

Севастопольский государственный университет
Кафедра информационных систем

Курс лекций по дисциплине

"АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ"
(АиП)

Лектор: Сметанина Татьяна Ивановна

Лекция 2

Потоки и файлы

Потоки и файлы

Файл – это именованная совокупность данных. Файлы хранятся на устройствах энергонезависимого хранения (но не обязательно).

Работа со внешними устройствами (клавиатура, дисплей) реализована в ОС как работа с файлами.

Приложению, запущенному в ОС, доступны потоки стандартного ввода и вывода (**stdin** – клавиатура, **stdout** – дисплей).

Современные операционные системы предоставляют возможность при запуске приложения перенаправлять потоки ввода/вывода. Это дает возможность вводить данные не из клавиатуры, а из файла, выводить данные не на экран, а в файл.

Все это происходит без изменения кода программ.

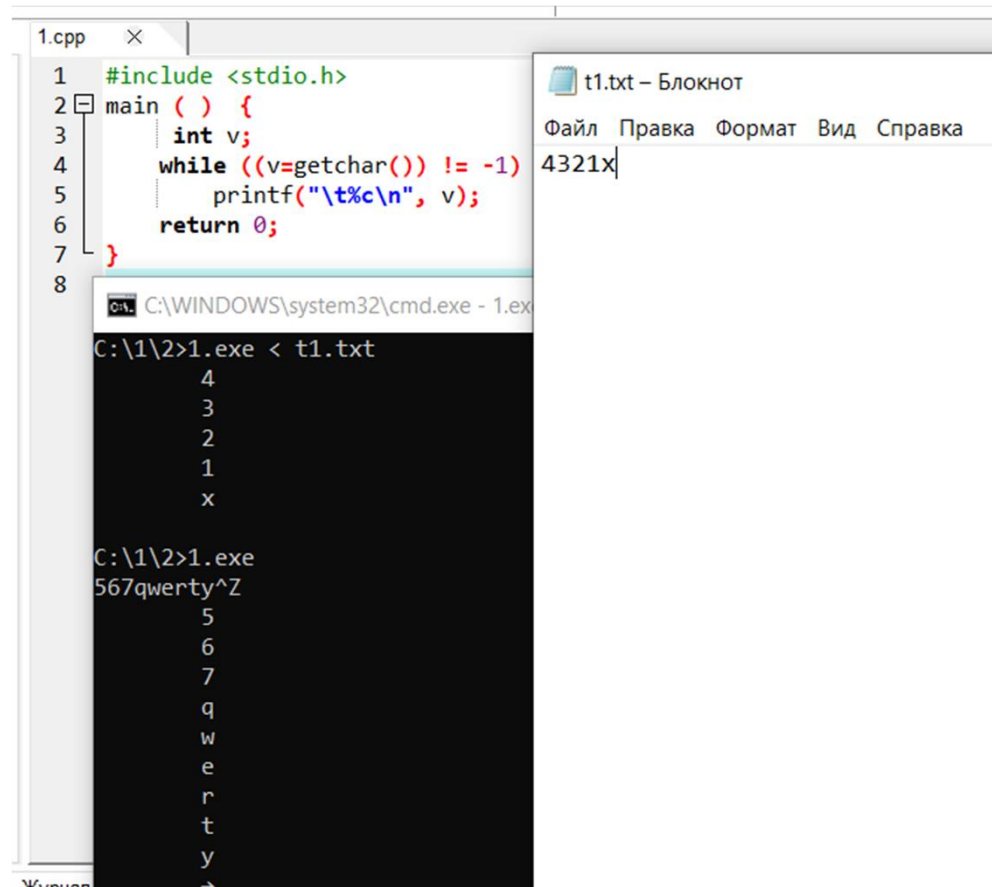
Повторим:

1. Стандартный ввод-вывод

Во многих системах **клавиатуру можно заменить файлом**, перенаправив ввод с помощью значка `<`. Так, если программа **prog** использует **getchar**, то командная строка

prog < infile

предпишет программе читать символы из файла **infile**, а не с клавиатуры.



The screenshot displays a C++ development environment with three windows:

- 1.cpp**: Contains the following code:

```
1 #include <stdio.h>
2 main ( ) {
3     int v;
4     while ((v=getchar()) != -1)
5         printf("\t%c\n", v);
6     return 0;
7 }
```
- t1.txt - Блокнот**: A Notepad window containing the text "4321x".
- C:\WINDOWS\system32\cmd.exe - 1.exe**: A command prompt window showing the execution of the program. It first runs `1.exe < t1.txt`, which outputs the characters 4, 3, 2, 1, and x on separate lines. Then, it runs `1.exe` without arguments, which outputs the characters 5, 6, 7, q, w, e, r, t, and y on separate lines.

Потоки и файлы

Т.е. поток ввода был перенаправлен с клавиатуры на файл.

Также можно перенаправить поток вывода, пример можно посмотреть в 13 лекции.

Перед тем как начать изучение работы с файлами в языке C, необходимо уяснить, в чем разница между потоками и файлами.

