

# Lecture09: File I/O

## 5/6/2013

Slides modified from Yin Lou, Cornell CS2022: Introduction to C

1

## File I/O

- Before a file can be read or written, it has to be opened by the library function `fopen()`. `fopen` takes a file name like `"data.txt"`, does some housekeeping and negotiation with the operating system, and returns a pointer to be used in subsequent reads or writes of the file.
- This pointer points to a type `FILE`.

```
FILE *fp;
```

## Text vs. Binary Files

- Text files are divided into lines. Each byte in a text file represents a character, therefore readable by humans
- In binary files, each byte may not be a character.

## File operations

- Open a file: `fopen()`
- Close a file: `fclose()`
- Read a binary file: `fgetc()`, `getc()`, `fgets()`, `puts()`, `fscanf()`, `fread()`
- Write a binary file: `fputc()`, `putc()`, `fputs()`, `puts()`, `fprintf()`, `fwrite()`
- File positioning: `fseek()`, `ftell()`, `fgetpos()`, `fsetpos()`

## fopen()

```
FILE *fopen(char *name, char *mode);  
void fclose(FILE* stream)
```

- fopen returns a pointer to a FILE.
- mode can be
  - “r” - read
  - “w” - write
  - “a” - append
  - “b” can be appended to the mode string to work with binary files. For example, “rb” means reading binary file.

## Text Files – fprintf()

```
int fprintf(FILE *fp, char *format, ...);
```

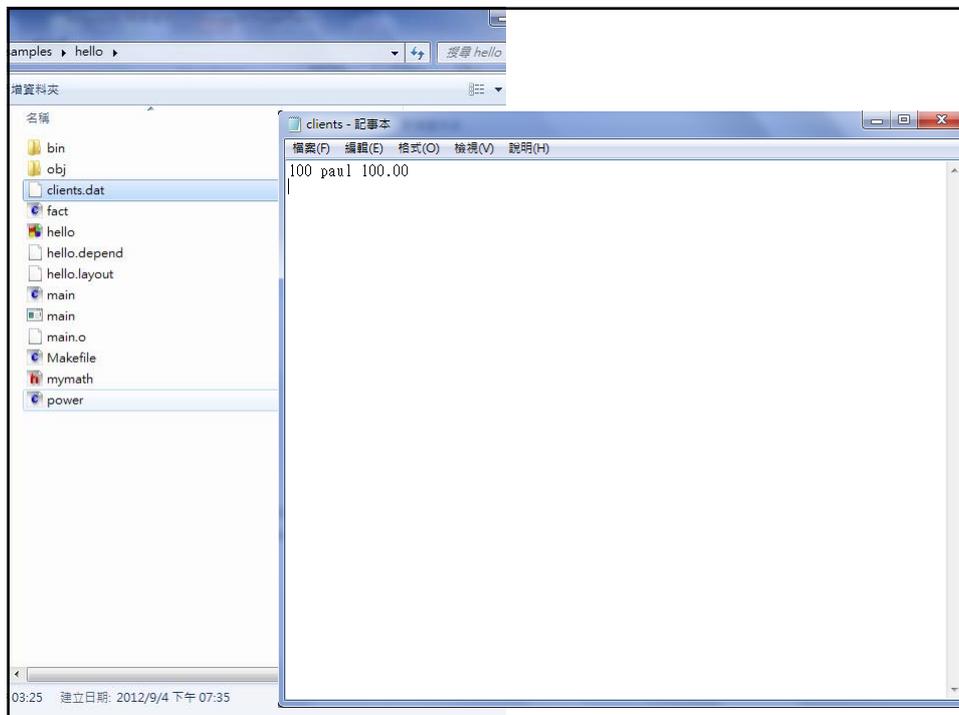
Similar to printf().

On success, the total number of characters written is returned.

If a writing error occurs, a negative number is returned.

