บทที่ 9
การสร้างไฟล์
(disk file operation)

9.1 หลักการทำงาน

ก่อนที่จะศึกษาเรื่องการเปิดไฟล์ ปิดไฟล์ สร้างไฟล์ และจัดการอย่างอื่นกับไฟล์ ขอให้ผู้อ่านทำความรู้จักกับหลักการทั่ว ๆ ไปอันเป็นหลักการทำงานของไฟล์ และการสื่อสารกับโปรแกรมควบคุมระบบสักเล็กน้อยเสียก่อน

```
typedef struct-buffer {
      int-fd ;     /* file descriptor */
      int-cleft , /* character left in buffer */
      int-mode ;  /* how you will work with file */
```

```
    char * -nextc ; /* location of next charachter */
    char *-buff ; /* location of buffer */
} FILE
extern FILE - efile [-MAXFILE]
```

เป็นการกำหนดให้ -efile [ ] เป็นอะเรย์ของโครงสร้างแบบ FILE [1] (ขอให้สังเกตเครื่องหมายขีด (-) ซึ่งเรายอมให้เป็นอักขระเริ่มของชื่อตัวแปรได้) จึงเห็นได้ว่า FILE เป็นตัวแปรโครงสร้างมี tag type ชื่อ -buffer และประกอบไปด้วยตัวแปรภายใน 5 ตัวคือ

-fd เป็น int -cleftเป็น int -modeเป็น int *-nextc เป็น pointer ชี้ไปที่ char และ *-buff เป็น pointer ชี้ไปที่ char โครงสร้างใด ๆ ที่มีลักษณะนี้เราใช้ชื่อร่วมว่า FILE ตามตัวอย่างเรากำหนดให้ FILE ประกอบด้วยไฟล์แบบ (typedef) เดียวกันรวมทั้งสิ้น -MAXFILEไฟล์คือ -efile[0], -efile[1], ..., -efile [MAXFILE] หากกำหนดให้ -MAXFILE เท่ากับ 10 ก็แสดงว่า FILE ประกอบไปด้วยไฟล์แบบเดียวกัน (เรียกว่าอะเรย์ของไฟล์) รวม 10 ไฟล์ คือ - efile[0], -efile[1], ..., -efile[9] ที่พร้อมที่จะให้เราเปิดใช้

---

[1] ดูเรื่อง typedef command ในบทที่ 8 ตัวอย่างที่อาจทำให้มองเห็นที่ใช้และแนวการใช้ typedef คือ

```
typedef int SIZE ;
SIZE ret, num ;
```

เป็นการกำหนดให้ตัวแปร ret และ num มีข้อมูล (data type) แบบเดียวกันกับ SIZE คือเป็น int ทั้งคู่ ในที่นี้ SIZE เป็นชื่อร่วมที่ใครต้องการร่วมใช้ก็ได้แต่เมื่อใช้แล้วต้องเป็นตัวแปรแบบ int ทั้งหมด

การเปิดไฟล์ (กรณี buffered file) มีรูปแบบดังนี้ (กรณีสั่งเปิดใช้ไฟล์เดียว)

```
FILE * f1 * fopen ( ) ;

f1 = fopen (filename, mode) ;
```

เป็นการกำหนดให้ f1 เป็น pointer(เรียก f1 ว่า file pointer)ชี้ไปที่โครงสร้างแบบ FILE ส่วน f1 = fopen (filename, mode) เป็นการกำหนดเป้าให้ f1 มิให้ชี้ไปที่กองขยะหรือชี้ไปที่ ๆ เราไม่ต้องการตามแบบข้างบน f1 = fopen (filename, mode) ก็คือกำหนดให้ f1 ชี้ไปที่ไฟล์ชื่อ filename ที่เราต้องการใช้งานตาม mode ที่ต้องการ mode ประกอบด้วย "w" คือ write "r" คือ read และ "a" คือ append เช่น

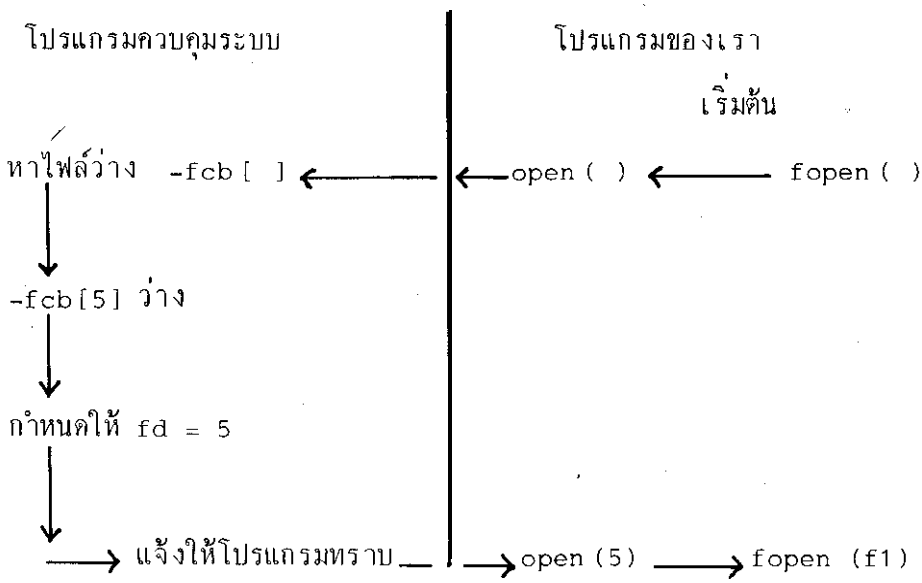
```
'FILE * fl * fopen ( );

if ((fl=fopen (fname, "w")) = = NULL)
   printf ("I can't create % s n", fname) ;
   exit (1) ;
}
```

เป็นการกำหนดให้ f1 ชี้ไปที่สมาชิกของอะเรย์ - efile [ ] ว่ามีไฟล์ใดยังไม่ได้ใช้บ้าง โดย fopen ( ) จะทำหน้าที่ตรวจสอบหาไฟล์ที่ยังไม่ได้ใช้ดังกล่าว หากพบว่ายังมีไฟล์ใดว่างอยู่ fopen ( ) จะชี้ไปที่ไฟล์นั้น แล้ว f1 จะมี rvalue ที่ชี้ไปที่ไฟล์นั้นพร้อมทั้งเก็บข้อมูลและสถานะของไฟล์นั้นเอาไว้ตามตัวอย่างข้างบนถ้า fopen ( ) ไม่พบไฟล์ใดว่างเลยก็จะแจ้งมาว่าสร้างไฟล์ชื่อ fname ไม่ได้

การทำงานของ disk file operation นั้นจะทำงานประสานกันระหว่างโปรแกรมของเรากันโปรแกรมควบคุมระบบ (operating system, OS) โดยใน OS จะมีอะเรย์สำหรับเก็บ file descriptor (fd,ดูโครงสร้างตามแบบ FILE ในตอนต้น) สมมุติอะเรย์ดังกล่าวคือ -fcb[ ] ซึ่งมีขนาดเท่ากับ -MAXFILE fd จึงเป็นตัวกลางทำหน้าที่สื่อสารระหว่างโปรแกรมกับ OS

