower(c) == true);
}
```

문자열 처리 함수들 (#include <cstring>)

함수	설명
strlen(s)	문자열 s의 길이를 구한다.
strcpy(s1, s2)	s2를 s1에 복사한다.
strcat(s1, s2)	s2를 s1의 끝에 붙여넣는다.
strcmp(s1, s2)	s1과 s2를 비교한다.
strncpy(s1, s2, n)	s2의 최대 n개의 문자를 s1에 복사한다.
strncat(s1, s2, n)	s2의 최대 n개의 문자를 s1의 끝에 붙여넣는다.
strncmp(s1, s2, n)	최대 n개의 문자까지 s1과 s2를 비교한다.
strchr(s, c)	문자열 s안에서 문자 c를 찾는다.
strstr(s1, s2)	문자열 s1에서 문자열 s2를 찾는다.



문자열 길이, 복사

배열끼리 equal 하는건 * (포인터, 위치)를 복사한거니까 (배열) = (배열) 불가
dst는 당연히 src 길이 이상의 배열이어야 함.

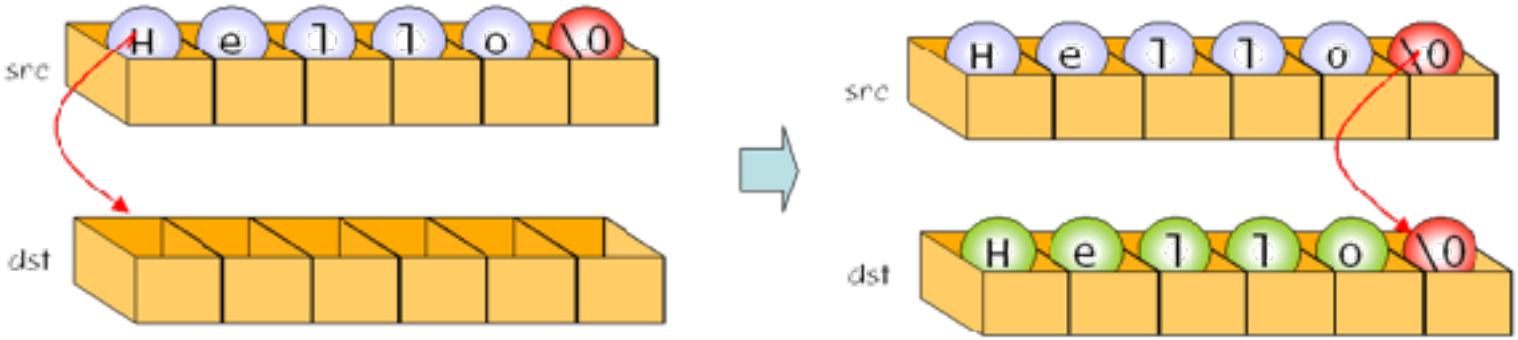
- 문자열의 길이
  - strlen(“Hello”)는 5를 반환
- 문자열 복사

```
char dst[6];
char src[6] = “Hello";
strcpy(dst, src);
```

src 는 const
(읽기 전용)

```
void mystrcpy() (char* dst, const char* src) {
    int i = 0;
   while (src[i] != '\0') {
       dst[i] = src[i];
       ++i;
   } dst[i] = '\0';
}
```

