Лекция 24. Файловый ввод/вывод.

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

Курс «Программирование», весенний семестр, 2025
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

Содержание

- Функции: fopen(), getc(), putc(), fclose(), fprintf() , fscant(), fgets(), fputs(), rewind(), fseek(), ftell(), fflush(), fgetpos(), fsetpos(), feof(), terror(), ungetc(), setvbuf(), tread(), fwrite()
- Обработка файлов с использованием семейства стандартных функций ввода-вывода Си
- Текстовые и двоичные режимы, текстовый и двоичный форматы, буферизированный и небуферизированный ввод-вывод
- Применение функций, которые позволяют осуществлять последовательный и произвольный доступ в файлы

Введение

Файлы являются неотъемлемой частью современных компьютерных систем. Они используются для хранения программ, документов, данных, корреспонденции, форм, изображений, фотографий, музыкальных произведений, видеоклипов и несметного числа других видов информации. Программист должен уметь писать программы, которые создают файлы, записывают в файлы и читают из файлов.

Понятие файла (1)

Файл — это именованный раздел хранилища, обычно расположенный на HDD или SSD.

Для операционной системы файл выглядит не так просто. Крупный файл может храниться в нескольких *отдельных фрагментах* или содержать дополнительные данные, которые позволяют операционной системе определять вид этого файла (*метаинформация*).

Но за все это отвечает операционная система, а не программист.

Понятие файла (2)

В языке Си **файл** рассматривается как непрерывная последовательность байтов, каждый из которых может быть прочитан индивидуально.

Это соответствует файловой структуре в среде Unix, откуда Си берет свое начало.

Поскольку *другие среды* могут не соответствовать в точности этой модели, в Си предлагаются два способа представления файлов: **текстовый режим** и **двоичный** (*бинарный*) **режим**.

Понятие файла (3)

Перенаправление в файл

books > bklist

Перенаправление из файла

books < bklist

Этот метод *прост* и используется когда нет потребности другого взаимодействия с файлом.

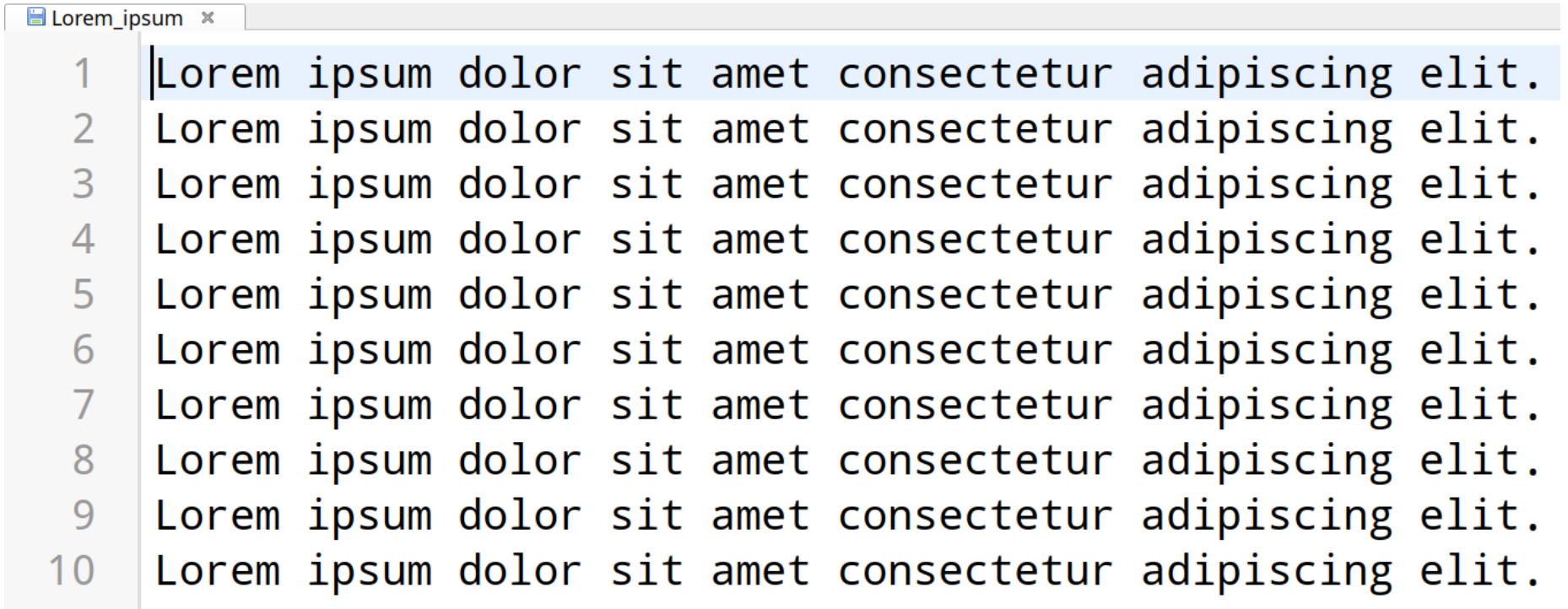
Представления файла (1)

