ЛАБОРАТОРНАЯ РАБОТА № 15
РАБОТА С ФАЙЛАМИ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1. ФАЙЛЫ

Файл – это именованная область памяти на внешнем запоминающем устройстве. Файлы предназначены для записи и хранения информации.

Файлы располагаются на жестких дисках или на внешних носителях, хотя файл может располагаться на так называемом электронном диске (в оперативной памяти). Файл не имеет фиксированной длины, т.е. может увеличиваться и уменьшаться.

Файлы бывают **двоичные** (или **бинарные**), в которых могут быть записаны любые байты в таком же виде, в котором они представлены в памяти компьютера, а также **текстовые**, в которые можно записывать только символы (буквы, цифры, скобки и т.п.)

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 символом перевода строки (**'\n'**).

Перед работой с файлом в программе его необходимо открыть, а после работы – закрыть. При открытии файла с ним связывается *поток ввода-вывода*. Выводимая информация последовательно записывается в поток, вводимая информация считывается из потока.

2. СТАНДАРТНЫЕ ФУНКЦИИ ДЛЯ РАБОТЫ С ФАЙЛАМИ

Чтобы организовать взаимодействие программы с файлом, необходимо:

- объявить переменную, которая называется *указателем файла*;
- *открыть* файл, т.е. связать указатель файла с конкретным именем файла;
- *читать* информацию из файла;
- *писать* информацию в файл;
- *закрыть* файл, когда программа завершила все операции с ним.

Для работы с файлом используется специальная переменная, которая называется *указателем на файл* (или потоком). Это указатель на структуру **FILE**, которая определена в заголовочном файле **<stdio.h>**. В этой структуре хранится вся информация, необходимая программе для работы с файлом (адрес буфера, положение текущего байта в буфере, открыт файл на чтение или на запись, были ли ошибки при работе с файлом и др.).

Объявляется указатель на файл так:

```
FILE *fp;
```

где **fp** – это имя указателя на файл

Если необходимо работать одновременно с несколькими файлами, для каждого необходимо объявить свой указатель:

```
FILE *fp1, *fp2, *fp3;
```

2.1. ОТРЫТИЕ / ЗАКРЫТИЕ ФАЙЛА

fopen – открыть файл

Понятие «открыть файл» означает сделать его активным и заблокировать обращение других программ к этому файлу. При закрытии файла он освобождается (теперь с ним могут работать другие программы) и все ваши изменения вносятся на диск.

Чтобы открыть файл, надо вызвать функцию **fopen**, которая попытается открыть файл и записать его адрес в переменную – указатель на файл. После этого все обращения к файлу выполняются не по имени файла, а через этот указатель.

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

Описание

Открывает файл, заданный именем **name**.

Mode – указатель на строку символов, определяющих способ доступа к файлу:

"**r**" — открыть файл для чтения (файл должен существовать);

"**w**" — открыть пустой файл для записи; если файл существует, то его содержимое теряется;

"**a**" — открыть файл для добавления информации в конец файла, если файл не существует, он создается;

"**r+**" — открыть файл для чтения и записи (файл должен существовать);

"**w+**" — открыть пустой файл для чтения и записи; если файл существует, то его содержимое теряется;

"**a+**" — открыть файл для чтения и дополнения, если файл не существует, то он создаётся.

Дополнительно в строку **mode** может быть добавлен символ **`b'** для открытия файла в двоичном режиме или **'t'** – в текстовом (по умолчанию задается текстовый режим).

Возвращаемое значение

Возвращает указатель на **FILE** для открытого файла. Если при открытии файла произошла ошибка, то функция возвращает **NULL**.

Если файл открывается на *чтение*, это возможно в следующих случаях:

- неверно задано имя файла или файла нет на диске;
- файл используется другой программой и заблокирован.

