**Лабораторная работа №3**

Двоичные и текстовые файлы.

**Цель:** реализовать схему для работы с данными. Создать меню программы и функции для его реализации.

## Файлы с данными

Для удобства обращения информация в запоминающих устройствах хранится в виде файлов. ***Файлом*** называют способ хранения информации на физическом устройстве. Работа с файлом похожа работу с консолью: с помощью функций форматированного ввода сохраняются данные в файл, с помощью функций форматированного вывода считываются данные из файла.

***Файл*** – *именованный* набор байтов, который может быть сохранен на некотором накопителе и рассматриваемый в процессе обработки как единое целое. Под файлом понимается некоторая последовательность байтов, которая имеет своё, *уникальное* имя, например, ***файл.txt***. В одной директории не могут находиться файлы с одинаковыми именами. Под именем файла понимается не только его название, но и расширение, например: file.**txt** и file.**dat –** разные файлы, хоть и имеют одинаковые названия. Существует такое понятие, как ***полное имя файлов*** – это полный адрес к директории файла с указанием имени файла, например: D:\docs\file.txt.

В *Cи* отсутствуют операторы для работы с файлами. Все необходимые действия выполняются с помощью функций, включенных в стандартную библиотеку. Прототипы функций для работы с файлами содержатся в заголовочном файле ***<stdio.h>***. Они позволяют работать с различными устройствами, такими, как диски, принтер, коммуникационные каналы и т.д. Эти устройства отличаются друг от друга. Однако файловая система преобразует их в *единое абстрактное логическое устройство*, называемое **потоком**. В ОС существует три стандартных потока: стандартный поток вывода ***stdout***, стандартный поток ввода ***stdin*** и стандартный поток вывода ошибок ***stderr***.

Различают два вида файлов: текстовые и двоичные.

**Текстовый поток** – это последовательность символов. Они организуются по строкам, каждая из которых заканчивается символом «конца строки». Конец самого файла обозначается символом «конца файла». При записи информации в текстовый файл, просмотреть который можно с помощью любого текстового редактора, все данные преобразуются к символьному типу и хранятся в символьном виде. При передаче символов из потока на экран, часть из них не выводится (например, символ возврата каретки, перевода строки). В текстовом режиме каждый разделительный символ строки автоматически преобразуется в пару (возврат каретки – переход на новую строку).

**Двоичный поток** – это последовательность байтов, которые однозначно соответствуют тому, что находится на внешнем устройстве. В двоичных файлах информация считывается и записывается в виде блоков определенного размера, в которых могут храниться данные любого вида и структуры.

Открытый файл представляется как последовательность считываемых или записываемых данных. При открытии файла с ним связывается поток ввода-вывода. Выводимая информация записывается в поток, вводимая информация считывается из потока.

Основные операции, которые выполняются с файлами:

* 1. *Определение* переменной типа FILE*.*
  2. *Открытие файл*, для того, чтобы к нему можно было обращаться. Соответственно, открывать файл можно для чтения, записи, чтения и записи, переписывания или записи в конец файла и т.п. При открытии файла могут возникнуть ошибки – файла может не существовать, это может быть файл не того типа, отсутствие прав на работу с файлом и т.д. Всё это необходимо учитывать.
  3. *Работа с файлом* – запись и чтение. Здесь также нужно помнить, что операции выполняются не с памятью с произвольным доступом, а с буферизированным потоком (абстрактным логическим устройством), что добавляет свою специфику.

1. *Закрытие файл*. Так как файл является внешним по отношению к программе ресурсом, то если его не закрыть, то он продолжит находиться в памяти, возможно, даже после закрытия программы (например, нельзя будет удалить открытый файл или внести изменения и т.п.). Кроме того, иногда необходимо не закрывать, а «переоткрыть» файл для того, чтобы, например, изменить режим доступа.

