Лабораторная работа №3
Тема: Текстовые файлы.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Файл – это именованный объект, хранящий данные (программа или любая другая информация) на каком-либо носителе. Файл, как и массив, – это совокупность данных.

Отличия файла от массива:

1. Файлы в отличие от массивов располагаются не в оперативной памяти, а на жестких дисках или на внешних носителях, хотя файл может располагаться на так называемом электронном диске (в оперативной памяти).
2. Файл не имеет фиксированной длины, т.е. может увеличиваться и уменьшаться.
3. Перед работой с файлом его необходимо открыть, а после работы – закрыть.

Файловая система – это совокупность файлов и управляющей информации на диске для доступа к файлам. Или по-другому – это совокупность программных средств для доступа к файлам. Существует довольно много файловых систем и правила именования файлов в них могут незначительно отличаться.

Имена файлов состоят из *двух частей*, разделенных точкой: имя файла и расширение. Файлы хранятся в *каталогах (директориях)*. Каталоги могут иметь такие же имена, что и файлы. Допускаются вложенные каталоги (подкаталоги).

Различают два вида файлов: *текстовые и бинарные*.

Текстовые файлы могут быть просмотрены и отредактированы с клавиатуры любым текстовым редактором и имеют очень простую структуру: последовательность ASCII-символов. Эта последовательность символов разбивается на строки, каждая из которых заканчивается двумя кодами: 13, 10 (0xD, 0xA). Примеры известных текстовых фалов: *.bat, *.c, *.asm, *.cpp.

Бинарные файлы – это файлы, которые не имеют структуры текстовых файлов. Каждая программа для своих бинарных файлов определяет собственную структуру.

Библиотека языка C/C++ содержит функции для работы как с текстовыми, так и с бинарными файлами.

Функции открытия и закрытия файла. Перед работой с файлом, его необходимо открыть.

Функция открытия файла *fopen():*

```
FILE *fopen(char *filename, char *mode);
```

где *FILE* – структурный тип, который связан с физическим файлом и содержит всю необходимую информацию для работы с ним (указатель на текущую позицию в файле, тип доступа и др.).

`char *filename` - задает физическое местонахождение (*путь*) и *имя* открываемого файла;

`char *mode` - тип доступа к файлу, который может принимать значения, указанные в таблице.

Функция *fopen()* при успешном открытии файла возвращает указатель на структуру типа *FILE*, называемый *указателем на файл*. Эта структура связана с физическим файлом и содержит всю необходимую информацию для работы с ним (указатель на текущую позицию в файле, тип доступа и др.). Возвращаемое функцией значение нужно сохранить и использовать для ссылки на открытый файл. Если произошла ошибка при открытии файла, то возвращается *NULL*.

Таблица**:** тип доступа к файлам.

“r”	Открыть файл для чтения.
“w”	Открыть файл для записи. Если файл существует, то его содержимое теряется.
“a”	Открыть файл для записи в конец файла. Если файл не существует, то он создается.
“r+”	Открыть файл для чтения и записи. Файл должен существовать.
“w+”	Открыть файл для чтения и записи. Если файл существует, то его содержимое теряется.
“a+”	Открыть файл для чтения и записи в конец файла. Если файл не существует, то он создается.

К комбинациям вышеперечисленных литералов могут быть добавлены также “t” либо “b”:

“t”	Открыть файл в текстовом режиме.
“b”	Открыть файл в бинарном режиме.

Возможны следующие режимы доступа: “*w+b*”, “*wb+*”, “*rw+*”, “*w+t*”, “*rt+*” и др. Если режим не указан, то файл открывается в текстовом режиме.

После работы с файлом он должен быть закрыт функцией *fclose()*.

Для этого необходимо в указанную функцию передать указатель на *FILE*, который был получен при открытии функцией *fopen()*. При завершении программы незакрытые файлы автоматически закрываются системой.