Обобщенное средство организации ввода или вывода называется потоком, в то время как конкретные данные на устройстве называются файлом. Очень важно понимать, каким образом происходит взаимодействие потоков и файлов.

Поток — это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику.

Система ввода/вывода C поддерживает единый интерфейс, не зависящий от того, к какому конкретному устройству или файлу осуществляется доступ. Доступ дает ОС через свою реализацию потоков данных.

Файлы

Работа с файловой системой (ввод/вывод) в языке C реализуется с помощью функций библиотеки **stdio.h**.

Файл – это именованный объект, хранящий данные любого типа на каком-либо носителе. Файлы используются для размещения данных, предназначенных для длительного хранения. Каждому файлу присваивается уникальное имя (используемое далее для обращения к нему), атрибуты файла (время создания, права доступа и т.д.).

Размер файла может меняться от 0 до всего доступного пространства. Перед работой с файлом его необходимо открыть, а после работы – закрыть.

Файлы

В языке С работа с файлами ведется через указатель на тип **FILE**.

Указатель на файл – это переменная, идентифицирующая конкретный дисковый файл (его адрес) и используемая для организации ввода/вывода в конкретный файл. Он указывает на структуру **FILE**, содержащую служебные сведения о файле, такие как имя файла, статус, указатель текущей позиции чтения/записи и другие.

Формат объявления указателя на файл следующий:

```
FILE * f1, * f2; // указатель на файл;
```

Для программиста открытый файл доступен для последовательного считывания и записи данных. При открытии файла ОС связывает с ним *поток ввода-вывода*. Выводимая информация записывается в поток, вводимая информация считывается из потока.

Открытие файла - `fopen`

Прежде чем производить действия с файлом (чтение, запись) файл должен быть открыт.

Функция **`fopen()`** открывает для использования поток, связывает файл с данным потоком и возвращает указатель на область памяти, где расположена структура типа `FILE` с заполненными полями. Функция **`fopen()`** имеет следующий прототип:

```
FILE *fopen(const char *имя_файла,  
             char *режим_открытия) ;
```

имя_файла — это указатель на строку символов, в которой хранится имя файла и путь к нему (не забываем, что `\` = `\\`).

режим_открытия — это указатель на строку символов, в которой указывается режим открытия файла.

Допустимые режимы:

r — открытие текстового файла для чтения;

w — создание текстового файла для записи;

a — добавление информации в конец текстового файла.

Открытие файла - **fopen**

При работе с текстовыми файлами к символу, указывающему режим открытия, добавляется символ «**t**» (по умолчанию), а при работе с бинарными — «**b**». Если необходимо и читать и записывать в файл, то добавляется символ «**+**». При возникновении ошибки во время открытия файла, функция **fopen** возвращает значение **NULL**

Пример:

```
FILE *f;  
if ((f = fopen("test.txt", "r")) == NULL) {  
    printf("Ошибка при открытии файла.\n");  
    return 1;  
}
```

Такая проверка помогает обнаружить ошибку, например, отсутствие файла на носителе, до попытки из этого файла что-либо прочитать. Всегда нужно вначале получить подтверждение, что функция **fopen()** выполнилась успешно, и лишь затем выполнять с файлом операции ввода и вывода.

Заккрытие файла - `fclose`

Функция **`fclose()`** разрывает связь указателя с файлом и закрывает его для дальнейшего использования, до тех пор пока он вновь не будет открыт. При этом сначала в файл записываются все данные, которые еще оставались в буфере, и выполняется закрытие файла на уровне операционной системы. Завершение работы с файлом без закрытия потока может привести к потере данных, порче файла и другим ошибкам. При завершении приложения все открытые им файлы закрываются ОС.

`int fclose(FILE * stream);`

где **`stream`** — указатель файла. Возвращение нуля означает успешную операцию закрытия. В случае же ошибки возвращается **`EOF`**.

Например:

`fclose(f);`

Заккрытие файла - fclose

Чтобы проверить наличие ошибки при последней операции с файлом, можно использовать функцию **ferror()**.

```
#include <stdio.h>
int main() {
    FILE * ptrFile = fopen("1.txt", "r");
    if (ptrFile == NULL) puts("Ошибка открытия файла");
    else {
        fputc('x', ptrFile);    // записать символ 'x'
        if ( ferror(ptrFile) ) // если ошибка
            puts("Ошибка записи файла 1.txt");
        fclose (ptrFile);      // закрыть файл
    }
    return 0;
}
```

```
int fcloseall(void);
```

Функция закрывает все открытые файлы. Возвращает количество закрытых файлов или EOF, если возникает ошибка.

Изменение режима доступа к файлу - **freopen**

Если требуется изменить режим доступа к файлу, то сначала необходимо закрыть данный файл, а затем вновь его открыть с другим режимом доступа. Для этого используют стандартную функцию **freopen**, описанную в **stdio.h** как

```
FILE* fopen (const char* file_name,  
             const char* mode, FILE *stream) .
```

Эта функция сначала закрывает файл, связанный с потоком **stream** (как это делает функция **fclose**), а затем открывает файл с именем **file_name** и режимом доступа **mode**, записывая файловый указатель в **stream**.

