

시스템 프로그래밍 깃허브 및 명령어 50개

<https://github.com/notecoding/system>

2021963016 김용효

시스템 프로그래밍 명령어 50개
명령어43 + 옵션7 = 50

basename.c

```
// basename.c - 파일 경로에서 파일명만 추출
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    // 명령행 인자가 정확히 2개인지 확인 (프로그램명 + 경로)
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <path>\n", argv[0]);
        return 1;
    }

    char *path = argv[1]; // 첫 번째 인자(경로)를 path 변수에 저장

    // strchr로 경로에서 마지막 '/' 문자의 위치를 찾을
    // 리눅스/유닉스에서 디렉토리 구분자는 '/'
    char *base = strchr(path, '/');

    // '/'가 발견되었다면 (즉, 경로에 디렉토리가 포함되어 있다면)
    if (base) {
        // base + 1: '/' 다음 문자부터 출력 (파일명만 출력)
        printf("%s\n", base + 1);
    } else {
        // '/'가 없다면 전체 경로가 파일명이므로 그대로 출력
        printf("%s\n", path);
    }

    return 0;
}
```

파일경로에서 파일명만 추출

```
user@DESKTOP-TDA53C2:~$ ./basename /test/test2/test.txt
test.txt
user@DESKTOP-TDA53C2:~$
```

bc.c

간단한 계산기

```
// bc.c - 간단한 계산기 (기본 사칙연산만)
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

double evaluate_expression(const char *expr) {
    // 매우 간단한 구현 - 단일 연산만 지원
    char op;
    double num1, num2;

    // sscanf로 "숫자 연산자 숫자" 형태의 문자열을 파싱
    if (sscanf(expr, "%lf %c %lf", &num1, &op, &num2) == 3) {
        switch (op) {
            case '+': return num1 + num2;
            case '-': return num1 - num2;
            case '*': return num1 * num2;
            case '/':
                // 0으로 나누기 방지
                if (num2 == 0) {
                    fprintf(stderr, "Division by zero\n");
                    return 0;
                }
                return num1 / num2;
            default:
                fprintf(stderr, "Unknown operator: %c\n", op);
                return 0;
        }
    }
}
```

```
user@DESKTOP-TDA53C2:~$ ./bc "3+4"
7
user@DESKTOP-TDA53C2:~$ ./bc "10 / 2"
5
user@DESKTOP-TDA53C2:~$ ./bc "5.5 * 3"
16.5
user@DESKTOP-TDA53C2:~$ ./bc "5-5"
0
user@DESKTOP-TDA53C2:~$
```

인자 없을 시 대화형 UI

```
user@DESKTOP-TDA53C2:~$ ./bc
Simple calculator (type 'quit' to exit)
bc> 3+5
8
bc> 7*8
56
bc> 9/9
1
bc> quit
user@DESKTOP-TDA53C2:~$
```

→ ' 화살표 방향 순서로 읽음

bc.c (계속)

```
} else {  
    // 단일 숫자인 경우 (연산자가 없음)  
    return atof(expr);  
}  
}  
  
int main(int argc, char *argv[]) {  
    char line[256];  
  
    // 명령행 인수가 있으면 각각을 계산  
    if (argc > 1) {  
        // 명령행 인수로 표현식 받기  
        for (int i = 1; i < argc; i++) {  
            double result = evaluate_expression(argv[i]);  
            printf("%.6g\n", result); // %.6g: 소수점 6자리까지, 불필요한 0 제거  
        }  
    } else {  
        // 대화형 모드  
        printf("Simple calculator (type 'quit' to exit)\n");  
        while (1) {  
            printf("bc> ");  
            // 사용자 입력 받기  
            if (!fgets(line, sizeof(line), stdin)) {  
                break;  
            }  
        }  
    }  
}
```



```
// 개행 문자 제거 (fgets는 \n도 포함해서 읽음)  
line[strcspn(line, "\n")] = 0;  
  
// "quit" 입력 시 종료  
if (strcmp(line, "quit") == 0) {  
    break;  
}  
  
double result = evaluate_expression(line);  
printf("%.6g\n", result);  
}  
  
return 0;  
}
```

cal.c

```
user@DESKTOP-TDA53C2: ~$ ./cal
      June 2025
Su Mo Tu We Th Fr Sa
                1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
```

달력 출력

```
// cal.c - 달력 출력
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int is_leap_year(int year) {
    // 윤년 판별: 4로 나누어떨어지고 100으로 나누어떨어지지 않거나, 400으로 나누어떨어짐
    return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
}

int days_in_month(int month, int year) {
    // 각 달의 일수 (1월=31일, 2월=28일, ...)
    int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    // 2월이고 윤년이면 29일
    if (month == 2 && is_leap_year(year)) {
        return 29;
    }
    return days[month - 1]; // 배열 인덱스는 0부터 시작하므로 -1
}

int day_of_week(int day, int month, int year) {
    // Zeller's congruence 공식으로 요일 계산
    if (month < 3) {
        month += 12; // 1월과 2월을 전년도 13월, 14월로 처리
        year--;
    }
}
```



```
int k = year % 100; // 년도의 뒤 두 자리
int j = year / 100; // 년도의 앞 두 자리
int h = (day + (13 * (month + 1)) / 5 + k + k / 4 + j / 4 - 2 * j) % 7;
return (h + 5) % 7; // 0=Sunday, 1=Monday, ...

void print_calendar(int month, int year) {
    char *months[] = {"January", "February", "March", "April", "May", "June",
                     "July", "August", "September", "October", "November", "December"};

    // 달력 제목 출력
    printf("    %s %d\n", months[month - 1], year);
    printf("Su Mo Tu We Th Fr Sa\n");

    int first_day = day_of_week(1, month, year); // 1일의 요일
    int days = days_in_month(month, year); // 해당 월의 총 일수

    // 첫 주의 공백 출력 (1일이 일요일이면 공백 없음)
    for (int i = 0; i < first_day; i++) {
        printf("    ");
    }

    // 날짜 출력
```

cal.c (계속)

```
for (int day = 1; day <= days; day++) {
    printf("%2d ", day); // 2자리로 정렬하여 출력
    // 토요일이면 줄바꿈
    if ((day + first_day) % 7 == 0) {
        printf("\n");
    }
}

// 마지막 줄이 완성되지 않았으면 줄바꿈
if ((days + first_day) % 7 != 0) {
    printf("\n");
}

int main(int argc, char *argv[]) {
    // 현재 날짜 정보 가져오기
    time_t t = time(NULL);
    struct tm *tm_info = localtime(&t);
    int month = tm_info->tm_mon + 1; // tm_mon은 0부터 시작하므로 +1
    int year = tm_info->tm_year + 1900; // tm_year는 1900년부터의 연수
}
```



```
70
71 // 명령행 인수에 따라 년도 또는 월/년도 설정
72 if (argc == 2) {
73     year = atoi(argv[1]); // 년도만 지정
74 } else if (argc == 3) {
75     month = atoi(argv[1]); // 월과 년도 지정
76     year = atoi(argv[2]);
77 }
78
79 // 월 범위 체크
80 if (month < 1 || month > 12) {
81     fprintf(stderr, "cal: invalid month\n");
82     return 1;
83 }
84
85 print_calendar(month, year);
86 return 0;
87 }
```

cat.c

```
// cat - 파일 내용 출력
#include <stdio.h>

int main(int argc, char *argv[]) {
    FILE *file;
    char ch;

    // 명령행 인수로 파일명이 주어졌는지 확인
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    // 파일을 읽기 모드로 열기
    file = fopen(argv[1], "r");
    if (!file) {
        perror("cat"); // 파일 열기 실패 시 에러 메시지 출력
        return 1;
    }

    // 파일을 한 문자씩 읽어서 출력
    while ((ch = fgetc(file)) != EOF) {
        putchar(ch); // 읽은 문자를 표준 출력에 출력
    }

    fclose(file); // 파일 닫기
    return 0;
}
```

파일 내용 출력

```
user@DESKTOP-TDA53C2:~$ ./cat test.txt
Line 1
Line 1
Line 1
Line 1
Line 2
Line 2
Line 2
Line 2
```


chmod.c

```
user@DESKTOP-TDA53C2:~$ ./chmod 600 test.txt
user@DESKTOP-TDA53C2:~$ ls -l test.txt
-rw----- 1 user user 5 Jun  8 19:19 test.txt
```

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <string.h>

mode_t parse_mode(const char *mode_str) {
    mode_t mode = 0;

    // 3자리 8진수 모드 파싱 (예: 755, 644)
    if (strlen(mode_str) == 3 &&
        mode_str[0] >= '0' && mode_str[0] <= '7' && // 첫 번째 자리 검증
        mode_str[1] >= '0' && mode_str[1] <= '7' && // 두 번째 자리 검증
        mode_str[2] >= '0' && mode_str[2] <= '7') { // 세 번째 자리 검증

        // 8진수를 비트 시프트로 변환
        // 소유자 권한 (6비트 시프트) | 그룹 권한 (3비트 시프트) | 기타 권한
        mode = (mode_str[0] - '0') << 6 |
              (mode_str[1] - '0') << 3 |
              (mode_str[2] - '0');
    }

    return mode;
}
```



```
int main(int argc, char *argv[]) {
    // 정확히 3개의 인수 필요 (프로그램명, 모드, 파일명)
    if (argc != 3) {
        fprintf(stderr, "Usage: %s mode file\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    // 모드 문자열을 파싱하여 mode_t 값으로 변환
    mode_t mode = parse_mode(argv[1]);

    if (mode == 0) {
        fprintf(stderr, "Invalid mode: %s\n", argv[1]);
        exit(EXIT_FAILURE);
    }

    // chmod 시스템 콜로 파일 권한 변경
    if (chmod(argv[2], mode) == -1) {
        perror("chmod"); // 실패 시 에러 메시지 출력
        exit(EXIT_FAILURE);
    }

    return 0;
}
```

clear.c

화면 지우기

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

int main() {
    // ANSI escape sequence로 화면 지우기
    // \033[2J: 전체 화면 지우기, \033[H: 커서를 홈 위치(좌상단)로 이동
    printf("\033[2J\033[H");
    fflush(stdout); // 출력 버퍼 강제로 비우기 (즉시 화면에 반영)
    return 0;
}
```

```
user@DESKTOP-TDA53C2:~$ ls
basename      bc.c  cat  chmod.c  cp.c  df.c  echo.c  find.c  head.c  test.txt
basename.c    cal  cat.c  clear    cut.c  dirname.c  env.c  free.c  id.c
bc            cal.c  chmod  clear.c  date.c  du.c  factor.c  grep.c  test
user@DESKTOP-TDA53C2:~$ ls
basename      bc.c  cat  chmod.c  cp.c  df.c  echo.c  find.c  head.c  test.txt
basename.c    cal  cat.c  clear    cut.c  dirname.c  env.c  free.c  id.c
bc            cal.c  chmod  clear.c  date.c  du.c  factor.c  grep.c  test
user@DESKTOP-TDA53C2:~$ ls
basename      bc.c  cat  chmod.c  cp.c  df.c  echo.c  find.c  head.c  test.txt
basename.c    cal  cat.c  clear    cut.c  dirname.c  env.c  free.c  id.c
bc            cal.c  chmod  clear.c  date.c  du.c  factor.c  grep.c  test
user@DESKTOP-TDA53C2:~$ ls
basename      bc.c  cat  chmod.c  cp.c  df.c  echo.c  find.c  head.c  test.txt
basename.c    cal  cat.c  clear    cut.c  dirname.c  env.c  free.c  id.c
bc            cal.c  chmod  clear.c  date.c  du.c  factor.c  grep.c  test
user@DESKTOP-TDA53C2:~$ ls
basename      bc.c  cat  chmod.c  cp.c  df.c  echo.c  find.c  head.c  test.txt
basename.c    cal  cat.c  clear    cut.c  dirname.c  env.c  free.c  id.c
bc            cal.c  chmod  clear.c  date.c  du.c  factor.c  grep.c  test
user@DESKTOP-TDA53C2:~$ ./clear
```

