บทที่ 9 ไฟล์อินพุต/เอาต์พุต

สิ่งสำคัญประการหนึ่งที่ทำให้คอมพิวเตอร์เป็นเครื่องมือที่ใช้ประโยชน์ได้นอกเหนือจากการคำนวณเพียงอย่างเดียว ก็คือ ความสามารถในการบันทึกข้อมูลในหน่วยความจำสำรอง ซึ่งทำให้เราสามารถใช้เครื่องคอมพิวเตอร์เป็นที่เก็บข้อมูล ค้นหาข้อมูล ตลอดจนการนำข้อมูลที่มีอยู่มาใช้ประโยชน์ต่างๆ อันได้แก่ ระบบบัญชี ระบบทะเบียน ฐานข้อมูลสำมะโนครัว ระบบ MIS (Management Information System) ระบบ GIS (Geographic Information System) เป็นต้น ในภาษา C และภาษาอื่นๆทั่วไป ก็จะสนับสนุนให้ผู้เขียนสามารถเขียนโปรแกรมให้บันทึกข้อมูลลงไปในหน่วยความจำสำรอง หรือที่เรียกกันว่า *ไฟล์* นั่นเอง โดยในภาษา C นั้น จะมีตัวแปรชนิด FILE ไว้สำหรับเก็บข้อมูลที่เป็นไฟล์ และมีฟังก์ชันมาตรฐานหลายๆฟังก์ชันไว้สำหรับใช้งานไฟล์ อันได้แก่ File Input, File Output และอื่นๆ

9.1 ตัวแปรไฟล์ (File variable)

ในภาษา C จะมีตัวแปรชนิด File ไว้สำหรับใช้บันทึก File ซึ่งได้ถูกกำหนดนิยามไว้แล้วในไฟล์ชื่อ *stdio.h* ตัวแปร FILE นี้ จัดเป็น *structure* ชนิดหนึ่ง การใช้ไฟล์จะเริ่มต้นโดยการประกาศตัวแปรไฟล์ ซึ่งมีวิธีประกาศดังนี้

```
FILE  *file_variable ;
```

ตัวอย่าง เช่น

```
FILE  *in_file ;
```

ให้สังเกตว่า คำศัพท์ที่ใช้ในการประกาศตัวแปรชนิดไฟล์คือ คำว่า FILE ซึ่งต้องใช้ตัวพิมพ์ใหญ่ทั้งหมด และต้องประกาศเป็นตัวแปร Pointer เท่านั้น ไม่มีการประกาศเป็นตัวแปรเก็บค่าธรรมดา

หลังจากประกาศตัวแปรไฟล์แล้วก็ทำการเปิดไฟล์ และกำหนดให้ตัวแปร Pointer ดังกล่าวชี้ไปที่ตำแหน่งที่อยู่เริ่มต้นของโครงสร้างไฟล์

***ไฟล์มาตรฐาน* (*Standard Files*)**

ในภาษา C จะมี File มาตรฐานที่ Compiler จัดการสร้างขึ้นเองโดยอัตโนมัติ กล่าวคือ จะสร้างไฟล์เหล่านี้ทุกครั้งที่รันโปรแกรม ไฟล์เหล่านี้ได้แก่

ไฟล์	คำอธิบาย
stdin	อินพุตมาตรฐาน (เปิดสำหรับอ่าน)
stdout	เอาต์พุตมาตรฐาน (เปิดสำหรับเขียน)
stderr	ข้อผิดพลาดมาตรฐาน (เปิดสำหรับเขียน)

stdin ย่อมาจาก *Standard Input* หมายถึง ไฟล์ที่ถูกป้อนเข้ามาในลักษณะ re*direct* ซึ่งถ้าใน DOS หรือ UNIX จะทำได้โดยใช้เครื่องหมาย < โดยสั่งงานในลักษณะ

```
program_name  arguments < filename
```

ส่วน *stdout* ย่อมาจาก *Standard Output* หมายถึง ไฟล์ที่ถูกสร้างขึ้นมาเพื่อใช้บันทึกผลที่ได้การแสดงผล ถ้าไม่มีการกำหนดชื่อไฟล์อะไรเป็นพิเศษ โปรแกรมก็จะทำการแสดงออกทางหน้าจอ แต่ใน DOS หรือ UNIX เราสามารถระบุชื่อไฟล์ที่จะใส่สำหรับบันทึกข้อมูลนี้ได้ โดยใช้เครื่องหมาย > โดยทำในลักษณะ

```
program_name  arguments > filename
```