Стандартная последовательность операторов, необходимая для открытия и закрытия файла:

```
#include <stdio.h>

FILE *f;
if(!(f = fopen(“readme.txt”, ”r+t”)))
{
  printf(“Невозможно открыть файл \n”);  return;
}
// Работа с файлом
fclose(f);                        // Закрытие файла
```

Функции чтения/записи в файл. Функции для работы с текстовым файлом: *fprintf(), fscanf(), fgets(), fputs()*. Формат параметров этих функций очень похож на формат функций *printf(), scanf(), gets()* и *puts()*. Схожи не только параметры, но и действия. Отличие лишь в том, что *printf(), scanf()* и другие работают по умолчанию с консолью (экран, клавиатура), а *fprintf(), fscanf()* – с файлами (в том числе и со стандартными потоками *stdin*, *stdout* и др.), поэтому у них добавлен параметр, являющийся указателем на структуру *FILE*, которая была рассмотрена выше.

Пример 1*: Записать в текстовый файл числа от 0 до 1000 кратные 3.*

```
#include "stdafx.h"
#include<stdio.h>

int _tmain(int argc, _TCHAR* argv[])
{
  int a;
  FILE *f;
  if(!(f = fopen("test.txt", "w+t")))
  {
    printf("Невозможно создать файл\n");    return 0;
  }
  for(a=0;a<1000;a+=3)
  {
    fprintf(f, "%d, ",a);
  }
  printf("Данные записаны в файл\n");
  fclose(f);
  return 0;
}
```

Функции для работы с текстовыми файлами удобно использовать при создании текстовых файлов, ведении файлов-протоколов (log-файлов) и т.п. Но при создании баз данных целесообразно использовать функции для работы с бинарными файлами: *fwrite()* и *fread()*. Эти функции без каких-либо изменений копируют блок данных из оперативной памяти в файл и

соответственно из файла – в память. Такой способ обмена данными требует меньше времени:

```
     unsigned fread (void *ptr, unsigned size, unsigned n, FILE
*stream);
     unsigned fwrite(void *ptr, unsigned size, unsigned n,
FILE*stream);
```

где: **ptr* – указатель на буфер;

size – размер блока;

n – количество блоков;

stream* – указатель на структуру ***FILE* открытого файла.

Пример 2: Записать в бинарный файл и прочитать из бинарного файла список абитуриентов, информация об абитуриенте представлена структурой.

```
#include "stdafx.h"
#include <stdio.h>

struct abitur
{ char name[32];
  int mark[3];
};

int _tmain(int argc, _TCHAR* argv[])
{ struct abitur inf;
  int a;
  FILE *f;

  if(!(f=fopen("inf.dat","w+")))
  { printf("Ошибка создания файла\n"); return 0; }
  for(;;)
  { printf("Введите ФИО (пустая строка -- конец списка): ");
    fflush(stdin);
    gets(inf.name);
    if(!inf.name[0]) break;
    printf("\n Введите три оценки, полученные на экзаменах:
");
    scanf("%d%d%d", &inf.mark[0], &inf.mark[1],
&inf.mark[2]);
    fwrite(&inf, 1,sizeof(inf), f);
  }
  fclose(f);

  printf("\nСписок абитуриентов:\n");
  if(!(f=fopen("inf.dat","r")))
  {  printf("Ошибка создания файла\n"); return 0;}
  while(1)
```

```
        {   if(sizeof(inf) != fread(&inf, sizeof(inf), 1,f))
            break;   /* Если не удалось прочитать необходимое
                        количество байт, то заканчиваем чтение */
            printf("%s  %d  %d  %d  \n", inf.name, inf.mark[0],
inf.mark[1], inf.mark[2]);
        }
        fclose(f);
        return 0;
    }
```

Важно понимать, что нет однозначного метода, который можно было бы использовать для отличия текстового файла от бинарного, поэтому любой бинарный файл может быть открыт для работы с ним как с текстовым; а текстовый может быть открыт как бинарный. Но такой работы с файлом обычно приводит к ошибкам. Поэтому рекомендуется работать с конкретным файлом в том режиме, в котором он был создан.

