



Файлы

Алтайский государственный университет
Факультет математики и ИТ
Кафедра информатики
Барнаул 2014

Лекция 15

- Текстовые файлы
- Бинарные файлы
- Прочие функции для работы с файлами
- Предопределенные файлы



Несколько заданий для самопроверки

Задание 1

- Что выведет на экран следующая программа?

```
#include <stdio.h>

void main() {
    char x[80] = "А вы знаете, что 2x2=4?";
    *(x+1) = 'ж' - 3;
    *(x+2) = x[7];
    x[3] = 0;
    printf("%s", x);
}
```

Ага

Задание 2

- Перепишите следующий фрагмент программы, используя цикл `do...while` вместо цикла `while`.

```
int Sum = 0;
int Max = 1900;

while (Max < 1950) {
    Sum = Sum + (Max - 1900);
    printf("Sum: %d\n", Sum);
    Max = Max + 5;
}
```

```
int Sum = 0;
int Max = 1900;

do {
    Sum = Sum + (Max - 1900);
    printf("Sum: %d\n", Sum);
    Max = Max + 5;
} while (Max < 1950);
```

Задание 3

- Перепишите следующий фрагмент программы, используя цикл `do...while` вместо цикла `while`.

```
int Sum = 0;
int Max;

printf("Max=");
scanf("%d", &Max);

while ((Max < 1950)) {
    Sum = Sum + (Max - 1900);
    printf("Sum: %d\n", Sum);
    Max = Max + 5;
}
```

```
int Sum = 0;
int Max;

printf("Max=");
scanf("%d", &Max);

if(Max < 1950)
    do {
        Sum = Sum + (Max - 1900);
        printf("Sum: %d\n", Sum);
        Max = Max + 5;
    } while (Max < 1950);
```



Текстовые файлы

- Общие сведения
- Открытие текстовых файлов
- Возможные ошибки
- Чтение/запись в текстовые файлы
- Примеры

Файлы

Файл – именованная область на внешнем носителе

Файл

ы

Текстовые

е

каждый байт (каждые 2 байта)
интерпретируется как код
символа

ASCII (1 байт на символ)

UNICODE (2 байта на символ)

*.txt, *.log,
*.htm, *.html

Двоичные (бинарные)

е)

совокупность байт,
интерпретация которых
может быть разной

*.doc, *.exe,
*.bmp, *.jpg,
*.wav, *.mp3,
*.avi, *.mpg

Этапы работы с файлами

I этап. открыть файл (сделать его активным, приготовить к работе)

для чтения ("r", англ. *read*)

```
f = fopen("qq.dat", "r");
```

для записи ("w", англ. *write*)

```
f = fopen("qq.dat", "w");
```

для добавления ("a", англ. *append*)

```
f = fopen("qq.dat", "a");
```

Переменная –
указатель на файл:

```
FILE *f;
```

II этап: работа с файлом

```
fscanf ( f, "%d", &n ); // ввести значение n
```

```
fprintf ( f, "n=%d", n ); // записать значение n
```

III этап: закрыть (освободить) файл

```
fclose ( f );
```

Работа с файлами

Особенности:

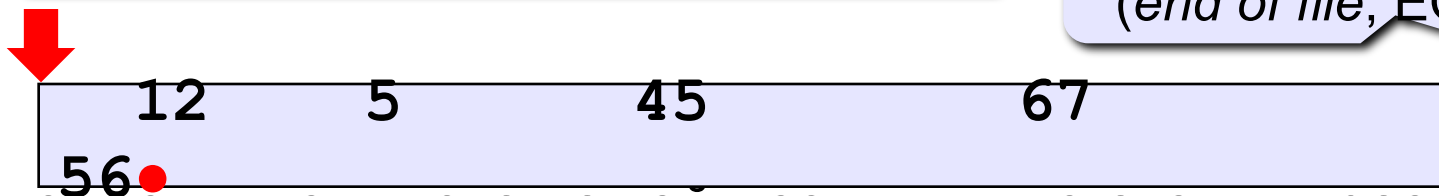
- имя файла упоминается только в команде **fopen**, обращение к файлу идет через указатель **f**;
- файл, который открывается на чтение, должен **существовать**
- если файл, который открывается на запись, существует, старое содержимое **уничтожается**
- данные (*этим способом*) записываются в файл в текстовом виде
- когда программа заканчивает работу, все файлы закрываются автоматически
- после закрытия файла переменную **f** можно использовать еще раз для работы с другим файлом

