



ОПЕРАЦІЇ СИМВОЛЬНОГО ВВОДУ-ВИВОДУ

Питання 2.4.

Нагадування про ввід-вивід

```
1  #include <stdio.h>
2  #include <string.h>      // for strlen() prototype
3  #define DENSITY 62.4     // human density in lbs per cu ft
4  int main()
5  {
6      float weight, volume;
7      int size, letters;
8      char name[40];       // name is an array of 40 chars
9
10     printf("Hi! What's your first name?\n");
11     scanf("%s", name);
12     printf("%s, what's your weight in pounds?\n", name);
13     scanf("%f", &weight);
14     size = sizeof name;
15     letters = strlen(name);
16     volume = weight / DENSITY;
17     printf("Well, %s, your volume is %.2f cubic feet.\n", name, volume);
18     printf("Also, your first name has %d letters,\n", letters);
19     printf("and we have %d bytes to store it.\n", size);
20
21     return 0;
22 }
```

Символьний ввід-вивід у мові C

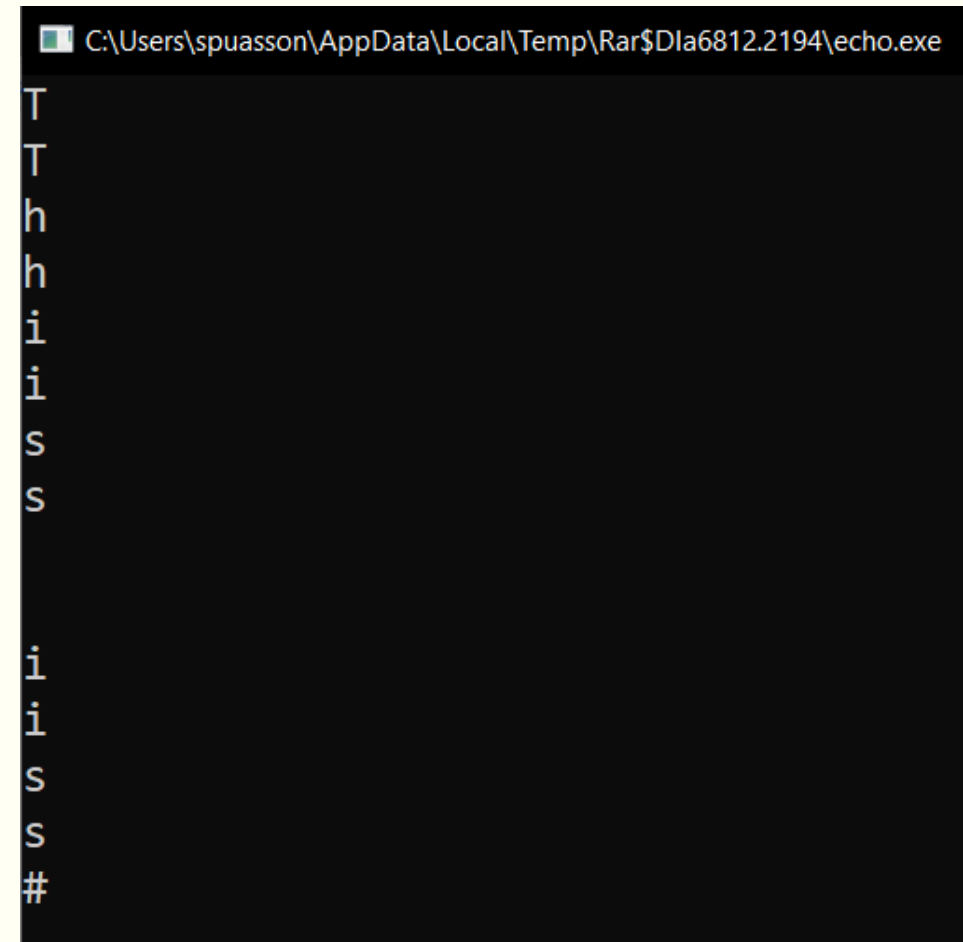
- Функції вводу-виводу переміщують інформацію в програму та з неї;
 - функції `getchar()` і `putchar()` виконують введення та виведення по одному символу за раз.
- Спочатку функції вводу-виводу не були частиною визначення мови C.
 - На практиці моделлю для цих функцій служила реалізація C для ОС Unix.
- Багато постачальників реалізацій мови C пропонують додаткові функції вводу-виводу, які задіюють спеціальні засоби обладнання.
 - Інші функції або сімейства функцій включаються в окремі ОС, які підтримують, наприклад, спеціальні графічні інтерфейси на зразок таких у Windows і Macintosh.

