Лекция 25. Файловый ввод/вывод. Часть 2

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(), ferror(), ungetc(), setvbuf(), fread(), 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.

Повторение (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.

Повторение (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**.

Повторение (3)

Представления файлов:

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

Двоичный VS текстовый режимы (1)

```
ftell(FILE *stream)
```

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

В **текстовом режиме**, работает относительно модели файловой системы (т.е. учитывает символы EOF), т.к порой подсчёт байтов не даёт осмысленной информации.

В **двоичном режиме**, работает идентично во всех файловых моделях, считает по байтам.

Двоичный VS текстовый режимы (2)

Функция `ftell()` и `fseek()` вне могут обеспечить одинаковую работу на всех системах, стандарт ANSI вводит ограничения:

- В двоичном режиме реализации не обязаны поддерживать режим `SEEK_END`. Более переносимый подход предусматривает чтение всего файла байт за байтом, пока не встретится `EOF`. Но последовательное чтение файла для нахождения конца медленнее, чем просто переход в его конец. *Директивы условной компиляции препроцессора обсуждаемые в лекции 21, предлагают более систематизированный способ для поддержки выбора альтернативного кода.*

Двоичный VS текстовый режимы (3)

- В текстовом режиме будут гарантированно работать только следующие вызовы `fseek()`:

1. `fseek(file, 0L, SEEK_SET)` — Перейти в начало файла
2. `fseek(file, 0L, SEEK_CUR)` — Оставаться в текущей позиции
3. `fseek (file, 0L, SEEK_END)` — Перейти в конец файла
4. `fseek(file, ftell_pos, SEEK_SET)` — Перейти в позицию `ftell_pos` от начала файла; `ftell_pos` — это значение, возвращаемое функцией `ftell()`

Функции fgetpos() и fsetpos() (1)

Одна потенциальная проблема с функциями `fseek()` и `ftell()` заключается в том, что они ограничивают размеры файлов значениями, которые могут быть представлены типом **long**.

Как тогда обрабатывать файлы за пределами 2 миллиардов символов?

Функции fgetpos() и fsetpos() (2)

Вместо этого созданы функции которые работают с новым специализированным форматом `fpos_t` (с англ. *file position type* – тип для позиции в файле). Переменная или объект может указывать на позицию внутри файла *(не может быть массивом)*. Тип `fpos_t` ***не является фундаментальным***, а определяется в терминах других типов. ***Реализация*** может предоставить какой-то тип, удовлетворяющий нуждам конкретной платформы;

Функции fgetpos() и fsetpos() (3)

```
int fgetpos(FILE * restrict stream,
            fpos_t * restrict pos);
```

Вызов `fgetpos()` помещает текущее значение типа `fpos_t` в ячейку, указанную `pos`; это значение описывает позицию в файле.

Функция возвращает ноль в случае успеха и ненулевое значение при отказе.

Функции fgetpos() и fsetpos() (4)

```
int fsetpos(FILE *stream,
            const fpos_t *pos);
```

Вызов `fsetpos()` приводит к использованию значения типа `fpos_t` из ячейки, заданной с помощью `pos`, для установки указателя файла в позицию, которую отражает это значение. Функция возвращает ноль в случае успеха и ненулевое значение при отказе.

Значение `fpos_t` должно было быть получено предыдущим вызовом `fsetpos()`.

Функции fgetpos() и fsetpos() (5)

```
FILE *file = fopen("example.txt", "r");
fpos_t position;
for (size_t i = 0; i < 10; i++)
   int ch = fgetc(file);
fgetpos(file, &position);
while ((ch = fgetc(file)) != EOF)
   putchar(ch);
putchar('\n');
fsetpos(file, &position);
```

Функции fgetpos() и fsetpos() (6)

```
fgetpos(file, &position)//position = 10
```

Надвинулась ночь, и войско наконец остановилось; пять с лишним часов проскакали они, но не одолели еще и половины пути. Расположились на ночлег широким кругом, при свете звезд и тусклого месяца. EOF

```
fsetpos(file, &position);
```

Переосмысление fopen() (1)

Обычно первым шагом в применении стандартного ввода-вывода является вызов функции `fopen()` для открытия файла. Однако помимо файла настраиваются два буфера чтение/запись, а также обновляет структуру `FILE *` и заполняет ее информацией о файле. Помимо этого возвращается указатель на эту структуру, чтобы другие функции могли взаимодействовать с ней.