지워진 모습

```
user@DESKTOP-TDA53C2:~$
```

cp.c

```
user@DESKTOP-TDA53C2:~$ cat test.txt
test
user@DESKTOP-TDA53C2:~$ cat test2.txt
test2
user@DESKTOP-TDA53C2:~$ ./cp test.txt test2.txt
user@DESKTOP-TDA53C2:~$ cat test2.txt
test
user@DESKTOP-TDA53C2:~$ █
```

파일 복사

```
// cp - 파일 복사
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    FILE *src, *dst;
    char ch;

    // 소스 파일과 대상 파일 이름이 모두 필요
    if (argc < 3) {
        fprintf(stderr, "Usage: %s <source> <destination>\n", argv[0]);
        return 1;
    }

    // 소스 파일을 읽기 모드로 열기
    src = fopen(argv[1], "r");
    if (!src) {
        perror("cp: source");
        return 1;
    }
}
```



```
// 대상 파일을 쓰기 모드로 열기 (없으면 생성, 있으면 덮어쓰기)
dst = fopen(argv[2], "w");
if (!dst) {
    perror("cp: destination");
    fclose(src); // 소스 파일 닫기
    return 1;
}

// 소스 파일에서 한 문자씩 읽어서 대상 파일에 쓰기
while ((ch = fgetc(src)) != EOF) {
    fputc(ch, dst);
}

fclose(src); // 소스 파일 닫기
fclose(dst); // 대상 파일 닫기
return 0;
}
```

cut.c

```
user@DESKTOP-TDA53C2:~$ nano test.txt
user@DESKTOP-TDA53C2:~$ cat test.txt
tes t
tes tteest
te st ets
et s ets
se te st set
user@DESKTOP-TDA53C2:~$ ./cut 1 test.txt
tes
tes
te
et
se
user@DESKTOP-TDA53C2:~$ ./cut 2 test.txt
t
tteest
st
s
te
user@DESKTOP-TDA53C2:~$
```

텍스트 필드 추출

```
// cut - 텍스트 필드 추출
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
    FILE *file;
    char line[1024], *token;
    int field = 1;

    // 필드 번호와 파일명이 필요
    if (argc < 3) {
        fprintf(stderr, "Usage: %s <field_number> <filename>\n", argv[0]);
        return 1;
    }

    field = atoi(argv[1]); // 추출할 필드 번호
    // 파일을 읽기 모드로 열기
    file = fopen(argv[2], "r");
    if (!file) {
        perror("cut");
        return 1;
    }
}
```



```
23 // 파일을 한 줄씩 읽기
24 while (fgets(line, sizeof(line), file)) {
25     // 공백과 탭으로 첫 번째 토큰 분리
26     token = strtok(line, " \t");
27     // 지정된 필드까지 토큰 이동
28     for (int i = 1; i < field && token; i++) {
29         token = strtok(NULL, " \t"); // 다음 토큰 가져오기
30     }
31     // 해당 필드가 존재하면 출력
32     if (token) {
33         printf("%s", token);
34     }
35     printf("\n");
36 }
37
38 fclose(file);
39 return 0;
40
41 }
```

date(-u).c

현재 날짜와 시간 출력

-u 옵션 사용시 UTC 시간 출력

```
// date - 현재 날짜와 시간 출력 (-u 옵션 사용)
#include <stdio.h>
#include <time.h>
#include <string.h>

int main(int argc, char *argv[]) {
    time_t t = time(NULL);
    struct tm *tm;

    // -u 옵션이 있으면 UTC 시간, 없으면 로컬 시간
    if (argc > 1 && strcmp(argv[1], "-u") == 0) {
        tm = gmtime(&t); // UTC 시간
    } else {
        tm = localtime(&t); // 로컬 시간
    }

    printf("%s", asctime(tm));
    return 0;
}
```

```
user@DESKTOP-TDA53C2:~$ ./date
Thu Jun 12 20:28:11 2025
user@DESKTOP-TDA53C2:~$ ./date -u
Thu Jun 12 11:28:15 2025
```

df.c

```
user@DESKTOP-TDA53C2:~$ gcc df.c -o df
user@DESKTOP-TDA53C2:~$ ./df
Filesystem      1K-blocks      Used    Available     Use%    Mounted on
filesystem      1055762868    1700900  1000358496      0%      /
user@DESKTOP-TDA53C2:~$
```

파일 시스템 디스크 공간 사용량 표시

```
// df.c - 파일시스템 디스크 공간 사용량 표시
#include <stdio.h>
#include <sys/statvfs.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    struct statvfs stat;
    // 인수가 없으면 루트 디렉토리("/")를 기본값으로 사용
    const char *path = (argc > 1) ? argv[1] : "/";

    // statvfs 시스템 콜로 파일시스템 정보 가져오기
    if (statvfs(path, &stat) != 0) {
        perror("statvfs");
        return 1;
    }

    // 파일시스템 통계 계산
    unsigned long block_size = stat.f_frsize; // 블록 크기
    unsigned long total_blocks = stat.f_blocks; // 전체 블록 수
    unsigned long free_blocks = stat.f_bavail; // 일반 사용자가 사용 가능한 블록 수
    unsigned long used_blocks = total_blocks - stat.f_bfree; // 사용 중인 블록 수
```



```
    // KB 단위로 변환
    unsigned long total_size = (total_blocks * block_size) / 1024;
    unsigned long used_size = (used_blocks * block_size) / 1024;
    unsigned long free_size = (free_blocks * block_size) / 1024;

    // df 명령어와 유사한 형태로 출력
    printf("Filesystem\t1K-blocks\tUsed\tAvailable\tUse%\tMounted on\n");
    printf("%s\t\t%lu\t\t%lu\t\t%lu\t\t\t%lu%%\t\t%s\n",
           "filesystem", total_size, used_size, free_size,
           (used_size * 100) / total_size, path); // 사용률 계산

    return 0;
}
```

dirname.c

```
// dirname.c - 파일 경로에서 디렉토리 경로만 추출
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    // 명령행 인자가 정확히 2개인지 확인 (프로그램명 + 경로)
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <path>\n", argv[0]);
        return 1;
    }

    // strdup로 경로 문자열 복사 (원본 수정 방지)
    char *path = strdup(argv[1]);
    // 마지막 '/' 문자의 위치 찾기
    char *last_slash = strrchr(path, '/');

    if (last_slash) {
        // '/' 위치에 널 문자 삽입하여 디렉토리 경로만 남김
        *last_slash = '\0';
        printf("%s\n", path);
    } else {
        // '/'가 없으면 현재 디렉토리를 의미하는 "." 출력
        printf(".\n");
    }

    free(path); // strdup으로 할당한 메모리 해제
    return 0;
}
```

```
user@DESKTOP-TDA53C2:~$ ./dirname test/test2/test.txt
test/test2
user@DESKTOP-TDA53C2:~$
```

파일 경로에서 디렉토리 경로만 추출

du.c

```
user@DESKTOP-TDA53C2: ~$ gcc du.c -o du
user@DESKTOP-TDA53C2: ~$ ./du
210 .
```

디스크 사용량 표시

```
// du.c - 디스크 사용량 표시
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <dirent.h>
#include <string.h>
#include <unistd.h>

long get_dir_size(const char *path) {
    DIR *dir;
    struct dirent *entry;
    struct stat st;
    long total_size = 0;
    char full_path[1024];

    // 디렉토리 열기
    dir = opendir(path);
    if (!dir) {
        perror("opendir");
        return 0;
    }

    // 디렉토리 내 모든 항목 순회
    while ((entry = readdir(dir)) != NULL) {
        // "."과 ".." 디렉토리는 제외
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0)
            continue;

        snprintf(full_path, sizeof(full_path), "%s/%s", path, entry->d_name);
        if (stat(full_path, &st) == 0) {
            if (S_ISDIR(st.st_mode)) {
                // 디렉토리라면 재귀적으로 크기 계산
                total_size += get_dir_size(full_path);
            } else {
                // 파일이면 크기 추가
                total_size += st.st_size;
            }
        }
    }

    closedir(dir);
    return total_size;
}

int main(int argc, char *argv[]) {
    // 인수가 없으면 현재 디렉토리 사용
    const char *path = (argc > 1) ? argv[1] : ".";
    long size = get_dir_size(path);

    // KB 단위로 변환하여 출력
    printf("%ld\t%s\n", size / 1024, path);
    return 0;
}
```



```
snprintf(full_path, sizeof(full_path), "%s/%s", path, entry->d_name);
if (stat(full_path, &st) == 0) {
    if (S_ISDIR(st.st_mode)) {
        // 디렉토리라면 재귀적으로 크기 계산
        total_size += get_dir_size(full_path);
    } else {
        // 파일이면 크기 추가
        total_size += st.st_size;
    }
}
}

closedir(dir);
return total_size;
}

int main(int argc, char *argv[]) {
    // 인수가 없으면 현재 디렉토리 사용
    const char *path = (argc > 1) ? argv[1] : ".";
    long size = get_dir_size(path);

    // KB 단위로 변환하여 출력
    printf("%ld\t%s\n", size / 1024, path);
    return 0;
}
```


echo.c

텍스트 출력

```
// echo - 텍스트 출력
#include <stdio.h>

int main(int argc, char *argv[]) {
    // 첫 번째 인수부터 마지막까지 출력
    for (int i = 1; i < argc; i++) {
        printf("%s", argv[i]);
        // 마지막 인수가 아니면 공백 추가
        if (i < argc - 1) printf(" ");
    }
    printf("\n"); // 마지막에 개행 추가
    return 0;
}
```

```
user@DESKTOP-TDA53C2:~$ ./echo hello
hello
user@DESKTOP-TDA53C2:~$
```

factor.c

```
user@DESKTOP-TDA53C2:~$ ./factor 25
25: 5 5
user@DESKTOP-TDA53C2:~$ ./factor 135
135: 3 3 3 5
user@DESKTOP-TDA53C2:~$ ./factor 123479123748
123479123748: 2 2 3 71 1847 78467
user@DESKTOP-TDA53C2:~$
```

