Лекція 17

Робота з файлами. Текстові файли

Поняття файлу

- **Файл** (file — шухляда, тека, папка) — інформаційний об'єкт, що містить дані або програми і розміщується на поіменованій ділянці носія даних, сутність, елемент (одиниця носія інформації; англ. media unit), що дозволяє отримати доступ до певного ресурсу обчислювальної системи і має такі ознаки:
  - фіксована назва (назва файлу — послідовність символів (англ. string), число чи щось інше, що однозначно характеризує файл);
  - певну логічну будову (структуру) і відповідні йому операції читання/запису.
- На практиці ***це іменований блок інформації, який зберігається на носії інформації.***

Поняття файлу

- Файл обов'язково має назву і може мати будь-який розмір інформації (максимальна довжина назви та розміру файлу обмежується властивостями конкретної файлової системи).

- ***Файл*** - це впорядкована сукупність даних, що зберігається на диску і займає іменовану область зовнішньої пам'яті.

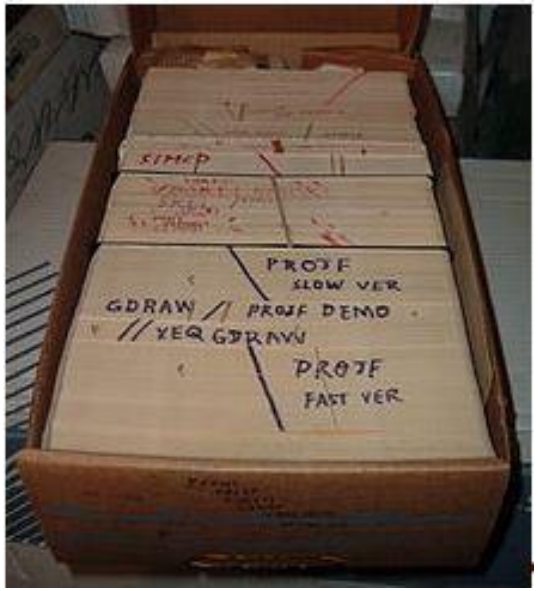
- Файл - це довільний блок інформації, або пристрій вводу-виводу, асоційований із ним (як середовище, засіб передачі "довільних блоків інформації").

- Кожна комп'ютерна програма відкриває принаймні три файли стандартних потоків:
  - вхідний файл (stdin) - асоціюється із клавіатурою,
  - вихідний файл (stdout) - із дисплеєм терміналу,
  - файл виводу повідомлень про помилки (stderr) - із дисплеєм терміналу.

- Файлом може бути також ділянка оперативної пам'яті програми.

Історія

- Слово «файл» вперше було публічно використане в контексті зберігання даних комп'ютером в лютому 1950-го.
- У 1952 році «файл» використовували для означення інформації, що зберігалась на перфокартах.



Файл перфокарт

Двійка твердих дисків системи IBM 305

Структура файлу

- Файли можна умовно поділяти на файли ***простої та складної структури*** (хоча точка зору на структуру файлу залежить від тієї програми, яка його обробляє).

- Файли простої структури складаються з послідовності записів (records) - елементарних одиниць, в термінах яких виконуються операції обміну з файлом. Записи можуть бути:
  - рядками, якщо це текстовий файл;
  - двійковими даними фіксованої довжини;
  - двійковими даними змінної довжини.

- *Файли складної структури можуть бути самого різного виду.*

- Складна структура файлу може бути змодельована записами шляхом додавання відповідних керуючих символів.

- ***Файли інтерпретуються*** операційною системою або програмами їх обробки.

Атрибути файлів

- Ім'я (Name) - назва файлу в символьній формі, сприймається користувачем.
- Тип (Type) - тип збереженої у файлі інформації. Окремий атрибут тип необходний для систем, які підтримують різні типи файлів. Наприклад, в системі Ельбрус значенням атрибута тип файлу є число, кодує тип: 0 - дані, 2 - код, 3 - текст і т.д. Однак більш загальноприйнятим підходом є підхід, прийнятий в системах MS DOS, Windows, UNIX: тип файлу кодується розширенням імені, наприклад, book.txt- текстовий файл (.txt), що містить текст книги.
- Розміщення (Location) - покажчик на розміщення файлу на пристрої.
- Розмір (Size) - поточний розмір файлу.
- Захист (Protection) - керуюча інформація, що задає повноваження читання, зміни і виконання файлу.
- Час і дата. Наприклад, у всіх системах зберігається дата створення файлу і дата останньої модифікації файлу. Остання відіграє важливу роль при компіляції (збірці) великих програмних проектів, так як утиліти для збірки проектів (наприклад, make) визначають по співвідношенню дат останньої модифікації файлів вихідного коду і двійкового коду, чи слід перекомпілювати вихідний файл.

