Лекция 3.
Бинарные файлы
Файлы записей

Бинарный файл (двоичный файл)
Бинарными называются файлы с данными в двоичном формате, что означает, что каждому значению соответствует определенная последовательность битов. Например, для представления числа 7 в такой системе счисления требуется 4 бита: 0111. Бинарные файлы также могут содержать заголовки и дополнительные метаданные, которые описывают структуру и содержимое файла. Они используются программами для корректной интерпретации и обработки данных, записанных в двоичном коде. Любой файл, хранящийся на жестком диске компьютера, является бинарным для самого компьютера, так как в конечном счете на аппаратном уровне машина может воспринимать только двоичный код. Однако для пользователя файлы бывают либо бинарными, либо текстовыми.

Формат хранения данных. Бинарные файлы содержат информацию в двоичном формате, то есть в виде последовательности нулей и единиц. Текстовые файлы, напротив, хранят данные в виде символов, используя определенную кодировку, такую как ASCII или UTF-8.

Читаемость данных. Одной из важных особенностей бинарных файлов является то, что человек не может их воспринимать без специальных программ или средств для их интерпретации. Текстовые файлы, с другой стороны, содержат информацию в виде понятного нам текста. Ее можно прочитать с помощью любого текстового редактора.

Объем хранимых данных. Бинарные файлы обеспечивают более компактное хранение данных, так как они сохраняют исходный двоичный код без лишней информации. Текстовые файлы, в свою очередь, обычно требуют большего объема памяти. Кроме того, использование двоичных данных обеспечивает их более точное представление, так как текстовые форматы могут потерять точность при конвертации чисел или при хранении особенных символов.

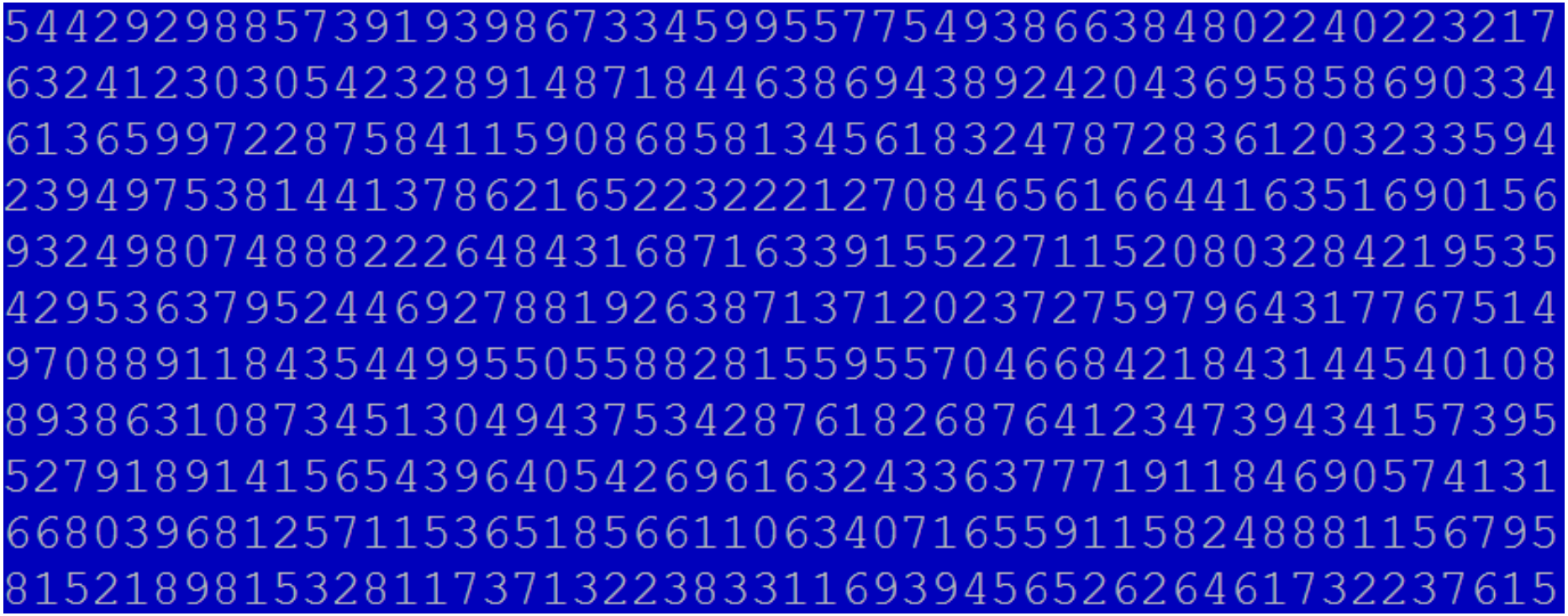
Обработка данных. Бинарные файлы могут быть менее доступными для работы с данными в сравнении с текстовыми файлами. Их содержимое не может быть прочитано или отредактировано обычным текстовым редактором, и для работы требуется специализированное программное обеспечение. Кроме того, разработчики программ должны быть особенно внимательны при работе с бинарными файлами, чтобы избежать проблем совместимости или потери данных. Все это делает их более сложными для работы по сравнению с текстовыми файлами.

