11장 입출력과 운영체제

printf()

- 특징
  - 임의의 개수의 인자 출력
  - 간단한 변환 명세나 형식을 사용한 출력 제어

printf()

printf(*control_string, other_argument*)

- 예

```
printf("she sells %d %s for $%f", 99, "sea
 shells", 3.77);
```

control_string: "she sells %d %s for $%f"

other_arguments: 99, "sea shells", 3.77

- 변환 명세는 %로 시작하여 변환문자로 끝남

printf()

printf() 변환 문자	
변환 문자	대응되는 인자의 출력 형태
c	문자
d,i	10진 정수
u	부호 없는 10진 정수
o	부호 없는 8진 정수
x, X	부호 없는 16진 정수
e	부동 소수점 수; 예: 7.123000e+00
E	부동 소수점 수; 예: 7.123000E+00
f	부동 소수점 수; 예: 7.123000
g	e형식과 f 형식 중 짧은 쪽
G	E형식과 f 형식 중 짧은 쪽
s	문자열
p	대응되는 인자가 void 포인터임; 그 값이 16진수 형태로 출력됨
n	대응되는 인자는 정수형 포인터로서 그 값은 현재까지 출력된 문자의 개수임; 인자는 변환되지 않음
%	%%의 형식으로 %를 출력 스트림에 씀; 대응되는 인자는 없음.

printf()

- 예제

```
printf("she sells %d %s for $%f", 99, "sea
  shells", 3.77);
```

변환 형식	대응되는 인자
%d	99
%s	"sea shells"
%f	3.77

printf()

- %와 변환문자 사이에 올 수 있는 것들
  - 플래그 문자들
    - 빼기 기호
    - 더하기 기호
    - 공백
    - # 기호
    - 0
  - 필드 폭
  - 정밀도
  - h 또는 l
  - L

printf()

선언과 초기화			
`char c = 'A', s[] = "Blue moon!";`			
변환 형식	대응되는인자	필드 내에서 출력형태	비고
`%c`	`c`	`"A"`	필드 폭 1 (디폴트)
`%2c`	`c`	`" A"`	필드 폭 2, 우측 정렬
`%-3c`	`c`	`"A  "`	필드 폭 3, 좌측 정렬
`%s`	`s`	`"Blue moon!"`	필드 폭 10 (디폴트)
`%3s`	`s`	`"Blue moon!"`	공간이 더 필요함
`%.6s`	`s`	`"Blue m"`	정밀도 6
`%-11.8s`	`s`	`"Blue moo  "`	정밀도 8, 좌측 정렬

printf()

선언과 초기화			
int i = 123; double x = 0.123456789;			
변환 형식	대응되는인자	필드 내에서 출력형태	비고
%d	i	"123"	필드 폭 3 (디폴트)
%05d	i	"00123"	영으로 채움
%7o	i	" 173"	우측 정렬, 8진수
%-9x	i	"7b "	좌측 정렬, 16진수
%-#9x	i	"0x7b "	좌측 정렬, 16진수
%10.5f	x	" 0.12346"	필드 폭 10, 정밀도 5
%-12.5e	x	"1.23457-01 "	좌측 정렬, e-형식

scanf()

scanf(*control_string, other_argument*)

- 예

```
char      a, b, c, s[100];
int       n;
double    x;
scanf("%c%c%c%d%s%lf", &a,&b,&c,&n,s,&x);
```

  - *control_string*: "%c%c%c%d%s%lf"
  - *other_arguments*: &a, &b, &n, s, &x

scanf()

scanf( ) 변환 문자		
변환 문자	입력 스트림에서 대응되는 문자	대응 인자의 포인터 형
c	공백을 포함한 모든 문자	**char**
d, i	10진 정수 (부호는 옵션)	**integer**
u	10진 정수 (부호는 옵션)	**unsigned integer**
o	8진수 (부호는 옵션)	**unsigned integer**
x, X	16진수 (부호는 옵션)	**unsigned integer**
e, E, f, g, G	실수 (부호는 옵션)	**floating type**
s	공백 없는 문자열	**Char**
p	printf() 함수의 %p에 의해 출력되는 것으로일반적으로 부호 없는 16진 정수임	**void ***
n, %, [...]	다음 표 참조	

scanf()

scanf( ) 변환 문자	
변환 문자	설명
n	입력 스트림의 문자와 짝을 이루지 않는다. 대응되는 인자는 정수형 포인터로서, 지금까지 읽어들인 문자의 개수를 저장한다.
%	입력 스트림에서 하나의 %와 짝을 이룬다. 대응되는 인자는 없다.
[...]	각괄호 [ ] 안에 있는 문자들을 스캔 집합이라 한다. 이것은 무엇이 짝을 이루는가를 결정한다. (아래 설명을 참조하여라.) 대응되는 인자는 문자 배열의 기본 주소에 대한 포인터이고, 이 배열은 끝에 자동적으로 추가되는 널 문자를 포함하여 대응되는 모든 문자들을 포함할 만큼 큰 크기를 가져야 한다.