Позиционирование в файле. Каждый открытый файл имеет, так называемый *указатель на текущую позицию* в файле (это нечто подобное указателю в памяти). Все операции над файлами (чтение и запись) работают с данными с этой позиции. При каждом выполнении функции чтения или записи указатель смещается на количество прочитанных или записанных байт, т.е. устанавливается сразу за прочитанным или записанным блоком данных в файле. В этом случае осуществляется так называемый *последовательный доступ* к данным, который очень удобен, когда нам необходимо последовательно работать с данными в файле. Это демонстрируется во всех вышеприведенных примерах чтения и записи в файл. Но иногда необходимо читать или писать данные в произвольном порядке, что достигается путем установки указателя на некоторую заданную позицию в файле функцией *fseek()*.

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

Параметр *offset* задает количество байт, на которое необходимо сместить указатель соответственно параметру *whence*. Приводим значения, которые может принимать параметр *whence*:

SEEK_SET	0	Смещение выполняется от начала файла.
SEEK_CUR	1	Смещение выполняется от текущей позиции указателя.
SEEK_END	2	Смещение выполняется от конца файла.

Величина смещения может быть как положительной, так и отрицательной, но нельзя смещаться за пределы начала файла.

Такой доступ к данным в файле называют *произвольным*.

Пример 3: *Найти по номеру запись из бинарного файла, который был создан в примере 2.*

```
#include "stdafx.h"
#include <stdio.h>

struct abitur
{ char name[32];
  int mark[3];
};
int _tmain(int argc, _TCHAR* argv[])
{ struct abitur inf;
  int n;
  FILE *f;
  if(!(f=fopen("inf.dat","r")))
  { printf("Ошибка создания файла\n");
    return 0;
  }
  while(1)
  { printf("Введите номер записи (0 - выход): ");
    scanf("%d", &n);
    if(!n) break;
    fseek(f,sizeof(struct abitur)*(n-1),SEEK_SET);
    if(sizeof(inf) != fread(&inf,1,sizeof(inf),f))
      printf("Конец списка\n");
    else
      printf("%s %d %d %d", inf.name, inf.mark[0],
inf.mark[1], inf.mark[2]);
  }
  fclose(f);
  return 0;
}
```

Иногда необходимо определить позицию указателя. Для этого можно воспользоваться функцией ***ftell()***:

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

Возвращает значение указателя на текущую позицию файла. В случае ошибки возвращает число (-1).

int fsetpos(FILE *stream, **const long** *pos) - устанавливает значение указателя чтения-записи (указатель на текущую позицию) файла в позицию заданную значением по указателю **pos**. Возвращает нуль при корректном выполнении, и любое не нулевое значение при ошибке.

int fgetpos(FILE *stream, **long** *pos) - помещает в переменную, на которую указывает **pos**, значение указателя на текущую позицию в файле. Возвращает нуль при корректном выполнении, и любое не нулевое значение при ошибке. [3]

Пример 4: Программа создает простой файл последовательного доступа, который можно использовать в программе учета оплаты счетов, помогающей следить за суммами задолженности клиентов компании. Для каждого клиента программа просит ввести номер счета, имя клиента и баланс (то есть сумму, которую клиент должен компании за товары и услуги, полученные в прошлом). Данные на каждого из клиентов образуют "запись" для этого клиента. В этом приложении в качестве ключа записи используется номер счета. Другими словами, файл будет создаваться и поддерживаться упорядоченным по номерам счетов. Эта программа подразумевает, что пользователи вводят записи в порядке возрастания номеров. В более удобной для работы системе регистрации счетов должна обеспечиваться возможность сортировки, чтобы пользователь мог вводить записи в произвольном порядке. В этом случае записи должны сначала упорядочиваться, а затем уже записываться в файл.

```
#include "stdafx.h"
#include <stdio.h>

int _tmain(int argc, _TCHAR* argv[])
{
  int account;
  char name[30];
  float balance;
  FILE *cfPtr;
  if ((cfPtr =  fopen("clients.dat", "w")) == NULL)
       printf ("File could not be opened\n");
  else {
       printf("Enter the account, name, and balance.\n");
       printf ("Enter EOF to end input (Ctrl+Z).\n");
       printf("? ");
       scanf("%d%s%f", &account, name, &balance);
       while ( !feof(stdin)) {
            fprintf(cfPtr, "%d %s %.2f\n", account, name,
balance);
            printf("? ");
            scanf("%d%s%f", &account, name, &balance);
       }
  fclose (cfPtr);
  }
  return 0;
}
```

