FILE OPERATIONS

030523300- Computer Programming
Asst. Prof. Dr. Choopan Rattanapoka

File และ การใช้งาน

- ชนิดของ File ในภาษา C แบ่งเป็น
  - **Text file** เป็นแฟ้มข้อมูลที่ใช้ในการเก็บตัวขระปกติ มีการสิ้นสุดของบรรทัดที่แน่นอน ซึ่งก็คือ ‘\n’ ในภาษา C สามารถอ่าน หรือเขียนแฟ้มข้อมูลที่ละบรรทัดได้
  - **Binary file** เป็นแฟ้มข้อมูลที่ใช้เก็บเลขฐาน 2 ต่อเนื่องกัน จะไม่มีขอบเขตของบรรทัด เวลาใช้งานจึงต้องกำหนดขนาดในการ อ่าน หรือเขียน เป็นจำนวนไบต์ (byte)

การกำหนดชื่อของแฟ้มข้อมูล

- DOS และ Windows 3.X
  - ชื่อของแฟ้มข้อมูลจะมีความยาวสูงสุดได้เพียง 8 ตัวอักษร ตามด้วย ‘.’ และนามสกุลอีก 3 ตัวอักษร
- Windows 95 หรือ สูงกว่า
  - ชื่อของแฟ้มข้อมูลจะมีความยาวสูงสุดได้ 256 ตัวอักษร
- ตัวอักษรที่ห้ามใช้ในการตั้งชื่อแฟ้มข้อมูล
  - / \ : * ? “ < > |
- ตำแหน่งที่อยู่ของแฟ้มข้อมูล เรียกว่า **Path** เช่น
  - C:\myprogram\mysubdir\readme.txt

คำสั่งจัดการแฟ้มข้อมูลในภาษา C

- กลุ่มคำสั่งที่ใช้ในการจัดการเกี่ยวกับแฟ้มข้อมูล แบ่งออกเป็น 4 ประเภทใหญ่ๆคือ
  - **เปิด**แฟ้มข้อมูล
  - **ปิด**แฟ้มข้อมูล
  - **อ่าน**แฟ้มข้อมูล
  - **เขียน**แฟ้มข้อมูล

การเปิดแฟ้มข้อมูล

- ทุกครั้งที่จะทำงานต่างๆ เกี่ยวกับแฟ้มข้อมูล จำเป็นจะต้องทำการเปิดแฟ้มข้อมูลนั้นขึ้นมาก่อน
- เงื่อนไขในการเปิดแฟ้มข้อมูล
  - ถ้าแฟ้มข้อมูลที่ถูกเปิด **ไม่มีอยู่ในระบบ** การเปิดแฟ้มข้อมูลในภาษา C จะเป็นการ **สร้างแฟ้มข้อมูลนั้นขึ้นมาใหม่**
  - ถ้าแฟ้มข้อมูลที่ถูกเปิด **มีอยู่ในระบบ** การเปิดแฟ้มข้อมูลในภาษา C จะเป็นการ **แก้ไขแฟ้มข้อมูลนั้น**
- คำสั่ง (function) ที่ใช้ในการเปิดแฟ้มข้อมูลในภาษา C คือ

```
FILE  *fopen(const char *filename , const char *mode);
```

การใช้งาน fopen

```
FILE  *fopen(const char *filename , const char *mode);
```

- **Return value**
  - คืนค่า **NULL** ถ้าเปิดแฟ้มข้อมูลไม่สำเร็จ
  - คืนค่า **pointer** ที่ชี้ไปยังแฟ้มข้อมูลนั้น ถ้าเปิดแฟ้มข้อมูลสำเร็จ

- **Parameters**
  - **const char *filename** ชื่อแฟ้มข้อมูลที่ต้องการจะเปิด เช่น **“C:\\Program Files\\Hello.txt”**
  - **const char *mode** โหมดที่จะทำงานกับแฟ้มข้อมูลนี้

