Файловый ввод и вывод

Стандартные функции

Стандартные функции управления файловым вводом и выводом `fopen`, `fscanf`, `fprintf`, `fclose` и `feof` используют тип данных **FILE**. Значение типа **FILE** содержит информацию об открытом файле. Приведенные стандартные функции возвращают или принимают в качестве входного параметра указатель на значение типа **FILE**. Указатель следует воспринимать как адрес в памяти, по которому расположено значение.

Ниже приведено определение двух переменных `fin` и `fout`, которые используются для хранения указателей на значения типа **FILE**. При определении указателя перед его идентификатором ставится символ `*`.

```
FILE *fin, *fout;
```

Для открытия файла используется функция `fopen`. Данная функция имеет два входных параметра. Первый параметр – это строка, содержащая имя открываемого файла. Второй параметр – это строка, содержащая флаги с которыми открывается файл. Флаг `r` говорит о том, что файл открывается для чтения, а флаг `w` – для записи.

Ниже приведен пример открытия файла с именем `input.txt` для чтения. Функция `fopen` возвращает указатель на значение типа **FILE**, содержащее информацию об открытом файле. Данный указатель сохраняется в переменную `fin`.

```
fin = fopen("input.txt", "r");
```

В следующем примере файл с именем `output.txt` открывается для записи.

```
fout = fopen("output.txt", "w");
```

Операция открытия файла может закончиться неудачно. В этом случае функция `fopen` возвратит нулевой указатель. Подобные ситуации необходимо отслеживать и соответствующим образом реагировать. В следующем фрагменте программы при неудачном открытии файла `input.txt` в стандартный поток вывода (на экран) будет напечатано сообщение о возникшей ошибке. При этом, функция, инициировавшая открытие файла, будет завершена и возвратит значение `-1`.

```
fin = fopen("input.txt", "r");
if(!fin)
{
    printf("Can't open file input.txt ...\n");
    return -1;
}
```

Для чтения из файла используется функция `fscanf`. Данная функция является обобщением ранее рассмотренной функции `scanf`. В качестве первого входного параметра функция `fscanf` принимает указатель на состояние открытого файла, из которого предстоит чтение. Вторым параметром является строка, содержащая спецификаторы формата ввода. Далее, в функцию `fscanf` передаются адреса переменных, в которые должны быть записаны считанные значения. Возвращает функция `fscanf` количество успешно считанных значений или константу `EOF` (обычно это `-1`) в случае конца файла.

Допустим, в файле `input.txt` записаны три значения типа **double**.

```
1.0 2.0 3.0
```

В следующем фрагменте программы пять раз последовательно вызывается функция `fscanf`, которая каждый раз пытается считать из файла значение типа **double** и записать считанное значение в переменную `x`. При первом, втором и третьем вызове функция `fscanf` возвратит значение `1`. При этом в переменную `x` сначала будет записано значение `1.0`, затем значение `2.0`, а потом – значение `3.0`.

При четвертом и пятом вызове функция `fscanf` будет возвращать константу `EOF`, свидетельствующую о конце файла. Весь файл был прочитан.

```
double x;
...
fscanf(fin, "%lf", &x);     // return 1      x == 1.0
fscanf(fin, "%lf", &x);     // return 1      x == 2.0
fscanf(fin, "%lf", &x);     // return 1      x == 3.0
fscanf(fin, "%lf", &x);     // return EOF    x == 3.0
fscanf(fin, "%lf", &x);     // return EOF    x == 3.0
```

Допустим, в файле `input.txt` сначала записаны два значения типа **double**, затем символ `a`, а потом еще одно значение типа **double**.

```
1.0 2.0 a 3.0
```

Снова рассмотрим фрагмент программы с вызовами функции `fscanf`. В первых двух вызовах функция `fscanf` возвратит значение `1`. При этом сначала в переменную `x` будет записано значение `1.0`, а затем значение `2.0`. При последующих вызовах функция `fscanf` будет возвращать `0`. Значение переменной `x` не будет меняться.

```
double x;
...
fscanf(fin, "%lf", &x);     // return 1      x == 1.0
fscanf(fin, "%lf", &x);     // return 1      x == 2.0
fscanf(fin, "%lf", &x);     // return 0      x == 2.0
fscanf(fin, "%lf", &x);     // return 0      x == 2.0
fscanf(fin, "%lf", &x);     // return 0      x == 2.0
```

Как правило для обработки числовой последовательности, записанной в файле, используется цикл следующего вида.

```
double x;
...
for(;fscanf(fin, "%lf", &x) == 1;)
{
    ...
}
```

Данный цикл выполняется до тех пор, пока успешно считывается очередное числовое значение. Цикл завершается либо в случае конца файла, либо когда очередное значение в файле не удовлетворяет спецификатору формата ввода.

