# Linux System Programming #4-2

**Spring 2020**

**School of Computer Science and Engineering,**

**Soongsil University, Seoul, Korea**

**Jiman Hong**

**jiman@acm.org**

# getc(3), fgetc(3), getchar(3)

```
#include <stdio.h>
int getc(FILE *fp);
int fgetc(FILE *fp);
int getchar(void);
```
리턴 값: 성공 시 다음 문자, 파일 끝이나 에러 시 EOF

- **한 번에 하나의 문자를 읽는 라이브러리 함수**
  - getc() / fgetc() : 주어진 *fp 파일 스트림형 포인터에서 한 문자씩 (파일 스트림이 가리키는 바로 다음 문자를) 읽고 원래 데이터 타입인 unsigned char를 int로 변환한 다음 리턴 값으로 리턴 => 파일 스트림의 버퍼에서 맨 앞의 문자의 ASCII 코드 값을 리턴
    - 파일의 끝에 도달했을 경우에는 EOF를 리턴
    - fgetc()는 함수로 구현
  - getchar() : stdin으로부터 문자를 하나 입력 받는 함수
    - == fgetc(stdin)
    - unsigned char 형을 int 형으로 변환하여 리턴, 파일의 끝에 도달했을 경우에는 EOF를 리턴

# getc() 예제

<ssu_getc.c>

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

int main(void)
{
   int character;
   while ((character = getc(stdin)) != EOF)
      if (putc(character, stdout) == EOF) {
         fprintf(stderr, "standard output error\n");
         exit(1);
      }
   if (ferror(stdin)) {
      fprintf(stderr, "standard input error\n");
      exit(1);
   }
   exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./ssu_getc
Hello stdin, stdout
Hello stdin, stdout
```

# ferror(3), feof(3), clearerr(3)

```
#include <stdio.h>
int ferror(FILE *fp);
리턴 값: 에러가 없으면 0, 에러가 발생하면 0 이 아닌 값
int feof(FILE *fp);
리턴 값: 조건이 참이면 0이 아닌 값(true), 참이 아니면 0(false)
void clearerr(FILE *fp);
```

- **FILE 스트림 객체에 설정된 에러 플래그를 설정하거나 확인하는 라이브러리 함수**
  - ferror()
    - 입출력 에러 발생 여부 확인
    - 주어진 파일 스트림 FILE *fp의 읽기 또는 쓰기 시 에러를 검사. 에러가 발생하면 FILE *fp을 닫고 rewind()를 호출하거나 clearerr()를 호출할 때까지 FILE *fp의 에러 플래그 설정 (FILE 구조체 내 _flag에 저장_
  - feof()
    - EOF(-1)를 구분하지 않는 함수들을 위해 EOF가 리턴되는 이유가 에러인지 파일의 끝인지를 구분하는데 사용 (FILE 구조체 내 _flag에 저장) clearerr()
    - 인자로 지정된 파일 스트림 FILE *fp 의 에러 플래그와 EOF 플래그를 재설정
    - clearerr()를 호출하여 명시적으로 EOF 플래그를 크리어 해주지 않으면 파일의 끝 다음에 실제 입력된 내용이 있더라도 EOF 플래그가 유지됨
    - FILE 구조체 내 _flag에 저장된 에러나 -1을 초기화하는데 사용

# ferror() 예제

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

int main(int argc, char *argv[])
{
   FILE *fp;
   int character;
   if (argc != 2) {
      fprintf(stderr, "usage: %s <filename>\n", argv[0])
      exit(1);
   }
   if ((fp = fopen(argv[1], "r")) == NULL) {
      fprintf(stderr, "fopen error for %s\n", argv[1]);
      exit(1);
   }
   character = fgetc(fp);
   while (!feof(fp)) {
      fputc(character, stdout);
      if (ferror(fp)) {
         fprintf(stderr, "Error detected!!\n");
         clearerr(fp);
      }
   character = fgetc(fp);
   }

   fclose(fp);
   exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# cat ssu_test.txt
Linux System Programming!
Unix System Programming!
Linux Mania
Unix Mania
root@localhost:/home/oslab# ./ssu_ferror ssu_test.txt
Linux System Programming!
Unix System Programming!
Linux Mania
Unix Mania
```

