ГЛАВА 10

ОРГАНИЗАЦИЯ РАБОТЫ С ФАЙЛАМИ

ОБЩИЕ СВЕДЕНИЯ

Язык С кроме стандартного ввода данных с клавиатуры и вывода результатов на экран предоставляет также возможность обмена при операциях ввода/вывода со внешними устройствами, в том числе, с файлами на диске.

В С не предусмотрены никакие предопределенные структуры файлов (такие как файлы последовательного или прямого доступа): все файлы рассматриваются как последовательности, *потоки байтов*.

Для файла определён *маркер* (указатель чтения/записи). Он определяет текущую позицию, к которой осуществляется доступ.

С началом работы любой программы автоматически открываются некоторые стандартные потоки, например, стандартный ввод (его имя - ***stdin***) и стандартный вывод (его имя - ***stdout***). По умолчанию они связаны с клавиатурой и экраном терминала соответственно. Поэтому в тех функциях ввода/вывода, которые использовались до сих пор, не указывалось, из какого потока берутся или куда помещаются данные: это известно по умолчанию.

Однако возможно в своей программе открывать другие потоки, связывать их либо с файлами на диске, либо с физическими устройствами (например, принтером), записывать в них или считывать из них информацию. Для этого служат функции ввода/вывода *верхнего уровня*.

Доступ к потоку осуществляется с помощью *указателя*. Указатель на файл описывается следующим образом:

```
FILE * fp;
```

Тип FILE - это структура, определенная в <*stdio.h*> с помощью средства *typedef* и содержащая некоторую информацию о файле: например, флаги состояния файла, размер буфера, указатель на буфер и др. Описанный указатель можно связать с конкретным файлом в момент открытия данного файла. Это осуществляется с помощью функции

fopen ("*путь к файлу*", "*тип доступа*") ,

которая возвращает указатель на файл или NULL в случае ошибки.

Например, в результате выполнения оператора

```
fp = fopen ("ex1.txt", "w");
```

файл ex1.txt открыт для записи, а в программе на этот файл можно сослаться с помощью указателя *fp* (т.е. функция *fopen()* берет внешнее представление файла - его физическое имя - и ставит ему в соответствие внутреннее логическое имя, которое далее и будет использоваться в программе).

В качестве типа доступа могут быть указаны следующие параметры:

"*w*" - существующий файл открывается для записи, при этом его старое содержимое затирается. Маркер файла указывает на его начало. Если файл еще не существовал, то он создается (если это возможно);

"*r*" - существующий файл открывается для чтения (с начала). Если файл не существует - это ошибка;

"*a*" - файл открывается для дозаписи в его конец.
О других типах доступа можно прочитать в книгах по языку С.

По окончании работы с файлом он должен быть закрыт. Для этого используется функция

fclose (указатель_файла).

Для чтения/записи данных в файл имеются функции, аналогичные уже известным функциям ввода/вывода:

fprintf(), fscanf(), fputs(), fgets(), getc(), putc(), fgetc(), fputc().

Функции *getc()/fgetc(), putc()/fputc()* по своим действиям идентичны, отличие состоит только в том, что *getc()* и *putc()* реализованы как макроопределения, а *fgetc()* и *fputc()* - как настоящие функции.

Прототипы всех файловых функций, а также необходимые константы находятся в файле <*stdio.h*> .

Чтобы продемонстрировать работу с этими функциями, рассмотрим простой пример.