소인수분해

```
// factor.c - 소인수분해
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void factorize(long long n) {
    printf("%lld:", n);

    // 2로 나누기 (찍수 처리)
    while (n % 2 == 0) {
        printf(" 2");
        n /= 2;
    }

    // 3부터 sqrt(n)까지 줄수로 나누기
    for (long long i = 3; i * i <= n; i += 2) {
        while (n % i == 0) {
            printf(" %lld", i);
            n /= i;
        }
    }

    // n이 2보다 큰 소수인 경우 (남은 수가 소수)
    if (n > 2) {
        printf(" %lld", n);
    }

    printf("\n");
}
```



```
int main(int argc, char *argv[]) {
    // 최소 하나의 숫자가 필요
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <number> [number2] ...\n", argv[0]);
        return 1;
    }

    // 모든 인수에 대해 소인수분해 실행
    for (int i = 1; i < argc; i++) {
        long long n = atoll(argv[i]); // 문자열을 long long으로 변환
        if (n <= 0) {
            fprintf(stderr, "factor: '%s' is not a valid positive integer\n", argv[i]);
            continue;
        }
        factorize(n);
    }

    return 0;
}
```

find.c

```
user@DESKTOP-TDA53C2:~$ ./find . test.txt
./test.txt
./test/test2/test.txt
user@DESKTOP-TDA53C2:~$
```

파일 검색

```
// find - 파일 검색 (간단한 버전)
#include <stdio.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>

void search_dir(char *path, char *name) {
    DIR *dir = opendir(path);
    struct dirent *entry;
    struct stat st;
    char full_path[1024];

    if (!dir) return; // 디렉토리 열기 실패 시 종료

    // 디렉토리 내 모든 항목 검사
    while ((entry = readdir(dir))) {
        // "."과 ".." 디렉토리는 제외
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0)
            continue;

        // 전체 경로 생성
        snprintf(full_path, sizeof(full_path), "%s/%s", path, entry->d_name);

        // 파일명에 검색어가 포함되어 있는지 확인
        if (strstr(entry->d_name, name)) {
            printf("%s\n", full_path);
        }
    }
}
```



```
// 디렉토리인 경우 재귀적으로 검색
if (stat(full_path, &st) == 0 && S_ISDIR(st.st_mode)) {
    search_dir(full_path, name);
}

closedir(dir);
}

int main(int argc, char *argv[]) {
    // 경로와 검색할 이름이 필요
    if (argc < 3) {
        fprintf(stderr, "Usage: %s <path> <name>\n", argv[0]);
        return 1;
    }

    // 지정된 경로에서 검색 시작
    search_dir(argv[1], argv[2]);
    return 0;
}
```

free.c

```
user@DESKTOP-TDA53C2:~$ gcc free.c -o free
user@DESKTOP-TDA53C2:~$ ./free
          total      used      free      shared  buff/cache   available
Mem:      16231652    871796    15359856        0         0    15646216
Swap:      4194304         0     4194304
user@DESKTOP-TDA53C2:~$
```

메모리 사용량 표시

```
// free.c - 메모리 사용량 표시
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    FILE *fp;
    char line[256];
    long mem_total = 0, mem_free = 0, mem_available = 0;
    long swap_total = 0, swap_free = 0;

    // /proc/meminfo 파일 열기 (리눅스 메모리 정보)
    fp = fopen("/proc/meminfo", "r");
    if (!fp) {
        perror("fopen /proc/meminfo");
        return 1;
    }
}
```



```
// 파일을 한 줄씩 읽어서 필요한 정보 추출
while (fgets(line, sizeof(line), fp)) {
    if (strncmp(line, "MemTotal:", 9) == 0) {
        sscanf(line, "MemTotal: %ld kB", &mem_total);
    } else if (strncmp(line, "MemFree:", 9) == 0) {
        sscanf(line, "MemFree: %ld kB", &mem_free);
    } else if (strncmp(line, "MemAvailable:", 13) == 0) {
        sscanf(line, "MemAvailable: %ld kB", &mem_available);
    } else if (strncmp(line, "SwapTotal:", 10) == 0) {
        sscanf(line, "SwapTotal: %ld kB", &swap_total);
    } else if (strncmp(line, "SwapFree:", 9) == 0) {
        sscanf(line, "SwapFree: %ld kB", &swap_free);
    }
}

fclose(fp);

// free 명령어와 유사한 형태로 출력
printf("          total      used      free      shared  buff/cache   available\n");
printf("Mem:      %ld %ld %ld %ld %ld %ld\n",
       mem_total, mem_total - mem_free, mem_free, 0, 0, mem_available);
printf("Swap:     %ld %ld %ld\n",
       swap_total, swap_total - swap_free, swap_free);

return 0;
}
```

grep.c

```
1 // grep - 패턴 검색
2 #include <stdio.h>
3 #include <string.h>
4
5 int main(int argc, char *argv[]) {
6     FILE *file;
7     char line[1024];
8
9     // 패턴과 파일명이 필요
10    if (argc < 3) {
11        fprintf(stderr, "Usage: %s <pattern> <filename>\n", argv[0]);
12        return 1;
13    }
14
15    // 파일 열기
16    file = fopen(argv[2], "r");
17    if (!file) {
18        perror("grep");
19        return 1;
20    }
21
22    // 파일을 한 줄씩 읽으면서 패턴 검색
23    while (fgets(line, sizeof(line), file)) {
24        // strstr로 패턴이 포함된 줄인지 확인
25        if (strstr(line, argv[1])) {
26            printf("%s", line); // 패턴이 포함된 줄 출력
27        }
28    }
29
30    fclose(file);
31    return 0;
32 }
```

패턴 검색

```
user@DESKTOP-TDA53C2:~$ cat test2.txt
asdfasdfasdf
asdfasdfasdf
asdfasdfasdfasdf
asdsad
asdfasdf
asdf
hello

asdfasdf

asdfsadf
asdf
asdfasd
asfd
user@DESKTOP-TDA53C2:~$ ./grep hello test2.txt
hello
```

head.c

```
// head - 파일의 처음 몇 줄 출력
#include <stdio.h>

int main(int argc, char *argv[]) {
    FILE *file;
    char line[1024];
    int count = 10, lines = 0; // 기본값: 10줄

    // 파일명이 필요
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    // 파일 열기
    file = fopen(argv[1], "r");
    if (!file) {
        perror("head");
        return 1;
    }

    // 파일을 한 줄씩 읽으면서 지정된 줄 수만큼 출력
    while (fgets(line, sizeof(line), file) && lines < count) {
        printf("%s", line);
        lines++; // 출력한 줄 수 증가
    }

    fclose(file);
    return 0;
}
```

파일의 처음 몇줄 출력

```
user@DESKTOP-TDA53C2:~$ cat > test2.txt
asdfasdfasdf
asdfasdf
asdfasdfasdf
asdfasdf
asdfasdf

asdf
asdf
asdf
asdfasdf
asdfasdf
1234
1234
123512465
12341234
123423
user@DESKTOP-TDA53C2:~$ ./head test2.txt
asdfasdfasdf
asdfasdf
asdfasdfasdf
asdfasdf
asdfasdf
```

id.c

```
// id - 사용자 ID 정보 출력
#include <stdio.h>
#include <unistd.h>
#include <pwd.h>
#include <grp.h>

int main() {
    uid_t uid = getuid();           // 사용자 ID 가져오기
    gid_t gid = getgid();           // 그룹 ID 가져오기
    struct passwd *pw = getpwuid(uid); // 사용자 이름 정보 가져오기
    struct group *gr = getgrgid(gid); // 그룹 이름 정보 가져오기

    // 사용자 ID 출력
    printf("uid=%d", uid);
    if (pw) printf("(%s)", pw->pw_name); // 사용자 이름 출력

    // 그룹 ID 출력
    printf(" gid=%d", gid);
    if (gr) printf("(%s)", gr->gr_name); // 그룹 이름 출력

    printf("\n");
    return 0;
}
```

사용자 id 정보 출력

```
user@DESKTOP-TDA53C2:~$ ./id
uid=1000(user) gid=1000(user)
user@DESKTOP-TDA53C2:~$
```

ln(-s).c

```
user@DESKTOP-TDA53C2:~$ ./ln test.txt hard_test.txt
user@DESKTOP-TDA53C2:~$ ./ln -s test.txt sym_test.txt
user@DESKTOP-TDA53C2:~$ ls -la test.txt hard_test.txt sym_test.txt
-rw----- 2 user user 12 Jun 10 21:15 hard_test.txt
lrwxrwxrwx 1 user user  8 Jun 10 21:16 sym_test.txt -> test.txt
-rw----- 2 user user 12 Jun 10 21:15 test.txt
user@DESKTOP-TDA53C2:~$ cat test.txt
Hello World
```

심볼릭링크 및 하드링크

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

int main(int argc, char *argv[]) {
    int symbolic = 0; // 심볼릭 링크 플래그
    int opt;

    // 명령행 옵션 파싱 (-s 옵션)
    while ((opt = getopt(argc, argv, "s")) != -1) {
        switch (opt) {
            case 's':
                symbolic = 1; // 심볼릭 링크 생성 모드
                break;
            default:
                fprintf(stderr, "Usage: %s [-s] target linkname\n", argv[0]);
                exit(EXIT_FAILURE);
        }
    }

    // 옵션을 제외하고 정확히 2개의 인수 필요
    if (argc - optind != 2) {
        fprintf(stderr, "Usage: %s [-s] target linkname\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    char *target = argv[optind]; // 링크할 대상 파일
    char *linkname = argv[optind + 1]; // 생성할 링크 이름

    int result;
    if (symbolic) {
        // 심볼릭 링크 생성
        result = symlink(target, linkname);
    } else {
        // 하드 링크 생성
        result = link(target, linkname);
    }

    // 링크 생성 실패 시 에러 처리
    if (result == -1) {
        perror("ln");
        exit(EXIT_FAILURE);
    }

    return 0;
}
```



```
// 옵션을 제외하고 정확히 2개의 인수 필요
if (argc - optind != 2) {
    fprintf(stderr, "Usage: %s [-s] target linkname\n", argv[0]);
    exit(EXIT_FAILURE);
}

char *target = argv[optind]; // 링크할 대상 파일
char *linkname = argv[optind + 1]; // 생성할 링크 이름

int result;
if (symbolic) {
    // 심볼릭 링크 생성
    result = symlink(target, linkname);
} else {
    // 하드 링크 생성
    result = link(target, linkname);
}

// 링크 생성 실패 시 에러 처리
if (result == -1) {
    perror("ln");
    exit(EXIT_FAILURE);
}

return 0;
}
```


ls(-l,-a).c

```
user@DESKTOP-TDA53C2:~/test$ ./ls
test.txt test2 ls ls.c test
user@DESKTOP-TDA53C2:~/test$ ./ls -l
total 36
-rw-r--r--  1 user    user      12 Jun 10 21:26 test.txt
drwxr-xr-x  2 user    user    4096 Jun 08 19:08 test2
-rwxr-xr-x  1 user    user   16904 Jun 12 20:05 ls
-rw-r--r--  1 user    user   3637 Jun 12 20:05 ls.c
drwxr-xr-x  2 user    user    4096 Jun 10 21:28 test
user@DESKTOP-TDA53C2:~/test$ ./ls -a
test.txt .. test2 ls ls.c test .
```

디렉토리 내의 파일 목록을 나열

-l
상세 정보 표시

-a
숨김 파일 포함

ls(-l,-a).c (계속)