## Организация работы с файлами

Когда поток открывается для ввода-вывода, он связывается со стандартной структурой типа ***FILE***, которая определена в стандартной библиотеке ***<stdio.h>.***

Вся работа с файлом выполняется через ***файловую переменную - указатель*** на структуру типа ***FILE*.** Этот указатель используется во всех операциях с файлами. Он хранит идентификатор файлового потока и информацию, которая нужна, чтобы им управлять, включая указатель на его буфер, индикатор позиции в файле и индикаторы состояния.

Формат объявления указателя на файл следующий:

**FILE \*идентификатор;**

Пример

***FILE \*f;***

Каждый файл завершается маркером конца файла (end-of-file marker или EOF), Null определяет пустой указатель.

## Функции для работы с файлами

Прежде чем совершать с файлом какие-либо действия, файл должен быть открыт. Создание и выделение памяти под объект типа FILE осуществляется с помощью функции fopen

### Функция открытия файла

Функция fopen получает два аргумента – строку с адресом файла и строку с режимом доступа к файлу. Имя файла может быть как абсолютным, так и относительным.

**FILE \*fopen(const char \*fname, const char \*mode);**

Открывает файл с именем ***fname*** для работы в режиме, указанном строкой ***mode***, и fopen возвращает указатель на открытый файл (объект FILE), с помощью которого далее можно осуществлять доступ к файлу.

Имя\_файла – **const char \*fname** – это указатель на строку символов, в которой хранится имя файла и может содержать относительный или абсолютный путь к открываемому файлу:

1) "file.txt" – открывается файл file.txt из текущей папки

***Замечание:*** при запуске exe-файла «текущая папка» – та, где он находится; при отладке в IDE папка может быть иной, например, в Visual Studio при открытом консольном решении с именем Console файл следует разместить в папке Console/Console, а при запуске исполняемого файла не из IDE – в папке Console/Debug.

2) "Z:\\Иванов Иван Иванович\\АиП\\lab2\\my.dat" – открывается файл my.dat из студенческой папки Иванова на диске Z:;

Пример:

FILE \* ptrFile = fopen("Z:\\Иванов Иван Иванович\\АиП\\Lab2\\Lab2\\file2.txt", "w");

3) в следующем варианте имя файла запрашивается у пользователя:

#include <stdio.h>

#include <locale>

#include <cstdio>

void main()

{

setlocale(0, "rus");

char buf[80];

printf("\nВведите имя файла:");

fflush(stdin);

gets(buf);

FILE \* ptrFile = fopen(buf, "w");

if (ptrFile != NULL)

{

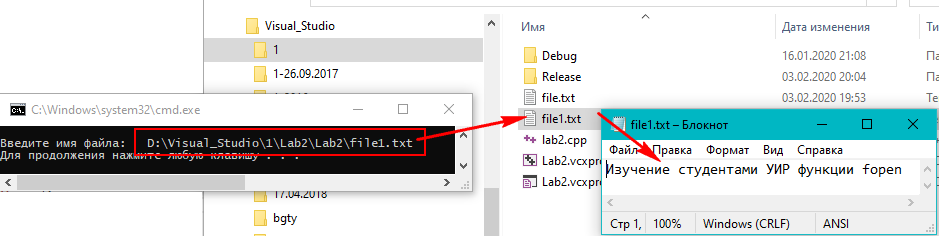
fputs("Изучение студентами УИР функции fopen ", ptrFile);

// записать строку в файл

}

fcloseall();

}



Для открытия файла используется также функция, общий вид которой:

**fopen\_s(FILE\*\* pFile, const char \*filename, const char \*mode);**

где pFile – указатель на файловый указатель.

Режим доступа (открытия) ***mode*** – это указатель на строку символов, в которой указывается режим открытия файла и тип файла.