Односимвольний ввід-вивід: getchar() і putchar()

- Демонстрація: ехо-вивід

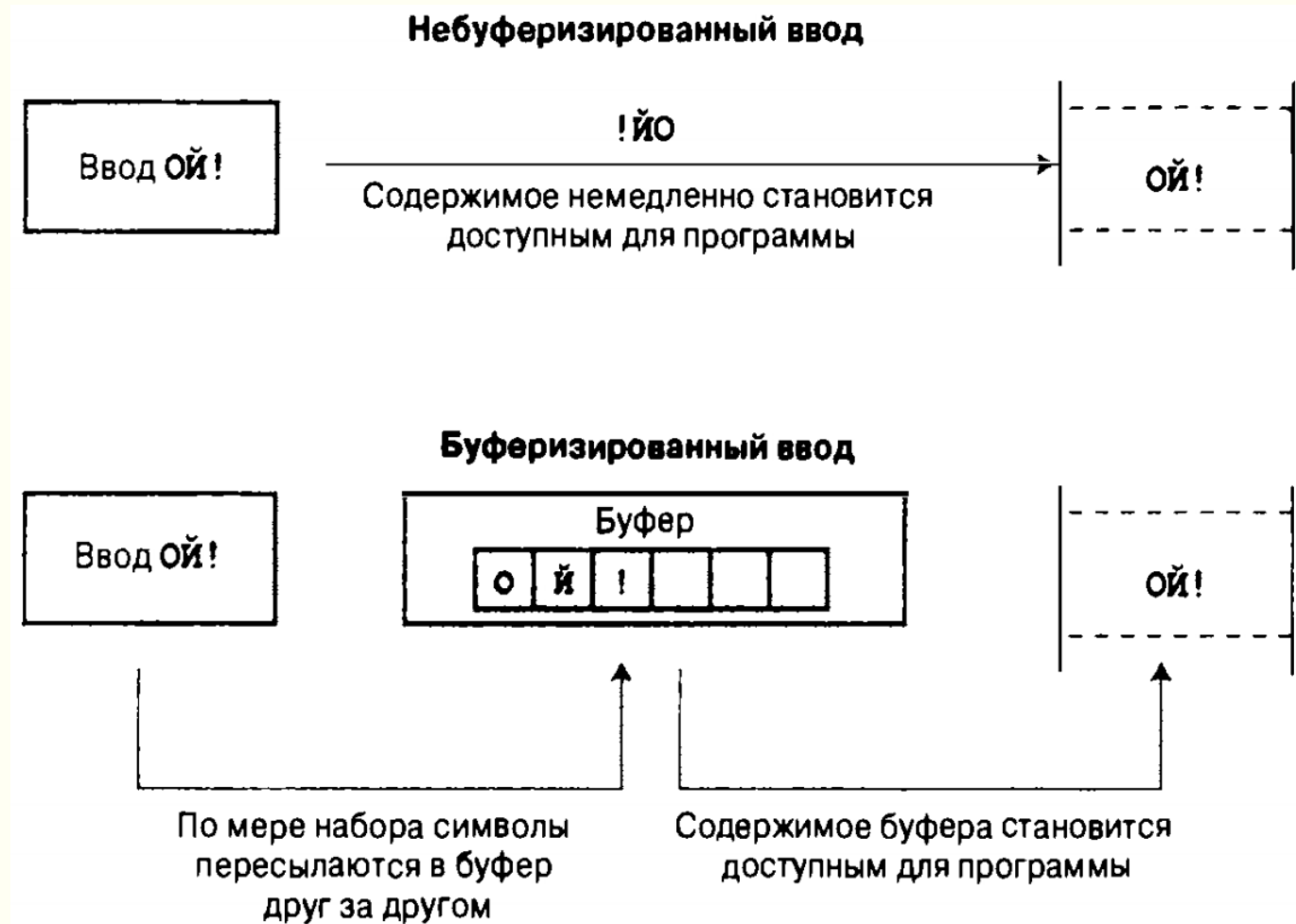
```
1  #include <stdio.h>
2  int main(void)
3  {
4      char ch;
5
6      while ((ch = getchar()) != '#')
7          putchar(ch);
8
9      return 0;
10 }
```

- Чи можна краще завершувати ввід, не використовуючи “#”?
- Всі символи рядка виводяться тільки після натиснення Enter.
 - Буферизація?