Задача А

Демонстрация основных функций работы с файлами.

```
#include <stdio.h>

void main()
{ int n;
  char str[50], str1[50], ch;
  FILE *fp;

// заполняем файл
  fp = fopen ("ex.txt","w");
  puts ("Введите целое число"); scanf("%d",&n);
  fprintf (fp, "%d\n", n);
  puts ("Введите символ"); ch=getchar();
  putc (ch, fp);
  puts ("Введите строку"); gets (str);
  fputs (str, fp);
  fclose (fp);

// читаем из файла
  if((fp = fopen("ex.txt","r")) != NULL)
```

```
  {
   fscanf (fp, "%d", &n); printf ("n=%d\n", n);
   ch = getc (fp); putchar (ch);
   fgets (str1, 50, fp); puts (str1);
   fclose (fp);
  }
  else printf ("\n Нельзя открыть файл для чтения!");
}
```

Второй параметр у функции *fgets()* - количество N считываемых символов, включая '\0'. Эта функция прекращает работу после чтения N-1 символа, либо после чтения '\0'. В любом случае в конец строки добавляется '\0'. Функция *fgets()* - возвращает адрес прочитанной строки либо NULL - по исчерпании файла или в случае ошибки.

В случае исчерпания файла или ошибки функция *getc()* возвращает EOF

Функция *putc()* возвращает записанную литеру либо (в случае ошибки) - EOF.

Функция *fputs()* возвращает код последнего прочитанного символа, если все в порядке, и EOF в случае ошибки. Эта функция *не переводит* строку автоматически.

Рассмотрим ещё один пример.

Задача B

В командной строке передается имя считываемого файла. Каждый третий его символ переписывается в выходной файл, имеющий то же самое имя и расширение ***.red*** .

```
#include <stdio.h>
void main(int argc, char *argv[])
{
 int ch, count=1;
 FILE *in, *out;
 static char name[20]; // под имя выходного файла
 if(argc<2)
  puts ("Нужно имя файла в командной строке!");
  else
    {
     if((in = fopen (argv[1], "r")) != NULL)
      {
        strcpy (name, argv[1]);
        strcat (name, ".red");
        out = fopen (name, "w");
        while ((ch = getc (in)) != EOF)
       if (count++ %3==0)
```

```
          putc (ch, out);
          fclose (in);
          fclose (out);
      }
      else printf ("\n Нельзя открыть файл %s", argv[1]);
   }
}
```

Все приведенные выше функции обрабатывали файл последовательно - символ за символом. Язык С предоставляет возможность работать с файлом и как с массивом: непосредственно достигать любого определенного байта. Для позиционирования файла служит функция

fseek (указатель на файл, смещение от нач. точки, нач. точка)

Второй аргумент имеет тип *long*, его значение может быть больше и меньше нуля. Он показывает, как далеко (в байтах) следует продвинуться от начальной точки. Третий аргумент является кодом, определяющим положение начальной точки в файле. Установлены следующие значения для кода:

0	-	начало файла;
1	-	текущая позиция;
2	-	конец файла.

В случае успеха функция *fseek()* возвращает 0, а если была ошибка (например, попытка выхода за левую границу файла), то возвращается -1.

Рассмотрим ещё один пример.

Задача С

Осуществить побайтную чередующуюся печать файла в прямом и обратном направлении (если в файле было слово ЛУНА, то будет напечатано ЛАУННУАЛ).

```
#include <stdio.h>
void main(int argc, char *argv[])
{
 long offset = 0L;
 FILE *fp;
 if (argc<2)
  puts ("Нужно имя файла в командной строке!");
  else
    {
      if ((fp = fopen(argv[1], "r")) == NULL)
       printf ("Нельзя открыть файл %s", argv[1]);
       else
         {
```

```
        while fseek(fp, offset++,          0)==0)
            {
             putchar (getc (fp));
             if(fseek(fp, - (offset+1), 2)==0)  //  см. замечание после текста
                                                //  программы!
              putchar (getc (fp));
            }
    fclose(fp);
  }
}
```

Замечание. В выражении (offset+1) добавляемая величина зависит от числа символов, записываемых системой в конец файла. Проверьте это экспериментально на своей машине.

При работе с файлами структур удобно использовать функции *fread/fwrite*, которые имеют следующий формат:

fread (ptr, size, N, fp) - из файла *fp* считываются *N* элементов размером *size* байт каждый в область памяти с адресом *ptr*. В случае успеха возвращается число считанных байтов. При ошибке или конце файла - EOF.

