# Input/Output and the Operating Systems

Fall 2015

Jinkyu Jeong
(jinkyu@skku.edu)

# I/O Functions

- **Formatted I/O**
  - printf( ) and scanf( )
  - fprintf( ) and fscanf( )
  - sprintf( ) and sscanf( )

```
int printf(const char* format, …);
int scanf(const char* format, …);

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

int sprintf(char *s, const char* format, …);
int sscanf(const char *s, const char* format, …);
```

# Formating Rule (1)

- control_string
- other_arguments

```
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)

| Conversion character | Corresponding argument |
|---|---|
| %d | 99 |
| %s | "sea shells" |
| %f | 3.77 |

# Formating Rule (2)

- **Decimal numbers**

| Types | Conversion character | |
|---|---|---|
| | Signed | Unsigned |
| `char` | `%hhd` | `%hhu` |
| `short` | `%hd` | `%hu` |
| `int` | `%d` | `%u` |
| `long` | `%ld` | `%lu` |
| `long long` | `%lld` | `%llu` |

- **Hexadecimal**
  - d → x
- **Octal**
  - d → o

# Formating Rule (3)

- Floating-point numbers

| Types | Conversion character | |
|---|---|---|
| | Decimal | e-format |
| `float` | `%f` | `%e` |
| `double` | `%f` | `%e` |
| `long double` | `%Lf` | `%Le` |

- Decimal format
  - 31.41592
- e-format
  - 3.141592e+01

- Others

| Types | Conversion character |
|---|---|
| `char` | `%c` |
| `string` | `%s` |
| `pointer` | `%p` |

# Formating Rule (4)

- **Pitfalls**
  - Be careful of specifying conversion character

```
#include <stdio.h>

int main(void)
{
   short a = 0;
   short b = 0;
   scanf("%d", &a);
   printf("%hd %hd\n", a, b);
   return 0;
}
```

```
$ ./a.out
65537
1 1
```

```
#include <stdio.h>

int main(void)
{
   long long l = 0xF00000000000FFFF;
   /* -1152921504606781441 */
   printf("%d\n", l);
   return 0;
}
```

```
$ ./a.out
65536
```

# Example: Format Rule in printf()

| Declarations and initializations | | | |
|---|---|---|---|
| `char c = 'A', s[] = "Blue moon!";` | | | |
| **Format** | **Corresponding argument** | **How it is printed** | **Remarks** |
| `%c` | `c` | `"A"` | Field width 1 by default |
| `%2c` | `c` | `" A"` | Field width 2, right adjusted |
| `%-3c` | `c` | `"A  "` | Field width 3, left adjusted |
| `%s` | `s` | `"Blue moon!"` | Field width 10 by default |
| `%3s` | `s` | `"Blue moon!"` | More space needed |
| `%.6s` | `s` | `"Blue m"` | Precision 6 |
| `%-11.8s` | `s` | `"Blue moo   "` | Field width 11, Precision 8, left adjusted |

# Example: Format Rule in printf()

**Declarations and initializations**

```
int    i = 123;
double x = 0.123456789;
```

| Format | Corresponding argument | How it is printed | Remarks |
|---|---|---|---|
| %d | i | "123" | Field width 3 by default |
| %05d | i | "00123" | Padded with zeros |
| %7o | i | "    173" | Right adjusted, octal |
| %-9x | i | "7b       " | Left adjusted, hexadecimal |
| %-#9x | i | "0x7b     " | Left adjusted, hexadecimal |
| %10.5f | x | "   0.12346" | Field width 10, precision 5 |
| %-12.5e | x | "1.23457e-01 " | Field width 12, Left adjusted, precision 5, e-format |

# sscanf() and sprintf()

- int sprintf(char *s, const char* format, …);
- int sscanf(const char *s, const char* format, …);

```
char str1[] = "1 2 3 go", str2[100], tmp[100];
int a, b, c;

sscanf(str1, "%d%d%d%s", &a, &b, &c, tmp);
sprintf(str2, "%s %s %d %d %d\n", tmp, tmp, a, b, c);
printf("%s", str2);
```