Представления файлов:

- текстовое содержимое
- двоичное содержимое
- текстовый режим
- двоичный режим

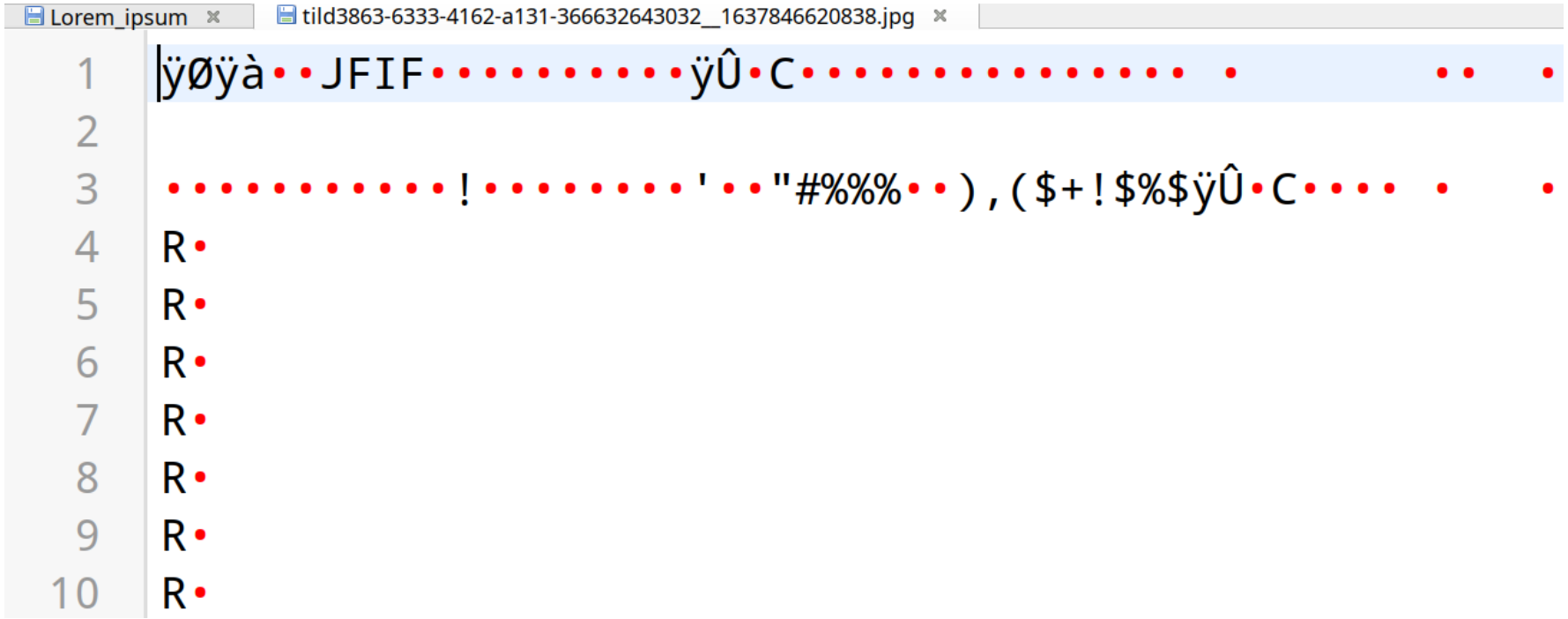
Представления файла (2)

Если в файле *двоичные коды* символов (к примеру, *ASCII* или *Unicode*) используются главным образом для представления текста, почти как в строках *Cu-style*, то такой файл является **текстовым**, т.е. имеет текстовое содержимое.



Представления файла (3)

Если двоичные значения в файле представляют код на *машинном языке*, числовые данные , кодировку изображения или музыкального произведения, то содержимое будет *двоичным*.