Последовательный доступ

- при открытии файла курсор устанавливается в начало

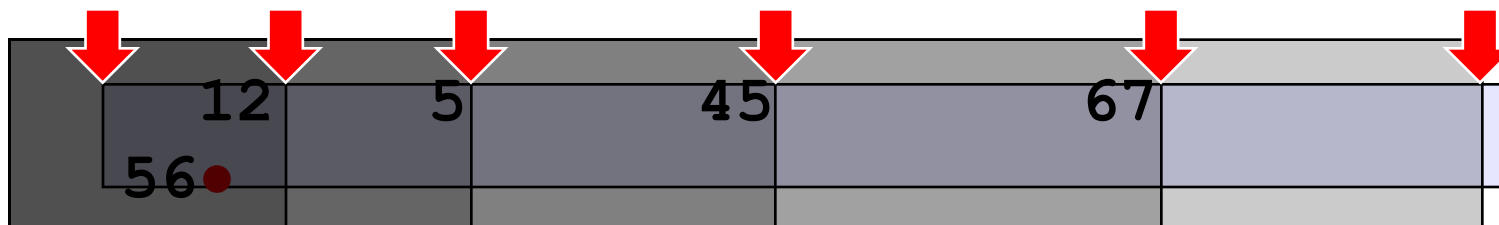
```
f = fopen("qq.dat", "r");
```

конец файла
(*end of file*, EOF)




- чтение выполняется с той позиции, где стоит курсор
- после чтения курсор сдвигается на первый непрочитанный символ

```
fscanf ( f, "%d", &x );
```



Как вернуться назад?

Ошибки при открытии файла

 Если файл открыть не удалось, функция `fopen` возвращает **NULL** (нулевое значение)!

```
FILE *f;  
f = fopen("qq.dat", "r");  
if ( f == NULL ) {  
    puts("Файл не найден.");  
    return;  
}
```

- неверное имя файла
- нет файла
- файл заблокирован другой программой

```
FILE *f;  
f = fopen("qq.dat", "w");  
if ( f == NULL ) {  
    puts("Не удалось открыть файл.");  
    return;  
}
```

- неверное имя файла
- файл «только для чтения»
- файл заблокирован другой программой

Пример

Задача: в файле `input.txt` записаны числа (в столбик), сколько их – неизвестно. Записать в файл `output.txt` их сумму.



Можно ли обойтись без массива?

Алгоритм:

1. Открыть файл `input.txt` для чтения.
2. $S = 0$;
3. Прочитать очередное число в переменную x .
4. Если не удалось, перейти к шагу 7.
5. $S += x$;
6. Перейти к шагу 3.
7. Закрыть файл `input.txt`.
8. Открыть файл `output.txt` для записи.
9. Записать в файл значение S .
10. Закрыть файл `output.txt`.

цикл с условием
«пока есть данные»

Как определить, что числа кончились?



Функция `fscanf` возвращает количество удачно прочитанных чисел;
0, если была ошибка при чтении;
– 1, если достигли конца файла.

```
FILE *f;  
int n, x;  
f = fopen("input.txt", "r")  
...  
n = fscanf ( f, "%d", &x );  
if ( n != 1 )  
    puts ( "Не удалось прочитать число" );
```

- дошли до конца файла
- встретили «не число»

Программа

```
void main()
{
    FILE *f;
    int n, x, S = 0;
    f = fopen ( "input.txt", "r" );
    if ( f == NULL ) {
        printf("Файл не найден.");
        return;
    }
    while ( 1 ) {
        n = fscanf ( f, "%d", &x );
        if ( n != 1 ) break;
        S += x;
    }
    fclose ( f );
    f = fopen ( "output.txt", "w" );
    fprintf ( f, "S = %d", S );
    fclose ( f );
}
```

ошибка при
открытии
файла

ЦИКЛ ЧТЕНИЯ ДАННЫХ:
выход при $n \neq 1$.

