



# **РАБОТА С ФАЙЛАМИ ДАННЫХ**

**Дальневосточный государственный университет путей сообщения,  
кафедра «Вычислительная техника и компьютерная графика»,  
к.т.н., доцент Белозеров Олег Иванович.**

# Вопросы лекции:

1

Открытие и закрытие файлов

2

Функция бинарного чтения и записи

3

Форматный ввод и вывод

4

Дополнительные возможности

# В чем отличие СИ от СИ++?

**С++** — это улучшенный С.

У этих **языков** одинаковый на 99% синтаксис и команды, но С — это больше про структурное и процедурное программирование, а **С++** — про объектно-ориентированное программирование.

# 1. Открытие и закрытие файлов

Стандартная библиотека Си содержит набор функций для работы с файлами. Прототипы функций ввода-вывода и используемые для этого типы данных описаны в стандартном заголовочном файле (для Си++).

# Открытие файла

**FILE \*fopen(const char \*path, const char \*mode);**

path – путь к файлу (например, имя файла или абсолютный путь к файлу),

mode – режим открытия файла.

## Режимы открытия

r – Открыть существующий файл на чтение.

w – Открыть файл на запись. Старое содержимое файла теряется, в случае отсутствия файла он создаётся.

a – Открыть файл на запись. Если файл существует, то запись производится в его конец.

t – Открыть текстовый файл.

b – Открыть бинарный файл.

+ – Разрешить и чтение, и запись.

## Ошибки при открытии

- В случае выполнения функция `fopen` возвращает ненулевой указатель на структуру типа `FILE`.
- В случае ошибки возвращается нулевой указатель.
- Код ошибки помещается в глобальную системную переменную `errno`.
- `errno` описана в стандартном заголовочном файле `errno.h` (Си) и `cerrno` (Си++).

- Макрос `errno` возвращает последний номер ошибки. Этот макрос заменяется на модифицируемое значение типа `int`, поэтому `errno` может изменять ход работы программы.
- При запуске программы `errno` инициализируется нулем. Некоторые функции стандартной Си-библиотеки могут изменить его значение на значение отличное от нуля, чтобы сигнализировать о возникновении ошибок.
- Вы также можете изменить значение `errno` или обнулить его, на ваше усмотрение.



В этом же заголовочном файле `<cerrno>` определены также, по крайней мере, следующие две константы со значениями отличными от нуля:

Константа	Описание
EDOM	Ошибка области допустимых значений: Некоторые математические функции работают только с определенными значениями, которые называют своей областью допустимых значений. Например, квадратный корень вычисляется только для неотрицательных чисел, поэтому функция <code>sqrt</code> устанавливает макрос <code>ERRNO</code> в <code>EDOM</code> , если вызывается с отрицательным аргументом.
ERANGE	Ошибка диапазона значений: Диапазон значений, которые могут быть представлены типами данных, ограничен. Например, математические функции, такие как <code>pow</code> , могут с легкостью вернуть значение, которое не поместится ни в одном типе данных. В этих случаях, <code>ERRNO</code> устанавливается в <code>ERANGE</code> .

В C++ ошибка всегда объявляется как макрос.

## Пример программы:

```
1// пример использования функции assert
2#include <iostream>                                // для оператора cout
3#include <cerrno>                                    // для макроса errno
4#include <cmath>                                     // для функции sqrt
5
6int main()
7{
8    sqrt(-1);
9    std::cout << "EDOM = " << EDOM
10               << "nerrno = " << errno << std::endl;
11    if (errno == EDOM)    // если значения макросов errno и EDOM равны
12        std::cout << "Ошибка области допустимых значений/n";
13    return 0;
14}
```

EDOM = 33

errno = 33

Ошибка области допустимых  
значений

## Примеры открытия

**FILE \*f, \*g, \*h;**

// 1. Открыть текстовый файл "abcd.txt" для чтения

**f = fopen("abcd.txt", "rt");**

// 2. Открыть бинарный файл

// "c:\Windows\Temp\tmp.dat"

// для чтения и записи

**g = fopen("c:/Windows/Temp/tmp.dat", "wb+");**

// 3. Открыть текстовый файл

// "c:\Windows\Temp\abcd.log"