Переосмысление fopen() (2)

Если файл открыт в текстовом режиме, будет получен текстовый поток, в двоичном – двоичный поток.

Эта структура данных обычно включает индикатор позиции в файле, предназначенный для определения текущей позиции в потоке. Она также содержит *индикаторы для ошибок* и *конца файла*, *указатель на начало буфера*, *идентификатор файла* и *счетчик количества байтов, действительно с копированных в буфер*.

Переосмысление fopen() (3)

После `fopen()` логичным является вызов одной из функций ввода например `fgets()`.
Вызов любой такой функции приводит к тому, что порция данных копируется из файла в буфер.
Размер буфера зависит от реализации, но обычно он имеет 512 или кратное этому числу количество байтов, такое как 4 096 или 16 384. *(По мере увеличения объемов памяти компьютера, размеры буферов также имеют тенденцию к росту)*

Переосмысление fopen() (4)

Вдобавок к заполнению буфера первоначальный вызов функции устанавливает значения в структуре, указываемой посредством `fp`.

В частности, устанавливается текущая позиция в потоке данных и количество байтов, скопированных в буфер. Обычно текущая позиция начинается с байта 0.

Переосмысление fopen() (5)

После инициализации структуры данных и буфера, функция ввода читает запрошенные данные из буфера. В результате индикатор позиции устанавливается так, чтобы указывать на символ, следующий за последним прочитанным символом. Поскольку все функции ввода из семейства stdio.h используют тот же самый буфер, вызов любой такой функции возобновляет чтение там, где оно было остановлено предыдущим вызовом любой из функций.

Переосмысление fopen() (6)

Когда функция ввода обнаруживает, что все символы из буфера прочитаны, она запрашивает ***копирование*** из файла в буфер следующей порции данных с объемом, равным размеру буфера. В такой манере функции ввода могут читать все содержимое вплоть до конца файла. После того, как функция прочитает последний символ финальной порции данных, она устанавливает индикатор конца файла в истинное значение. Следующий вызов любой функции ввода возвратит `EOF`.

Переосмысление fopen() (7)

Функции вывода работают в аналогичном стиле производят запись в буфер. Когда буфер заполняется, данные копируются в файл.

Другие функции ввода/вывода (1)

```
int ungetc(int c, FILE * fp)
```

Функция `ungetc()` заталкивает символ, указанный в `c`, обратно во входной поток. В случае заталкивания символа во входной поток он будет прочитан следующим вызовом стандартной функции ввода.

Если `ungetc()` используется с недопустимым символом (например, `EOF`), поведение не определено.

Другие функции ввода/вывода (2)

```c
FILE *file = fopen("example.txt", "r");
int ch;
ch = fgetc(file);
printf("Первый символ: %c\n", ch);
if (ungetc(ch, file) == EOF) {
    fclose(file);
    return 1;
}
```

Другие функции ввода/вывода (3)

