Робота з файлами

Для довгострокового збереження великих об'ємів даних використовуються ***файли***.

Більшість комп'ютерних програм працює з файлами.

Текстові процесори створюють файли документів.

Програми баз даних створюють файли й роблять у них пошук інформації.

Компілятори зчитують файли з вихідним кодом і генерують файли, що виконуються.

Сам по собі **файл** - *це набір байтів, що зберігаються на деякому пристрої*

Операційна система *управляє* файлами, стежачи за їхнім місцем розташування, розміром, датою створення тощо.

Якщо тільки ви не програмуєте на рівні операційної системи, про ці питання можна не турбуватися.

Усе, що потрібно, - **задати спосіб зв'язку програми з файлом**, а також **методи**, які використовуються програмою при читанні вмісту файлу, запису у файл, а також при створенні нового файлу й запису в нього.

Файл – це поіменована сукупність даних, яка зберігається на пристрої.

Будь-який файл являє собою послідовність байтів.

В залежності від того, який сенс та призначення мають ці байти, дані поділяють на **текстові** та **бінарні** (двійкові).

Текстовий файл

Текстовий файл - текстовий потік даних, фактично рядок, що має ім’я та зберігається на пристрої.

Текстові файли містять байти, що є кодами алфавітних, цифрових символів та знаків пунктуації (пробілів, табуляцій та символів переходу на новий рядок).

Бінарний файл

Бінарний файл – це сукупність будь якого типу даних, що так само знаходиться в пристрої.

Бінарні файли можуть містити будь-які значення байтів, в тому числі і такі, яким не відповідає жодного графічного знаку для виведення на екран.

При цьому для того, щоб зчитувати конкретний файл, потрібно знати дані якого типу та в якому порядку записані на пристрої.

Файл – це єдиний спосіб, за допомогою якого дані, що обробляються програмою, можуть бути отримані ззовні, а результати програми передані у зовнішній світ.

Фізичний файл – послідовність байтів в зовнішній пам’яті; поіменована область зовнішньої пам’яті.

Байт	Байт	Байт	…	…	…	…	…	константа EOF (-1)

Логічний (внутрішній) файл – той, що описаний і використовується в програмі; він представляє собою покажчик на початок фізичного файла.

FILE *<ім’я файла>;

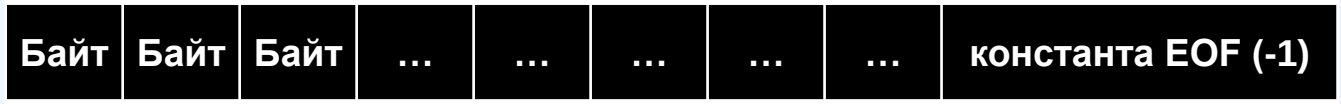
З логічної точки зору (на відміну від фізичної) файл – це група логічно пов'язаних записів. У свою чергу, **запис** – *це група логічно пов'язаних полів даних*.

Наприклад:

#include <stdio.h>

FILE *f, *g;

В мові С файли неструктуровані і нетипізовані.



Відкритий файл повертає покажчик на структуру **FILE** (визначену в **stdio.h**).
Ця структура містить інформацію, що використовується під час роботи з файлом.

Структура **FILE** включає дескриптор файла, тобто індекс у масиві операційної системи, який називається таблицею відкритих файлів.
Кожний елемент масиву містить блок управління файлом (**FCB**), який використовується операційною системою для доступу до файла.

Умовно розділяємо файли	або за типом елементів (при відкритті-закритті) або за способом доступу
За типом елементів	бінарні текстові (хоча за своєю структурою всі файли є бінарними)
За способом доступу (визначається трактовкою та способом роботи)	прямого доступу послідовного доступу

Бібліотека С підтримує три рівня вводу-виводу

- потокове введення-виведення
- потокове введення-виведення нижнього рівня
- потокове введення-виведення для консолі портів (залежить від конкретної ОС)

При вводі з диску або при зчитуванні з файла *дані розміщуються в буфері ОС*, а потім побайтно або порціями передаються програмі.