запись
результата

Обработка массивов

Задача: в файле `input.txt` записаны числа (в столбик), сколько их – неизвестно, но не более 100. Переставить их в порядке возрастания и записать в файл `output.txt`.



Можно ли обойтись без массива?

Проблемы:

- для сортировки надо удерживать в памяти все числа сразу (массив);
- сколько чисел – неизвестно.

Решение:

- 1) выделяем в памяти массив из 100 элементов;
- 2) записываем прочитанные числа в массив и считаем их в переменной **N**;
- 3) сортируем первые **N** элементов массива;
- 4) записываем их в файл.

Чтение данных в массив

Функция, которая читает массив из файла, возвращает число прочитанных элементов (не более MAX):

```
int ReadArray ( int A[], char fName[], int MAX )
{
    int N=0, k;
    FILE *f;
    f = fopen ( fName, "r" );
    while ( 1 ) {
        k = fscanf ( f, "%d", &A[N] );
        if ( k != 1 ) break;
        N ++;
        if ( N >= MAX ) break;
    }
    fclose(f);
    return N;
}
```

массив

имя файла

предел

заканчиваем цикл
если не удалось
прочитать ...

... или заполнили
весь массив

Программа

```
int ReadArray(int A[], char fName[], int MAX)
{
    ...
}
```

```
void main()
{
    int A[100], N, i;
    FILE *f;
    N=ReadArray ( A, "input.txt", 100 );
    ... // сортировка первых N элементов
    f = fopen("output.txt", "w");
    for ( i = 0; i < N; i ++ )
        fprintf ( f, "%d\n", A[i] );
    fclose ( f );
}
```

Вывод отсортированного массива в файл

Обработка текстовых данных

Задача: в файле `input.txt` записаны строки, в которых есть слово-паразит "**короче**". Очистить текст от мусора и записать в файл `output.txt`.

Файл `input.txt` :

Мама, короче, мыла, короче, раму.

Декан, короче, пропил, короче, бутан.

А роза, короче, упала на лапу, короче, Азора.

Каждый, короче, охотник желает, короче, знать, где ...

Результат – файл `output.txt` :

Мама мыла раму.

Декан пропил бутан.

А роза упала на лапу Азора.

Каждый охотник желает знать, где сидит фазан.

Обработка текстовых данных

Особенность:

надо одновременно держать открытыми два файла (один в режиме чтения, второй – в режиме записи).

Алгоритм:

1. Открыть оба файла.
2. Прочитать строку.
3. Удалить все сочетания "*, короче,*".
4. Записать строку во второй файл.
5. Перейти к шагу 2.
6. Закрыть оба файла.

пока не кончились
данные

Работа с файлами

```
void main()  
{  
    char s[80], *p;  
    int i;  
    FILE *fIn, *fOut;  
    fIn = fopen("input.txt", "r");  
    fOut = fopen("output.txt", "w");  
    ... // обработать файл  
    fclose(fIn);  
    fclose(fOut);  
}
```

указатель
для поиска

файловые
указатели

открыть файл для чтения

открыть файл
для записи

закрывать
файлы

Обработка текстовых данных

Чтение строки s:

```
char s[80], *p;
FILE *fIn;
... // здесь надо открыть файл
    строка   длина   файл
p = fgets ( s, 80, fIn );
if ( p == NULL )
    printf("Файл закончился.");
else printf("Прочитана строка:\n%s", s);
```

Обработка строки s:

```
while ( 1 ) {
    p = strstr ( s, ", короче, " );
    if ( p == NULL ) break;
    strcpy ( p, p + 9 );
}
```