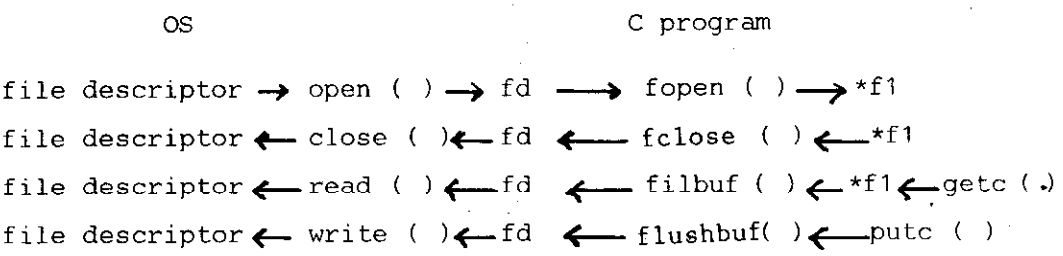
สมมุติเราต้องการเปิดไฟล์ด้วยฟังก์ชัน fopen ( ) และสมมุติว่า -MAXFILE มีค่าได้สูงสุดเท่ากับ 16 ไฟล์เมื่อเราสั่งเปิดไฟล์ โปรแกรมกับ OS จะประสานงานกันดังนี้(ดูไดอะแกรม)



หมายความว่า เมื่อเราเรียก fopen ( ) ฟังก์ชัน fopen ( ) จะเรียกฟังก์ชัน open ( ) ให้ติดต่อกับ OS เพื่อดูว่า file descripter (fd) ใดว่างอยู่บ้าง (ใน CP/M จะเป็นการดูว่า file control block ใดว่างอยู่) OS จะค่อย ๆ ตรวจดูที่อะเรย์ -fcb [ ] (fcb คือ flie control block) ว่ามีสมาชิกใดว่างอยู่เมื่อพบก็จองที่นั้นไว้ให้แก่ไฟล์ที่

เราประสงค์จะเปิดแล้วแจ้งค่า fd ส่งคืนให้ (แล้วใช้ fd ดังกล่าวเป็นตัวสื่อสารระหว่างโปรแกรมกับอะเรย์ -fcb[ ] ใน OS) ค่า fd ที่ส่งคืนมาให้ fopen ( ) นั้นจะถูกแทน (ใส่) ลงใน -fd ซึ่งเป็นสมาชิกของโครงสร้าง FILE แปลว่า fopen ( ) ส่ง file pointer คือ *f1 ไปยัง -efile[ ] ต่อทันทีที่ได้รับมาจาก OS ผ่าน open ( ) หรือนัยหนึ่ง fopen ( ) จะเป็นผู้บอก file pointer แก่เรา

ภาพแสดงการสื่อสารระหว่าง OS กับโปรแกรมและแบบการส่งต่อ fd ปรากฏดังนี้ ในที่นี้รวมการทำงานหลังจากเปิดไฟล์แล้วด้วยคือเมื่อเราต้องการปิดไฟล์ (fclose ( )) ต้องการขอรับข้อมูลที่เก็บไว้ในไฟล์ (getc ( )) และต้องการเก็บข้อมูลไว้ในไฟล์ (putc ( ))



ลองดูกรณี getc ( ) กล่าวคือ เมื่อเราต้องการขอข้อมูล (อักขระ) จากfile นั้น getc ( ) จะรับเอา fd เป็นอาร์กิวเมนต์แล้วตามไปตรวจดูที่สมาชิกของโครงสร้าง FILE ชื่อ -cleft ว่ายังมีอักขระใดปรากฏอยู่แล้วในบัฟเฟอร์ (buffer) ถ้ามีก็จะส่งค่ากลับ (return) ถ้าไม่มีก็จะเรียกฟังก์ชั่น filbuf ( ) ให้ไปหา OS เพื่อจัดการเติมอักขระลงในบัฟเฟอร์ (พร้อมทั้ง updete ตัวแปรชื่อ -cleft และ -nextc ในโครงสร้าง FILE ด้วย)

สำหรับการทำงานของ putc ( ) ก็เป็นไปในทำนองเดียวกัน

9.2 การเปิด/ปิดไฟล์

หลังจากเข้าใจการทำงานเรื่องไฟล์ดีแล้ว เรามาเริ่มศึกษาเกี่ยวกับการเปิด-ไฟล์กันดูบ้างหลังจากนั้นคือ เมื่อเข้าใจดีแล้วก็จะเป็นการสรุป I/O routine ที่มีใช้ในภาษา C (ต้องตรวจดูคู่มือคอมไพเลอร์ของเราเองว่ามี I/O routine ตามที่สรุปไว้ให้หรือไม่ด้วย หากไม่มีหรือมีแต่ไม่เหมือนกันให้ใช้ตามคู่มือ ถ้าไม่มีเลยคงจะเป็นหน้าที่เราจะต้องสร้างขึ้นมาใช้เองในรูป my.lib (หมายถึง my library)

ในการเปิดไฟล์นั้น OS จำเป็นต้องทราบข้อมูลบางประการเกี่ยวกับไฟล์เสียก่อน ข้อมูลเหล่านั้นคือ overhead information ซึ่งเราต้องกำหนดไว้ในไฟล์ แต่โดยมากจะเก็บไว้แล้วในไฟล์ชื่อ stdio.h ซึ่งเพียงแต่เราผนวกไฟล์นี้ไว้กับโปรแกรมของเราในรูป # include "stdio.h" ก็ถือว่าใช้ได้ ข้อมูลที่กล่าวถึงดังกล่าว จะเป็นโครง-สร้างดังนี้ (ถ้าจะทำงานกับไฟล์เราต้องมี preprocess or # include "stdid.h" เสมอ)

```
typedef struct -buffer {
    int - fd ;
    int - cleft ;
    int - mode ;
    char *- nextc ;
    char *- buff ;
} FILE ;
extern FILE - efile [-MAXFILE] ;
```

จะเห็นว่าเรากำหนดให้โครงสร้าง FILE เป็นโครงสร้าง (ตัวแปร) ภายนอกที่ทุกฟังก์ชั่นสามารถเรียกใช้ได้ โดยที่ -MAXFILE เป็น symbolic comtant ที่นิยามไว้ด้วย # define ในไฟล์ชื่อ stdio.h ใช้แสดงจำนวนไฟล์ที่เราสามารถสั่งเปิดใช้ได้พร้อม ๆ กัน

หากถามว่าเปิดได้กี่ไฟล์เรื่องนี้คงต้องพลิกดูคู่มือคอมไพเลอร์ (doeumentation) ของท่านเอง หากไม่แจ้งไว้ก็ให้เรียก (list) ไฟล์ stdio.h มาดูว่ากำหนด -MAXFILE ไว้เท่าไร หากพบว่า # define-MAXFILE 16 ก็แปลว่าเราสามารถเปิดไฟล์ใช้พร้อมกันได้ 16 ไฟล์ ดังนี้เป็นต้น