Представления файла (4)

Содержимое файла в системе Unix и для языка Си имеет один и тот же формат с точки зрения представления, содержимое всех файлов хранится в двоичной форме (нули и единицы).

Однако отличительной особенностью в различных ОС является символ переноса строки:

- Unix "\n"
- Macintosh до выхода OS X "\r" (в Unix это возврат каретки)
- MS-DOS "\r\n" (в Windows, Блокнот сохраняет в формате MS-DOS

Представления файла (5)

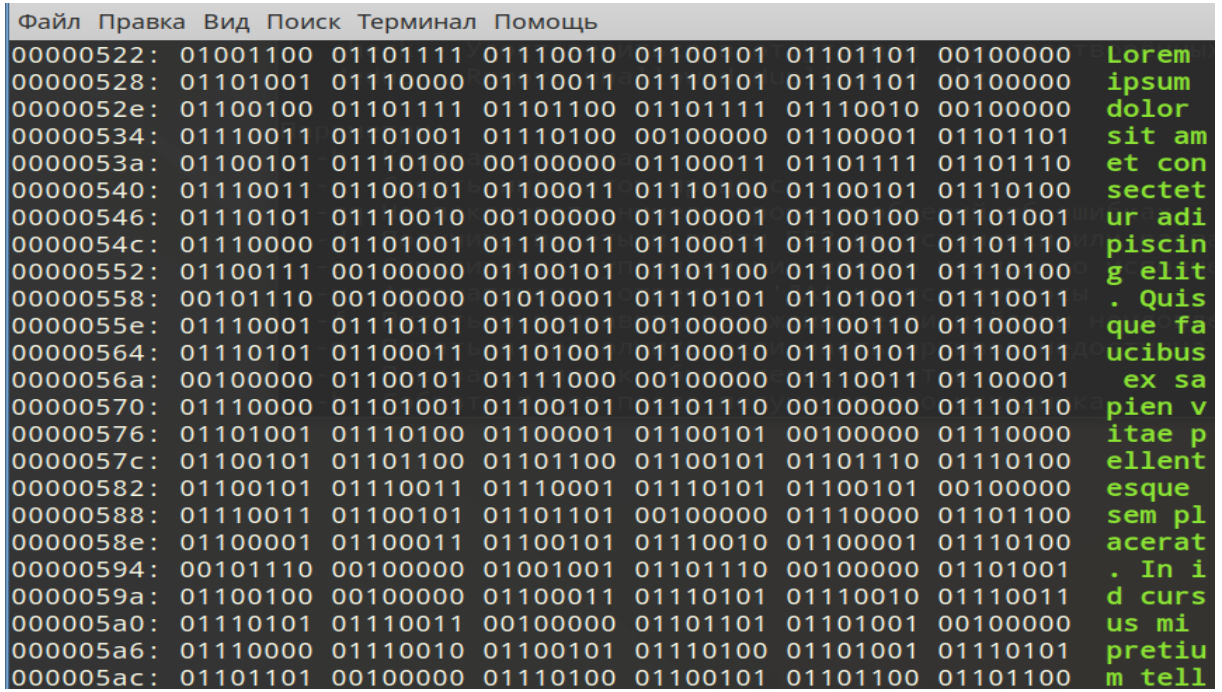
Большая часть редакторов после перехода на **Unicode** сформировала унифицированные требования к *символам переноса строки*.

Однако файловая система принимает решение о том как будет выглядеть файл после сохранения.

Может дополнять строки нуливыми символами до нужного размера, чтобы хранить файл в файловой системе или хранить длины строк в начале строки и т. п.

Представления файла (6)

В **двоичном режиме** программе доступен каждый байт файла. (всё представлено машинным кодом)



Можно использовать ***xxd -b filename*** в терменале Unix, чтобы увидеть двоичное представление

Представления файла (6)

В **текстовом режиме** то, что видит программа, может отличаться от того, что хранится в файле.

В **текстовом режиме** при чтении файла представление локальной среды для таких символов, как *конец строки* или *конец файла*, сопоставляется с их представлением в Си.

