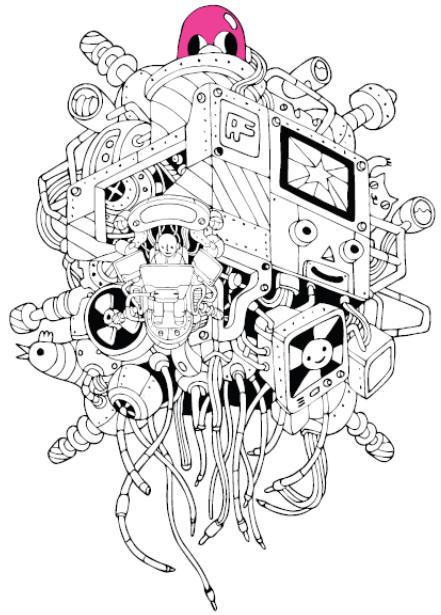
프로그래밍 언어 I



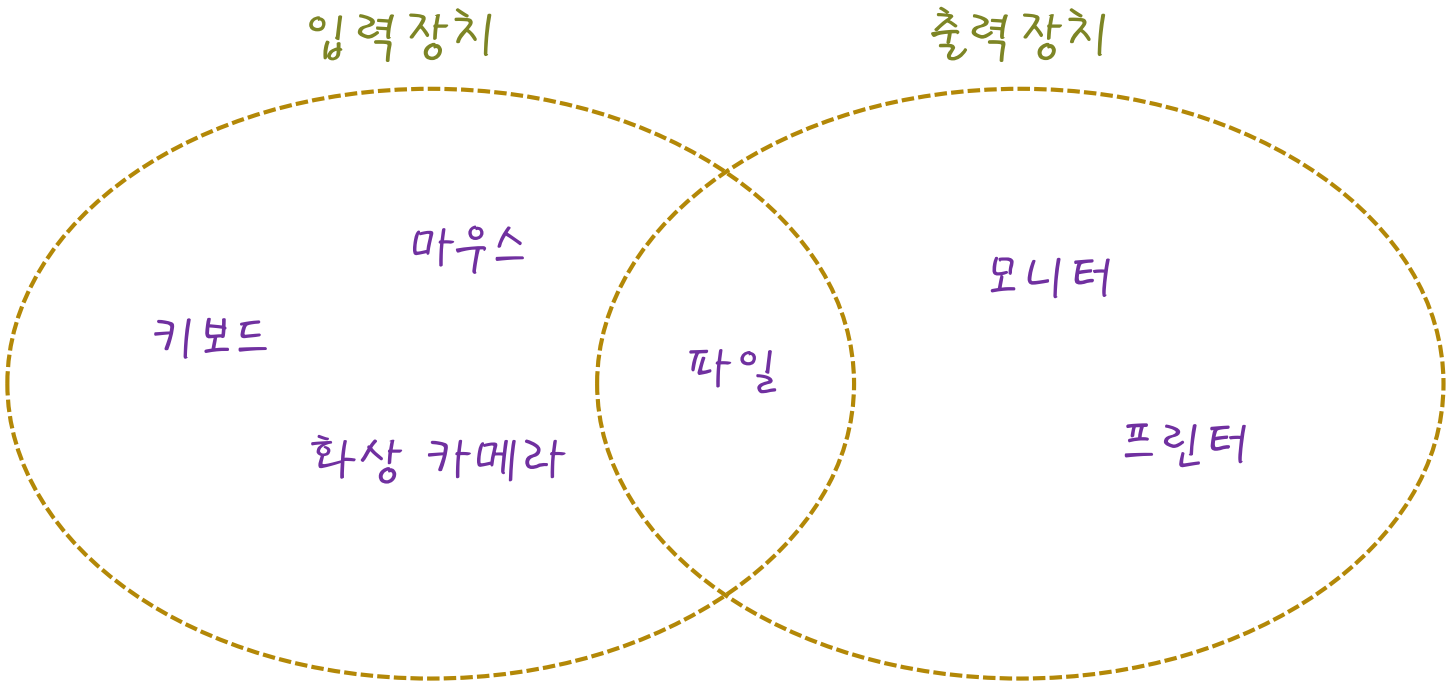
Chapter 21. 문자와 문자열 관련 함수

프로그래밍 언어 I

Chapter 21-1. 스트림과 데이터의 이동

Chapter 21. 문자와 문자열 관련 함수

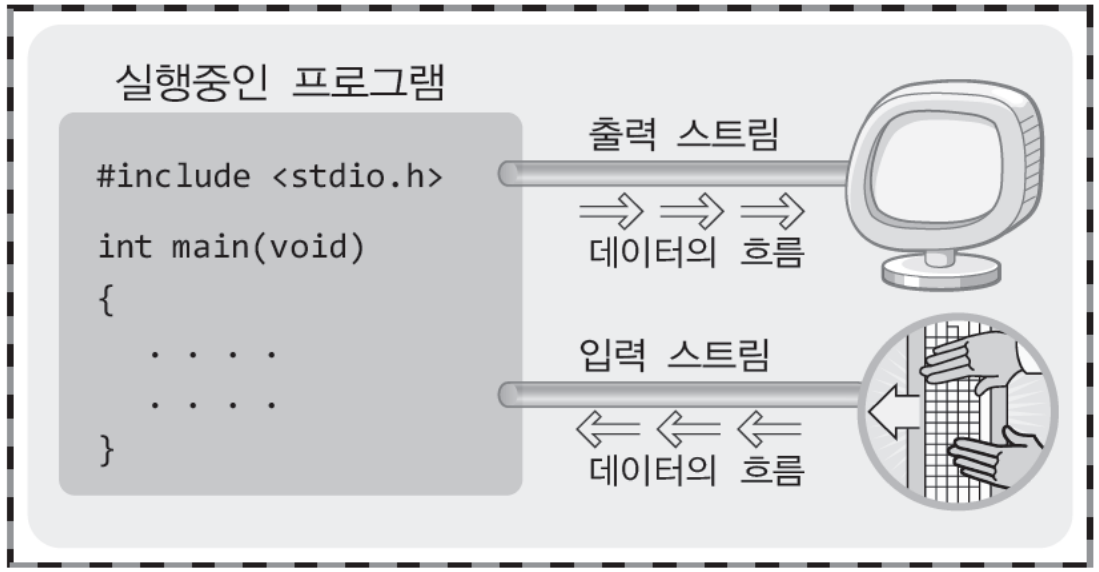
무엇이 입력이고 무엇이 출력인가



입출력 장치는 매우 포괄적이다.

데이터를 컴퓨터 내부로 받아들이는 것이 입력이고 외부로 전송하는 것이 출력이다.

데이터의 이동수단이 되는 스트림(Stream)



콘솔 입출력을 위한 스트림은 프로그램이 시작되면 OS에 의해서 자동으로 생성된다.

✈ 데이터의 입출력이 가능한 이유!

출력의 경로가 되는 출력 스트림과 입력의 경로가 되는 입력 스트림이 존재하기 때문

✈ 입출력 스트림이란?

OS가 데이터의 입출력을 위해 놓아주는 소프트웨어적인 형태의 다리!

스트림의 생성과 소멸

• stdin	표준 입력 스트림	키보드 대상으로 입력
• stdout	표준 출력 스트림	모니터 대상으로 출력
• stderr	표준 에러 스트림	모니터 대상으로 출력

√ **stdin**과 **stdout**은 각각 **표준 입력 스트림**과 **표준 출력 스트림을** 의미하는 이름들이다.

√ **stderr**은 **표준 에러 스트림**이라 하며, 출력의 대상은 **stdout**과 마찬가지로 모니터이다.

√ 출력 리다이렉션이라는 것을 통해서 **stdout**과 **stderr**이 향하는 데이터 전송의 방향을 각각 달리 할 수 있다.

√ **stdin**, **stdout**, **stderr**은 모두 **프로그램 시작과 동시에 자동으로 형성되고 프로그램 종료시 자동으로 소멸된다.**