Операції над файлами

- Хоча набір операцій над файлами і особливо їх позначень відрізняється від системи до системи, можна виділити наступні основні операції над файлами.
- Створення файлу (Create). Створюється заголовок файлу; спочатку його вміст (пам'ять) порожньо.
- Запис в файл (Write). Як правило, відбувається записами (records) або блоками- більшими логічними одиницями інформації, що об'єднують кілька записів, з метою оптимізації операцій введення-виведення.
- Читання з файлу (Read). Зазвичай також виконується записами або блоками.
- Пошук позиції всередині файлу (позиціонування) (Seek). Позиція задається номером запису або блоку, або спеціальними іменами, які позначають початок файлу (позиція перед першим записом) іліконец файлу (позиція після останнього запису).
- Видалення файлу (Delete). Залежно від реалізації системи файлів, помилкове видалення файлу може бути фатальним (UNIX) або виправних (MS DOS).
- Скорочення файлу (Truncate).
- Відкриття файлу (Open) - пошук файлу в структурі директорій по його символьному імені (шляху) і зчитування його заголовка і одного або декількох суміжних блоків в буферав основний пам'яті.
- Закриття файлу (Close) - запис вмісту буферів в блоки файлу; оновлення файлу у зовнішній пам'яті відповідно до його поточним станом; звільнення всіх структур в основний пам'яті, пов'язаних з файлом.
- Для виконання операцій обміну з файлом (read, write), як правило, файл необхідно відкрити. Закриття файлу є обов'язком користувача процесу; однак, якщо він з якоїсь причини цього не виконує, то закриває всі файли, відкриті процесом, операційна система після завершення або припинення процесу.

Типи файлів - імена та розширення

тип файлу	розширення імені	Функціональність
виконуваний код (завантажувальний модуль)	exe, com, bin або відсутній	готова до виконання програма в бінарному машинному коді
об'єктний модуль	obj, o	відкомпільоване програма в бінарному коді
вихідний код на мові програмування	c, cc, Java, pas, asm, a	вихідний код на різних мовах (Сі, Паскаль і ін.)
командний файл	bat, sh	файл з командами для командного інтерпретатора
текст	txt, doc	текстові дані, документи
документ для текстового процесора	wp, tex, rtf, doc	документ в форматі будь-якого текстового процесора
бібліотека	lib, a, so, dll, mpeg, mov, rm	бібліотеки модулів для програмування
файл для друку або візуалізації	arc, zip, tar	ASCII або бінарний файл у форматі для друку або візуалізації
архів	arc, zip, tar	кілька файлів, згрупованих в один файл, для архівації або зберігання
мультимедіа	mpeg, mov, rm	бінарний файл, який містить аудіо- або відео інформацію

Методи доступу до файлів

- Традиційно розрізняються файли послідовного та прямого доступа.
- Файл послідовного доступу - це файл, доступ до якого можливий тільки позиціонуванням на початок і кінець і потім операціями обміну виду вважаєте оновити наступну (попередню) запис.
- Файл прямого доступу - це файл, для якого можливий безпосередній доступ по номеру запису і операція обміну з явним зазначенням номера запису. (при виконанні обміну з файлом завжди існує деяка поточна позіція по файлу, яка вказує на деякий запис, на позицію перед початком або після кінця файлу)
- В операціях над файлом послідовного доступу довільна установка позиції ***не допускається***, а дозволені тільки операції, автоматично пересувають поточну позицію на наступну (попередню) запис.
-
- Подібна особливість пов'язана з різницею пристроїв, на яких розміщені файли (наприклад, магнітна стрічка - по суті справи, послідовне пристрій), проте необхідність організації послідовних або прямих файлів може бути пов'язана з суттю завдання.
-
- Мабуть, послідовний доступ використовується частіше: саме так відбувається введення даних, виведення результатів на друк або на екран.