ส่วน *stderr* ย่อมาจาก *Standard Error* หมายถึง ไฟล์ที่ถูกสร้างขึ้นมาเพื่อบันทึก Error Message ในกรณีปกติ ไฟล์นี้ จะไม่ต่างไปจาก *stdout* กล่าวคือ จะถูกแสดงออกทางหน้าจอเช่นเดียวกัน แต่ในกรณีที่กำหนดให้ Standard Output เป็นไฟล์ ถ้าหากข้อมูลทุกอย่างของ Output ถูกส่งไปเก็บในไฟล์ที่ว่านี้ เราจะไม่สามารถรู้ข้อผิดพลาดของ

โปรแกรม ไฟล์ *stderr* นี้จะทำหน้าที่บันทึกข้อผิดพลาด แล้วแสดงออกทางหน้าจอ เพื่อให้ผู้ใช้สามารถรับรู้ข้อผิดพลาดของโปรแกรมได้

9.2 ฟังก์ชันเปิด/ปิดไฟล์ (File open/close function)

ในการใช้ไฟล์ ก่อนอื่นเราจะต้องสั่งให้โปรแกรมทำการอ่านไฟล์นั้นมาก่อน โดยใช้ฟังก์ชัน *fopen* และควรจะทำการสั่งให้ปิดไฟล์โดยใช้ฟังก์ชัน *fclose* เมื่อเลิกใช้ เพื่อโปรแกรมจะได้ทำการเก็บบันทึกข้อมูลนั้นลงไปในไฟล์ และจะได้เลิกใช้เนื้อที่หน่วยความจำส่วนนั้น

ฟังก์ชันเปิดไฟล์ : fopen

วิธีใช้ : `file_variable = fopen(name, mode) ;`

ตารางที่ 9.1 โหมดของฟังก์ชัน *fopen*

MODE	Process	FILE NOT EXIST	FILE EXIST
"r"	read	Return NULL	Normal
"w"	write	Create	Overwrite
"a"	append	Create	Write at the end of file
"r+"	read-write	Return NULL	Normal
"w+"	read-write	Create	Overwrite
"a+"	append	Create	Write at the end of file

ตัวอย่างการใช้ ได้แก่

```
in_file = fopen("test.txt", "r") ;
```

ฟังก์ชันปิดไฟล์ : fclose

fclose มีไว้สำหรับปิดไฟล์เมื่อเลิกใช้งาน โดยมีวิธีการใช้ดังนี้

```
fclose(file_variable) ;
```

ตัวอย่าง เช่น

```
(void)fclose(in_file) ;
```

9.3 ฟังก์ชันไฟล์อินพุต/เอาต์พุต (File Input/Output function)

fgetc เป็นฟังก์ชันสำหรับอ่านตัวอักษร 1 ตัวจากไฟล์เข้ามา ถ้าเกิดไม่มีตัวอักษรให้อ่านแล้ว ฟังก์ชันนี้จะคืนค่า EOF (EOF ถูกกำหนดไว้แล้วใน *stdio.h*) สังเกตว่า **fgetc** จะคืนค่ามาเป็น integer ไม่ใช่ character ซึ่งเป็นสิ่งจำเป็น เพราะ EOF ไม่ใช่ตัวอักษร โดยมีวิธีใช้ดังนี้

```
int_variable = fgetc(in_file) ;
```

fputc เป็นฟังก์ชันสำหรับเก็บตัวอักษร 1 ตัวลงในไฟล์ โดยมีวิธีใช้ดังนี้

```
fputc(character, file)
```