```
void my_strcpy(char* dst, const char* src) {
   int len = strlen(src);
   memcpy(dst, src, 1 + sizeof(char)*n);  // 마지막에 끝낸다는 표시는 복사를 안했기 때문에
}
```

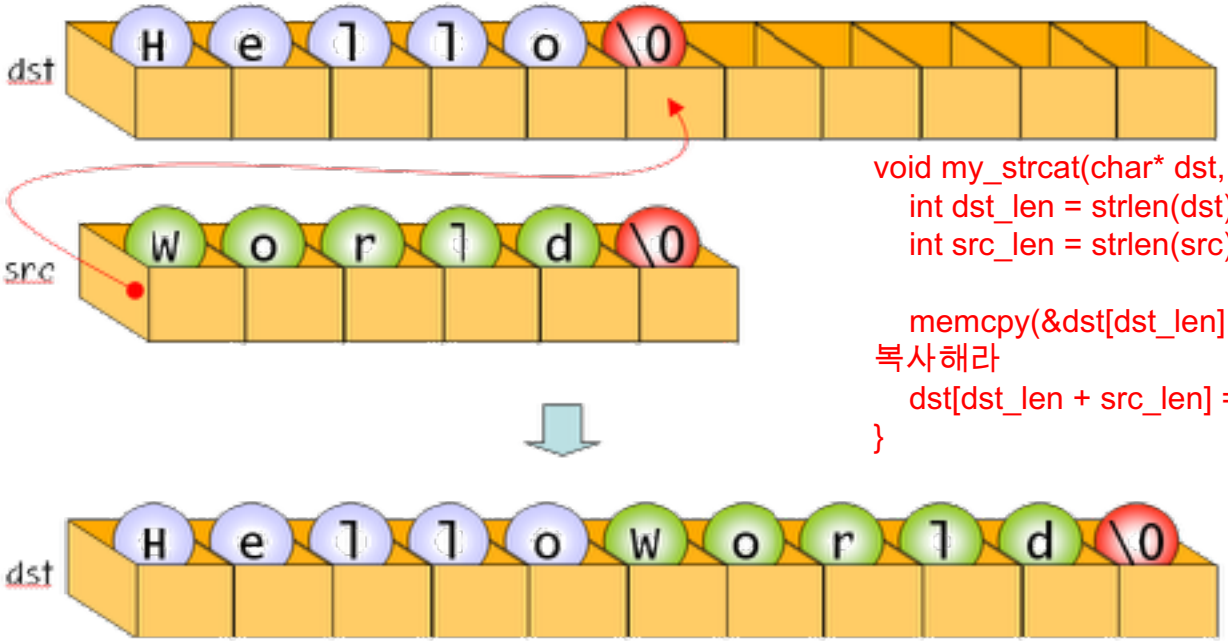


문자열 연결

- 문자열 연결

```
char dst[12] = "Hello";
char src[6] = "World";
strcat(dst, src);
```

index를 초과하면 프로그램은 죽음




```
void my_strcat(char* dst, const char* src) {
    int dst_len = strlen(dst);
    int src_len = strlen(src);

    memcpy(&dst[dst_len], str);  // '\0' 에서부터 str를 복사해라
    dst[dst_len + src_len] = '\0';
}
```

예 제

```
// strcpy와 strcat
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
      char string[80];

      strcpy( string, "Hello World from " );
      strcat( string, "strcpy() " );
      strcat( string, "and " );
      strcat( string, "strcat()!" );
      cout << string << endl;
      return 0;
}
```

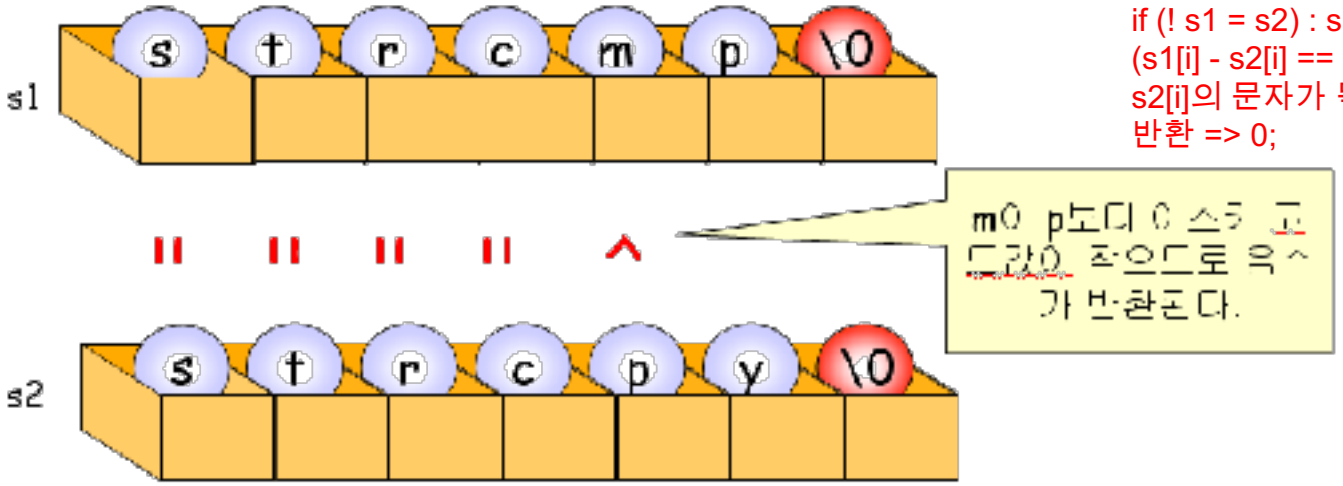
```
Hello World from strcpy() and strcat()!
```

문자열 비교