Пример 5: Чтение и распечатка последовательного файла, созданного в предыдущем примере.

```
#include "stdafx.h"
#include <stdio.h>

int _tmain(int argc, _TCHAR* argv[])
{
```

```
          int account;
          char name[30];
          float balance;
          FILE *cfPtr;
          if ((cfPtr = fopen("clients.dat", "r")) == NULL)
               printf("File could not be openedXn") ;
          else {
               printf("%-10s%-13s%s\n", "Account", "Name",
"Balance");
               fscanf(cfPtr, "%d%s%f", &account, name, &balance);
               while (!feof(cfPtr)) {
                    printf("%-10d%-13s%7.2f\n", account, name,
balance);
                    fscanf(cfPtr, "%d%s%f", &account, name,
&balance);
               }
          fclose(cfPtr);
          }
          return 0;
        }
```

Пример 6: Программа позволяет менеджеру по кредиту получить список клиентов с нулевым сальдо (клиентов, которые не должны денег), клиентов с положительным сальдо (которым компания должна какую-то сумму денег) и клиентов с отрицательным сальдо (которые должны компании деньги за уже полученные товары и услуги).

```
#include "stdafx.h"
#include <stdio.h>

int _tmain(int argc, _TCHAR* argv[])
{
int request, account;
float balance;
char name[30];
FILE *cfPtr;
if ((cfPtr = fopen("clients.dat", "r")) == NULL)
    printf("File could not be opened\n");
else {
    printf("Enter request\n");
    printf(" 1 - List accounts with zero balances\n");
    printf(" 2 - List accounts with credit balances\n");
    printf(" 3 - List accounts with debit balances\n");
    printf(" 4 - End of run\n? ");
    scanf("%d", &request);
         while (request != 4) {
         fscanf(cfPtr, "%d%s%f", &account, name, &balance);
         switch (request) {
         case 1:
              printf("\nAccounts with zero balances:\n");
              while(!feof(cfPtr)) {
```

```
                    if (balance ==0)
                    printf("%-10d%-13s%7.2f\n", account,
name, balance);
                    fscanf (cfPtr, "%d%s%f", &account, name,
&balance);
                }
            break;
            case 2:
                printf("\nAccounts with credit balances :
\n") ;
                while (!feof(cfPtr)) {
                    if (balance < 0)
                    printf("%-10d%-13s%7.2f\n", account,
name, balance);
                    fscanf(cfPtr, "%d%s%f", &account, name,
&balance); }
            break;
            case 3:
                printf("\nAccounts with debit balances: \n");
                while (!feof(cfPtr)) {
                    if (balance >0)
                    printf("%-10d%-13s%7.2f\n", account,
name, balance);
                    fscanf(cfPtr, "%d%s%f", &account, name,
&balance);
                }
            break;
        }
        rewind(cfPtr);
        printf("\n? ");
        scanf("%d", &request);
        }
        printf("End of run.\n");
        fclose(cfPtr);
    }
    return 0;
    }
```

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Варианты заданий

Задача 1 - максимум 4 балла.

Задача 1 + 2 - максимум 8-9 баллов.