Представления файла (7)

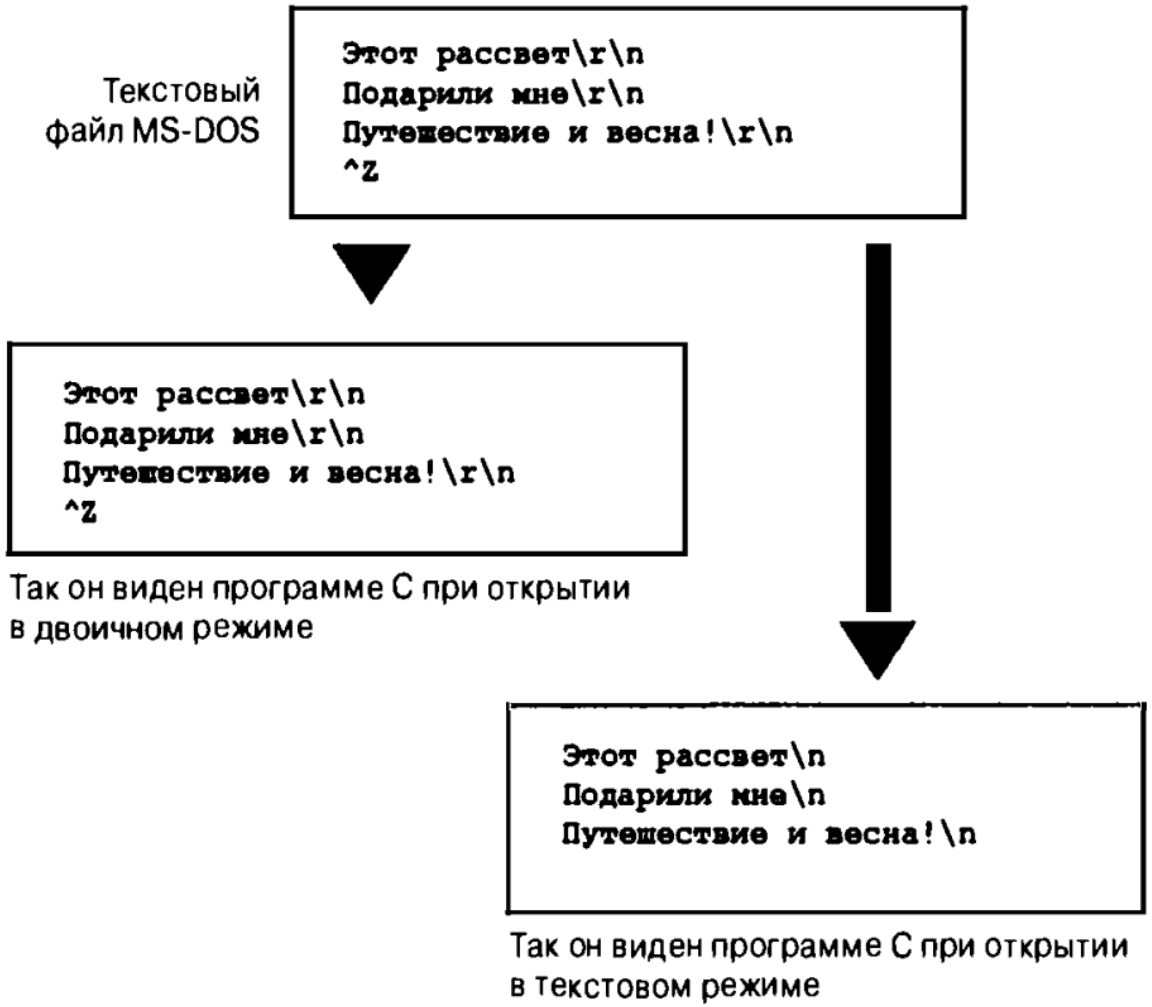


Рис. 13.1. Двоичное и текстовое представления

Уровни файлового ввода/вывода (1)

Различают два уровня файлового ввода/вывода:

Низкоуровневый ввод/вывод предусматривает использование основных служб ввода/вывода, предоставляемых операционной системой.

Высокоуровневый ввод/вывод предполагает применение **стандартного пакета** библиотечных функций Си и определений из заголовочного файла *stdio.h*.

Стандарт Си поддерживает только **стандартный пакет** ввода-вывода, т.к. нет никакой возможности гарантировать, что все операционные системы могут быть представлены одинаковой низкоуровневой моделью ввода-вывода.

Отдельные реализации могут также предлагать низкоуровневые библиотеки, но они не гарантируют показатель переносимости исходного кода.

Уровни файлового ввода/вывода (2)

Каждый запуск программы на Си автоматически открывает три файла:

- стандартным вводом
- стандартным выводом
- стандартным выводом ошибок

С точки зрения системы стандартный ввод это клавиатура вашего компьютера, а стандартный вывод его экран.