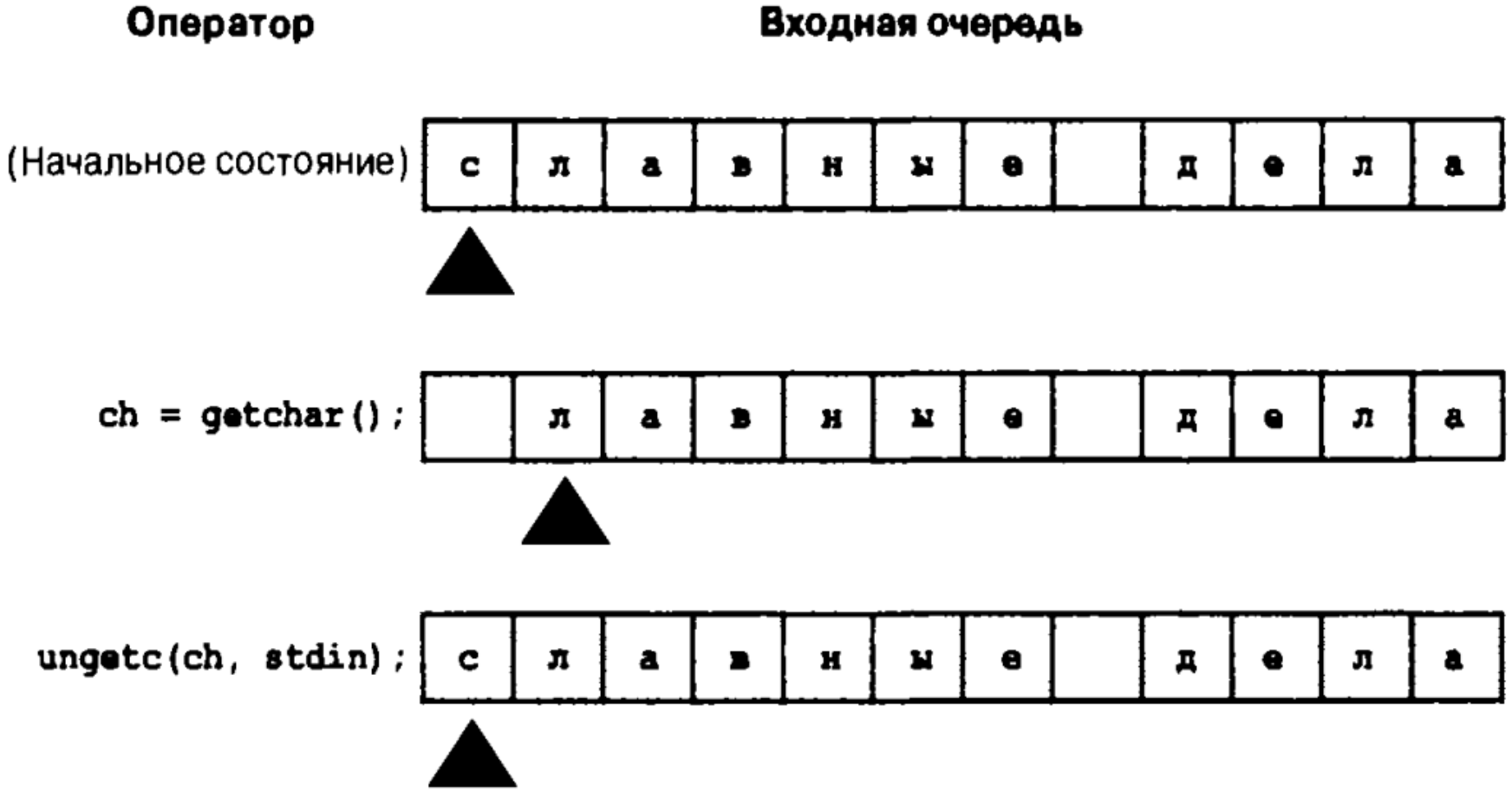

Рис. 13.2. Функция `ungetc()`

Другие функции ввода/вывода (4)

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

Вызов функции `fflush()` приводит к тому, что любые незаписанные данные в буфере вывода отправляются в выходной файл, идентифицируемый с помощью *fp*.

Этот процесс называется **сбросом буфера**.

Если *fp* — нулевой указатель, то сбрасываются все буферы вывода.

Результат использования функции `fflush()` на входном потоке не определен.