Буфери

- Негайний ехо-вивід символів, які вводяться, на екрані є прикладом *небуферизованого вводу*.
 - Затриманий ехо-вивід ілюструє *буферизований ввід*.
 - Введені символи накопичуються в тимчасовій області – буфері.
- Існує 2 види буферизації:
 - Повністю буферизований ввід-вивід
 - Порядково буферизований ввід-вивід
- Розмір буфера залежить від системи.
 - Найбільш поширені розміри – 512 та 4096 байтів

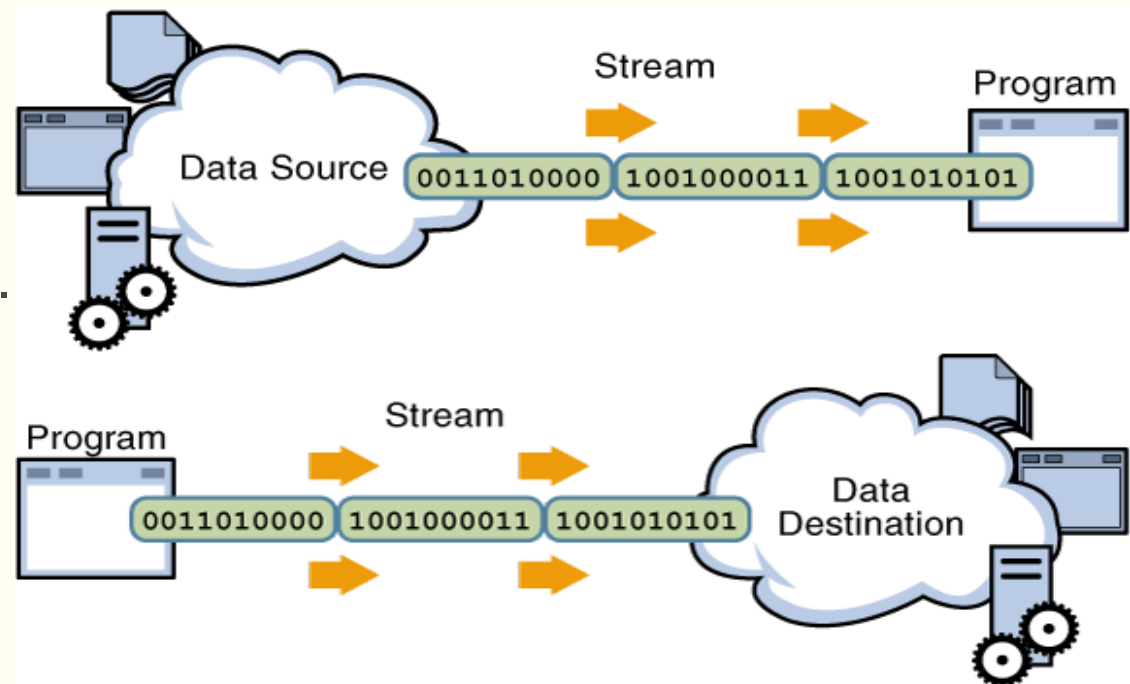


Буферизований та небуферизований ввід-вивід

- Стандарт ANSI C вимагає буферизованого вводу-виводу.
 - Деякі системи не підтримують небуферизований ввід-вивід.
- Зазвичай компілятор допускає небуферизований ввід-вивід.
 - **IBM PC**: спеціальний заголовковий файл `conio.h` надає сімейство функцій для небуферизованого вводу: `getche()`, `getch()`.
 - **Unix**: буферизацією управляє система: функція `ioctl()` вказує тип вводу, а `getchar()` поводитьсья відповідним чином.
 - **ANSI C**: функції `setbuf()` та `setvbuf()` надають певний контроль на буферизацією, проте менш ефективні.
- Припускаємо використання буферизованого вводу.