## Example – create and write to a file

```
#include <stdio.h>
int main()
{
    FILE *fp;
    char name[10];
    double balance;
    int account;
    if ((fp = fopen("clients.dat", "w")) == NULL) {
        printf("File could not be opened\n");
    }
    else {
        printf("Enter one account, name, and balance.\n");
        scanf("%d%s%lf", &account, name, &balance);
        fprintf(fp, "%d %s %.2f\n", account, name, balance);
        fclose(fp);
    }
    return 0;
}
```



## Text Files – fscanf()

```
int fscanf(FILE *fp, char *format, ...);
```

- Similar to scanf.
- On success, return the number of items successfully filled. This count can match the expected number of items or be less (even zero) due to a matching failure, a reading error, or the reach of the *end-of-file*. And, if *end-of-file* happens before any data could be successfully read, [EOF](#) is returned.

```
int feof(FILE *fp);
```

- return 1/0 end-of-file is reached

## Text Files – read from a file

```
#include <stdio.h>
int main()
{
    FILE *fp;
    char name[10];
    double balance;
    int account;
    if ((fp = fopen("clients.dat", "r")) == NULL) {
        printf("File could not be opened\n");
    }
    else {
        fscanf(fp, "%d%s%lf", &account, name, &balance);
        printf("%d %s %.2f\n", account, name, balance);
        fclose(fp);
    }
    return 0;
}
```

## Text Files – character I/O

```
int fputc(int c, FILE *fp);
int putc(int c, FILE *fp);
```

Write a character *c* to a file. `putc()` is often implemented as a MACRO (hence faster). On success, the character written is returned. If a writing error occurs, [EOF](#) is returned

```
int fgetc(FILE *fp);
int getc(FILE *fp);
```

Read a character *c* to a file. `getc()` is often implemented as a MACRO (hence faster). On success, the character read is returned. If a read error occurs, [EOF](#) is returned.

## Text Files – character I/O

```
#include <stdio.h>
int main()
{
    FILE *source_fp, *dest_fp;
    int ch;

    if ((source_fp = fopen("source.txt", "r")) == NULL)
        printf("cannot open source file\n");
    if ((dest_fp = fopen("dest.txt", "w")) == NULL)
        printf("cannot open dest file\n");
    while ((ch = getc(source_fp)) != EOF)
        putc(ch, dest_fp);

    fclose(source_fp);
    fclose(dest_fp);
    return 0;
}
```

## Text Files – character I/O

```
int ungetc(int c, FILE *fp);
```

Push back a character read from a file pointer.

```
int feof(FILE *fp);
```

return 1/0 *end-of-file* is reached

## Text Files – character I/O

```
#include <stdio.h>
int main ()
{
    FILE * fp;
    int c;
    fp = fopen ("source.txt","r");
    if (fp == NULL) {
        printf("cannot open the file\n");
        return 0;
    }
    while (!feof (fp)) {
        c = getc(fp);
        if (c == '#')
            ungetc('@', fp);
        else
            putchar(c, stdout); // stdout is the screen
    }
    return 0;
}
```

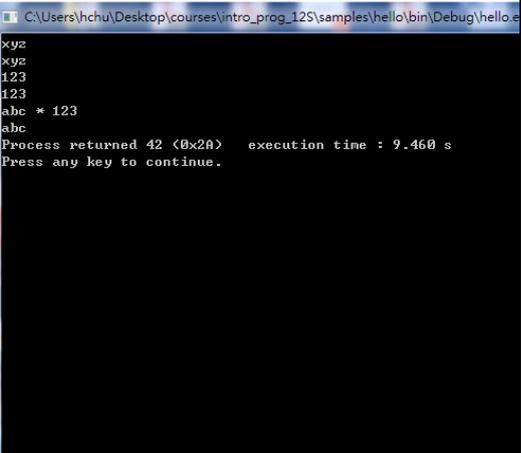
## Text Files – standard input & output

- `FILE *stdin` // screen input as a file
- `FILE *stdout` // screen output as a file

## Text Files – stdin, stdout

```
#include <stdio.h>

int main ()
{
    int c;
    while ((c =
fgetc(stdin)) != '*')
    {
        fputc(c, stdout);
    }
}
```



```
C:\Users\hchu\Desktop\courses\intro_prog_12S\samples\hello\bin\Debug\hello.e
xyz
xyz
123
123
abc * 123
abc
Process returned 42 (0x2A)   execution time : 9.460 s
Press any key to continue.
```

## In-Class Exercise 9.1

Write a program that converts all letters in a file, called "source.txt" to upper case. (Characters other than letters shouldn't be changed) The program should obtain the file name from the command line and write its output to stdout.