// для дописывания в конец файла

**h = fopen("c:\\Windows\\Temp\\abcd.log", "at");**

"r"	Создает файл для чтения (по умолчанию файл открывается как текстовый).
"w"	Создает файл для записи (по умолчанию файл открывается как текстовый).
"a"	Дописывает информацию к концу файла (по умолчанию файл открывается как текстовый).
"rb"	Открывает двоичный файл для чтения.
"wb"	Создает двоичный файл для записи.
"ab"	Дописывает информацию к концу двоичного файла.
"r+"	Открывает файл для чтения/записи (по умолчанию файл открывается как текстовый).
"w+"	Создает файл для чтения/записи (по умолчанию файл открывается как текстовый).
"a+"	Дописывает информацию к концу файла или создает файл для чтения/записи (по умолчанию открывается как текстовый файл).
"r+b"	Открывает двоичный файл для чтения / записи.
"w+b"	Создает файл для чтения / записи.
"a+b"	Дописывает информацию к концу файла или создает двоичный файл для чтения.
"rt"	Открывает текстовый файл для чтения.
"wt"	Открывает текстовый файл для записи.
"at"	Дописывает информацию к концу текстового файла.
"r+t"	Открывает текстовый файл для чтения/записи.
"w+t"	Создает текстовый файл для чтения/записи.
"a+t"	Открывает или создает текстовый файл для чтения/записи.

# Заккрытие файла

## **int fclose(FILE \*f);**

- в случае успеха функция fclose возвращает ноль
- при ошибке – отрицательное значение.

Функция fclose() используется для закрытия потока, ранее открытого с помощью fopen(). Она сохраняет в файл данные, находящиеся в дисковом буфере, и выполняет операцию системного уровня по закрытию файла. Вызов fclose() освобождает блок управления файлом, связанный с потоком, и делает его доступным для повторного использования.

**FILE \*f;**

```
f = fopen("tmp.res", «wb»);
```

```
if (f == 0) {
```

```
    perror("Не могу открыть файл для записи");
```

```
    exit(1); // завершить работу программы с
```

```
//кодом 1
```

```
}
```

```
...
```

```
if (fclose(f) < 0) {
```

```
    perror("Ошибка при закрытии файла");
```

```
}
```

## 2. Функция бинарного чтения и записи

### Функция бинарного чтения

**size\_t fread(**

char \*buffer, // Массив для чтения данных

size\_t elemSize, // Размер одного элемента

size\_t numElems, // Число элементов для

//чтения

FILE \*f // Указатель на структуру FILE

);

- возвращает число прочитанных элементов
- может быть меньше, чем numElems

# Функция бинарной записи

```
size_t fwrite(  
    char *buffer, // Массив записываемых данных  
    size_t elemSize, // Размер одного элемента  
    size_t numElems, // Число записываемых  
    //элементов  
    FILE *f // Указатель на структуру FILE  
);
```

- возвращает число записанных элементов
- может быть меньше, чем numElems



```
FILE *f;  
double buff[100];  
size_t res;  
...  
//Пытаемся прочесть 100 вещественных  
//чисел из файла  
res = fread(buff, sizeof(double), 100, f);  
//res равно реальному количеству  
//прочитанных чисел
```

```
FILE *f;  
double buff[100];  
size_t num;  
...  
//Записываем 100 вещественных чисел  
//в файл  
res = fwrite(buff, sizeof(double), 100, f);  
//В случае успеха res == 100
```

### 3. Форматный ввод и вывод

#### Форматный ввод

int **fscanf**(FILE \*f, const char \*format, ...);

%d – целое десятичное число типа int (d - от decimal)

%lf – вещ. число типа double (lf - от long float)

%c – один символ типа char

%s – ввод строки. Из входного потока выделяется слово, ограниченное пробелами или символами перевода строки '\n'. Слово помещается в массив символов. Конец слова отмечается нулевым байтом.