ลองเริ่มเปิดไฟล์ด้วย fopen ( ) กันเลยเพราะเราได้ศึกษาเรื่องนี้มาบ้างแล้วในตอน 9.1 ดังนี้ [1]

เมื่อสั่ง fopen ( ) ฟังก์ชั่น fopen ( ) จะจัดการดังนี้คือ (1) บรรจุข้อมูลที่จำเป็นลงในโครงสร้างชื่อ FILE ทั้งใน OSและในโปรแกรมของเราเพื่อให้ OS และโปรแกรมของเราสื่อสารกันได้ (สื่อสารผ่าน fd) (2) ส่งค่าคืนเป็น pionter f1 ชี้ไปยังโครงสร้างที่เก็บข้อมูลดังกล่าวเอาไว้

การกำหนดลักษณะให้กำหนดเป็น

```
FILE * f1 * fopen ( ) ;
```

โดยที่ FILE คือ โครงสร้างแบบ FILE ตามที่นิยามไว้ใน stdio.h ขณะที่ *f1 คือ pointer ชี้ไปที่โครงสร้างแบบ FILE นั้น *f1 จึงเป็นสิ่งที่ทำให้เราเข้าถึงไฟล์ที่ต้ององการได้ และหากเราต้องการเข้าถึงไฟล์มากกว่า 1 ไฟล์เช่นต้องการเปิด 2 ไฟล์พร้อมกันให้ใช้ pointer 2 ตัวคือ *f1 และ *f2 ดังนี้

---

[1] fopen ( ) ต้องเป็น pointer เพราะต้องชี้ไปที่โครงสร้างแบบ FILE

```
FILE * f1 * f2 * fopen ( )
```
[1]

หรือถ้าประสงค์จะเปิดไฟล์มากกว่านี้ก็กระทำได้ ขอแต่เพียงกำหนดจำนวน pointer ให้เท่ากับจำนวนไฟล์ที่ต้องการเปิดก็แล้วกัน

จากนั้นเราจะต้องแจ้งให้คอมไพเลอร์ทราบว่า (1) ต้องการให้คอมไพเลอร์เข้าถึงไฟล์ชื่อว่าอะไร (2) เราต้องการเปิดไฟล์เพื่อทำอะไร และ (3) จะตามหาข้อมูลอ้างอิงไฟล์นั้นได้จากที่ใด ซึ่งกระทำได้ด้วยคำสั่งต่อไปนี้

```
f1 = fopen (filename, mode) ;
```

---

[1] fopen ( ) และ fclose ( ) เป็นรูทีนที่ใช้สำหรับเปิดและปิด buffered file การอ้างถึง buffered file จำเป็นต้องอาศัย pointer เรียกว่า file pointer โดยที่ fopen ( ) และ fclose ( ) จะชี้ไปที่โครงสร้างแบบ FILE รูปแบบของ fopen ( ) และ fclose ( ) ปรากฏดังนี้

```
FILE * fopen (namefile, mode)
char * namefile ;
char * mode ;
```

เช่น `FILE * filepnt, * fopen ( ) ;`

```
filepnt = fopen ("NAME", "w");
```

การปิดไฟล์มีรูปแบบดังนี้

```
fclose (filepnt)
FILE * filepnt ;
```

filepnt คือ file pointer ที่ส่งมาจาก fopen ( )

Filename หมายถึงชื่อไฟล์ที่เราแจ้งไปและเก็บไว้บนดิสค์ mode หมายถึงวัตถุประสงค์ที่เราต้องการใช้ไฟล์นั้น เช่น "w", "r", "a" และ f1 คือ pointer ที่ชี้ไปยังโครงสร้าง FILE ณ สมาชิก (คือไฟล์) ที่ต้องการ การเปิดไฟล์เพื่อเขียน ("w") หรือเพิ่ม ("a") ข้อมูลนั้นเราไม่จำเป็นต้องเปิดไฟล์ไว้ล่วงหน้าเพราะฟังก์ชั่น fopen ( ) จะเปิดไฟล์ให้ใช้งานได้ทันที เรื่องนี้จึงเป็นสิ่งที่ควรระมัดระวังเพราะถ้าเราระบุชื่อไฟล์เพื่อ "w" ซ้ำกันกับชื่อที่เคยเปิดไว้แล้ว การสั่ง fopen ( ) ครั้งหลังนี้จะลบข้อมูลเดิมทิ้งไปทั้งหมด

การเปิดไฟล์นั้น คอมไพเลอร์จะแจ้ง error message ให้ทราบเสมอ เช่น disk full, defective disk และอื่น ๆ เสมอ errorเหล่านี้นิยามไว้ใน stdio.h ในลักษณะ # define หากไม่มีerrormessage ก็แปลว่าเราพร้อมที่จะบันทึกข้อมูลลงในไฟล์ก็ได้แล้ว หากมี errorให้แก้ไขให้ถูกต้องตามความจำเป็น

ต่อไปนี้เป็นตัวอย่างโปรแกรมสั่งเปิดไฟล์เพื่อบันทึกข้อมูลในรูป ASCII character

โปรแกรม 1

```
/* simple program to write ASCII text to a disk file */
# include "stdio.h"
# define "CLEARS 12"
main ( )
{
    char fname [80], c;
    int  ic-count ;
    FILE * f1, *fopen ( ) ;
    putchar (CLEARS) ;
    get-f (fname) ;
    if ( (f1 = fopen (fname, "w") ) = = NULL) {
```

```
        printf ("I can't create%s\n", fname);
        exit (1) ;
       }

    putchar (CLEARS) ;
    puts ("enter text (control-Q to end):") ;
    while((c = getchar( ))!='\021') /*021 = cont-Q*/
           aputc (c, f1) ;
    fclose (f1) ;
}
/* function to get the name of the file */
/* to which you wish to write */
/* it has been limited to 12 character-, including file type */
/* an example is PPPPPPPP.SSS */
int get-f(fname)
char name [ I ;
{
    int i, c-count, flag;
    char c ;
    puts("\n the file name can not have a primary name of") ;
    puts("\n more than 8 character, a period, and a file"), ;
    puts("\n extension of 3 character xxxxxxxx.yyy\n") ;
    flag = 1 ;
    while (flage) {
    puts("\n enter the name of the output file");
    gets (name) ;
    c-count = strlen (name) ;
    if(c-coutn > 12) {
           puts("filename too long\n") ;
    else if(c-count = = 12 && name [8] ! = '.')
           puts("filename and extension cannot exceed 12 chars \n");
    else
           flag = 0 ;
     }
 }
```