ตัวอย่าง เช่น

```
fputc('c', out_file) ;
```

หมายถึง เป็นการสั่งให้โปรแกรมเก็บตัวอักษร c ลงในไฟล์ *out_file*

ตัวอย่างที่ 9.1 : ตัวอย่างการใช้ฟังก์ชัน fgetc และ fputc

```
/* Program for counting the number of characters in the file
      "input.txt" and copy to "output.txt"    */
#include <stdio.h>
#include <stdlib.h>
#define IN_NAME "input.txt"
#define OUT_NAME "output.txt"
int main()
{
```

```
        int    count = 0 ;
        FILE   *in_file ;
        FILE   *out_file ;
        int    ch ;
        printf("%s %s\n", IN_NAME, OUT_NAME) ;
        in_file = fopen(IN_NAME, "r") ; /* Open for reading */
        out_file = fopen(OUT_NAME, "w") ; /* Open for writing */
        if ( in_file == NULL || out_file == NULL ) { /* Error Check */
               printf("open file error \n") ;
               exit(1) ;
        }
        while(1) {
               ch = fgetc(in_file) ;
               if (ch == EOF) break ;
               else {
                      count++ ;
                      fputc(ch, out_file) ;
               }
        }
        printf("number of character in %s is %d\n", IN_NAME, count) ;
        fclose(in_file) ;
        fclose(out_file) ;
        return 0;
}
```

ผลการรันโปรแกรม

```
leibniz:~/program$ cat input.txt
Time is a great teacher, but unfortunately it kills all its pupils.
Berlioz
leibniz:~/program$ cat output.txt
cat: cannot open output.txt
leibniz:~/program$ a.out
input.txt output.txt
number of character in input.txt is 76
leibniz:~/program$ cat output.txt
Time is a great teacher, but unfortunately it kills all its pupils.
Berlioz
```

fgets สำหรับอ่าน string จากไฟล์ ซึ่งจะอ่าน string ทีละ 1 บรรทัด และอ่าน *size* ตัวอักษร หรืออ่านจนถึงสุดบรรทัด โดยมีวิธีใช้ดังนี้

```
        string_ptr = fgets(string, size, file) ;
```

หรือ

```
        (void) fgets(string, size, file) ;
```

โดยที่

string_ptr เป็นค่า Pointer *string* ที่อ่านกลับมากรณีที่อ่านได้ และเป็น NULL กรณีที่เป็น EOF หรือเกิดข้อผิดพลาดขึ้น

string เป็น pointer ที่ชี้ไปที่หน่วยความจำที่สำหรับเก็บค่า string ที่อ่านเข้ามา

size ขนาดของ string ที่จะอ่านเข้ามา

ตัวอย่าง การใช้ได้แก่

```
        char   string[100] ;
        ...
        (void)fgets(string, sizeof(string), in_file);
```

fputs ฟังก์ชันนี้คล้ายกับ **fgets** จะต่างกันที่แทนที่จะอ่าน string จากไฟล์ จะทำการเขียน string ลงไฟล์ โดยจะคืนค่าที่ไม่เป็นจำนวนเต็มลบในกรณีที่ทำงานปกติ และคืนค่า EOF ในกรณีที่เกิดปัญหาขึ้น วิธีการใช้เป็นดังนี้

```
        value = fputs(string, file) ;
```

หรือ

```
        (void)fputs(string, file) ;
```

ตัวอย่าง เช่น

```
(void)fputs(string, out_file) ;
```

ตัวอย่างที่ 9.2 : ตัวอย่างการใช้ฟังก์ชัน fgets และ fputs (คล้ายกับตัวอย่างที่ 9.1)

```
#include <stdio.h>
#include <stdlib.h>
#define FILE_NAME "input.txt"
#define OUT_NAME "output.txt"
int main()
{
       FILE   *in_file ;
       FILE   *out_file ;
       char   str[100], *str_ptr ;
       in_file = fopen(FILE_NAME, "r") ;
       out_file = fopen(OUT_NAME, "w") ;
       if ( in_file == NULL || out_file == NULL ) {
              printf("open file error \n") ;
              exit(1) ;
       }
       while(1) {
              str_ptr = fgets(str, 100, in_file) ;
              if (str_ptr == NULL) break ;
              else
                     fputs(str, out_file) ;
       }
       fclose(in_file) ;
       fclose(out_file) ;
       return 0;
}
```