√ **이외의 스트림들은 프로그래머가 직접 형성**해야 한다. 예를 들어 파일 입출력을 위한 스트림은 직접 형성해야 한다.

스트림이라 불리는 이유는 데이터의 이동을 한 방향으로만 형성하기 때문이다.

물이 한 방향으로 흐르듯 스트림도(스트림은 물의 흐름을 의미함) 한 방향으로만

데이터가 이동한다.

프로그래밍 언어 I

Chapter 21-2. 문자 단위 입출력 함수

Chapter 21. 문자와 문자열 관련 함수

문자 출력 함수

√ 하나의 문자를 출력하는 두 함수

```c
#include <stdio.h>

int putchar(int ch);
// 인자로 전달된 문자를 표준출력장치(stdout)에 출력.
// 반환값: 함수호출 성공시 쓰여진 문자
//         실패하면 EOF(-1) 반환

int fputc(int ch, FILE* stream);
// 두 번째 인자를 통해 출력할 대상을 지정
// 두 번째 인자로 stdout이 전달되면 putchar() 함수와 동일한 결과를 얻는다.
// 반환값: 함수호출 성공시 쓰여진 문자
//         실패하면 EOF(-1) 반환
```

문자 입력 함수

√ 하나의 문자를 입력 받는 두 함수

```
#include <stdio.h>

int getchar(void);
// 표준입력장치(stdin)로부터 입력된 문자를 반환

int fgetc(FILE* stream);
// 지정된 stream으로부터 한 문자를 읽어서 반환
// 입력 받을 대상을 인자로 전달
// 반환값: 성공시 읽은 문자를 int형으로 변환하여 반환
//         파일의 끝에 도착하거나 실패하면 EOF(-1) 반환
```

getchar() 함수와 fgetc() 함수의 관계는 putchar() 함수와 fputc() 함수의 관계와 같다.