Вариант	Задание
1	1. Дан текстовый файл F1. Ввести с клавиатуры число N. Определить сколько в файле имеется слов, состоящих из N символов, и записать это число в новый файл F2. 2. Даны текстовый файл и символьная строка s. Удалить из файла все строки, кроме тех, которые содержат в качестве фрагмента строку s. Дополнительных файлов и коллекций не использовать.
2	1. Дан текстовый файл F1. В новый файл F2 записать текст файла F1, удаляя при этом все лишние пробелы. Дополнительных коллекций не использовать. Пример. Входной файл: “ аа бб ”, выходной файл: “аа бб”. 2. Дан текстовый файл F1. Преобразовать файл так, чтобы каждая последовательность одинаковых символов была представлена как NS, где N - количество последовательных повторяющихся символов, а S - сам символ. Пример: “AABBB\1Fdddss” -> “2A3B1\1F3d2ss”. Также реализовать обратное преобразование. Дополнительных файлов и коллекций не использовать.
3	1. Дан текстовый файл F1 и последовательность символов. В новый файл F2 записать данные из файла F1, исключая данную последовательность. Дополнительных коллекций не использовать. 2. Дан текстовый файл F1. В файле записана матрица целочисленных элементов. По запросу с клавиатуры удалить столбец или строку с минимальной суммой элементов. Дополнительных коллекций и файлов не использовать.

4	1. Даны два текстовых файла, каждый из которых содержит N чисел. Записать новый файл, содержащий сумму элементов файлов, где 1-й элемент первого складывается с последним элементом второго, 2-й - с предпоследним и т.д. Дополнительных коллекций не использовать. Дополнительных файлов и коллекций не использовать. 2. Создать два файла: F1 и F2. Целое число, введенное с клавиатуры, должно попасть в F1, если число четное и в F2, если число нечетное. При этом при добавлении числа в файл должен поддерживаться порядок чисел по возрастанию. По запросу с клавиатуры создать файл F3, который является объединением чисел, находящихся в файлах F1 и F2, с сохранением порядка по возрастанию.
5	1. Дан текстовый файл, содержащий цифры, разделенные запятой. Отсортировать цифры в файле по возрастанию, используя сортировку методом вставкок (insertion sort). 2. Даны два текстовых файла: F1 и F2. Создать текстовый файл F3, который содержит разницу этих файлов. Если строка в файле F1 и F2 совпадает, то в файле F3 она не указывается. Если строка в файле F1 и F2 не совпадает, то в файл F3 записывается номер строки и ее содержимое из файла F1 и ее содержимое из файла F2.
6	1. Инвертировать текстовый файл без использования дополнительных массивов и файлов. 2. Дан текстовый файл. С клавиатуры ввести слово. В файле удалить каждое слово, находящееся перед введенным словом. Дополнительных файлов и коллекций не использовать.
7	1. Инвертировать каждое слово в текстовом файле. 2. Дан текстовый файл F1, в котором символом ‘*’ нарисован квадрат. Создать новый файл F2, в который записать размер стороны квадрата. Из файла F3 считать новую сторону квадрата. Изменить файл F1 так, чтобы сторона нарисованного квадрата соответствовала высоте, указанной в F3. Пример квадрата со стороной 3, нарисованного символом “*”: *** * * ***