Допустимые режимы:

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| **r** | Файл открывается только для чтения. |
| **w** | Файл открывается только для записи. Файл будет создан. Если соответствующий физический файл существует, он будет перезаписан. |
| **a** | Файл открывается для записи в конец (для дозаписи) или создается, если не существует. |
| **r+** | Файл открывается для чтения и записи. |
| **w+** | Файл открывается для записи и чтения. Если соответствующий физический файл существует, он будет перезаписан. |
| **a+** | Файл открывается для записи в конец (для дозаписи) или создается, если не существует. |

**Типы файла**

Типы файла **бинарный** (b); **текстовый** (t). *По умолчанию используется текстовый тип*. При работе с текстовыми файлами к символу, указывающему режим открытия, добавляется символ «***t***» (можно опустить), а при работе с бинарными – «***b***».

Например:

FILE \*f = fopen(s, "wb"); // s – строка, где прописан путь к файлу

FILE \*k = fopen("h:\ex.dat", "rb");

**Возвращаемое значение**

Если файл был успешно открыт, функция возвращает указатель на объект файла, который используется для идентификации потока и выполнения операций с файлом. В противном случае, возвращается нулевой указатель Null.

Поэтому после открытия файла следует обязательно проверить, удалась ли эта операция. Для этого надо сравнить указатель, который вернула fopen, с константой NULL (nullptr) из стандартной библиотеки:

FILE \* fp = fopen("text.txt", "r+b");

if (fp = = NULL)

{

//Обработка ситуации "Не удалось открыть файл"

}

Проблему вызывает случай, когда открывается сразу несколько файлов: если один из них нельзя открыть, то остальные также должны быть закрыты.

### Функция закрытия файла

**int fclose(FILE \*указатель\_на \_файл);**

Закрывает поток, который был открыт с помощью вызова fopen() и записывает в файл все данные, которые еще оставались в дисковом буфере. Доступ к файлу после выполнения функции будет запрещен. Возвращение нуля означает успешную операцию закрытия. В случае ошибки возвращается EOF.

Пример:

FILE \*k = fopen("h:\ex.dat", "rb");

fclose(k);

Функция

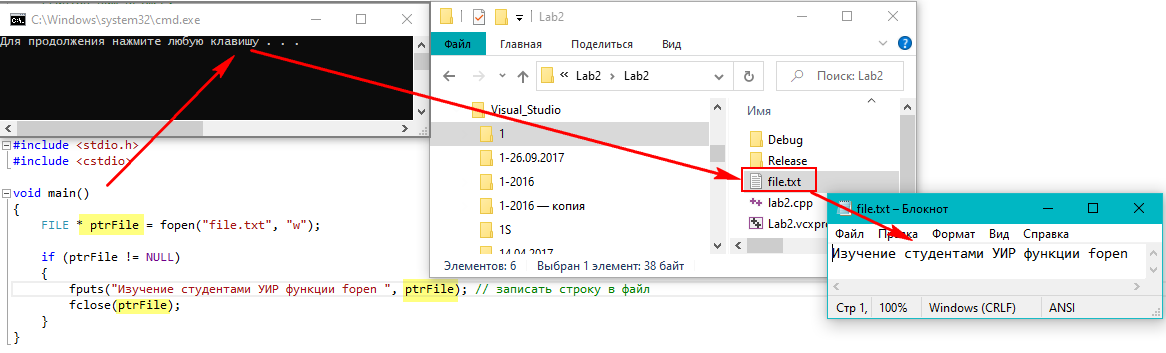
**int fcloseall(void);**

Закрывает все открытые файлы, или EOF, если возникает ошибка.

Пример:

fcloseall();

Пример: записать в файл *file.txt*, открытый по записи w, строку "Изучение студентами УИР функции fopen".