Робота з файлами

- Коли потік відкривається для введення-виведення, він зв'язується зі стандартною структурою типу FILE, яка визначена в **stdio.h**. Структура FILE містить необхідну інформацію про файл.
- Відкриття файлу здійснюється за допомогою функції **fopen ()**, яка повертає покажчик на структуру типу FILE, який можна використовувати для подальших операцій з файлом.
- **FILE *** fopen (name, type);
- name - ім'я файлу (включаючи шлях),
- type - покажчик на рядок символів, що визначають спосіб доступу до файлу:
  - "R" - відкрити файл для читання (файл повинен існувати);
  - "W" - відкрити порожній файл для запису; якщо файл існує, то його вміст втрачається;
  - "A" - відкрити файл для запису в кінець (для додавання); файл створюється, якщо він не існує;
  - "R +" - відкрити файл для читання і запису (файл повинен існувати);
  - "W +" - відкрити порожній файл для читання і запису; якщо файл існує, то його вміст втрачається;
  - "A +" - відкрити файл для читання і доповнення, якщо файл не існує, то він створюється.

- Значення, що повертається - покажчик на відкритий потік. Якщо виявлена помилка, то повертається значення NUL

Робота з файлами

- Значення, що повертається: значення 0, якщо потік успішно закритий; константа EOF, якщо сталася помилка.

```
#include <stdio.h>
int main () {
   FILE * fp;
   char name [] = "my.txt";
   if ((fp = fopen (name, "r")) == NULL)
   {
     printf ( "Не вдалося відкрити файл");
     getchar ();
     return 0;
   }
   // відкрити файл вдалося
   ... // необхідні дії над даними
   fclose (fp);
   getchar ();
   return 0;
}
```

Функції роботи з файлами

- ***Читання символу з файлу:***
- char fgetc (потік); - аргументом функції є покажчик на потік типу FILE. Функція повертає код ліченого символу. Якщо досягнуто кінця файлу або виникла помилка, повертається константа EOF.
- ***Запис символу в файл:***
- fputc (символ, потік); - аргументами функції є символ і покажчик на потік типу FILE. Функція повертає код ліченого символу.
- Функції ***fscanf () і fprintf ()*** аналогічні функціям scanf () і printf (), але працюють з файлами даних, і мають перший аргумент - покажчик на файл.
- fscanf (потік, "ФорматВвода", аргументи);
- fprintf (потік, "ФорматВивода", аргументи);

Функції роботи з файлами

- Функції **fgets ()** і **fputs ()** призначені для введення-виведення рядків, вони є аналогами функцій gets () і puts () для роботи з файлами.

- **fgets** (ВказівникНаРядок, КількістьСимволів, потік);

- Символи читаються з потоку до тих пір, поки не буде прочитаний символ нового рядка '\ n', який включається в рядок, або поки не настане кінець потоку EOF, чи не буде прочитано максимальне символів. Результат поміщається в покажчик на рядок і закінчується нуль символом '\ 0'. Функція повертає адресу рядка.

- **fputs** (ВказівникНаРядок, потік);

Копіює рядок в потік з поточної позиції. Завершальний нуль символ не буде копіюватися.

Приклад

- Ввести число і зберегти його у файлі s1.txt. Вважати число з файлу s1.txt, збільшити його на 3 та зберегти в файлі s2.txt.