```
go go 1 2 3
```

# fopen() and fclose()

- **A file should be opened before being used**
  - Why?
- **After used, it is better to be closed**
  - To flush the buffer (fflush)
- **BTW, what is a file?**
  - A sequence of bytes (characters)
  - These bytes can be accesses sequentially/randomly

# fopen() and fclose()

- **`FILE *fopen(const char *filename, const char *mode);`**
  - Performs housekeeping to use a file
    - Access right
    - Availability
    - Data structures for a file
  - Returns
    - A file pointer on success
    - NULL when fails
  - Mode is either "r" or "w" or "a"
    - For "w" or "a", a new file is created if it doesn't exist
    - "r+" for open a text file for read/write
    - "rb" to read a binary file, "wb" to write a binary file
- **`int fclose(FILE *fp);`**
  - Close a file
  - Return 0 on success

# Standard Files

- **The system opens the three standard files**
  - `stdin,  stdout,   stderr`

- **printf/scanf functions work with stdout/stdin**
  - Screen/keyboard in most cases

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

```
scanf(…) == fscanf(stdin, …)
printf(…) == fprintf(stdout, …)
```

# fgetc() and fputc()

- **int fgetc(FILE *fp);**
- **int getc(FILE *fp);**
  - Read a character from an opened file
- **int fputc(int c, FILE *fp);**
- **int putc(int c, FILE *fp);**
  - Write a character to an opened file

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

int main(int argc, char** argv)
{
   int c;
   FILE *fp, *tmp_fp;
   if (argc != 2 ) {
      fprintf(stderr, "\n%s%s%s\n\n%s\n\n",
      "Usage:   ", argv[0], "   filename",
      "This file will be doubled and some
      letters capitalized.");
      exit(1);
   }
   fp = fopen(argv[1], "r+");
   tmp_fp = tmpfile();
   while ((c = getc(fp)) != EOF)
      putc(toupper(c), tmp_fp);
   rewind(tmp_fp);
   fprintf(fp, "---\n");
   while((c = getc(tmp_fp)) != EOF)
      putc(c, fp);
   return 0;
}
```

tmpfile() crease a temporary file that will be deleted when it is closed or when the program exits

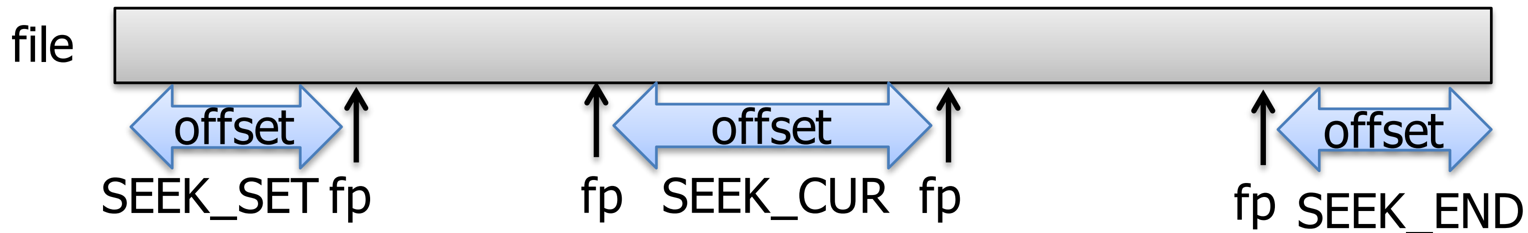
toupper() capitalizes a character

exit() exits the program immediately

rewind() rewinds a file

# Random Access

- `long ftell(FILE* fp)`
  - Returns the position **fp** points to in the file
- `int fseek(FILE* fp, long offset, int place)`
  - Initially, **fp** points the beginning of the file
  - The **fp** points the next byte to be accesses
  - fseek() sets the value of **fp**
  - **place**
    - SEEK_SET, SEEK_END, SEEK_CUR



```
/* Write a file backwards. */

#include <stdio.h>

#define MAXSTRING 100

int main(void)
{
   char fname[MAXSTRING];
   int c;
   FILE *ifp;
   fprintf(stderr, "\nInput a filename:  ");
   scanf("%s", fname);
   ifp = fopen(fname, "rb");       /* binary mode for MS_DOS */
   fseek(ifp, 0, SEEK_END);        /* move to end of the file */
   fseek(ifp, -1, SEEK_CUR);       /* back up one character */
   while (ftell(ifp) > 0 ) {
      c = getc(ifp);               /* move ahead one character */
      putchar(c);
      fseek(ifp, -2, SEEK_CUR);    /* back up two characters */
   }
   return 0;
}
```