scanf()

- 제어 문자열은 다음과 같은 것을 포함할 수 있음
  - 여백
  - %이외의 공백문자가 아닌 일반 문자
  - %로 시작해서 변환 문자로 끝나는 변환 명세
  - h
  - l
  - L

scanf() 예제

```
int    i;
char   c;
char   string[15];
scanf("%d , %*s %% %c %5s %s", &i, &c, string,
  &string[5]);
```

* 입력 스트림 : `45 , ignore_this  %  C read_in_this**`
  - i : 45
  - c : C
  - string[0-5] : "read_"
  - string[5-14] : "in_this**"
  - scanf()는 4를 리턴

fprintf()/fscanf()

- 각각 printf()와 scanf() 함수의 파일 버전
- 함수 원형

```
int fprintf(FILE *fp, const char *format, …)
int fscanf(FILE *fp, const char *format, …)
```

- fprintf(stdout,...);와 printf(...);는 같은 의미
- fscanf(stdin, ...);은 scanf(...);와 같은 의미

sprintf()/sscanf()

- 각각 printf()와 scanf() 함수의 문자열 버전
- 함수 원형

```
int sprintf(char *, const char *, ...);
int sscanf(const char *, const char *, ...);
```

fopen()

- fopen(filename, mode) 형태의 함수 호출은 filename 파일을 mode에 지정된 모드로 열고, 파일 포인터를 리턴함

fopen()

- 모드

모드	의미
"r"	읽기 위해 문서 파일 열기
"w"	쓰기 위해 문서 파일 열기
"a"	첨부하기 위해 문서 파일 열기
"rb"	읽기 위해 이진 파일 열기
"wb"	쓰기 위해 이진 파일 열기
"ab"	첨부하기 위해 이진 파일 열기

  - 모드 뒤의 +는 파일을 읽기와 쓰기로 모두 연다는 것을 의미함

fopen()/fclose()

- 파일 열기와 닫기의 전형적인 예제 코드

```
#include <stdio.h>
int main(void){
    int    a, sum = 0;
    FILE   *ifp, *ofp;
    ifp = fopen("my_file", "r");
    ofp = fopen("outfile", "w");
    .....
    fclose(ifp);
    fclose(ofp);
}
```

파일의 임의의 위치 접근

- **ftell(file_ptr)**
  - 파일 위치 지시자의 현재 값을 리턴
- **fseek(file_ptr, offset, place)**
  - 파일 위치 지시자를 place부터 offset 바이트 떨어진 곳을 나타내는 값으로 설정함
  - place의 값은 0(SEEK_SET), 1(SEEK_CUR), 2(SEEK_END) 중 하나가 될 수 있는데, 이것들은 각각파일의 처음, 현재 위치, 파일의 끝을 나타냄

파일의 임의의 위치 접근

- 파일을 역으로 출력하는 프로그램

```
#include <stdio.h>
int main(void){
   char  fname[100];  int   c;  FILE  *ifp;
   fprintf(stderr, "\nInput a filename: ");
   scanf("%s", fname);
   ifp = fopen(fname, "rb");
   fseek(ifp, 0, SEEK_END);
   fseek(ifp, -1, SEEK_CUR);
   while (ftell(ifp) > 0) {
      c = getc(ifp);
      putchar(c);
      fseek(ifp, -2, SEEK_CUR) ; }
   return 0;
}
```

파일 기술자 입출력

- 파일 기술자 : 파일과 연관된 음이 아닌 정수
- 프로그램 실행시 자동으로 열리는 파일 및 기술자

파일명	연관된 파일 기술자
표준 입력	0
표준 출력	1
표준 에러	2

파일 기술자 입출력

- 전형적인 예제

```
int main(void){
    char mybuf[100], *p;
    int in_fd, out_fd;
    in_fd = open("my_file", O_RDONLY);
    out_fd = open("outfile", O_WRONLY, 0600);
    .....
    n = read(in_fd, mybuf, 100);
    write(out_fd, mybuf, n);
    .....
    close(in_fd);
    close(out_fd);
}
```

파일 접근 허가

- UNIX/LINUX 파일에는 사용자별 접근허가가 있음
- 사용자 종류 : 소유자(u), 그룹(g), 다른 사용자(o)
- 접근 종류 : 읽기(r), 쓰기(w), 실행(x)
- 사용자 별 접근 종류 설정 함 (9 bit)
- 예 (ls -l의 결과)

```
-rwxr--r-- 1 kmh  prof  57 10월 20 13:04 im.c
```