Аналогично действует функция *fwrite()*:

fwrite (ptr, size, N, fp) - В файл *fp* записываются *N* элементов по *size* байт каждый, расположенные в области памяти, начиная с адреса *ptr*.

Вот простой пример использования этих функций:

```
typedef struct
  {
    char author [30];
    char title [50];
    int pages;
  } BOOK;
BOOK b1={"Kernighan", "C Language", 256}, b2;
FILE *fp;
void main()
{ ...
  fp=fopen("struct.txt", "w+"); // открыли файл на чтение и запись
                                // одновременно
  fwrite(&b1, sizeof(BOOK), 1, fp);
  fseek(fp, 0, 0); // вернули маркер  на начало файла
  fread(&b2, sizeof(BOOK), 1, fp);
  printf("Автор - %s, название - %s, число страниц - %d\n",
                                        b2.author, b2.title, b2.pages);
}
```

Потоки данных, с которыми работают функции верхнего уровня, по умолчанию *буферизованы*. Это означает, что при открытии потока с ним автоматически связывается определенная область оперативной памяти, называемая *буфером*, и все операции чтения/записи ведутся через буфер.

Например, при чтении данных из потока блок данных помещается во входной буфер, из которого эти данные считываются при каждом запросе. Когда буфер опорожнен, в него передается следующая порция данных. При операциях записи данные помещаются вначале в буфер, а после того как он заполнится, его содержимое "выгружается" - записывается в соответствующий выходной поток.

Буферизация хороша хотя бы тем, что она повышает эффективность ввода/вывода, т.к. ОС передает за одну операцию большие блоки данных, а не вызывает системные функции ввода/вывода для каждого одиночного элемента данных. (Пользователь считает, что он работает с диском, а на самом деле, незаметно для него, получается так, что он работает с быстродействующей оперативной памятью).

Размер буфера фиксирован (в < *stdio.h* > описана специальная константа BUFSIZ, величина которой обычно равна 512). Имеется возможность изменить размер буфера, а также связать поток не с системным, а своим буфером - массивом размера BUFSIZ. Для этого используются функции *setbuf()* и *setvbuf()*.

Функции верхнего уровня одинаково реализуются в различных ОС и на разных машинах, поэтому их использование позволяет писать переносимые программы.

Рассмотрим теперь другой класс функций файлового ввода/вывода - функции *нижнего уровня* (или системные вызовы), которые в своей работе непосредственно используют средства ввода/вывода ОС.

Эти функции, в отличие от потоковых функций, выполняют *небуферизованный* ввод/вывод, т.е. для чтения/записи отдельного байта специально осуществляется вызов системной функции. Поэтому обработка большого количества байтов с помощью функций нижнего уровня может оказаться гораздо более длительной, чем если использовать потоковые функции.

Преимущество системных вызовов перед потоковыми состоит в следующем. Последовательность кодов, выполняемая при вызове обычной функции, включается в исполняемый модуль программы. Поэтому добавление в программу новой функции может увеличить ее размер на тысячи байтов. Коды же, выполняемые при системном вызове, размещаются в адресном пространстве ядра ОС, следовательно, добавление еще одного системного вызова окажет незначительное влияние на размер программы. Все функции нижнего уровня работают не с указателем на файл, как потоковые функции, а с *дескриптором файла*. Дескриптор - это неотрицательное целое число,

которое используется для ссылок на открытый файл во внутренних таблицах системы. (Например, поток **stdin** имеет дескриптор = 0, поток **stdout** - 1).

Вот описания некоторых системных вызовов.

Открытие файла:

open ("путь к файлу", "способ доступа")

Возможные режимы доступа, которые могут комбинироваться с помощью операции | (ИЛИ), следующие:

O_RDONLY - открыть только для чтения;
O_WRONLY - открыть только для записи;
O_RDWR - открыть для чтения и записи;
O_APPEND - открыть для дозаписи в конец файла;
O_CREAT - создать файл, если он не существует.