โปรแกรมเริ่มด้วยการกำหนดตัวแปรแบบต่าง ๆ รวมทั้งอะเรย์ชื่อ fname[ ] สำหรับบรรจุชื่อไฟล์รวมถึง pointer คือ f1 และ fopen( ) สำหรับชี้ไปที่ไฟล์ในโครงสร้างชื่อ FILE จากนั้นจึงล้างจอแล้วเรียกฟังก์ชั่น get-f ( ) ทำงานรับชื่อไฟล์

ฟังก์ชั่น get-f ( ) ทำหน้าที่รับอักขระเข้ามาโดยคอยตรวจดูด้วยว่าเราบันทึกเกิน 12 อักขระ (รวมจุดและ extension อีก 3 อักขระ) หรือไม่

เมื่อบันทึกชื่อไฟล์เรียบร้อยแล้วเราจะเรียกฟังก์ชั่น fopne( )ให้ทำงานโดยส่งอาร์กิวเมนต์คือ fname และ "w" ไปให้ เนื่องจากเราใช้ write mode ไฟล์ชื่อ fname จะถูกสั่งเปิด โดยฟังก์ชั่น exit (1)ทำหน้าที่แจ้ง OS ว่าการเปิดไฟล์ไม่สำเร็จ exit( ) ใช้สำหรับสั่งเลิกทำคำสั่งถ้าอาร์กิวเมนต์เป็น nonzero เช่น exit (1) ถ้าอาร์กิวเมนต์เป็น 0 เช่น exit (0) แปลว่าทุกอย่างเรียบร้อยดี

เมื่อเปิดไฟล์ได้ while loop จะทำหน้าที่รับข้อมูล (text) บรรจุลงไฟล์ด้วยฟังก์ชั่น getchar ( ) ด้วยคำสั่ง c = getchar ( ); คือ รับอักขระผ่านแป้นพิมพ์คราวละ 1 อักขระมาเก็บลงไว้ในที่ชื่อ c แล้วส่ง c และ f1 เป็นอาร์กิวเมนต์ไปให้ฟังก์ชั่น aputc ( ) ในทุกครั้งที่ getchar ( )รับข้อมูล โดย f1 เป็น file pointer ชี้ไปที่โครงสร้างชื่อ FILE และ pointer ชื่อ *-buff ในโครงสร้าง ก็จะชี้ต่อไปยังบัฟเฟอร์ที่เก็บหรือรับเอาอักขระเหล่านี้รวบรวมไว้

บัฟเฟอร์ buffer คือ พื้นที่ส่วนหนึ่งในส่วนความจำเพื่อรับข้อมูลไว้ชั่วคราวก่อนส่งต่อไปเก็บในที่ถาวร (เช่นบนดิสค์) กล่าวคือเมื่อบัฟเฟอร์รับข้อมูลไว้เต็ม โดยมากจะกำหนดให้เก็บได้ 128 หรือ 256 อักขระแล้วแต่ระบบ บัฟเฟอร์จะถ่ายเท (flush) ข้อมูลลงดิสค์ เมื่อถ่ายเทหมดบัฟเฟอร์ก็ว่างลงพร้อมที่จะรับข้อมูลได้อีก ทุกครั้งที่ถ่ายเทข้อมูลออกไปตัวแปรต่าง ๆ ในโครงสร้าง เช่น -cleft, *-nextc จะถูกปรับค่าใหม่-

(update) สู่ค่าเริ่มต้นที่เหมาะสมเพื่อนับและชี้ข้อมูลในบัฟเฟอร์ได้อย่างถูกต้องและดำเนินการรับข้อมูลและถ่ายเทข้อมูลอยู่ดังนี้เรื่อย ๆ ไปตราบเท่าที่เรายังไม่กด control-Q ซึ่งใช้แสดงให้ทราบว่าเราบันทึกข้อมูลหมดแล้ว

เมื่อกด control-Q โปรแกรมจะเรียกฟังก์ชั่น fclose ( )โดยส่ง file pointer คือ f1 ไปให้ฟังก์ชั่นfclose ( )จะทำหน้าที่ถ่ายเทข้อมูลชุดสุดท้ายแล้วปิดไฟล์

มีฟังก์ชั่นที่ถูกเรียกใช้ในที่นี้อีก 3 ฟังก์ชั่นคือ gets ( ), strlen ( ) และ aputc ( )ที่เราไม่ค่อยคุ้นเคยนอกนั้นเราเคยใช้มาบ้างแล้วเช่นputchar( ), puts( ) และ getchar ( ) จึงขออธิบายสรุปเพิ่มเติมไว้ดังนี้

gets ( )รับอักขระจากแป้นพิมพ์เรื่อยไปจนพบอักขระแสดง end of line หรือ end of file

strlen ( ) นับจำนวนอักขระในสตริง

putchar ( ) แสดงอักขระออกทางจอภาพ

puts ( ) แสดงสตริงออกทางจอภาพ

putc ( ) ใส่อักขระลงใน file buffer ทีละอักขระ

getchar ( ) รับอักขระเข้าทางแป้นพิมพ์คราวละอักขระ

getc ( )รับอักขระจากไฟล์คราวละ อักขระ

สำหรับการอ่านข้อมูลจากไฟล์เราสามารถเขียนโปรแกรมอ่านข้อมูลและแสดงผลบนจอภาพดังนี้

โปรแกรม 2

```
# include "stdio.h"
# define CLEARS 12
# define MAXTXT 2001
main (argc, argv)
int argc ;
char ** argv ; /* **argv equivalent to * argv [ 1 */
 {
     char c ;
     FILE * f1 * fopen ( ) ;
     putchar (CLEARS) ;
     if (argc! = 2) {
          puts (" n usage :program name filename") ;
          e x i t   (1) ;
      }
     if ((f1 = fopen (argv [1], "r")) = = NULL) {
          printf ("I can't open % s\ n" argv [1]);
          exit (1) ;
      }
     while ((c = getc (f1))! = EOF)
          putchar (c) ;
     fclose (f1) ;
 }
```

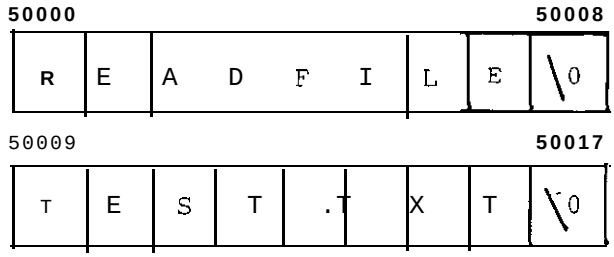
ขอให้สังเกตที่ main ( ) จะพบว่าใน main ( )รับอาร์กิวเมนต์ argc และ argv ไว้ ส่งที่ควรแก่การสงสัยก็คือ main( ) ที่เราเคยศึกษาผ่านมานั้นไม่เคยมีหรือไม่เคยต้องการอาร์กิวเมนต์ เนื่องจากเป็นโปรแกรมแรกหรือฟังก์ชั่นแรกสุด ถ้าหากต้องรับ