โหมดการทำงานในการเปิดแฟ้มข้อมูล

- ใน const char *mode สามารถระบุโหมดการทำงานดังนี้
  - **r** เป็นการเปิดแฟ้มข้อมูลเพื่ออ่าน (read) ถ้าแฟ้มข้อมูลนั้นไม่มีอยู่ในระบบ fopen จะคืนค่า NULL
  - **w** เป็นการเปิดแฟ้มข้อมูลเพื่อเขียน (write) ถ้าแฟ้มข้อมูลนั้นไม่มีอยู่ในระบบ จะเป็นการสร้างแฟ้มข้อมูลใหม่ แต่ถ้ามีแฟ้มข้อมูลอยู่ในระบบ จะเป็นเขียนทับแฟ้มข้อมูลที่เปิด
  - **a** เป็นการเปิดแฟ้มข้อมูลเพื่อเขียนต่อ (append) ถ้าแฟ้มข้อมูลไม่มีอยู่ในระบบจะเป็นการสร้างแฟ้มข้อมูลใหม่ แต่ถ้ามีแฟ้มข้อมูลอยู่ในระบบจะเป็นการเขียนต่อท้าย
  - **r+** หรือ **w+** เป็นการเปิดแฟ้มข้อมูลเพื่ออ่านและเขียน ถ้าแฟ้มข้อมูลไม่มีอยู่จะสร้างแฟ้มข้อมูลใหม่ แต่ถ้ามีแฟ้มข้อมูลอยู่แล้วการเขียนจะเป็นการเขียนทับ
  - **a+** เป็นการเปิดแฟ้มข้อมูลเพื่ออ่านและเขียน ถ้าแฟ้มข้อมูลไม่มีอยู่จะสร้างแฟ้มข้อมูลใหม่ แต่ถ้ามีแฟ้มข้อมูลอยู่แล้วการเขียนจะเป็นการเขียนต่อท้าย

โหมดการทำงานในการเปิดแฟ้มข้อมูล (ต่อ)

- ใน const char *mode สามารถระบุโหมดการทำงานดังนี้
  - **t** เป็นการเปิดแฟ้มข้อมูลประเภท **text file** ในกรณีที่ไม่ระบุภาษา C จะเป็นการทำงานเป็นโหมด **text file**
  - **b** เป็นการเปิดแฟ้มข้อมูลประเภท **binary file**

- **ตัวอย่าง** เปิดแฟ้มข้อมูล **c:\tmp\hello.txt** ที่เป็นแฟ้มข้อมูลแบบ **text** เพื่ออ่านและเขียนแบบไม่ทับข้อมูลเก่าในแฟ้มข้อมูล
  - **fopen("c:\\tmp\\hello.txt", "a+t");**

การปิดแฟ้มข้อมูล

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

- **Parameter**
  - เอา pointer ของ FILE ที่ได้จากการเปิดแฟ้มข้อมูลมาส่งเพื่อใช้ปิด
- **Return value**
  - ถ้าการปิดแฟ้มข้อมูลเรียบร้อย คำสั่ง fclose จะคืนค่าเป็น 0
  - ถ้ามีข้อผิดพลาดระหว่างปิดแฟ้มข้อมูล fclose จะคืนค่าเป็น EOF

ตัวอย่างการเปิด/ปิด แฟ้มข้อมูล

```
#include <stdio.h>
int main(int argc, char **argv) {
    FILE *fp;
    if((fp = fopen("test.txt", "w")) == NULL)  {
            printf("Can not open file\n");  return 1;
    }
    if(fclose(fp) != 0)  {
            printf("Error while closing file\n");
            return 1;
    }
    system("PAUSE");
    return 0;
}
```