При виводі в файл дані так само накопичуються в буфері, а при заповненні буфера записуються у вигляді єдиного блока на диск.

Буфери реалізуються у вигляді ділянок основної пам'яті.

Таким чином, ***потік*** – це файл разом з наданими засобами буферизації.

Потоки вводу-виводу - це об'єкти типу FILE, до яких можна отримати доступ і маніпулювати ними лише за допомогою вказівників типу **FILE ***

Кожен потік вводу виводу пов'язаний з зовнішнім фізичним пристроєм (файл, стандартний вхідний потік, принтер, послідовний порт тощо).

Потоки вводу-виводу можуть використовуватися як для неформатованих, так і для форматованих входів і виходів.

Вони чутливі до локалі і можуть виконувати широкі(багатобайтові) перетворення, якщо це необхідно.

Всі потоки отримують доступ до одного і того ж локального об'єкта: останнього встановлено з **setlocale**.

Операції над файлами

- отримання адреси (&)
- порів-няння
- присвою-вання
- розадресація (розіменування) (*)
- приведення типів
- додавання з константою, віднімання, арифметичні операції з покажчиками: інкремент (++), декремент (--),

Функції бібліотеки С, що підтримують обмін даними на рівні потоку, дозволяють обробляти дані різних розмірів і форматів.

При роботі з потоком можна:

- Відкривати і закривати потоки
- Вводити і виводити дані або їх порції встановленої довжини
- Керувати буферизацією потоку і розміром буфера
- Отримувати і встановлювати покажчик поточної позиції

Функції для роботи з файловою системою

Назва	Що робить
`fopen()`	Відкриває файл
`fclose()`	Закриває файл
`putc()`	Записує символ у файл
`fputc()`	Те саме, що і putc()
`getc()`	Читає символ з файлу
`fgetc()`	Те саме, що і getc()
`fgets()`	Читає рядок з файлу
`fputs()`	Записує рядок у файл

Для роботи з файловою системою існує заголовочний файл **<stdio.h>**.

Назва	Що робить
`fseek()`	Встановлює покажчик поточної позиції на певний байт файлу
`ftell()`	Повертає поточне значення покажчина у файлі
`fprintf()`	Для файлу те саме, що printf() для консолі
`fscanf()`	Для файлу те саме, що scanf() для консолі
`feof()`	Повертає значення true (істина), якщо досягнуто кінець файлу
`error()`	Повертає значення true, якщо виникла помилка
`rewind()`	Встановлює покажчик поточної позиції на початок файлу
`remove()`	Знищує файл
`fflush()`	Дозапис потоку у файл

Відкриття файлу

Покажчик на потік отримує значення в результаті виконання функції відкриття файлу.

При невдалому відкритті покажчику надається значення `NULL`.

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

`const char*path` – рядок з іменем файлу, зв'язаного з потоком,

`const char*mode` – рядок режиму відкриття файлу

Наприклад:

```
FILE *f=fopen("C:\\t.txt","r");
```

Режими відкриття файла

Режим	Опис
r	Відкрити існуючий файл для читання, покажчик на початок файла
w	Відкрити файл для запису. Якщо файл непустий, то вміст губиться, якщо файл відсутній, то він створюється
a	Відкрити файл для дозапису. Якщо файл існує, то дозапис виконується в кінець
t	Відкрити файл як текстовий
b	Відкрити файл як бінарий
+	Дозволити і читання, і запис

Потік може бути відкритий в:

текстовому (**t**) режимі
За замовчанням встановлюється текстовий режим.

двійковому (**b**) режимі

В **MS DOS** и **MS Windows** є різниця в роботі з текстовими і бінарними файлами, в **Unix** різниці між текстовим і бінарним файлом немає.