В случае успеха функция возвращает дескриптор открытого файла; в случае ошибки - число -1.

Чтение файла:

*read (int fd, char *buf, int nbytes)*

Из файла с дескриптором *fd* в массив *buf* считываютя *nbytes* байтов, начиная с текущей позиции маркера файла. При успешном завершении возвращается число считанных байтов; при достижении конца файла в процессе чтения - число 0; в случае ошибки - значение -1.

Запись в файл:

*write (int fd, char *buf, int nbytes)*

В файл с дескриптором *fd* из массива *buf* переписываются *nbytes* байтов, начиная с текущей позиции маркера файла. Значение указателя увеличивается на число записанных байтов. Возвращается число записанных байтов или -1.

Позиционирование файла:

lseek (int fd, long offset, int n)

Маркер файла с дескриптором *fd* смещается на величину *offset* относительно места, заданного параметром *n*.

Возможные значения *n*:

0 - начало файла;
1 - текущая позиция;
2 - конец файла.

Функция возвращает величину текущей позиции или -1.

Например, длину файла в байтах можно определить следующим образом:

```
file_size = lseek (fd, 0, 2);
```

Задача D

Написать функцию, которая читает любое число байтов из любого места файла.

```
int get(int fd, long pos, char *buf, int n)
{
  if(lseek(fd, pos, 0)>=0)
   return read (fd, buf, n);
   else return -1;
}
```

Задача E

Пример реализации функции верхнего уровня *getchar()* с помощью системных вызовов.

Небуферизованная версия:

```
int getchar()
{ char c;
  return ( read (0, &c, 1)== 1) ? (unsigned char)c : EOF;
}
```

Версия с буферизацией (считывается блок данных, но при каждом обращении к функции выдается только одна литера):

```
int getchar()
{
  static char buf[BUFSIZ];
  static char *bufp=buf;
  static int n=0;
  if(n==0)  // буфер пуст
   {
     n = read(0, buf, sizeof buf);
     bufp = buf;
   }
  return (--n>=0) ? (unsigned char) *bufp++ : EOF;
}
```

Функции потока и функции нижнего уровня в общем случае несовместимы (из-за того, что одни выполняют буферизацию, а другие - нет). Поэтому при работе с файлом надо использовать либо те, либо другие функции, т.к. иначе, при одновременном доступе к файлу разными способами, может произойти потеря данных в буфере.

ЗАДАНИЕ НА ПРОГРАММИРОВАНИЕ

Каждый студент должен решить две задачи, номера которых определяются его порядковым номером.

Номер первой задачи:

НПЗ=2*(N-1)%35+1, где N - номер студента.

При решении задач имена входного и выходного файлов рекомендуется задавать как аргументы командной строки. Необходимо выдавать диагностику при ошибочных ситуациях в работе с файлами.

Задача 1

Экспертами компании по перевозкам грузов на верблюдах установлено, что верблюд безопасно для жизни может перевезти 7640 прутьев. Данные о каждом караване верблюдов находятся в файле. Для каждого каравана набор данных представляет собой группу строк: имя погонщика, число верблюдов, число корзин, перевозимых каждым верблюдом и число прутьев в каждой корзине. Распечатать состояние верблюдов. Например,

ТОМ ДЖОНС
  верблюды в порядке
БОБ УЭЙТ
  недопустимые грузы:
    верблюд 3 перевозит 7645 прутьев
    верблюд 8 перевозит 8006 прутьев

Задача 2↑

Сравнить время, затрачиваемое на подсчёт числа пробелов в текстовом файле объёмом свыше 1000 символов функциями верхнего и нижнего уровня.

Задача 3

Найти максимальную длину строки в текстовом файле и распечатать все строки файла, имеющие такую длину.

