
Буферизированный и не буферизированный ввод вывод. Файловый ввод/вывод в поток. Смысл обнаружения конца файла. Обработка потока символов. Примеры.

В языке Си **буферизированный и небуферизированный ввод/вывод** определяют, как данные передаются между программой и устройствами ввода/вывода (например, консоль, файл, сеть).

Для понимания, напишем программу, которая будет выводить введенный текст (Эхо программы):

```
#include <stdio.h>
int main(void)
{
    char ch;
    while ((ch = getchar()) != '#')
        putchar(ch);
    return 0;
}
```

Буферизированный ввод

Буферизированный ввод - это ввод, при котором вводимые символы накапливаются и хранятся во временной памяти, которую называют *буфером*.

\$./a.out

Планируете[Enter]



Б у ф е р	П										
	л	П									
	А	л	П								
	н	а	л	П							
	и	н	а	л	П						
	...										
	\n	е	т	е	у	р	и	н	а	л	П

На картинке показан буферизированный ввод, и, если нажать Enter, то содержимое буфера передается программе.

Преимущества:

- Передача нескольких значений ввода в одном буфере быстрее чем отправка значений по одному.
- До нажатия Enter можно править значения передаваемые в буфер.

Небуферизированный ввод

Небуферизированный ввод - это ввод, при котором вводимые символы немедленно отправляются на обработку программы.

\$./a.out

ППлланнииррууетте



ВВ

ВЫ

ОВ

ДО

Д

На картинке введенный текст сразу же выводится на экран, не дожидаясь нажатия Enter или чего либо.

Преимущества:

- *Нажатие клавиши мгновенно обрабатывается программой, что может быть использовано при разработке интерактивных приложений или компьютерных игр.*

Виды буферизированного ввода

Существуют два вида буферизации ввода-вывода - *полностью буферизированный* и *построчно буферизированный*.

При *полностью буферизированном* вводе-выводе буфер отправляется в программу после того как буфер переполнится. Размер буфера зависит от системы и меняется в зависимости от запуска разрабатываемого приложения.

При *построчно буферизированном* вводе-выводе буфер сбрасывается всякий раз, когда появляется символ новой строки ('\n').

В ANSI C построчно буферизированный ввод считается стандартом.

Реализация не буферизированного и буферизированного ввода зависит от как *hardware* так и *software* составляющей

Файловый ввод-вывод в поток

Работа с файлами в языке Си основана на **потоках**. Потоки — это нечто абстрактное, что позволяет работать с вводом-выводом независимо от физического устройства.

Основные этапы работы с файлами

1. Открытие файла

- Используется функция `fopen()`, которая связывает файл с потоком.
- Сигнатура: `FILE *fopen(const char *filename, const char *mode);`
- Параметры:
 - `filename` - имя файла
 - `mode` - режим работы с файлом:
 1. `"r"` - чтение
 2. `"w"` - запись
 3. `"a"` - добавление в конец файла
 4. `"r+"` - чтение и запись (файл должен существовать)
 5. `"w+"` - чтение и запись (файл создается или перезаписывается)
 6. `"a+"` - чтение и запись, добавление в конец файла.
- Пример:

```
FILE *file = fopen("example.txt", "r");
if (file == NULL)
{
    perror("Ошибка открытия файла");
    return 1;
}
```

2. Чтение из файла

- Основные функции:
 1. `fgetc(FILE *stream)` - считывает один символ.
 2. `fgets(char *str, int n, FILE *stream)` - считывает строку.
 3. `fread(void *ptr, size_t size, size_t count, FILE *stream)` - читает блок данных.
- Примеры:

```
// Считываем символ за символом
char ch;
while ((ch = fgetc(file)) != EOF) {
    putchar(ch);
}
```

```
// Считываем строку
char buffer[100];
if (fgets(buffer, sizeof(buffer), file)) {
    printf("Прочитано: %s", buffer);
}
```

3. Запись в файл

- Основные функции:
 1. `fputc(int char, FILE *stream)` - записывает один символ.
 2. `fputs(const char *str, FILE *stream)` - записывает строку.
 3. `fwrite(const void *ptr, size_t size, size_t count, FILE *stream)` - записывает блок данных.
- Примеры:

```
FILE *file = fopen("output.txt", "w");
if (file) {
    fputc('A', file);           // Записываем символ
    fputs("Привет, мир!\n", file); // Записываем строку
    fclose(file);
}
```

4. Заккрытие файла

- Функция `fclose(FILE *stream)` завершает работу с файлом, освобождая связанные ресурсы.
- Пример: `fclose(file)`
Прочее:
- Функция `perror(const char *msg)` выводит сообщение об ошибке.
- Функция `feof(FILE *stream)` проверяет, достигнут ли конец файла.