อาร์กิวเมนต์ ปัญหาก็คือใครเป็นผู้ส่งอาร์กิวเมนต์มาให้

ที่จริงแล้วแม้แต่ใน main ( ) ก็ต้องรับอาร์กิวเมนต์ ตามตัวอย่าง argc คือ จำนวน command line argument และ argv คือ pointer ที่ชี้ไปที่ command line argument เรียกว่า argument vector

command line argument คือ พารามิเตอร์ที่สั่งให้โปรแกรม ณ จังหวะเวลาที่โปรแกรมถูกเรียกทำงาน

สมมุติเราเก็บโปรแกรม 1 ไว้ในชื่อ TEST.TXT (คือไฟล์ชื่อ TEST.TXT) และโปรแกรม 2 ชื่อ READFILE ดังนั้น command line argument ที่ส่งไปให้ main( ) ในที่นี้ argc มีค่าเท่ากับ 2 และ argv [0] คือ READFILE ขณะที่ argv [1] คือ TEST.TXT และเนื่องจาก argv[0] และ argv[1] เป็นอะเรย์(argv [ ] เป็นอะเรย์ของอะเรย์) ดังนั้น argv[0] และ argv[1] จึงเป็น pointer ชี้ไปที่แอดเดรสของสตริงอันเป็นชื่อของไฟล์ดังนี้ สมมุติเริ่มที่แอดเดรส 50000



การเรียก argv[0] จึงมีผลเสมือนเรียกไฟล์ชื่อ READFILE และการเรียก argv[1] มีผลเสมือนเรียกไฟล์ TEST.TXT จึงอาจสรุปได้ว่าตัวแปร argc เก็บจำนวนอาร์กิวเมนต์เอาไว้ ขณะที่ argv[ ] ทำหน้าที่แจ้งให้ทราบว่าให้ตามไปรับอาร์กิวเมนต์ที่ใด (ในส่วน

ความจำ) ตัวแปรที่สองจึงเป็นอาร์กิวเมนต์ที่ main ( ) ต้องรับไว้ก่อนที่จะเริ่มทำคำสั่ง (execution)

สำหรับ ** argv ในคำสั่ง char **argv; แสดงว่า **argv เป็น pointer ที่ชี้ไปที่ pointer ที่ชี้ไปที่ char หรือการเขียนใหม่เป็น char*argv [1; อาจเข้าใจง่ายกว่า char*argv [ ]; แปลว่า argv [ ] เป็นอะเรย์ของ pointer ที่ชี้ไปที่ char

ลองดูตัวอย่างโปรแกรมต่อไปนี้ ซึ่งเป็นโปรแกรมแสดงความถี่ของจำนวนอักษร (ASCII letter) ใน text file โดยพล็อตกราฟด้วยวิธี direct cursor control

```
/* program to plot read ASCII text file */
/* and plot frequency of letters */
# include "stdio.h"
# define CLEARS 12
# define CURSOR '\033y'  /* viewpoint terminal */
# define BACK '\010'     /* backspace */
main (argc, argv)
int argc ;
char ** argv ;
{
int unit, max, cc, i, j, k, c, let [27];
char dit ;
FILE*f1*fopen ( );
putchar (CLEARS);
                        /* draw the axes */
for(j=1, j<22, ++j)
    set-cur (j, 1, '|') ;
set-cur (22, 1, '-') ;
```

```
for (j=0; j<78, ++j)
   putchar ('-') ;
                          /* clear the array */
for  (j=0,  j<27;  ++j)
    let [j] = 0  ;
                         /*open the file */
if((f1 = fopen(argv[1], "r")) = = NULL)
      printf ("can't open% s\n", argv[1]);
      exit (1) ;
 }
                      /* count the number of character */
while ((c = getc(f1)! = EOF) {
      c = toupper (c) ;
     if (c > = 'A' && c< 'Z')
         let [c-'A'] + = 1 ;
 }
fclose(f1) ;         /* close the file */
j = 27 ;
max = f-max (j, let);    /*find biggest */
set-cur (1, 2, BACK) ;
printf("%d", max) ;    /* and print it */
unit = max/20           /*scale to biggest */
i  f    (unit<1)
     unit = 1 ;
                        /* do histogram of the count */
for(j=0, cc=0; j<27; ++i, ++j) {
      cc+= 3 ;
      k  = 21- (let [j]%unit) ;
      dit  = (let[j] = = 0)? ' ' : '*';
```

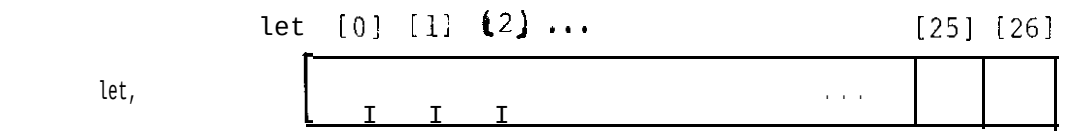
```
        for(i=k; i<22; ++i)
              set-cur (i, cc, dit) ;
        {
        set-cur(23, 2,' ');   /* print axes label */
        for (j='A'; j< = 'Z'; ++j)
               printf("%c", j) ;
        set-cur(22, 1, BACK); /* prevent scroll */
}
/* function that use row-column */
/* to position the letter to be printed */
set-cur (row, col, let)
int row, col, let ;
{
        printf ("%s%c%c%c", CURSOR, row+31, col+31, let);
}
/* function to find the largest count */
/*needs   array        */
/* and maximun number of elements in array */
f-max (el, num)
int el, num I 1 ;
{
        int biggest ;
        for (i=0, biggest = 0, i<el; ++i)
              if (num [i] > = biggest!
                    biggest = num [i];
              return (biggest) ;

 }
```

โปรแกรมเริ่มต้นด้วยการกำหนดตัวแปรสั่ง ปิดไฟล์ ล้างจอ วาดภาพแกนตั้ง วาดภาพแกนนอน ล้างอะเรย์ชื่อ let [ ] แล้วนับจำนวนอักษรที่อ่านจากไฟล์มาเก็บไว้ใน อะเรย์ let [ ]โดย let [0] เก็บจำนวนอักษร a ใน text let [1]เก็บจำนวน อักษร bในtext...let [25]เก็บจำนวนอักษร z ใน text แล้วปิดไฟล์ จากนั้นจึงเรียก ฟังก์ชัน f-max( )และ set-cur ( )ทำงานสร้างฮิสโตแกรม คำอธิบายโดยละเอียด ปรากฏดังนี้

โปรแกรมเริ่มต้นด้วยการเรียกฟังก์ชั่น set-cur ( )เพื่อวาดแกนโคออร์ดิ-เนต โดยฟังก์ชัน set-cur ( ) รับอาร์กิวเมนต์คือ row, col และอักขระที่ต้องการ ให้พิมพ์ การวาดแกนโคออร์ดิเนตเริ่มด้วยการวาดแกนดิ่ง อักขระที่ให้พิมพ์คือ "|" และ วาดแกนนอน อักขระที่ให้พิมพ์คือ "-"