В таких системах при відкритті бінарного файла в рядок **mode** потрібно додавати літеру “**b**” (binary), а при відкритті текстового – літеру “**t**” (text).

Крім того, при відкритті можна дозволити виконувати як читання, так і запис (символ **+**).

Літери “**b**” і “**t**”" можна використовувати, навіть якщо в операційній системі немає різниці між текстовим і бінарним файлами – вони будуть просто проігноровані.

Приклад

```
FILE *f, *g, *h;

f = fopen("abcd.txt", "rt");

g = fopen("c:/Windows/Temp/tmp.dat", "wb+");

h = fopen("c:\\Windows\\Temp\\abcd.log", "at");

if(f=fopen(filename,"rt")==0)
```

Відкрити текстовий файл "**abcd.txt**" для читання

Відкрити бінарний файл для читання і запису

Відкрити текстовий файл для дописування в кінець

Відкриваємо для читання і перевіряємо, чи виникає помилка при відкритті файле

Функція видалення файлу з вказаним ім’ям

Функція

int remove (const char* filename);

видаляє файл з вказаним ім’ям;

подальша спроба відкрити файл з цим ім’ям викличе помилку.

Функція повертає ненульове значення у разі невдалої спроби.

Функція зміни імені файлу

Функція

int rename (const char* oldname, const char* newname);

змінює ім’я файлу.

Повертає ненульове значення у випадку, якщо спроба змінити ім’я виявилася невдалою. Перший параметр задає старе ім’я, другий параметр — нове ім’я.

Функція створення тимчасового файлу

Функція

FILE* tmpfile (void);

створює тимчасовий файл з режимом доступу «wb+», який автоматично видаляється при його закритті або звичайному завершенні програмою своєї роботи.

Ця функція повертає потік або NULL якщо не змогла створити файл.

Функції обробки помилок роботи з файлами

Помилки, які можуть виникати при відкритті потока:

- файл, що зв'язаний з потоком, не знайдений (при читанні)
- диск заповнений (при записі);
- диск захищений від запису.

В усіх цих випадках покажчик набуває значення `NULL (0)`.

Для виводу повідомлення про помилку при відкритті потока використовується бібліотечна функція з файла `<stdio.h>`