***Замечание:*** Функции, возвращающие указатель, в том числе, ***fopen***, считаются небезопасными в ряде новых компиляторов, например, ***Visual Studio 2015***. Если их использование приводит не просто к предупреждению, а к генерации ошибок, есть 2 основных способа решения проблемы:

1) в соответствии с рекомендациями компилятора, заменить старые названия функций на их безопасные версии, например, strcpy на ***strcpy\_s*** и fopen на ***fopen\_s***. При этом может измениться и способ вызова функций, например,

FILE \*myfile;

fopen\_s(&myfile, "data.txt", "wt");

вместо

FILE \*myfile = fopen\_s("data.txt", "wt");

2) в начало файла (до всех #include) включить директиву

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <locale>

#include <cstdio>

### Буферизация данных

Когда данные выводятся, они сначала помещаются в буфер. Очистка буфера осуществляется

* если он заполнен;
* если поток закрывается;
* если явно указывается, что необходимо очистить буфер;
* очищается, если программа завершилась удачно. вместе с этим закрываются и все файлы. в случае ошибки выполнения этого может не произойти.

Форсировать выгрузку буфера можно с помощью вызова функции fflush. Прототип функции fflush:

**int fflush(FILE \* filestream);**

Пример с очисткой:

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

void main() {

FILE \*file;

char c;

file = fopen("C:/c/test.txt", "w");

do {

c = getch();

fprintf(file, "%c", c);

fprintf(stdout, "%c", c);

fflush(file);

} while (c != 'q');

fclose(file);

getch();

}

Если данный поток был открыт для записи (или, если он был открыт для обновления и последняя операция ввода/вывода была операцией вывода) любые незаписанные данные в выходном буфере записываются в файл.

Если аргумент является нулевым указателем, то открытый файл очищается. Поток остается открытым после этого вызова.

## Форматированный файловый ввод-вывод

### fscanf

Выбор с**пособа чтения или записи данных** зависит от того, какой должна быть структура файла. Если файл *форматированный*, то есть, является текстовым и состоит из лексем, разделённых стандартными разделителями (пробел, табуляция, перевод строки), обмен данными с ним можно выполнять методами (функциями). Функции fscanf() и fprintf() позволяют осуществить форматированный ввод-вывод аналогичный соответствующим функциям форматированного ввода-вывода, только делают это применительно к файлу. Первым параметром этих функций указывается файловая переменная, в остальном работа совпадает со стандартными scanf и printf.

***fscanf*** –функция форматного ввода предназначена для ввода информации с преобразованием ее из текстового представления в бинарное. Информацию, записанную в текстовый файл в привычном для человека виде (т.е. так, что ее можно прочитать или ввести в файл, используя текстовый редактор), функция fscanf читает из текстового файла и преобразует ее во внутреннее представление данных в памяти компьютера. Информация о количестве читаемых элементов, их типах и особенностях представления задается с помощью формата. В случае функции ввода формат – это строка, содержащая описания одного или нескольких вводимых элементов. Форматы, используемые функцией fscanf, аналогичны применяемым функцией scanf. Каждый элемент формата начинается с символа процента "%".

***Внимание:*** fscanf считывает строку только до первого пробела.

Прототип функции fscanf:

**int fscanf(FILE \*f, const char \*format, ...);**

где f – указатель на объект типа FILE, который связан с потоком;

format – Cи-строка, содержащая текст, который будет выведен на поток. Опционально, строка может содержать встроенные метки форматирования, которые заменяются значениями, указанными в последующих дополнительных аргументах и отформатированы требуемым образом.

### fprintf

Функция fprintf используется для форматного вывода в файл. Данные при выводе преобразуются в их текстовое представление в соответствии с форматной строкой. Ее отличие от форматной строки, используемой в функции ввода fscanf, заключается в том, что она может содержать не только форматы для преобразования данных, но и обычные символы, которые записываются без преобразования в файл. Форматы, как и в случае функции fscanf, начинаются с символа процента "%". Они аналогичны форматам, используемым функцией fscanf. Небольшое отличие заключается в том, что форматы функции fprintf позволяют также управлять представлением данных, например, указывать количество позиций, отводимых под запись числа, или количество цифр после десятичной точки при выводе вещественного числа.

Прототип функции fprintf:

**int fprintf(FILE \* f, const char \* format, ...);**

Пример: Дан текстовый файл. Подсчитать количество в нем слов и вывести на экран.