```
#include <stdio.h> // 표준 입출력 함수를 (printf 등)
#include <stdlib.h> // 일반적인 함수를 (exit, malloc 등)
#include <dirent.h> // 디렉토리 처리 함수를 (opendir, readdir 등)
#include <sys/stat.h> // 파일 상태 정보 구조체와 함수를 (stat 등)
#include <pwd.h> // 사용자 id + 사용자 이름 변환용
#include <grp.h> // 그룹 id + 그룹 이름 변환용
#include <time.h> // 시간 처리 함수를 (strftime 등)
#include <string.h> // 문자열 처리 함수를
#include <unistd.h> // getopt 함수 사용을 위한 헤더
#include <getopt.h> // 명령행 옵션 파싱을 위한 헤더

// 파일의 권한을 문자열 형태로 변환 (예: -rw-r--r--)
void mode_to_string(mode_t mode, char *str) {
    // 파일의 종류 판단: 디렉토리(d), 링크(l), 일반파일(-) 등
    str[0] = S_ISDIR(mode) ? 'd' :
             S_ISLNK(mode) ? 'l' :
             S_ISCHR(mode) ? 'c' :
             S_ISBLK(mode) ? 'b' :
             S_ISFIFO(mode) ? 'p' :
             S_ISSOCK(mode) ? 's' : '-';

    // 사용자 권한 (rwx)
    str[1] = (mode & S_IRUSR) ? 'r' : '-';
    str[2] = (mode & S_IWUSR) ? 'w' : '-';
    str[3] = (mode & S_IXUSR) ? 'x' : '-';

    // 그룹 권한 (rwx)
    str[4] = (mode & S_IRGRP) ? 'r' : '-';
    str[5] = (mode & S_IWGRP) ? 'w' : '-';
    str[6] = (mode & S_IXGRP) ? 'x' : '-';

    // 기타 사용자 권한 (rwx)
    str[7] = (mode & S_IROTH) ? 'r' : '-';
    str[8] = (mode & S_IWOTH) ? 'w' : '-';
    str[9] = (mode & S_IXOTH) ? 'x' : '-';

    str[10] = '\0'; // 문자열 끝 표시
}

// -l 옵션인 타 상세 정보 출력하는 함수
void print_long_format(const char *path, const char *name) {
    struct stat file_stat; // 파일 상태 정보 저장 구조체
    char full_path[1024]; // 파일 전체 경로 저장
    char mode_str[11]; // 권한 문자열 (예: -rw-r--r--)
    char time_str[13]; // 시간 문자열 (예: Jun 12 17:00)
    struct passwd *pw; // 사용자 정보
    struct group *gr; // 그룹 정보
```



```
// 경로 + 파일 이름 + 전체 경로 구성
snprintf(full_path, sizeof(full_path), "%s/%s", path, name);

// 파일 상태 정보 얻기
if (stat(full_path, &file_stat) == -1) {
    perror("stat"); // 오류 메시지 출력
    return;
}

// 파일 모드를 문자열로 변환 (예: -rw-r--r--)
mode_to_string(file_stat.st_mode, mode_str);

// UID/GID + 사용자명/그룹명
pw = getpwuid(file_stat.st_uid);
gr = getgrgid(file_stat.st_gid);

// 수정 시간 + 사람이 보기 쉬운 문자열로 변환
struct tm *timeinfo = localtime(&file_stat.st_mtime);
strftime(time_str, sizeof(time_str), "%b %d %H:%M", timeinfo);

// 출력 형식: 권한 링크수 사용자 그룹 크기 시간 이름
printf("%s %3ld %s %s %3ld %s %s\n",
       mode_str,
       file_stat.st_nlink, // 하드 링크 수
       pw ? pw->pw_name : "unknown", // 사용자명 (없으면 unknown)
       gr ? gr->gr_name : "unknown", // 그룹명
       file_stat.st_size, // 파일 크기 (bytes)
       time_str, // 수정 시간
       name); // 파일 이름

// -l 옵션이 없을 때 단순히 파일 이름만 출력
void print_simple_format(const char *name) {
    printf("%s ", name); // 파일명 뒤에 공백 붙여 출력
}

int main(int argc, char *argv[]) {
    DIR *dir; // 디렉토리 포인터
    struct dirent *entry; // 디렉토리 항목 구조체
    char *directory = "."; // 기본 디렉토리는 현재 디렉토리
    int long_format = 0; // -l 옵션 여부 (0: 꺼짐, 1: 켜짐)
    int show_hidden = 0; // -a 옵션 여부 (숨김파일 출력)
    int opt; // getopt 결과 저장 변수
```

ls(-l,-a).c (계속)

```
// getopt로 옵션 파싱 (-l, -a만 처리)
while ((opt = getopt(argc, argv, "la")) != -1) {
    switch (opt) {
        case 'l':
            long_format = 1; // 상세 출력 모드 켜기
            break;
        case 'a':
            show_hidden = 1; // 숨김파일 포함 모드 켜기
            break;
        case '?': // 알 수 없는 옵션 처리
            fprintf(stderr, "알 수 없는 옵션: -%c\n", optopt);
            fprintf(stderr, "사용법: %s [-l] [-a] [디렉토리]\n", argv[0]);
            return 1;
        default:
            fprintf(stderr, "옵션 처리 오류\n");
            return 1;
    }
}

// 옵션 다음에 오는 인수가 디렉토리 경로라면 설정
if (optind < argc) {
    directory = argv[optind];
}

// 디렉토리 열기
dir = opendir(directory);
if (!dir) {
    perror("ls"); // 디렉토리 열기 실패
    return 1;
}

// -l 옵션이면 total 블록 수 먼저 출력
if (long_format) {
    long total_blocks = 0;
    rewinddir(dir); // 디렉토리 포인터 위치 초기화

    // 디렉토리 내부 파일 순회
    while ((entry = readdir(dir))) {
        // 숨김 파일은 옵션 없으면 건너뛰기
        if (!show_hidden && entry->d_name[0] == '.')
            continue;

        struct stat file_stat;
        char full_path[1024];
        snprintf(full_path, sizeof(full_path), "%s/%s", directory, entry->d_name);

        // 파일 상태 정보 가져오기
        if (stat(full_path, &file_stat) == 0) {
            total_blocks += file_stat.st_blocks; // 블록 수 누적
        }

        // 512바이트 단위를 1KB 단위로 나누어 출력
        printf("total %ld\n", total_blocks / 2);
        rewinddir(dir); // 다시 출력 위해 포인터 초기화
    }
}

// 상세 파일 목록 출력
while ((entry = readdir(dir))) {
    if (!show_hidden && entry->d_name[0] == '.')
        continue;

    if (long_format) {
        print_long_format(directory, entry->d_name); // 상세 출력
    } else {
        print_simple_format(entry->d_name); // 간단 출력
    }
}

if (!long_format) {
    printf("\n"); // 줄 바꿈
}

closedir(dir); // 디렉토리 닫기
return 0; // 정상 종료
```



```
struct stat file_stat;
char full_path[1024];
snprintf(full_path, sizeof(full_path), "%s/%s", directory, entry->d_name);

// 파일 상태 정보 가져오기
if (stat(full_path, &file_stat) == 0) {
    total_blocks += file_stat.st_blocks; // 블록 수 누적
}
}

// 512바이트 단위를 1KB 단위로 나누어 출력
printf("total %ld\n", total_blocks / 2);
rewinddir(dir); // 다시 출력 위해 포인터 초기화
}

// 상세 파일 목록 출력
while ((entry = readdir(dir))) {
    if (!show_hidden && entry->d_name[0] == '.')
        continue;

    if (long_format) {
        print_long_format(directory, entry->d_name); // 상세 출력
    } else {
        print_simple_format(entry->d_name); // 간단 출력
    }
}

if (!long_format) {
    printf("\n"); // 줄 바꿈
}

closedir(dir); // 디렉토리 닫기
return 0; // 정상 종료
}
```

mkdir.c

```
// mkdir - 디렉토리 생성
#include <stdio.h>
#include <sys/stat.h>

int main(int argc, char *argv[]) {
    // 명령행 인자 개수 확인 (프로그래밍 + 디렉토리명)
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <directory>\n", argv[0]);
        return 1;
    }

    // mkdir 시스템 콜로 디렉토리 생성
    // 0755: 소유자(rwx), 그룹(rx), 기타(rx) 권한 설정
    if (mkdir(argv[1], 0755) == -1) {
        perror("mkdir"); // 실패 시 시스템 에러 메시지 출력
        return 1;
    }

    return 0;
}
```

디렉토리 생성

```
user@DESKTOP-TDA53C2:~$ ./mkdir test5
user@DESKTOP-TDA53C2:~$ cd test5
user@DESKTOP-TDA53C2:~/test5$
```

mv.c

```
// mv - 파일 이동/이름변경
#include <stdio.h>

int main(int argc, char *argv[]) {
    // 명령행 인자 개수 확인 (프로그램명 + 소스 + 목적지)
    if (argc < 3) {
        fprintf(stderr, "Usage: %s <source> <destination>\n", argv[0]);
        return 1;
    }

    // rename 시스템 콜로 파일 이동/이름 변경
    // argv[1]: 원본 파일/디렉토리, argv[2]: 목적지 파일/디렉토리
    if (rename(argv[1], argv[2]) == -1) {
        perror("mv"); // 실패 시 시스템 에러 메시지 출력
        return 1;
    }

    return 0;
}
```

파일 이동/이름변경

```
user@DESKTOP-TDA53C2:~/test10$ ls
mv mv.c test.txt
user@DESKTOP-TDA53C2:~/test10$ ./mv test.txt test10.txt
user@DESKTOP-TDA53C2:~/test10$ ls
mv mv.c test10.txt
user@DESKTOP-TDA53C2:~/test10$
```

od(-x).c