Желательно уметь различать эти два случая. Первый случай говорит о том, что входные данные были корректными, и все они были обработаны (во всяком случае прочитаны). Во втором случае на вход программы были поданы некорректные входные данные.

Стандартная функция `feof` позволяет решить эту задачу. Данная функция на вход принимает указатель на состояние открытого файла. Функция возвращает ненулевое значение если был достигнут конец файла, в противном случае возвращает `0`.

```
if(!feof(fin))
{
    // Error. Wrong input data.
}
```

Для печати в файл используется функция `fprintf`. Данная функция является обобщением ранее рассмотренной функции `printf`. В качестве первого входного параметра функция `fprintf` принимает указатель на состояние открытого файла, в который предстоит запись. Вторым параметром является строка формата печати. Последующие параметры задают значения, которые должны быть напечатаны при помощи соответствующих спецификаторов, присутствующих в строке формата печати.

В следующем фрагменте программы в файл записывается значение типа **double**, хранящееся в переменной `x`.

```
double x;
...
```

```
fprintf(fout, "%f", x);
```

Для закрытия файла используется функция `fclose`. Данная функция на вход принимает указатель на состояние открытого файла. Ниже приводится фрагмент программы, в котором закрываются ранее открытые файлы `input.txt` и `output.txt`.

```
fclose(fin);
fclose(fout);
```

Пример

В качестве примера рассмотрим решение следующей задачи. В файле `input.txt` записана последовательность вещественных чисел. Программа должна за один проход прочитать эту последовательность из файла `input.txt`, вычислить ее среднее арифметическое и записать результат в файл `output.txt`.

Программа будет иметь следующую структуру.

```
#include <stdio.h>

int calculateAverage(FILE *fin, FILE *fout);

int main(void)
{
    ...
}

int calculateAverage(FILE *fin, FILE *fout)
{
    ...
}
```

Вся вычислительная часть вынесена в функцию `calculateAverage`. Данная функция имеет два входных параметра. Параметр `fin` – это указатель на состояние открытого входного файла, из которого предстоит чтение. Параметр `fout` – это указатель на состояние открытого выходного файла, в который будет записан результат. Функция возвращает целочисленное значение. В случае успеха возвращается значение `0`. В случае ошибки (некорректные входные данные) функция возвращает значение `-1`.

```
1  int calculateAverage(FILE *fin, FILE *fout)
2  {
3      double x, sum;
4      int    count;
5
6      for(sum = 0., count = 0; fscanf(fin, "%lf", &x) == 1;)
7      {
8          sum += x;
9          count ++;
10     }
11
12     if(!feof(fin))
13         return -1;
14
15     fprintf(fout, "%f", count ? sum / count : 0.);
16
17     return 0;
18 }
```

В строке 3 содержится определение переменных `x` и `sum` типа **double**. В переменную `x` будет записываться очередное успешно считанное значение из входного файла. В переменной `sum` будет храниться сумма всех успешно считанных значений из входного файла.

В строке 4 содержится определение переменной `count` типа **int**. В данной переменной будет храниться число успешно считанных значений из входного файла.

В строках 6 - 10 содержится цикл **for**. В рамках инициализирующего выражения переменным `sum` и `count` присваиваются нулевые значения. Это соответствует ситуации пустой последовательности. Проверочное выражение включает вызов функции `fscanf` для считывания значения типа **double** и сравнения результата выполнения функции `fscanf` с

единицей. Таким образом, цикл будет продолжаться до тех пор, пока будет осуществляться успешное считывание очередного значения типа **double** из входного файла.

Тело цикла представляет собой составной оператор, внутри которого к текущему значению переменной `sum` прибавляется очередное считанное значение (строка 8), а счетчик `count` увеличивается на единицу (строка 9).

В строках 12-13 присутствует оператор **if**, с помощь которого проверяется, был ли достигнут конец входного файла. Если конец входного файла не был достигнут, то это означает некорректные входные данные. В этом случае функция `calculateAverage` завершает свое выполнение и возвращает значение `-1`.

В случае корректных входных данных осуществляется переход к строке 15. В этом месте содержится вызов функции `fprintf`, которая печатает среднее арифметическое считанной последовательности в выходной файл. Среднее арифметическое вычисляется при помощи условной операции `?:`. Предполагается, что среднее арифметическое пустой последовательности равно нулю.

После осуществляется переход к строке 17. Функция `calculateAverage` завершает свое выполнение и возвращает значение `0`.