Уровни файлового ввода/вывода (3)

Стандартный ввод **-** обеспечивает ввод данных с клавиатуры Это файл, который читается с помощью функций ***getchar(), scanf()*** и подобными.

Стандартный вывод - место, куда направляется обычный вывод программы. Он используется функциями ***putchar(), puts(), printf()*** и подобными.

Стандартный вывод ошибок – предназначен чтобы предоставить логически обособленное место для отправки сообщений об ошибках.

Уровни файлового ввода/вывода (4)

По сравнению с ***низкоуровневым вводом/выводом*** стандартный пакет ***ввода/вывода***, помимо переносимости, обладает еще двумя преимуществами.

Во-первых, в нем доступно множество специализированных функций, которые упрощают решение разнообразных задач, связанных с ***вводом/выводом***.

Например, функция ***printf()*** преобразует различные формы данных в строковый вывод, подходящий для терминалов.

Уровни файлового ввода/вывода (5)

Во-вторых, ***ввод/вывод*** являются буферизированными. Это значит, что информация передается крупными порциями (обычно по 512 и более байтов), а не по одному байту за раз.

Например, когда ***программа читает файл***, порция данных считывается в буфер — промежуточную область памяти. Такая буферизации существенно ***увеличивает скорость передачи данных***. Затем программа ***может исследовать отдельные байты*** в этом буфере.

Буферизация происходит “за кулисами”, поэтому ***создается иллюзия посимвольного доступа***.

Можно также буферизировать низкоуровневый ввод/вывод, но большую часть работы придется проделать вручную.)

Функция fopen() (1)

Функции ***fopen()*** открывается файл. Эта функция объявлена в заголовочном файле ***stdio.h.***

Ее первым аргументом является ***имя файла*** – это адрес строки, содержащей имя файла.

Второй аргумент — строка, ***идентифицирующая режим***, в котором файл должен быть открыт.

После успешного открытия файла функция ***fopen()*** возвращает указатель файла, который затем другие функции ввода/вывода могут использовать для указания этого файла.

Функция fopen() (2)

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

Функция fopen() (3)

```
FILE * fp;
fp = fopen("wacky.txt", "r");
```

Указатель файла (в примере это `fp`) имеет тип указателя на `FILE`; здесь `FILE` - производный тип, определенный в ***stdio.h***. `fp` не ссылается на действительный файл, он указывает на объект данных, содержащий информацию о файле, включая сведения о буфере, который применяется для файлового ввода-вывода.

Функция fopen() (4)

Функция `fopen()` возвращает нулевой указатель (также определенный в stdio.h), если ей не удается открыть файл. Когда указатель `fp` равен NULL, программа прекращает выполнение.

Функция `fopen()` может отказать (в открытии файла) из-за переполнения диска, отсутствия файла в искомом каталоге, недопустимого имени, ограничений доступа или аппаратной проблемы. Это лишь небольшая часть причин отказа, даже минимальные меры по отлавливанию ошибок могут иметь большое значение.

Функция getc() и putc() (1)

Функции `getc()` и `putc()` работают очень похоже на `getchar()` и `putchar()`.

Отличие заключается в том, что этим новым функциям потребуется указать, с каким файлом работать.

```
ch = getc(fp);
            //извлечение символа из файла
putc(ch, fp);
            //Запись символа в файл
```

Функция getc() и putc() (2)

```
int ch; FILE *fp;
unsigned long count = 0;
if (argc != 2) {
printf("Использование: %s имя файла\n",
                    argv[0]);
exit(EXIT_FAILURE);  }
if ((fp = fopen(argv[1], "r")) == NULL){
printf("Не удается открыть %s\n",
                              argv[1]);
exit(EXIT FAILURE); }
```

Функция getc() и putc() (3)

```
while ((ch = getc(fp)) != EOF)
{
    putc(ch, stdout) ;
    // то же, что и putchar (ch) ;
    count++;
}
fclose (fp);
printf("Файл %s содержит %lu
символов\n", argv[1], count);
```

Функция getc() и putc() (4)

```
while ((ch = getc(fp)) != EOF)
```

Функция `getc()` возвращает специальное значение `EOF`, если она пытается прочитать символ и обнаруживает, что достигнут конец файла.

Таким образом, программа Си выясняет, что она достигла конца файла, только после попытки чтения за концом файла.

В некоторых других языках программирования, предусмотрена специальная функция для проверки на предмет конца файла попыткой чтения.