искать ", короче,"

Выйти из цикла,
если не нашли

удалить 9 символов

Полный цикл обработки файла

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

читаем
строку

```
while ( 1 ) {  
    p = fgets ( s, 80, fIn );  
    if ( p == NULL ) break;
```

если нет больше
строк, выйти из
цикла

```
while ( 1 ) {  
    p = strstr ( s, ", короче, " );  
    if ( p == NULL ) break;  
    strcpy ( p, p + 9 );  
}
```

обработка
строки

```
fputs ( s, fOut );
```

```
}
```

запись "очищенной"
строки

Упражнения

В файле `input.txt` записаны строки, сколько их – неизвестно.

1. Заменить во всем тексте «в общем» на «короче» и записать результат в файл `output.txt`.
2. Заменить во всем тексте «короче» на «в общем» и записать результат в файл `output.txt`.



Бинарные файлы

- Общие сведения
- Открытие бинарных файлов
- Поблочные чтение/запись
- Примеры

Бинарные (двоичные) файлы

Особенности:

- данные хранятся во внутреннем **машинном формате** (в текстовом редакторе не прочитать)
- можно читать и записывать любой кусок памяти (просто биты...)
- принцип бутерброда (открыть – работать – закрыть)
- обращение к файлу через указатель

Файловые указатели

```
FILE *fp;
```

Открытие и закрытие двоичных файлов

Открытие файла

```
fp = fopen ( "input.dat" , "rb" ) ;
```

"rb" = *read binary* (чтение)

"wb" = *write binary* (запись)

"ab" = *append binary* (добавление)

Ошибки при открытии

```
if ( fp == NULL ) {  
    printf ("Файл открыть не удалось." ) ;  
}
```

Закрытие файла

```
fclose ( fp ) ;
```

Чтение по блокам

Чтение в начало массива

размер одного
блока

указатель
на файл

```
int A[100];  
n = fread ( A, sizeof(int), 100, fp );
```

прочитано
фактически

адрес области
памяти («куда»):
 $A \leftrightarrow \&A[0]$

размер
переменной
целого типа

количество
блоков

Чтение в середину массива

```
int A[100];  
n = fread ( A+5, sizeof(int), 2, fp );
```

читается 2 целых числа:
 $A[5], A[6]$

Запись по блокам

Запись с начала массива

размер одного
блока

указатель
на файл

```
int A[100];  
n = fwrite( A, sizeof(int), 100, fp );
```

записано
фактически

адрес области
памяти («откуда»):
 $A \Leftrightarrow \&A[0]$

размер
переменной
целого типа

количество
блоков

Запись отдельных элементов массива

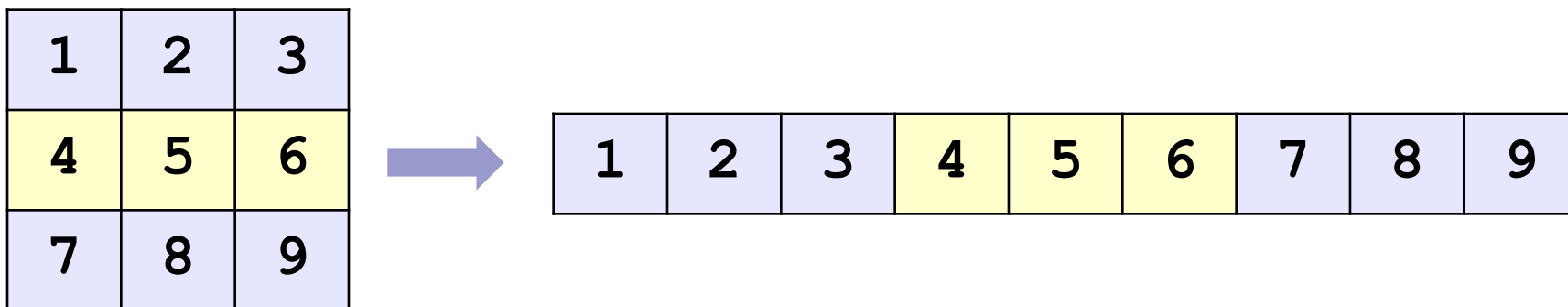
```
int A[100];  
n = fwrite( A+5, sizeof(int), 2, fp );
```