Задача 4

В файле находятся только целые числа. Определить, имеет ли последовательность чисел, находящихся в файле, нечетную длину, и если да, то переменной *middle* присвоить значение среднего элемента файла. В противном случае присвоить этой переменной значение первого числа файла.

Задача 5↑

Создать файл, содержащий сведения о книгах в библиотеке. Структура записи: шифр книги, автор, название, год издания, местоположение (номер стеллажа, полка).

Предусмотреть возможность корректировки файла по вводимому коду корректировки, например:

1 - удалить запись (по шифру XXX);

2 - добавить новую запись;

3 - изменить запись (по введённой фамилии автора и названию книги);

4 - получить информацию о книге с шифром XXX.

Задача 6

В текстовый файл вставить пробелы таким образом, чтобы каждая строка имела длину 80 символов (пробелы в строке должны быть вставлены равномерно).

Задача 7

Каждая строка файла содержит название горной вершины и ее высоту. Используя структуру для описания понятия ***вершина***, получить название самой высокой вершины по данным из файла.

Задача 8

В текстовом файле подсчитать количество строк, которые оканчиваются буквой ’s’.

Задача 9↑

Из текстового файла выбросить все гласные. Новый файл не создавать.

Задача 10

Каждая строка файла содержит следующие данные: пол, имя, рост. Распечатать средний женский рост и имя самого высокого мужчины по данным из файла. Использовать структуру для описания понятия ***человек***.

Задача 11

В файле находится текст программы на языке С. Создать выходной файл, в который переписать содержимое исходного файла, убрав комментарии из текста программы.

Задача 12

Написать программу, которая работает в одном из двух режимов. Если в текущем каталоге имеется файл "***tabl_umn.txt***", то распечатать построчно его содержимое. В противном случае создать файл с таким именем и записать туда таблицу умножения для чисел от 2 до 9.

Задача 13

Подсчитать количество пустых строк в текстовом файле.

Задача 14

Используя структуру для определения понятия ***студент*** (состоящую из полей ФИО, курс, группа, оценки в сессию) распечатать фамилии и имена отличников первого курса и долю их от общего числа отличников. (Данные находятся в файле).

Задача 15

Используя структуру с полями пол, ФИО, возраст, распечатать количество девушек по имени "Elena" и имена тех, кому 19 лет. (Данные находятся в файле).

Задача 16↑

Дан произвольный текст объёмом не менее 1000 символов. Отредактировать его таким образом, чтобы все строки, кроме последней, имели фиксированную длину *n*.

Правила редактирования:

- слова не переносятся;
- знак препинания не отделяется от слова, за которым он стоит;
- строки выравниваются за счёт равномерно вставляемых пробелов.

Задача 17

В текстовом файле подсчитать количество строк, которые начинаются с буквы ’**f**’.

Задача 18↑

Из текстового файла выбрость все пробельные символы. Новый файл не создавать.

Задача 19

В файле находятся вещественные числа. Определить количество чисел в наиболее длинной возрастающей последовательности элементов файла.

Задача 20

Написать программу записи в файл и чтения из файла элементов массива структур для регистрации автомашин с полями:

- марка машины;
- год выпуска;
- цвет;
- номер.

Задача 21

В текстовом файле подсчитать количество строк, которые начинаются и оканчиваются одной и той же буквой.

Задача 22↑

В текстовом файле заменить все последовательности идущих подряд пробелов одним пробелом, т.е., "сжать" файл. Новый файл не создавать.

Задача 23

Проверить наличие баланса всех видов скобок в текстовом файле.

Задача 24

Дан текстовый файл F1. Переписать его содержимое в файл F2, разбив на строки таким образом, чтобы каждая строка либо оканчивалась точкой, либо содержала 40 литер, если среди них нет точки.

Задача 25

В командной строке задается имя входного файла и целое число N. Распечатать последние N строк указанного файла.