Завершення клавіатурного вводу

- Символ завершення вводу повинен не зустрічатись у звичайному тексті.
 - У мові C пристрої вводу-виводу розглядаються таким же чином, що і файли (область пам'яті, в якій зберігається інформація).
 - Концептуально програма на C має справу не з файлом напряму, а з потоком – ідеалізованим током даних, на який відображається реальний ввід-вивід
- Наслідок: при роботі з клавіатурним вводом можна використовувати ті ж прийоми, що й при роботі з файлами.



Кінець файлу

- Функції вводу в С мають вбудований засіб знаходження кінця файлу.
 - Використовується вбудований символ <Ctrl+Z> - маркер кінця файлу.

Фраза:

Робот Бишоп
плавно открыл люк
и ответил на свой вызов.

Фраза в файле:

```
Робот Бишоп\nплавно открыл люк\nи ответил на свой вызов.\n^Z
```

- Інший підхід – пам'ятати розмір файлу: Unix використовує його для всіх файлів
- Функція `getchar()` повертає спеціальне значення при досягненні кінця файлу – EOF (end of file).
 - Зазвичай EOF визначено у файлі `stdio.h`: `#define EOF (-1)`
 - Не завжди -1, проте програмісту це не важливо.

Кінець файлу

- Використовуємо EOF:

- while ((ch = getchar()) != EOF) {...}

```
#include <stdio.h>
int main(void)
{
    int ch;

    while ((ch = getchar()) != EOF)
        putchar(ch);

    return 0;
}
```

- Зауваження до коду:

- Не потрібно знати дійсне значення EOF.
 - Тип змінної ch змінено з char на int.
 - Оскільки функція getchar() має тип int, деякі компілятори попереджають про можливу втрату даних при присвоєнні значення, яке вона повертає, змінній типу char.
 - При натисненні <Enter> символи з буфера обробляються, після чого виводиться копія рядка.

Перенаправлення і файли

- У попередній програмі припускаємо, що пристроєм вводу є клавіатура.
 - Як програма дізнається, звідки брати дані?
- За замовчуванням використовується **стандартний пакет вводу-виводу**.
 - Джерелом даних вважається стандартний потік вводу - stdin.
 - Представляє собою все, що налаштовано як звичайний метод зчитування даних: клавіатура, сенсор, голосовий ввід тощо.
 - У сучасних системах цей потік можна переорієнтувати, наприклад, на файл.
- Як змусити програму працювати з файлами?
 - 1) явно використовувати спеціальні функції для відкриття, закриття, зчитування, запису в файли тощо.
 - 2) перенаправити ввід і вивід в інші канали, наприклад, у файл із файлу.
- Функція `getchar()` насправді не цікавиться, звідки потік stdin отримує свої дані.
 - Підтримка перенаправлення залежить від ОС, а не мови C.

Перенаправлення в Unix, Linux та командному рядку Windows

- Перенаправлення вводу для скомпільованого файлу `echo_eof(.exe)` на текстовий файл `words`:
 - **`echo_eof < words`**
 - Символ `<` представляє операцію перенаправлення в Unix, Linux и DOS / Windows.
 - Файл стає пристроєм вводу-виводу!
- Перенаправлення виводу.
 - **`echo_eof > mywords`**
 - Знак `>` призводить до створення нового файлу з назвою `mywords` та переадресує в нього вивід `echo_eof`.
 - Перенаправлення перепризначає потік `stdout` з пристрою відображення (екрану) на файл `mywords`.
 - Якщо файл вже існує, зазвичай він очищається.
- Після обробки комбінації `<Ctrl+D>` або `<Ctrl+Z>` програма завершиться, а на екрані знову відобразиться запрошення на введення.

Комбіноване перенаправлення

- `echo_eof < mywords > savewords`
 - Створити копію файла `mywords` та назвати її `savewords`.
- Теж працює: `echo_eof > savewords < mywords`
 - Не працює: `echo_eof < mywords > mywords`
- Загальні правила комбінування:
 - Операція перенаправлення з'єднує виконувану програму з файлом даних. Так не можна з'єднувати 2 файли чи 2 програми.
 - Ці операції не дозволяють отримувати ввід з кількох файлів, а вивід направляти в понад один файл.
 - Зазвичай пробіли між операціями не є обов'язковими.
 - за деякими виключеннями, коли застосовуються спеціальні символи командного рядка.
 - Наприклад, можна было бы иметь команду `echo_eof < words`.
- У середовищах Unix, Linux і Windows також доступна операція `>>` (додає дані в кінець існуючого файлу) та операція конвеєра (`|`), яка дозволяє з'єднати вивід одєї програми із вводом іншої.