записывается 2 целых числа:

$A[5], A[6]$

Работа с матрицами

Хранение в памяти: построчно



Запись матрицы

```
int A[3][3];  
FILE *fp = fopen("output.dat", "wb");  
... // здесь заполняем матрицу  
n = fwrite(A, sizeof(int), 9, fp);
```

Пример

Задача: прочитать массив из файла `input.dat`, умножить все элементы на 2 и вывести в файл `output.dat`.

Структура программы:

```
#include <stdio.h>
void main()
{
    const int N = 10;
    int i, A[N], n;
    FILE *fp;
    // чтение данных и файла input.dat
    for ( i = 0; i < n; i ++ )
        A[i] = A[i] * 2;
    // запись данных в файл output.dat
}
```

прочитано
фактически

Работа с файлами

Чтение данных:

```
fp = fopen( "input.dat", "rb" );  
if ( fp == NULL ) {  
    printf("Файл открыть не удалось.");  
    return;  
}  
n = fread ( A, sizeof(int), N, fp );  
if ( n < N ) printf("Не хватает данных в файле");  
fclose ( fp );
```

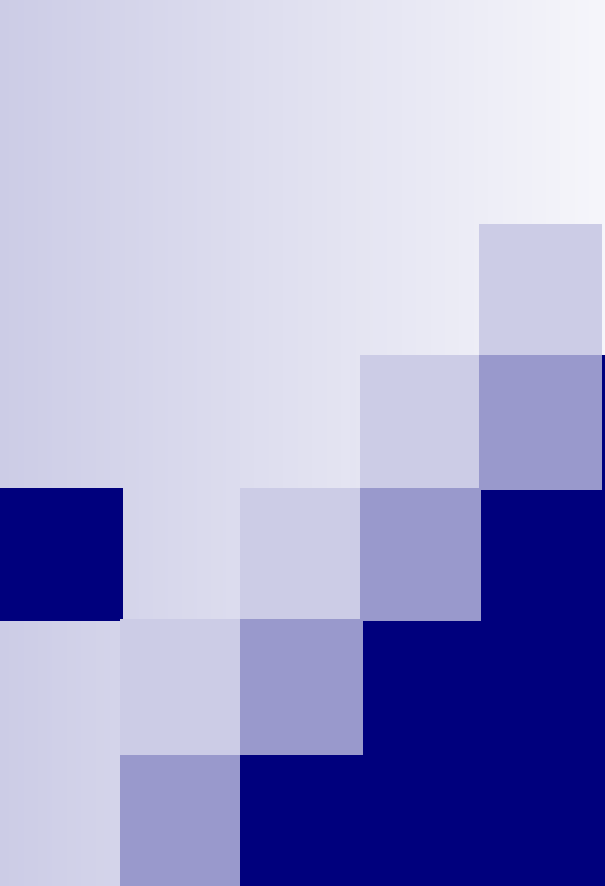
критическая
ошибка

некритическая
ошибка

Запись данных:

```
fp = fopen( "output.dat", "wb" );  
fwrite ( A, sizeof(int), n, fp );  
fclose ( fp );
```

СКОЛЬКО
прочитали



Прочие функции для работы с файлами

- Позиционирование в файле
- Удаление файла
- Переименование файла
- Создание временного файла

Позиционирование в файле

`fseek()` – установка указателя файла
в нужную позицию

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

0 – все ОК,
-1 – ошибка

указатель
на файл

смещение

точка отсчета

Для бинарного файла:

- `offset` – смещение в байтах относительно точки отсчета
- `origin` – `SEEK_SET` (начало), `SEEK_CUR` (текущая позиция), `SEEK_END` (конец файла)

Для текстового файла:

- `offset` должен быть нулем или значением, полученным `ftell()`
- `origin` – всегда `SEEK_SET` (начало)

Позиционирование в файле

`ftell()` – текущая позиция файлового указателя

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

позиция или
-1L – ошибка

указатель
на файл

Возвращает

- смещение в байтах относительно начала файла
- -1L, если произошла ошибка

Пример. Определение размера файла

```
long fsize(char *filename) {
    FILE *fp;
    long size;
    fp = fopen( filename, "rb" );
    if ( fp == NULL ) return -1L;
    fseek(fp,0,SEEK_END);
    size = ftell(fp);
    fclose(fp);
    return size;
}

void main() {
    ...
    printf("%ld\n", fsize("input.txt"));
    ...
}
```

позиционирование в
конец файла

текущая позиция =
размер файла

Позиционирование в файле

`feof()` – достигнут ли конец файла?

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

0 – не конец,
1 – конец

указатель
на файл

Возвращает

- 0, если конец файла не достигнут
- 1, если указатель файла достиг символа «конец файла»

Удаление файла

`remove()` – удалить файл

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

0 – успех,
!=0 – ошибка

имя
файла

Возвращает

- 0, если удаление прошло успешно
- иное, если произошла ошибка

Переименование файла

`rename ()` – переименовать файл

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

0 – успех,
!=0 – ошибка

старое имя
файла

новое имя
файла

Возвращает

- 0, если операция прошла успешно
- иное, если произошла ошибка

Создание временного файла

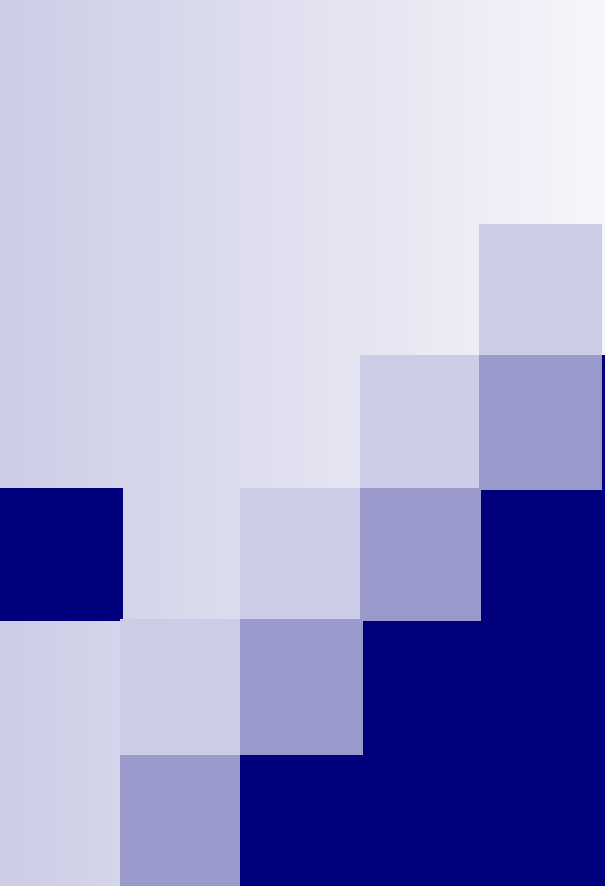
`tmpfile()` – создать временный файл

```
FILE *tmpname ();
```

указатель на файл

Возвращает

- указатель на временный файл, открытый в режиме “wb+”, и автоматически удаляемый при закрытии файла или завершении программы
- NULL, если произошла ошибка



Предопределенные файлы

- Стандартный поток вывода
- Стандартный поток ввода
- Стандартный поток ошибок

Предопределенные файлы

Когда программа начинает работу, открываются три стандартных потока:

- `stdin` – стандартный поток ввода (“rt”)
- `stdout` – стандартный поток вывода (“wt”)
- `stderr` – стандартный поток ошибок (“wt”)

```
printf("Hello!");  
scanf("%d", &x);
```



```
fprintf(stdout, "Hello!");  
fscanf(stdin, "%d", &x);
```

```
c:\>myprog.exe 1> stdout.txt
```