```
int strcmp( const char *s1,   const char *s2 );
```

s1[i] - s2[i]

반환값	s1과 s2의 관계
<0	s1이 s2보다 작다
0	s1이 s2와 같다.
>0	s1이 s2보다 크다.



if (s1[i] = s2[2])
if (! s1 = s2) : s1과 s2가 같으면~
(s1[i] - s2[i] == 0이면 s1[i]값과 s2[i]의 문자가 똑같다는 말 => true 반환 => 0;

예제

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
	char s1[80];		// 첫번째 단어를 저장할 문자배열
	char s2[80];		// 두번째 단어를 저장할 문자배열
	int result;

	cout << "첫번째 단어를 입력하시오:";
	cin >> s1;
	cout << "두번째 단어를 입력하시오:";
	cin >> s2;
	result = strcmp(s1, s2);
	if( result < 0 )
		cout << s1 << "이 " << s2 << "보다 앞에 있습니다." << endl;
	else if( result == 0 )
		cout << s1 << "이 " << s2 << "와 같습니다." << endl;
	else
		cout << s1 << "이 " << s2 << "보다 뒤에 있습니다." << endl;
	return 0;
}
```

```
첫번째 단어를 입력하시오:cat
두번째 단어를 입력하시오:dog
cat이 dog보다 앞에 있습니다.
```

example code : char *strncpy(s1, s2, n)

```
#include <iostream>
#include <cstring>

int main()
{
    const char* src = "hi";
    char dest[6] = {'a', 'b', 'c', 'd', 'e', 'f'};
    std::strncpy(dest, src, 5);

    std::cout << "The contents of dest are: ";
    for (char c : dest) {
        if (c) {
            std::cout << c << ' ';
        } else {
            std::cout << "\\0" << ' ';
        }
    }
    std::cout << '\n';
}
```

Output:
The contents of dest are: h i \0 \0 \0 f

중간 점검 문제

1. C-문자열 src를 C-문자열 dst로 복사하는 문장을 써라.
2. “String"을 저장하려면 최소한 어떤 크기 이상의 문자 배열이 필요한가?
3. 문자열을 서로 비교하는 함수는?

배열 처리 함수 (#include <cstring>)

함수	설명
memcpy(s1, s2, n)	s2로부터 s1으로 n bytes를 복사한다. strcpy() 나 memmove() 보다 빠르게 수행된다.
memset(s1, ch, n)	s1을 ch 의 값으로 n bytes 만큼 채운다. 무조건 ch 단위로만 사용 가능
memmove(s1, s2, n)	s2로부터 s1으로 n bytes를 복사한다. s1과 s2가 가리키는 배열이 겹칠 수 있다.

std::fill
특정 인자로 채워주는 것.

example code : void *memcpy(s1, s2, n)

```
#include <iostream>
#include <cstdint>
#include <cstring>

int main()
{
    // simple usage
    char source[] = "once upon a midnight dreary...", dest[4];
    std::memcpy(dest, source, sizeof dest);
    for (char c : dest)
        std::cout << c << '\n';

}
```

Output:

```
o
n
c
e
```

example code : void *memcpy(s1, s2, n)

```
#include <iostream>
#include <cstring>

int main()
{
    int a[20];
    std::memset(a, 0, sizeof a);
    for (int ai : a) std::cout << ai;
}
```

Output:

00000000000000000000

example code : void *memmove(s1, s2, n)

```
#include <iostream>
#include <cstring>

int main()
{
    char str[] = "1234567890";
    std::cout << str << '\n';
    std::memmove(str + 4, str + 3, 3); // copies from [4, 5, 6] to [5, 6, 7]
    std::cout << str << '\n';
}
```

