四、標準I/O函式庫 (Standard I/O Library)

標準函式庫

- 為ANSI C標準，在<stdio.h>中宣告
- 為library functions而不是system calls，system calls是直接存取低階的linux核心所提供的服務，而library functions為較高階的程式介面
- 大部分的Library functions為user space，而system calls為kernel space
- Library functions提供更強的功能，例如FILE pointer所提供的功能比file descriptor還要多，它的好處如下：
  - Buffering (網路I/O要特別注意緩衝區的影響)
  - Formatted I/O (像printf，scanf等等)
  - 支援line-oriented functions
  - 移值性(portability)較好

Streams與FILE結構

- FILE pointer (FILE *) vs file descriptor
  - standard I/O 的操作是使用FILE pointer而不是使用file descriptor
- FILE結構包含所有standard I/O操作所需的資訊，如下：
  - 底層所用的file descriptor
  - 指向緩衝區(user space)的指標
  - 一個計數器，記錄有多少位元組在緩衝區中
  - 包含錯誤旗標(error bits)
  - EOF資訊
- 程式通常不會直接存取FILE結構，而是把指向FILE結構的指標(FILE *)當作參數讓各種函式來處理

標準輸入，輸出與錯誤

```
#include <stdio.h>

    FILE *stdin;
    FILE *stdout;
    FILE *stderr;
```

- 標準輸入 stdin
  - 其底層的file descriptor為STDIN_FILENO
- 標準輸出 stdout
  - 其底層的file descriptor為STDOUT_FILENO
- 標準錯誤 stderr
  - 其底層的file descriptor為STDERR_FILENO

緩衝(Buffering) (1/2)

- 緩衝的主要目的為增進效率，也就是減少read或write的系統呼叫次數。標準函式庫會自動嘗試將所有開啟的stream做緩衝(也就是會自動配置記憶體，用的人不需自己花時間處理這部分)
- 有三種緩衝方式可以選擇
  - 不做緩衝 (unbuffered)
    - 每次寫入就直接對應到write的系統呼叫，像stderr就屬於這類
  - 以整行做緩衝 (line buffered)
    - 遇到換行或緩衝區滿了才做flush
  - 全部緩衝 (fully buffered)
    - 緩衝區滿了才做flush

緩衝(Buffering) (2/2)

```
#include <stdio.h>

    void setbuf(FILE *stream, char *buf);
    void setbuffer(FILE *stream, char *buf, size_tsize);
    void setlinebuf(FILE *stream);
    int setvbuf(FILE *stream, char *buf, int mode , size_t size);
    int fflush(FILE *stream);
```

- 前四個函式只能在緩衝區為空的情況下才能使用
- 可以用setvbuf函式來改變緩衝方式，其mode可以為_IONBF，_IOLBF，_IOFBF三種，分別代表不做緩衝，以行做緩衝，全部做緩衝
- 前三個函式只是setvbuf的別名，其功能相同，由上而下分別與下列呼叫相同
  - setvbuf(stream, buf, buf ? _IOFBF : _IONBF, BUFSIZ);
  - setvbuf(stream, buf, buf ? _IOFBF : _IONBF, size_tsize);
  - setvbuf(stream, (char *)NULL, _IOLBF, 0);
- fflush函式將user space的緩衝區資料透過系統呼叫write函式寫出，如果參數為NULL，其對象為所有開啟的streams

開啟與關閉檔案

```
#include <stdio.h>

    FILE *fopen(const char *path, const char *mode);
    FILE *fdopen(int fildes, const char *mode);
    FILE *freopen(const char *path, const char *mode, FILE *stream);
    int fclose(FILE *stream);
```

- 前三者用來開啟檔案，fopen的參數為檔名。fdopen用來將file descriptor包裝成FILE pointer，以享受標準I/O函式庫的好處(例如formatted I/O)。freopen跟dup2的功用相同，它會將參數中的stream先關閉，然後再開啟檔案，關閉的檔案與新開啟的檔案會使用相同的file descriptor
- fclose函式用來關閉已開啟的檔案，緩衝區的資料會被flush，緩衝區佔據的記憶體也會被釋放

開啟檔案的模式

模式	可讀	可寫	檔案讀寫指標	清除內容	建新檔
r	Yes	No	檔案開頭	No	No
r+	Yes	Yes	檔案開頭	No	No
w	No	Yes	檔案開頭	Yes	Yes
w+	Yes	Yes	檔案開頭	Yes	Yes
a	No	Yes	檔案結尾	No	Yes
a+	Yes	Yes	檔案結尾	No	Yes

所有模式的後面可以加b，代表開啟為binary，例如r+可以變成r+b或是rb+

此外所建新檔的權限為0660

讀寫檔案

- 在unformatted I/O下，讀寫檔案的方式可以分成三類
  - 一次讀(寫)一個字元(character-at-a-time)
  - 一次讀(寫)一整行(line-at-a-time)
  - 直接讀寫binary資料(direct I/O)

一次讀(寫)一個字元(1/2)

CODE. 4-1

```
#include <stdio.h>

    int fgetc(FILE *stream);
    int getc(FILE *stream);
    int getchar(void);
    int ungetc(int c, FILE *stream);
```

- 前三個函式用來讀取一個字元，fgetc與getc功能相同，只是getc實作上為巨集定義，而getchar與getc(stdin)相同
- getc的效率比fgetc還好，但是因為是巨集，所以有些限制
  - 不能作為函式指標，也就是getc不能當參數傳給其它函式
  - getc的參數不能是有副作用(side effect)的參數 (例如getc(*f++);)
- 前三個函式的傳回值皆為unsigned char轉換成int而成的，之所以要是int是因為這樣才能包含所有的回傳值(例如要包含EOF，其值為-1)
- ungetc函式可以將一個字元放回緩衝區中，下次的讀取就會讀到由ungetc放進去的部分