# ungetc(3)

```
#include <stdio.h>
int ungetc(int c, FILE *fp);
리턴 값: 성공 시 c, 에러 시 EOF
```

- **한번 문자를 읽은 후 그 문자를 다시 스트림에 되돌려 놓는 라이브러리 함수**
- **주어진 문자를 unsigned char 형으로 입력 스트림에 리턴**
  - 단, 주어진 문자가 EOF이면 스트림에 리턴 불가
  - 리턴된 문자는 다음번 fgetc(), getc(), getchar() 가 호출될 때 다시 읽혀짐
- **ungetc() 호출로 인해 파일 자체의 변경은 없음**
  - 한 번에 한 문자만을 리턴
- **어떠한 조건을 검사하기 위해 하나의 문자를 여러 번 읽은 후, 조건 검사되기 이전의 상태로 복원하기 위해 사용**
  - => 여러번 ungetc() 호출하면 역순으로 출력

# ungetc() 예제

<ssu_expr.txt>
123+456*789

<ssu_ungetc.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
int main(void)
{
   char operator;
   FILE *fp;
   int character;
   int number = 0;
   if ((fp = fopen("ssu_expr.txt", "r")) == NULL) {
      fprintf(stderr, "fopen error for ssu_expr.txt\n");
      exit(1);
   }
   while (!feof(fp)) {
      while ((character = fgetc(fp)) != EOF && isdigit(character))
         number = 10 * number + character – 48;

     fprintf(stdout, " %d\n", number);
      number = 0;
   if (character != EOF) {
         ungetc(character, fp);
         operator = fgetc(fp);
         printf("Operator => %c\n", operator);
     }
   }
   fclose(fp);
   exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# vi ssu_expr.txt
root@localhost:/home/oslab# ./ssu_ungetc
 123
Operator => +
 456
Operator => *
 789
Operator =>
 0
```

# putc(3), fputc(3), putchar(3)

```
#include <stdio.h>
int putc(int c, FILE *fp);
int fputc(int c, FILE *fp);
int putchar(int c);
```

- **지정된 파일 스트림에 하나의 문자를 쓰는 기능을 수행하는 라이브러리 함수**
  - Int c 로 지정한 바이트를 unsigned char로 변환한 다음 지정한 파일 스트림으로 쓰는 함수
- **putc()**
  - 보통 매크로로 구현
- **fputc()**
  - 함수로 구현
- **putchar()**
  - 표준 출력으로 문자를 하나 출력

# fgets(3), gets(3)

```
#include <stdio.h>
char *fgets(char *buf, int n, FILE *fp);
char *gets(char *buf);
```
리턴 값: 성공 시 buf, 파일 끝이나 에러 시 NULL

- **줄 단위 입력에 사용되는 라이브러리**
  - 파일 스트림으로부터 입력된 문자열 입력
- **fgets()**
  - 개행 문자/파일의 끝(EOF)에 도달할 때까지 파일로부터 읽어 인자 buf에 저장하며 마지막은 항상 NULL 문자로 채움
    - 개행 문자는 버퍼에 그대로 보관
    - 문자열의 끝은 항상 NULL 문자로 끝나기 때문에 개행 문자를 포함하여 최대 n-1개의 문자를 한꺼번에 읽을 수 있음

    만약 한 라인이 n보다 길면 먼저 n-1개의 문자만 미리 읽고 fgets() 호출 시 나머지 부분을 읽음
- **gets()**
  - 준 입력으로부터 개행 문자/EOD 를 만날 때까지 문자를 읽고 buf에 저장
  - 사용자는 버퍼 크기를 직접 지정하지 못함
  - 개행 문자를 NULL로 대체하여 읽음
  - ANSI C gets()를 표준으로 규정
    - 단, 버퍼 크기를 사용자가 지정하지 못하기 때문에 라인의 길이가 버퍼 크기보다 크면 메모리에서 버퍼 영역을 넘어선 주소 공간에 데이터를 쓸 수 있음 => gets()를 사용하는 것보다 fgets(stdin, ...)를 사용

# fgets() 예제 1

<ssu_fgets_1.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define BUFFER_SIZE 1024

int main(void)
{
   char buf[BUFFER_SIZE];
   while (fgets(buf, BUFFER_SIZE, stdin) != NULL)
      if (fputs(buf, stdout) == EOF) {
         fprintf(stderr, "standard output error\n");
         exit(1);
      }

   if (ferror(stdin)) {
      fprintf(stderr, "standard input error\n");
      exit(1);
   }
   exit(0);

}
```

실행결과

```
root@localhost:/home/oslab# ./ssu_fgets_1

Hi fgets, fputs

Hi fgets, fputs
```