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
  FILE * S1, * S2;
  int x, y;
  system ( "chcp 1251");
  system ( "cls");
  printf ( "Введіть число:");
  scanf ( "% d", & x);
  S1 = fopen ( "S1.txt", "w");
  fprintf (S1, "% d", x);
  fclose (S1);
  S1 = fopen ( "S1.txt", "r");
  S2 = fopen ( "S2.txt", "w");
  fscanf (S1, "% d", & y);
  y + = 3;
  fclose (S1);
  fprintf (S2, "% d \ n", y);
  fclose (S2);
  return 0;
}
```

- Результат виконання
- 2 файли

Текстовий файл

- **Те́кстовий файл** — форма подання послідовності символів у комп'ютері, де кожен символ із задіяного набору символів кодується одним байтом чи послідовністю двох, трьох і т. д. байтів.
- На відміну від терміна «текстовий формат», що характеризує вміст даних, термін «текстовий файл» **стосується файлу та характеризує його як контейнер**, який зберігає такі дані.
- **Текстовий файл** — послідовність символів (переважно друкованих знаків, що належать тому чи іншому набору символів). Ці символи зазвичай згруповані в рядки (англ. lines, rows).
- Іноді кінець текстового файлу (особливо тоді, коли в файловій системі не зберігається інформація про розмір файлу) також позначається спеціальними знаками (одним або більше), відомими як маркери кінця файлу.
-
- Текстовий файл може містити як форматований, так і неформатований текст.
- Текстовим файлам протиставляються **двійкові (бінарні) файли**, в яких інформація організована за іншими принципами (вона містить інформацію, не прив'язану до набору символів).

Текстовий файл:Переваги та недоліки

- **Переваги**
- *Універсальність* — текстовий файл може бути прочитаний (так чи інакше) на будь-якій системі або ОС, особливо, якщо йдеться про однобайтові кодування на кшталт ASCII, які не схильні до проблеми, характерної для інших форматів файлів — для них не важлива різниця в порядку байтів або довжині машинного слова на різних платформах.
- *Стійкість* — кожне слово та символ у такому файлі самодостатні і, якщо трапиться пошкодження байтів у такому файлі, то зазвичай можна відновити дані за контекстом або продовжити обробку решти вмісту, в той час як у стиснених чи двійкових файлів пошкодження декількох байтів може зробити файл абсолютно невідновним. Багато систем управління версіями розраховані на текстові файли і з двійковими файлами можуть працювати лише як з єдиним цілим.

- *Формат текстового файлу вкрай простий і його можна змінювати текстовим редактором — програмою, яка входить в комплект практично будь-якої ОС.*

Текстовий файл:Переваги та недоліки

- **Недоліки**
- У великих нестиснутих текстових файлів *низька інформаційна ентропія* — ці файли займають більше місця, ніж мінімально необхідно. Хоча ця ж надмірність інформації визначає підвищену стійкість до збоїв у каналах передачі даних і при отриманні даних з носіїв, наприклад, з магнітної стрічки.
- *Деякі операції з текстовими файлами неефективні.*
- Наприклад, якщо в файлі зустрінеться число, обчислювальна система до початку операцій з ним повинна буде перетворити його в свій внутрішній формат, застосувавши порівняно складну процедуру конвертації числа; щоб перейти на 1000-ий рядок, потрібно порахувати попередні 999 рядків; складно замінити один рядок іншим, тощо. Тому при роботі з великими обсягами даних текстові файли застосовують лише як проміжний формат, що забезпечує інтероперабельність.

Формати, засновані на текстових файлах

- В силу своєї простоти текстові файли нерідко використовуються для ***зберігання службової інформації*** (наприклад, логів): оскільки операція додавання в кінець текстового файлу нових даних **не вимагає значних обчислювальних ресурсів** (незалежно від уже наявного обсягу файлу і виду текстових даних, що додаються), ведення текстових лог-файлів зазвичай відбувається ефективно та непомітно для користувача і для інших додатків (аж до вичерпання дискового простору).
- Текстовий формат служить основою для багатьох спеціалізованих форматів (наприклад, .ini, SGML, HTML, XML, TeX, вихідних текстів мов програмування).
-
- В текстовому файлі текст може зберігатися як в неформатованому, так і в форматованому або розміченому вигляді (наприклад, **Rich Text Format, HTML**), де кожен символ чи група символів (рядки, абзаци, таблиці тощо) може бути відформатований (визначений шрифт, накреслення, розмір і т. д.).

Розширення імен файлів

- В DOS і Windows для файлів з неформатованим текстом зазвичай **використовується розширення .txt.** Проте, текстовими можуть бути файли з будь-яким іншим розширенням або й без нього. Наприклад, вихідні коди програм зазвичай зберігаються в файлах з розширеннями, відповідними мові програмування, якою вони написані (.bas, .pas, .c тощо).
-
- Форматований текст (текст із розміткою) зазвичай зберігається у файлах з розширенням, відповідним формату або мові розмітки — **.rtf, .htm, .html** тощо.

Кодування

- 8-бітний текст
- Історично для кодування текстових файлів застосовувалися 7-бітний набір символів ASCII, а також 8-бітні EBCDIC та різні розширення ASCII. У 8-бітних кодових сторінках у першій половині кодової таблиці загальноприйнято використовувати символи, відповідні ASCII.
-
- Перевагою 8-бітного представлення тексту є програмна простота та незалежність від проблеми порядку байтів або довжини машинного слова на різних платформах. Недолік — багато різних, часом несумісних стандартів.
-

Кодування

- Unicode в текстових файлах
- Застосування Unicode у текстових файлах хоча й переважно ***вирішує «проблему кодувань» та стандартизує вживання керуючих символів, але створює свої проблеми***.
- У більшості сучасних систем неподільною одиницею інформації в потоці даних є байт (октет, 8 біт), яких для кодування одного символу Юнікоду потрібно декілька.
- Як вихід, застосовуються несумісні між собою системи: UTF-8 і дві версії **UTF-16** (UTF-16LE та UTF-16BE з протилежним між собою порядком байтів).
- Іноді в початок файлу додають спеціальний символ-маркер **(U + FEFF)**, що дозволяє розпізнати формат однозначно.
- UTF-8 має перевагу зворотної сумісності з ASCII, однак програмна обробка тексту в UTF-8 ускладнюється непостійним розміром символу. Тексти в Юнікоді відрізняються ще більшою надмірністю, ніж 8-бітові.

Кодування

- Символи керування

- Різні операційні системи дотримуються свого уявлення про символи нового рядка та кінця файлу.
- В UNIX символ нового рядка — одиничний символ LF (код 0xA)

- Mac OS — символ CR (код 0xD)

- DOS і Windows — послідовність двох символів: CR і LF.

Позиціонування в текстовому файлі

- **Повторювані фрагменти в тексті.** Вихідний текстовий файл може містити вкладені одна в одну *фрагменти виду (12 ...)*, *обмежені дужками*, що включають в себе *довільний текст і лічильник його повторень у вигляді цілої константи* (рядки цифр). Потрібно згенерувати вихідний текст, «розкривши» повторення.
- Наявність вкладених фрагментів визначає ***рекурсивний характер програми***.
- Кожен фрагмент повинен оброблятися окремим викликом рекурсивної функції.
- Для усунення проблем, пов'язаних зі зберіганням повторюваного фрагмента довільної довжини, пропонується запам'ятати початкову позицію фрагмента в файлі і перечитувати його при циклічному виведення.
- Початковою точкою рекурсії найзручніше вважати виявлення дужки що в поточному потоці (тобто при виклику вона вважається вже прочитаної).

Позиціонування в текстовому файлі