จากนั้นจึงล้างอะเรย์ let [ ] ให้ว่างเพื่อเก็บจำนวนนับของอักขระจาก text file ดังภาพ



แล้วเริ่มเปิดไฟล์โดยอาศัย command line argumentคือ argv [1] หากเปิดไม่ได้จะ แจ้งerror messegeว่า "can't open" ถ้าเปิดได้ file pointer คือ f1 จะชี้ ไปที่text file ในโครงสร้าง FILE ที่ต้องการ

การอ่านข้อมูลจากไฟล์เราเรียกใช้ฟังก์ชัน getc ( ) โดยอ่านไปเรื่อย ๆ จนกว่าจะพบ EOF ในแต่ละครั้งที่อ่านเราเรียกฟังก์ชัน toupper ( ) มาทำหน้าที่แปลง อักษรให้เป็นอักษรตัวใหญ่โดยมีคำสั่ง if คอยตรวจเช็คว่าอักขระที่รับผ่าน getc ( ) มา

นั้นเป็นตัวอักษรหรือไม่ ถ้าใช่ก็ให้นับแล้วเก็บไว้ ณ ตำแหน่งที่เหมาะสมในอะเรย์ let [ ]

ตัวอย่างเช่นข้อความใน text file คือ zoo คำสั่ง let[c-'A'] + = 1; จะทำงานดังนี้

1. เมื่อส่งอักษร z เข้ามาจะพบว่า let[25] = let[25] + 1 ; แต่ let[25] = 0 เพราะล้างอะเรย์ดังนั้น let[25] + 1 = 0+1 = 1 หรือใน let [25]เก็บเลข 1 เลข 25 มาจากรหัสแอสกี้ของ z กับ A ลบกันคือ 90-65

2. เมื่อส่งอักษร o เข้ามาจะพบว่า let[14] = let[14]+1; แต่let [14] = 0 เพราะล้างอะเรย์ดังนั้น let [14]+1 = 0+1 หรือใน let [14] เก็บ เลข 1 เลข 14 มาจากรหัสแอสกี้ของ o กับ A ลบกันคือ 79-65

3. เมื่อส่งอักษร o ตัวที่ 2 เข้ามาจะพบว่า let[14] = let[14]+1 = 1 + 1 = 2 ตอนนี้ let [14]จะเก็บหมายเลข 2 เอาไว้ เลข 2 ก็คือจำนวนอักษร o ใน text file

ปิดไฟล์ด้วยฟังก์ชัน fclose (f1) ;

เรียกฟังก์ชัน f-max ( ) ให้ทำหน้าที่ตรวจดูว่า ตัวแปร let [ ] ตัวใด เก็บจำนวนนับไว้มากที่สุด หรือนัยหนึ่งคือตรวจดูว่ามีอักขระใดมากที่สุด (เป็นเรื่องของการ search) เมื่อพบให้ใส่จำนวนดังกล่าวในที่ชื่อ max จำนวนที่สูงที่สุดนี้เราจะใช้สำหรับ กำหนดสเกลของแกนเพื่อให้ฮิสโตแกรมสูงไม่เกิน20บรรทัด (เราอาจใช้น้อยกว่า 20 บรรทัดก็ได้ถ้าต้องการรูปที่เล็กลง)เช่นอักษร m ใน text file มีทั้งสิ้น 4000 ตัว ซึ่ง มากกว่าอักษรตัวอื่น ๆ เมื่อเรากำหนดให้แท่งสูงสุดสูงไม่เกิน 20 บรรทัด ก็แปลว่าเรา

ใช้ 20 ในแกนแทน 4000 หรือแต่ละหน่วยของสเกลแทนจำนวนเท่ากับ 4000/20 = 200 หน่วย ในโปรแกรมเรากำหนดให้ผลหารที่น้อยกว่า 1 มีค่าเป็น 1 (ดูคำสั่ง unit = max/20 ; if (unit < 1) unit = 1 ;)

for loop คือ

```
for (j=0; cc=0; j<27; ++j){
    cc + = 3 ;
    k = 21-(let[j]/unit) ;
    dit = (let[j] = = 0)? ' ':'*' ;
    for (i=k; i<22; ++i)
        set-cur (i, cc, dit) ;
}
```

ใช้สำหรับพล็อตกราฟคราวละสดมภ์โดยเริ่มตั้งแต่ let [0] ก่อนเมื่อเสร็จ let [0] คือจำนวนอักษร a ก็เลื่อนไปสู่สดมภ์ต่อไป สดมภ์ห่างกัน 3 อักขระ การพล็อตใช้ดอกจันเป็นเครื่องหมาย ถ้าไม่มีข้อมูลให้เว้นว่าง สั่งพล็อตกราฟโดยเรียกฟังก์ชัน set-cur (i, cc, dit) ; i คือ row cc คือ column และ dit คือดอกจัน

ตัวอย่างเช่น text file มีอักขระ a ทั้งสิ้น 3000 ตัวคือ let [0] = 3000 จะพบว่า

k = 21 - (3000/200) ;

7

for loop คือ for (i=k; i < 22; ++i)

set-cur (i, cc, dit) ;

จะวนเวียนทำงานโดยพิมพ์ดอกจัน (*) ลงในพิกัด (7,3), (8,3), (9,3), ... (21, 3) รวมทั้งสิ้น 15 ดอก

รอบต่อไปตรวจที่ let[1] คืออักษร b สมมุติมีอักษร b ทั้งสิ้น 2000 ตัวคือ let[1] = 2000 จะพบว่า

$$k_1 = 21 - (2000/200)_1$$

for loop คือ

```
for(i = k; i<22; ++i)
    set - cur (i, cc, dit) ;
```

จะวงเวียนทำงานโดยพิมพ์ดอกจันทน์ลงในพิกัด (11,6), (12, 6), ... , (21, 6)

ดังนี้เรื่อยไปจนกระทั่งครบทุกอักษร

จากนั้นจึงพิมพ์ชื่ออักษรคือ A, B, ..., Z กำกับใต้ฮิสโตแกรมโดยพิมพ์ที่บรรทัดที่ 23 ณ สดมภ์ที่ 2 เป็นต้นไป

ท้ายสุดคือบังคับเคอร์เซอร์ให้ขยับขึ้นไปที่พิกัด (22, 1) ทั้งนี้ เพื่อเป็นการป้องกันมิให้เกิดการเลื่อนจอ (scroll) ด้วยเหตุบังเอิญหรือเหตุอื่น ๆ

ผู้อ่านสามารถดัดแปลงโปรแกรมนี้เพื่อสั่งพล็อตกราฟลักษณะอื่นได้ตามที่เห็นว่าเหมาะสมเพื่อให้ได้กราฟที่สวยงามหรือละเอียดมากน้อยตามต้องการ