# fgets() 예제 2

<ssu_fgets_2.c>

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

#define BUFFER_MAX 256

int main(void)
{
   char command[BUFFER_MAX];
   char *prompt = "Prompt>>";

   while (1) {
      fputs(prompt, stdout);
      if (fgets(command, sizeof(command), stdin) == NULL)
         break;
      system(command);
   }

   fprintf(stdout, "Good bye...\n");
   fflush(stdout);
   exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./ssu_fgets_2
Prompt>>ls /
bin   dev  initrd.img    lib64  mnt   root  snap  tmp   vmlinuz
boot  etc  initrd.img.old  lost+found  opt  run  srv  usr  vmlinuz.old
cdrom  home  lib    media proc  sbin  sys   var

Prompt>>./ssu_fgets_2
Prompt>>Good bye...
Prompt>>Good bye...
```

# fputs(3), puts(3)

```
#include <stdio.h>
int fputs(const char *str, FILE *fp);
int puts(const char *str);
```
리턴 값: 성공 시 음이 아닌 값, 에러 시 EOF

- **줄 단위 출력에 사용되는 라이브러리**
  - 파일 스트림으로부터 문자열 출력
- **fputs()**
  - 인자로 지정된 FILE *fp에 인자 str이 가리키는 문자열에서 NULL 문자를 제외하고 출력
  - 개행 문자를 추가하지 않음
- **puts()**
  - 인자 str이 가리키는 문자열을 표준 출력 스트림인 stdout에 출력
  - 문자열의 끝인 NULL은 개행 문자로 대체

# fputs() 예제

```
<ssu_fputs.c>
#include <stdio.h>
#include <stdlib.h>
#define BUFFER_SIZE 1024
int main (int argc, char *argv[])
{
   char buf[BUFFER_SIZE];
   FILE  *fp;
   if (argc != 2) {
      fprintf(stderr, "usage: %s <file>\n", argv[0]);
      exit(1);
   }
   if ((fp = fopen(argv[1], "w+")) == NULL) {
      fprintf(stderr, "fopen error for %s\n", argv[1]);
      exit(1);
   }
   fputs("Input String >> ", stdout);
   gets(buf);
   fputs(buf, fp);
   rewind(fp);
   fgets(buf, sizeof(buf), fp);
   puts(buf);
   fclose(fp);
   exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./ssu_fputs  ssu_hello.txt

Input String >> Hello OSLAB~!

Hello OSLAB~!

root@localhost:/home/oslab# cat ssu_hello.txt

Hello OSLAB~!root@localhost:/home/oslab#
```

# fread(3), fwrite(3)

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t nobj, FILE *fp);
size_t fwrite(const void *ptr, size_t size, size_t nobj, FILE *fp);
```
리턴 값: 성공적으로 읽거나 쓴 객체들의 개수

- **이진 파일의 입출력을 위한 라이브러리 함수**
- **fread()**
  - size 크기를 갖는 데이터를 한 단위로 하여 nobj 인자만큼 읽도록 하는 함수
  - 읽을 파일은 NULL로 끝날 필요는 없음
  - 에러가 발생하였거나 파일의 끝에 도달하면 리턴
    - feof()와 ferror()를 호출
      - 함수 호출의 리턴 값이 읽을 데이터 객체 개수보다 적은 경우, 파일의 끝인지 또는 다른 에러인지 확인
  - 파일의 내용이 구조체로 되어 있는 경우 유용하게 사용 => sizeof 연산자
- **fwrite()**
  - size 크기만큼의 데이터를 nobj 인자만큼 출력

# fwrite() 예제 p.187

# ftell(3), fseek(3), rewind(3)

```
#include <stdio.h>
long ftell(FILE *fp);
리턴 값: 성공 시 파일의 현재 오프셋, 에러 시 –1L을 리턴하고 errno 설정
int fseek(FILE *fp, long offset, int whence);
리턴 값: 성공 시 0, 에러 시 0이 아닌 값
void rewind(FILE *fp);
```

