

## Лекция 21

# Потоковая обработка файлов на Си

---

Преподаватель Палехова Ольга Александровна,  
кафедра О7 БГТУ «Военмех»

# Файлы

**Файл** – это область во внешней памяти, имеющая имя.

**Действия** над файлом:

- **удаление**

```
int remove ( const char * );
```

0 – файл удален успешно

имя файла

- **переименование** и/или **перемещение**

```
int rename ( const char *, const char * );
```

0 – переименование  
успешно

старое имя файла

новое имя файла

Для перемещения файла нужно указать его полное имя в новом расположении

- **создание** файлов, **чтение** данных из них и **запись** данных в файлы осуществляется с помощью **ПОТОКОВ**.

# Потоки

---

Язык Си содержит средства **низкоуровневой** и **высокоуровневой** обработки файлов. Высокоуровневая обработка осуществляется с помощью **потоков**.

**!** **Поток** – это источник или получатель данных, связанный с устройством внешней памяти или с каким-то другим внешним устройством.

## Потоки:

- **текстовый** – это последовательность символов, объединенных в строки, разделенные символом ‘**\n**’ (в одних ОС один символ (*LF*, код 10), в других два символа (*CR LF*, коды 13 10)).
- **бинарный** – это последовательность непреобразованных байтов.

**!** Для использования потоков требуется включение заголовочного файла **stdio.h**

# Создание потока

Объявление потока:

```
FILE * имя_переменной;
```

структура, содержащая всю информацию, необходимую для управления потоком

Альтернативные названия: *поток, указатель на поток, файловая переменная, указатель на файл, файловый указатель, логический файл*

Когда программа начинает работу, уже открыты три потока: **stdin** (связан с клавиатурой), **stdout** и **stderr** (связаны с монитором).

Для открытия других потоков используется функция

```
FILE * fopen ( char * , char * );
```

имя файла

режим открытия

# Режимы открытия потоков

---

Основные режимы открытия:

! Основные режимы открытия создают **текстовые** потоки

- **"w"** (*write*) – создать файл и открыть для записи. Если файл не существует, он будет создан. Если файл существует, то он будет удален и создан новый файл;
- **"r"** (*read*) – открыть файл для чтения. Файл должен существовать;
- **"a"** (*append*) – открыть файл для дополнения. Если файл не существует, он будет создан. Если файл существует, запись будет осуществляться в конец файла.

! При невозможности открыть файл в указанном режиме функция *fopen()* возвращает **NULL**

# Режимы открытия потоков


---

Расширенные режимы открытия:

- **"w+"** – создать файл и открыть его для записи и чтения;
- **"r+"** – открыть файл для чтения и редактирования;
- **"a+"** – открыть файл для дополнения и чтения.

Для **бинарных** потоков режимы открытия дополняются буквой **b**:

- **"wb"**, **"rb"**, **"ab"**
- **"w+b"**, **"r+b"**, **"a+b"** или **"wb+"**, **"rb+"**, **"ab+"**

 При невозможности открыть файл в указанном режиме функция *fopen()* возвращает **NULL**

# Перенаправление и закрытие потоков

---

Существующий поток можно перенаправить на файл (обычно применяется к стандартным потокам):

```
FILE * freopen ( char *, char *, FILE * );
```

имя файла

режим открытия

имя потока

При невозможности открыть файл в указанном режиме функция *freopen()* возвращает **NULL**

После завершения работы с потоком его надо закрыть:

```
int fclose ( FILE * );
```

При штатном завершении программы все открытые потоки закрываются *автоматически*.