Робота з файлом напряму

- Для файлів слід розрізняти роботу з
 - текстовим та двійковим вмістом,
 - текстовим та двійковим форматами,
 - текстовим та двійковим режимами.
- Вміст усіх файлів – двійковий, проте враховується тип даних.
 - Для вмісту обох видів у Unix використовується один файловий формат.
 - Каталоги Unix підтримують лічильник розміру файлу, що допомагає відстежити кінець, а перехід на новий рядок – ‘\n’.
 - Блокнот Windows підтримує текстовий формат MS-DOS (EOF=<Ctrl+Z>), інші додатки можуть використовувати формат, ближчий до Unix.
 - Доступні інші формати: підтримка рядків однакової довжини, кодування довжини рядка на його початку тощо.
- Мова C вводить для текстових файлів 2 режими доступу: двійковий і текстовий.
 - У двійковому доступний кожен окремий байт файлу, у текстовому – нюанси для окремих символів.
 - У Unix/Linux файлова структура лише одна, тому представлення однакові.

Двійкове та текстове представлення

Текстовый
файл MS-DOS

```
Этот рассвет\r\nПодарили мне\r\nПутешествие и весна!\r\n^Z
```

```
Этот рассвет\r\nПодарили мне\r\nПутешествие и весна!\r\n^Z
```

Так он виден программе С при открытии
в двоичном режиме

```
Этот рассвет\nПодарили мне\nПутешествие и весна!\n
```

Так он виден программе С при открытии
в текстовом режиме

Рівні вводу-виводу

- Крім представлення файлу в більшості випадків можна обирати один з двох рівнів вводу-виводу:
 - Низькорівневий ввід-вивід передбачає використання основних служб вводу-виводу для ОС.
 - Стандартний високорівневий ввід-вивід передбачає застосування стандартного пакету бібліотечних функцій мови C та визначень із заголовкового файлу `stdio.h`.
- Стандарт C підтримує тільки стандартний пакет вводу-виводу, оскільки немає гарантій щодо однаковості низькорівневої моделі вводу-виводу в різних ОС.
 - Окремі реалізації можуть пропонувати низькорівневі бібліотеки, проте стандарт C визначає переносиму модель вводу-виводу.

Стандартні файли

- Програми на С автоматично відкривають 3 файли (пристрої вводу-виводу):
 - Стандартний ввід (stdin) – читається за допомогою `getchar()`, `scanf()`.
 - Стандартний вивід (stdout) – використовується функціями `putchar()`, `puts()`, `printf()`.
 - Стандартний вивід помилок (stderr) – надає логічно відокремлене місце для відправки повідомлень про помилки.
- Стандартний пакет вводу-виводу має ще кілька переваг, крім переносимості:
 - Доступно багато спеціалізованих функцій, які спрощують вирішення проблем із вводом-виводом.
 - Ввід і вивід є буферизованими: інформація передається крупними порціями, а не побайтово.



count.c

Стандартний ввід-вивід

```
1 #include <stdio.h>
2 #include <stdlib.h> // exit() prototype
3
4 int main(int argc, char *argv[])
5 {
6     int ch;           // place to store each character as read
7     FILE *fp;         // "file pointer"
8     unsigned long count = 0;
9     if (argc != 2)
10    {
11        printf("Usage: %s filename\n", argv[0]);
12        exit(EXIT_FAILURE);
13    }
14    if ((fp = fopen(argv[1], "r")) == NULL)
15    {
16        printf("Can't open %s\n", argv[1]);
17        exit(EXIT_FAILURE);
18    }
19    while ((ch = getc(fp)) != EOF)
20    {
21        putc(ch, stdout); // same as putchar(ch);
22        count++;
23    }
24    fclose(fp);
25    printf("File %s has %lu characters\n", argv[1], count);
26
27    return 0;
28 }
```

- Застосовується для зчитування файлу та підрахунку кількості символів у ньому.
 - Деякі ОС можуть не розпізнати argv[0].
- Функція exit() припиняє роботу програми та закриває всі відкриті файли.
 - Стандарт вимагає, щоб при успішному завершенні програми застосовувалось значення 0 або макрос EXIT_SUCCESS, а при невдалому – EXIT_FAILURE.
 - Макроси та прототип exit() знаходяться в заголовковому файлі stdlib.h.
 - У рамках ANSI C використання return в початковому виклику main() дає такий же результат, як і виклик exit().