Если файл открывается на *запись*, операция может закончиться неудачно, если

- на диске нет места;
- файл защищен от записи;
- неверно задано имя файла (например, оно содержит две точки, знак вопроса и т.п.).

fclose – закрыть файл

```
int fclose(FILE *fp);
```

Описание

Закрывает файл, ассоциированный с **fp**. При этом информация, которая находилась в буфере вывода, связанном с **fp**, перед закрытием записывается на диск.

Возвращаемое значение

Возвращает **0**, если файл успешно закрыт, и **-1** в противном случае.

Пример

```
#include <stdio.h>
int main()
{
  FILE *fp = fopen("Example.dat", "r");
  if (fp == NULL)
    printf("Файл не открыт\n");
  else
  {
    fclose(fp);
    printf("Файл закрыт\n");
  }
  return 0;
}
```

2.2. ЧТЕНИЕ ИНФОРМАЦИИ ИЗ ТЕКСТОВОГО ФАЙЛА

fscanf - форматный ввод из файла

Если файл открыт как текстовый, можно читать из него данные с помощью функции **fscanf**. Она полностью аналогична **scanf**, но служит для *ввода из файла*, а не с клавиатуры. Кроме того, ее первый параметр – *указатель на файл*, из которого читаются данные.

```
int fscanf(FILE *fp, char *format,...)
```

Описание

Читает символы из входного потока **fp**. Считанные символы преобразуются в соответствии со *строкой формата* (**format**). Преобразованные данные запоминаются по указанным аргументам-указателям, идущим после строки формата. Эти аргументы, должны быть указателями, по которым **fscanf** запишет введенные значения.

Возвращаемое значение

Функция **fscanf** возвращает результат – количество чисел, которые ей удалось прочитать. Если мы запрашивали одно число, то значение переменой n может быть равно единице (если все нормально) или нулю (если данные закончились или ошибочные, например, вместо чисел введено слово). Для успешного чтения данные в файле должны отделяться пробелом или символом перехода на новую строчку (он вставляется в файл при нажатии на клавишу **Enter**).

Функция читает только одно слово и останавливается на первом пробеле. Поэтому функция **fscanf** применяется тогда, когда надо читать файл по словам.

Пример

```
int main()
{
  char str[256];
  FILE *fp;

  fp = fopen("Example.dat", "r"); // Открываем для чтения
  if (fp == NULL) // Если не открылся,
    return 1;       // выходим
  fscanf(fp, "%s", str);  // Прочитает до первого пробела
  printf("%s", str);
  fclose(fp);             // закрываем файл

  getch();
  return 0;
}
```

fgets – прочитать строку из файла

Функция **fscanf** читает данные успешно в том случае, если известно, в каком порядке и как лежат данные в файле. Если же нужно прочитать строку целиком, лучше испоьзовать функцию **fgets**.

```
char *fgets(char *str, int n, FILE *fp);
```

Описание

Считывает байты из потока **fp** в строку по указателю **str**. Чтение заканчивается, когда прочитано **n** байт, встречается символ **‘\n’** (новая строка), конец файла или возникает ошибка чтения. Строка завершается **0**. Символ **'\n'** (новая строка) включается в строку (перед **0**).

Возвращаемое значение

Возвращает строку **str** в случае успеха. Если не было прочитано ни одного символа и произошла ошибка чтения или встречен **EOF**, возвращает **NULL**, а строка

str остается без изменения. Если возникла ошибка при чтении, возвращает **NULL**, а в строке возможен мусор.

Пример

Прочитать из файла "Example.dat" одну строку размером не более 256 байт.

```
int main()
{
  char str[256];
  FILE *fp;
  int len = 256;

  fp = fopen("Example.dat", "r"); // Открываем для чтения
  if (fp == NULL) // Если не открылся,
    return 1;       // выходим
  fgets(str, len, fp);   // читаем из файла в строку str
  printf("%s", str);

  fclose(fp);             // закрываем файл
  return 0;
}
```

Прочитать все строки из файла. Для этого используем цикл **while**, в котором читаем очередную строку и выводим ее на дисплей. Цикл заканчиваем, когда функция **fgets** вернет **NULL**.

```
int main()
{
  char str[256];
  FILE *fp;
  int len = 256;

  fp = fopen("Example.dat", "r"); // Открываем для чтения
  if (fp == NULL) // Если не открылся,
    return 1;       // выходим
  while (fgets(str, len, fp) != NULL)  // читаем из файла
    printf("%s", str);
  fclose(fp);             // закрываем файл
  return 0;
}
```

fgetc – прочитать байт из файла

Для чтения одного байта можно использовать функцию **fgetc**.

```
int fgetc(FILE *fp);
```

Описание

Читает очередной байт из потока **fp**.

