Операции с файлами

Стандарты POSIX

POSIX (англ. portable operating system interface — переносимый интерфейс операционных систем) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (системный API – application programming interface), ***библиотеку языка С*** и набор приложений и их интерфейсов.

Международная организация по стандартизации (ISO) совместно с Международной электротехнической комиссией (IEC) приняли стандарт POSIX под названием **ISO/IEC 9945** (действующая редакция – **ISO/IEC 9945:2009**).

POSIX обеспечивает стандартизацию **интерфейса командной строки**.

Файлы в POSIX

Понятие **«Файл»** в POSIX является первичным. Его нельзя строго определить, но можно пояснить с помощью других понятий и отношений.

Файл - объект, допускающий чтение и/или запись и имеющий такие атрибуты, как ***права доступа*** и тип.

Типы файлов:

- обычный файл,
- символьный и блочный *специальные файлы*,
- канал,
- символьная ссылка,
- *сокет*
- каталог.

Основные операции с файлами (определяются правами доступа):

- Чтение
- Запись
- Выполнение.

Файлы в Си

Для работы с файлами в программах на Си нужно определить указатель на файл («дескриптор») и режим доступа к файлу. Указатель должен быть указателем на специальный тип FILE.

Режимы доступа:

	Если файл существует	**Если файл не существует**
"r"	Файл открывается для чтения с 0-го байта	Указатель получает значение NULL
"w"	Файл открывается для записи, ранее записанное теряется («запись поверх»)	Файл создается и открывается для записи с 0-го байта
"a"	Запись производится в конец файла («дописывание»)	Файл создается и открывается для записи с 0-го байта

Файлы в Си

Режимы доступа (небезопасные):

	Если файл существует	**Если файл не существует**
"r+"	Файл открывается для чтения и записи с 0-го байта	Runtime error
"w+"	Файл открывается для чтения и записи (перезаписи)	Файл создается и открывается для чтения и записи с 0-го байта
"a+"	Запись производится в конец файла («дописывание»)	Файл создается и открывается для записи с 0-го байта

Файлы и потоки

Все файлы в Си представляются в виде логических псевдоустройств, называемых потоками (ввода/вывода). Потоки бывают текстовые и двоичные (бинарные).

Несмотря на различную природу реальных файлов, работа с потоками устроена единообразно.

Всегда существует три потока «по умолчанию»

- ввод: **`stdin`**
- вывод: **`stdout`**
- диагностика: **`stderr`**.

Функции для работы с файлами (потоками)

1) Открытие и закрытие
2) Проверка конца файла
3) Чтение и запись символа
4) Чтение и запись строк
5) Форматный ввод и вывод
6) Позиционирование в файле
7) Блочный ввод и вывод

Открытие и закрытие

`fopen()` возвращает

- указатель типа **`FILE*`**, если поток удалось открыть
- **`NULL`**, если произошла ошибка (При этом переменной **`errno`** будет присвоен код ошибки).

`fclose()` возвращает

- 0, если поток данных и связанный с ним файл успешно закрыты
- **`EOF`** (-1), если во время работы функции возникла ошибка. (При этом переменной **`errno`** будет присвоен код ошибки).

*Значение **`errno`** определены в **`errno.h`***

См. пример lect-08-01.c (ошибку открытия смоделировать легко, ошибку закрытия — трудно).

Открытие и закрытие

Если объявлен **`FILE *df`**

`df = fopen("datafile.txt","r");` неправильно!

Правильно (можно) так:

```
if((df = fopen("datafile.txt","r"))!=NULL)
{
   … работа с файлом …
  if (fclose(df) == EOF) printf ("Error closing!\n");
  else printf ("Closing OK\n");;
}
else
{
   … обработка ошибки …
}
```

Где выполнять **`fclose()`**?

Открытие и закрытие

Пусть нужно записать некую строку в файл с именем, задаваемым пользователем, причем выбрать вариант — переписывать файл или дописывать в него.

См. пример lect-08-02.c

Открытие и закрытие

Нюансы:

```
if((df = fopen("datafile.txt","r"))== NULL)
{
  printf("Error! No data present!");
  exit(1);
}
```

1) Противоречит «структурному» подходу

2) Разве это нештатное завершение?

Открытие и закрытие

`freopen()` – перенаправление вывода. В частности, **`stderr`** может быть перенаправлен в файл (лог-файл). Тогда можно следить за сообщениями об ошибках, которые генерируются в процессе работы программы.

(см. примеры lect-08-03.c,

lect-08-03a.c – эксперименты!).

«Классические» лог-файлы имеют текстовый формат.

`fileno()` – возвращает целое положительное число (номер дескриптора файла) или -1 при ошибке

(см. пример lect-08-04.c).