Output:

1234567890

1234456890

문자열 -> 숫자 변환 함수 (#include <cstdlib>)

함수	설명
int atoi(const char *str)	str 이 가리키는 문자열을 정수로 해석하여 반환한다.
double atof(const char *str)	str 이 가리키는 문자열을 부동소수점수로 해석하여 반환한다.
long strtol(const char *str, char **str_end, int base)	str 이 가리키는 문자열을 base-진수 정수로 해석하여 반환한다. base 는 0,2,3,…,36 의 정수이다. str_end 는 해석된 마지막 문자의 다음 문자를 가리킨다.

- 공백문자를 무시하고 첫번째 공백문자가 아닌 문자부터 시작하여 유효한 숫자로 해석할 수 있는 최대한 수만큼의 글자를 읽어 숫자로 변환한다. (Discards any whitespace characters until the first non-whitespace character is found, then takes as many characters as possible to form a valid integer number representation and converts them to an integer value.)

- 변환된 값이 없으면 0을 반환한다.

example code : int atoi(const char *str)

```
#include <iostream>
#include <cstdlib>

int main()
{
    const char *str1 = "42";
    const char *str2 = "3.14159";
    const char *str3 = "31337 with words";
    const char *str4 = "words and 2";

    int num1 = std::atoi(str1);
    int num2 = std::atoi(str2);
    int num3 = std::atoi(str3);
    int num4 = std::atoi(str4);

    std::cout << "std::atoi(\"" << str1 << "\") is " << num1 << '\n';
    std::cout << "std::atoi(\"" << str2 << "\") is " << num2 << '\n';
    std::cout << "std::atoi(\"" << str3 << "\") is " << num3 << '\n';
    std::cout << "std::atoi(\"" << str4 << "\") is " << num4 << '\n';
}
```

Output:

```
std::atoi("42") is 42
std::atoi("3.14159") is 3
std::atoi("31337 with words") is 31337
std::atoi("words and 2") is 0
```

example code : double atof(const char *str)

```
#include <iostream>
#include <cstdlib>

int main()
{
        std::cout << std::atof("0.0000000123") << "\n"
                << std::atof("0.012") << "\n"
                << std::atof("15e16") << "\n"
                << std::atof("-0x1afp-2") << "\n"
                << std::atof("inF") << "\n"
                << std::atof("Nan") << “\n";
}
```

Output:

```
1.23e-08
0.012
1.5e+17
-107.75
inf
nan
```

example code : long strtol(...)

```
#include <iostream>
#include <string>
#include <cerrno>
#include <cstdlib>

int main()
{
    const char* p = "10 200000000000000000000000000000 30 -40";
    char *end;
    std::cout << "Parsing '" << p << "':\n";
    for (long i = std::strtol(p, &end, 10);
         p != end;
         i = std::strtol(p, &end, 10))
    {
        std::cout << ''' << std::string(p, end-p) << "' -> ";
        p = end;
        if (errno == ERANGE){
            std::cout << "range error, got ";
            errno = 0;
        }
        std::cout << i << '\n';
    }
}
```

Output:

```
Parsing '10 200000000000000000000000000000 30 -40':
'10' -> 10
' 200000000000000000000000000000' -> range error, got 9223372036854775807
' 30' -> 30
' -40' -> -40
```

formatted string using sprintf() (#include <cstdio>)

함수	설명
int sprintf(char* buffer, const char* format, ...);	format 에 따라 만들어진 문자열을 buffer 에 저장한다. format 문자열은 format tag를 포함할 수 있는데 각각의 tag 는 그 다음 형식 인자값의 출력 형식을 지정하는데 쓰인다. format tag 는 %[flags][width][.precision][length]specifier 의 형태로 이루어진다.