Возвращаемое значение

Возвращает байт, прочитанный из файла **fp**. Если байт не может быть прочитан (если встретится конец файла или ошибка чтения), то возвращается значение **EOF**.

Пример

Прочитать побайтно **10** байт из файла. Так как заранее знаем, сколько байт прочитать, используем цикл **for**.

```
int main()
{
  char str[256];
  FILE *fp;
```

```
  int len = 256;

  fp = fopen("Example.dat", "r"); // Открываем для чтения
  if (fp == NULL) // Если не открылся,
    return 1;       // выходим
  int i;
  for(i = 0; i < 10; i++)
    str[i] = fgetc(fp);   // читаем из файла
  str[i] = 0;
  printf("%s", str);
  fclose(fp);             // закрываем файл
  getch();
  return 0;
}
```

Прочитать побайтно все байты из файла. Так как заранее неизвестно, сколько байт прочитать, используем цикл **while** и читаем, пока не возвратится конец файла (**EOF**) . Размер файла должен быть не более **256** символов.

```
int main()
{
  char str[256];
  FILE *fp;
  fp = fopen("Example.dat", "r"); // Открываем для чтения
  if (fp == NULL) // Если не открылся,
    return 1;       // выходим
  int i = 0;
  while((str[i++] = fgetc(fp)) != EOF); // Пока не EOF
  str[i] = 0;
  printf("%s", str);
  fclose(fp);             // закрываем файл

  getch();
  return 0;
}
```

2.3. ЗАПИСЬ ИНФОРМАЦИИ В ТЕКСТОВЫЙ ФАЙЛ

fprintf - форматный вывод в файл

Если файл открыт как текстовый, информацию можно записать в него помощью функции **fprintf**. Она полностью аналогична **printf**, но служит для *вывода в файл*, а не на дисплей. Кроме того, ее первый параметр – *указатель на файл*, в который записываются данные.

```
int fprintf(FILE *fp, char *format,...);
```

Описание

Осуществляет форматированный вывод в поток **fp**. Описание **format** совпадает с аналогичным описанием для **printf**.

Возвращаемое значение

Возвращает количество записанных символов, либо отрицательное значение в случае ошибки.

Пример

```
int main()
{
  int x = 123;
```

```
    int y = 456;
    FILE *fp;
    fp = fopen("Example.dat", "w"); // Открываем для записи
    if (fp == NULL) // Если не открылся,
      return 1;       // выходим
    fprintf(fp, "x = %d, y = %d\n", x, y);   // читаем из файла
    fclose(fp);             // закрываем файл

    return 0;
  }
```

В результате создается текстовый файл **Example.dat**, в котором будет записано: «**x = 123, y = 456**».

fputs – запись строки в файл

Для записи строки целиком в файл используют функцию **fputs**.

```
int fputs(char *str, FILE *fp);
```

Описание

Записывает строку **str**, исключая завершающий байт **0**, в поток **fp**.

Возвращаемое значение

Возвращает **0** в случае успеха и не **0** при ошибке записи.