```
user@DESKTOP-TDA53C2:~$ cat > test.txt
asdfasdfasdf
asdfasdf
asdf
12314
asdf
qwerqwer
user@DESKTOP-TDA53C2:~$ ./od test.txt
00000000 141 163 144 146 141 163 144 146 141 163 144 146 012 141 163 144
00000020 146 141 163 144 146 012 141 163 144 146 012 061 062 063 061 064
00000040 012 141 163 144 146 012 161 167 145 162 161 167 145 162 012
user@DESKTOP-TDA53C2:~$ ./od -x test.txt
00000000 61 73 64 66 61 73 64 66 61 73 64 66 0a 61 73 64
00000020 66 61 73 64 66 0a 61 73 64 66 0a 31 32 33 31 34
00000040 0a 61 73 64 66 0a 71 77 65 72 71 77 65 72 0a
user@DESKTOP-TDA53C2:~$
```

파일 내용을 8진수로 출력

-X
파일 내용을 16진수로 출력

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

int main(int argc, char *argv[]) {
    FILE *file = stdin; // 기본적으로 표준 입력에서 읽음
    int hex_mode = 0; // 16진수 모드 플래그
    int opt;

    // getopt로 명령행 옵션 처리
    while ((opt = getopt(argc, argv, "x")) != -1) {
        switch (opt) {
            case 'x':
                hex_mode = 1; // -x 옵션: 16진수 출력 모드
                break;
            default:
                fprintf(stderr, "Usage: %s [-x] [file]\n", argv[0]);
                exit(EXIT_FAILURE);
        }
    }

    // 파일명이 주어진 경우 해당 파일 열기
    if (optind < argc) {
        file = fopen(argv[optind], "rb"); // 바이너리 읽기 모드
        if (!file) {
            perror("od");
            exit(EXIT_FAILURE);
        }
    }

    unsigned char buffer[16]; // 16바이트씩 읽기 위한 버퍼
    size_t bytes_read;
    long offset = 0; // 파일 내 위치(오프셋) 추적

    // 파일을 16바이트씩 읽어서 처리
    while ((bytes_read = fread(buffer, 1, 16, file)) > 0) {
        printf("%07lo ", offset); // 8진수로 오프셋 출력

        if (hex_mode) {
            // 16진수 모드: 각 바이트를 16진수로 출력
            for (size_t i = 0; i < bytes_read; i++) {
                printf("%02x ", buffer[i]);
            }
        } else {
            // 기본 모드: 각 바이트를 8진수로 출력
            for (size_t i = 0; i < bytes_read; i++) {
                printf("%03o ", buffer[i]);
            }
        }

        printf("\n");
        offset += bytes_read; // 오프셋 증가
    }

    // 표준 입력이 아닌 경우에만 파일 닫기
    if (file != stdin) {
        fclose(file);
    }

    return 0;
}
```



```
unsigned char buffer[16]; // 16바이트씩 읽기 위한 버퍼
size_t bytes_read;
long offset = 0; // 파일 내 위치(오프셋) 추적

// 파일을 16바이트씩 읽어서 처리
while ((bytes_read = fread(buffer, 1, 16, file)) > 0) {
    printf("%07lo ", offset); // 8진수로 오프셋 출력

    if (hex_mode) {
        // 16진수 모드: 각 바이트를 16진수로 출력
        for (size_t i = 0; i < bytes_read; i++) {
            printf("%02x ", buffer[i]);
        }
    } else {
        // 기본 모드: 각 바이트를 8진수로 출력
        for (size_t i = 0; i < bytes_read; i++) {
            printf("%03o ", buffer[i]);
        }
    }

    printf("\n");
    offset += bytes_read; // 오프셋 증가
}

// 표준 입력이 아닌 경우에만 파일 닫기
if (file != stdin) {
    fclose(file);
}

return 0;
}
```

pwd.c

```
// pwd - 현재 작업 디렉토리 출력
#include <stdio.h>
#include <unistd.h>

int main() {
    char cwd[1024]; // 현재 작업 디렉토리 경로를 저장할 버퍼

    // getcwd로 현재 작업 디렉토리 경로 획득
    if (getcwd(cwd, sizeof(cwd))) {
        printf("%s\n", cwd); // 경로 출력
    } else {
        perror("pwd"); // 실패 시 에러 메시지 출력
        return 1;
    }
    return 0;
}
```

현재 작업 디렉토리 출력

```
user@DESKTOP-TDA53C2:~$ ./pwd
/home/user
user@DESKTOP-TDA53C2:~$
```

rm.c

```
// rm - 파일 삭제
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    // 명령행 인자 개수 확인 (프로그램명 + 파일명)
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    // unlink 시스템 콜로 파일 삭제
    // unlink는 파일의 링크를 제거하여 파일을 삭제함
    if (unlink(argv[1]) == -1) {
        perror("rm"); // 실패 시 시스템 에러 메시지 출력
        return 1;
    }

    return 0;
}
```

파일 삭제

```
user@DESKTOP-TDA53C2:~$ ./rm test10.txt
user@DESKTOP-TDA53C2:~$ cat test10.txt
cat: test10.txt: No such file or directory
user@DESKTOP-TDA53C2:~$
```


rmdir.c

```
// rmdir - 빈 디렉토리 삭제
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    // 명령행 인자 개수 확인 (프로그램명 + 디렉토리명)
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <directory>\n", argv[0]);
        return 1;
    }

    // rmdir 시스템 콜로 빈 디렉토리 삭제
    // 디렉토리가 비어있지 않으면 삭제 실패
    if (rmdir(argv[1]) == -1) {
        perror("rmdir"); // 실패 시 시스템 에러 메시지 출력
        return 1;
    }

    return 0;
}
```

빈 디렉토리 삭제

```
user@DESKTOP-TDA53C2:~$ cd test5
user@DESKTOP-TDA53C2:~/test5$ cd
user@DESKTOP-TDA53C2:~$ ./rmdir test5
user@DESKTOP-TDA53C2:~$ cd test5
-bash: cd: test5: No such file or directory
user@DESKTOP-TDA53C2:~$
```

seq.c

수열 생성

```
// seq.c - 수열 생성
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int start = 1, end = 1, step = 1; // 시작값, 끝값, 증가값 초기화

    // 명령행 인자 개수에 따른 처리
    if (argc == 2) {
        // seq LAST: 1부터 LAST까지
        end = atoi(argv[1]);
    } else if (argc == 3) {
        // seq FIRST LAST: FIRST부터 LAST까지
        start = atoi(argv[1]);
        end = atoi(argv[2]);
    } else if (argc == 4) {
        // seq FIRST INCREMENT LAST: FIRST부터 INCREMENT씩 증가하여 LAST까지
        start = atoi(argv[1]);
        step = atoi(argv[2]);
        end = atoi(argv[3]);
    } else {
        fprintf(stderr, "Usage: %s [FIRST [INCREMENT]] LAST\n", argv[0]);
        return 1;
    }
}
```



```
// 증가값이 0이면 무한루프 방지
if (step == 0) {
    fprintf(stderr, "seq: INCREMENT must not be 0\n");
    return 1;
}

// 양수 증가값: 시작부터 끝까지 증가
if (step > 0) {
    for (int i = start; i <= end; i += step) {
        printf("%d\n", i);
    }
} else {
    // 음수 증가값: 시작부터 끝까지 감소
    for (int i = start; i >= end; i += step) {
        printf("%d\n", i);
    }
}

return 0;
}
```

```
user@DESKTOP-TDA53C2:~$ ./seq 5
1
2
3
4
5
user@DESKTOP-TDA53C2:~$ ./seq 3 7
3
4
5
6
7
user@DESKTOP-TDA53C2:~$ ./seq 2 3 11
2
5
8
11
user@DESKTOP-TDA53C2:~$ ./seq 10 -2 2
10
6
4
2
```

shred(-n).c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <time.h>

int shred_file(const char *filename, int passes) {
    // 파일을 읽기/쓰기 바이너리 모드로 열기
    FILE *file = fopen(filename, "r+b");
    if (!file) {
        perror("shred");
        return -1;
    }

    // fstat으로 파일 크기 정보 획득
    struct stat st;
    if (fstat(fileno(file), &st) == -1) {
        perror("shred");
        fclose(file);
        return -1;
    }

    size_t file_size = st.st_size; // 파일 크기 저장
    // 파일 크기만큼 메모리 할당
    unsigned char *buffer = malloc(file_size);
    if (!buffer) {
        fprintf(stderr, "shred: memory allocation failed\n");
        fclose(file);
        return -1;
    }
}
```



```
    srand(time(NULL)); // 난수 생성기 초기화

    // 지정된 횟수만큼 덮어쓰기 수행
    for (int pass = 0; pass < passes; pass++) {
        printf("shred: %s: pass %d/%d...\n", filename, pass + 1, passes);

        // 버퍼를 랜덤 데이터로 채움
        for (size_t i = 0; i < file_size; i++) {
            buffer[i] = rand() % 256; // 0-255 범위의 랜덤 바이트
        }

        // 파일 시작 위치로 이동하여 랜덤 데이터 쓰기
        fseek(file, 0, SEEK_SET);
        fwrite(buffer, 1, file_size, file);
        fflush(file); // 버퍼를 디스크에 강제 쓰기
        fsync(fileno(file)); // 시스템 버퍼도 디스크에 동기화

        free(buffer);
        fclose(file);

        // 덮어쓰기 완료 후 파일 삭제
        if (unlink(filename) == -1) {
            perror("shred");
            return -1;
        }
    }
}
```

```
user@DESKTOP-TDA53C2:~$ cat test2.txt
test
user@DESKTOP-TDA53C2:~$ ./shred test2.txt
shred: test2.txt: pass 1/3...
shred: test2.txt: pass 2/3...
shred: test2.txt: pass 3/3...
user@DESKTOP-TDA53C2:~$ cat test2.txt
cat: test2.txt: No such file or directory
user@DESKTOP-TDA53C2:~$ cat test3.txt
test3
user@DESKTOP-TDA53C2:~$ ./shred -n 5 test3.txt
shred: test3.txt: pass 1/5...
shred: test3.txt: pass 2/5...
shred: test3.txt: pass 3/5...
shred: test3.txt: pass 4/5...
shred: test3.txt: pass 5/5...
user@DESKTOP-TDA53C2:~$ cat test3.txt
cat: test3.txt: No such file or directory
user@DESKTOP-TDA53C2:~$
```

파일을 복구 불가능하게
덮어쓰고 삭제

-n

n번 덮어쓰고 삭제

shred(-n).c(계속)

```
    return 0;
}

int main(int argc, char *argv[]) {
    int passes = 3; // 기본 덮어쓰기 횟수
    int opt;

    // getopt로 -n 옵션 처리
    while ((opt = getopt(argc, argv, "n:")) != -1) {
        switch (opt) {
            case 'n':
                passes = atoi(optarg); // 덮어쓰기 횟수 설정
                if (passes <= 0) {
                    fprintf(stderr, "shred: invalid number of passes\n");
                    exit(EXIT_FAILURE);
                }
                break;
            default:
                fprintf(stderr, "Usage: %s [-n passes] file...\n", argv[0]);
                exit(EXIT_FAILURE);
        }
    }

    // 파일명이 주어지지 않은 경우
    if (optind >= argc) {
        fprintf(stderr, "Usage: %s [-n passes] file...\n", argv[0]);
        exit(EXIT_FAILURE);
    }
}
```



```
    // 주어진 모든 파일에 대해 shred 수행
    for (int i = optind; i < argc; i++) {
        if (shred_file(argv[i], passes) == -1) {
            exit(EXIT_FAILURE);
        }
    }

    return 0;
}
```

sort.c

```
user@DESKTOP-TDA53C2:~$ cat > mixed.txt << EOF
zebra
apple
Banana
cherry
Apple
EOF
user@DESKTOP-TDA53C2:~$ ./sort mixed.txt
Apple
Banana
apple
cherry
zebra
user@DESKTOP-TDA53C2:~$
```