# Чтение данных из файла

## Функции ввода одного символа:

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

при достижении  
конца файла  
возвращают **EOF**

## Функция ввода строки:

```
char * fgets (char *, int, FILE *);
```

при достижении конца файла  
возвращает **NULL**

## Функция форматного ввода:

```
int fscanf (FILE *, const char *, ...);
```

при достижении конца файла  
возвращает **EOF**

## Функция прямого ввода:

```
int fread (void *, size_t, size_t, FILE *);
```

количество  
читанных  
значений

куда  
поместить  
читанное

размер  
одного  
данного

количество  
считываемых  
значений



# Запись данных в файл

## Функции вывода одного символа:

```
int fputc (int, FILE *);  
int putc (int, FILE *);
```

возвращают  
записанный символ  
или **EOF** при ошибке

## Функция вывода строки:

```
int fputs (char *, FILE *);
```

в случае ошибки  
возвращает **EOF**

## Функция форматного вывода:

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

в случае ошибки возвращает  
отрицательное число

## Функция прямого вывода:

```
int fwrite (void *, size_t, size_t, FILE *);
```

количество  
записанных  
значений

откуда взять  
данные для  
записи

размер  
одного  
данного

количество  
записываемых  
значений

# Текстовые файлы

**Задача 1:** В файл `input.txt` записано некоторое количество целых чисел. Записать в файл `output.txt` их сумму.

```
#include <stdio.h>
int main()
{
    int x, S = 0;
    if ( freopen("input.txt", "r", stdin) == NULL )
    {
        perror ("Ошибка открытия файла");
        return 1;
    }
    if ( freopen("output.txt", "w", stdout) == NULL )
    {
        perror ("Ошибка создания файла");
        return 2;
    }
    while ( scanf ( "%d", &x ) > 0 )
        S += x;
    printf ( "S = %d", S );
    return 0;
}
```

перенаправить поток *stdin* на файл и проверить! состояние

*perror()* выводит сообщение в поток *stderr*

перенаправить поток *stdout* на файл и проверить! состояние

читать числа, пока есть, что читать

записать результат

потоки закрываются автоматически

# Текстовые файлы

Другой вариант решения:

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    FILE *f;
```

```
    int x, S = 0;
```

```
    if ( (f = fopen ( "input.txt", "r" )) == NULL )
```

```
    {
```

```
        perror("Ошибка открытия файла");
```

```
        return 1;
```

```
    }
```

```
    while ( fscanf ( f, "%d", &x ) > 0 )
```

```
        S += x;
```

```
    fclose ( f );
```

```
    if ( (f = fopen ( "output.txt", "w" )) == NULL )
```

```
    {
```

```
        perror("Ошибка создания файла");
```

```
        return 2;
```

```
    }
```

```
    fprintf ( f, "S = %d", S );
```

```
    fclose ( f );
```

```
    return 0;
```

```
}
```

открыть файл для чтения и  
**проверить!** состояние

*perror()* выводит сообщение  
и пояснение по ошибке

читать числа, пока  
есть, что читать

закрывать файл

открыть файл для записи и  
**проверить!** состояние

записать данные

закрывать файл

# Текстовые файлы

**Задача 2:** Пронумеровать строки в заданном текстовом файле.

```
#include <stdio.h>
```

```
int main ()
```

```
{
```

```
FILE * f1, *f2;
```

```
char s[1000];
```

```
int x=0;
```

```
if ( (f1 = fopen("test.txt", "r")) == NULL )
```

```
{
```

```
    perror ("Error open file");
```

```
    return 1;
```

```
}
```

```
if ( (f2 = fopen("temp.txt", "w")) == NULL )
```

```
{
```

```
    perror ("Error open file");
```

```
    return 2;
```

```
}
```

```
while ( fgets (s, 1000, f1) )
```

```
{
```

```
    fprintf (f2, "%5d : ", ++x);
```

```
    fputs (s, f2);
```

```
}
```

```
fclose (f1);
```

```
fclose (f2);
```

```
remove ("test.txt");
```

```
rename ("temp.txt", "test.txt");
```

```
return 0;
```

```
}
```

открыть исходный  
файл для чтения

создать новый файл для  
замены исходного

читать построчно,  
пока есть, что читать

записать в новый файл  
пронумерованные строки

закрывать файлы

исходный файл удалить,  
новый переименовать

# Текстовые файлы

**Задача 3:** Дан файл, фразы в котором отделяются друг от друга точками, а слова - пробелами. Записать каждую фразу на отдельной строке.

```
#include <stdio.h>
int main ()
{
    FILE * f1, *f2;
    int c;
    if ( ( f1 = fopen("test1.txt", "r" ) ) == NULL )
    {
        perror ("Error open file"); return 1;
    }
    if ( ( f2 = fopen("temp.txt", "w" ) ) == NULL )
    {
        perror ("Error open file"); return 2;
    }
    while ( ( c = fgetc(f1) ) != EOF )
    {
        if ( c == '\n' ) fputc ( ' ', f2 );
        else
        {
            fputc (c, f2 );
            if ( c == '.' ) fputc ('\n', f2 );
        }
    }
    fclose (f1); fclose (f2);
    remove ("test1.txt");
    rename ("temp.txt", "test1.txt");
    return 0;
}
```

открыть исходный  
файл для чтения

создать новый файл  
для замены исходного

читать посимвольно,  
пока есть, что читать

закрывать файлы

исходный файл удалить,  
новый переименовать

# Возврат символа в поток ввода

**Задача 5:** В текстовом файле записано несколько строк целых чисел, разделенных пробелами. Подсчитать, сколько чисел содержится в каждой строке, и вывести эту информацию на экран.

## Алгоритм:

*count* = 0

цикл пока в файле есть числа

    прочитать число

*count* = *count* + 1

    считать все пробелы после числа

    если следующий символ LF или закончился файл,

        то вывести *count*, затем *count* = 0

    иначе вернуть последний считанный символ в поток ввода

конец цикла

## Возврат символа в поток ввода