Функція fopen()

Строка режима	Описание
"r"	Открыть текстовый файл для чтения
"w"	Открыть текстовый файл для записи с усечением существующего файла до нулевой длины или созданием файла, если он не существует
"a"	Открыть текстовый файл для записи с добавлением данных в конец существующего файла или созданием файла, если он не существует
"r+"	Открыть текстовый файл для обновления (т.е. для чтения и записи)
"w+"	Открыть текстовый файл для обновления (чтения и записи), предварительно выполнив усечение файла до нулевой длины, если он существует, или создав файл, если его нет
"a+"	Открыть текстовый файл для обновления (чтения и записи) с добавлением данных в конец существующего файла или созданием файла, если он не существует; читать можно весь файл, но записывать допускается только в конец файла
"rb", "wb", "ab", ab+", "a+b", "wb+", "w+b", "ab+", "a+b"	Подобны предыдущим режимам, за исключением того, что вместо текстового режима они используют двоичный режим
"wx", "wbx", "w+x", "wb+x" или "w+bx"	(C11) Подобны режимам без буквы x, за исключением того, что они отказываются работать, если файл существует, и открывают файл в монопольном режиме, если это возможно

Особливості перезаписування файлів

- Нові режими запису C11 з літерою x володіють новими характеристиками:
 - 1) В одному з традиційних режимів запису при відкритті існуючого файлу `open()` стирає вміст файлу. Режими з буквою x забезпечують у такому випадку відмову функції `open()`.
 - 2) Якщо середовище дозволяє, можливість монопольного доступу в режимах з x запобігає доступу до файлу від інших програм або потоків, поки поточний процес не закриє цей файл.
- При використанні будь-якого режиму "w" без букви x для існуючого файлу його вміст усікається так, щоб програма могла почати роботу з чистого листа.
 - Проте спроба відкрити існуючий файл, застосовуючи C11-режим з буквою x, завершиться відмовою.
-

Вказівник файла

- Після успішного відкриття файлу функція `fopen()` повертає **вказівник файла**, який можуть використовувати інші функції вводу-виводу для вказування цього файла.
 - Його тип – вказівник на `FILE` (породжений тип, визначений в `stdio.h`, - структура).
 - Вказівник посилається не на файл, а на об'єкт даних, що містить інформацію про цей файл, зокрема про буфер для файлового вводу-виводу.
 - Буфер використовується функціями вводу-виводу (заповнюється та спустошується).
- Функція `fopen()` повертає нульовий вказівник (`NULL` – також визначений в `stdio.h`), якщо файл не вдається відкрити.
 - Тоді програма перериває роботу.
 - Можливі причини: переповнення диску, відсутність файлу за шляхом, недопустима назва, обмеження доступу, апаратна проблема тощо.

Функції `getc()` та `putc()`

- Працюють дуже подібно до `getchar()` та `putchar()`.
 - Відмінність: потрібно вказати, з яким файлом працювати.
 - Даний оператор означає “отримати символ з файла, який ідентифікується `fp`”:
`ch = getc(fp);`
- Аналогічно, `putc(ch, fpout);`
 - У списку аргументів `putc()` спочатку задається символ, а потім вказівник файла.
- Функція `getc()` повертає спеціальне значення EOF, якщо намагається прочитати символ та виявляє, що досягнуто кінець файлу.
 - Така поведінка не схожа на інші мови, в яких передбачена спеціальна функція для перевірки кінця файлу через спробу зчитування.
 - Для уникнення проблем з читанням порожнього файлу при файловому вводі повинен застосовуватись цикл `while` або `for` (не `do-while`).