문자 입출력 관련 예제

```c
#include <stdio.h>
int main(void)
{
    char ch1, ch2;

    printf("문자 하나를 입력하고 <엔터> 키를 누르시오: ");
    ch1 = getchar();
    ch2 = fgetc(stdin);

    putchar(ch1);
    fputc(ch2, stdout);

    printf("ch1의 ASCII code=%d, ch2의 ASCII code=%d\n", ch1, ch2);
    return 0;
}
```

char_io

```
문자 하나를 입력하고 <엔터> 키를 누르시 오: P Enter
P
ch1의 ASCII code=80, ch2의 ASCII code=10
```

실행결과

문자를 int형 변수에 저장하는 이유는 EOF를 설명하면서 함께 설명한다.

문자 입출력에서의 EOF

√ EOF의 의미

- EOF는 End Of File의 약자로서, 파일의 끝을 표현하기 위해서 정의해 놓은 상수이다.
- 파일을 대상으로 fgetc() 함수가 호출되었을 때 파일에 끝에 도달을 하면 EOF가 반환된다.

√ 콘솔 대상의 fgetc(), getchar() 함수호출로 EOF를 반환하는 경우

- 함수호출의 실패
- Windows에서 Ctrl+Z 키, Linux에서 Ctrl+D 키가 입력이 되는 경우

문자 입출력에서의 EOF

eof

```c
#include <stdio.h>

int main(void)
{
    char ch;

    while (1)
    {
        ch = getchar();
        if (ch == EOF)
            break;
        putchar(ch);
    }

    return 0;
}
```

키보드에는 EOF가 존재하지 않는다.

따라서 EOF를 Ctrl+Z 키와 Ctrl+D 키로 약속해 놓은 것이다.

예제에서 보이듯이. 하나의 문장이 입력되어도 문장을 이루는 모든 문자들이 반복된 getchar 함수의 호출을 통해서 입력될 수 있다.

실행결과

```
Hi~ Enter
Hi~
I like BME. Enter
I like BME.
^Z Enter
```

반환형이 int이고, int형 변수에 문자를 담는 이유는?

```
int getchar(void);
int fgetc(FILE * stream);
```

√ 반환형이 char형이 아닌 int형인 이유는?

- char형은 예외적으로 signed char가 아닌 unsigned char로 표현하는 컴파일러가 존재한다.
- 파일의 끝에 도달했을 때 반환하는 EOF는 -1로 정의되어 있다.
- char를 unsigned char로 표현하는 컴파일러는 EOF에 해당하는 -1을 반환하지 못한다.
- int는 모든 컴파일러가 signed int로 처리한다. 따라서 -1의 반환에 무리가 없다.

프로그래밍 언어 I

Chapter 21-3. 문자열 단위 입출력 함수

Chapter 21. 문자와 문자열 관련 함수

문자열 출력 함수: puts(), fputs()

```c
#include <stdio.h>

int puts(const char* str);
// str이 가리키는 문자열을 표준출력장치(stdout)에 쓴 후, 개행문자(\n)를 추가한다.
// 반환값: 성공시 양수를 반환.
           실패시 EOF 반환.

int fputs(const char* str, FILE* stream);
// str이 가리키는 문자열을 stream에 쓴다. 개행문자(\n)를 추가하지 않음.
// 반환값: 성공시 양수를 반환.
           실패시 EOF 반환.
```

문자열 출력 함수: puts(), fputs()

puts_fputs

```c
#include <stdio.h>

int main(void)
{
    char *str1 = "Biomedical Engineering";
    char* str2 = "Inje University";

    puts(str1);
    puts(str2);

    fputs(str1, stdout);
    fputs(str2, stdout);
    putchar('\n');
    fputs(str1, stdout);

    return 0;
}
```

실행결과

```
Biomedical Engineering
Inje University
Biomedical EngineeringInje University
Biomedical Engineering
```

puts 함수가 호출되면 문자열 출력 후 자동으로 개행이 이뤄지지만, fputs 함수가 호출되면 문자열 출력 후 자동으로 개행이 이뤄지지 않는다는 사실에 주목!