ผลการรันโปรแกรม ผลที่ได้จะเหมือนกับในตัวอย่างที่ 9.1

fprintf ในการสั่งให้โปรแกรมแสดงผลออกทางหน้าจอหรือทาง Printer เราจำเป็นจะต้องแปลงข้อมูลที่ต้องการจะแสดงให้เป็นตัวอักษร ทั้งนี้ ก็เพราะหน้าจอ หรือ Printer สามารถแสดงได้เฉพาะตัวอักษรเท่านั้น ไม่สามารถทำความเข้าใจข้อมูลชนิดอื่นได้ ยกตัวอย่างเช่น เลขจำนวนเต็ม 567 จะต้องแปลงเป็นตัวอักษร 5, 6, 7 จึงจะสามารถแสดงผลออกมาได้ ฟังก์ชัน *printf* จะทำหน้าที่แปลงให้เป็นตัวอักษร แล้วแสดงออกทาง Standard Output (โดยมากจะหมายถึงหน้าจอ) ส่วนฟังก์ชัน *fprintf* จะต่างกับ *printf* ตรงที่ว่า แทนที่จะแสดงออกทางหน้าจอ จะเขียนลงไปในไฟล์ ดังนั้นจะต้องมีอาร์กิวเมนต์ที่ระบุถึงไฟล์ที่จะเขียนลงไปเพิ่มขึ้นมา

รูปแบบการใช้เป็นดังนี้

```
       count = fprintf(file, format, parameter_1, parameter_2, ...) ;
```

หรือ

```
       (void)fprintf(file, format, parameter_1, parameter_2, ...) ;
```

โดยที่

count จำนวนตัวอักษณที่ถูกส่งมา หรือ -1 ในกรณีที่เกิด Error ขึ้น

format String ที่ใช้สำหรับกำหนดรูปแบบในการแสดงผล

parameter_1, parameter_2, ... ค่าของตัวแปรที่ต้องการแสดงผล

ตัวอย่าง เช่น

```
(void)fprintf(out_file, "result = %d\n", int_variable) ;
```

ให้สังเกตว่า *printf* จะมีค่าเท่ากับ *fprintf* ที่มี argument ตัวแรกสุดเป็น *stdout*

sprintf ฟังก์ชัน*sprintf* จะคล้ายกับ *fprintf* ต่างกันที่ argument แรกจะเป็น String หรือกล่าวโดยสรุปคือ เป็นฟังก์ชันสำหรับแปลงข้อความที่ต้องการแสดง แล้วเก็บในรูปของ String รูปแบบการใช้เป็นดังนี้

```
(void)sprintf(string, format, parameter_1, parameter_2, ...) ;
```

fscanf *fscanf* เป็นฟังก์ชันที่ใช้สำหรับการอ่านข้อมูลจากไฟล์ มีลักษณะคล้ายกับ *scanf* ต่างกันที่ว่า *scanf* จะอ่านข้อมูลจาก STANDARD INPUT (stdin) รูปแบบการใช้เป็นดังนี้

```
number = fscanf(file, format, &parameter_1, &parameter_2, ...) ;
```

หรือ

```
(void)fscanf(file, format, &parameter_1, &parameter_2, ...) ;
```

โดยที่

number หมายถึงจำนวนข้อมูล (Parameter) ที่สามารถอ่านเข้ามาได้

file หมายถึงชื่อไฟล์ที่จะทำการอ่าน

format String ที่กำหนดรูปแบบในการอ่าน

parameter_1, parameter_2 หมายถึง ข้อมูลที่จะทำการอ่านเข้ามา จะต้องกำหนดโดยใช้ Pointer

ตัวอย่าง เช่น

```
(void)fscanf(in_file, "%d %d", &a, &b) ;
```

sscanf *sscanf* คล้ายกับ *fscanf* ต่างกันตรงที่ว่าแทนที่จะอ่านจากไฟล์ จะอ่านจาก String แทน โดยมีรูปแบบการใช้ดังนี้

```
char   line[100] ;
int    a, b ;
       ...
(void)sscanf(line, "%d %d", &a, &b) ;
```