```
// sort - 텍스트 정렬
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// qsort에서 사용할 문자열 비교 함수
int compare(const void *a, const void *b) {
    return strcmp(*(char**)a, *(char**)b); // 두 문자열을 사전식으로 비교
}

int main(int argc, char *argv[]) {
    FILE *file;
    char **lines = NULL; // 문자열 포인터 배열 (각 줄을 저장)
    char buffer[1024]; // 한 줄을 읽기 위한 임시 버퍼
    int count = 0, capacity = 10; // 현재 줄 수, 배열 용량

    // 명령행 인자 개수 확인
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    file = fopen(argv[1], "r");
    if (!file) {
        perror("sort");
        return 1;
    }
}
```



```
// 문자열 포인터 배열 초기 할당
lines = malloc(capacity * sizeof(char*));

// 파일에서 한 줄씩 읽어서 배열에 저장
while (fgets(buffer, sizeof(buffer), file)) {
    // 배열이 가득 찬 경우 크기를 2배로 확장
    if (count >= capacity) {
        capacity *= 2;
        lines = realloc(lines, capacity * sizeof(char*));
    }
    lines[count] = strdup(buffer); // 문자열 복사하여 저장
    count++;
}

// qsort로 문자열 배열 정렬
qsort(lines, count, sizeof(char*), compare);

// 정렬된 문자열을 출력 후 메모리 해제
for (int i = 0; i < count; i++) {
    printf("%s", lines[i]);
    free(lines[i]); // 각 문자열 메모리 해제
}

free(lines); // 포인터 배열 메모리 해제
fclose(file);
return 0;
}
```

텍스트 정렬

stat.c

파일 또는 디렉토리의 상태 정보를
보여주는 명령어

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <time.h>
#include <pwd.h>
#include <grp.h>

// 파일 권한을 문자열로 출력하는 함수
void print_permissions(mode_t mode) {
    printf("(");
    printf((S_ISDIR(mode)) ? "d" : "-"); // 디렉토리 여부
    printf((mode & S_IRUSR) ? "r" : "-"); // 소유자 읽기 권한
    printf((mode & S_IWUSR) ? "w" : "-"); // 소유자 쓰기 권한
    printf((mode & S_IXUSR) ? "x" : "-"); // 소유자 실행 권한
    printf((mode & S_IRGRP) ? "r" : "-"); // 그룹 읽기 권한
    printf((mode & S_IWGRP) ? "w" : "-"); // 그룹 쓰기 권한
    printf((mode & S_IXGRP) ? "x" : "-"); // 그룹 실행 권한
    printf((mode & S_IROTH) ? "r" : "-"); // 기타 읽기 권한
    printf((mode & S_IWOTH) ? "w" : "-"); // 기타 쓰기 권한
    printf((mode & S_IXOTH) ? "x" : "-"); // 기타 실행 권한
    printf(")");
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s file\n", argv[0]);
        exit(EXIT_FAILURE);
    }
}
```



```
user@DESKTOP-TDA53C2:~$ echo "Hello World" > test.txt
user@DESKTOP-TDA53C2:~$ ./stat test.txt
File: test.txt
Size: 12
Blocks: 8
Mode: 644 (-rw-r--r--)
Uid: 1000/user
Gid: 1000/user
Access: Tue Jun 10 21:38:20 2025
Modify: Tue Jun 10 21:38:20 2025
Change: Tue Jun 10 21:38:20 2025
user@DESKTOP-TDA53C2:~$
```

```
struct stat st;
// stat 시스템 콜로 파일 상태 정보 획득
if (stat(argv[1], &st) == -1) {
    perror("stat");
    exit(EXIT_FAILURE);
}

// 파일 기본 정보 출력
printf(" File: %s\n", argv[1]);
printf(" Size: %ld\n", st.st_size); // 파일 크기 (바이트)
printf(" Blocks: %ld\n", st.st_blocks); // 할당된 블록 수
printf(" Mode: %o ", st.st_mode & 0777); // 8진수 권한 표시
print_permissions(st.st_mode); // 권한을 문자로 표시
printf("\n");

// 소유자와 그룹 정보 출력
struct passwd *pw = getpwuid(st.st_uid); // UID로 사용자 정보 획득
struct group *gr = getgrgid(st.st_gid); // GID로 그룹 정보 획득
printf(" Uid: %d/%s\n", st.st_uid, pw ? pw->pw_name : "unknown");
printf(" Gid: %d/%s\n", st.st_gid, gr ? gr->gr_name : "unknown");

// 파일 시간 정보 출력
printf("Access: %s", ctime(&st.st_atime)); // 마지막 접근 시간
printf("Modify: %s", ctime(&st.st_mtime)); // 마지막 수정 시간
printf("Change: %s", ctime(&st.st_ctime)); // 마지막 상태 변경 시간

return 0;
}
```

tail.c

파일의 마지막 몇 줄 출력

```
user@DESKTOP-TDA53C2:~$ echo -e "line 1\nline 2\nline 3\nline 4\nline 5\nline 6\nline 7\nline 8\nline 9\nline 10\nline 11\nline 12\nline 13\nline 14\nline 15" > test.txt
user@DESKTOP-TDA53C2:~$ ./tail test.txt
line 6
line 7
line 8
line 9
line 10
line 11
line 12
line 13
line 14
line 15
user@DESKTOP-TDA53C2:~$
```

// tail - 파일의 마지막 몇 줄 출력

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

```
int main(int argc, char *argv[]) {
```

```
    FILE *file;
    char *lines[10]; // 마지막 10줄을 저장할 배열
    char buffer[1024]; // 한 줄을 읽기 위한 버퍼
    int count = 0, i;
```

// 명령행 인자 개수 확인

```
if (argc < 2) {
    fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
    return 1;
}
```

```
file = fopen(argv[1], "r");
if (!file) {
    perror("tail");
    return 1;
}
```



```
// 배열 초기화 (모든 포인터를 NULL로 설정)
for (i = 0; i < 10; i++) lines[i] = NULL;
```

```
// 파일에서 한 줄씩 읽어서 순환 배열에 저장
while (fgets(buffer, sizeof(buffer), file)) {
    // 기존 메모리가 있으면 해제
    if (lines[count % 10]) free(lines[count % 10]);
    // 새로운 줄을 복사하여 저장 (순환 배열 방식)
    lines[count % 10] = strdup(buffer);
    count++; // 총 줄 수 증가
}
```

// 출력 시작 위치 계산

```
int start = (count < 10) ? 0 : count % 10; // 10줄 미만이면 0부터, 이상이면 순환 위치부터
int total = (count < 10) ? count : 10; // 출력할 총 줄 수
```

// 마지막 최대 10줄을 순서대로 출력

```
for (i = 0; i < total; i++) {
    printf("%s", lines[(start + i) % 10]); // 순환 배열에서 순서대로 출력
}
```

// 할당된 메모리 해제

```
for (i = 0; i < 10; i++) {
    if (lines[i]) free(lines[i]);
}
fclose(file);
return 0;
```

표준 입력을 파일과
표준 출력에 동시에 출력

tee(-a).c

-a
기존 파일뒤에 덧붙임

```
user@DESKTOP-TDA53C2: $ echo "Hello, world!" | ./tee output.txt
Hello, world!
user@DESKTOP-TDA53C2: $ ls output.txt
output.txt
user@DESKTOP-TDA53C2: $
```

```
user@DESKTOP-TDA53C2: $ echo "Another line." | ./tee -a output.txt
Another line.
user@DESKTOP-TDA53C2: $ cat output.txt
Hello, world!
Another line.
user@DESKTOP-TDA53C2: $
```

```
// tee.c - 표준 입력을 파일과 표준 출력에 동시에 출력
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
int main(int argc, char *argv[]) {
    FILE *fp = NULL;    // 출력 파일 포인터
    char buffer[1024];   // 한 줄씩 읽기 위한 버퍼
    int append_mode = 0; // 추가 모드 플래그

    // -a 옵션 처리 (파일에 추가 모드)
    int file_index = 1;
    if (argc > 1 && strcmp(argv[1], "-a") == 0) {
        append_mode = 1; // 추가 모드 활성화
        file_index = 2;  // 파일명 인덱스 조정
    }

    // 파일명이 주어진 경우 파일 열기
    if (argc > file_index) {
        // append_mode에 따라 "a"(추가) 또는 "w"(쓰기) 모드로 열기
        fp = fopen(argv[file_index], append_mode ? "a" : "w");
        if (!fp) {
            perror("fopen");
            return 1;
        }
    }
}
```



```
// 표준 입력을 읽어서 표준 출력과 파일에 동시에 출력
while (fgets(buffer, sizeof(buffer), stdin)) {
    // 표준 출력에 출력 (화면에 표시)
    printf("%s", buffer);

    // 파일이 열려있으면 파일에도 출력
    if (fp) {
        fprintf(fp, "%s", buffer);
        fflush(fp); // 버퍼를 즉시 파일에 쓰기
    }

    // 파일이 열려있으면 닫기
    if (fp) {
        fclose(fp);
    }

    return 0;
}
```


명령어 실행 시간을 기록

time.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/time.h>
#include <sys/resource.h>

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s command [args...]\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    struct timeval start, end;
    struct rusage usage;

    gettimeofday(&start, NULL);
```



```
// fork로 자식 프로세스 생성
pid_t pid = fork();
if (pid == 0) {
    // 자식 프로세스: 주어진 명령어 실행
    execvp(argv[1], &argv[1]); // argv[1]부터 실행할 명령어와 인자들
    perror("time");
    exit(EXIT_FAILURE);
} else if (pid > 0) {
    // 부모 프로세스: 자식 프로세스 대기 및 시간 측정
    int status;
    // wait4로 자식 프로세스 종료 대기하면서 자원 사용량 정보도 수집
    wait4(pid, &status, 0, &usage);

    // 명령어 실행 종료 시간 기록
    gettimeofday(&end, NULL);

    // 실제 경과 시간 계산 (wall clock time)
    double real_time = (end.tv_sec - start.tv_sec) +
        (end.tv_usec - start.tv_usec) / 1000000.0;

    // 사용자 CPU 시간 계산
    double user_time = usage.ru_utime.tv_sec +
        usage.ru_utime.tv_usec / 1000000.0;

    // 시스템 CPU 시간 계산
    double sys_time = usage.ru_stime.tv_sec +
        usage.ru_stime.tv_usec / 1000000.0;
```

```
// 시간 정보를 stderr로 출력 (표준 에러)
fprintf(stderr, "\nreal\t%.3fs\n", real_time); // 실제 경과 시간
fprintf(stderr, "user\t%.3fs\n", user_time); // 사용자 CPU 시간
fprintf(stderr, "sys\t%.3fs\n", sys_time); // 시스템 CPU 시간

// 자식 프로세스의 종료 상태 반환
return WEXITSTATUS(status);
} else {
    // fork 실패
    perror("fork");
    exit(EXIT_FAILURE);
}
```



```
user@DESKTOP-TDA53C2:~$ ./time ./sleep 5

real    5.003s
user    0.001s
sys     0.000s
user@DESKTOP-TDA53C2:~$
```

sleep.c

```
// sleep - 지정된 시간만큼 대기
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    // 명령행 인자 개수 확인 (프로그래밍 + 초)
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <seconds>\n", argv[0]);
        return 1;
    }