Изменение режима доступа к файлу - freopen

```
#include <stdio.h>
int main() {
    int x;
    FILE *f= fopen("1.txt","w"); //открытие
    if (f == NULL) {
        printf("Ошибка при открытии файла.\n");
        return (1);
    }
    fputc(65,f); //запись в файл 65
    freopen ("1.txt", "r", f); //переоткрытие файла
    if (f == NULL) {
        printf("Ошибка при повторном открытии файла");
        return (1);
    }
    x=fgetc(f); //чтение целого числа
    printf("x=%d",x); //печать на экран (?)
    fclose(f); //закрытие файла
    return 0;
}
```

Функции работы с файлами в С

Далее кратко остановимся на некоторых, наиболее важных функциях языка С, обеспечивающих работу с файлами по организации ввода и вывода информации. В таблице приведен список наиболее часто используемых из них.

Наименование	Назначение функции
<code>fopen()</code>	Открывает файл
<code>fclose()</code>	Закрывает файл
<code>putc()</code> , <code>fputc()</code>	Записывает символ в файл
<code>getc()</code> , <code>fgetc()</code>	Читает символ из файла
<code>fgets()</code>	Читает строку из файла
<code>fputs()</code>	Записывает строку в файл
<code>fseek()</code>	Устанавливает указатель текущей позиции на определенный байт файла
<code>ftell()</code>	Возвращает текущее значение указателя текущей позиции в файле

Функции работы с файлами в C

Наименование	Назначение функции
<code>fprintf()</code>	Для файла то же, что <code>printf()</code> для консоли
<code>fscanf()</code>	Для файла то же, что <code>scanf()</code> для консоли
<code>feof()</code>	Возвращает значение <code>true</code> если достигнут конец файла
<code>ferror()</code>	Возвращает значение <code>true</code> , если произошла ошибка
<code>rewind()</code>	Устанавливает указатель текущей позиции в начало файла и сбрасывает состояние ошибки в потоке
<code>remove()</code>	Удаляет файл
<code>fflush()</code>	Принудительная запись содержимого буфера в файл или устройство

Посимвольный вывод

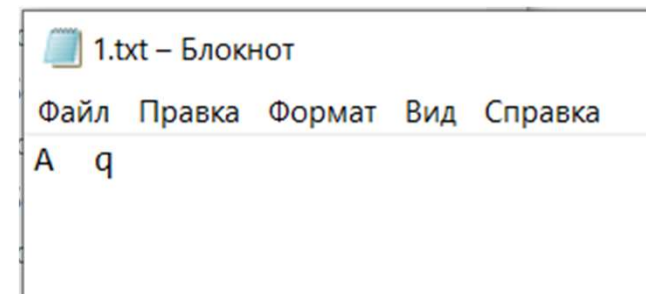
В языке C определены две эквивалентные функции, предназначенные для вывода (записи) символа в файл: **putc()** и **fputc()**.

Функция **putc()** записывает символ в открытый ранее файл для записи.

```
int putc(int ch, FILE* stream);
```

где **stream** – это указатель на файл, а **ch** – выводимый в него символ. В случае если запись выполнялась успешно, то возвращается записанный символ, иначе возвращается **EOF**.

```
FILE *f;  
if ((f = fopen("1.txt", "w")) == NULL) {  
    printf("Ошибка при открытии файла.\n");  
    return (1);  
}  
fputc(65, f); fputc(' ', f);  
putc(32, f);  putc('q', f);  
fclose(f);
```



Посимвольный ввод

Ввод (чтение) символа из файла, открытого для чтения, обеспечивается парой функций: `getc()` и `fgetc()`.

`int getc(FILE *stream) .`

Функция `getc()` возвращает целое значение, младший байт которого содержит **ASCII**– код символа. Старший байт при этом обнулен. При достижении конца файла, функция `getc()` возвращает **EOF**. Пример фрагмента программы чтения символов до конца текстового файла:

```
char ch;  
do  
{ ch = getc(f);    //чтение символа из файла  
  printf("%c",ch); //вывод на экран  
} while(ch!=EOF);  
  
//ошибка есть?
```

Функция `getc()` возвращает **EOF** и в случае ошибки.

```
// ch = fgetc(f);
```

Определение конца файла `feof()`

Как выше отмечалось при достижении конца файла функция `getc()` возвращает **EOF**. Однако **EOF** может быть возвращен и в случаях, когда конец файла еще не достигнут – при ошибке ввода. Для решения этой проблемы в **C** имеется функция `feof()`, которая определяет, достигнут ли конец файла.

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

Если конец файла достигнут, то `feof()` возвращает **true**, иначе **нуль**. Поэтому следующий код будет читать файл до тех пор, пока не будет достигнут конец файла:

```
while(1) {  
    ch = getc(f);  
    if (feof(f)) break;  
    printf("%c", ch);  
}
```

Организация ввода и вывода строк

Функции **fgets()** и **fputs()** обеспечивают построчный ввод и вывод информации. Первая читает строки символов из файла, а вторая записывает строки в файл.


```
int fputs(const char *str, FILE *stream);  
char *fgets(char *str, int len, FILE *stream);
```

Функция **fputs()** записывает в поток строку, на которую указывает **str**. В случае ошибки эта функция возвращает **EOF**.