Другие функции ввода/вывода (5)

```
FILE *file = fopen("output.txt", "w");
if (file == NULL) {
   return 1;}
fprintf(file, "Привет, мир!\n");
// Данные записаны в буфер, но еще не
сохранены в файле
if (fflush(file) != 0) {
   fclose(file);
   return 1;}
```

Другие функции ввода/вывода (6)

```
int setvbuf(FILE * restrict fp,
            char * restrict buf,
            int mode, size_t size);
```

Функция `setvbuf()` устанавливает альтернативный буфер, предназначенный для применения стандартными функциями ввода-вывода. Она вызывается после того, как файл был открыт, и перед выполнением любой другой операции на потоке данных. Указатель *fp* идентифицирует поток, а *buf* указывает на используемое хранилище. Значение *buf*, не равное `NULL`, говорит о том, что буфер вы создаете самостоятельно.

Другие функции ввода/вывода (7)

```
int setvbuf(FILE * restrict fp,
            char * restrict buf,
            int mode, size_t size);
```

Для *mode* доступны следующие варианты:

_IOFBF — означает полную буферизацию (буфер сбрасывается, когда полон)

_IOLBF — построчную буферизацию (буфер сбрасывается, когда полон или когда в него записан символ новой строки)

_IONBF — отсутствие буферизации. Функция возвращает ноль при успешном завершении и ненулевое значение в противном случае.

Другие функции ввода/вывода (8)

```
size_t buffer_size = 100; //100 байт
char *buffer = malloc(buffer_size);
if (buffer == NULL)
  {fclose(file); return 1;}
if (setvbuf(file, buffer, _IOFBF,
buffer_size) != 0)
  {free(buffer);fclose(file);return 1;}
if (fflush(file) != 0)
  {free(buffer);fclose(file);return 1;}
```

Другие функции ввода/вывода (9)

```
size_t fwrite(const void * restrict ptr,
              size_t size, size_t nmemb,
              FILE * restrict fp);
```

Функция `fwrite()` записывает двоичные данные в файл. Тип `size_t` определен в терминах стандартных типов Си. Указатель `ptr` - это адрес порции данных, предназначенной для записи. Аргумент `size` представляет размер в байтах порции данных, подлежащих записи, а `nmemb` — количество таких порций.

Другие функции ввода/вывода (10)

```
size_t fread(void * restrict ptr,
             size_t size, size_t nmemb,
             FILE * restrict fp);
```

Функция `fread()` принимает такой же набор аргументов, как и `fwrite()`. На этот раз *`ptr`* представляет собой адрес области памяти, куда помещаются данные, прочитанные из файла, а *`fp`* идентифицирует читаемый файл. Эту функцию следует использовать для чтения данных, которые были записаны в файл с помощью `fwrite()` .

Другие функции ввода/вывода (11)

```
int numbers[] = {10, 20, 30, 40, 50};
int num_elements = sizeof(numbers) /
sizeof(int);
FILE *file = fopen("example.dat", "wb");
if (file == NULL) {return 1;}
if (fwrite(numbers, sizeof(int),
num_elements, file) != num_elements)
{ fclose(file);  return 1; }
fclose(file);
```

Другие функции ввода/вывода (12)

```
int numbers[] = {};int num_elements = 5;
file = fopen("example.dat", "rb");
if (file == NULL)
  { return 1; }
int read_numbers[num_elements];
if (fread(read_numbers, sizeof(int),
   num_elements, file) != num_elements)
{ fclose(file); return 1; }
fclose(file);
```

Другие функции ввода/вывода (13)

