Урок 12. Потоковый ввод-вывод в стандарте С

Цель работы: Изучить функции ввода-вывода данных и управления выводом на экран

Под вводом-выводом в программировании понимается процесс обмена информацией между оперативной памятью и внешними устройствами: клавиатурой, дисплеем, магнитными накопителями и т. п. Ввод — это занесение информации с внешних устройств в оперативную память, а вывод — вынос информации из оперативной памяти на внешние устройства. Такие устройства, как дисплей и принтер, предназначены только для вывода; клавиатура — устройство ввода. Магнитные накопители (диски, ленты) используются как для ввода, так и для вывода.

Основным понятием, связанным с информацией на внешних устройствах ЭВМ, является понятие файла. Всякая операция ввода-вывода трактуется как операция обмена с файлами: ввод — это чтение из файла в оперативную память; вывод — запись информации из оперативной памяти в файл. Поэтому вопрос об организации в языке программирования ввода-вывода сводится к вопросу об организации работы с файлами.

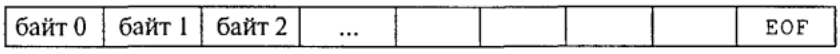
Элементы файловой переменной могут иметь разный тип и, соответственно, разные длину и форму внутреннего представления. Внутренний файл связывается с внешним (физическим) файлом с помощью стандартной процедуры Assign. Один элемент файловой переменной становится отдельной записью во внешнем файле и может быть прочитан или записан с помощью одной команды. Попытка записать в файл или прочитать из него величину, не совпадающую по типу с типом элементов файла, приводит к ошибке.

Аналогом понятия внутреннего файла в языках С/С++ является понятие потока. Отличие от файловой переменной Паскаля состоит в том, что потоку в С не ставится в соответствие тип.

Поток — это байтовая последовательность, передаваемая в процессе ввода-вывода.

Поток должен быть связан с каким-либо внешним устройством или файлом на диске. В терминологии С это звучит так: поток должен быть направлен на какое-то устройство или файл.

Основные отличия файлов в С состоят в следующем: здесь отсутствует понятие типа файла и, следовательно, фиксированной структуры записи файла. Любой файл рассматривается как байтовая последовательность:



Стрелочкой обозначен указатель файла, определяющий текущий байт файла. EOF является стандартной константой — признаком конца файла.

Существуют стандартные потоки и потоки, объявляемые в программе. Последние обычно связываются с файлами на диске, создаваемыми программистом. Стандартные потоки назначаются и открываются системой автоматически. С началом работы любой программы открываются 5 стандартных потоков, из которых основными являются следующие:

- stdin — поток стандартного ввода (обычно связан с клавиатурой);
- stdout — поток стандартного вывода (обычно связан с дисплеем);

• stderr — вывод сообщений об ошибках (связан с дисплеем).

Кроме этого, открывается поток для стандартной печати и дополнительный поток для последовательного порта.

Работая ранее с программами на С, используя функции ввода с клавиатуры и вывода на экран, мы уже неявно имели дело с первыми двумя потоками. А сообщения об ошибках, которые система выводила на экран, относились к третьему стандартному потоку. Поток для работы с дисковым файлом должен быть открыт в программе.

Работа с файлами на диске. Работа с дисковым файлом начинается с объявления указателя на поток. Формат такого объявления:

```
FILE *имя_указателя;
```

Например:

```
FILE  *fp;
```

Слово FILE является стандартным именем структурного типа, объявленного в заголовочном файле stdio.h . В структуре FILE содержится информация, с помощью которой ведется работа с потоком, в частности: указатель на буфер, указатель (индикатор) текущей позиции в потоке и т.д.

Следующий шаг — открытие потока, которое производится с помощью стандартной функции fopen( ). Эта функция возвращает конкретное значение для указателя на поток и поэтому ее значение присваивается объявленному ранее указателю. Соответствующий оператор имеет формат:

```
имя_указателя =: fopen (имя_файла, режим__открытия);
```

Параметры функции fopen( ) являются строками, которые могут быть как константами, так и указателями на символьные массивы. Например:

```
fp = fopen("test.dat", "r" );
```

Здесь test.dat — это имя физического файла в текущем каталоге диска, с которым теперь будет связан поток с указателем fp.

Параметр режима r означает, что файл открыт для чтения. Что касается терминологии, то допустимо употреблять как выражение «открытие потока», так и выражение «открытие файла».