Функция **fgets()** читает из потока строку, до тех пор, пока не будет прочитан символ новой строки или количество прочитанных символов не станет равным **len-1**. Если был прочитан разделитель строк '**\n**', он преобразуется в '**\0**' и записывается в строку. При успешном завершении работы функция возвращает **str**, а в случае ошибки — пустой указатель (**NULL**).

Организация ввода и вывода строк

Пример:

```
#include <stdio.h>
#include <string.h>
int main() {
char str[60];  FILE *f;
if((f = fopen("1.txt", "w"))== NULL) {
    printf("Ошибка при открытии файла.\n");
    return(1);
}
do {
    printf("Введите строку (  - для выхода) : \n");
    gets(str);
    if (strcmp(str, "")==0) break;
    strcat(str, "\n");
    fputs(str, f);
} while(1);
fclose(f);
return 0;
}
```

Организация ввода и вывода строк

Программа записывает в файл строки, считываемые с клавиатуры. Ввод в файл завершается вводом пустой строки.

Форматированный ввод-вывод

Функция **fprintf** преобразует, форматирует и печатает свои аргументы в указанный поток вывода под управлением строки формата. Возвращает количество напечатанных символов

```
int *fprintf(FILE * указатель_на_файл,  
             const char * управляющая_строка) ;
```

Управляющая_строка определяет строку форматирования аргументов, начинается знаком % и заканчивается символом-формата:

% [флаг] [ширина] [.точность] [h|l] символ_формата

где **ширина** — минимальное количество позиций, отводимых под выводимое значение,

точность — количество позиций, отводимых под дробную часть числа

Символ формата:

d - int;

f, (e, g) - float (double) ;

c - char;

s - строка символов .

Форматированный ввод-вывод

Функция **fscanf** читает символы из указанного входного потока, интерпретирует их согласно спецификациям строки **format** и рассылает результаты в свои остальные аргументы, которые являются указателями.

В качестве результата **fscanf** возвращает количество успешно введенных элементов данных. По исчерпанию файла она выдает **EOF**.

```
int *fscanf(FILE * указатель_на_файл,  
            const char * управляющая_строка) ;
```

Функция **fscanf** прекращает работу, когда оказывается, что исчерпался формат или вводимая величина не соответствует управляющей спецификации.

Пример:

Форматированный ввод-вывод

```
#include <stdio.h>
int main() {
    char x;
    FILE *in,*out;           // описание указателей на файлы
    in = fopen("c:\\1\\old.txt","rt");
    if (in == NULL) {
        printf(" Не могу открыть входной файл \n");
        return 1;}
    out = fopen("c:\\1\\new.txt","wt");
    if (out== NULL) {
        printf("Не могу открыть выходной файл \n");
        return 1;}

    while (1) {              // вечный цикл
        fscanf(in,"%c",&x);   // чтение
        if (feof(in)) break;  // выход из цикла
        fprintf(out,"%c",x);   // запись
    }
    fclose(in);
    fclose(out);
    return 0;
}
```


Удаление файлов

Функция **remove()** удаляет указанный файл. Прототип функции:

```
int remove(const char *file_name) .
```

В случае успешного выполнения эта функция возвращает нуль, иначе – ненулевое значение.

В фрагменте программы демонстрируется удаление файла, имя которого вводится с клавиатуры:

```
#include <stdio.h>
int main() {
    char str[80];
    printf("\n Введите имя файла для удаления ");
    gets(str);
    if(*str)
        if(remove(str)) {
            printf("\nНельзя удалить файл\n");
            return 1;
        }
        else printf("\nФайл успешно удален\n");
    return 0;
}
```

Позиционирование в файле

Каждый открытый файл имеет так называемый указатель на текущую позицию в файле. Все операции над файлами (чтение и запись) выполняются с данными с этой позиции. При каждом выполнении функции чтения или записи, указатель смещается на количество прочитанных или записанных байт, то есть устанавливается сразу за прочитанным или записанным блоком данных в файле. Это так называемый последовательный доступ к данным. Последовательный доступ удобен, когда необходимо последовательно работать с данными в файле. Но иногда необходимо читать или писать данные в произвольном порядке. Это достигается путем установки указателя на некоторую заданную позицию в файле функцией **fseek()**.

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

Параметр **offset** задает количество байт, на которое необходимо сместить указатель в направлении, указанном **whence**. Приводим значения, которые может принимать параметр **whence**:

Позиционирование в файле

SEEK_SET	0	Смещение выполняется от начала файла
SEEK_CUR	1	Смещение выполняется от текущей позиции указателя
SEEK_END	2	Смещение выполняется от конца файла

Величина смещения может быть как положительной, так и отрицательной, но нельзя сместиться за пределы начала файла.

Наряду с функцией **fseek()**, для перемещения указателя текущей позиции в файле на начало может быть использована функция **rewind()**. Кроме того эта функция выполняет сброс состояния ошибки в потоке, если она возникла при работе с файлом.