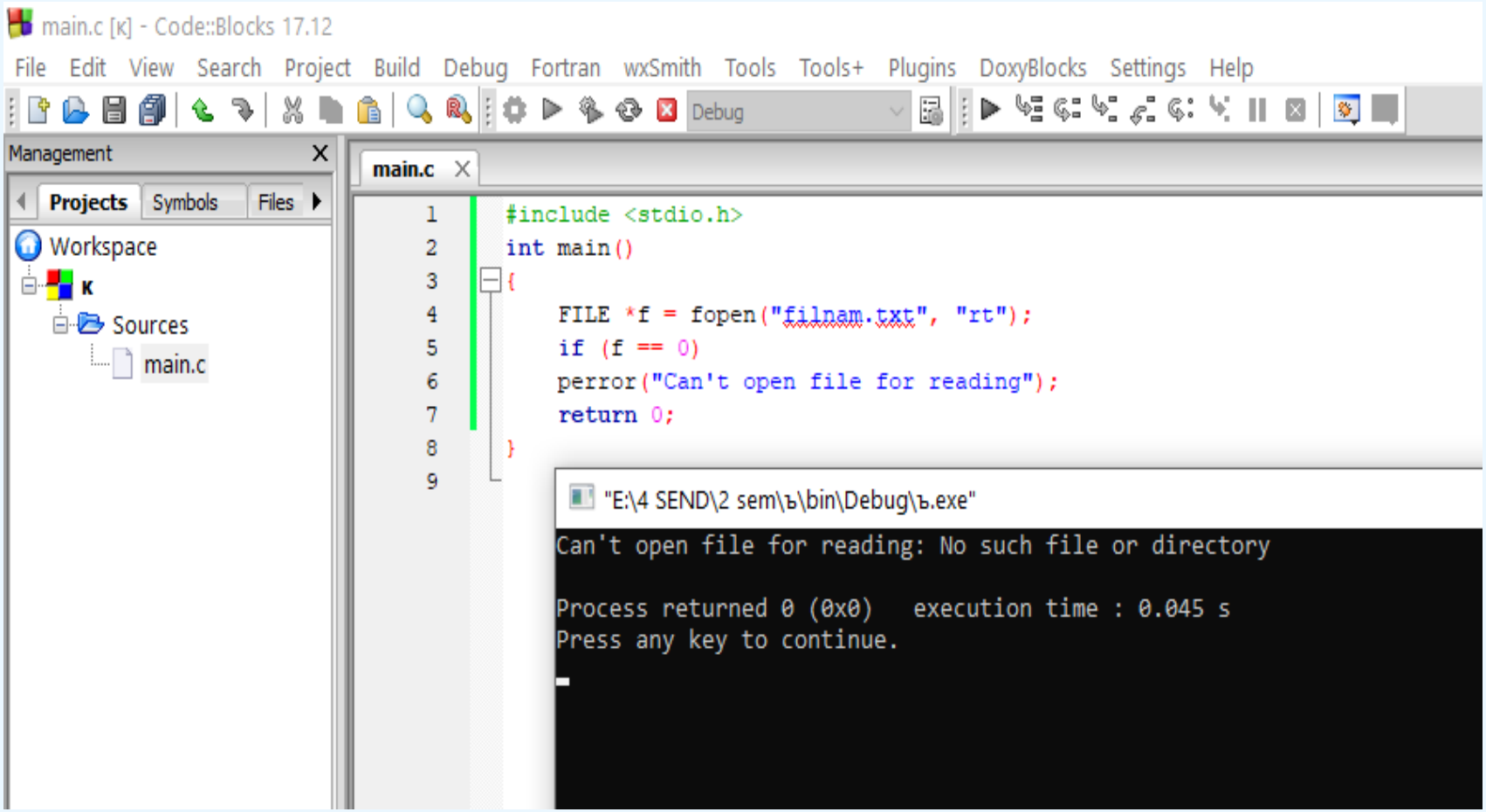
```
void perror (const char*s);
```

Текст повідомлення відповідає номеру помилки.

```
FILE *f = fopen("filnam.txt", "rt");
if (f == 0) {
    perror("Can’t open file for reading"); }
```

Функція **`perror()`** друкує спочатку користувацьке повідомлення про помилку, а після двокрапки системне повідомлення.

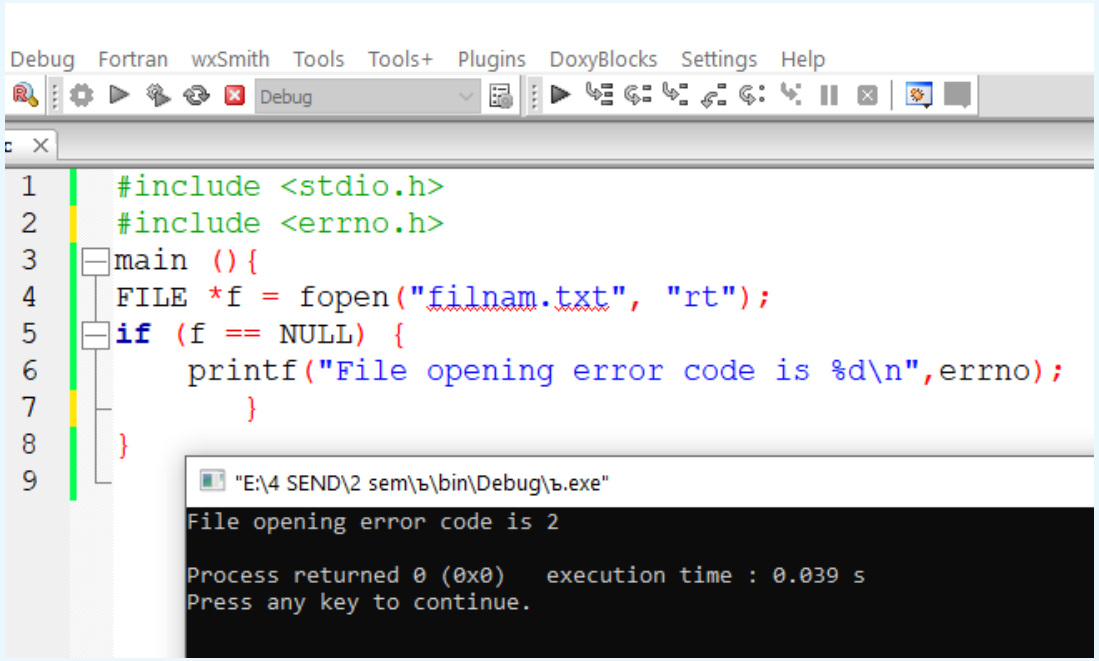
Наприклад, у випадку помилки із-за відсутності файла буде виведено:


main.c [к] - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
Management
Projects Symbols Files
Workspace
к
Sources
main.c
main.c
1 #include <stdio.h>
2 int main()
3 {
4 FILE *f = fopen("filnam.txt", "rt");
5 if (f == 0)
6 perror("Can't open file for reading");
7 return 0;
8 }
9
"E:\4 SEND\2 sem\ъ\bin\Debug\ъ.exe"
Can't open file for reading: No such file or directory
Process returned 0 (0x0) execution time : 0.045 s
Press any key to continue.

Номер помилки заноситься в змінну

`int errno`

(визначена в заголовочному файлі `errno.h`).



Закриття файлу

`int fclose(FILE *fp);`

Закриває файл.

Повертає нуль у випадку успіху і `EOF` у випадку невдачі.

```
FILE *f;
if (fclose(f) < 0){// Вивести повідомлення про помилку
        error("Closing error");
//printf("Closing error");
            }
```

Кінець файлу

В `<stdio.h>` визначена константа `EOF`, яка позначає кінець файла (від'ємне ціле число -1). Функція

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

повертає ненульове значення, якщо в результаті виконання останньої операції читання з потоку досягнуто кінця файла.

Розв’язок вигляду

```
while (!feof(f))  {
fread(&x,sizeof(x),1,f); s+=x; }
```

дасть невірну відповідь, тому що тут неправильно перевіряється кінець файла. Флаг EOF встановлюється не тоді, коли читається останній байт з файла, а після того, як відбулася спроба прочитати за межами файла. Тобто ви читаєте останній елемент, EOF==false, далі пробуєте прочитати за межами файла, EOF==true, читається невідомо що, заноситься в змінні, і лише тоді цикл переривається.

Щоб позбутися цього, потрібно перевіряти* EOF *одразу після читання і до тієї дії, яку потрібно виконати над зчитаними значеннями.

Мова С розглядає будь-який файл як ***послідовний потік байтів***.

Кожний файл закінчується ***маркером кінця*** файлу.

Коли файл ***відкривається***, йому ставиться у відповідність потік.

Потоки забезпечують канали передачі даних між файлами і програмами.

На початку виконання програми автоматично відкриваються три файли і пов'язані з ним потоки:

стандартне введення,

стандартне виведення,

стандартна помилка.

Стандартний потік введення дозволяє програмі зчитувати дані з клавіатури. Для роботи зі стандартним потоком уведення використовується покажчик файла ***stdin***.

Стандартний потік виведення дозволяє виводити дані на екран дисплея. Для роботи зі стандартним потоком виведення використовується покажчик файла ***stdout***.

Для роботи зі стандартним потоком помилок використовується покажчик файла ***stderr***.

Існує багато способів організації записів у файлі.

Найбільш розповсюджений називається послідовним файлом.