ตัวอย่างที่ 9.3 ตัวอย่างการใช้ฟังก์ชัน fscanf และ fprintf

```
#include <stdio.h>
#include <stdlib.h>
void insertion_sort(int *a, int number)        /*Insertion Sort Function*/
{
       int    i, j ;
       int    x ;
       for ( i = 1 ; i < number ; i++ ) {
              x = *(a + i) ;
              for ( j = i ; j > 0 ; j-- ) {
                     if ( x < *(a + j - 1) )
                            *(a + j) = *(a + j - 1) ;
                     else
                            break ;
              }
              *(a + j) = x ;
       }
}
int main(int argc, char **argv)         /* Use command-line argument */
{
       FILE   *in_file ;
       FILE   *out_file ;
       int    *score ;
       int    i, n, number ;

       if ( argc != 3 ) {
              printf("Usage : command input-filename output-filename\n" ) ;
              exit(1) ;
       }

       in_file = fopen(*(argv + 1), "r") ;
       if ( in_file == NULL ) {
              printf("Cannot open %s \n", *(argv + 1)) ;
              exit(1) ;
       }
       out_file = fopen(*(argv + 2), "w") ;
       if ( out_file == NULL ) {
              printf("Cannot create %s \n", *(argv + 2)) ;
              exit(1) ;
       }
       score = (int *)malloc(100 * sizeof(int)) ;
       if ( score == NULL ) {
              printf("Not enough memory\n") ;
              exit (1) ;
       }
       i = 0 ;
       while(1) {
```

```
            n = fscanf(in_file, "%d", score + i) ;
            if ( n < 1 )
                  break ;
            i++ ;
      }
      number = i ;
      printf("number of data is %d\n", number) ;
      insertion_sort(score, number) ;
      for ( i = 0 ; i < number ; i++ )
            fprintf(out_file, "%d\n", *(score + i)) ;
      fclose(in_file) ;
      fclose(out_file) ;
      return 0;
}
```

ผลการรันโปรแกรม

```
leibniz:~/program$ cat score.txt
77
82
53
25
64
leibniz:~/program$ cat result.txt
cat: cannot open result.txt
leibniz:~/program$ a.out score.txt result.txt
number of data is 5
leibniz:~/program$ cat result.txt
25
53
64
77
82
```

9.4 Unformatted Input/Output Function

ในภาษา C นอกจากเราสามารถจะจัดการกับไฟล์ที่เป็น Text File ซึ่งหมายถึง ไฟล์ที่มีข้อมูลเป็นตัวอักษร ที่ใช้รหัสตามมาตรฐาน ASCII หรือ เรียกว่า ASCII Text File เรายังสามารถใช้ไฟล์แบบ Binary หรือไฟล์ที่มีข้อมูลเป็นเลขฐานสองได้ ซึ่งไฟล์แบบนี้ จะมีประโยชน์มากในกรณีที่ต้องการเก็บข้อมูลมาก ๆ เนื่องจากการอ่านข้อมูลจากไฟล์ และการเขียนข้อมูลลงในไฟล์ จะสามารถทำได้เร็วขึ้น และง่ายขึ้น เพราะไม่มีความจำเป็นต้องทำการแปลงจากตัวอักษรเป็นค่าตัวเลข หรือแปลงตัวเลขเป็นตัวอักษร ซึ่งเป็นการเพิ่มประสิทธิภาพของโปรแกรม ฟังก์ชันที่สำหรับใช้จัดการกับการอ่านหรือเขียนที่ใช้ไฟล์แบบ Binary จะมีชื่อเรียกว่า ฟังก์ชัน Unformatted Input/Output หรือฟังก์ชัน Binary Input/Output เนื่องจากไม่สามารถทำการกำหนดรูปแบบการอ่านหรือเขียนได้ อนึ่ง ในกรณีที่ข้อมูลเป็นตัวอักษร การใช้ไฟล์แบบ Text File หรือ Binary File จะให้ผลที่ไม่แตกต่างกัน

ในภาษา C จะมีฟังก์ชันที่ใช้เกี่ยวกับ Binary Input/Output อยู่ 2 ฟังก์ชันคือ *fread* และ *fwrite*

fread

fread เป็นฟังก์ชันสำหรับอ่านข้อมูลจากไฟล์ที่เป็นแบบ Binary หรือที่เก็บข้อมูลเป็นเลขฐานสอง โดยมีรูปแบบดังนี้

```
size_t fread( void* storage, size_t size,
              size_t count, FILE* file_pointer )
```