문자열 입력 함수: gets(), fgets()

```
char *gets(char *str);
```

- 표준입력장치(stdin)으로부터 개행문자나 EOF를 만날 때까지 문자들을 읽어서 str이 가리키는 곳에 문자열 형식으로 저장.
- 개행문자는 저장되지 않으며, 문자열의 맨 끝에 널 문자('\0')를 추가함.
- 반환값: 성공시 str을 반환
  실패시 NULL 포인터가 반환됨.

```
char* fgets(char* str, int n, FILE* stream);
```

- stream으로부터 문자들을 읽어서 str이 가리키는 곳에 문자열 형식으로 저장하되 다음 중 어느 한 가지 조건이라도 만나면 즉시 리턴한다.
  1. n-1개의 문자를 읽었을 때
  2. 개행문자를 만났을 때
  3. EOF를 만났을 때
- 개행문자는 읽어서 저장함.
- 문자열의 맨 끝에 널 문자('\0')를 추가함.
- 반환값: 성공시 str을 반환
  실패시 NULL 포인터가 반환됨.

문자열 입력 함수: gets, fgets

```
int main(void)
{
   char str[7];
   gets(str);
}
```

이 경우 입력되는 문자열의 길이가 배열을 넘어설 경우
할당 받지 않은 메모리를 참조하는 오류가 발생한다.

```
int main(void)
{
   char str[7];
   fgets(str, sizeof(str), stdin);
}
```

stdin으로부터 문자열을 입력 받아서 str에 저장을 하되, 널 문자를 포함하여 sizeof(str)의 크기만큼 저장을 해라.

fgets 함수의 호출의 예

fgets

```c
#include <stdio.h>

int main(void)
{
   char str[7];
   int i;

   printf("문자열을 입력하시오: ");

   for (i = 0; i < 3; i++)
   {
      fgets(str, sizeof(str), stdin);
      printf("Read %d: %s\n", i + 1, str);
   }

   return 0;
}
```

실행결과/

```
문자열을 입력하시오: 12345678901234567890 Enter
Read 1: 123456
Read 2: 789012
Read 3: 345678
```

6개의 문자씩 끊어서 읽히고 있다.

즉, 한번의 fgets 함수호출당 최대 6개의 문자만 읽혀진다.

fgets 함수의 호출의 예

```c
#include <stdio.h>

int main(void)
{
   char str[7];
   int i;

   printf("문자열을 입력하시오: ");

   for (i = 0; i < 3; i++)
   {
      fgets(str, sizeof(str), stdin);
      printf("Read %d: %s\n", i + 1, str);
   }

   return 0;
}
```

fgets

실행결과2

```
문자열을 입력하시오: We Enter
Read 1: We

are Enter
Read 2: are

BME students. Enter
Read 3: BME st
```