Кінець файлу

```
// правильное проектное решение #1
int ch;                // переменная int для хранения EOF
FILE * fp;
fp = fopen("wacky.txt", "r");
ch = getc(fp);         // получить первоначальный ввод
while (ch != EOF)
{
    putchar(ch);        // обработать ввод
    ch = getc(fp);      // получить следующий ввод
}
```

```
// правильное проектное решение #2
int ch;
FILE * fp;
fp = fopen("wacky.txt", "r");
while (( ch = getc(fp)) != EOF)
{
    putchar(ch);        // обработать ввод
}
```

Невдале проектне рішення (дві проблеми)

```
int ch;
FILE * fp;
fp = fopen("wacky.txt", "r");
while (ch != EOF)    // первым используется неопределенное значение ch
{
    ch = getc(fp);    // получить ввод
    putchar(ch);      // обработать ввод
}
```

- 1) коли змінна `ch` вперше порівнюється з `EOF`, їй ще не присвоєно значення.
- 2) якщо `getc()` повертає `EOF`, то цикл намагається обробити `EOF` так, наче це допустимий символ.
 - Можна було надати `ch` фіктивне значення та помістити всередину циклу оператор `if`.
 - Проте є правильні проектні рішення.

Функція `fclose()`

- `fclose(fp)` закриває файл, що ідентифікується `fp`, за необхідності скидуючи буфери.
 - У захищеній програмі слід переконатись, що файл успішно закрито (функція поверне 0, інакше – EOF):
 - `if (fclose(fp) != 0)`
 `printf("Помилка при закритті файла %s\n", argv[1]);`
- Причини невдалого завершення: заповнено жорсткий диск, знімний пристрій зберігання вилучено або сталася помилка введення-виведення.

Файловий ввід-вивід: fprintf() та fscanf()

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #define MAX 41
5
6  int main(void)
7  {
8      FILE *fp;
9      char words[MAX];
10
11     if ((fp = fopen("wordy", "a+")) == NULL)
12     {
13         fprintf(stdout, "Can't open \"wordy\" file.\n");
14         exit(EXIT_FAILURE);
15     }
16
17     puts("Enter words to add to the file; press the #");
18     puts("key at the beginning of a line to terminate.");
19     while ((fscanf(stdin, "%40s", words) == 1) && (words[0] != '#'))
20         fprintf(fp, "%s\n", words);
21
22     puts("File contents:");
23     rewind(fp); /* go back to beginning of file */
24     while (fscanf(fp, "%s", words) == 1)
25         puts(words);
26     puts("Done!");
27     if (fclose(fp) != 0)
28         fprintf(stderr, "Error closing file\n");
29
30     return 0;
31 }
```

- Працюють аналогічно printf() і scanf(), проте мають додатковий перший аргумент – вказівник файла.
- Дана програма дозволяє додавати слова в файл.
 - З режимом "a+" може здійснюватись зчитування та запис у файл.
 - При першому запуску створюється файл wordy, програма дозволяє поміщати в нього слова по одному в рядку.
 - При подальшому запуску програма дозволяє дописувати слова до існуючого вмісту.

Довільний доступ

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define CNTL_Z '\032' /* eof marker in DOS text files */
4  #define SLEN 81
5  int main(void)
6  {
7      char file[SLEN];
8      char ch;
9      FILE *fp;
10     long count, last;
11
12     puts("Enter the name of the file to be processed:");
13     scanf("%80s", file);
14     if ((fp = fopen(file, "rb")) == NULL)
15     {
16         /* read-only mode */
17         printf("reverse can't open %s\n", file);
18         exit(EXIT_FAILURE);
19     }
20     fseek(fp, 0L, SEEK_END); /* go to end of file */
21     last = ftell(fp);
22     for (count = 1L; count <= last; count++) {
23         fseek(fp, -count, SEEK_END); /* go backward */
24         ch = getc(fp);
25         if (ch != CNTL_Z && ch != '\r') /* MS-DOS files */
26             putchar(ch);
27     }
28     putchar('\n');
29     fclose(fp);
30
31     return 0;
32 }
```

- Функція `fseek()` дозволяє трактувати файл як масив та переходити безпосередньо до будь-якого байту в файлі, відкритому за допомогою `fopen()`.
 - Програма відображає вміст файлу в зворотному порядку.
 - Функція `ftell()` повертає поточну позицію в файлі як значення типу `long`.
 - Застосовується двійковий режим, тому програма може мати справу як з текстовими файлами MS-DOS, так і з файлами Unix.
 - Проте її робота може виявитись некоректною в середовищі, в якому для текстових файлів використовується якийсь інший формат.