Типы данных. Бинарные файлы могут хранить любые данные, включая числа, строки, изображения, звуковые файлы и другие. Текстовые файлы, с другой стороны, предназначены для хранения текстов, состоящих из букв, цифр, общих и специальных символов. Чтобы компьютер смог работать с текстовым файлом (например, программой, написанной на высокоуровневом языке), он сначала переводит его в бинарный вид с помощью компилятора.

Пусть имеется целочисленный заполненный массив из 1000 чисел. Записать числа в файл.

```
1   int i, x[1000];
2   // пусть массив x заполнен
3   FILE *f;
4   f = fopen("numbers.num", "wb");
5   for(i=0; i<1000; i++)
6   {
7       fprintf( f, "%d", x[i]);
8   }
```

В файл записаны числа без разделителя



Числа с разделителем

fpintf(f, “%d ”, x[i]);

fpintf(f, “%5d”, x[i]);

```
53 42 98 66 6 88 66 54 29 43 75 32 4 98 82 4
47 47 1 27 93 34 75 24 71 48 19 21 90 1 1 43
44 99 61 50 39 79 56 20 22 83 4 79 34 86 28.
81 85 81 60 78 67 87 55 90 35 26 11 77 28 12
72 24 63 34 26 2 65 82 74 40 18 78 19 52 64
85 49 32 98 80 99 85 35 89 21 61 52 50 41 64
65 79 9 43 81 74 78 7 14 96 85 85 0 1 36 85
```

Пусть имеется целочисленный файл. Заполнить массив числами из файла

```
1. FILE *f;
2. f = fopen("numbers.num", "rb");
3. int i=0, n[100];
4. while( !feof(f) )
5. {
6.    fscanf(f, "%d", &n[i]);
7.    i++;
8. }
```

Пусть имеется файл, содержащий целые числа. Подсчитать количество чисел в файле

```
1. int count=0, x;
2. FILE *f;
3. f = fopen("numbers.num", "rb");
4. while(!feof(f))
5. {
6.     fscanf(f, "%d", &x);
7.     count++;
8. }
```

Генератор случайных чисел

- **#include<stdlib.h>**
- Функция **srand()** в качестве аргумента просит первоначальное случайное число.
- Функция **rand()** генерирует целое положительное число
- **int chislo = rand();**

void srand(unsigned int seed);

Функция **srand** выполняет инициализацию генератора случайных чисел **rand**. Генератор псевдо-случайных чисел инициализируется с помощью аргумента **seed**. Для любого другого значения, передаваемого через параметр **seed** и используемого при вызове функции **srand**, алгоритм генерации псевдо-случайных чисел может генерировать различные числа с каждым последующим вызовом **rand**. Если использовать одно и то же значение **seed**, с каждым вызовом функции rand алгоритм генерировать ту же самую последовательность чисел. Удобно использовать функцию времени **time**. Значение, возвращенное функцией **time** ,отличается каждую секунду, позволяет получать различные случайные последовательности чисел при каждом новом вызове функции **rand**.

