Примеры из K&R

Высокоуровневый ввод-вывод. Реализация команды cat

Язык Си обладает обширным набором средств ввода/вывода, в основе которых лежит концепция потока данных. Поток — это файл или иной объект ввода/вывода, включая терминал и любое иное физическое устройство. Сведения о потоках хранятся в данных типа FILE (определенном, наряду с другими средствами ввода/вывода, в `stdio.h`). Объект типа FILE создается вызовом функции `fopen`, возвращающей указатель на созданный объект (указатель файлового типа). Указатели этого типа (file pointer) используются в качестве аргументов в большинстве функций ввода/вывода (см. справочник по функциям станд. библиотеки). Информация, содержащаяся в объекте типа FILE, включает текущую позицию в потоке (указатель позиции в файле), указатели на буферы, используемые потоком, а так же данные о том, произошла ли ошибка, и достигнут ли конец файла. Как правило, потоки, если они не связаны с интерактивными устройствами, буферизуются.

Потоки бывают текстовые и двоичные (бинарные). Текстовый поток состоит из последовательности текстовых символов, разбитых на строки. Каждая строка состоит из некоторого числа (возможно, нуля) символов, за которыми следует символ новой строки ‘\n’ , принадлежащий этой же строке. Двоичные потоки — это последовательности значений данных типа char. Поскольку в Си любые данные можно отобразить на массив значений типа char, двоичные потоки могут прозрачно записывать внутренние данные. Любая реализация допускает игнорирование различий между текстовыми и двоичными потоками. В unix –системах текстовые и двоичные потоки не различаются.

При запуске программы Си автоматически открываются три стандартных текстовых потока: стандартное устройство ввода (`stdin`), стандартное устройство вывода (`stdout`), и стандартное устройство сообщений об ошибках (`stderr`). Все три выражения — `stdin`, `stdout`, `stderr` — имеют тип указатель на FILE.

Для демонстрации работы с файлами, реализуем простую версию команды cat.

Команда `cat f1 f2 … fN` выводит в стандартный выходной поток содержимое файлов `f1, f2, …, fN`. Если аргументов нет, выводится стандартный входной поток.

```
#include <stdio.h>
/* cat: конкатенация файлов, версия 1 */
main(int argc, char *argv[])
{
    FILE *fp;
    void fllecopy(FILE *, FILE *);
    if (argc == 1)    /* нет аргументов; копируется станд. ввод */
         filecopy(stdin, stdout);
    else
        while (--argc > 0)
          if ((fp = fopen(*++argv, "r")) == NULL) {
             printf("cat: не могу открыть файл %s\n", *argv);
             return 1;
           } else {
                      filecopy(fp, stdout);
                      fclose(fp);
            }
```

```
    return 0;
   }

/* filecopy: копирует файл ifp в файл ofp */

 void filecopy(FILE *ifp, FILE *ofp)
{
     int c;
     while ((c = getc(ifp)) != EOF)
        putc(c, ofp);
}
```

Управление ошибками (stderr, exit, perror)

Обработку ошибок в cat нельзя признать идеальной. Например, если cat является частью конвейера «`cat f1 f2 | sort`», то диагностическое сообщение о том, что не удалось открыть файл `f2`, попадет напрямую по «трубопроводу» программе sort как одна из строк для сортировки. А при выполнении команды «`cat f1 f2 > f3`», в которой стандартный вывод перенаправлен в файл `f3`, диагностическое сообщение вместе со всем выводом программы cat также попадет в файл `f3`.

Чтобы лучше справиться с этой проблемой, программе помимо стандартного вывода `stdout` придается еще один выходной поток, называемый `stderr`. Вывод в `stderr` обычно отправляется на экран, даже если вывод `stdout` перенаправлен в другое место.