Ниже представлено содержание функции `main`.

```
1  int main(void)
2  {
3      FILE *fin, *fout;
4      int status;
5
6      fin = fopen("input.txt", "r");
7      if(!fin)
8      {
9          printf("Can't open input.txt ...\n");
10         return -1;
11     }
12
13     fout = fopen("output.txt", "w");
14     if(!fout)
15     {
16         printf("Can't open output.txt ...");
17         fclose(fin);
18         return -1;
19     }
20
21     status = calculateAverage(fin, fout);
22     if(status)
23         printf("File input.txt contains non double values ...\n");
24
25     fclose(fin);
26     fclose(fout);
27
28     return status;
29 }
```

В строке 3 определяются переменные `fin` и `fout`. Переменная `fin` будет хранить указатель на состояние открытого входного файла, а переменная `fout` – указатель на состояние открытого выходного файла.

Строка 4 содержит определение переменной `status` типа **int**. В этой переменной будет храниться результат выполнения функции `calculateAverage`.

В строке 6 осуществляется вызов функции `fopen` с целью открыть для чтения файл с именем `input.txt`. Строки 7-11 содержат оператор **if**, с помощью которого проверяется успешность открытия файла. В случае ошибки функция `fopen` вернет нулевой указатель. В этом случае в стандартный поток вывода будет напечатано сообщение об ошибке (строка 9) и функция `main` завершит свое выполнение с кодом `-1` (строка 10).

В строке 13 осуществляется вызов функции `fopen` с целью открыть для записи файл с именем `output.txt`. Строки 14-19 содержат оператор **if**, с помощью которого проверяется успешность открытия файла. В случае ошибки функция `fopen` вернет нулевой указатель.

В этом случае в стандартный поток вывода будет напечатано сообщение об ошибке (строка 16). Будет вызвана функция `fclose` (строка 17) с целью закрытия входного файла. Далее, функция `main` завершит свое выполнение с кодом `-1` (строка 18).

В случае успешности открытия входного и выходного файлов будет вызвана функция `calculateAverage` (строка 21). Если эта функция вернет ненулевое значение (строки 22-23), то в стандартный поток вывода будет напечатано сообщение о том, что файл `input.txt` содержит некорректную числовую последовательность.

Далее, в строке 25 содержится вызов функции `fclose`, закрывающий входной файл. В строке 26 содержится вызов функции `fclose`, закрывающий выходной файл.

В строке 28 функция `main` завершает свое выполнение и возвращает значение, которое вернула функция `calculateAverage`.

Тестирование программы

При тестировании программы должны быть рассмотрены три случая: 1) входной файл `input.txt` отсутствует, 2) входной файл `input.txt` содержит некорректные данные; 3) входной файл `input.txt` содержит корректные данные.

В первом и втором случае программа должна завершиться с ненулевым статусом. Проверить статус, который вернула только что выполненная программа, можно с помощью команды.

```
echo $?
```

В третьем случае необходимо убедиться в том, что программа вернула нулевой статус, а дальше проверить содержимое выходного файла `output.txt`.

Подготовить входные данные можно двумя способами. Первый способ – отредактировать входной файл `input.txt` с помощью редактора `vim`. Второй способ – воспользоваться стандартной программой `echo` и механизмом перенаправления стандартного вывода программы в файл. Рассмотрим следующую команду.

```
echo "1.0 2.0 3.0 4.0" > input.txt
```

Программа `echo` печатает переданные ей аргументы командной строки в свой стандартный поток вывода. Символ `>` говорит о том, что стандартный поток вывода программы будет перенаправлен в файл `input.txt`. Таким образом, после успешного выполнения приведенной выше команды будет создан новый (перезаписан новым содержимым существующий) файл `input.txt`. Данный файл будет содержать следующую споследовательность символов.

```
1.0 2.0 3.0 4.0
```

Процедура подготовки входного файла, запуска программы, просмотра статуса ее завершения, просмотра выходного файла может быть автоматизирована с помощью использования составной команды. Пример составной команды приведен ниже.

```
echo "1.0 2.0 3.0 4.0" > input.txt; ./average; echo $?; cat output.txt
```

Составная команда представляет собой последовательность индивидуальных команд, отделенных друг от друга символом `;`. Выполнение составной команды заключается в последовательном выполнении входящих в ее состав индивидуальных команд.

Потоки

Рассмотренные стандартные функции `fopen`, `fscanf`, `fprintf`, `fclose` и `feof` реализуют так называемый *потоковый* ввод и вывод. При успешном вызове функции `fopen` говорят о создании потока ввода (вывода), связанного с открытым файлом. Говорят, что функция `fprintf` печатает в поток вывода, а функция `fscanf` считывает из потока ввода. Функция `fclose` закрывет поток.

Существуют глобальные переменные `stdin` и `stdout`, указывающие на значения типа **FILE**. Переменная `stdin` указывает на состояние стандартного потока ввода, а переменная `stdout` – на состояние стандартного потока вывода.

Вызов функции `scanf` эквивалентен вызову функции `fscanf`, при котором на место первого параметра подставлено значение переменной `stdin`. Аналогично, вызов функции `printf` эквивалентен вызову функции `fprintf`, при котором на место первого параметра подставлено значение переменной `stdout`.