9.3 Unbuffered file I/O หรือ low-level file I/O

unbuffered file I/O คือฟังก์ชั่นที่ทำงานติดต่อตรงกับ OS ฟังก์ชั่นเหล่านี้จะใช้ file descriptor ชี้ตำแหน่งไฟล์แทน file pointer ดังที่เคยศึกษาผ่านมาในตอน 9.1 และ 9.2 โดย file descriptor จะมีค่าเป็น 3 ค่า คือ 0, 1, 2

โดย 0 หมายถึงอ่านค่าจากไฟล์ (read) 1 หมายถึงบันทึกข้อมูลลงไฟล์ (writing) และ 2 หมายถึงทั้งอ่านและบันทึก (ค่า 0 จะถูกกำหนดให้ sdin, ค่า 1 ถูกกำหนดให้ sdout และ 2 ถูกกำหนดให้ sterr)

โดยปกติ unbuffered file I/O (ประกอบด้วย open ( ), read ( ), write ( ), close ( ) และอื่น ๆ) จะถูกเรียกทำงานโดย buffered file I/O การสั่งทำงานห้ามใช้ทั้ง buffered file I/O และ unbuffered file I/O ในเวลาเดียวกัน ให้เลือกใช้อย่างใดอย่างหนึ่งเพียงอย่างเดียว

1) ฟังก์ชัน open ( )

รูปไวยากรณ์ของ open ( ) คือ

```
open (name, mode) 1/
char * name ;
int mode
```

name หมายถึงชื่อไฟล์ที่เราสั่งเปิด ซึ่งเป็น character string อาจใช้ command line argument เป็นชื่อไฟล์ได้

mode คือสิ่งที่เราต้องการทำงานโดยที่

1/ ฟังก์ชัน open ( ) ใช้สำหรับสั่งเปิดไฟล์ที่มีอยู่ ถ้าไฟล์ใดที่เราต้องการเปิดใหม่ให้ใช้ create ( ) ซึ่งมีรูปไวยากรณ์เป็น

```
create (name)
char name ;
```

แต่คอมไพเลอร์บางรุ่นอาจไม่มีฟังก์ชั่นซึ่งเราก็สามารถใช้ open ( ) แทนได้แต่ขอให้ระวังอย่าใช้ชื่อไฟล์ซ้ำ การสั่ง open ( ) จะลบข้อมูลเดิมทิ้งถ้าใช้ write mode

= 0 สำหรับอ่านข้อมูลจากไฟล์ (read)

= 1 สำหรับบันทึกข้อมูลลงไฟล์ (write)

= 2 สำหรับทั้งอ่านและบันทึก (read and write)

รูปแบบการใช้งานคือ

```
file descriptor = open ( name, mode) ;
```

file descriptor (fd) คือตัวเลขชี้หมายเลขของไฟล์ในอะเรย์ (ดูตอน 9.1) ซึ่งมีความหมายเหมือน X ในคำสั่ง OPEN # X ในภาษาเบสิค

ตัวอย่างเช่น

```
fdl = open (name, 0) ;
fd2 = open (adress, 0 ) ;
```

แสดงว่าเราต้องการเปิดไฟล์ชื่อ name ที่เก็บชื่อลูกค้าและไฟล์ชื่อ address ที่เก็บบ้านเลขที่ของลูกค้าเพื่ออ่าน

เราสามารถใช้ command line argument แทนชื่อไฟล์และใช้คำสั่ง if เพื่อตรวจว่ามีมีปัญหาในการเรียกเปิดไฟล์หรือไม่ได้ดังนี้

```
if (fdl = open (argv[1], 0) = = ERR){
      printf (" n can't open %s", argv[1] ) ;
      exit (1) ;

}
```

2) ฟังก์ชัน read ( )

รูปไวยากรณ์ของ read ( ) ปรากฏดังนี้

```
read (fildes, buffer, count)
int fildes ;
char * buffer ;
int count ;
```

fildes คือตัวเลขแสดง file descriptor ที่รับมาจาก open ( ) แปลว่าเราจะใช้ read ( ) ไม่ได้จนกว่าจะได้ทำการเปิดไฟล์ด้วย open ( ) มาแล้วและได้รับ fd จาก open ( ) ขอให้สังเกตว่าใน read ( ) ไม่มีชื่อไฟล์เป็นอาร์กิวเมนต์ การสื่อสารกับไฟล์ใน read ( ) จึงสื่อสารผ่าน fd มิใช่ผ่านชื่อไฟล์

buffer หมายถึงที่รับข้อมูลชั่วคราวเพื่อถ่ายเทข้อมูลกันระหว่างดิสค์กับโปรแกรม ในที่นี้ buffer เป็นอะเรย์ ซึ่งมีขนาดเท่ากับจำนวนไบท์ที่เราประสงค์จะรับข้อมูลจากดิสค์มาเก็บไว้ในแต่ละคราว เช่นมีขนาดเท่ากับ 1 เซคเตอร์ (คือ 128 ไบท์ตาม CP/M หรือ 512 ไบท์ตาม UNIX)

ดังนั้น buffer จึงหมายถึงแอดเดรสในส่วนความจำที่เก็บข้อมูลไว้ให้อ่าน

count หมายถึงกลุ่มข้อมูล หรือจำนวนข้อมูลที่จะสั่งให้อ่านในแต่ละคราว โดยปกติจะมีค่าเท่ากับจำนวนไบท์ในเซคเตอร์บนดิสค์ (128 ไบท์สำหรับ CP/M และ 512 ไบท์ สำหรับ UNIX)

แต่เราอาจกำหนดให้ count มีค่าเท่ากับ 1 แปลว่าเราสั่งอ่านข้อมูลจากไฟล์คราวละ 1 อักขระ

รูปแบบที่ใช้ในทางปฏิบัติคือ

```
num-byte = read(file descriptor, buffer, count) ;
```

แปลว่าเมื่อ read ได้รับอาร์กิวเมนต์ก็จะทำงานคืออ่านข้อมูลในไฟล์หมายเลข fd โดยอ่านข้อมูลในบัฟเฟอร์คราวละ count หน่วย เมื่ออ่านจบ read ( ) จะส่งสัญญาณเป็นค่าส่งคืนมา ใส่ใน num-byte โดยอาจส่งเป็นจำนวนไบท์ทั้งหมดในบัฟเฟอร์มาให้ ซึ่งแสดงว่าเหตุการณ์ปกติ หรือส่งเลข -1 ซึ่งแสดงว่ามีข้อผิดพลาด (ERR = -1 นิยามไว้ใน - stdio.h) หรือส่งเลข 0 ซึ่งแสดงว่าอ่านจนจบไฟล์ (EOF) แล้ว

ตัวอย่างเช่น

```
while ((count = read (fd1, buffer, 128)) > 0) {
        write (fd2, buffer, count) ;

}
```

หรือ

```
while (num-byte = read (fd2, buffer, BUFF)){
    printf("%d", num-byte) ;
    if (num-byte = = ERR) {
        printf ("trouble reading %s\n", argv[2]) ;
        exit (ERR) ;

    }
```

3) ฟังก์ชัน write ( )