*#include <string.h>*

*#include <stdio.h>*

*#include <iostream>*

*#include <fstream>*

*#include <ctype.h>*

*using namespace std;*

*void main()*

*{*

*setlocale(LC\_ALL,"Russian");*

*FILE \*in;*

*int ch, word=0;*

*if((in=fopen("Стр36№13.txt","r"))!=NULL)*

*{*

*if(in!=NULL)*

*puts("Файл ""Стр36№13.txt"" найден\n");*

*while (!feof(in))*

*{*

*fscanf(in, "%c", &ch);*

*putc(ch,stdout);*

*if(isspace((unsigned char)ch)!=0) // unsigned char для русского алфавита*

*{*

*word++;*

*}*

*}*

*word++;*

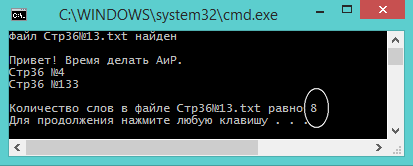
*printf("\n\nКоличество слов в файле ""Стр36№13.txt"" равно %d\n", word);*

*fclose(in);*

*}*

*}*

}

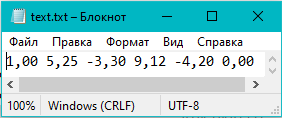


Функция ***isspace*** проверяет параметр сharacter, является ли он символом пробела. Обратите внимание на то, что символ пробела – это, на самом деле, несколько символов.

Для данной функции символами пробела являются следующие символы:

|  |  |  |
| --- | --- | --- |
| **' '** | 0x20 | Символ пробела (SPC) |
| **'\t'** | 0x09 | Горизонтальная табуляция (TAB) |
| **'\n'** | 0x0a | Символ новой строки (LF) |
| **'\v'** | (0x0b) | Вертикальная табуляция (VT) |
| **'\f'** | (0x0c) | feed (FF) |
| **'\r'** | (0x0d) | возврат каретки на начало строки (CR) |

Пример. Создаем текстовый документ text.txt:



Считаем из него данные и выведем на экран.

#include <string.h>

#include <stdio.h>

#include <iostream>

#include <fstream>

#include <ctype.h>

using namespace std;

void main()

{

setlocale(LC\_ALL, "Russian");

int i = 1;

FILE \*fp1 = fopen("text.txt", "w"); //Создание файла

float a = 1;

cout << "Введите число а ( 0 - завершение): " << endl;

while (a!=0) {

cin >> a;

fprintf(fp1, "%4.2f%c", a,' '); // цифры через пробел 1.0 5.25 -3.3 9.12 -4.2

}

fclose(fp1);

FILE \*fp = fopen("text.txt", "r");

if (fp == NULL) {

printf("\nНе удалось открыть файл"); getchar(); exit(1);

}

while (!feof(fp)) {

i = fscanf(fp, "%f", &a);

if (feof(fp)) break; //Если файл кончился, выйти из цикла

//здесь выполняется обработка очередного значения a, например:

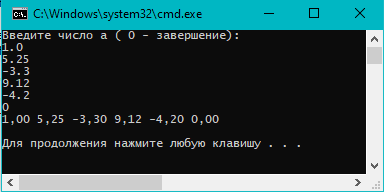
printf("%.2f ", a);

}

printf("\n\n");

fclose(fp);

}



***Замечание:*** Ввод/вывод функциями библиотеки <stdio.h> буферизован, то есть, данные помещаются в область памяти заданного размера, обмен данными происходит не отдельными байтами, а «порциями». Поэтому перед чтением данных желательно очищать буфер от возможных «остатков» предыдущего чтения методом ***fflush***, а после записи данных следует обязательно закрывать файл методом ***fclose***, иначе данные могут быть потеряны. Заметим, что консольный ввод/вывод «обычными» методами scanf и printf также буферизован.