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

Функция устанавливает указатель текущей позиции выделенного файла в начало файла.

Макросы для работы с файлами

Макрос **NULL** определяет значение пустого указателя.

Макрос **EOF**, часто определяемый как **-1**, является значением, возвращаемым тогда, когда функция ввода пытается выполнить чтение после конца файла.

Макрос **FOPEN_MAX** определяет целое значение, равное максимальному числу одновременно открытых файлов.

Стандартные потоки ввода-вывода

Функции для посимвольного ввода и вывода: **getc** и **putc**, считывают из потока и записывают в поток ровно один символ. Поток может иметь конец. Для обозначения конца потока используется специальный символ EOF.

Рассмотрим на примере:

```
#include <stdio.h>
int main () {
    int c;
    while ( (c = getc(stdin)) != EOF ) {
        putc(c, stdout);
    }
    return 0;
}
```

Стандартные потоки ввода-вывода

Программа считывает символы из стандартного потока ввода и печатает их в стандартный поток вывода. Если запустить и ввести набор символов, то программа просто повторит их. Программа работает с входными данными построчно.

При вводе строки «**Hello world**» будет выведено:

Hello world

При работе с потоками ввода и вывода символы накапливаются в специальном буфере, осуществляя ввод и вывод только по достижению конца строки (символа ' \n ') или по заполнению буфера. Данная программа будет повторять введенные строки, пока на вход не поступит сигнал конца потока **EOF**.

Его можно ввести с клавиатуры с помощью комбинации **Ctrl + z**.

Важно понимать, что потоки ввода и вывода независимы, несмотря на то, что набираемые символы и символы, выводимые программой, отображаются рядом в одном окне.

Потоки и файлы

Поток можно представить как последовательность символов, доступной нам только с одного конца.

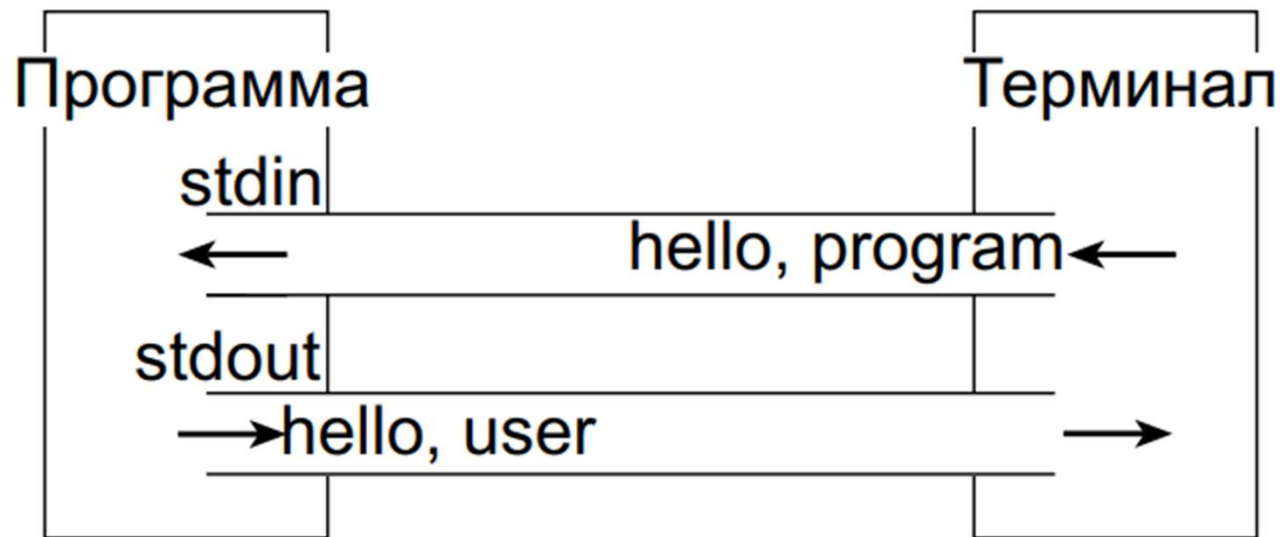
Его можно сравнить с концом длинной трубы, другой конец которой нам не виден. С потоком можно производить следующие действия:

- открыть,
- положить символ (запись),
- взять символ (чтение),
- протолкнуть,
- закрыть.

Каждый раз, когда мы кладём символ в поток, все остальные символы продвигаются к другому концу потока, но, по достижении ими конца не вываливаются из него.