การเขียนแฟ้มข้อมูลประเภท Text

- คำสั่งในการเขียนแฟ้มข้อมูลประเภท **text**

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

- การใช้งานคล้ายกับคำสั่ง **printf** เพียงแต่เพิ่ม parameter เข้ามา 1 ตัวคือ FILE *fp
- **<u>การคืนค่า</u>**
  - ถ้าการเขียนสำเร็จจะคืนค่าเป็น**จำนวนเต็มบวก** เท่ากับ จำนวนไบต์ที่เขียนลงแฟ้มข้อมูล
  - ถ้ามีข้อผิดพลาดระหว่างการเขียนข้อมูล **จำนวนเต็มลบ** จะถูกคืนค่ากลับ

ตัวอย่างการเขียนแฟ้มข้อมูลประเภท Text

```
#include <stdio.h>
int main(int argc, char **argv) {
    FILE *fp;
    int I = 25;  char grade = 'A';
    if((fp = fopen("test.txt", "w")) == NULL) {
            printf("Can not open file");  return 1;
    }
    if( fprintf(fp, "I = %d , grade = %c\n", I, grade) < 0) {
            printf("Can not write file");  return 1;
    }
    printf("work completed\n");
    fclose(fp);
    system("PAUSE");
    return 0;
}
```

test.txt

I = 25, grade = A

การอ่านแฟ้มข้อมูลประเภท Text

- คำสั่งในการอ่านแฟ้มข้อมูลประเภท **text** ทีละบรรทัด

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

- การใช้งานคล้ายกับคำสั่ง **scanf** เพียงแต่เพิ่ม parameter เข้ามา 1 ตัวคือ FILE *fp
- **การคืนค่า**
  - EOF ถ้าการอ่านถึงจุดสิ้นสุดของแฟ้มข้อมูล หรือ มีข้อผิดพลาดระหว่างการอ่าน

ตัวอย่างการอ่านแฟ้มข้อมูลประเภท Text

```
#include <stdio.h>
int main(int argc, char **argv) {
    FILE *fp;
    int   x, y;
    fp = fopen("test.txt", "r");
    fscanf(fp, "%d %d", &x, &y);
    fclose(fp);
    printf("X = %d,  Y = %d\n", x, y);
    system("PAUSE");
    return 0;
}
```

test.txt

```
10   5
11   20
```

Monitor

```
X = 10,  Y = 5
```

ตัวอย่างการอ่านแฟ้มข้อมูลประเภท Text (ต่อ)

```
#include <stdio.h>
int main(int argc, char **argv) {
    FILE *fp;
    int   x, y;
    fp = fopen("test.txt", "r");
    while(fscanf(fp, "%d %d", &x, &y) != EOF) {
        printf("X = %d,  Y = %d\n", x, y);
    }
    fclose(fp);
    system("PAUSE");
    return 0;
}
```

test.txt

```
10   5
11   20
```

Monitor

```
X = 10,  Y = 5
X = 11,  Y = 20
```

การอ่าน/เขียนแฟ้มข้อมูลประเภท **Text** แบบตัวอักษร

- คำสั่งในการเขียนแฟ้มข้อมูลประเภท **text** แบบตัวอักษรอย่างเดียว
  - int **fputc**(int c, FILE *fp);
  - int **fputs**(const char *s, FILE *fp);
- คำสั่งในการอ่านแฟ้มข้อมูลประเภท **text** แบบตัวอักษรอย่างเดียว
  - int **fgetc**(FILE *fp); -> EOF เมื่อสุดแฟ้ม
  - char ***fgets**(char *s, int size, FILE *fp); -> NULL เมื่อสุดแฟ้ม

ตัวอย่างการใช้ fputc

```
#include<stdio.h>

int main(int argc, char **argv)
{
        char *ch = "Control Systems";
        FILE *fp;

        if( (fp = fopen("c:\\test2.txt","w"))==NULL ){
                printf("error");        return 1;
        }
        while(*ch != NULL){
                fputc(*(ch++),fp);
        }

        fclose(fp);
        system(“PAUSE”);
        return 0;
}
```