### Функции fgetc() и fputc(),fgets() и fputs()

Если текстовый файл состоит из неструктурированных строк (абзацев) текста, разделённых символами перевода строки, то при работе с такими данными используются следующие функции:

* fgetc и fputc (записывает один символ в текущую позицию указанного открытого файла) – для посимвольного чтения и посимвольной записи данных.

Прототипы:

int getc(FILE \* указатель\_на \_файл);

int putc(int символ, FILE \* указатель\_на \_файл);

* fgets и fputs – для чтения и записи строк с указанным максимальным размером.

Прототипы:

**char \*fgets(char \*строка, int длина, FILE \*указатель\_на \_файл);**

– читает строку символов из текущей позиции указанного открытого файла до тех пор, пока не будет прочитан символ перехода на новую строку, или количество прочитанных символов не станет равным (длина– 1);

**int fputs(const char \*строка, FILE \*указатель\_на \_файл);**

– записывает строку символов в текущую позицию указанного открытого файла.

***Замечание:*** Без дополнительной обработки прочитанные из файла строки при выводе будут содержать лишние пустые строки между строками данных. Это происходит потому, что функция fgets читает строку файла вместе с символом перевода строки (точней, под Windows – с парой символов \r\n, интерпретируемых как один), а puts добавляет к выводимой строке ещё один перевод строки.

Пример: переписываем данные из одного файла (\*in) и файл (\*out).

#include <string.h>

#include <stdio.h>

#include <iostream>

#include <cctype>

using namespace std;

void main()

{

setlocale(LC\_ALL,"Russian");

FILE \*in, \*out;

int ch;

if((in=fopen("in1.txt","r"))!=NULL)

{

if(in!=NULL)

puts("Файл in1.txt найден");

out=fopen("temp.txt","w");

puts("Файл temp.txt создан");

while((ch=getc(in))!=EOF)

putc(ch,out);

puts("Копирование успешное\n");

fclose(in);

fclose(out);

}

if((in=fopen("in2.txt","r"))!=NULL)

{

if(in!=NULL)

puts("Файл in2.txt найден");

out=fopen("in1.txt","w");

while((ch=getc(in))!=EOF)

putc(ch,out);

puts("Копирование успешное\n");

fclose(in);

fclose(out);

}

if((in=fopen("temp.txt","r"))!=NULL)

{

if(in!=NULL)

puts("Файл temp.txt найден");

out=fopen("in2.txt","w");

while((ch=getc(in))!=EOF)

putc(ch,out);

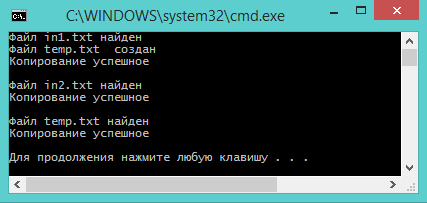
puts("Копирование успешное\n");

fclose(in);

fclose(out);

}

}



## Прототип функции feof

**int feof(FILE \* filestream);**

**\* filestream –** указатель на объект типа FILE, идентифицируемый поток.

Функция feof проверяет, достигнут ли конец файла, связанного с потоком, через параметр filestream. Возвращается значение, отличное от нуля, если конец файла был действительно достигнут.

Вызов данной функции, как правило, выполняется после выполнения предыдущей операции с потоком, например операции считывания, которая постепенно двигает внутренний указатель файла в конец.

## Неформатированные файловый ввод-вывод

С помощью функций fwrite() и fread() осуществляется блоковый ввод/вывод и работа с бинарными файлами.

### Запись в файл fwrite

*fwrite(адрес записываемой величины, размер одного экземпляра, количество записываемых величин, имя логического файла);*

Прототип:

**size\_t fwrite(const void \* ptrvoid, size\_t size, size\_t count, FILE \* filestream);**

где:

* ptrvoid – указатель на массив элементов, которые необходимо записать в файл;
* size – размер в байтах каждого элемента массива;
* count – количество элементов, каждый из которых занимает size байт;
* filestream – указатель на объект типа FILE, который связан с потоком вывода.