int printf(...) // 매개변수 개수가 정해지지 않았을 때

sprintf format specifier

- c Character
- d or i Signed decimal integer
- e Scientific notation (mantissa/exponent) using e character
- E Scientific notation (mantissa/exponent) using E character
- f Decimal floating point
- g Uses the shorter of %e or %f.
- G Uses the shorter of %E or %f
- o Signed octal
- s String of characters
- u Unsigned decimal integer
- x Unsigned hexadecimal integer
- X Unsigned hexadecimal integer (capital letters)
- p Pointer address
- n Nothing printed
- % Character

sprintf format flags

- - Left-justify within the given field width; Right justification is the default (see width sub-specifier).
- + Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a -ve sign.
- (space) If no sign is going to be written, a blank space is inserted before the value.
- # Used with o, x or X specifiers the value is preceded with 0, 0x or 0X respectively for values different than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed.
- 0 Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).

sprintf format width

- (number) Minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
- * The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

sprintf format .precision

- .number For integer specifiers (d, i, o, u, x, X) – precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E and f specifiers – this is the number of digits to be printed after the decimal point. For g and G specifiers – This is the maximum number of significant digits to be printed. For s – this is the maximum number of characters to be printed. By default all characters are printed until the ending null character is encountered. For c type – it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed.
- .* The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

sprintf format length

- h The argument is interpreted as a short int or unsigned short int (only applies to integer specifiers: i, d, o, u, x and X).
- l The argument is interpreted as a long int or unsigned long int for integer specifiers (i, d, o, u, x and X), and as a wide character or wide character string for specifiers c and s.
- L The argument is interpreted as a long double (only applies to floating point specifiers – e, E, f, g and G).

example code : sprintf(…)

```
#include <cstdio>
#include <limits>
#include <cstdint>
#include <cinttypes>

int main()
{
    char  b[500];
    std::sprintf(b, ”Strings:\n”); cout<<b;

    const char* s = "Hello";
    std::sprintf(b,”\t[%10s]\n\t[%-10s]\n\t[%*s]\n\t[%-10.*s]\n\t[%-*.*s]\n",
        s, s, 10, s, 4, s, 10, 4, s); cout<<b;
    std::sprintf(b, ”Characters:\t%c %%\n", 65); cout<<b;

    std::sprintf(b,“Integers\n”); cout<<b;
    std::sprintf(b,”Decimal:\t%i %d %.6i %i %.0i %+i %u\n", 1, 2, 3, 0, 0, 4, -1); cout<<b;
    std::sprintf(b,”Hexadecimal:\t%x %x %X %#x\n", 5, 10, 10, 6); cout<<b;
    std::sprintf(b,”Octal:\t%o %#o %#o\n", 10, 10, 4);  cout<<b;

    std::sprintf(b,”Floating point\n"); cout<<b;
    std::sprintf(b,”Rounding:\t%f %.0f %.32f\n", 1.5, 1.5, 1.5);  cout<<b;
    std::sprintf(b,”Padding:\t%05.2f %.2f %5.2f\n", 1.5, 1.5, 1.5);  cout<<b;
    std::sprintf(b,”Scientific:\t%E %e\n", 1.5, 1.5);  cout<<b;
    std::sprintf(b,”Hexadecimal:\t%a %A\n", 1.5, 1.5);  cout<<b;
    std::sprintf(b,”Special values:\t0/0=%g 1/0=%g\n", 0.0/0.0, 1.0/0.0);  cout<<b;

    std::sprintf(b,”Variable width control:\n”);  cout<<b;
    std::sprintf(b,”right-justified variable width: '%*c'\n", 5, ‘x');  cout<<b;
    int r = std::sprintf(b,”left-justified variable width : '%*c'\n", -5, ‘x’);  cout<<b;
    std::sprintf(b,”(the last printf printed %d characters)\n", r);  cout<<b;
}
```

Output

```
Strings:
        [     Hello]
        [Hello     ]
        [     Hello]
        [Hell      ]
        [Hell      ]
Characters:     A %
Integers
Decimal:        1 2 000003 0  +4 4294967295
Hexadecimal:    5 a A 0x6
Octal:  12 012 04
Floating point
Rounding:       1.500000 2 1.3000000000000000444089209850062 6
Padding:        01.50 1.50  1.50
Scientific:     1.500000E+00 1.500000e+00
Hexadecimal:    0x1.8p+0 0X1.8P+0
Special values: 0/0=nan 1/0=inf
Variable width control:
right-justified variable width: '    x'
left-justified variable width : 'x    '
(the last printf printed 40 characters)
```

Q & A