```
void more (FILE * fd) {
long pp; // Поточна позиція фрагмента повторення
char c; int n = 0; // Кількість повторів
while (1) {
        pp = ftell (fd); // Запам'ятати поточну позицію
        char c = getc (fd);
        if (! isdigit (c)) break;
        n = n * 10 + c-'0 '; // Накопичення константи
        }
if (n == 0) n = 1; // Відсутність константи - повторити 1 раз
while (n -! = 0) {// Повторювати фрагмент
        fseek (fd, pp, SEEK_SET); // Повернутися на початок
        while ((c = getc (fd))! = EOF && c! = ')') {
                if (c == '(') more (fd); // Вкладений фрагмент -
                else                     // рекурсивний виклик після '('
                        putchar (c); // Перечитать фрагмент до ')'
                }}}
void main () {FILE * fd = fopen ( "d310-00.txt", "r"); more (fd); fclose (fd); }
```

Позиціонування в текстовому файлі

- **З main** функція **викликається при встановленій початкової позиції** файлу, що за **замовчуванням визначає одноразовий перегляд його вмісту**. В цьому випадку ознакою **завершення фрагмента служить кінець файлу (EOF)**.

Позиціонування в текстовому файлі

- **Посторінковий перегляд тексту**. Для перегляду тексту в довільному порядку необхідно заздалегідь послідовно прочитати файл, **зробивши «закладки» в потрібних місцях**, в нашому випадку - **запам'ятати адреси у файлі кожної сторінки тексту**, викликавши функцію **ftell** перед читанням черговий двадцятки рядків.

Позиціонування в текстовому файлі

```
void main () {
FILE * fd; char name [30] = "94.txt", str [80];
int i, n, NP;                    // Кількість сторінок у файлі
long * POS;                      // Масив адрес початку сторінок у файлі
if ((fd = fopen (name, "r")) == NULL) return;
for (n = 0; fgets (str, 80, fd)! = NULL; n ++);
NP = n / 20; if (n% 20! = 0) NP ++;  // Кількість рядків
fseek (fd, 0, SEEK_SET);      // Повернутися в початок файлу
POS = new long [NP];          // Динамічний масив "закладок"
for (n = 0; n <NP; n ++) {            // Перегляд сторінок файлу
        POS [n] = ftell (fd);  // Запам'ятати початок сторінки
        for (i = 0; i <20; i ++)        // Читання рядків сторінки
                if (fgets (str, 80, fd) == NULL)
                        break;      // Кінець файлу - вихід з циклу
        if (i <20) break;         // Неповна сторінка - вихід
   }
```

Позиціонування в текстовому файлі