엔터 키의 입력도 문자열의 일부로 받아 들임을 보임

공백을 포함하는 문자열을 읽어 들임을 보임

프로그래밍 언어 I

Chapter 21-4. 표준 입출력 버퍼

Chapter 21. 문자와 문자열 관련 함수

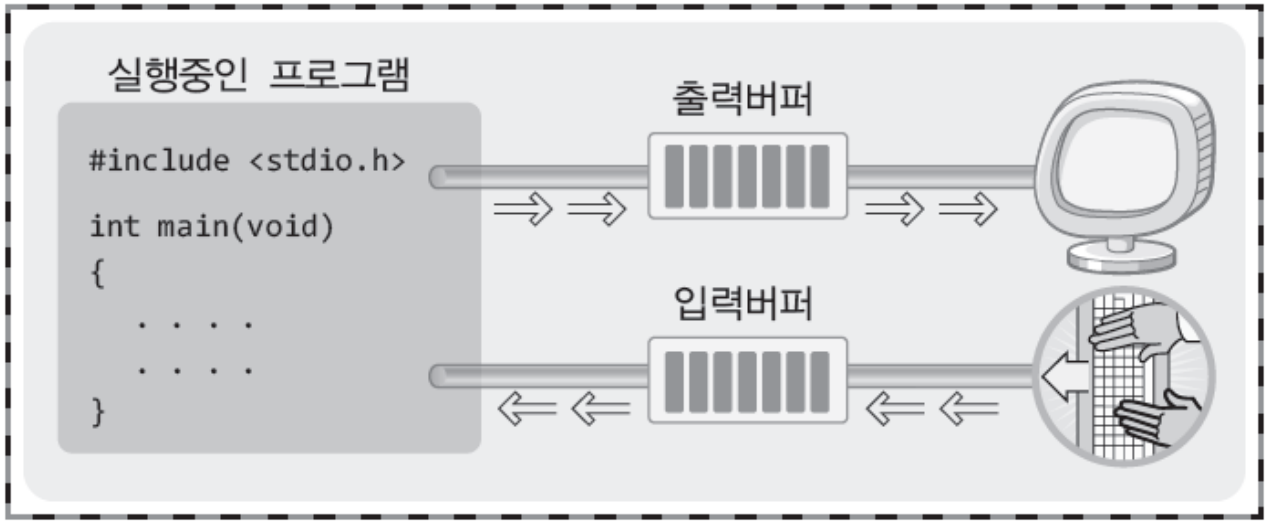
표준 입출력 기반의 버퍼와 버퍼링의 이유

√ 입출력 버퍼

- 버퍼는 특정 크기의 메모리 공간을 의미한다.
- 운영체제는 입력과 출력을 돕는 입출력 버퍼를 생성하여 제공한다.
- 표준 입출력 함수를 기반으로 데이터 입출력 시 입출력 버퍼를 거친다.

√ 입출력 버퍼에 데이터가 전송되는 시점

- 호출된 출력함수가 반환이 되는 시점이 출력버퍼로 데이터가 완전히 전송된 시점이다.
- 엔터를 입력하는 시점이 키보드로 입력된 데이터가 입력버퍼로 전달되는 시점이다.



버퍼링을 하는 이유는 데이터 이동의 효율과 관련이 있다. 데이터를 모아서 전송하면, 하나씩 전송하는 것보다 효율적이다.

출력버퍼를 비우는 fflush() 함수

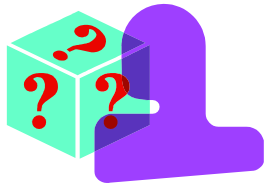
```
int fflush(FILE* stream);
```

- stream이 쓰기 위해 열려 있을 때, 출력 버퍼에 써 넣지 않은 데이터가 있으면 즉시 써 넣는다.
- 만일 stream이 NULL 포인터이면 쓰기 위해 열려 있는 모든 스트림에 대해 위의 동작이 수행된다.
- 반환값: 성공시 0을 반환
  실패시 EOF를 반환

▶ 인자에 해당하는 출력버퍼를 비운다.

출력버퍼를 비운다는 것은 출력버퍼에 저장된 데이터를 지우는 것이 아니라,

출력버퍼에 저장된 데이터를 목적지로 최종 전송함을 뜻한다.



출력버퍼의 경우와 달리 **입력버퍼의 비움은 입력버퍼에 저장된 데이터의 소멸**을 뜻한다.

그리고 fflush 함수는 **출력버퍼를 대상으로 정의된 함수**이다. 따라서 **fflush(stdin)**과

같은 형태의 함수호출은 그 결과를 보장받지 못한다.

그렇다면 입력버퍼는 어떻게 비워야 할까?

입력버퍼는 어떻게 비워야 하나요?

no_flush_input_buffer

```c
#include <stdio.h>

int main(void)
{
    char id[7];// 주민번호 앞 6자리 저장용
    char name[10];// 이름 저장

    fputs("주민번호 앞 6자리 입력: ", stdout);
    fgets(id, sizeof(id), stdin);

    fputs("이름 입력: ", stdout);
    fgets(name, sizeof(name), stdin);

    printf("주민번호: %s\n", id);
    printf("이름: %s\n", name);
    return 0;
}
```

실행결과1

```
주민번호 앞 6자리 입력:123456
이름 입력:주민번호:123456
이름:
```

엔터 키가 남아서 문제가 되는 상황

실행결과2

```
주민번호 앞 6자리 입력:123456-1234567
이름 입력:주민번호:123456
이름:-1234567
```

말 안 듣는 사람들 때문에 문제되는 상황

입력버퍼는 어떻게 비워야 하나요?

flush_input_buffer

```c
void flush_input_buffer(void)
{
    while (getchar() != '\n');
}

int main(void)
{
    char id[7];    // 주민번호 앞 6자리 저장용
    char name[10]; // 이름 저장

    fputs("주민번호 앞 6자리 입력: ", stdout);
    fgets(id, sizeof(id), stdin);

    flush_input_buffer(); // flush input buffer

    fputs("이름 입력: ", stdout);
    fgets(name, sizeof(name), stdin);

    printf("주민번호: %s\n", id);
    printf("이름: %s\n", name);
    return 0;
```