- **표준 입출력 스트림의 읽기, 쓰기 및 위치 조회**
- **ftell()**
  - 파일 스트림의 현재 오프셋 위치가 파일의 시작부터 몇 바이트만큼 떨어져 있는지 알려주는 함수
  - 입출력 가능으로 오픈된 파일 스트림의 현재 오프셋 위치를 fseek()의 offset에 적합한 형태(long int)로 리턴
  - 에러 경우는 파일 스트림의 현재 오프셋 위치가 가리키고 있는 값이 너무 커서 long 형으로 표현할 수 없는 경우
- **fseek()**
  - 파일 입출력 시 파일 스트림의 현재 오프셋 위치를 재조정하여 지정한 위치로 이동시키는 함수
  - fseek()를 사용하기 전 반드시 파일이 오픈되어 있어야 함
  - 파일 스트림의 현재 오프셋 위치는 offset 인자 값과 인자 whence에 의해 설정
  - offset / whence 값은 저수준 입출력 함수의 lseek() 과 동일.
  - 저수준 함수인 lseek()와 유사
- **rewind():**
  - 파일 스트림의 현재 오프셋 위치를 맨 처음 위치로 설정하고 파일의 맨 처음으로 이동.

# fseek() 예제

<ssu_fseek.c>

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

int main(void)
{
   char *fname = "ssu_test.txt";
   FILE *fp;
   long position;
   int character;

   if ((fp = fopen(fname, "r")) == NULL) {
      fprintf(stderr, "fopen error for %s\n", fname);
      exit(1);
   }

   printf("Input number >>");
   scanf("%ld", &position);
   fseek(fp, position - 1, SEEK_SET);
   character = getc(fp);
   printf("%ldth character => %c\n", position, character);
   exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ls -al ssu_test.txt
-rw-rw-r-- 1 oslab oslab 74 Jan  9 18:48 ssu_test.txt
root@localhost:/home/oslab# ./ssu_ftell
The size of <ssu_test.txt> is 74 bytes
```

# ftell() 예제

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

int main(void)
{
   char *fname = "ssu_test.txt";
   FILE *fp;
   long fsize;

   if ((fp = fopen(fname, "r")) == NULL) {
      fprintf(stderr, "fopen error for %s\n", fname);
      exit(1);
   }

   fseek(fp, 0, SEEK_END);
   fsize = ftell(fp);
   printf("The size of <%s> is %ld bytes\n", fname, fsize);
   exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./ssu_fseek
Input number >>8
8th character => y
```

# ftello(3), fseeko(3)

```
#include <stdio.h>
off_t ftello(FILE *fp);
리턴 값: 성공 시 파일의 현재 오프셋, 에러 시 (off_t)(-1)을 리턴하고 errno가 설정됨
int fseeko(FILE *fp, off_t offset, int whence);
리턴 값: 성공 시 0, 에러 시 0이 아닌 값
```

- **ftello(), fseeko()**
  - 표준 입출력 스트림의 읽기, 쓰기 위치 조회에 사용되는 라이브러리 함수
  - ftello(), fseeko() = ftell(), fseek()

# fgetpos(3), fsetpos(3)

```
#include <stdio.h>
int fgetpos(FILE  *fp, fpos_t *pos);
```
리턴 값: 성공 시 0, 에러 시 0이 아닌 값
```
int fsetpos(FILE *fp, const fpos_t *pos);
```
리턴 값: 성공 시 0, 에러 시 0이 아닌 값을 리턴하고 errno가 설정됨

- **표준 입출력 스트림의 현재 오프셋 위치를 얻거나 지정할 수 있는 라이브러리 함수**
- **fgetpos()**
  - 파일 스트림의 현재 오프셋 위치를 인자 pos에 저장
- **fsetpos()**
  - 지정된 파일 스트림의 현재 오프셋 위치를 이전에 저장된 파일 스트림의 오프셋 위치로 되돌림

# printf(3), fprintf(3), sprintf(3), snprintf(3)

```
#include <stdio.h>
int printf(const char *format, ...);
int fprintf(FILE *fp, const char *format, ...);
리턴 값: 성공 시 출력된 문자 개수, 출력 에러 시 음의 값
int sprintf(char *buf, const char *format, ...);
int snprintf(char *buf, size_t n, const char *format, ...);
리턴 값: 성공 시 배열에 저장된 문자 개수, 부호화 에러 시 음의 값
```

- 서식화 된 문자들을 출력하는 기능을 가진 라이브러리 함수
- printf() : 표준 출력에 서식화 된 문자들을 데이터를 변환하여 출력하는 함수
- fprintf() : printf()와 기능은 동일. sprintf() : 파일이 아니라 인자 buf가 가리키는 문자 배열에 출력하는 함수. 마지막에는 '\0' 문자를 채워줌
- snprintf() : sprintf()에 2번째 인자로 size_t가 추가된 함수

