**Лабораторная работа №17**

**«Бинарные файлы»**

**Теория**

Стандартная библиотека <stdio.h> содержит также:

size\_t fread(void \*buf, size\_t kol, size\_t n, FILE \*f) - считывает данные из файла f, buf – адрес области памяти, куда происходит считывание файла, kol - количество считываемых элементов данных, n - длина каждого элемента данных (в байтах);

size\_t fwrite(const void \*buf, size\_t kol, size\_t n, FILE \*f) - записывает данные в файл f, buf – адрес области памяти, откуда происходит запись файла, kol - количество считываемых элементов данных, n - длина каждого элемента данных (в байтах);

int fseek(FILE \*f, longint kb,int ot) - указатель текущей позиции файла f смещается на kb байт относительно точки отсчета ot.

***ot*** может принимать следующие значения:

SEEK\_SET – отсчет от начала файла

SEEK\_END – отсчет от конца файла

SEEK\_CUR – отсчет от текущей позиции указателя файла

int fgetpos(FILE \*f, fpos\_t \*pos) – возвращает текущую позицию в файле. Тип fpos\_t описывает указатель на позицию в файле.

int fsetpos (FILE \* f, const fpos\_t \*pos) – устанавливает файл в указанную позицию.

int ftell(FILE \*f) – функция возвращает текущее значение указателя текущей позиции в файле, связанном с указателем файла f.

**Практика**

1. Пример. Мы берём входной текстовый массив, читаем его словом за слово и компактно пишем в файл **(lab17\_1.cpp).**
2. В отличие от примера 1, в примере **2** мы записываем весь массив целиком **(lab17\_2.cpp).**
3. Пример демонстрирует чтение массива, полученного в предыдущем примере. Мы лишь должны позаботиться, чтобы принимающее поле было достаточно для того, чтобы туда поместилось всё содержимое файла **(lab17\_3.cpp).**
4. Пример **cut\_and\_glue** (разрезание и склеивание), производит следующие действия. Мы вводим имя файла, затем тип операции «c» (cut). После этого программа запрашивает размер блоков, на которые мы хотим поделить файл. После того, как мы укажем этот размер, будут созданы несколько файлов, которые содержат куски из исходного файла. Потом мы можем произвести обратный процесс (склеивание). Для этого на запрос программы мы должны ввести исходное имя файла и указать операцию «g» **(lab17\_4\_cut\_glue.cpp).**
5. Пример. **(lab17\_5.cpp).** Отсортировать бинарный файл, содержащий структуры с информацией о студентах, в алфавитном порядке фамилий. Структура содержит следующие поля: фамилия студента, количество оценок (не более пяти), массив оценок.

Сортировку файла выполним с использованием алгоритма модифицированного пузырька. Для этого в цикле каждый раз считываются из файла два соседних элемента. Если фамилия первого студента больше фамилии второго студента, то считанные из файла структуры надо записать назад в файл в обратном порядке. Для этого курсор файла смещается от текущей позиции в сторону начала файла на количество байт, равное объему двух считанных структур. После этого структуры записываются в обратном порядке. Переменная flag, фиксирующая факт обмена, получает при этом значение истина, что означает необходимость дальнейшего выполнения внешнего цикла (файл еще не отсортирован).

При новом выполнении цикла будут прочитаны из файла две очередные структуры, это обеспечивается путем перемещения файлового указателя на величину i\*n байт от начала файла. Здесь i – номер первой структуры, считываемой на очередном шаге цикла (совпадает со значением параметра цикла), n – объем одной структуры в байтах.

Для определения объема файла и изменения его размеров (отсечения части файла) следует использовать функции из библиотеки **<io.h>**.

Получить размер файла в байтах можно с помощью функции

**long int filelength( int fd)**

Функция возвращает объем файла, выраженный в байтах. Параметр fd – дескриптор файла.

Для получения дескриптора файла следует обратиться к функции

**int \_fileno(FILE \*f), где f - указатель на файл.**

1. Имеется некоторый файл. Создать новый файл с именем исходного файла, присоединив к имени «1». После этого переписать исходный файл в созданный, используя функции fread и fwrite. Имя файла можно указать внутри программы. Исходный файл должен находиться в той же папке, что и исходный текст программы.
2. Подсчитывает частоту вхождения того или иного символа в указанном файле. Формируем массив для подсчета размером 256. Читаем файл посимвольно. По коду символа выбираем элемент массива и увеличиваем его значение. После того, как файл полностью прочитан, публикуем результат.
3. Создать бинарный файл, записав в него произвольные действительные числа. Затем на место максимального элемента записать сумму положительных элементов, а на место минимального элемента – сумму отрицательных элементов. (При поиске минимального и максимального элементов запоминается адрес этих элементов, т.е. смещение относительно начала файла. При записи на их места найденных сумм предварительно файловый указатель смещается на величину найденного смещения).
4. Кодирование файла. Простую перекодировку указанного файла. Запросить имя файла. Если файл удалось открыть, запросить пароль. После того, как пароль введён, читать файл и побайтно произвести операцию «исключающее или» с очередным байтом пароля. После того, как некоторый блок исходного файла обработан, нужно отмотать файл на длину прочитанного блока и записать новое содержимое блока на прежнее место. Для этого можно использовать функции:

**fpos\_t pos;**

**fgetpos(f,&pos**); // Запоминаем текущую позицию в файле

**fsetpos(f,&pos);** // Перемотка файла на сохранённую позицию

После этого переходить к обработке следующего блока.

Для того чтобы восстановить файл в исходном виде, нужно вызвать программу повторно и повторить перезапись с тем же паролем.