실행결과1

```
주민번호 앞 6자리 입력: 123456-1234567
이름 입력: 홍길동
주민번호: 123456
이름: 홍길동
```

어떠한 경우에도 주민번호 6자리만 입력 받도록 재 구현된 예제

실행결과2

```
주민번호 앞 6자리 입력: 123456
이름 입력: 홍길동
주민번호: 123456
이름: 홍길동
```

프로그래밍 언어 I

Chapter 21-5. 입출력 이외의 문자열 관련 함수

Chapter 21. 문자와 문자열 관련 함수

문자열의 길이를 반환하는 함수: strlen()

```
#include <string.h>
size_t strlen(const char* str);  // 문자열 str의 길이를 반환. 널 문자는 포함하지 않음.
```

size_t의 일반적인 선언

```
typedef unsigned int size_t;
```

typedef에 관해서는 후에 설명

```
int main(void)
{
   char str[] = "1234567";
   printf("%u\n", strlen(str));      // 문자열 str의 길이 7이 출력된다.
}
```

문자열의 길이를 반환하는 함수: strlen()

마지막에 삽입되는 널 문자를 없애는 예제

strlen

```c
#include <stdio.h>
#include <string.h>

int main(void)
{
    char str[100];
    int len1, len2, len3;

    printf("문자열을 입력하시오: ");
    fgets(str, sizeof(str), stdin);

    len1 = strlen(str);
    printf("입력한 문자열의 길이: %d, 내용: %s\n", len1, str);

    str[len1 - 1] = '\0';  // 맨 뒤의 개행문자('\n') 제거

    len2 = strlen(str);
    printf("개행문자를 제거한 문자열의 길이: %d, 내용: %s\n", len2, str);

    len3 = sizeof(str);
    printf("배열 str의 크기: %d 바이트\n", len3);

    return 0;
}
```

실행결과

```
문자열을 입력하시오: INJE BME
입력한 문자열의 길이: 9, 내용: INJE BME

개행문자를 제거한 문자열의 길이: 8, 내용: INJE BME
배열 str의 크기: 100 바이트
```

문자열을 복사하는 함수들: strcpy(), strncpy()

```
#include <string.h>
char* strcpy(char* destination, const char* source);
```

- source가 가리키는 문자열(널 문자 포함)을 destination이 가리키는 배열에 복사함.
- destination이 가리키는 배열은 source가 가리키는 문자열(널 문자 포함)을 포함할 수 있도록 커야함.

```
char* strncpy(char* destination, const char* source, size_t num);
```

- source가 가리키는 문자열의 처음 num개의 문자만을 destination이 가리키는 배열에 복사함.
- 만일 source가 가리키는 문자열의 길이가 num보다 길면 destination에 널 문자가 추가되지 않음.
- 만일 source가 가리키는 문자열의 길이가 num보다 짧으면 num개의 문자를 복사한 후에 destination이 가리키는 배열의 나머지는 모두 0으로 채움.

- 반환값: 복사된 문자열의 시작 번지(destination)가 반환됨.

문자열을 복사하는 함수들: strcpy(), strncpy()

```c
#include <stdio.h>
int main(void)
{
   char str1[30] = "Inje BME";
   char str2[30];

   strcpy(str2, str1);

   return 0;
}
```

str1에 저장된 문자열을 str2에 단순히 복사!

strcpy 함수를 호출하는 경우 배열의 범위를 넘어서 복사가 진행될 위험이 있다.

```c
#include <stdio.h>
int main(void)
{
   char str1[30] = "Inje BME";
   char str2[30];

   strncpy(str2, str1, sizeof(str2));

   return 0;
}
```

str1에 저장된 문자열을 str2에 복사하되 최대 sizeof(str2)의 반환 값 크기만큼 복사한다.

strncpy() 함수를 잘못 사용한 예

strncpy

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

int main(void)
{
   char str1[20] = "1234567890";
   char str2[20];
   char str3[5];

   /* case 1 */
   strcpy(str2, str1);
   puts(str2);

   /* case 2 */
   strncpy(str3, str1, sizeof(str3));
   puts(str3);

   /* case 3 */
   strncpy(str3, str1, sizeof(str3)-1);
   str3[sizeof(str3) - 1] = '\0';
   puts(str3);
   return 0;
}
```

실행결과

```
1234567890
12345傚傚傚傚傚?234567890
1234
```

두 번째 strncpy 함수호출 후의 결과에 이상이 보이는 이유는 복사하는 과정에서 문자열의 끝을 의미하는 널 문자가 복사되지 않았기 때문이다. 문자열을 복사할 때에는 항상 널 문자의 복사까지 고려해야 한다.

문자열을 덧붙이는 함수들: strcat(), strncat()