Смысл обнаружения конца файла

Конец файла - невидимый символ, который обозначает конец потока данных.

Обнаруживать конец файла необходимо при:

1. чтении данных из файла

2. обработке потоков данных

Пример на чтение файла по строкам:

```
FILE *file = fopen("example.txt", "r");
if (file == NULL) {
    perror("Ошибка открытия файла");
    return 1;
}

char line[256];
while (fgets(line, sizeof(line), file) != NULL) {
    printf("%s", line);
}

fclose(file);
```

В цикле `while` мы делаем условие, что нужно читать строки, пока не встретим `NULL` - то есть, строки не будет.

Итог:

Обнаружение конца файла позволяет:

- Остановить чтение данных корректно.
- Избежать ошибок из-за попытки чтения после конца файла.
- Повысить надёжность работы с потоками.

Обработка потока символов

Обработка потока символов - это работа с данными, которые передаются в виде последовательности символов, обычно через стандартные потоки (`stdin`, `stdout`, `stderr`) или файлы. Это всё есть в библиотеке `<stdio.h>`.

Функции для обработки потока символов:

1. Чтение символов

- `getchar()`
 - Читает один символ из потока ввода `stdin`.
 - Возвращает код символа или `EOF` если достигнут конец файла.
 - Пример:

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

```
printf("Введите текст:\n");
int ch;
while ((ch = getchar()) != EOF) // Читаем символы
    putchar(ch); // Выводим прочитанный символ
return 0;
}
```

- `fgetc(FILE *stream)`

- Считывает один символ из указанного потока (`stream`).
- Возвращает `EOF` , если достигнут конец файла или произошла ошибка.
- Пример:

```
FILE *file = fopen("input.txt", "r");
if (file) {
    int ch;
    while ((ch = fgetc(file)) != EOF)
        putchar(ch); // Выводим символы из файла
    fclose(file);
}
```

```
PS C:\Users\user\Desktop\coding\c> gcc main.c
PS C:\Users\user\Desktop\coding\c> ./a.exe
Привет я абобус!
PS C:\Users\user\Desktop\coding\c> █
```

2. Запись символов

- `putchar(int char)`

- Записывает символ в стандартный вывод (`stdout`).
- Возвращает записанный символ или `EOF` при ошибке.
- Пример:

```
putchar('A'); // Выведет: A
```

- `fputc(int char, FILE *stream)`

- Записывает указанный символ в поток
- Пример:

```
FILE *file = fopen("output.txt", "w");
if (file) {
    fputc('Z', file);
    fputc('V', file);
}
```

```
fclose(file);  
}
```

3. Чтение строк

- `fgets(char *str, int n, FILE *stream)`
 - Читает строку из потока, включая символ новой строки (`\n`), и добавляет символ (`\0`) для завершения строки.
 - Прекращает чтение, если:
 1. Встречен символ новой строки.
 2. Прочитано `n-1` символов.
 3. Достигнут конец файла.
- Пример:

```
char buffer[100];  
FILE *file = fopen("input.txt", "r");  
if (file) {  
    while (fgets(buffer, sizeof(buffer), file) != NULL) {  
        printf("Прочитано: %s", buffer);  
    }  
    fclose(file);  
}
```

4. Запись строк

- `fputs(const char *str, FILE *stream)`
 - Записывает строку в указанный поток.
 - Не добавляет символ новой строки `\n` автоматически.
- Пример:

```
FILE *file = fopen("output.txt", "w");  
if (file) {  
    fputs("СИГМА СИГМА БОЙ!\n", file);  
    fclose(file);  
}
```


Изменение стандартных потоков `stdin` `stdout`

Для записи/чтения файла можно изменить место, откуда вводить и откуда выводить текст.

1. `stdin`

- При запуске программы нужно добавить символ `<`, который перенаправит поток ввода из стандартного (в терминал), в какой-либо файл. То есть, `stdin` теперь будет читать текст из файла.
- Пример: `a.exe < input.txt`

2. `stdout`

- Аналогично `stdin`, но нужно добавить `>`. Текст будет выводиться в файл.
- Пример: `a.exe > output.txt`