# printf(), fprintf(), sprintf(), snprintf() 플래그

| 플래그 | 설명 |
|---|---|
| - | 출력을 필드 안에서 왼쪽으로 정렬 |
| + | 부호 있는 변환에서 항상 부호를 표시 |
| (공백) | 부호가 만들어지지 않은 경우 공백을 앞에 붙임 |
| # | 다른 형식을 이용해서 변환(ex:16진수 형식의 경우 0x) |
| 0 | 필드를 채울 때 빈칸 대신 0들을 앞에 붙임 |

# printf(), fprintf(), sprintf(), snprintf() 길이 수정자

| 길이 수정자 | 설명 |
|---|---|
| hh | 부호 있는/없는 char |
| h | 부호 있는/없는 short |
| l | 부호 있는/없는 long 또는 넓은 문자 |
| ll | 부호 있는/없는 long long |
| j | intmax_t 또는 uintmax_t |
| z | size_t |
| t | ptrdiff_t |
| L | long double |

# printf(), fprintf(), sprintf(), snprintf() 변환 형식

| 변환 형식 | 설명 |
|---|---|
| d, i | 부호 있는 십진수 |
| o | 부호 없는 8진수 |
| u | 부호 없는 십진수 |
| x, X | 부호 없는 16진수 |
| f, F | double 부동소수점 수 |
| e, E | 지수 형식의 double 부동소수점 수 |
| g, G | 변환되는 값에 따라 f나 F, e, E로 해석 |
| a, A | 16진 지수 형식의 double 부동소수점 수 |

| | |
|---|---|
| c | 문자 |
| s | 문자한열 |
| p | void를 가리키는 포인터 |
| n | 부호 있는 정수를 가리키는 포인터 |
| % | %문자 자체 |
| C | 넓은 문자 |
| S | 넓은 문자열 |

# scanf(3), fscanf(3), sscanf(3)

```
#include <stdio.h>
int scanf(const char *format, ...);
int fscanf(FILE *fp, const char *format, ...);
int sscanf(const char *buf, const char *format, ...);
```

리턴 값: 성공 시 배정된 입력 항목들의 개수 리턴, 어떠한 변화도 일어나기 전에 파일 끝에 도달한 경우에는 EOF 리턴, 읽기 에러 발생시 EOF를 리턴하고 errno가 설정됨

- **데이터를 format에서 지시하는 대로 서식화 변환하여 입력받는 라이브러리 함수**
- **scanf() <- stdin**
- **fscanf() <- fp가 가리키는 파일**
- **sscanf() <- 파일이 아닌 인자 buf가 가리키는 문자 배열에서 읽음**
- **format 서식(형식 지정자)**
  - 인자들이 어떻게 변환 배정되는지는 format으로 지정된 서식 문자열이 결정

# 변환 형식

| 변환 형식 | 설명 |
|---|---|
| d | 부호 있는 십진 정수 |
| i | 부호 있는 정수, 밑은 입력의 형식에 따라 달라짐 |
| o | 부호 없는 8진수 |
| u | 부호 없는 십진 정수 |
| x | 부호 없는 16진 정수 |
| a, A, e, E, f, F, g, G | 부동소수점 수 |
| c | 문자 |
| s | 문자열 |
| [ | ]까지 나열된 문자들 중 하나와 부합 |
| [^ | ]까지 나열된 문자열 이외의 한 문자와 부합 |
| P | void를 가리키는 포인터 |
| n | 부호 있는 정수를 가리키는 포인터 |
| % | %문자 자체 |
| C | 넓은 문자 |
| S | 넓은 문자열 |

# fscanf() 예제

<ssu_fscanf.c>

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

#define BUFFER_SIZE 1024

int main(void)
{
   char *fname = "ssu_test.dat";
   char name[BUFFER_SIZE];
   FILE *fp;
   int age;

   fp = fopen(fname, "r");
   fscanf(fp, "%s%d", name, &age);
   fclose(fp);
   fp = fopen(fname, "w");
   fprintf(fp, "%s is %d years old\n", name, age);
   fclose(fp);
   exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# cat > ssu_test.dat
HongGilDong 20
root@localhost:/home/oslab# ./ssu_fscanf
root@localhost:/home/oslab# cat ssu_test.dat
HongGilDong is 20 years old
```

# Fileno(3)