โดย size_t จะเป็นชนิดข้อมูลที่ถูกกำหนดไว้ในไฟล์ stdlib.h ให้เป็นชนิด integer ฟังก์ชัน fread จะทำการอ่านข้อมูลที่มีขนาด size ไบต์ จำนวน count ตัว จากไฟล์ที่ file_pointer ชี้อยู่ แล้วทำการเก็บลงใน storage ในกรณีที่จำนวนข้อมูลมีไม่ถึง count ตัว (เช่น อาจจะสิ้นสุดไฟล์ก่อน) ฟังก์ชันจะอ่านเฉพาะข้อมูลที่มีอยู่ ค่าที่ฟังก์ชัน fread คืนจะเป็นจำนวนข้อมูลที่อ่านได้ *ไม่ใช่จำนวนไบต์ที่อ่านได้* ตัวอย่างการใช้ฟังก์ชัน fread ได้แก่

```
int    ss[100] ;
...
fread(ss, sizeof(int), 10, fp) ;
```

```
/* read 10 integer number from file fp and store in ss */
```

fwrite

ฟังก์ชัน fwrite เป็นฟังก์ชันสำหรับเขียนข้อมูลลงในไฟล์ในลักษณะเลขฐานสอง กล่าวคือ ไม่มีการแปลงเป็นตัวอักษร โดยมีรูปแบบดังนี้

```
size_t fwrite( const void* storage, size_t size,
               size_t count, FILE* file_pointer )
```

fwrite จะทำการเขียนข้อมูลที่อยู่ที่ตำแหน่ง storage ที่มีขนาด size ไบต์ และมีจำนวน count ตัว ลงในไฟล์ที่ file_pointer ชี้อยู่ แล้วทำการคืนจำนวนข้อมูลที่เขียนลงไปในไฟล์ ตัวอย่างการใช้ฟังก์ชัน fwrite ได้แก่

```
int   buf[100] ;
fwrite( buf, sizeof(int), 10, fp ) ;
/* write 10 integer number of buf to fp */
```

ตัวอย่างที่ 9.4 โปรแกรมสำหรับอ่านข้อมูลจากไฟล์แบบ Text แล้วเก็บในรูปของไฟล์แบบ Binary

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define FILE_NAME "input.txt"
#define OUT_NAME "output.bin"
int main()
{
  FILE  *in_file ;
  FILE  *out_file ;
  char  str[100], *str_ptr ;
  int   count ;
  in_file = fopen(FILE_NAME, "r") ;
  out_file = fopen(OUT_NAME, "wb") ;
  if ( in_file == NULL || out_file == NULL ) {
    printf("open file error \n") ;
    exit(1) ;
  }
  while(1) {
    str_ptr = fgets(str, 100, in_file) ;
    if (str_ptr == NULL)
      break ;
    else {
      count = strlen( str ) ;
      fwrite(str, 1, count, out_file) ;
    }
  }
  fclose(in_file) ;
  fclose(out_file) ;
  return 0;
}
```

ตัวอย่างที่ 9.5 ตัวอย่างการใช้ fwrite

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv)
{
  FILE  *in_file ;
  FILE  *out_file ;
  int   *int_ptr, i, n ;

  out_file = fopen(*(argv+1), "wb") ;
  if ( in_file == NULL || out_file == NULL ) {
    printf("open file error \n") ;
    exit(1) ;
  }

  int_ptr = (int *)malloc( 1000 * sizeof(int) ) ;
  n = atoi(*(argv + 2)) ;

  for ( i = 0 ; i < n ; i++ )
    *(int_ptr + i) = i ;
```

```
  fwrite( int_ptr, sizeof(int), n, out_file ) ;
  fclose(out_file) ;
  return 0;
}
```

ตัวอย่างที่ 9.6 ตัวอย่างการใช้ fread ในการอ่านข้อมูลจากไฟล์ที่บันทึกแบบ Binary (ใช้ไฟล์เอาต์พุตจากตัวอย่างที่แล้ว)