Робота функції fseek()

- Перший аргумент – вказівник FILE на файл, у якому буде виконано пошук.
 - Файл повинен бути відкритим за допомогою fopen().
- Другий аргумент – **зміщення**.
 - Показує, наскільки далеко (значення типу long) потрібно зміститись від початкової точки.
- Третій аргумент задає режим, що ідентифікує стартову позицію.
 - Починаючи зі стандарту ANSI, в заголовковому файлі stdio.h вказано іменовані константи для режимів:

Режим	Звідки вимірюється зміщення
SEEK_SET	Від початку файлу
SEEK_CUR	Від поточної позиції
SEEK_END	Від кінця файлу

```
fseek(fp, 0L, SEEK_SET); // перейти в начало файла
fseek(fp, 10L, SEEK_SET); // перейти на 10 байтов от начала файла
fseek(fp, 2L, SEEK_CUR); // перейти вперед на 2 байта от текущей позиции
fseek(fp, 0L, SEEK_END); // перейти в конец файла
fseek(fp, -10L, SEEK_END); // перейти назад на 10 байтов от конца файла
```

Робота функції ftell()

- Функція ftell() має тип long та повертає поточну позицію в файлі.
 - У стандарті ANSI C оголошена в stdio.h.
 - ftell() спочатку була реалізована в Unix, тому вона вказує позицію в файлі, повертаючи кількість байтів від його початку, причому перший байт отримує номер 0.
 - В ANSI C таке визначення застосовується до файлів, відкритих у двійковому режимі, проте не обов'язково до файлів, відкритих у текстовому режимі.
 - Тому застосовуємо двійковий режим.
- Функція ftell() може працювати по-різному в текстовому та двійковому режимах.
 - Формати текстових файлів багатьох ОС відрізняються від Unix.
 - У стандарті ANSI C стверджується, що при текстовому режимі ftell() повертає значення, яке може використовуватись у якості другого аргументу fseek().
 - Наприклад, в MS-DOS функція ftell() може повертати кількість байтів, при підрахунку якої комбінація `\r\n` розглядається як 1 байт.

Переносимість коду

- В ідеальному випадку `fseek()` та `ftell()` повинні відповідати моделі Unix.
 - Проте відмінності в реальних системах іноді роблять це неможливим.
 - Тому стандарт ANSI знижує очікування від цих функцій.
- Деякі обмеження:
 - У двійковому режимі реалізації не повинні підтримувати режим `SEEK_END`.
 - Більш переносимий підхід передбачає зчитування всього файлу байт за байтом, поки не зустрінеться кінець.
 - Проте послідовне зчитування повільніше за простий перехід у кінець файлу.
 - У текстовому режимі будуть гарантовано працювати тільки наступні виклики `fseek()`:

Вызов функции	Результат
<code>fseek(file, 0L, SEEK_SET)</code>	Перейти в начало файла
<code>fseek(file, 0L, SEEK_CUR)</code>	Оставаться в текущей позиции
<code>fseek(file, 0L, SEEK_END)</code>	Перейти в конец файла
<code>fseek(file, ftell_pos, SEEK_SET)</code>	Перейти в позицию <code>ftell_pos</code> от начала файла; <code>ftell_pos</code> — это значение, возвращаемое функцией <code>ftell()</code>

Функції `fgetpos()` та `fsetpos()`

- Потенційна проблема `fseek()` та `ftell()`: обмеження розміру файлів типом `long`.
 - В ANSI C з'явилися 2 нові функції позиціонування для роботи з крупними файлами.
 - Використовують новий тип `fpos_t`.
 - Змінна або об'єкт даних типу `fpos_t` може вказувати позицію всередині файлу та не може бути масивом.
- Функція ***fgetpos()*** має прототип:
 - `int fgetpos(FILE * restrict stream, fpos_t * restrict pos);`
 - Виклик `fgetpos()` поміщає поточне значення типу `fpos_t` у комірку, вказану в `pos` (позицію в файлі).
 - Функція повертає 0 при успіху та ненульове значення у випадку відмови.
- Прототип функції ***fsetpos()***:
 - `int fsetpos(FILE *stream, const fpos_t *pos);`
 - Виклик `fsetpos()` призводить до використання значення типу `fpos_t` з комірки, заданої за допомогою `pos` (установка вказівника файлу в дану позицію).
 - Функція повертає 0 при успіху та ненульове значення у випадку відмови.
 - Значення `fpos_t` повинно бути отримано попереднім викликом `fgetpos()`.



ДЯКУЮ ЗА УВАГУ!

Додаткові можливості описано в главі 13 (Прата, с. 548-558).