```
while (1) {
        printf ( "page number (0 ..% d):", NP-1); scanf ( "% d", & n);
        if ((n> = NP) | | (n <0)) break;
        fseek (fd, POS [n], SEEK_SET); // Позиціонуватися на сторінку
        for (i = 0; i <20; i ++) {// Повторне читання сторінки
                if (fgets (str, 80, fd) == NULL) break;
        printf ( "% s", str);
    }}}
```

Позиціонування в текстовому файлі

- За допомогою позиціонування в тексті можна ввести будь-які системи його інтерпретації, в тому числі аналогічні механізмів, що використовуються в мовах програмування.
- Для моделювання виклику функції безпосередньо над текстом програми необхідно:
  1. Одноразово переглянути текст, виділити в ньому заголовки функцій і викликом функції ftell запам'ятати їх адреси в тексті. Створити таблицю імен, зберігши в ній пари «ім'я-адреса»;
  2. Створити в програмі стек, що містить «точки повернення», які також є адресами в тексті;
  3. Розпочати інтерпретацію тексту з того, що знайти в таблиці ім'я main та виконати за допомогою fseek позиціонування до її тіла в файлі;
  4. Якщо при читанні тексту зустрічається «виклик» функції, то після його прочитання зберегти в стеці поточну адресу в текстовому файлі, отримавши його через ftell. Потім знайти в таблиці ім'я викликається функції і позиціонуватися до її початку за допомогою fseek;
  5. Якщо час читання виявляється завершити використання функції, то з стека витягується адреса (точка повернення) і до неї виробляється позиціонування за допомогою fseek.

Позиціонування в текстовому файлі

```
void main () {
FILE * fd; char cc, name [30] = "94-03.txt";
long POS; int cnt;
if ((fd = fopen (name, "r + w")) == NULL) return;
while (1) {
        POS = ftell (fd); // Запам'ятати адресу символу
        if ((cc = getc (fd)) == EOF) break;
        if (cc> = '0' && cc <= '9') {// Прочитана цифра
            fseek (fd, POS, SEEK_SET); // Повернутися на 1 символ
            fscanf (fd, "% d", & cnt); // і прочитати лічильник - 6 символів
            cnt ++; // Збільшити лічильник
            fseek (fd, POS, SEEK_SET);
    // Повернутися на початок лічильника
            fprintf (fd, "% 06d", cnt);
          // і записати лічильник - 6 символів
                  break; }          }
fclose (fd);}
```

Приклади

- Відкриємо файл і запишемо в нього Hello World

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

void main() {
    FILE *file;
    //Відкриваємо текстовий файл з правми на запис
    file = fopen("C:/c/test.txt", "w+t");
    //Пишемо в файл
    fprintf(file, "Hello, World!");
    // Зачиняємо файл
    fclose(file);
    getch();
}
```

Приклади

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define ERROR_OPEN_FILE -3

void main() {
    FILE *file;
    char buffer[128];
    file = fopen("C:/c/test.txt", "w");
    if (file == NULL) {
        printf("Error opening file");
        getch();
        exit(ERROR_OPEN_FILE);
    }
    fprintf(file, "Hello, World!");
    freopen("C:/c/test.txt", "r", file);

    if (file == NULL) {
        printf("Error opening file");
        getch();
        exit(ERROR_OPEN_FILE);
    }
    fgets(buffer, 127, file);
    printf("%s", buffer);
    fclose(file);
    getch();
}
```

Приклади

- **feof**
  Функція int feof (FILE * stream); повертає істину, якщо кінець файлу досягнутий. Функцію зручно використовувати, коли необхідно пройти весь файл від початку до кінця. Нехай є файл з текстовим вмістом text.txt. Вважаємо посимвольний файл і виведемо на екран.

Приклади

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
 void main() {
    FILE *input = NULL;
    char c;
    input = fopen("D:/c/text.txt", "rt");
    if (input == NULL) {
       printf("Error opening file");       _getch();       exit(0);    }
    while (!feof(input))
    {
        c = fgetc(input);
        fprintf(stdout, "%c", c);
    }
     fclose(input);
    _getch();  }
```

Дякую за увагу!