Существуют следующие режимы открытия потока и соответствующие им параметры:

Параметр	Режим
r	открыть для чтения
w	создать для записи
a	открыть для добавления
r+	открыть для чтения и записи
w+	создать для чтения и записи
a+	открыть для добавления или создать для чтения и записи

Открытие уже существующего файла для записи ведет к потере прежней информации в нем. Если такой файл еще не существовал, то он создастся. Открывать для чтения можно только существующий файл.

Поток может быть открыт либо для текстового, либо для двоичного (бинарного) режима обмена.

Текстовый файл - это последовательность символов, которая делится на строки специальными кодами — возврат каретки (код 13) и перевод строки (код 10). Если файл открыт в текстовом режиме, то при чтении из такого файла комбинация символов «возврат каретки — перевод строки» преобразуется в один символ \n — переход к новой строке.

При записи в файл осуществляется обратное преобразование.

При работе с двоичным файлом никаких преобразований символов не происходит, т.е. информация переносится без всяких изменений.

Указанные выше параметры режимов открывают текстовые файлы. Если требуется указать на двоичный файл, то к параметру добавляется буква b. Например: rb, или wb, или r + b. В некоторых компиляторах текстовый режим обмена обозначается буквой t, т.е. записывается a + t или rt .

Если при открытии потока по какой-либо причине возникла ошибка, то функция fopen( ) возвращает значение константы NULL. Эта константа также определена в файле stdio.h. Ошибка может возникнуть из-за отсутствия открываемого файла на диске, нехватки места в динамической памяти и т.п. Поэтому желательно контролировать правильность прохождения процедуры открытия файла. Рекомендуется следующий способ открытия:

```
FILE  *fp;
if  (fp = fopen("test.dat", "r")== NULL
{
puts("Не могу открыть  файл\n");
return;
}
```

В случае ошибки программа завершит выполнение с закрытием всех ранее открытых файлов.

Закрытие потока (файла) осуществляет функция fclose( ), прототип которой имеет вид:

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

Здесь fptr обозначает формальное имя указателя на закрываемый поток. Функция возвращает ноль, если операция закрытия прошла успешно. Другая величина означает ошибку.

Запись и чтение символов. Запись символов в поток производится функцией putc( ) с прототипом

```
int  putc(int  ch,  FILE  *fptr);
```

Если операция прошла успешно, то возвращается записанный символ. В случае ошибки возвращается константа EOF.

Считывание символа из потока, открытого для чтения, производится функцией gets( ) с прототипом

```
int  gets(FILE  *fptr);
```

Функция возвращает значение считываемого из файла символа. Если достигнут конец файла, то возвращается значение EOF. Заметим, что это происходит лишь в результате чтения кода EOF.

Исторически сложилось так, что gets( ) возвращает значение типа int. То же можно сказать и про аргумент ch в описании функции puts( ). Используется же в обоих случаях только младший байт. Поэтому обмен при обращении может происходить и с переменными типа char.

Пример 1. Составим программу записи в файл символьной последовательности, вводимой с клавиатуры. Пусть признаком завершения ввода будет символ *.

```
//Запись символов в файл
#include <stdio.h>
#include <locale.h>
int  main( )
{
setlocale(LC_ALL, "RUS");
FILE *fp;
char c;
if((fp = fopen("test.dat", "w")) == NULL)
      {
      puts("Не могу открыть файл!\n");
      return;
      }
puts("Вводите символы. Признак конца -  *");
while((c = getchar( )) != '*')
putc(c, fp);
fclose(fp);
}
```

В результате на диске (в каталоге, определяемом системой) будет создан файл с именем test.dat, который заполнится вводимыми символами. Символ * в файл не запишется.

Пример 2. Файл, созданный в результате работы предыдущей программы, требуется последовательно прочитать и содержимое вывести на экран.

```
//Чтение символов из файла
#include  <stdio.h>
#include  <conio.h>
#include <locale.h>
int  main( )
{
setlocale(LC_ALL, "RUS");
FILE *fp;
char c;
void  clrscr( );
if((fp = fopen("test.dat", "r")) == NULL)
      {
      puts("Не удалось открыть файл!\n");
      return;
      }
while((c = getc(fp)) != EOF)  putchar(c);
fclose(fp);
}
```

Связь между результатом работы предыдущей программы и данной программой осуществляется через имя физического файла, которое в обоих случаях должно быть одним и тем же.

Запись и чтение целых чисел. Запись целых чисел в поток без преобразования их в символьную форму производится функцией putw( ) с прототипом

```
int  putw(int,  FILE  *fptr );
```

Если операция прошла успешно, то возвращается записанное число. В случае ошибки возвращается константа EOF.