Параметры: seed - целое значение, используемое для рандомизации псевдо-случайных чисел. Функция не возвращает никакого значения.

```
int seed = time(NULL);   // time.h
srand(seed);
cout << "1. " << rand() << endl;
cout << "2. " << rand() << endl;
cout << "3. " << rand() << endl;
```

Генератор случайных чисел

RAND_MAX это специальная константа языка Си, в которой содержится максимальное целое число, которое может быть возвращено функцией rand()

```
int chislo;
srand(time(NULL));
chislo = rand();
 //от 0 до числа RAND_MAX
 printf("RAND_MAX=%d\n",RAND_MAX);
```

RAND_MAX =32767

Генерация случайных чисел

```
1. int j, i;
2. srand(time(NULL));
3. for(j=0; j<10; j++)
4.  {
5.     i = rand() % 100; // остаток от деления на 100
6.      printf("%d\n", i);
7.    }
```

Генерация случайных чисел

Регулирование диапазона чисел:

начальное значение + rand()% конечное значение

1. int chislo = 3 + rand() % 7;
2. int chislo = -3 + rand() % 7;
3. float chislo = (float)(rand() % 100)/10;
4. float chislo = (float)(rand())/ RAND_MAX;

Ввод/вывод записей

Функция **fread( )** - для чтения блоков данных из потока.

size_t fread(void *ptr, unsigned size, unsigned n, FILE *fp);

Функция **fread** считывает массив размером **n** элементов, каждый из которых имеет размер **size** байт, из потока, и сохраняет его в блоке памяти, на который указывает **ptr**. Индикатор положения файла увеличивается на число записанных байтов.

Параметры: ptr - указатель на блок памяти, размер которого должен быть минимум (**size*n**) байт, **size** - размер в байтах каждого считываемого элемента, **n** - количество элементов, каждый из которых имеет размер **size** байт, **fp** - указатель на файл, который связан с потоком ввода.

Возвращаемое значение: возвращается объект, который содержит общее количество успешно считанных элементов. Если возвращаемое значение отличается от количества элементов, значит произошла ошибка или был достигнут конец файла. **fread** не определяет, действительно ли произошла ошибка или был достигнут конец файла. Для определения необходимо использовать функции **feof** и **ferror**.

Функция **fwrite( )** – для записи блоков данных в поток

size_t fwrite(void *ptr, unsigned size, unsigned n, FILE *fp);

Функция **fwrite** записывает массив размером **n** элементов, каждый из которых имеет размер **size** байт, в блок памяти, на который указывает **ptr**. Индикатор положения потока увеличивается на общее число записанных байтов.

Параметры: ptr - указатель на массив элементов, которые необходимо записать в файл, **size** - размер в байтах каждого элемента массива, *n* - количество элементов, каждый из которых занимает **size** байт, **fp** - указатель на файл, который связан с потоком вывода.

Возвращаемое значение: общее число элементов, которые были записаны. Возвращаемое значение, в этом случае имеет тип данных size_t. Если возвращаемое значение отличается от количества элементов, значит произошла ошибка.

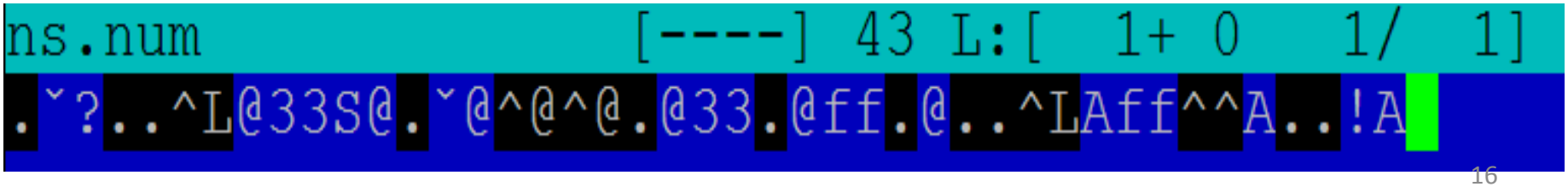
Функция sizeof()