    // 문자열을 정수로 변환하여 대기 시간 설정
    unsigned int seconds = atoi(argv[1]);
    sleep(seconds); // 지정된 초만큼 프로세스 대기

    return 0;
}
```

지정된 시간만큼 대기

touch.c

```
// touch - 빈 파일을 생성하거나, 존재하는 파일의 시간 정보를 현재 시간으로 갱신
#include <stdio.h> // 표준 입출력 함수 사용을 위한 헤더 (예: printf, perror 등)
#include <fcntl.h> // 파일 열기 관련 상수 정의 (예: O_CREAT 등)
#include <unistd.h> // 시스템 호출 함수 사용 (예: open, close 등)
#include <utime.h> // 파일 시간 정보 갱신을 위한 utime 함수 정의

int main(int argc, char *argv[]) {
    int fd; // 파일 디스크립터를 저장할 변수

    // 프로그램 실행 시 인자로 파일 이름이 주어졌는지 확인
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]); // 사용법 안내 출력
        return 1; // 인자가 부족하면 오류 종료
    }

    // 파일 열기 또는 없으면 생성 (쓰기 권한은 없고, 존재 여부만 확인)
    fd = open(argv[1], O_CREAT, 0644);
    /*
     - argv[1] : 사용자가 입력한 파일 이름
     - O_CREAT : 파일이 없으면 새로 생성
     - 0644 : 새로 만들 경우 파일 권한 설정 (rw-r--r--)
     |      | -> 소유자: 읽기/쓰기, 그룹: 읽기, 기타: 읽기
    */

    if (fd == -1) { // 파일 열기(생성)에 실패했는지 확인
        perror("touch"); // 오류 메시지 출력
        return 1; // 프로그램 비정상 종료
    }

    close(fd); // 파일 디스크립터 닫기 (생성만 하고 실제 내용은 안 다루므로 닫기만 하면 됨)

    // 명시적으로 접근 시간(atime)과 수정 시간(mtime)을 현재 시간으로 갱신
    if (utime(argv[1], NULL) == -1) {
        /*
         utime() 함수는 파일의 접근시간, 수정시간을 바꿔줌
         NULL을 넘기면 현재 시간으로 자동 갱신됨
         => touch 명령의 핵심 기능과 동일
        */
        perror("utime"); // 시간 변경 실패 시 에러 메시지 출력
        return 1;
    }

    return 0; // 성공적으로 실행 완료
}
```

빈 파일 생성 또는 시간 갱신

```
user@DESKTOP-TDA53C2:~$ cat myfile.txt
cat: myfile.txt: No such file or directory
user@DESKTOP-TDA53C2:~$ ./touch myfile.txt
user@DESKTOP-TDA53C2:~$ ls -l myfile.txt
-rw-r--r-- 1 user user 0 Jun 12 21:04 myfile.txt
user@DESKTOP-TDA53C2:~$ ./touch myfile.txt
user@DESKTOP-TDA53C2:~$ ls -l myfile.txt
-rw-r--r-- 1 user user 0 Jun 12 21:06 myfile.txt
```

tr.c

문자 변환

```
// tr - 문자 변환
#include <stdio.h>

int main(int argc, char *argv[]) {
    int ch;
    char *from, *to;
    // 변환할 문자 집합과 변환될 문자 집합 두 개의 인자가 필요
    if (argc < 3) {
        fprintf(stderr, "Usage: %s <from> <to>\n", argv[0]);
        return 1;
    }
    from = argv[1]; // 변환할 문자들 (예: "abc")
    to = argv[2];   // 변환될 문자들 (예: "xyz")
    // 표준 입력에서 한 문자씩 읽어서 처리
    while ((ch = getchar()) != EOF) {
        int found = 0;
        // from 문자열에서 현재 문자와 일치하는 것을 찾을 때까지
        for (int i = 0; from[i] && !found; i++) {
            // 일치하는 문자를 찾았고 대응하는 to 문자가 있으면
            if (ch == from[i] && to[i]) {
                putchar(to[i]); // 대응 문자로 변환하여 출력
                found = 1;
            }
        }
        // 변환할 문자가 아니면 원본 그대로 출력
        if (!found) putchar(ch);
    }
    return 0;
}
```

```
user@DESKTOP-TDA53C2:~$ gcc -o tr tr.c
user@DESKTOP-TDA53C2:~$ echo "hello" | ./tr 'el' 'ip'
hippo
user@DESKTOP-TDA53C2:~$
```

uniq.c

```
// uniq - 중복 줄 제거
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]) {
    FILE *file;
    char current[1024], previous[1024] = ""; // 현재 줄과 이전 줄을 저장할 버퍼

    // 파일명이 인자로 주어져야 함
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    file = fopen(argv[1], "r");
    if (!file) {
        perror("uniq");
        return 1;
    }

    // 파일에서 한 줄씩 읽어서 처리
    while (fgets(current, sizeof(current), file)) {
        // 현재 줄이 이전 줄과 다른 줄이면 출력 (연속된 중복 줄만 제거)
        if (strcmp(current, previous) != 0) {
            printf("%s", current);
            strcpy(previous, current); // 현재 줄을 이전 줄로 저장
        }
    }

    fclose(file);
    return 0;
}
```

중복 줄 제거

```
user@DESKTOP-TDA53C2:~$ gcc -o uniq uniq.c
user@DESKTOP-TDA53C2:~$ cat test.txt
Line 1
Line 1
Line 1
Line 1
Line 2
Line 2
Line 2
Line 2
user@DESKTOP-TDA53C2:~$ ./uniq test.txt
Line 1
Line 2
user@DESKTOP-TDA53C2:~$
```

uptime.c

시스템 가동시간과 부하 평균 시간을 구함

```
user@DESKTOP-TDA53C2:~$ ./uptime
20:03:54 up 1:32, load average: 0.00, 0.00, 0.00
```

```
// uptime.c - 시스템 가동 시간과 부하 평균(load average)을 표시하는 프로그램

#include <stdio.h> // 표준 입출력 함수 사용 (printf 등)
#include <stdlib.h> // 일반 함수 사용 (exit, malloc 등)
#include <time.h> // 시간 관련 함수 (time, localtime 등)
#include <sys/sysinfo.h> // sysinfo 시스템 호출을 위한 헤더

int main() {
    struct sysinfo info; // 시스템 정보를 담는 구조체
    time_t current_time; // 현재 시간(초 단위) 저장 변수
    struct tm *time_info; // 현재 시간을 구조화한 형태 (시, 분, 초 등)

    // sysinfo 시스템 호출을 통해 시스템 정보를 가져옴
    // 실패 시 perror로 오류 메시지 출력 후 종료
    if (sysinfo(&info) != 0) {
        perror("sysinfo");
        return 1;
    }

    // 현재 시간을 초 단위로 얻음 (Unix epoch 기준 1970-01-01부터 초 계산)
    current_time = time(NULL);

    // current_time을 지역 시간 구조체로 변환 (시, 분, 초, 월, 일 등 세부 정보 포함)
    time_info = localtime(&current_time);

    // info.uptime은 시스템 부팅 후 경과한 총 초(seconds)
    // 이를 일, 시, 분 단위로 나눔
    int uptime_days = info.uptime / 86400; // 1일 = 86400초
    int uptime_hours = (info.uptime % 86400) / 3600; // 나머지 시간 계산 (1시간 = 3600초)
    int uptime_minutes = (info.uptime % 3600) / 60; // 나머지 분 계산 (1분 = 60초)
```



```
// 현재 시간을 시:분:초 형식으로 출력, 예: 20:02:22
printf("%02d:%02d:%02d up ",
       time_info->tm_hour, time_info->tm_min, time_info->tm_sec);

// 만약 부팅 후 1일 이상 지났으면 "N day(s)," 형식으로 출력
if (uptime_days > 0) {
    // 하루 이상이면 복수형 s 붙이기
    printf("%d day%s, ", uptime_days, uptime_days > 1 ? "s" : "");
}

// 부팅 후 시간과 분을 출력, 예: 1:31 (1시간 31분)
printf("%d:%02d, ", uptime_hours, uptime_minutes);

// 부하 평균(load average) 출력
// info.loads 배열은 부하 평균을 1/65536 단위로 저장하므로 나누어서 실수로 변환
// 1분, 5분, 15분 동안의 부하 평균을 각각 소수점 두 자리까지 표시
printf("load average: %.2f, %.2f, %.2f\n",
       (float)info.loads[0] / 65536.0,
       (float)info.loads[1] / 65536.0,
       (float)info.loads[2] / 65536.0);

return 0; // 프로그램 정상 종료
```

WC.C

단어, 줄, 문자 수 세기

```
// wc - 단어, 줄, 문자 수 세기
#include <stdio.h>

int main(int argc, char *argv[]) {
    FILE *file;
    int lines = 0, words = 0, chars = 0; // 줄 수, 단어 수, 문자 수 카운터
    char ch, prev = ' '; // 현재 문자와 이전 문자 (단어 구분을 위해)

    // 파일명이 인자로 주어져야 함
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    file = fopen(argv[1], "r");
    if (!file) {
        perror("wc");
        return 1;
    }

    // 파일에서 한 문자씩 읽어서 처리
    while ((ch = fgetc(file)) != EOF) {
        chars++; // 문자 수 증가
        if (ch == '\n') lines++; // 개행 문자면 줄 수 증가
    }
}
```



```
// 현재 문자가 공백/탭/개행이고 이전 문자가 그렇지 않으면 단어 끝
if (ch == ' ' || ch == '\n' || ch == '\t') {
    if (prev != ' ' && prev != '\n' && prev != '\t') words++;
}
prev = ch; // 이전 문자로 저장
}

// 파일이 공백으로 끝나지 않는 경우 마지막 단어 처리
if (prev != ' ' && prev != '\n' && prev != '\t' && chars > 0) words++;

// "줄수 단어수 문자수 파일명" 형식으로 출력
printf("%d %d %d %s\n", lines, words, chars, argv[1]);
fclose(file);
return 0;
}
```

```
user@DESKTOP-TDA53C2:~$ cat test.txt
Line 1
Line 1
Line 1
Line 1
Line 2
Line 2
Line 2
Line 2
user@DESKTOP-TDA53C2:~$ ./wc test.txt
8 16 56 test.txt
user@DESKTOP-TDA53C2:~$
```

whoami.c

```
// whoami - 현재 사용자명 출력
#include <stdio.h>
#include <unistd.h>
#include <pwd.h>

int main() {
    // getuid(): 현재 프로세스의 사용자 ID를 얻음
    // getpwuid(): 사용자 ID로 passwd 구조체(사용자 정보)를 얻음
    struct passwd *pw = getpwuid(getuid());
    if (pw) {
        // pw_name: passwd 구조체의 사용자명 필드
        printf("%s\n", pw->pw_name);
    } else {
        perror("whoami");
        return 1;
    }
    return 0;
}
```

현재 사용자명 출력