Проверка конца файла

В **stdio.h**

```
#define EOF (-1)
```

Применяется для текстовых файлов. На клавиатуре - <CTRL>+D в Linux/MacOS и <CTRL>+Z в Windows.

```
while(!feof(df))
{
    … работа с файлом …
}
```

(Пример lect-08-05.c) Эксперименты с **feof()**

Другой вариант — «кончились байты», т. к. можно получить сведения о количестве байтов в файле.

Задача из учебника (Керниган и Ритчи): «Напишите программу печатающую значение EOF»

```
printf("%c",(char)EOF);
```
(см. пример lect-08-06.c)

Чтение и запись символов

`fgetc()` – читает один символ (байт) из потока. При достижении конца файла возвращает **`EOF`**. При ошибке чтения возвращает **`EOF`** и код ошибки в **`errno`**.

`fputc()` – записывает один символ в поток. При ошибке записи возвращает **`EOF`** и код ошибки в **`errno`**.

`ungetc()` – возвращает только что прочитанный символ в поток (в большинстве случаев не имеет смысла).

См. пример lect-08-07.c (закомментированый **`ungetc()`** и **`ungetc()`** без комментария).

Чтение и запись строк

`fgets()` – читает строку из файла, включая символ перевода строки (**`\n`**), но не более **`MAXLEN-1`** символов. Возвращает указатель на строку. После последнего прочитанного символа добавляется **`´\0´`**.

Если достигнут конец файла, возвращает **`NULL`**.

Поскольку контролируется длина строки, можно использовать для ввода с клавиатуры (из **`stdin`**).

См. пример lect-08-08.c.

Чтение и запись строк

`fputs()` – пишет строку в указанный поток (включая **`stdout`**). Если запись не удалась, возвращает **`EOF`**.

Длина строки не контролируется, т. к. строка уже кем-то сформирована.

См. пример lect-08-09.c.

putw() и getw()

Нестандартные функции, унаследованные от Turbo C (Borland). Поддерживаются в **`stdio.h`** для gcc.

Записывают и считывают коды значений. Позволяют создавать и читать не текстовые (бинарные) файлы.

Если с помощью **`putw()`** записаны числа, то они читаются с помощью **`getw()`**.

Чтобы сформировать текст с помощью **`putw()`**, нужно использовать ASCII-коды в hex.

См. примеры lect-08-10.c, lect-08-10a.c.

Форматный ввод и вывод

`fscanf()` – аналог «обычного» **`scanf()`**, но первый аргумент — «указатель» на файл (имя потока). Нет смысла использовать для **`stdin`**.

Нюанс: возвращает количество прочитанных переменных. Это имеет смысл проверять.

Для «строковых» элементов данных пробелы недопустимы.

Полезна, если точно известен «формат» файла данных.

См. пример lect-08-11.c.

Форматный ввод и вывод

`fprintf()` – аналог «обычного» **`printf()`**, но первый аргумент — «указатель» на файл (имя потока). Нет смысла использовать для **`stdin`**.

Нюанс: возвращает количество выведенных переменных.

В спецификаторах форматов можно указывать переменные ширину и точность!

См. пример lect-08-12.c.

Позиционирование в файле

`fseek()` позволяет установить текущую позицию в файле (в байтах) указав значение **`offset`** (целое) от начала отсчета.

Началом отсчета м.б. **`SEEK_SET`** (начало файла) или **`SEEK_END`** (конец файла). Если ошибки нет, функция возвращает 0, иначе — **`errno`**.

Дальнейшее чтение (запись) пойдет с указанной позиции.

`ftell()` выводит текущую позицию (в байтах) в виде целого числа (**`long int`**). В случае ошибки возвращает -1.

`rewind()` возвращает указатель позиции в начало файла.

См. примеры lect-08-16.c, lect-08-17.c, lect-08-18.c.

Блочный ввод и вывод

`fwrite()` позволяет записывать в файл целиком массивы. Файл получается не текстовый (бинарный, двоичный).

Требуется указать адрес переменной, размер переменной (блока), количество блоков и имя потока («указатель» на файл).

Возвращает число реально записанных блоков (можно контролировать).

См. примеры lect-08-19.c, lect-08-20.c.

Блочный ввод и вывод

`fread()` позволяет считывать из бинарного файла целиком массивы.

Требуется указать адрес переменной, размер переменной (блока), количество блоков и имя потока («указатель» на файл).

Возвращает число реально прочитанных элементов (можно контролировать).

См. пример lect-08-21.c

Подробности и дополнительная информация см., например, http://cpp.com.ru/shildt_spr_po_c/about.html