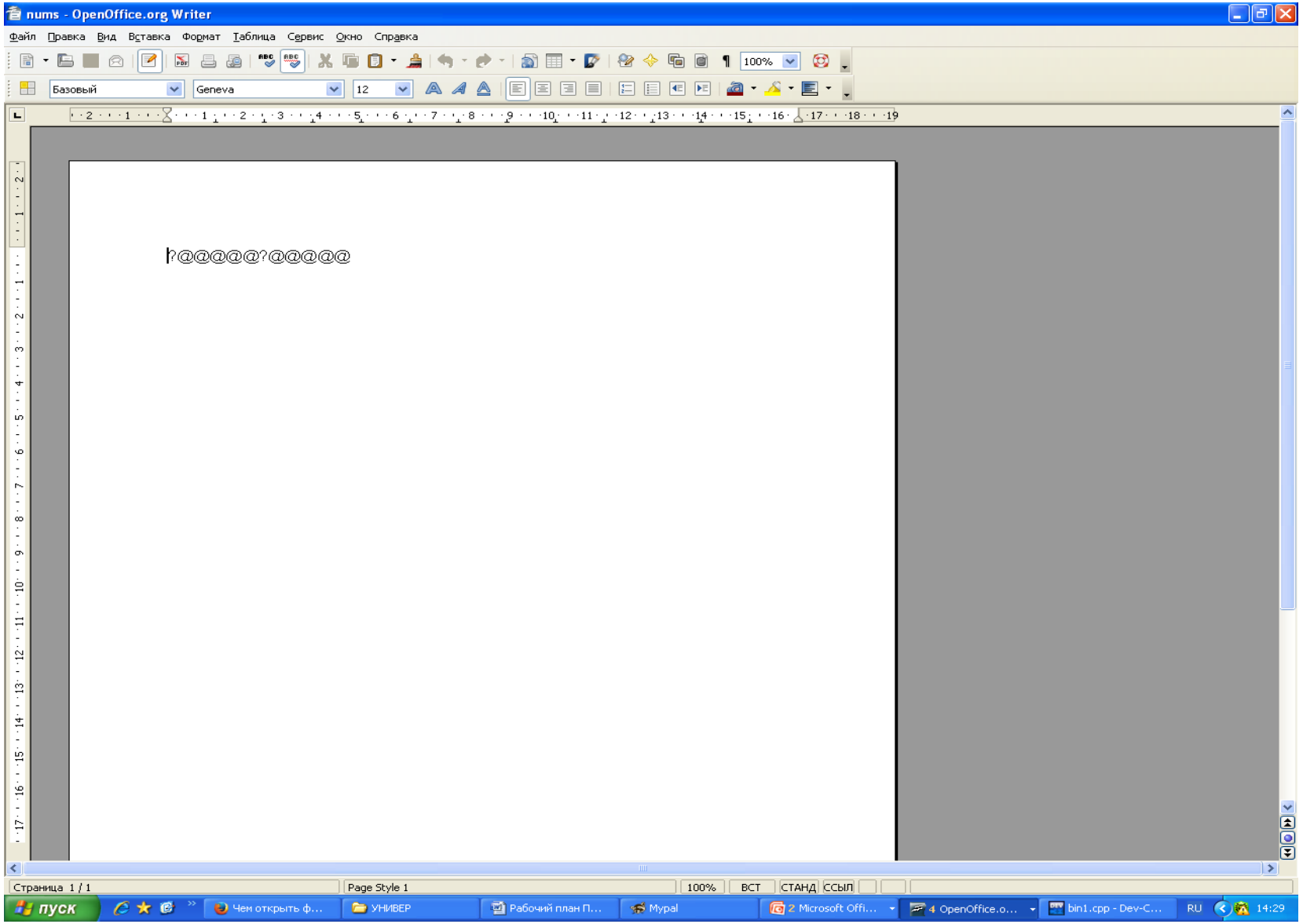
С помощью операции **sizeof** можно определить размер памяти которая соответствует идентификатору или типу. Если в качестве выражения указанно имя массива, то результатом является размер всего массива (т.е. произведение числа элементов на длину типа)

1. char buf[120];
2. fgets(buf, sizeof(buf)-1, fp);
3. int x[100];
4. int n=sizeof(x);
5. int m=sizeof(int*100);
6. int k=sizeof(400);

Запись массива в файл

```
1. float A[10];
2. int i;
3. FILE *f;
4. f=fopen("nums.num", "wb");
5. for(i=0; i<10; i++)
6.     scanf("%f", &A[i]);
7. fwrite(A, sizeof(A), 1, f);
8. fclose(f);
```




nums - OpenOffice.org Writer
Файл Правка Вид Вставка Формат Таблица Сервис Окно Справка
Базовый
Geneva
12
100%
?@@@@@?@@@@@
Страница 1 / 1
Page Style 1
100%
ВСТ
СТАНД
ССЫЛ
пуск
Чем открыть ф...
УНИВЕР
Рабочий план П...
Mypal
2 Microsoft Offi...
4 OpenOffice.o...
bin1.cpp - Dev-C...
RU
14:29