```
#include <string.h>

char* strcat(char* destination, const char* source);
```

- source가 가리키는 문자열을 destination이 가리키는 문자열 뒤에 추가함.
- destination이 가리키는 원래의 문자열 맨 뒤의 널 문자는 source가 가리키는 문자열의 첫 문자로 덮어 써지며, destination이 가리키는 문자열 맨 뒤에 널 문자가 새로이 추가됨.
- destination이 가리키는 배열은 source가 가리키는 문자열을 추가할 수 있도록 충분히 커야 함.

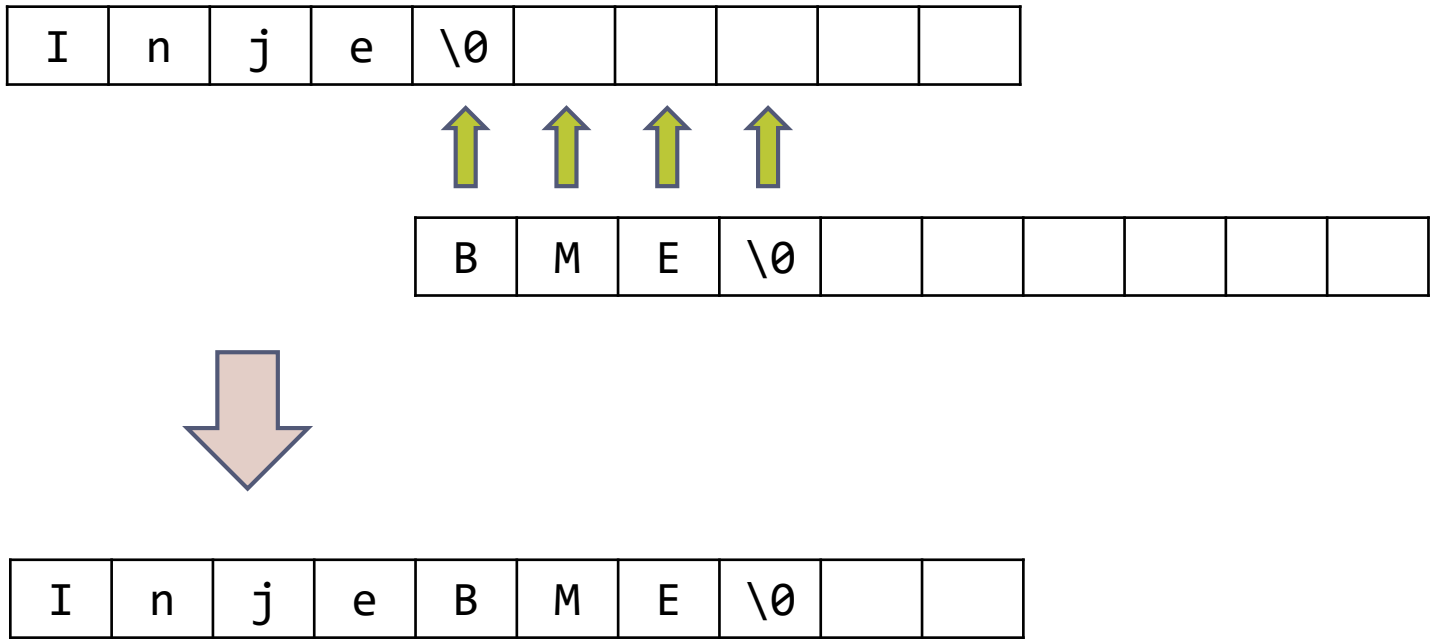
```
char* strncat(char* destination, const char* source, size_t num);
```

- source가 가리키는 문자열의 처음 num개의 문자만을 destination이 가리키는 문자열 뒤에 추가함.
- destination이 가리키는 원래의 문자열 맨 뒤의 널 문자는 source가 가리키는 문자열의 첫 문자로 덮어 써지며, destination이 가리키는 문자열 맨 뒤에 널 문자가 새로이 추가됨.
- destination이 가리키는 배열은 source가 가리키는 문자열을 추가할 수 있도록 충분히 커야 함.

- 반환값: 덧붙여진 문자열(destination)의 시작 번지

문자열을 덧붙이는 함수들: strcat(), strncat()

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

int main(void)
{
    char str1[10] = "Inje";
    char str2[10] = "BME";

    strcat(str1, str2);
    puts(str1);

    return 0;
}
```



문자열을 덧붙이는 함수들: strcat(), strncat()

strcat

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

int main(void)
{
    char str1[10] = "Inje";
    char *str2 = "BME";
    char str3[20] = "Bio";
    char str4[] = "1234567890";

    /* case 1 */
    strcat(str1, str2);
    puts(str1);

    /* case 2 */
    strncat(str3, str4, 7);
    puts(str3);

    return 0;
}
```

실행결과

```
InjeBME
Bio1234567
```

문자열을 비교하는 함수들: strcmp(), strncmp()

```
#include <string.h>

int strcmp(const char* str1, const char* str2);
```

- str1이 가리키는 문자열의 첫 글자와 str2가 가리키는 문자열의 첫 글자와의 비교를 시작한다.
- 이 비교는 두 문자가 서로 다르거나 널 문자를 만나면 끝난다.