Задача 26

Создать 2 файла, содержащих сведения о игроках хокейных команд "Динамо" и "Спартак". Структура записей файлов:

- фамилия, имя игрока;
- число заброшенных шайб;
- число сделанных голевых передач.

По данным, извлекаемым из этих файлов, создать новый файл, содержащий данные о шести самых результативных игроках обеих команд.

Задача 27↑

Треугольник Паскаля - таблица чисел, являющихся биноминальными коэффициентами. В этой таблице по боковым сторонам равнобедренного треугольника стоят 1, а каждое из остальных чисел равно сумме двух чисел, стоящих над ним слева и справа (см. рис.1).

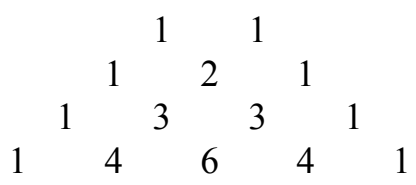

Рис. 1

В строке с номером *n+1* выписаны коэффициенты разложения бинома $(a+b)^n$.

Написать программу, которая работает в одном из двух режимов. Если в текущем каталоге имеется файл "tr_pasc.txt", то распечатать его содержимое в виде, представленном на рис.1. В противном случае создать файл с таким именем и записать туда треугольник Паскаля *n-го* порядка. Параметр *n* (*n*<12) задаётся в командной строке.

Задача 28

Написать программу сравнения двух файлов: должна печататься первая строка, в которой они различаются. Если файлы идентичны, то выдать сообщение.

Задача 29

Создать файл, содержащий сведения о том, какие из 5 предложенных дисциплин желает слушать студент. Структура записи:

- фамилия студента;
- № группы;
- средний балл;
- 5 дисциплин, где '*' показывает выбранную дисциплину.

Создать файл, содержащий данные о тех, кто желает прослушать дисциплину XX. Если желающих больше 10, то отобрать тех студентов, у которых более высокий средний балл.

Задача 30

Написать программу, действующую аналогично команде *cat* в операционной системе *Unix* (см. Б.Керниган, Д.Ритчи. Язык программирования Си. -М.: Финансы и статистика, 1992, стр. 157-158).

Задача 31↑

Дан текст, в котором начало каждого абзаца отмечено символом '@'. Отредактировать этот текст по следующим правилам:

а) первая строка имеет отступ *k* позиций;

б) все строки текста, кроме последних строк абзацев, должны иметь фиксированную длину *n*;

в) при редактировании слова не переносятся. Знак препинания не отделяется от слова, за которым он стоит. Строки выравниваются за счёт дополнительно вносимых пробелов.

Задача 32

Создать файл, в который записать результаты соревнований по 6 видам спорта летней Олимпиады 1992 года.

Написать программу, выполняющую следующие функции:

- выдать список призёров страны NNN;

- выдать таблицу призёров (золото, серебро, бронза) по запрашиваемому виду спорта.

Задача 33↑

Создать файл, содержаший сведения о товарах, хранящихся на складе: шифр товара, наименование товара, количество единиц, стоимость единицы. Все записи должны быть отсортированы в порядке возрастания шифра товара.

Иметь возможность по введённому коду корректировки:

а) изменить/добавить запись о товаре с шифром XXX;

б) удалить запись о товаре с шифром XXX;

в) получить информацию о всех товарах, в наименовании которых содержится заданный ключ.

Задача 34

В файле находятся вещественные числа. Определить количество элементов файла, величина которых меньше среднего арифметического всех элементов данного файла.

Задача 35↑

Создать файл, содержаший сведения об ассортименте обуви в магазине. Структура записи: артикул, наименование, размер, количество пар, стоимость одной пары. Артикул начинается с буквы D для дамской, M - для мужской, C - для детской обуви.

Написать программу, выдающую следующую информацию:

- наличие и стоимость обуви артикула XX;

- список дамской обуви заданного размера с указанием наименования и имеющегося в наличии числа пар каждой модели.