Пример

Переписать строку из одного файла в другой. Для этого открываем первый файл для чтения, читаем из него строку в массив, затем закрываем файл и открываем новый, но уже для записи, и пишем строку в него. Для перезаписи нескольких строк лучше открыть оба файла сразу, объявив для этого два указателя на файл.

```
int main()
{
  char str[256];
  FILE *fp;
  int len = 256;

  fp = fopen("file_in.dat", "r"); // Открываем для чтения
  if (fp == NULL) // Если не открылся,
    return 1;       // выходим
  fgets(str, len, fp);       // читаем из файла
  fclose(fp);             // закрываем файл

  fp = fopen("file_out.dat", "w"); // Откр. другой файл для записи
  if (fp == NULL)  // Если не открылся,
      return;      // выходим
  fputs(str, fp);       // пишем в файл
  fclose(fp);          // закрываем Файл
  return 0;
}
```

fputc – запись байта в файл

Для записи одного байта в файл можно использовать функцию **fputc**.

```
int fputc(int c, FILE *fp);
```

Описание

Записывает байт **c** в поток **fp**.

Возвращаемое значение

Функция возвращает байт, записанный в поток. В случае ошибки возвращает **EOF**.

Пример

```
int main()
{
  char str[256] = "Example of writing\n";
  FILE *fp;

  fp = fopen("Example.dat", "a"); // Открываем для дозаписи
  if (fp == NULL) // Если не открылся,
    return 1;       // выходим
  for(int i = 0; str[i]; i++) // по всем символам строки
    fputc(str[i], fp);

  fclose(fp);             // закрываем файл
  return 0;
}
```

2.4. ФУНКЦИИ ЧТЕНИЯ И ЗАПИСИ БЛОКОВ ДАННЫХ ЛЮБОГО ТИПА

Рассмотренная выше функция `fgets` позволяет читать текст построчно. Если нужно прочитать заданное число байт, или весь файл целиком или информацию из бинарного файла, используют функцию `fread`.

fread – читать из файла

```
int fread(void *p, unsigned sizelem, unsigned n, FILE *fp);
```

Описание

Читает `n` элементов из потока `fp` в массив по указателю `p`. Параметр `sizelem` – размер одного элемента массива в байтах. Функция используется для чтения бинарных файлов. `void *p` означает, что `p` может быть указателем на любой тип.

Возвращаемое значение

Возвращает количество полностью прочитанных элементов. В случае ошибки или встречи конца файла возвращаемое значение будет меньше `n`.

Пример

Прочитать все символы из файла. Читаем символы, следовательно, размер одного элемента – 1 байт. В переменной **numread** будет число реально прочитанных элементов. Файл нужно открывать как бинарный.

```
int main()
{
  char str[257]; // Рассчитываем на 256 символов + конец строки
  int size = 1;  // Размер одного элемента
  int number = 256, numread;
  FILE *fp;

  fp = fopen("Example.dat", "rb");
  if (fp == NULL)
    return 1;
  numread = fread(str, size, number, fp); // Читаем 256 элементов
                                          // размером 1 байт каждый
  str[numread] = 0;  // Ставим конец строки
  printf("size = %d\n", numread); // прочиталось numread элементов
  printf("%s", str);
  fclose(fp);
  getch();
  return 0;
}
```

Пример
Прочитать из файла целые числа. Читаем элементы типа **int**, следовательно, размер одного элемента – 4 байта (**sizeof(int)**). Читаем в массив типа **int**.

```
int main()
{
  int Arr[100];
  int size = 4; // Лучше написать sizeof(int)
  int number = 100, numread;
  FILE *fp;

  fp = fopen("Datas.dat", "rb");
  if (fp == NULL)
    return 1;
  numread = fread(Arr, size, number, fp); // Читаем 100 элементов
                                          // размером 4 байта каждый
                                // прочиталось numread элементов
  printf("size = %d\n", numread);
  for(int i=0; i<numread; i++)
      printf("%d\n", Arr[i]);

  fclose(fp);
  getch();
  return 0;
}
```

fwrite – писать в файл

```
int fwrite(void *b, unsigned sizelem, unsigned n, FILE *fp);
```