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

int main(int argc, char **argv)
{
  FILE  *in_file, *out_file ;
  int   *int_ptr, n, i ;

  if ( argc != 4 ) {
    printf( "Usage : command infile outfile number\n" ) ;
    exit(1) ;
  }
  in_file = fopen(*(argv+1), "rb") ;
  out_file = fopen(*(argv+2), "w") ;
  if ( out_file == NULL || in_file == NULL ) {
    printf("open file error \n") ;
    exit(1) ;
  }

  int_ptr = (int *)malloc( 1000 * sizeof(int) ) ;
  n = atoi(*(argv + 3)) ;

  fread( int_ptr, sizeof(int), n, in_file ) ;

  for ( i = 0 ; i < n ; i++ )
    fprintf( out_file, "%d ", *(int_ptr + i)) ;

  fclose(in_file) ;
  fclose(out_file) ;
  return 0;
}
```

แบบฝึกหัด

1. จงบอกข้อผิดพลาด

```
FILE file_ptr ;
file_ptr = fopen(“test.dat”, “a”) ;
```

2. จงบอกข้อผิดพลาด

```
FILE  *f_ptr ;
f_ptr = fopen(“test2.txt”, “r”) ;
...
fclose(*f_ptr) ;
```

3. จงบอกข้อผิดพลาด

```
#include <stdio.h>
void main()
{
      FILE  *fp ;
      int   c ;
      fp = fopen(“out.dat”, “r”) ;
      while (( c = getchar() ) != EOF )
            fputc( fp, c ) ;
      fclose( “out.dat” ) ;
}
```

4. เขียนโปรแกรมที่สั่งให้ผู้ใช้ใส่ชื่อ Source file และ Target file จากนั้น ทำการตรวจสอบว่า เป็นไฟล์เดียวกันหรือไม่ ถ้าไม่ ให้ทำการ Copy ข้อมูลจาก Source file ไปที่ Target file แล้วแสดงค่าจำนวนไบต์ของข้อมูลที่ทำการ Copy

5. แก้ไขโปรแกรมในข้อ 4 ให้สามารถรับข้อมูลจากผู้ใช้ไปใส่ไว้ในส่วนหัวของ Target file ได้ ในกรณีที่ผู้ใช้ต้องการเพิ่ม Comment เข้าไปในไฟล์
6. แก้ไขโปรแกรมในข้อ 4 ให้แทรกเลขบรรทัดลงไปใน Column แรกของทุกบรรทัด แล้วเว้นช่องว่างระหว่างเลขบรรทัดกับข้อมูลเป็นระยะ 1 TAB
7. เขียนโปรแกรมสำหรับรวมไฟล์แบบ Text 2 ไฟล์เข้าด้วยกัน
8. เขียนโปรแกรมสำหรับแบ่งไฟล์แบบ Text ออกเป็น 2 ไฟล์ที่มีจำนวนตัวอักษรเท่า ๆ กัน
9. เขียนโปรแกรมสำหรับอ่านข้อมูลระยะทางที่มีหน่วยเป็นไมล์จากไฟล์ แล้วทำการแปลงให้เป็นกิโลเมตร แล้วบันทึกลงในไฟล์อีกไฟล์หนึ่ง
10. เขียนโปรแกรมสำหรับทำการเปรียบเทียบข้อมูลในไฟล์ 2 ไฟล์ ในลักษณะบรรทัดต่อบรรทัด และให้บันทึกบรรทัดที่แตกต่างกัน โดยแสดงเลขบรรทัด และข้อความในบรรทัดนั้นของทั้ง 2 ไฟล์ ลงในไฟล์เอาต์พุต
11. จงออกแบบไฟล์สำหรับเก็บข้อมูลเกี่ยวกับผลการเรียน แล้วเขียนโปรแกรมสำหรับจัดเก็บ แก้ไข และแสดงข้อมูลที่เก็บไว้
12. เขียนโปรแกรมสำหรับอ่านไฟล์ แล้วทำการเปลี่ยนอักษรตัวเล็กในไฟล์นั้นให้เป็นอักษรตัวใหญ่ให้หมด
13. ออกแบบไฟล์สำหรับจัดเก็บประวัติบุคคลต่าง ๆ อันประกอบด้วย ชื่อ ที่อยู่ เบอร์โทรศัพท์ วันเกิด เป็นต้น แล้วเขียนโปรแกรมสำหรับจัดเก็บข้อมูล และแสดงผล