1. **Файлы**

Файл (англ. file) — это блок информации на запоминающем устройстве компьютера, имеющий определённое логическое представление (начиная от простой последовательности битов или байтов и заканчивая объектом сложной СУБД), соответствующие ему операции чтения-записи и, как правило, фиксированное имя, позволяющее получить доступ к этому файлу и отличить его от других файлов.

Условно можно выделить два типа операций с файлом — связанные с его открытием, и выполняющиеся без его открытия. Операции первого типа обычно служат для чтения/записи информации или подготовки к записи/чтению. Операции второго типа выполняются с файлом как с «объектом» файловой системы, в котором файл является мельчайшей единицей структурирования.

В практической части работы используются операции чтения списка из файла и записи списка в файл – поэтому подробного рассмотрения, в контексте данной работы, заслуживает только первый тип операций.

Язык программирования Си поддерживает множество функций стандартных библиотек для файлового ввода и вывода. Эти функции составляют основу заголовочного файла стандартной библиотеки языка Си <stdio.h>.

Функциональность ввода-вывода языка Си по текущим стандартам реализуется на низком уровне. Язык Си абстрагирует все файловые операции, превращая их в операции с потоками байтов, которые могут быть как «потоками ввода», так и «потоками вывода». В отличие от некоторых ранних языков программирования, язык Си не имеет прямой поддержки произвольного доступа к файлам данных; чтобы считать записанную информацию в середине файла, программисту приходится создавать поток, ищущий в середине файла, а затем последовательно считывать байты из потока.

Потоковая модель файлового ввода-вывода была популяризирована во многом благодаря операционной системе Unix, написанной на языке Си. Большая функциональность современных операционных систем унаследовала потоки от Unix, а многие языки семейства языков программирования Си унаследовали интерфейс файлового ввода-вывода языка Си с небольшими отличиями (например, PHP). Стандартная библиотека C++ отражает потоковую концепцию в своем синтаксисе.

Стандарт языка Си описывает два вида файлов — текстовые и двоичные — хотя операционная система не требует их различать:

- *текстовый файл* — файл, содержащий текст, разбитый на строки при помощи некоторого разделяющего символа окончания строки или последовательности. При считывании байтов из текстового файла, символы конца строки обычно связываются (заменяются) с переводом строки для упрощения обработки. При записи текстового файла одиночный символ перевода строки перед записью связывается со специфичной для ОС последовательностью символов конца строки.

- *двоичный файл* — файл, из которого байты считываются и выводятся в «сыром» виде без какого-либо связывания (подстановки). Так же существует и устоявшийся синоним двоичных файлов – бинарный. Сам термин появился из названия двоичной системы счисления - binary. По причине бессмысленности записи структурного типа данных в текстовый файл, в практической части данной работы, запись данных осуществляется именно в двоичные файлы.

*Работа с файлами* в языке Си проходит в несколько этапов:

**a)** работа начинается с объявления указателя на поток. Этот указатель структурного типа, в котором содержится информация, с помощью которой ведется работа с потоком, в частности: указатель на буфер, указатель (индикатор) текущей позиции в потоке и т.д.

**б)** следующий шаг — открытие потока, которое производится с помощью функции fopen(). Эта функция возвращает конкретное значение для указателя на поток, и поэтому ее значение присваивается объявленному ранее указателю. В параметрах данной функции указывается имя/путь к файлу, режим открытия файла (для чтения, записи, добавления или комбинации этих действий), текстовый файл или бинарный. Так же следует добавить, что открытие уже существующего файла для записи ведет к потере прежней информации в нем. Если такой файл еще не существовал, то он создастся. Открывать для чтения можно только существующий файл. Если при открытии потока по какой-либо причине возникла ошибка, то функция fopen() возвращает значение константы NULL.

**в)** после успешного открытия файла, выполняются действия по записи данных в файл, либо чтение информации из него. В библиотеке stdio.h, для этого, определены несколько функций, используемых для различных типов данных. Например:

- fgetc() / fputc() – для чтения/записи одиночного символа в поток;

- fgets() / fputs() – для чтения/записи строк (массивов символов) в поток;

- fread() / fwrite() – для чтения/записи блока данных;

Так как в практической части работы в файл пишутся элементы односвязного списка - структуры, используется последний способ, который и стоит рассмотреть подробнее.

Функция **fwrite:** *int fwrite(buffer, size, count, stream);*

*buffer* - указатель на записываемые данные;

*int size* - размер элемента в байтах;

*int count* - максимальное число записываемых элементов;

*stream* - указатель на структуру типа FILE.

*Описание функции*: Элементы по адресу *buffer* записываются в текущую позицию в потоке. Для каждого элемента запишется *size* байт. Индикатор позиции в потоке изменится на число байт, записанных успешно. Возвращаемое значение будет равно *count* в случае успешного завершения записи. В случае ошибки, возвращаемое значение будет меньше *count*.

Функция **fread**: *int fread (buffer, size, count, stream);*

*buffer* - указатель на память для размещения прочитанных данных;

*int size* - размер данных в байтах;

*int count* - максимальное число объектов для чтения;

*stream* - указатель на структуру типа FILE;

*Описание функции*: Функция читает объекты размера size из входного потока stream и располагает по адресу buffer. Индикатор позиции в потоке изменится на число успешно прочитанных байт. Возвращаемое значение будет равно count в случае успешного завершения чтения. В случае ошибки, возвращаемое значение будет меньше count.

Рассмотренные выше функции позволяют записывать/считывать обычные структуры целиком, но из-за нескольких особенностей, в практической части работы, пришлось использовать чтение/запись каждого из членов структуры отдельно. Вот некоторые из упомянутых проблем:

- данные функции некорректно работают с типом Ansistring, используемым для хранения строковой информации в практическом приложении. Для решения данной проблемы, используется функция преобразования AnsiString в массив символов при записи в файл; и обратное преобразование при чтении из файла.

- при записи указателя на структуру, и, последующем чтении, адреса структур не совпадают - что приводит к аварийному завершению программы, при попытке записи или чтения из “старых” адресов структур.

**г)** Закрытие файлового потока, при помощи функции fclose(). Она принимает один аргумент – указатель на файловый поток, который требуется закрыть (структуру типа FILE). Возвращает функция 0 – если поток успешно закрыт, или -1 – в противном случае.