Функция fwrite записывает массив размером – count элементов, каждый из которых имеет размер size байт, в блок памяти, на который указывает ptrvoid — текущая позиция в потоке. Индикатор положения потока увеличивается на общее число записанных байтов. Общее количество записанных байт (count\*).

Возвращаемое значение

Общее число элементов, которые успешно были записаны. Возвращаемое значение, в этом случае имеет тип данных size\_t.

Если возвращаемое значение отличается от количества элементов, значит произошла ошибка.

Например,

void main()

{

setlocale(LC\_ALL, "Russian");

int i = 1;

FILE \*fp1 = fopen("text1.txt", "w"); //Создание файла

float a = 1;

cout << "Введите число а ( 0 - завершение): " << endl;

while (a!=0)

{

cin >> a;

fwrite(&a, sizeof(float), 1, fp1);

}

fclose(fp1);

printf("\n\n");

}

### Чтение из файла fread

*fread(адрес величины, размер одного экземпляра, количество считываемых величин, имя логического файла);*

Прототип функции:

**size\_t fread(void \* ptrvoid, size\_t size, size\_t count, FILE \* filestream);**

где

* ptrvoid – указатель на блок памяти, размер которого должен быть минимум  (size\*count) байт;
* size – размер в байтах каждого считываемого элемента;
* count – количество элементов, каждый из которых имеет размер size байт;
* filestream – указатель на объект типа FILE, который связан с потоком ввода.

Функция fread считывает массив размером – count элементов, каждый из которых имеет размер size байт, из потока, и сохраняет его в блоке памяти, на который указывает ptrvoid. Индикатор положения потока увеличивается на общее число записанных байтов. Общее количество успешно считанных байт (count\*).

Возвращается объект типа size\_t, который содержит общее количество, успешно считанных, элементов. Если возвращаемое значение отличается от количества элементов, значит произошла ошибка или был достигнут конец файла.

Например,

#include <string.h>

#include <stdio.h>

#include <iostream>

#include <fstream>

#include <ctype.h>

using namespace std;

void main()

{

setlocale(LC\_ALL, "Russian");

int i = 1;

float a = 1;

FILE \*fp = fopen("text1.txt", "r");

if (fp == NULL) {

printf("\nНе удалось открыть файл"); getchar(); exit(1);

}

while (!feof(fp)) {

fread(&a, sizeof(float), 1, fp);

printf("%.2f ", a);

}

printf("\n\n");

fclose(fp);

}

**Пример.** Заполнить файл некоторым количеством целых случайных чисел. Вывести содержимое созданного файла на экран.

#include <cstdlib>

#include <iostream>

#include <ctime>

using namespace std;

int main()

{

FILE \*f, \*f1;

int dat;

cout << endl << endl << "In file : " << endl;

srand(time(0));

int n = rand() % 30 + 1;

cout << "File name? ";

char s[20];

cin.getline(s, 20);

f = fopen(s, "wb");

for (int i = 1; i <= n; i++)

{

dat = rand() % 101 - 50;

cout << dat << " ";

fwrite(&dat, sizeof(int), 1, f);

}

cout << endl << endl << "From file : " <<endl;

fclose(f);

f1 = fopen(s, "rb");

for (int i = 1; i <= n; i++)

{

fread(&dat, sizeof(int), 1, f);

cout << dat << " ";

}

cout << endl;

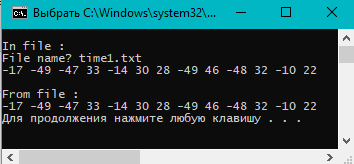
fclose(f1);

system("PAUSE");

return EXIT\_SUCCESS;

}

Макрос EXIT\_SUCCESS используется для возвращения ОС кода успешного завершения программы. Этот макрос заменяется на системно-зависимое выражение, которые используется в качестве аргумента для функции exit. Оно должно означать, что работа программы была завершена успешно. Противоположное значение может быть задано макросом EXIT\_FAILURE.