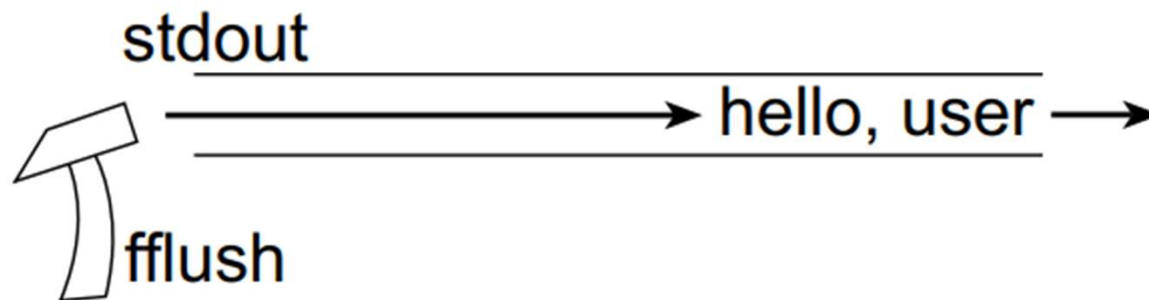
Потоки и файлы

Поток – это один конец трубы, используемый либо для ввода, либо для вывода данных, либо для того и другого одновременно. Потоки **stdin** и **stdout** являются стандартными потоками ввода и вывода программы.



Потоки и файлы

Команда **fflush(stdout)** вызывает «проталкивание» введённых данных до конца «трубы». Обычно потоки настроены так, что при появлении в входе символа перевода строки автоматически происходит «проталкивание»(**auto flush**).



При этом «труба» может «забиться», и придётся немного подождать, пока «труба» не освободиться для ввода следующих символов. И наоборот, забирая символ из потока, остальные символы продвигаются к концу.

Дозапись потока

Для дозаписи содержимого выводного потока в файл применяется функция **fflush()**:

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

Все данные, находящиеся в буфере записываются в файл, который указан с помощью **stream**. При вызове функции **fflush()** с пустым (**NULL**) указателем файла **stream** будет выполнена дозапись во все файлы, открытые для вывода.

При успешном выполнении **fflush()** возвращает нуль, иначе — **EOF**.

Функция **fflush** сбрасывает в связанный с потоком данных файл данные, находящиеся в буфере.

Если аргумент **stream** имеет значения:

- **NULL**, то сбрасываются буферы всех открытых в данный момент потоков данных;
- **stdout**, то сбрасывается буфер стандартного потока вывода.

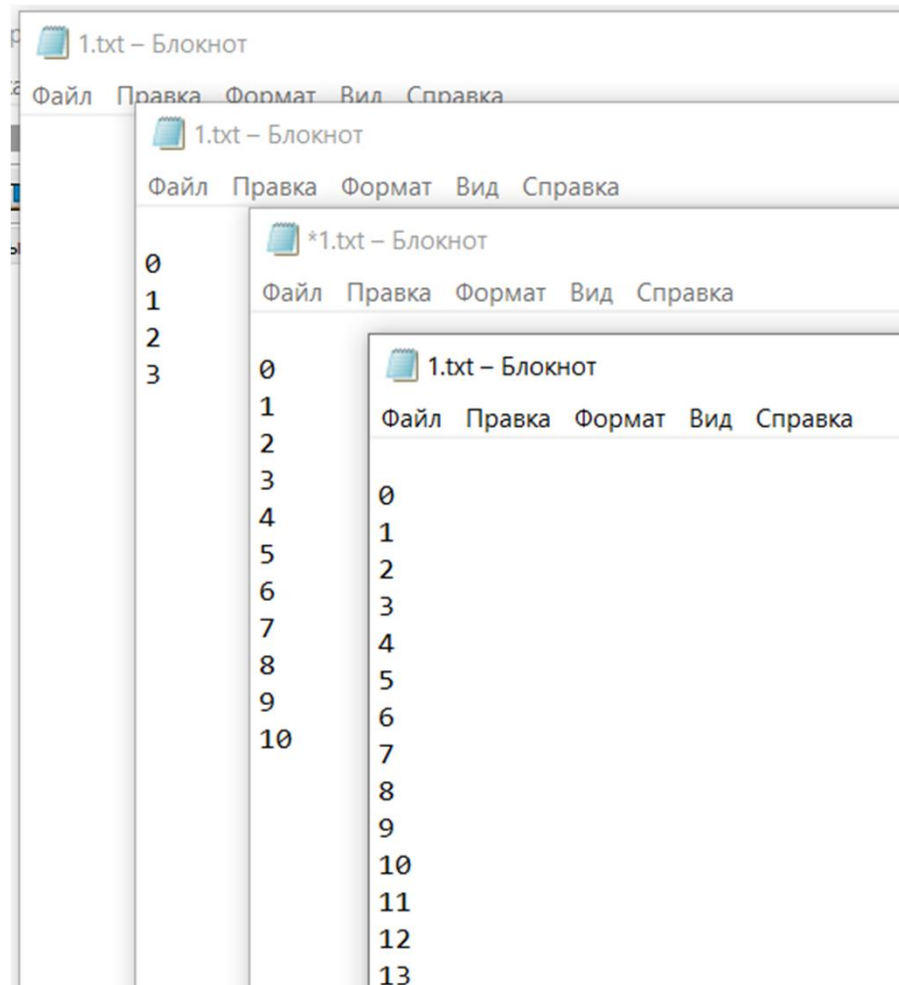
Дозапись потока

В примере отсчитывается время в секундах с момента запуска программы. Отсчитываемое время выводится в файл. Чтобы время выводилось сразу же в момент печати, используется функция **fflush**.

```
#include <stdio.h>    // Для printf, fflush
#include <unistd.h>    // Для sleep
int main () {
    int i=0;           // Счетчик секунд
    FILE *f=fopen("1.txt","r+");
    if (f==NULL) { printf("error"); return 1;
    }
    while (i<150) {
        fprintf (f, "\n%d", i); // Вывод
        fflush (f);             //Сброс буфера
        sleep (1);              //Задержка на 1 секунду
        i++;                    //Увеличение счетчика секунд на 1
    }
    fclose(f);
    return 0;
}
```

Дозапись потока

Благодаря **fflush(f)** изменение содержимого файла происходит в реальном времени.