ตัวอย่างการใช้ fputs

```
include<stdio.h>

int  main(int argc, char **argv)
{
   int  i;  char mytext[3][64] = {"Control Systems","Digital","Computer"};
   FILE *fp;

   if((fp = fopen("c:\\test3.txt","w"))==NULL ) {
        printf("error"); return 1;
   }
   for(i=0;i<3;i++){    fputs(mytext[i], fp);  }
   fclose(fp);
   system("PAUSE");
   return 0;
}
```

ตัวอย่างการใช้ fgetc

```
#include<stdio.h>

int  main(int argc,  char **argv)
{
        char  k;
        FILE *fp;

        if( (fp = fopen("c:\\test.txt","r"))==NULL ) {
                printf("error");     return 1;
        }
        while((k = fgetc(fp)) != EOF) {
                printf("%c", k);
         }

        fclose(fp);
        system(“pause”);
        return 0;
}
```

ตัวอย่างการใช้ fgets

```
#include<stdio.h>

int main(int argc, char **argv)
{
        char str[64];
        FILE *fp;

        if((fp = fopen("c:\\str_test.txt","r"))==NULL ){
                printf("error"); return 1 ;
        }
        while(fgets(str,64,fp) != NULL) {
                printf("%s", str);
        }
        fclose(fp);
        system("PAUSE");
        return 0;
}
```

การทำงานกับแฟ้มข้อมูลแบบ Binary

- คำสั่งที่ใช้ในการอ่านแฟ้มข้อมูลแบบ Binary

```
int   fread(void *buf,  int size,  int count, FILE *fp);
```

- คำสั่งที่ใช้ในการเขียนแฟ้มข้อมูลแบบ Binary

```
int   fwrite(void *buf, int size, int count,  FILE *fp);
```

- **Parameter**
  - **buf** ข้อมูล Buffer ที่จะใช้เขียนลงแฟ้มข้อมูล(fwrite) หรือ อ่านจากแฟ้มข้อมูลมาเก็บไว้(fread)
  - **size** ขนาดของประเภทข้อมูล (sizeof)
  - **count** จำนวนข้อมูลที่จะเขียน/อ่าน ลง แฟ้มข้อมูล

การจัดการเกี่ยวกับ file pointer

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

- ใช้เลื่อนตำแหน่ง **file pointer** ไปตำแหน่งต่าง ๆในแฟ้มข้อมูล
- **Parameter**
  - **offset** เป็นค่าที่นำไป บวก กับ ค่าของ **where** เพื่อกำหนดตำแหน่งของ **fp**
  - **where** เป็นจุดอ้างอิงของ **fp**
    - SEEK_SET ใช้แทนตำแหน่งเริ่มต้นของแฟ้มข้อมูล
    - SEEK_CUR ใช้แทนตำแหน่งปัจจุบันที่ **fp** ชี้อยู่
    - SEEK_END ใช้แทนตำแหน่งท้ายสุดของแฟ้มข้อมูล

```
void   rewind(FILE *fp);
```

- ใช้เลื่อนตำแหน่ง **file pointer** กลับไปที่จุดเริ่มต้นของแฟ้มข้อมูล

การทำงานของ fseek

monitor

A
G

```
#include <stdio.h>
int main(int argc, char **argv) {
        FILE *fp;
        if((fp = fopen("test.txt", "rb")) != NULL) {
                printf("%c\n", fgetc(fp));
                fseek(fp, 5, SEEK_CUR);
                printf("%c\n", fgetc(fp));
                fclose(fp);
        }
        system("PAUSE"); return 0;
}
```

แบบฝึกหัด fseek

```
#include<stdio.h>
int main(int argc, char **argv)
{
      FILE *fp;
      fp = fopen("c:\\kk.txt","rb");

      fseek(fp, 8, SEEK_SET);      printf("%c\n",fgetc(fp));
      fseek(fp,-3, SEEK_END);     printf("%c\n",fgetc(fp));
      fseek(fp,-3, SEEK_CUR);     printf("%c\n",fgetc(fp));
      fseek(fp, 3,SEEK_CUR);      printf("%c\n",fgetc(fp));

      fclose(fp); getch();
}
```

kk.txt

123456789