У ньому записи зберігаються за порядком, визначеним деяким полем, яке, у свою чергу, називається ключовим полем.

Створення файла

Для кожного файла, який використовується в програмі на мові С, мінімально необхідно виконати три дії:

- Крок 1 описати файл
- Крок 2 відкрити файл
- Крок 3 закрити файл

Крок 1

Кожний файл описується як покажчик на структуру **FILE**. Наприклад,

FILE *cFptr;

об'являє, що **cFptr** є покажчиком на структуру **FILE**.

Крок 2

Для відкриття файла і присвоєння значення змінній **cFptr** можна скористатися функцією ***fopen***.

cFptr = fopen(“myfile.dat”,”w”);

дозволяє відкрити файл під назвою **myfile.dat,** зв'язати його з покажчиком **cFptr** та присвоїти цьому покажчику значення покажчика на структуру **FILE**.

Можна додати перевірку:

if (cFptr = fopen(“myfile.dat”,”w”)) == NULL)

printf (“Файл не можна відкрити!”);

Другий аргумент функції **fopen** – **“w”** – вказує на режим відкриття файла, а саме на те, що файл повинен відкритися для запису.

Якщо файла не існує, він буде створений.

Якщо файл уже існує, в режимі “**w**” вміст файла буде знищений.

Структура **if** у програмі використовується для того, щоби визначити значення покажчика **cFptr**.

Якщо воно дорівнює **NULL** (відкриття файла не відбулося), друкується повідомлення про помилку.

Крок 3

Для закриття файла використовується функція ***fclose***.

fclose(cFptr);

закриває файл, який пов'язаний з покажчиком **cFptr** (для нашого прикладу - файл **myfile.dat**).

Зрозуміло, що, крім мінімально необхідних трьох операцій з файлом, програма виконує ще певні дії з ним.

Якщо файл відкритий для запису, у програмі природно існування операції запису у файл.

Це можна зробити за допомогою функції ***fprintf***.

Функція fprintf()

`int fprintf(FILE*fp, Формат, СписокЗмінних),`

де

`FILE*f` – покажчик на файл, в який виконується запис,

`const char*fmt` – форматний рядок,

`СписокЗмінних` - список змінних, що записуються в файл.

Виконує форматований ввід (аналогічно printf()) в файл, зв'язаний з потоком, указаним як перший параметр.

Повертає число записаних символів.

Файл, зв'язаний з потоком, мусить бути відкритий в режимі, що допускає запис (див. fopen()). Заголовочний файл: <stdio.h>

Приклад програми створення послідовного файла.


```
#include <stdio.h>
#include <conio.h>
int main (void) {
      float X, Y;
      FILE *fileX;
      clrscr();
      if (( fileX = fopen("fileX.txt","w")) == NULL)
            printf ("Error!");
      else {
            printf ("Input data. At the end of the introduction  press <ctrl>+z.\n") ;
            while ( !feof(stdin) ) {
                printf("X,Y? ");
                scanf("%f %f", &X,&Y);
                if ( !feof(stdin) )
                      fprintf(fileX, "%7.2f %7.2f\n",X, Y);
            }
      fclose (fileX);
      }
      return 0;
}
```

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

int main () {
    float X, Y;
    FILE *fileX;
    if (( fileX = fopen("fileX.txt","w")) == NULL)
        printf ("Error!");
    else {
        printf ("Input data. At the end of the introduction  press <ctrl>+z.\n") ;
        while ( !feof(stdin) ) {
              printf("X,Y? ");
              scanf("%f %f", &X,&Y);
              if ( !feof(stdin) )
              fprintf(fileX, "%7.2f %7.2f\n",X, Y);
        }
    fclose (fileX);
}
return 0;
```