Считывание целого числа из потока, открытого для чтения, производится функцией getw( ) с прототипом

```
int  getw(FILE  *fptr ) ;
```

Функция возвращает значение считываемого из файла числа.

Если прочитан конец файла, то возвращается значение EOF.

Пример 3. Составим программу, по которой в файл запишется последовательность целых чисел, вводимых с клавиатуры, а затем эта последовательность будет прочитана и выведена на экран. Пусть признаком конца ввода будет число 9999.

```
//Запись и чтение целых чисел
#include  <stdio.h>
#include  <conio.h>
#include  <locale.h>
int  main( )
{
setlocale(LC_ALL, "RUS");
FILE *fp;
int x;
void clrscr( );
//Открытие потока для записи
if(fp = fopen("test.dat", "w")) == NULL)
{
puts("Ошибка открытия файла для записи!\n") ,
return;
}
puts("Вводите числа. Признак конца - 9999");
scanf("%d", &x);
while(x != 9999)
{
putw(x, fp); scanf("%d", &x);
}
fclose(fp); //Закрытие  потока в режиме записи
//Открытие потока для чтения
if((fp = fopen("test.dat", "r")) == NULL)
{
puts("Ошибка  открытия файла для чтения!\n");
return;
}
while((x = getw(fp)) != EOF)  printf("\n%d", x);
fclose(fp);
}
```

После завершения ввода чисел в файл его необходимо закрыть.

При этом происходит сброс накопленных в буфере значений на диск. Только после этого можно открывать файл для чтения. Указатель устанавливается на начало потока, и дальнейшее чтение будет происходить от начала до конца файла.

Запись и чтение блоков данных. Специальные функции обмена с файлами имеются только для символьного и целого типов данных. В общем случае используются функции чтения и записи блоков данных. С их помощью можно записывать в файл и читать из файла вещественные числа, массивы, строки, структуры. При этом, как и для ранее рассмотренных функций, сохраняется форма внутреннего представления данных.

Функция записи блока данных имеет прототип

```
int fread(void *buf, int bytes, int n, FILE *fptr);
```

Здесь

buf — указатель на адрес данных, записываемых в файл;
bytes — длина в байтах одной единицы записи (блока данных);
n — число блоков, передаваемых в файл;
fptr — указатель на поток

Если запись выполнилась благополучно, то функция возвращает число записанных блоков (значение n).

Функция чтения блока данных из файла имеет прототип

```
int fwrite(void *buf,  int bytes, int n, FILE *fptr);
```

По аналогии с предыдущим описанием легко понять смысл параметров.

Пример 4. Следующая программа организует запись блоком в файл строки (символьного массива), а также чтение и вывод на экран записанной информации.

```
#include  <string.h>
#include  <stdio.h>
#include  <locale.h>
int main( )
{
setlocale(LC_ALL, "RUS");
FILE  *stream;
char msg[ ] = "this  is a test";
char  buf[20];
if((stream = fopen("DUMMY.FIL", "w+")) == NULL)
{
puts("Ошибка  открытия  файла\n");
return;
}
//Запись строки в файл
fwrite(msg,  strlen(msg)+1, 1, stream);
//Установка указателя на начало файла
fseek(stream, O, SEEK_SET);
//Чтение строки из файла
fread(buf,  strlen(msg)+ 1, 1,  stream);
printf("%s\n", buf);
fclose(stream);
}
```

В этой программе поток открывается в режиме w+ (создание для записи с последующим чтением). Поэтому закрывать файл после записи не потребовалось. Новым эле-

ментом данной программы по сравнению с предыдущими является использование функции установки указателя потока в заданную позицию. Ее формат:

```
int fseek(указатель_на поток,  смещение,
начало__отсчета)
```

Начало отсчета задается одной из констант, определенных в файле stdio.h:

```
SEEK_SET  (имеет значение 0) –  начало файла;
SEEK_CUR  (имеет значение 1) –  текущая  позиция;
SEEK_END  (имеет значение 2) –  конец файла.
```

Смещение определяет число байт, на которое надо сместить указатель относительно заданного начала отсчета. Смещение может быть как положительным, так и отрицательным числом. Оба параметра имеют тип long.

Форматный обмен с файлами. С помощью функции форматного вывода можно формировать на диске текстовый файл с результатами вычислений, представленными в символьном виде. В дальнейшем этот файл может быть просмотрен на экране, распечатан на принтере, отредактирован с помощью текстового редактора.