8	1. Дан текстовый файл, содержащий цифры, разделенные запятой. Отсортировать цифры в файле по убыванию, используя сортировку методом выбора (selection sort). Дополнительных файлов и коллекций не использовать. 2. Дан текстовый файл F1. В текстовом файле F2 через пробел записаны индексы символов в тексте файла F1. Удалить из файла F1 слова, в которые попадают эти индексы. Пример. Индексы: 2, 14. Текст: “abc def ghijklo rstu”. Результат: “def rstu”, т.к. Индекс 2 соответствует символу ‘b’ в слове “abc”, а индекс 14 - символу ‘o’ в слове “rstu”. Дополнительных файлов и коллекций не использовать.
9	1. Дан текстовый файл F1. В текстовый файл F2 записать все слова из F1, после которых стоит символ “!”. Слова в F2 разделить запятой и пробелом. После последнего должна стоять точка, В начале файла должно быть количество слов в этом файле. Дополнительных файлов и коллекций не использовать. 2. Дан текстовый файл F1. В текстовом файле F2 написаны правила замены подстрок вида “abc=defg”. Каждое правило записано с новой строки. Заменить все подстроки в F1 по правилам, записанным в F2. Для правила “abc=defg” нужно строку “abc” в файле F1 заменить на строку “defg”. Дополнительных файлов и коллекций не использовать.
10	1. Дан текстовый файл. Все согласные буквы сдвинуть в конец файла. Дополнительных коллекций и файлов не использовать. Предполагается, что текст - на английском языке. 2. Дан файл F1 в формате CSV (представление таблицы текстом, где столбцы разделены запятыми, а ряды - символом новой строки). В файле данные представлены тремя столбцами - номер группы студента, его фамилия (максимум 80 символов) и средний балл. Создать новый файл F2. Из F1 перенести в F2 записи всех студентов, у которых средний балл меньше 4. Из F1 этих студентов удалить. Удаление реализовать без дополнительных файлов и коллекций. Пример данных в файле F1: 999501,Иванов,9.5 999502,Петров,8.5 999501,Сидоров,7.5

11	1. Дан текстовый файл F1. Создать новый файл F2, в который записать по одному разу все символы, которые используются в файле F1. Дополнительных коллекций не использовать. 2. Дан файл с кодом на C. Реализовать одну из функциональностей проверки code-style: в случае, если подключаемые заголовочные файлы в файле с кодом расположены не в алфавитном порядке, исправить это. Дополнительных файлов и коллекций не использовать.
12	1. Дан текстовый файл F1. Создать новый файл F2, в него записать на отдельных строках все подстроки из файла F1, заключенные в кавычки. При перезапуске программы дынные а файл F2 должны дописаться в конец. 2. Дан текстовый файл. Сделать из него html-файл, оборачивая каждый абзац в тег . С клавиатуры задать цвет текста. Дополнительных файлов не создавать, дополнительных коллекций не использовать. Пример. Исходный текст: Абзац1 Абзац2 Абзац2 Результат для красного цвета текста: Абзац1 Абзац2 Абзац2
13	1. Дан текстовый файл. Заменить все символы ‘-’ на слово “дефис” в кавычках. Дополнительных файлов и коллекций не использовать. 2. Дан текстовый файл F1, в котором специальным образом выделены кодовые вставки: сначала следуют два символа ‘%’, затем в скобках название языка программирования, затем текст кода, затем два символа ‘%’. Найти все кодовые вставки и записать в новый файл F2 название языка, запятую, количество вставок. На новой строке в том же формате записывается информация по следующему языку программирования. Дополнительных файлов и коллекций не использовать. Пример текста с кодовой вставкой на c++: abcde %%(c++)int a; %% fghi

14	1. Даны N текстовый файлов, каждый с именем, являющимся его порядковым номером (“1.txt”, “2.txt”, “3.txt” и т.д.), N заранее не известно. Создать новый файл, в который записать текст из всех файлов, разделенный новой строкой. 2. Дан текстовый файл F1. Найти в этом файле все номера кредитных карт и заменить на строку “****”. Все электронные адреса, указанные в тексте файла F1 дописать в существующий файл F2. Электронным адресом будем считать строку, состоящую из произвольного количества символов, затем точка, затем строка “com”, “by” или “ru”. Дополнительных файлов и коллекций не использовать.
15	1. Дан текстовый файл F1. В текстовом файле F2 через пробел записаны порядковые номера слов из файла F1. Создать текстовый файл F3, в который записать слова из F1 под порядковыми номерами из F2. 2. Дан файл с кодом на C. Реализовать одну из функциональностей проверки code-style: проверку корректности отступов. После каждой открывающей фигурной скобки код на следующей строке должен быть на два пробела правее предыдущего и наоборот - для закрывающая скобки и кода на следующих после нее строках отступ должен уменьшаться. Исправить код в файле, если найдено ошибочное оформление. Дополнительных файлов и коллекций не использовать.