```
Input data. At the end of the introduction  press <ctrl>+z.
X,Y? 1 2
X,Y? 2.3 5
X,Y? 5 6.3
X,Y? 3.4 6.8
X,Y? ^Z
```

fileX.txt - Блокнот

Файл Правка Формат Вид

1.00	2.00
2.30	5.00
5.00	6.30
3.40	6.80

Програма створює простий файл послідовного доступу, який може використовуватися в програмі аналізу деяких дослідних даних. Для кожного досліду треба ввести два параметри – X і Y. Дані для кожного досліду створюють запис.

За допомогою оператора

fprintf(fileX, "%7.2f %7.2f\n",X, Y);

кожний запис буде записаний у файл **fileX.txt.** Функція **fprintf** – це аналог функції **printf**, за виключенням того, що функція **fprintf** у якості першого аргументу отримує покажчик файла, в який будуть записані дані.

Рядки

while (!feof(stdin))

та

if (!feof(stdin))

містять виклик функції ***feof***, щоби визначити, чи встановлений ***індикатор кінця файла***, пов'язаний з потоком **stdin**. Індикатор кінця файла встановлюється тоді, коли користувач вводить відповідну комбінацію клавіш. Для комп'ютерів IBM PC та сумісних із ними – це комбінація клавіш <ctrl> та <z>. Функція **feof** повертає ненульове значення (true), якщо індикатор кінця файла встановлений. У протилежному випадку ця функція повертає нуль.

Читання файла послідовного доступу

Для зчитування даних із файла послідовного доступу програма, як правило, починає читання з початку файла і зчитує всі дані послідовно до тих пір, поки вони не закінчаться.

Для зчитування одного запису з файла використовується функція ***fscanf***, яка аналогічна функції **scanf**, за виключенням того, що як перший аргумент вказується покажчик файла, із якого дані зчитуються.

Наведемо програму, яка демонструє зразок читання і виведення даних послідовного файла.

Функція fscanf()


```
int fscanf(FILE*fp, const char* Формат,
СписокАдр),
```

де

`FILE*f` – покажчик на файл, з якого відбувається читання,

`const char*fmt` – форматний рядок,

`СписокАдр` – список адрес змінних, в які заноситься інформація з файла.

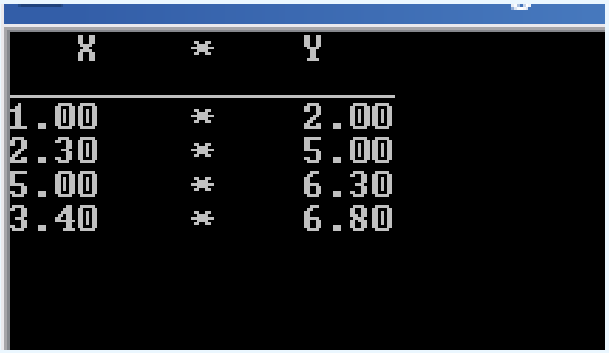
Виконує форматоване (аналогічно scanf()) ***читання*** значень змінних з файла, вказаним як перший параметр.

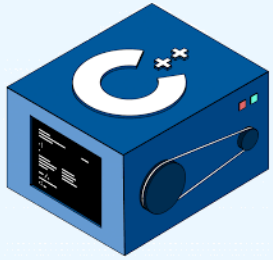
Функція повертає число змінних, яким присвоєне значення.

Файл, зв’язаний з потоком, мусить бути відкритий в режимі, що допускає читання. Заголовочний файл: <stdio.h>

Приклад читання і виведення даних послідовного файла.

```
#include <stdio.h>
#include <conio.h>
int main (void) {
    float X, Y;
    FILE *fileX;
    clrscr();
    if (( fileX = fopen("fileX.txt","r")) == NULL)
          printf ("Error!");
    else  {
          printf ("   X    *    Y   \n") ;
          printf ("_____________________\n") ;
          while ( !feof(fileX) ) {
                fscanf(fileX,"%f%f", &X, &Y);
                if ( !feof(fileX) )
                     printf("%-7.2f * %7.2f \n",X,Y);
          }
          fclose (fileX);
     } getch();
     return 0;
}
```





Дякую за увагу!