**Пример.** Найти сумму и количество целых чисел, записанных в бинарный файл.

#include <cstdlib>

#include <iostream>

using namespace std;

int main()

{

FILE \*f;

**int** **dat**, n = 0, sum = 0;

cout << "File name? ";

char s[20];

cin.getline(s, 20);

f = fopen(s, "rb");

while (fread(&dat, **sizeof(int)**, 1, f)) //пока не конец файла

{

n++;

cout << dat << " ";

sum += dat;

}

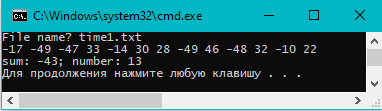
cout << endl;

cout << "sum: " << sum << "; number: " << n << endl;

fclose(f);

return EXIT\_SUCCESS;

}



**Пример.** Поместить в файл *n* записей, содержащих сведения о кроликах, содержащихся в хозяйстве: пол (m/f), возраст (в мес.), масса.

#include <cstdlib>

#include <iostream>

using namespace std;

struct krolik

{

char pol;

int vozrast;

double massa;

};

int main()

{

FILE \*f; krolik dat; int n;

cout << "File name? ";

char s[20];

cin.getline(s, 20);

f = fopen(s, "wb");

cout << "How many rabbits? "; cin >> n;

for (int i = 1; i <= n; i++)

{

cout << "What pol " << i << "th rabbit? "; cin >> dat.pol;

cout << "How old " << i << "th rabbit? "; cin >> dat.vozrast;

cout << "What is the mass of the " << i << "th rabbit? "; cin >> dat.massa;

fwrite(&dat, **sizeof(krolik)**, 1, f);

}

fclose(f);

return EXIT\_SUCCESS;

}

**Пример (продолжение).** В бинарном файле хранятся сведения о кроликах, содержащихся в хозяйстве: пол (m/f), возраст (в мес.), масса. Найти наиболее старого кролика. Если таких несколько, то вывести информацию о том из них, масса которого больше.

#include <cstdlib>

#include <iostream>

using namespace std;

struct krolik

{

char pol;

int vozrast;

double massa;

};

int main()

{

FILE \*f; krolik dat, max; int n;

cout << "File name? ";

char s[20];

cin.getline(s, 20);

f = fopen(s, "rb");

fread(&dat, sizeof(krolik), 1, f);

max = dat;

while (fread(&dat, **sizeof(krolik)**, 1, f))

{

if (dat.vozrast>max.vozrast) max = dat;

else if (dat.vozrast = = max.vozrast && dat.massa>max.massa) max = dat;

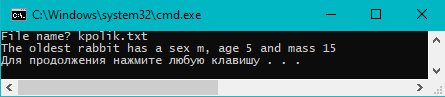
}

cout << "The oldest rabbit has a sex " << max.pol << ", age " << max.vozrast << " and mass " << max.massa << endl;

system("PAUSE");

return EXIT\_SUCCESS;

}



## ЗАДАНИЕ

В соответствии со своим вариантом разработать программу для работы с файлами на языке С. Добавить меню:

1. **Создать файлы с расширением txt**. Ввести данные в файл.

2. Считать записи из файла.

3. Добавить записи в файл.

4. **Промежуточные и итоговые результаты вывести на экран (файлы, суммы, числа, …).**

В справочной аэропорта хранится расписание вылета самолетов на следующие сутки. Для каждого рейса указаны номер рейса, тип самолета, количество посадочных мест, пункт назначения, время вылета. Имеются справочники по расстояниям между всеми возможными пунктами назначения и по расходу горючего на тысячу километров для каждого типа самолета. Написать программу для хранения указанной информации в файлах. Данная программа, кроме того, должна формировать в виде текстового файла заявку на горючее на следующий день (количество в тоннах). Три массива структур, первая заполняется вручную, 2 другие из текстового файла. Реализовать при помощи меню (вывод содержимого, формирование заявки и т.д).