```
int strncmp(const char* str1, const char* str2, size_t num);
```

- str1이 가리키는 문자열의 첫 글자와 str2가 가리키는 문자열의 첫 글자와의 비교를 시작한다.
- 이 비교는 다음 세 가지 조건 중 어느 하나를 만나면 끝난다.
  1. num개의 문자 비교가 끝나기 전에 두 문자열의 비교대상 문자가 서로 다를 때
  2. num개의 문자 비교가 끝나기 전에 널 문자를 만날 때
  3. num개까지의 비교대상 문자가 같을 때

- 반환값
  - ✓ 음수: str1이 가리키는 문자의 ASCII code가 str2가 가리키는 문자의 ASCII code보다 작을 때
  - ✓ 0: 두 문자열이 일치할 때.
  - ✓ 양수: str1이 가리키는 문자의 ASCII code가 str2가 가리키는 문자의 ASCII code보다 클 때

문자열 비교의 예

strcmp

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

int main(void)
{
    char *str1  = "Simple";
    char str2[] = "Simon";

    if (!strcmp(str1, str2))
        puts("두 문자열은 같습니다.\n");
    else
    {
        puts("두 문자열은 서로 다릅니다.\n");

        if (!strncmp(str1, str2, 3))
            puts("그러나 처음 세 문자는 같습니다.");
    }
    return 0;
}
```

실행결과

```
두 문자열은 서로 다릅니다.

그러나 처음 세 문자는 같습니다.
```

그 이외의 변환함수들

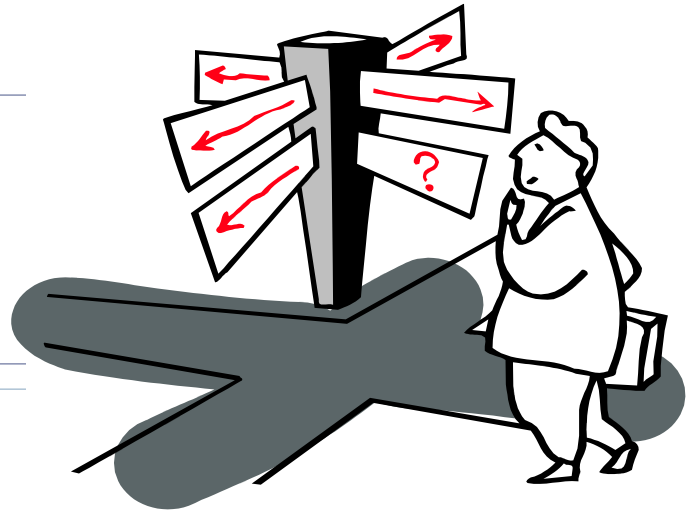
```
#include <stdlib.h>          헤더파일 stdlib.h에 선언

int atoi(const char* str);
// str로 주어진 문자열을 10진 정수로 간주하여 int형으로 변환.
// str이 가리키는 문자열의 앞 부분에 있는 공백 문자는 모두 건너뛰고 변환.
// 만일 변환 대상인 문자열 뒤에 공백문자로 구분된 또 다른 문자열이 있으면 이는 무시됨.
// str이 가리키는 문자열의 처음부터 변환할 수 없는 문자가 오면 0이 반환됨.

long int atol(const char* str);
// long int형으로 변환한다는 점 이외에는 atoi()와 같음

double atof(const char* str);
// double 형으로 변환한다는 점 이외에는 atoi()와 같음
```

그 이외의 변환함수들

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   char *stri1 = "1234";
   char *stri2 = "  -1234,-5678";
   char *stri3 = "ABC1234";
   char *strl = "1234567890";
   char *strf1 = "1234.5678";
   char *strf2 = "-1.2345678e-2";

   int    numi1 = atoi(stri1);
   int    numi2 = atoi(stri2);
   int    numi3 = atoi(stri3);
   long   numl  = atol(strl);
   double numf1 = atof(strf1);
   double numf2 = atof(strf2);

   printf("numi1 = %d\n", numi1);
   printf("numi2 = %d\n", numi2);
   printf("numi3 = %d\n", numi3);
   printf("numl  = %d\n", numl);
   printf("numf1 = %f\n", numf1);
   printf("numf2 = %f\n", numf2);

   return 0;
}
```

```
numi1 = 1234
numi2 = -1234
numi3 = 0
numl  = 1234567890
numf1 = 1234.567800
numf2 = -0.012346
```

실행결과



Chapter 21이 끝났습니다. 질문 있으신지요?