파일 접근 허가

- 연상 기호 및 8진수

연상기호	2진 표현	8진 표현
r--	100	04
-w-	010	02
--x	001	01
rw-	110	06
r-x	101	05
-wx	011	03
rwx	111	07

파일 접근 허가

- 파일 접근 허가의 예

연상기호	8진 표현
rw-------	0600
rw----r--	0604
rwxr-xr-x	0755
rwxrwxrwx	0777

파일 접근 허가

- open() 함수의 3 번째 인자에 파일 접근 허가 명시

```
out_fd = open("outfile", O_WRONLY, 0600);
```

- UNIX/LINUX에서 파일 접근 허가 변경 방법

```
$ chmod 666 file
$ chmod a+w file
$ chmod o-w file
```

C 프로그램에서 명령어 실행

- system() 함수를 사용하면 프로그램에서 운영체제 명령어를 실행할 수 있음
- 프로그램 내에서 명령어 라인으로부터 입력된 파일을 *vi*를 사용하여 편집하는 코드

```
char   command[MAXSTRING];
sprintf(command, "vi %s", argv[1]);
printf("vi on the file %s is coming up ...",
  argv[1]);
system(command);
```

popen()을 통한 프로세스 생성

- popen() : 파이프를 생성하고 프로세스를 생성하여 이 파이프를 통해 통신할 수 있게 함
- pclose()를 사용하여 닫음
- 예제

```
FILE *ifp;
int c;
ifp = popen("ls", "r");
. . .
c = getc(ifp);
. . .
pclose(ifp);
```

환경 변수

- 환경변수 출력 프로그램

```
#include <stdio.h>
int main(int argc, char *argv[], char *env[]){
   int   i;
   for (i = 0; env[i] != NULL; ++i)
      printf("%s\n", env[i]);
   return 0;
}
```

컴파일러 옵션

- cc/gcc 옵션

컴파일러에 유용한 옵션들	
-c	컴파일만 수행함, 대응되는 *.o* 파일 생성
-g	디버거를 사용할 수 있는 코드 생성
-o *name*	*name*에 실행 가능한 코드를 출력
-p	프로파일러를 사용할 수 있는 코드 생성
-v	많은 정보를 생성
-D*name*=*def*	*.c* 파일 상단에 #define *name def* 행을 삽입
-E	전처리기만 호출
-I*dir*	*dir* 디렉토리에서 #include 파일들을 찾음
-M	makefile 생성
-O	코드 최적화 수행
-S	대응되는 *.s* 파일에 어셈블러 코드 생성

프로파일러

- UNIX에서 *cc*에 *-p* 옵션을 사용하면 컴파일러로 하여금 각 루틴이 호출된 횟수를 알려주는 코드를 생성하게 함
- 이렇게 생성된 코드는 라이브러리 함수인 monitor()를 자동으로 호출하고 *mon.out* 파일을 생성함
- *prof* 명령을 사용하면 *mon.out* 파일을 사용하여 실행 프로파일을 생성함

프로파일러 사용 예

```
$ cc -p -o quicksort main.c quicksort.c
$ quicksort
$ prof quicksort
```

prof 출력

```
$ prof quicksort
% time    cumsecs  #call   ms/call   name
   46.9       7.18   9931      0.72   _partition
   16.1       9.64      1  2460.8     _main
   11.7      11.43  19863      0.09   _find_pivot
   10.8      13.08                    mcount
    6.9      14.13  50000      0.02   _rand
    6.4      15.12  19863      0.05   _quicksort
    1.4      15.33                    _monstartup
    0.0      15.33      1      0.00   _gettimeofday
    0.0      15.33      1      0.00   _profil
    0.0      15.33      1      0.00   _srand
    0.0      15.33      1      0.00   _time
```

라이브러리

- ar : 라이브러리 아카이브를 생성, 변경, 추출하는 명령
- 라이브러리에 있는 파일 보기 예

```
$ ar t /lib/libc.a
```

- 라이브러리 생성

```
$ ar rug file.a file1.o file2.o . . .
$ ranlib file.a
```

- 컴파일

```
$ cc main.c file.a
```

시간 측정

- 헤더 파일 time.h에 함수 원형이 정의된 함수들을 사용하여 컴퓨터 클록을 접근할 수 있음
- clock() : 프로그램이 사용한 시간에 대한 시스템의 최대 근사치 리턴
- time() : 1970년 1월 1일부터 경과된 시간을 초 단위로 환산하여 리턴

make

- 대형 프로그램을 한 파일에 저장하는 것은 매우 비효율적이고 많은 비용이 소요됨
- 이러한 대형 프로그램은 여러 개의 *.c* 파일로 나누어 작성하고 필요에 따라 개별적으로 컴파일하는 것이 훨씬 효율적임