ฟังก์ชัน write ( ) ทำงานเหมือน read ( ) ต่างกันเพียง write ( ) จะบันทึกข้อมูลลงดิสค์ รูปไวยากรณ์ของ write ( ) คือ

```
write (fildes, buffer, count)
int fildes ;
char * buffer ;
int count ;
```

เช่นคำสั่ง write (fd1, buffer, count) ; จะสั่งให้รับข้อมูลทั้งสิ้น count อักขระจาก buffer แล้วบันทึกลงในไฟล์ชื่อ fd1

ค่าส่งคืนจาก write ( ) คือจำนวนไบท์ที่บันทึกลงดิสค์ในแต่ละครั้ง (คือคราวละบัฟเฟอร์) หากจำนวนเลข count กับจำนวนเลขส่งคืนจาก write ( ) ต่างกันก็แสดงว่ามีปัญหาเกิดขึ้นแล้ว เช่น ดิสค์เต็มเป็นต้น และเพื่อป้องกันปัญหาเราควรมีข้อความสำหรับเตือนว่าได้เกิดปัญหาขึ้นแล้วเราควรเขียนโปรแกรมดังนี้

```
if (write (fd1, buffer, count)! = count) {
   printf("error occurred during write to%s", argv[1]);
   exit (1) ;
  }
```

4) ฟังก์ชัน close ( )

ฟังก์ชัน close ( ) ใช้สำหรับสั่งปิดไฟล์ มีรูปไวยากรณ์ดังนี้

```
close (fildes)
int fildes ;
```

การปิดไฟล์จะถูกเรียกใช้เมื่อเราประสงค์จะปิดไฟล์ เมื่อสั่ง close ( )ข้อมูลในบัฟเฟอร์สุดท้ายอาจถูกสั่งเท (flush) ลงดิสค์ หรือไม่ก็ได้แล้วแต่คอมไพเลอร์ เรื่องนี้หากมีฟังก์-ชัน flush ( ) ใช้ก็ควรเรียกใช้

ตัวอย่างการใช้ close ( )ปรากฏดังนี้

```
clole (fd1) ;
close, (fd2) ;
```

หรือ

```
if (close (fd1) = = ERR) {
printf ("can't c l o s e file%s n", argv [1]);
  e x i t (ERR);
}
  close fd2 ;
```

เมื่อปิดไฟล์แล้ว เราสามารถใช้ file descriptor ไปเปิดไฟล์ใหม่ได้อีก

นอกจากนี้เรายังมีฟังก์ชันที่เรียกว่า random access technique ที่ใช้ในกรณีที่เราประสงค์จะเข้าถึงข้อมูลในไฟล์.ณ ตำแหน่งใด ๆ ในไฟล์ โดยมิต้องเริ่มต้นหาไปตั้งแต่อักขระแรกของไฟล์ ไปจนถึงอักขระสุดท้าย ซึ่งจะช่วยลดเวลาการเข้าถึงข้อมูล -(access time)ลงได้มาก (ที่ศึกษาผ่านมา เช่น read ( ) เป็น sequential access ภายในไฟล์ เพราะจะต้องเริ่มอ่านตั้งแต่อักขระแรกเป็นต้นไป) เช่น lseek ( ) และ tell ( ) โดย tell ( ) ใช้บอกตำแหน่งปัจจุบันในไฟล์ จะกล่าวถึงฟังก์ชันทั้งหลายในส่วนที่เกี่ยวกับ I/O อีกครั้งเป็นการสรุป I/O ในภาคผนวกที่ 4 และต่อไปนี้จะเป็นตัวอย่างการใช้ฟังก์ชัน I/O ที่ศึกษาผ่านมาแล้ว

ตัวอย่างต่อไปนี้คือโปรแกรมสำหรับก๊อปปี้ไฟล์ที่มีอยู่ในที่อื่น โดยจะแสดงโปรแกรมนี้เป็น 3 แบบโดย 2 แบบแรกจะเขียนค่อนข้างย่อโดยตัด loop สำหรับเช็คข้อผิด-

พลาดออก ขณะที่โปรแกรมที่ 3 จะแสดง loop เหล่านี้ไว้โดยละเอียด

ก.

```
/* file copy - buffered */
# include "stdion.h"
main ( )
/* copies file named INPUT to named OUTPUT */
{
    int c ;
    FILE * file 1, *file 2 * fopen( ); /* open both files */
    file 1 = fopen("INPUT", "r");
    file2 = fopen("OUTPUT", "w") ;
    /* loop on each character. read */
    while((c = fgetc))! = EOF) {
            fputc (c, file 2);
    }
    fclose (file 1) ;
    fclose (file 2) ;
    exit (0) ;
}
```

ข.

```
/* filecopy-unbuffered */
# define SIZE 128
main ( )
/*copies file named INPUT to file named OUTPUT */
{
```

```
        int fdl, fd2, count ;
        char buffer [SIZE] ;
        /* open files */
        fdl = open ("INPUT", 0) ;
        fd2 = create ("OUTPUT", 0);
        /* loop on reads */
        while ((count = read (fd1, buffer, SIZE)) > 0){
              write (fd2, buffer, count) ;
         }
        close (fd1) ;
        close (fd2) ;
       exit(0)  ;
}
```

9.

```
/* program to copy existing file to new file */
/* the program is invoked as */
/*program name new filename existing-filename */
# include "stdio.h"
#  define  CLEARS  12
# define READF 0
#  define  WRITEF  1
# define BUFF  128    /* common sector for CP/M */
#  define  ERR-1       /* if  trouble  occurred */
char  buffer[BUFF]; /* pass data through here */
int fdl, fd2;
main (argc, argv)
int argc ;
char ** argv ;
{
```

```
int num_byte ;   /* check on data passed */
putchar (CLEARS);
                 /* enough command line argument? */
if (argc! = 3){
     puts ("need : destination and source filename-1 ;
     exit (ERR) ;
 }
                 /* try opening source file */
if ((fd2 = open (argv [2], READF)) = = ERR) {
     printf ("can't open%s\n" , argv [2]);
     exit (ERR) ;
 }
                 /* try creating destination' file.*/
if ((fd1 = open (argv [1], WRITEF)) = = ERR)
     printf ("can't create%s n", argv [1]),
     exit (ERR) ;
 }
                 /* do the copy */
puts ("\n starting to copy variable num_byte is :\n") ·
while (num_byte = read (fd2, buffer, BUFF)) {
       printf ("%d", rum-byte);
       if (num_byte = = ERR)
          printf ("trouble reading%s\n", argv [2]);
          exit (ERR) ;
     }
       if (write (fd1, buffer, num byte)! = num_byte)
          printf ("trouble writing%s\n", argv [1]) ;
          exit (ERR) ;
     }
```

```
                    /* wind i t  up */
        if (colse (fd1) = = ERR) {
              printf ("can't close file%s\n", argv [1]) ;
              exit (ERR) ;
        }
        close (fd2) ;
        putchar ('\007') ;  /*bell when done */
        puts ("\n all done") ;
}
```