# fgets() and fputs()

- `char *fgets(char *s, int n, FILE *fp);`
  - Read at most n-1 characters from the file associated with **fp**, and stores them into **s**
    - If fgets() meets a newline or EOF, no more characters are read
  - Returns
    - **s** when successful
    - NULL, otherwise
- `int fputs(const char *s, FILE *fp);`
  - Copies the string **s** into the file associated with **fp**, except for the NULL itself
    - c.f., puts() appends automatically appends a newline.
  - Returns
    - Nonnegative value when success
    - EOF, otherwise

# fgets() vs. gets()

- **`char *gets(char *s);`**
  - Read a line from stdin
  - Stops after an EOF or a newline
  - Dangerous it continues to store characters past the end of **s**
- **`char *fgets(char *s, int size, FILE *stream);`**
  - Read at most (size-1) characters
  - Stops after an EOF or a newline
  - Terminating null ('₩0') is stored

```
char buffer[1024];
gets(buffer);
fgets(buffer, 1024, stdin);
```

# fread() and fwrite()

- **Binary stream input/output**

- **`size_t fread(void *ptr, size_t size, size_t items, FILE *fp);`**
  - Reads at most **size*items** bytes from the file associated with **fp** and stores into the array at **ptr**

- **`size_t fwrite(const void *ptr, size_t size, size_t items, FILE *fp);`**
  - Reads **size*items** bytes from the array at **ptr** and writes them to the file associated with **fp**

# Example: fread() and fwrite()

```
#include <stdio.h>
struct card { int pips; char suit; };
int main(void)
{
    struct card deck[] = { {2,'s'}, {7,'d'}, {8,'c'} };
    FILE *fp = fopen( "unformatted", "wb" );
    fwrite(deck, sizeof(struct card), 3, fp);
    fclose(fp);
}
```

```
#include <stdio.h>
struct card { int pips; char suit; };
int main(void)
{
    struct card deck[10];
    FILE *fp = fopen( "unformated", "rb" );
    fread(deck, sizeof(struct card), 10, fp);
    fclose(fp);
    printf( "pips = %d, suit = %c\n", deck[6].pips, deck[7].suit );
}
```

# Executing Commands

```
int  system(const  chat *s);
```

- **system("date");  /* "date" is a command */**
  - Legal set of commands differ system to system

# Timing

- **There is a very accurate clock inside a computer**
- **<time.h> file defines clock_t and time_t**
- **`clock_t clock(void);`**
  - The time used by this program
- **`time_t time(time_t *p);`**
  - Seconds elapsed since 1/1/1970

# C Function Manuals

- **In Linux**
  - $ man printf

```
GETS(3)                    Linux Programmer's Manual                   GETS(3)

NAME
       fgetc, fgets, getc, getchar, gets, ungetc - input of characters and strings

SYNOPSIS
       #include <stdio.h>

BUGS
    Never use gets().  Because it is impossible to tell without knowing the data  in
    advance  how  many characters gets() will read, and because gets() will continue
    to store characters past the end of the buffer, it  is  extremely  dangerous  to
    use.  It has been used to break computer security.  Use fgets() instead.

       char *gets(char *s);

       int ungetc(int c, FILE *stream);

DESCRIPTION
       fgetc()  reads the next character from stream and returns it as an unsigned char
       cast to an int, or EOF on end of file or error.

       getc() is equivalent to fgetc() except that it may be  implemented  as  a  macro
       which evaluates stream more than once.

       getchar() is equivalent to getc(stdin).

       gets()  reads  a  line from stdin into the buffer pointed to by s until either a
       terminating newline or EOF, which it replaces with a null byte ('\0').  No check
```

- **How to use man**
  - $ man man
- **Other ways to access C manuals**
  - Google it
  - Books
  - MSDN (microsoft developers network)