  → make의 필요성
- *make* 유틸리티는 원시 파일을 관리하고 라이브러리와 그에 관련된 헤더 파일을 편리하게 접근하도록 하는 데 사용됨
- make는 makefile이라는 파일을 읽고, 종속관계 트리를 만들어 필요한 동작을 수행함

makefile

- *makefile*은 종속관계와 동작을 명시하는 *규칙*이라고 하는 목록들로 구성됨
- 각 규칙의 형식

```
file_1 file_2 … : source_file_1 source_file_2 …
<tab>command
```

- 예

```
all: main.o sort.o
   cc -o sort main.o sort.o
main.o: main.c sum.h
   cc -c main.c
```

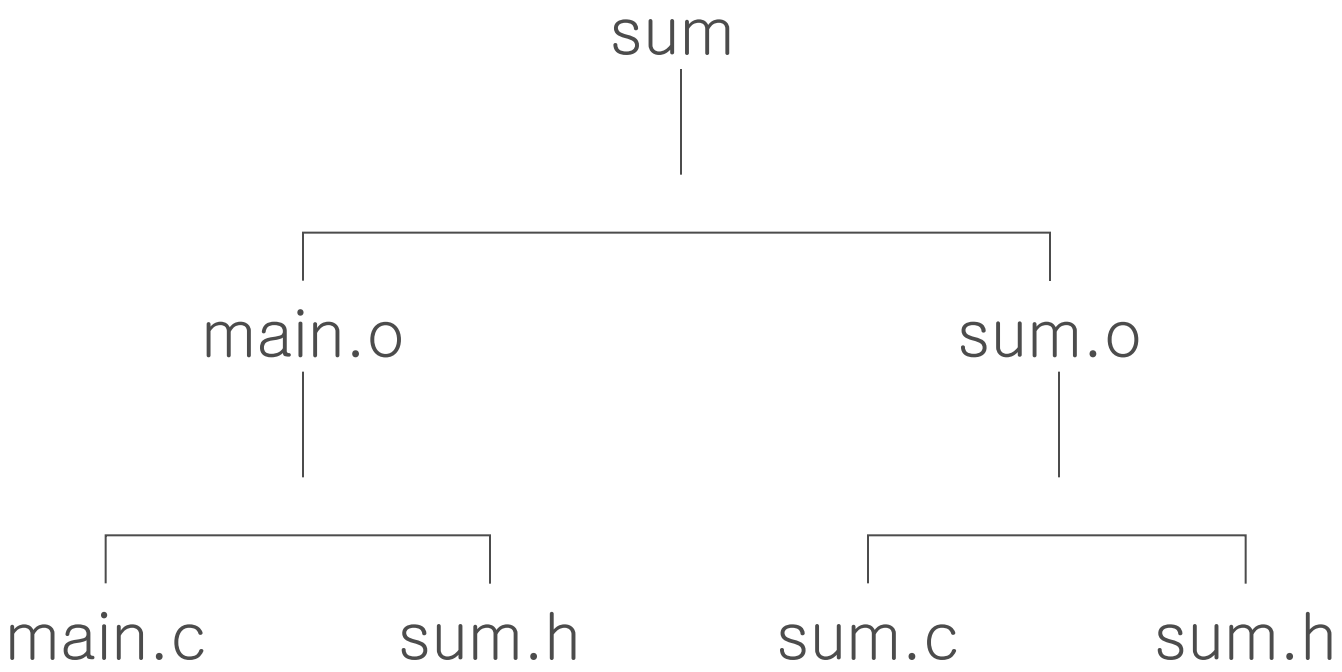
make 예제

- 예제
  - *main.c*, *sum.c*, *sum.h* 로 구성된 프로그램

```
sum: main.o sum.o
        cc -o sum main.o sum.o
main.o: main.c sum.h
        cc -c main.c
sum.o: sum.c sum.h
        cc -c sum.c
```

  - make *rule_label* 수행 (rule_label이 없으면 makefile의 첫 번째 규칙을 수행함)

종속트리



makefile 예제

- 예제 2

```
sum: main.o sum.o
        cc -o sum main.o sum.o
main.o: sum.h
        cc -c main.c
sum.o: sum.h
        cc -c sum.c
```

- 예제 3

```
sum: main.o sum.o
        cc -o sum main.o sum.o
main.o sum.o: sum.h
        cc -c $*.c
```

touch

- 파일에 새로운 시간 설정
- make는 파일 시간을 보고 동작할 것인지 결정
- 다시 전체를 컴파일 하고 싶을 때 touch를 수행 후 make 함
- 예

```
$ touch *.c
$ make
```