Описание
Записывает `n` элементов размером `sizelem` байтов каждый из `b` в поток `fp`.
Возвращаемое значение
Возвращает количество полностью записанных элементов, которое может быть меньше **n**, если имела место ошибка.
Пример
Записать в бинарный файл квадраты первых 100 чисел.

```
int main()
{
  int buffer[101];
  unsigned n = 101;
  FILE *fp;
  int wrtn, i;

  fp = fopen("Datas.dat", "wb"); // Открываем для записи
                                 // в бинарный файл
  if (fp == NULL)
    return 1;
  for(i = 0; i < n; i++) // Заполняем массив квадратами чисел
    buffer[i] = i * i;
  wrtn = fwrite(buffer, sizeof(int), n, fp); // Пишем весь массив
  fclose(fp);

  return 0;
}
```

2.5. ФУНКЦИИ ПОЗИЦИОНИРОВАНИЯ В ФАЙЛЕ

Позицию, откуда будет начинаться считывание или запись в файле можно задать (`fseek`) или определить (`ftell`).

fseek – позиционировать файл

```
int fseek(FILE *fp, long offset, int origin);
```

Описание

Устанавливает позицию в файле, связанном с потоком `fp`. Смещение в байтах (`offset`) является величиной со знаком и берется относительно `origin`. Значения `origin` определяются константами:

```
SEEK_SET  -  начало файла
SEEK_CUR  -  текущая позиция
SEEK_END  -  конец файла
```

Возвращаемое значение

Возвращает **0** в случае успеха и ненулевое значение в случае ошибки.

Пример:

```
int main()
{
  FILE *fp = fopen("file.dat", "r+b");
  if (fp == NULL)
    return 1;

  fseek(fp, 0L, SEEK_END);   // Перейти в конец файла
  fseek(fp, 0L, SEEK_SET);   // Перейти в начало файла
  fseek(fp, 20L, SEEK_SET);  // Перейти на смещение 20
  fclose(fp);

  return 0;
}
```

ftell – определить позицию в файле

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

Описание

Возвращает текущую позицию в файле, связанном с потоком `fp`. Если файл открыт в текстовом режиме, возвращаемое значение *может не отражать точного количества действительно считанных или записанных байтов*.

Возвращаемое значение

Возвращает текущую позицию в файле или **-1**, если встретилась ошибка, с установлением `errno`.

Пример

Прочитать весь текст из файла. Память для текста выделить динамически. Для этого нужно знать размер текста, поэтому используем функции `fseek` и `ftell` для определения длины файла (т.е. размера теста).

```
int main()
{
  long len, n;
  char* str;
  FILE *fp;

  fp = fopen("Example.dat", "r");
  if (fp == NULL)
    return 1;
  fseek(fp, 0L, SEEK_END);    // Переходим в конец файла
```

```
  len = ftell(fp);              // Читаем позицию в файле
  printf("Len = %d\n", len);    // Это и будет длина файла
  fseek(fp, 0L, SEEK_SET);      // Переходим снова в начало файла

  str = new char[len + 1];  // Выделяем память под массив символов
  n = fread(str, sizeof(char), len, fp); // Читаем весь текст
                                // целиком из файла в массив str
                // Теперь в массиве str будут все байты, считанные
                // из файла, включая все строки, символы перевода
                // строк, конца строк и т.д.
  str[n] = 0; // Ставим конец строки
  printf("%s\n", str);
  fclose(fp);     // Закрываем файл

  getch();
  return 0;
}
```

3. ПРИМЕРЫ

Пример 1. В каждой строке файла **input.txt** заменить все пробелы на символ подчеркивания и сохранить измененный текст в файл **output.txt**.