Чтение массива из файла

```
1. float A[10];
2. int i;
3. FILE *f;
4. f=fopen("nums.num", "rb");
5. fread(A, sizeof(A), 1, f);
6. for(i=0; i<10; i++)
7.         printf("%f ", A[i]);
8. fclose(f);
```

Ввод/вывод структур

```
struct data {
        int day;
        char month[10];
        int year;
                };

struct  data   mydata, *dat;
dat = &mydata;
```

Запись структуры в файл

1. fwrite(&mydata, sizeof(**struct data**), 1, fp);
2. //либо
3. fwrite(&mydata, sizeof(**mydata**), 1, fp);
4. // либо
5. fwrite(**dat**, sizeof(mydata), 1, fp);

Пример

1. FILE *f;
2. f=fopen("my_structs", "wb");
3. scanf("%d", &mydata.day);
4. scanf("%s", mydata.month);
5. scanf("%d", &mydata.year);
6. fwrite(&mydata, sizeof(struct data), 1, f);

Чтение структуры

1. fread(&mydata, sizeof(**struct data**), 1, fp);
2. // либо
3. fread(&mydata, sizeof(**mydata**), 1, fp);
4. // либо
5. fread(**dat**, sizeof(mydata), 1, fp);

```
while(fread(&mydata[length], sizeof(mydata),1, fp) == 1)
        length++;
```

Цикл для чтения из файла

Чтение записи с проверкой обнаружения конца файла в качестве условия окончания цикла

```
while( fread(dat, sizeof(mydata), 1, fp) != 0)
  {
     printf(“%d”, mydata.day);
     printf(“%s”, mydata.month);
     printf(“%d”, mydata.year);
  }
```

Передача параметров функции main()

int main ( )

int main (int argc, char *argv[])

./my_proga text1 text2

argc ➔ 3

argv[0] ➔ my_proga

argv[1] ➔ text1

argv[2] ➔ text2

```
1.  #include<stdio.h>
2.  int main ( int argc, char *argv[])
3.  {
4.      int i=0;
5.      printf ("Число параметров равно%d\n", argc);
6.      printf ("Имя программы %s\n", argv[0]);
7.      for (i=1; i < argc; i++)
8.          printf ("аргумент %d равен %s\n", i, argv[i]);
9.      return 0;
10. }
```

./my_proga myfile1.txt myfile2.txt myfile3.txt

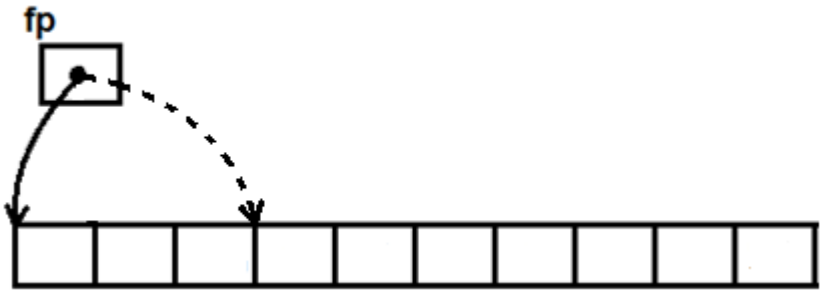
Параметры функции main() в качестве имен файлов

```
f1=fopen(argv[1], "rb");
f2=fopen(argv[2], "wb");
```

Способ задания начальной позиции

fseek(fp, sizeof(mydata)*3, **0**);

fwrite(&MyData,sizeof(MyData), 1, fp);



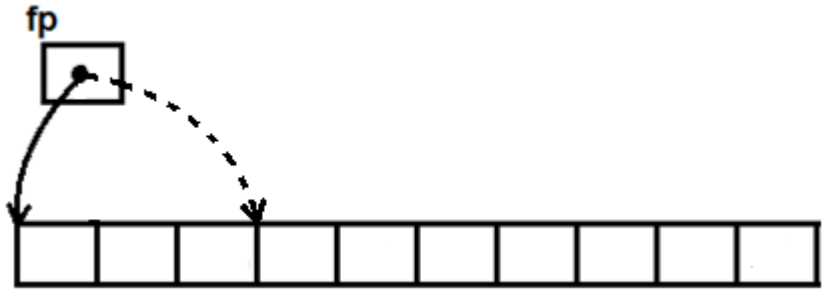
Способ задания начальной позиции

```
fread(&MyData, sizeof(mydata), 1, fp);
fseek(fp, sizeof(mydata)*2, 1);
fwrite(&mydata,sizeof(mydata), 1, fp);
```