## Text Files – Line I/O

```
int fputs(const char *s, FILE *fp);
```

Write a line of characters to a file. On success, a non-negative value is returned. On error, the function returns [EOF](#).

```
char* fgets(char *s, int n, File *fp);
```

Read characters from a file until it reaches the first new-line or (n-1) characters, in which it places the NULL character ('\0') at the end of the string.

On success, the function returns s. If the end-of-file is encountered before any characters could be read, the pointer returned is a NULL pointer (and the contents of s remain unchanged).

## Text Files – Line I/O

```
#include <stdio.h>
int main()
{
    FILE *source_fp, *dest_fp;
    char s[100];

    if ((source_fp = fopen("source.txt", "r")) == NULL)
        printf("cannot open source file\n");
    if ((dest_fp = fopen("dest.txt", "w")) == NULL)
        printf("cannot open dest file\n");

    while (fgets(s, 100, source_fp) != NULL)
        fputs(s, dest_fp);

    fclose(source_fp);
    fclose(dest_fp);
    return 0;
}
```

## In-Class Exercise 9.2

Write a program that reads a text file “source.txt”, counts the number of characters, the number of words, and the number of lines in this text file, and outputs these numbers on the screen.

## Binary Files – fread() and fwrite()

```
size_t fread(void *array, size_t size, size_t count,  
FILE *fp);
```

```
size_t fwrite(void *array, size_t size, size_t  
count, FILE *fp);
```

- array must be a pointer
- size - size of elements in this array
- count - number of elements in this array
- Return the total number of elements successfully read or written.

## Binary Files: read from a file

```
#include <stdio.h>  
int main()  
{  
    FILE *fp;  
    float value[3];  
    fp = fopen("account.txt", "rb");  
    fread(value, sizeof(float), 3, fp);  
    printf("%f\t%f\t%f\n", value[0], value[1], value[2]);  
    fclose(fp);  
    return 0;  
}
```

## Binary Files: write to a file

```
#include <stdio.h>
int main()
{
    FILE *fp;
    float value[3] = {1.0, 2.0, 3.0};
    fp = fopen("account.txt", "wb");
    fwrite(value, sizeof(float), 3, fp);
    fclose(fp);
    return 0;
}
```

## File positioning

Every opened file has a **file position** for the next read or write. When a file is first opened, the file position is set at the beginning of a file. When any read or write is performed, the file position moves automatically.

```
long int ftell(FILE *fp);
```

Return the current file position (byte offset from the start of a file) as a long integer.

```
int fseek(FILE *fp, long int offset, int whence);
```

Change the current file position to offset relative to whence.

## File positioning

```
int fseek(FILE *fp, long int offset, int whence);
```

Change the current file position to offset relative to whence.

Whence can be

SEEK_SET	Beginning of file
SEEK_CUR	Current file position
SEEK_END	End of file

## File positioning – fseek & ftell

```
#include <stdio.h>

int main ()
{
    FILE *fp;
    long int size;
    fp = fopen ("example.txt", "r");
    if (fp == NULL)
        printf("Error opening the file\n");

    fseek(fp, 0, SEEK_END);
    size = ftell(fp);
    fclose(fp);
    printf("Size of example.txt: %ld bytes.\n", size);
    return 0;
}
```

## In-Class Exercise 9.3

```
#include <stdio.h>
int main()
{
    FILE* fp;
    fp = fopen("example.txt",
              "w");
    fputs("This is an apple.",
          fp);
    fseek(fp, ? , SEEK_SET);
    fputs(" sam", fp);
    fclose(fp);
    return 0;
}
```

Modify the left program such that the file content will be changed to "This is a sample."

Change ? with a correct offset number.