Файлы текстовые и бинарные

Содержимое файлов представляет собой набор байтов (чисел от 0 до 255). Для интерпретации этих данных как полезной для человека информации нужно знать правила кодирования – декодирования. Для представления текста используемые символы были пронумерованы и сведены в таблицы.

Для записи машино-читаемых данных байты используются напрямую.

Файлы делятся на **текстовые** (для людей) и **бинарные** (для компьютеров).

Текстовые содержат только символы, распознаваемые текстовыми редакторами (в том числе и такие: '**\n**', '**\t**').

Бинарный (двоичный) файл – содержит данные в двоичном представлении, в котором компьютеру удобнее считывать, обрабатывать и сохранять данные (обычно в том виде, как они лежат в памяти компьютера). Скорость обработки данных в бинарном виде гораздо выше!

Бинарные файлы

Для работы с бинарными файлами могут быть использованы различные функции.

Рассмотрим наиболее распространенные: **fread** и **fwrite**. Эти функции позволяют читать и записывать блоки данных любого типа.

```
size_t fread(void * считываемое_данное,  
             size_t размер_элемента,  
             size_t число_элементов,  
             FILE *указатель_на_файл) ;
```

Функция считывает из файла указанное число данных заданного размера. Размер задается в байтах. Функция возвращает число прочитанных элементов. Если число прочитанных элементов не равно заданному, то при чтении возникла ошибка или встретился конец файла.

Одним из самых полезных применений функций **fread()** и **fwrite()** является чтение и запись данных пользовательских типов, особенно структур. Например:

Типизированные (бинарные файлы)

```
#include <stdio.h>
struct some_struct { /* ... */ };
int main () {
//    . . . . .
    FILE * fp;
    char filename[] = "some_file";
    struct some_struct buff[64]; // массив структур из 64-х
    fp =fopen(filename, "wb");
    if( fp == NULL ) {
        printf("Error opening file %s\n",filename);
    } else {
        fwrite(buff, sizeof(struct some_struct), 64, fp);
        fclose(fp);
    }
//    . . . . .
    fp =fopen(filename, "rb");
    if( fp == NULL ) {
        printf("Error opening file %s\n",filename);
    } else {
        fread(buff, sizeof(struct some_struct), 64, fp);
        fclose(fp);
    }
    return 0;
}
```

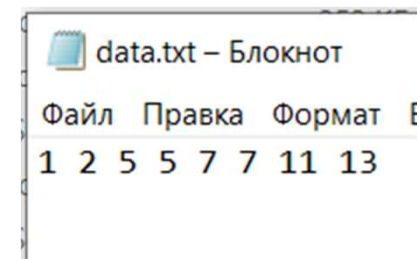
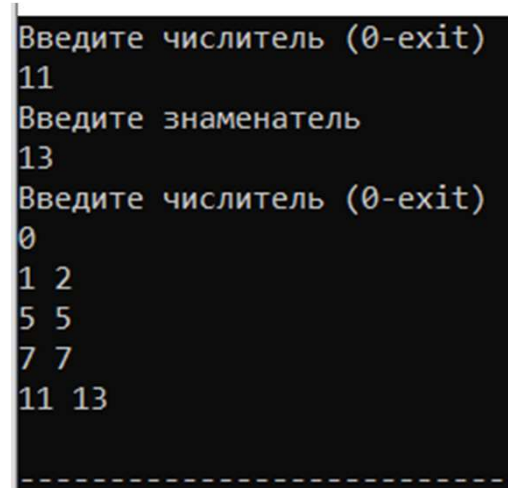
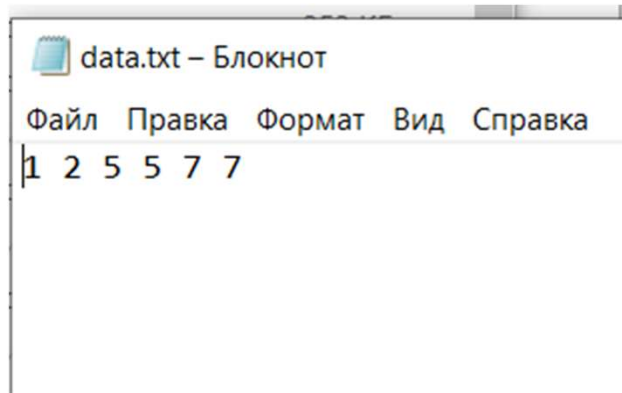
Отличие текстового и типизированного файлов