```
int ungetc ( int, FILE * );
```

EOF в случае ошибки

возвращаемый  
символ

поток

# Возврат символа в поток ввода

**Задача 5:** В текстовом файле записано несколько строк целых чисел, разделенных пробелами. Подсчитать, сколько чисел содержится в каждой строке, и вывести эту информацию на экран.

```
#include <stdio.h>
int main()
{
    FILE *f;
    int x, c, count = 0, line = 0;
    if ( (f = fopen("int.txt", "r")) == NULL)
    {
        perror ("Error open file"); return 1;
    }
    while ( fscanf(f, "%d", &x) != EOF )
    {
        count++;
        while ( ( c = fgetc(f) ) == ' ' );
        if (c=='\n' || c==EOF)
        {
            printf("%d - %d\n", ++line, count);
            count = 0;
        }
        else ungetc(c, f);
    }
    fclose(f);
    return 0;
}
```

читать числа, пока  
есть, что читать

пропустить пробелы.  
Цикл завершится, когда в  
с будет '\n' или цифра

вернуть цифру  
в поток ввода

# Бинарные файлы

**Задача 1:** В файл `input.dat` записано некоторое количество целых чисел. Записать в файл `output.dat` их сумму.

```
#include <stdio.h>
int main()
{
    FILE *f;
    int x, S = 0;
    if ( (f = fopen ( "input.dat", "rb" )) == NULL )
    {
        perror("Ошибка открытия файла");
        return 1;
    }
    while ( fread (&x, sizeof(int), 1, f) )
        S += x;
    fclose (f);
    if ( (f = fopen ( "output.dat", "wb" )) == NULL )
    {
        perror("Ошибка создания файла");
        return 2;
    }
    fwrite (&S, sizeof(int), 1, f);
    fclose (f);
    return 0;
}
```

режим открытия с буквой **b**

читать без преобразования  
по одному блоку размера  
*sizeof(int)*

возвращается количество считанных блоков

режим открытия с буквой **b**

записать данные  
без преобразования



# Бинарные (двоичные) файлы

---

## Особенности:

- можно читать и записывать любой кусок памяти (просто байты);
- данные хранятся во внутреннем **машинном формате** (в текстовом редакторе не прочитать).

## Следствия:

- однотипные данные будут занимать одинаковые объемы памяти;
- создание, просмотр и редактирование файла выполняется специализированной программой.

# Файлы последовательного и произвольного доступа

---

В зависимости от возможности вычислить положение данного в файле:

- файлы **последовательного** доступа – положение данного **не может быть вычислено**;
- файлы **произвольного** доступа – положение данного **можно вычислить**.



Большинство текстовых файлов – файлы последовательного доступа.

Файлы **последовательного** доступа невозможно редактировать, **нужно создавать новый** файл (затем старый удалить, новый переименовать).

Файлы **произвольного** доступа **можно редактировать** на месте.

# Файлы последовательного и произвольного доступа

---

**Пример.** Есть файлы последовательного и произвольного доступа, в которые записано некоторое количество целых чисел ( $>3$ ). Вывести на экран третье число из каждого файла.

**Алгоритм** обработки файла **последовательного** доступа:

- открыть файл
- прочитать первое число
- прочитать второе число
- прочитать третье число
- вывести число на экран
- заккрыть файл

**Алгоритм** обработки файла **произвольного** доступа:

- открыть файл
- переставить указатель текущей позиции в файле на конец второго числа
- прочитать число
- вывести число на экран
- заккрыть файл.

# Позиционирование в файле

Установить указатель текущей позиции в потоке **на начало потока**:

```
void rewind (FILE *);
```

Установить указатель текущей позиции в потоке **на произвольный байт**:

```
int fseek (FILE *, long, int);
```

в случае успеха  
возвращает 0

смещение

точка отсчета

Точки отсчета:

- 0 – *SEEK\_SET* – начало потока
- 1 – *SEEK\_CUR* – текущая позиция в потоке
- 2 – *SEEK\_END* – конец потока

смещение  $\geq 0$

смещение  $\leq 0$

Получить текущую позицию в потоке **относительно начала**

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