Общий вид функции форматного вывода:

```
int fprintf(указатель_на_поток,  форматная_строка,
список__переменных)
```

Использовавшаяся нами ранее функция printf ( ) для организации вывода на экран является частным вариантом функции fprintf ( ). Функция printf ( ) работает лишь со стандартным потоком stdin, который всегда связывается системой с дисплеем. Не будет ошибкой, если в программе вместо printf ( ) написать fprintf (stdin,...).

Правила использования спецификаторов форматов при записи в файлы на диске точно такие же, как и при выводе на экран.

Пример 5. Составим программу, по которой будет рассчитана и записана в файл таблица квадратных корней для целых чисел от 1 до 10. Для контроля эта же таблица выводится на экран.

```
//Таблица квадратных  корней
#include <stdio.h>
#include <iostream.h>
#include <math.h>
#include <locale.h>
int main( )
{
setlocale(LC_ALL, "RUS");
FILE *fp;
int  x;
fp = fopen("test.dat", "w");
//Вывод на экран и в файл шапки таблицы
printf("\tТаблица квадратных  корней\n");
fprintf(fp,"\tТаблица квадратных  корней\n");
printf("\tx \t\tsqrt(x)\n");
fprintf(fp, "\tx \t\tsqrt(x)\n");
//Вычисление и вывод таблицы квадратных корней
//на экран и в файл
for(x = 1; x <= 10;  x++)
{
printf("\t%f \t%f\n", float(x), sqrt(float(x)));
```

```
fprintf (fp, "\t%f \t%f\n" ,float(x) ,
sqrt(float(x)));
}
fclose(fp);
}
```

Если теперь с помощью текстового редактора (например, входящего в систему программирования) открыть файл test.dat , то на экране увидим:

Таблица квадратных корней

x	sqrt(x)
1.000000	1.000000
2.000000	1.414214
3.000000	1.732051
4.000000	2.000000
5.000000	2.236068
6.000000	2.449490
7.000000	2.645751
8.000000	2.828427
9.000000	3.000000
10.000000	3.162278

Теперь эти результаты можно распечатать, включить в текст отчета и т. п.

Форматный ввод из текстового файла осуществляется с помощью функции fscanf(), общий формат которой выглядит следующим образом:

```
int  fscanf(указатель_на_поток,  форматная_строка, список_адресов пере-
менных);
```

Данной функцией удобно пользоваться в тех случаях, когда исходные данные заранее подготавливаются в текстовом файле.

В следующем примере числовые данные из файла test.dat , полученного в результате выполнения предыдущей программы, вводятся в числовые массивы x и y. Для контроля значения элементов массивов выводятся на экран. Предварительно с помощью текстового редактора в файле test.dat удаляются две первые строки с заголовками. В результате в файле останутся только числа.

Пример 6.

```
//Ввод чисел из файла
#include <stdio.h>
#include <iostream.h>
#include <math.h>
#include <locale.h>
int main( )
{
setlocale(LC_ALL, "RUS");
FILE *fp;
int  i ;
float  x[10], y[10];
fp = fopen("test.dat", "r");
for(i = 0;  i < 10;  i++)
{
fscanf(fp,"%f %f", &x[i], &y[i]);
```

```
printf("%f %f\n ", x[i], y[i]);
}
fclose(fp);
}
```

Упражнения

1. Составить программу, которая формирует файл целых чисел, получаемых с помощью датчика случайных чисел.
2. Составить программу, которая в файле, сформированном программой из предыдущей задачи, находит наибольшее и наименьшее значения.
3. Составить программу, которая формирует файл из строчных латинских букв, выбираемых случайным образом.
4. Составить программу, которая в файле, сформированном программой из предыдущей задачи, подсчитает количество букв z-
5. Составить программу, записывающую на диск таблицу Менделеева.
6. Составить программу, которая в файле, сформированном в результате решения предыдущей задачи, будет отыскивать сведения о заданном химическом элементе.
7. Сведения о деталях, хранящихся на складе, содержат следующие атрибуты: название, количество, стоимость одной детали.
8. Составить программы, решающие следующие задачи:
   а) заполнить файл с информацией о деталях на складе;
   б) вычислить общую стоимость деталей;
   в) выяснить, какие детали имеются в наибольшем количестве, какие - в наименьшем;
   г) вывести информацию о наличии на складе деталей данного типа и их количестве;
   д) внести изменения в файл после выдачи со склада определенного количества данного вида деталей. Если какой-то тип деталей полностью выбран со склада, то уничтожить запись о ней в файле.

Информационные источники

Семакин И.Г., Шестаков А.П. Основы программирования: Учебник.- М.: Мастерство, 2002.- 432 с.