一次讀(寫)一個字元(2/2)

```
#include <stdio.h>

    int fputc(int c, FILE *stream);
    int putc(int c, FILE *stream);
    int putchar(int c);
```

- 此三個函式用來寫入一個字元到stream中，其與fgetc，getc與getchar相似。其中putchar(c)與putc(c, stdout)相同

一次讀(寫)一整行(1/2)

```
#include <stdio.h>

    char *fgets(char *s, int size, FILE *stream);
    char *gets(char *s);
```

- fgets函式一次從stream讀取一整行的資料(也就是讀到換行字元為止)放到s所指向的空間，最多讀取size-1個位元組(因為要留一個byte做NULL terminated)
- gets函式與fgets相似，只是它是讀取stdin的資料，另外fgets會保留換行字元，而gets不保留。由於gets無法指定讀取長度上限，因此容易造成buffer overflow，因此不建議使用。如果要讀取stdin一整行，可以用fgets(s,size,stdin)

一次讀(寫)一整行(2/2)

```
#include <stdio.h>

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

- 這兩個函式用來寫入一個字串(也就是一直寫到遇到NULL為止)的資料到stream中，其中fputs寫入的字串其結尾不包含NULL，而puts寫入的字串除了不包含NULL結尾外，還會自動送出一個換行字元
- puts跟gets一樣不安全，所以最好用fputs取代，只是要自己處理換行字元的問題

直接讀寫binary資料

```
#include <stdio.h>

    size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
    size_t  fwrite(const  void  *ptr,  size_t  size,  size_t  nmemb,  FILE *stream);
```

- fread函式來用讀取stream，它會讀取stream所指向的檔案內容，讀到的內容會放置在ptr指標所指的位置上，size為讀取的單位大小，nmemb為讀取的單位個數。特別注意，其傳回值為確實讀取的單位個數，而不是位元組(bytes)
- fwrite函式用來寫入stream，其參數意義與fread相同，除了ptr指標所指的為將被寫入的資料
- 傳回值<=nmemb
- 注意portability的問題(同樣的資料結構在不同的compiler下可能會有不同的結果)

取得stream狀態

```
#include <stdio.h>

    void clearerr(FILE *stream);
    int feof(FILE *stream);
    int ferror(FILE *stream);
    int fileno(FILE *stream);
```

- clearerr函式用來清除記錄在FILE結構中的錯誤記錄與EOF(清除error bits)
- feof函式用來判斷stream是否已經讀到結尾，如果是則回傳非零值(也就是true)
- ferror函式用來取得記錄在FILE結構中的錯誤代碼，如果有錯誤會回傳非零值
- fileno函式來用傳回該stream所使用的底層file descriptor (例如需要使用fstat函式取得檔案狀態時)，不建議與FILE pointer混合使用，否則容易出錯

移動stream讀寫位置

```
#include <stdio.h>

    int fseek(FILE *stream, long offset, int whence);
    long ftell(FILE *stream);
    void rewind(FILE *stream);
    int fgetpos(FILE *stream, fpos_t *pos);
    int fsetpos(FILE *stream, fpos_t *pos);
```

- fseek函式用來設定stream讀寫位置，offset為相對位置，whence有三種值，SEEK_SET，SEEK_CUR，SEEK_END，分別代表從檔案開頭，從目前位置，從檔案結尾算起。fseek函式會自動清除EOF狀態，在緩衝區的資料也會自動清除
- ftell函式用來取得目前stream讀寫位置
- rewind函式將stream讀寫指標移到開頭處，等同於fseek(stream, 0L, SEEK_SET)，除了rewind不會清除EOF狀態之外
- fgetpos與fsetpost分別為ftell與fseek的變形，因為在有些作業系統中，不是所有的檔案都是以bytes為單位，此時fpos_t可能為較複雜的資料結構

使用暫存檔

```
#include <stdio.h>

    FILE *tmpfile(void);
    char *tmpnam(char *s);
```

- tmpfile函式開啟一個唯一檔名的暫存檔，其開啟模式為w+b，暫存檔案在關閉後(或行程結束)會自動刪除。函式的傳回值為指向該檔的FILE pointer
- tmpname函式傳回一個目前不存在的檔名，然後程式設計師必須自行開啟。特別注意，如果參數s為NULL，其傳回值就指向tmpnam函式的內部static變數
- 這兩個函式無法指定產生路徑，預設路徑是在/tmp，由stdio.h中的P_tmpdir所定義

格式化I/O (Formatted I/O) (1/2)

```
#include <stdio.h>

    int printf(const char *format, ...);
    int fprintf(FILE *stream, const char *format, ...);
    int sprintf(char *str, const char *format, ...);
    int snprintf(char *str, size_t size, const char *format, ...);
```

- 這些函式用來格式化輸出
- printf函式用來格式化輸出到stdin，而fprintf可以指定輸出到哪個stream，其輸出的格式皆由format參數控制(man 3 printf)
- sprintf函式是將格式化好的資料放到str所指放的空間，其結尾會自動將上NULL，使用上必須注意是否會buffer overflow
- snprintf除了可以指定最大長度之外，與sprintf相似

格式化I/O (Formatted I/O) (2/2)

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

- 這些函式用來格式化輸入
- 特別注意，後面的參數必須為指標，例如
  int var;
  scanf("%d",&var)