Будем обрабатывать строки последовательно одну за другой. Для чтения из файла используем цикл **while**. Он завершится, когда функция **fgets** вернет значение **NULL**, то есть все строки будут обработаны.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <locale.h>

int main()
{
  char str[80];
  FILE* fin, * fout;

  setlocale(0, "Russian");
  fin = fopen("input.txt", "r");
  if (fin == NULL)
  {
    printf("Файл input.txt не открыт\n");
    return 1;
  }
  fout = fopen("output.txt", "w");
  if (fout == NULL)
  {
    printf("Файл output.txt не открыт\n");
    return 1;
  }

  while (fgets(str, 80, fin) != NULL) // читаем строку из файла
  {
    for(int i = 0; str[i]; i++) // См. все символы до конца строки
      if (str[i] == ' ')  // Если нашли,
        str[i] = '_';   // меняем символ
```

```
    fputs(str, fout); // выводим строку в другой файл
  }
  fclose(fin);
  fclose(fout);

  return 0;
}
```

Пример 2. Прочитать в массив строк все строки из файла. Вывести длину каждой строки и саму строку на дисплей.

В простейшей программе можно объявить статический массив строк и читать из файла в него все строки по очереди.

```
int main()
{
  char ArrStr[20][81];  // Массив указателей на будущие строки
  int count = 0;
  int len;
  FILE *fp;
  char *s;

  fp = fopen("Example.dat", "r"); // Открываем для чтения
  if (fp == NULL)
    return 1;
  do
  {
    s = fgets(ArrStr[count], 80, fp);  // Ввод строки из файла
    if (s != NULL) // Если строка прочиталась,
      count++;     // увеличиваем счетчик строк
  }while (!feof(fp));

  for (int i = 0; i < count; i++)
  {
    len = strlen(ArrStr[i]);  // Длина очередной строки
    printf("Lenth = %d\n", len); // Выводим ее
    printf("%s", ArrStr[i]);     // и саму строку
  }
  fclose(fp);
  getch();
  return 0;
}
```

Если строк в файле окажется больше, чем было объявлено в программе, будет выход за пределы массива, поэтому лучше заранее определить число строк. Например, так: сначала прочитаем строки в один временный массив и сосчитаем их количество, а уже затем динамически выделим память под них нужного размера.

```
int main()
{
  char **ArrStr; // Массив указателей на будущие строки
  char stmp[81]; // Вспомогательная строка
  int count = 0;
  int len;
  FILE *fp;
  char *s;
```

```
    fp = fopen("Example.dat", "r");
    if (fp == NULL)
      return 1;

    do  // Первый цикл – для подсчета числа строк в текстовом файле
    {
      s = fgets(stmp, 80, fp); // Ввод строки из файла
      if (s != NULL)
        count++;        // считаем строки
    }while (!feof(fp));
             // Посчитали кол-во строк
    ArrStr = new char*[count]; // Выделяем память под указатели
    fseek(fp, 0L, SEEK_SET);   // Переходим снова в начало файла

    for(int i = 0; i < count; i++) // Второй цикл – для чтения строк
    {
      ArrStr[i] = new char[81]; // Выделяем память под i-тую строку
      s = fgets(ArrStr[i], 80, fp); // Вводим i-тую строку из файла
    }

    for (int i = 0; i < count; i++) // Третий цикл - отображение
    {
      len = strlen(ArrStr[i]);    // Определяем размер строки
      printf("Lenth = %d\n", len); // и выводим на дисплей
      printf("%s", ArrStr[i]);     // и саму строку тоже
    }
    fclose(fp);
    getch();
    return 0;
  }
```

В общем случае удобнее читать сразу весь тест из файла функцией **fread** в один массив символов, а уже затем делить его на строки, на предложения или на слова.

ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ

В начале программы ОБЯЗАТЕЛЬНО выводить:
ФИО, группа, номер лаб. работы, номер варианта.

Задания

Переделать задание из лабораторной работы № 14, добавив в него еще один вариант: **читать обрабатываемый текст из файла**. Текстовый файл подготовить в каком-либо текстовом редакторе. Добавить в программу **вывод** результатов **в другой тестовый файл**.

Для получения максимального балла нужно выполнить следующие требования:

- *Память под считываемый текст должна выделяться динамически (по размеру файла).*
- *Алгоритмы анализа должны быть реализованы с помощью функций.*
- *Программа должна содержать цикл для возможного повторного ввода и анализа текста.*
- *Программа должна позволять вводить имя файла с клавиатуры.*