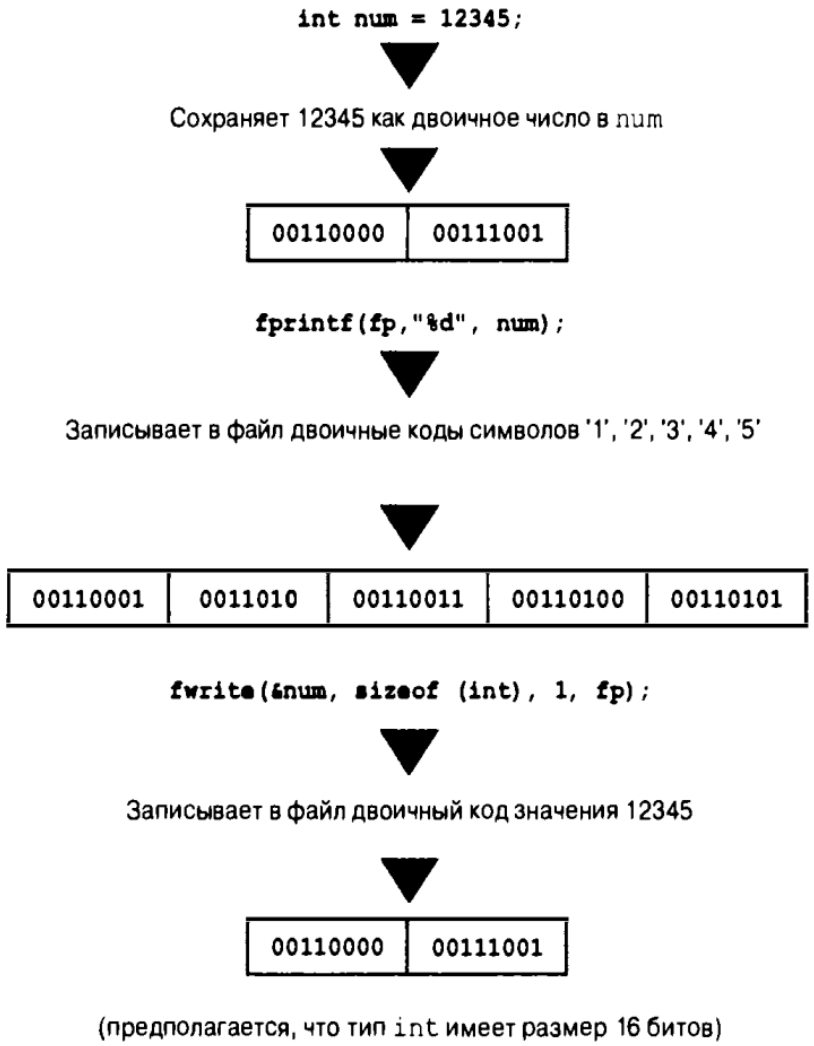

Рис. 13.3. *Двоичный и текстовый вывод*

Другие функции ввода/вывода (14)

```
int feof(FILE * stream);
int ferror(FILE * stream);
```

Когда стандартные функции ввода возвращают EOF, это обычно означает, что достигнут конец файла.

Тем не менее, возврат `EOF` может также указывать на возникновение ошибки чтения.

Функции `feof()` и `ferror()` позволяют проводить различие между этими двумя возможностями.

Другие функции ввода/вывода (15)

```
int feof(FILE * stream);
int ferror(FILE * stream);
```

Функция `feof()` возвращает ненулевое значение, если при последнем вызове функции ввода был обнаружен конец файла, и ноль в противном случае.

Функция `ferror()` возвращает ненулевое значение, если произошла ошибка чтения или записи, и ноль в противном случае.

Другие функции ввода/вывода (16)

```
file = fopen("example.txt", "r");
if (feof(file))
   printf("\nКонец файла достигнут.\n");
else {
   printf("\nПроизошла ошибка при чтении
файла.\n");
   fclose(file);
}
```

Другие функции ввода/вывода (17)

```
if (fprintf(file, "Привет, мир!") < 0 ||
                        ferror(file))
{fclose(file); return 1; }

if (fflush(file) != 0 || ferror(file))
{fclose(file); return 1; }

if (fclose(file) != 0 && ferror(file))
{ return 1; }
```

Спасибо за внимание

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

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