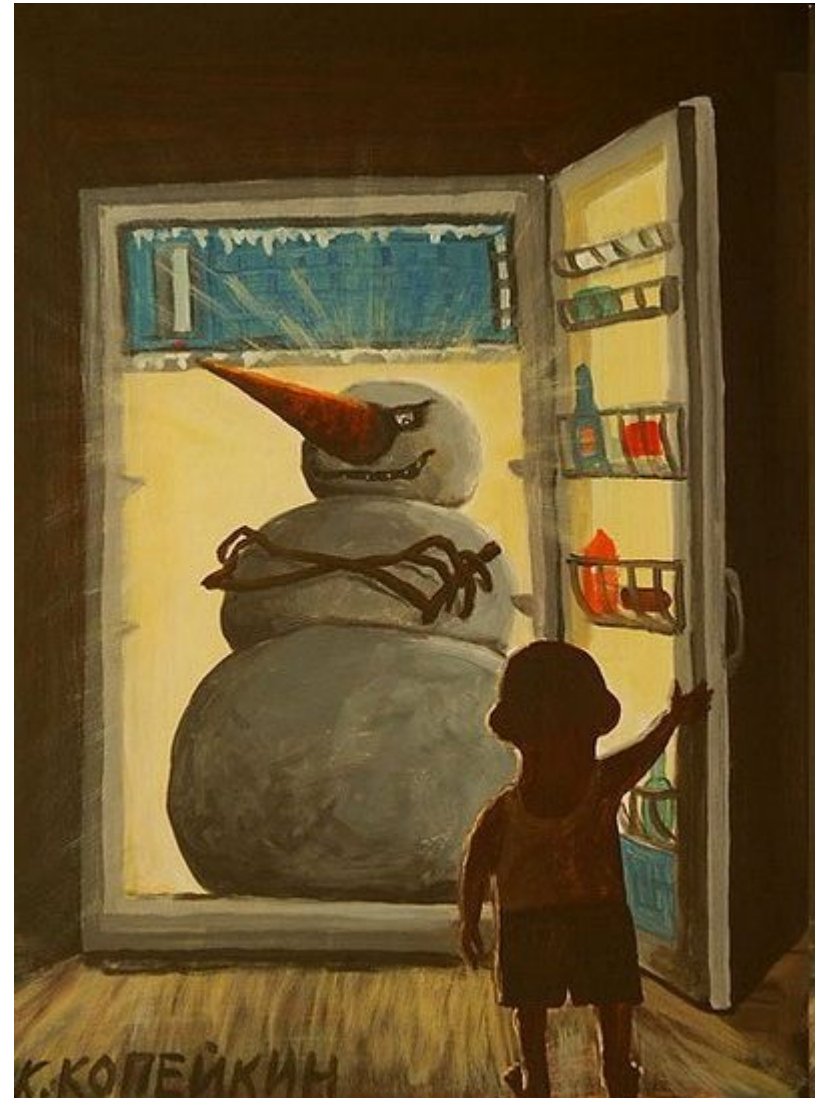
```
c:\>myprog.exe 2> stderr.txt
```

```
c:\>myprog.exe < 18
```

перенаправление потоков
ввода-вывода в ОС

Вопросы?

- **Текстовые файлы**
 - Общие сведения
 - Открытие текстовых файлов
 - Возможные ошибки
 - Чтение/запись в текстовые файлы
 - Примеры
- **Бинарные файлы**
 - Общие сведения
 - Открытие бинарных файлов
 - Поблочные чтение/запись
 - Примеры
- **Прочие функции для работы с файлами**
 - Позиционирование в файле
 - Удаление файла
 - Переименование файла
 - Создание временного файла
- **Предопределенные файлы**
 - Стандартный поток вывода
 - Стандартный поток ввода
 - Стандартный поток ошибок



Н. Копейкин. Начало сказки