0 - начальная позиция задана в начале файла;

1 - начальная позиция считается текущей;

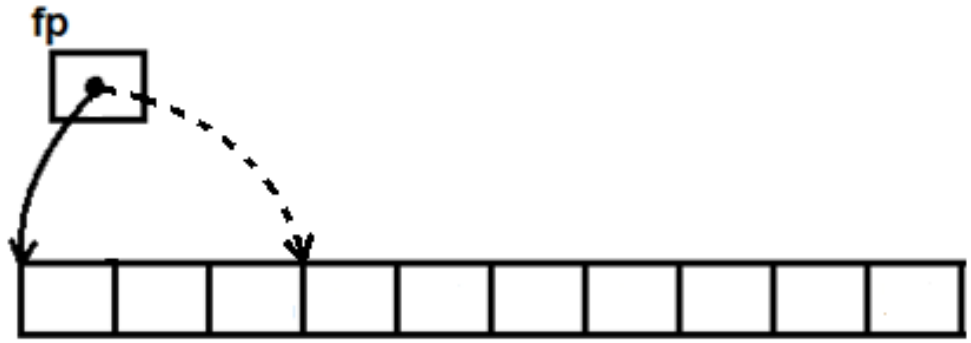
2 - начальная позиция задана в конце файла.



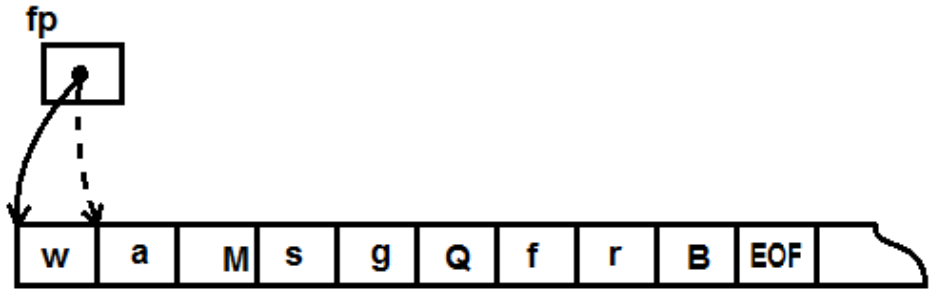
Способ задания начальной позиции

fseek(fp, sizeof(mydata)*(-7), **2**);

fwrite(&mydata,sizeof(mydata), 1, fp);



Последовательный и прямой доступ к записям файла




```
float a;
FILE *f;
f=fopen("numb","rb");
fscanf(f, "%f", &a);
fscanf(f, "%f", &a);
fscanf(f, "%f", &a);
printf("%f", a);
```

1.1 2.2 3.3 4.4 5.5 6.6

Прямой доступ. Обновление четвертой записи

1. FILE *fp;
2. fp=fopen("mystructs", "rb+");
3. fseek(fp, sizeof(mydata)*3, 0);
4. fread(&dat, sizeof(mydata), 1, fp);
5. scanf("%s", &mydata.month);
6. fseek(fp, sizeof(mydata)*3, 0);
7. fwrite(&mydata,sizeof(mydata), 1, fp);

Указатель текущей позиции в начало файла

Функция **rewind()** устанавливает внутренний указатель положения файла в начальное положение (начало файла).

Аналог - функция **fseek** для возвращения указателя на начало:

fseek(stream , 0 , SEEK_SET);

В отличие от функции **fseek**, **rewind** обнуляет индикатор ошибок. Для потоков, открытых в режиме обновления (чтения + записи) вызов функции **rewind** позволяет переключаться между режимами чтения и записи.

void rewind (FILE *fp);

Следует использовать, если чтение файла должно быть выполнено несколько раз.

Передача файлового указателя функции

```
1. void prosmotr(FILE *fp)
2. {    rewind(fp);
        // использование fread(…)    }

3. int main() {
4.      FILE *fp;
5.      fp=fopen("mystructs","rb+");
6.      prosmotr(fp);
7.      dobavlenie(fp);
8.      …  }
```