```
#include <stdio.h>
int fileno(FILE *fp);
```
리턴 값: 스트림에 연관된 파일 디스크립터 잘못된(invalid) 스트림일 경우 –1을 리턴하고 errno가 EBADF로 설정됨

- **데이터를 format에서 지시하는 대로 서식화 변환하여 입력받는 라이브러리 함수**
  - 파일 스트림에 연관된 파일 디스크립터를 얻을 때 사용하는 라이브러리 함수
  - dup(), fcntl() 등 파일 디스크립터를 인자로 하는 함수를 호출하는 경우 많이 사용

# tmpnam(3), tmpfile(3)

```
#include <stdio.h>
char *tmpnam(char *ptr);
리턴 값: 고유한 임시파일의 경로 이름을 가리키는 포인터고유한 이름을 더 만들 수 없는 경우 NULL 리턴
FILE *tmpfile(void);
리턴 값: 성공 시 파일 포인터, 에러 발생시 NULL을 리턴하고 errno가 설정됨
```

- **임시 파일을 생성하는 라이브러리 함수**
- **tmpnam()**
  - <stdio.h>에 정의되어 있는 P_tmpdir로 정의되어 있는 경로에 해당하는 접두어를 붙여 임시 파일 이름을 생성
  - 인자 ptr이 NULL인 경우 tmpnam()의 내부 버퍼를 이용해 그 포인터를 리턴
- **tmpfile()**
  - "w+" 모드로 임시 파일을 생성
  - 일반 파일과 달리 이 함수들에 의해 생성된 파일은 프로그램 수행 중에만 유효하며 파일을 닫거나 프로그램이 종료되면 자동으로 삭제

# ·tmpnam() 예제

<ssu_tmpnam.c>

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_LINE 4096
int main(void)
{
   char buf[MAX_LINE];
   char name[L_tmpnam];
   FILE *fp;
   printf("temp file 1 : %s\n", tmpnam(NULL));
   tmpnam(name);
   printf("temp file 2 : %s\n", name);
   if ((fp = tmpfile()) == NULL) {
      fprintf(stderr, "tmpfile error\n");
      exit(1);
   }
   fputs("tmpfile created temporary file.\n", fp);
   fseek(fp, 0, SEEK_SET);
   if (fgets(buf, sizeof(buf), fp) == NULL) {
      fprintf(stderr, "fgets error\n");
      exit(1);
   }
   fputs(buf, stdout);
   exit(0);
}
```

실행결과

```
oslab@localhost:~$ ./ssu_tmpnam
temp file 1 : /tmp/fileOj8S05
temp file 2 : /tmp/filefrLxrK
tmpfile created temporary file.
```

# ·tmpfile() 예제

<ssu_tmpfile.c>

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

#define BUFFER_SIZE 1024

int main(void)
{
   char buf[BUFFER_SIZE];
   char name[L_tmpnam];
   FILE *fp;

   printf("Temporary filename <<%s>>\n", tmpnam(name));

   if ((fp = tmpfile()) == NULL) {
      fprintf(stderr, "tmpfile create error!!\n");
      exit(1);
   }

   fputs("create tmpfile success!!\n", fp);
   rewind(fp);
   fgets(buf, sizeof(buf), fp);
   puts(buf);
   exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./ssu_tmpfile
Temporary filename <</tmp/files8NGKi>>
create tmpfile success!!
```

# tempnam(3)

```
#include <stdio.h>
char *tempnam(const char *directory, const char *prefix);
```
리턴 값: 고유한 경로 이름을 가리키는 포인터에러 발생시 NULL을 리턴하고 errno가 설정됨

- **임시 파일 이름을 생성하는 라이브러리 함수**
- **tempnam()**
  - tmpfile()와는 다르게 사용자가 임시 파일이 만들어질 디렉토리를 직접 지정 가능

# tempnam() 예제

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

int main(int argc, char *argv[])
{
   char *arg_directory = NULL;
   char *arg_prefix = NULL;

   if (argc != 3) {
      fprintf(stderr, "usage: %s <directory> <prefix>\n", argv[0]);
      exit(1);
   }

   arg_directory = argv[1][0] != ' ' ? argv[1] : NULL;
   arg_prefix = argv[2][0] != ' ' ? argv[2] : NULL;
   printf("created : %s\n", tempnam(arg_directory, arg_prefix));
   exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./ssu_tempnam " " PTF
created : /tmp/PTFyvoHBM
root@localhost:/home/oslab# ./ssu_tempnam /home Temp
created : /home/TempEPZ9AA
```