```
user@DESKTOP-TDA53C2:~$ gcc -o whoami whoami.c
user@DESKTOP-TDA53C2:~$ ./whoami
user
user@DESKTOP-TDA53C2:~$ █
```


바이너리 파일을 16진수
(hex) 덤프 형태로 출력

xxd.c

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

int main(int argc, char *argv[]) {
    FILE *file = stdin; // 기본적으로 표준 입력 사용

    // 파일명이 주어지면 해당 파일을 열음
    if (argc > 1) {
        file = fopen(argv[1], "rb"); // "rb": 바이너리 읽기 모드
        if (!file) {
            perror("xxd");
            exit(EXIT_FAILURE);
        }
    }

    unsigned char buffer[16]; // 한 번에 16바이트씩 읽어서 처리
    size_t bytes_read;
    long offset = 0; // 파일에서의 현재 위치(오프셋)

    // 16바이트씩 읽어서 hex dump 형식으로 출력
    while ((bytes_read = fread(buffer, 1, 16, file)) > 0) {
        printf("%08lx: ", offset); // 오프셋을 8자리 16진수로 출력
```

```
        // 표준 입력이 아닌 경우에만 파일 닫기
        if (file != stdin) {
            fclose(file);
        }

        return 0;
    }
}
```

```
user@DESKTOP-TDA53C2:~$ cat test.txt
asdfasdfasdf
asdfasdf
asdf
12314
asdf
querqwer
user@DESKTOP-TDA53C2:~$ ./xxd test.txt
00000000: 6173 6466 6173 6466 6173 6466 0a61 7364  asdfasdfasdf.asd
00000010: 6661 7364 666a 6173 6466 0a31 3233 3134  fasdf.asdf.12314
00000020: 0a61 7364 666a 7177 6572 7177 6572 0a   .asdf.querqwer.
user@DESKTOP-TDA53C2:~$
```

```
    // 16바이트를 2자리 16진수로 출력 (hex 부분)
    for (size_t i = 0; i < 16; i++) {
        if (i < bytes_read) {
            printf("%02x", buffer[i]); // 2자리 16진수로 출력
        } else {
            printf(" "); // 데이터가 없으면 공백
        }
        if (i % 2 == 1) printf(" "); // 2바이트마다 공백 추가
    }

    printf(" ");

    // ASCII 표현 부분 출력
    for (size_t i = 0; i < bytes_read; i++) {
        // 출력 가능한 문자면 그대로, 아니면 '.'으로 표시
        if (isprint(buffer[i])) {
            printf("%c", buffer[i]);
        } else {
            printf(".");
        }
    }

    printf("\n");
    offset += bytes_read; // 다음 줄을 위해 오프셋 증가
}
```

yes.c

```
// yes.c - 지정된 문자열을 무한히 출력
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
    const char *output = "y"; // 기본값은 "y"

    // 인자가 주어지면 해당 문자열들을 출력할 내용으로 사용
    if (argc > 1) {
        // 모든 인수를 공백으로 연결하여 하나의 문자열로 만들
        static char buffer[1024];
        strcpy(buffer, argv[1]); // 첫 번째 인자 복사
        for (int i = 2; i < argc; i++) {
            strcat(buffer, " "); // 공백 추가
            strcat(buffer, argv[i]); // 다음 인자 연결
        }
        output = buffer; // 연결된 문자열을 출력할 내용으로 설정
    }

    // 무한 루프로 지정된 문자열을 계속 출력
    while (1) {
        printf("%s\n", output);
        fflush(stdout); // 출력 버퍼를 즉시 비워서 바로 출력되도록 함
    }


    return 0;
}
```

```
test.txt
test.txt
test.txt
test.txt
test.txt
test.txt
test.txt
test.txt
test.txt
test.txt
^C
```




지정된 문자열을 무한히 출력

깃허브

main

 notecoding Update README.md 04e093 · 5 days ago 🕒 58 Commits

📁 0307	Update readme.md	last week
📁 0314	Update readme.md	5 days ago
📁 0321	Update readme.md	5 days ago
📁 0329	Add files via upload	3 months ago
📁 0404	Update readme.md	5 days ago
📁 0411	Update readme.md	5 days ago
📁 0418	Update readme.md	5 days ago
📁 0502	Update readme.md	5 days ago
📁 0509	Update readme.md	5 days ago
📁 0517	Update readme.md	5 days ago
📁 0523	Update readme.md	5 days ago
📁 0530	Update readme.md	last week
📄 README.md	Update README.md	5 days ago

 README  

시스템 프로그래밍 (System Programming)

운영체제와 하드웨어에 가까운 레벨에서 프로그래밍하는 기법과 개념들을 학습하고 실습하는 저장소입니다.

0314

Name	Last commit message
..	
readme.md	Update readme.md
readme.md	

Linux 파일 시스템 구조

Commits

History for `system` / 0314 on `main`

Commits on Jun 7, 2025

Update readme.md

notecoding authored 16 minutes ago

Commits on Mar 29, 2025

Create readme.md

notecoding authored on Mar 29

End of commit history for this file

0321

Name	Last commit message
..	
hello	Create hello.c
namenuminput	Create mission.c
readme.md	Update readme.md

readme.md

C 파일의 리눅스 실행 흐름 <리눅스에서 C파일 실행하기>

Commits

History for `system` / 0321 on `main`

Commits on Jun 7, 2025

- Update readme.md**
notecoding authored 13 minutes ago
- Update readme.md**
notecoding authored 16 minutes ago

Commits on Mar 21, 2025

- Create mission.c**
notecoding authored on Mar 21
- Create readme.md**
notecoding authored on Mar 21
- Create hello.c**
notecoding authored on Mar 21
- Create readme.md**
notecoding authored on Mar 21
- Create readme.md**
notecoding authored on Mar 21

End of commit history for this file

0328

0329로 되어있는 제목은 제목을
잘못지어서 그렇습니다

Name	Last commit message
..	
a.c	Add files via upload
readme.md	Create readme.md
readme.md	
c파일 대문자 입력하면 소문자 소문자입력하면 대문자로 변환하기	

Commits

History for [system](#) / [0329](#) on [main](#)

Commits on Mar 29, 2025

Add files via upload

 notecoding authored on Mar 29

Create readme.md

 notecoding authored on Mar 29

End of commit history for this file

0404

Name	Last commit message
..	
report	Add files via upload
readme.md	Update readme.md
readme.md	

생성형 AI (Generative AI) 및 레포트(ai를 활용한 과제)

Commits

History for `system` / `0404` on `main`

Commits on Jun 7, 2025

- Update readme.md**
notecoding authored 14 minutes ago
- Update readme.md**
notecoding authored 15 minutes ago

Commits on Apr 6, 2025

- Add files via upload**
notecoding authored on Apr 6
- Create readme.md**
notecoding authored on Apr 6
- Create readme.md**
notecoding authored on Apr 6

End of commit history for this file

0411

Name	Last commit message
..	
readme.md	Update readme.md
readme.md	

생성형 AI 활용 방법

Commits

History for `system` / `0411` on `main`

Commits on Jun 7, 2025

Update readme.md

 notecoding authored 13 minutes ago

Commits on Apr 24, 2025

Create readme.md

 notecoding authored on Apr 24

End of commit history for this file

0418

Name	Last commit message
..	
readme.md	Update readme.md
readme.md	

리눅스 명령어 가이드

Commits

History for [system](#) / **0418** on [main](#)

- Commits on Jun 7, 2025
 - Update readme.md**
notecoding authored 11 minutes ago
- Commits on Apr 24, 2025
 - Create readme.md**
notecoding authored on Apr 24
- End of commit history for this file

0502

Name	Last commit message
..	
copy.c	Create copy.c
fofen.c	Create fofen.c
fsize.c	Create fsize.c
readme.md	Update readme.md
readme.md	

컴퓨터 구조와 시스템 호출

Commits

History for `system` / `0502` on `main`

Commits on Jun 7, 2025

- Update readme.md**
notecoding authored 9 minutes ago

Commits on May 6, 2025

- Create copy.c**
notecoding authored on May 6
- Create fsize.c**
notecoding authored on May 6
- Create fofen.c**
notecoding authored on May 6
- Create readme.md**
notecoding authored on May 6

End of commit history for this file

0509

Name	Last commit message
..	
cptime.c	Create cptime.c
fchmod.c	Create fchmod.c
ftype.c	Create ftype.c
list1.c	Update list1.c
list2.c	Create list2.c
readme.md	Update readme.md
readme.md	

리눅스 파일 시스템 구조

Commits

History for [system](#) / 0509 on [main](#)


Commits on Jun 7, 2025

Update readme.md


 notecoding authored 6 minutes ago

Commits on May 14, 2025


Create list2.c

 notecoding authored 3 weeks ago


Update list1.c

 notecoding authored 3 weeks ago


Create list1.c

 notecoding authored 3 weeks ago

Create cptime.c

 notecoding authored 3 weeks ago


Create fchmod.c

 notecoding authored 3 weeks ago

Create ftype.c

 notecoding authored 3 weeks ago

Create readme.md

 notecoding authored 3 weeks ago

End of commit history for this file

0517로 되어있는 제목은 제목을
잘못지어서 그렇습니다

0516

Name	Last commit message
...	
execute1.c	Add files via upload
execute2.c	Add files via upload
execute3.c	Add files via upload
fork1.c	Add files via upload
fork2.c	Add files via upload
fork3.c	Add files via upload
forkwait.c	Add files via upload
pgrep1.c	Add files via upload
pgrep2.c	Add files via upload
readme.md	Update readme.md
redirect1.c	Add files via upload
redirect2.c	Add files via upload
syscall.c	Add files via upload
system.c	Add files via upload
waitpid.c	Add files via upload
readme.md	

리눅스 프로세스 구조

Commits

History for [system](#) / 0517 on [main](#)

Commits on Jun 7, 2025

Update readme.md

 notecoding authored 4 minutes ago

Commits on May 23, 2025

Update readme.md


 notecoding authored 2 weeks ago

Commits on May 22, 2025

Add files via upload

 notecoding authored 2 weeks ago

Update readme.md

 notecoding authored 2 weeks ago

Commits on May 21, 2025

Create readme.md

 notecoding authored 2 weeks ago

End of commit history for this file

0523

Name	Last commit message
..	
fork1.c	Update fork1.c
fork2.c	Update fork2.c
fork3.c	Update fork3.c
fork4.c	Update fork4.c
fork5.c	Update fork5.c
readme.md	Update readme.md
readme.md	

리눅스 프로세스 제어

Commits

History for `system` / 0523 on `main`

Commits on Jun 7, 2025

- Update readme.md**
notecoding authored 3 minutes ago

Commits on May 23, 2025

- Update fork5.c**
notecoding authored 2 weeks ago
- Update fork1.c**
notecoding authored 2 weeks ago
- Update fork2.c**
notecoding authored 2 weeks ago

0530

Name	Last commit message
..	
readme.md	Update readme.md
readme.md	

정규표현식(Regular Expression)이란?

Commits

History for `system` / 0530 on `main`

Commits on Jun 3, 2025

- Update readme.md**
notecoding authored 4 days ago
- Create readme.md**
notecoding authored 4 days ago

End of commit history for this file

점수 합산

0314 2주 후 제출 -0.6

0411 1주 후 제출 -0.3

명령어50개 +15

최종 점수:29.1점