Пример сохранения дробей в текстовом файле:

```
#include <stdio.h>
struct drob {
    int ch,zn;
};
int main() {
    drob d;
    FILE *f;
    f=fopen("data.txt", "a");
    if (!f) { printf ("error_write"); return 1;}
    while (1) {
        puts("Введите числитель (0-exit)");
        scanf("%d",&d.ch);
        if (d.ch==0) break;
        puts("Введите знаменатель");
        scanf("%i",&d.zn);
        fprintf(f,"%d %d ", d.ch, d.zn);
    }
    fclose(f);
}
```


Отличие текстового и типизированного файлов

```
f=fopen("data.txt","r");
if(!f) {
    printf("error_read");
    return 1;
}
while(1) {
    fscanf(f,"%d %d",&d.ch,&d.zn);
    if (feof(f)) break;
    printf("%d %d \n",d.ch,d.zn);
}
fclose(f);
}
```



Отличие текстового и типизированного файлов

Пример сохранения дробей в типизированном файле:

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

//-----константы и структуры-----
struct drob {
    int ch,zn;
};
const int size_drob=sizeof(drob); //размер структуры

//-----прототипы функций-----
int create_file(FILE *f); //запись в файл
int sort_file(FILE *f); //сортировка файла
int print_file(FILE *f); //вывод файла
```

Отличие текстового и типизированного файлов

```
//-----основная функция-----  
int main() {  
    char c;  
    FILE *f=fopen("data.dat", "r+b"); //Открытие  
    //существующего файла для чтения и записи в конец  
    if (!f) { //Создание нового файла для обновления  
        f=fopen("data.dat", "w+b");  
    }  
    if (!f) {  
        puts("Не могу открыть (создать) файл\n");  
        return 1;}  
}  
while (1)  
    puts("1- Запись в файл");  
    puts("2- Сортировка файла");  
    puts("3- Вывод файла");  
    puts("5- Выход");  
    puts("_____");  
    puts("Введите номер пункта меню\n");  
    c=getch();
```

Отличие текстового и типизированного файлов

```
switch (c) {  
    case '1':  
        create_file(f);  
        break;  
    case '2':  
        sort_file(f);  
        break;  
    case '3':  
        print_file(f);  
        break;  
    case '5':  
        return 0;  
}  
  
}  
  
}
```

Отличие текстового и типизированного файлов

```
//-----запись в файл-----  
  
int create_file(FILE *f) {  
    drob d;  
    fseek(f,0,SEEK_END); //указатель в конец файла  
    while (1) {  
        puts("Введите числитель (0-exit)");  
        scanf("%d",&d.ch);  
        if (d.ch==0)  
            return 1;  
        puts("Введите знаменатель");  
        scanf("%i",&d.zn);  
        fwrite(&d,size_drob,1,f);  
    }  
}
```

Отличие текстового и типизированного файлов

```
//-----вывод файла-----
int print_file(FILE *f) {
    drob d;
    int n;
    rewind(f);          //указатель в начало файла
    puts("_____");
    puts("|   числитель   |   знаменатель   |");
    do {
        n=fread(&d,size_drob,1,f); //чтение структуры
        if (n<1) break;           //если конец файла
        printf("|   %-13d |   %-14d|\n",d.ch, d.zn);
    } while (1);
    puts("_____");
    puts("Нажмите любую клавишу");
    getch();
    return 0;
}
```

Отличие текстового и типизированного файлов

```
int sort_file(FILE *f) {  
    long i,j;  
    drob d1,d2;  
  
    fseek(f,0,SEEK_END);           //указатель в конец  
    long len=ftell(f)/size_drob;   // количество записей  
    rewind(f);                     //указатель в начало  
    //пузырьковая сортировка
```

Отличие текстового и типизированного файлов

```
for(i=len-1; i>=1; i--)
    for (j=0; j<=i-1; j++) {
        fseek(f,j*size_drob,SEEK_SET); //указ. на j-ую запись
        fread(&d1,size_drob,1,f);      //читаем запись j в d1
        fread(&d2,size_drob,1,f);      //читаем след. в d2
        if (d1.ch*d2.zn>d2.ch*d1.zn) {
            fseek(f, (-2)*size_drob,SEEK_CUR);
            //указатель на 2 поз. назад
            //обмен значений
            fwrite(&d2,size_drob,1,f); //сначала записываем d2
            fwrite(&d1,size_drob,1,f); // затем записываем d1
        }
    }
puts("Сортировка успешна завершена");
getch();
return 0;
}
```


Отличие текстового и типизированного файлов

- 1- Запись в файл
- 2- Сортировка файла
- 3- Вывод файла
- 5- Выход

Введите номер пункта меню

числитель	знаменатель
1	3
17	18
1	4
5	7

Нажмите любую клавишу

Нажмите любую клавишу

- 1- Запись в файл
- 2- Сортировка файла
- 3- Вывод файла
- 5- Выход

Введите номер пункта меню

Сортировка успешно завершена

числитель	знаменатель
1	4
1	3
5	7
17	18

Нажмите любую клавишу