возвращает число успешно считанных элементов

```
int n, m; double a; char c; char str[256];
```

```
FILE *f;
```

```
...
```

```
fscanf(f, "%d", &n); // Ввод целого числа
```

```
fscanf(f, "%lf", &a); // Ввод вещественного числа
```

```
fscanf(f, "%c", &c); // Ввод одного символа
```

```
fscanf(f, "%s", str); // Ввод строки (выделяется очередное  
//слово из входного потока)
```

```
fscanf(f, "%d%d", &n, &m); // Ввод двух целых чисел
```

```
int fprintf(FILE *f, const char *format, ...);
```

- возвращает число успешно записанных элементов

## Некоторые форматы вывода

`%d` – вывод целого десятичного числа

`%10d` – вывод целого десятичного числа, для записи числа отводится 10 позиций, запись при необходимости дополняется пробелами слева

`%lf` – вывод вещественного число типа `double` в форме с фиксированной десятичной точкой

`%.3lf` – вывод вещественного число типа `double` с печатью трёх знаков после десятичной точки

`%12.3lf` – вывод вещественного число типа `double` с тремя знаками после десятичной точки, под число отводится 12 позиций

`%c` – вывод одного символа

`%s` – конец строки, т.е. массива символов. Конец строки задается нулевым байтом

```
#include <stdio.h>
#include <math.h>
#include <string.h>
int main() {
    int n = 4, m = 6;
    double x = 2.;
    char str[256] = "Print test";
    FILE *f = fopen("tmp.dat", "wt");
    if (f == 0) {
        perror("Не могу открыть файл для записи");
        return 1;
    }
}
```

```
fprintf(f, "n=%d, m=%d\n", n, m);
fprintf(f, "x=%.4lf, sqrt(x)=%.4lf\n", x, sqrt(x));
fprintf(
    f, "Строка \"%s\" содержит %d символов.\n",
    str, strlen(str)
);
fclose(f);
return 0;
}
```



## 4. Дополнительные возможности

Функции текстового преобразования  
sscanf и sprintf

```
char txt[256] = "-135.76"; double x;  
sscanf(txt, "%lf", &x);
```

```
char txt[256]; int x = 12345;  
sprintf(txt, "%d", x);
```

## Посимвольный ввод-вывод

```
int fgetc(FILE *f);
```

- ввести символ из потока f

```
int fputc(int c, FILE *f);
```

- вывести символ в поток f

## Построковый ввод-вывод

`char *fgets(char *line, int size, FILE *f);`

- ввести строку из потока `f`

`char *fputs(char *line, FILE *f);`

- вывести строку в поток `f`

## Позиционирование в файле

int fseek(FILE \*f,

long offset, – смещение в байтах

int whence – способ отсчета смещения

);

- установить текущую позицию в файле

SEEK\_CUR – смещение отсчитывается от текущей позиции

SEEK\_SET – смещение отсчитывается от начала файла

SEEK\_END – смещение отсчитывается от конца файла

## Позиционирование в файле

```
long ftell(FILE *f);
```

- получить текущую позицию в файле `f`
- `-1` при ошибке

# Позиционирование в файле

```
int feof(FILE *f);
```


- проверить, достигнут ли конец файла
- возвращает истину при достижении конца файла
- ложь в ином случае

# Пример

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main()
4  {
5      FILE *S3;
6      int x, y;
7      system("chcp 1251");
8      system("cls");
9
10     printf("Введите число : ");
11     scanf("%d", &x);
12
13     S3 = fopen("S3.txt", "w");
14     fprintf(S3, "%d\n", x);
15     fclose(S3);
16
17     S3 = fopen("S3.txt", "r");
18     fscanf(S3, "%d", &y);
19     y += 5;
20     fclose(S3);
21
22     S3 = fopen("S3.txt", "a");
23     fprintf(S3, "%d\n", y);
24     fprintf(S3, "%d\n", y+1);
25     fclose(S3);
26
27     return 0;
28 }
```

Введите число : 7

-----  
Process exited after 5.455 seconds with return value 0  
Для продолжения нажмите любую клавишу . . .

 S3 – Блокнот

Файл Правка Формат Вид Справка

7

12

13

СПАСИБО ЗА ВНИМАНИЕ!