Функция getc() и putc() (5)

```
int ch;
FILE * fp;
fp = fopen("wacky.txt", "r");
while (ch != EOF)
{
    ch = getc(fp);
    putchar(ch);
}
```

Функция getc() и putc() (5)

```
int ch;
FILE * fp;
fp = fopen("wacky.txt", "r");
while (ch != EOF)
{
    ch = getc(fp);
    putchar(ch);
}
```

Функция fclose() (1)

```
if (fclose(fp) != 0)
printf("Ошибка при закрытии файла %s\n",
                                argv[1]);
```

Функция `fclose(fp)` закрывает файл, идентифицируемый `fp`, при необходимости сбрасывая буферы. В более ответственной программе вы должны удостовериться, что файл закрыт успешно.

Функция `fclose()` возвращает значение 0, если файл был закрыт успешно, и `EOF` если нет.

Указатели на стандартные файлы (1)

Стандартный файл	Указатель файла	Обычное устройство
Стандартный ввод	`stdin`	Клавиатура
Стандартный вывод	`stdout`	Экран
Стандартный вывод ошибок	`stderr`	Экран

Все они имеют тип указателя на `FILE`, поэтому могут использоваться в качестве аргументов для стандартных функций ввода/вывода подобно `fp` в приведенном ранее примере.

Файловый ввод/вывод(1)

Для каждой функции ***ввода/вывода***, что были изучены ранее есть аналог функции ***файлового ввода/вывода***.

Функции ***файлового ввода/вывода*** `fprintf()` и `fscanf()` работают аналогично `printf()` и `scanf()`, отличаясь только наличием дополнительного первого аргумента, в котором идентифицируется подходящий файл.

Файловый ввод/вывод(2)

Функции ***файлового ввода/вывода*** `fprintf()` и `fscanf()`

```
FILE * fp; char words[41] ;
fp = fopen("wordy", "a+");
//Проверки
while ((fscanf(stdin,"%40s", words)==1)
                    && (words[0] != '#'))
    fprintf(fp, "%s\n", words);
… rewind(fp); …
```

Файловый ввод/вывод(3)

```
fgets(buffer, sizeof(buffer), stdin);
```

функция `fgets()` читает входные данные до появления первого символа новой строки (`"\n"`) до тех пор, пока не будет прочитано количество символов, на единицу меньше верхнего предела, либо пока не будет обнаружен конец файла; затем `fgets()` добавляет завершающий нулевой символ, что бы сформировать строку.

Файловый ввод/вывод(4)

```
fputs(buffer, stdout)
```

Функция `fputs()` принимает два аргумента: адрес строки и указатель файла.

Она записывает строку, находящуюся в указанной ячейке, в заданный файл. В отличие от `puts()`, функция `fputs()` при выводе не добавляет символ новой строки.

Файловый ввод/вывод(5)

```
if (fgets(buffer, sizeof(buffer), stdin)
                                    != NULL) {
 // Удаляем символ новой строки, если он есть
 size_t len = strlen(buffer);
 if (len > 0 && buffer[len - 1] == '\n’)
    buffer[len - 1] = '\0';
 fputs("Вы ввели: \"", stdout);
 fputs(buffer, stdout);
 }
else printf("Ошибка при чтении строки.\n");
```

Файловый ввод/вывод(6)

`fseek(FILE *stream, long offset, int whence)`

Функция `fseek()` позволяет трактовать файл подобно массиву и переходить непосредственно к любому байту в файле, открытом с помощью `fopen()`.

`SEEK_SET`: начало файла.

`SEEK_CUR`: текущая позиция указателя.

`SEEK_END`: конец файла.

Файловый ввод/вывод(7)

`ftell(FILE *stream)`

Функция `ftell()` возвращает текущую позицию в файле как значение long.

```
// Записываем большой объём данных
for (int i = 0; i < 1000000; i++) {
    fputc('A', file);
}
printf("Записано 1 миллион символов 'A'.\n");
// Получаем текущую позицию
long position = ftell(file);
printf("Текущая позиция: %ld\n", position);
```

Файловый ввод/вывод(8)

```
file = fopen("example.txt", "w+");
const char *text = "Hello, World!";
fputs(text, file);
 // Получаем текущую позицию указателя
long pos = ftell(file);
// Перемещаем указатель в начало файла
fseek(file, 0, SEEK_SET);
// Перемещаем указатель на 7-й байт
fseek(file, 7, SEEK_SET);
```

Спасибо за внимание

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

Курс «Программирование», весенний семестр, 2025
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)