Перепишем `cat` так, чтобы сообщения об ошибках отправлялись в `stderr`.

```
#include <stdio.h>
/*cat: конкатенация файлов, версия 2                                    */
main(int argc, char *argv[ ])
{
    FILE *fp;
    void filecopy(FILE *, FILE *);
    char *prog = argv [0];   /* имя программы                          */
    if (argc == 1)   /* нет аргументов; копируется станд. ввод        */
        filecopy(stdin, stdout);
    else
      while (--argc > 0)
        if ((fp = fopen(*++argv, "r")) == NULL) {
          fprintf(stderr, "%s: не могу открыть файл %s\n",
                                                 prog, *argv);
          exit(1);
        } else {
          filecopy(fp, stdout);
          fclose(fp);
        }
    if (ferror(stdout)) {
      fprintf (stderr, "%s: ошибка записи в stdout\n", prog);
      exit(2);
    }
exit(0);
}
```

Программа сигнализирует об ошибках двумя способами.

1) Сообщение об ошибке при помощи `fprintf` посылается в `stderr` с тем, чтобы оно попало на экран, а не оказалось в «трубопроводе» или в другом файле вывода. Имя программы, хранящееся в `argv[0]`, включено в сообщение, чтобы в случаях, когда данная программа работает совместно с другими, был ясен источник ошибки.

2) Обращение к библиотечной функции exit, завершает работу программы так, что аргумент функции exit доступен некоторому процессу («отцу»), запустившему на выполнение данную программу. Процесс-«отец» может как-то отреагировать на возвращенное ему значение.

Функция `ferror` выдает ненулевое значение, если в файле `fp` была обнаружена ошибка.

```
int ferror(FILE  *fp)
```

Хотя при выводе редко возникают ошибки, все же они встречаются (например, оказался переполненным диск); поэтому в программах широкого пользования они должны тщательно контролироваться.

Функция

```
int feof(FILE  *fp)
```

аналогична функции ferror; она возвращает ненулевое значение, если встретился конец указанного в аргументе файла.

В наших небольших иллюстративных учебных программах мы не заботились о выдаче статуса выхода, т.е. выдаче некоторого числа, характеризующего состояние программы в момент завершения: работа закончилась нормально или прервана из-за ошибки? Если работа прервана в результате ошибки, то какой? Любая серьезная (индустриальная) программа должна выдавать статус выхода (с помощью функции `exit()` или с помощью оператора `return`, расположенного в функции `main`).

В заголовочном файле `errno.h` определяются дополнительные средства, предназначенные для вывода сообщений об ошибках в стандартных библиотеках.
Во внешнюю (глобальную) переменную `errno` записываются коды ошибок библиотечных функций, определяемые реализацией. Традиционно, их имена в `errno.h` , начинаются на E. Все коды ошибок — это положительные целые. Библиотечные функции не должны обнулять `errno`.

Один из распространенных способов использования переменной `errno` — обнуление ее перед вызовом функции, затем проверка после вызова:

```
errno = 0;
x = sqrt(y);
if (errno) {
    fprintf (stderr, ”Аварийное завершение sqrt, код %d\n”, errno);
    x=0;
}
```

Примеры кодов ошибок: EDOM — значение аргумента не входит в область значений, определенных для данной математической функции (например, отрицательный аргумент функции `log`); ERANGE — значение, возвращаемое функцией, не входит в интервал допустимых значений (например, при возведении большого числа в большую степень функцией `pow`).

Функция `strerror` возвращает указатель на строку сообщения об ошибке, текст которого зависит от реализации. Строку сообщения модифицировать нельзя. При новой ошибке повторное обращение к `strerr` дает новое сообщение.

Функция perror выводит в выходной поток сообщений об ошибках следующую последовательность: строку аргумента s, двоеточие, пробел, краткое описание ошибки, код которой указан в переменной errno, и символ новой строки – '\n'.

```
#include <math.h>
#include <errno. h>
…
errno=0;
x= sqrt(y);
if (errno) {
    perror("Аварийное завершение sqrt");
    x=0;
}
```

Теперь, в случае аварийного завершения sqrt, последует вывод примерно следующего сообщения:

```
Аварийное завершение sqrt: domain error
```

Если при выполнении функции fopen возникает ошибка, ее код записывается в переменную errno, а функция возвращает нулевой указатель. Функция fclose закрывает должным образом открытый поток и очищает все внутренние буферы данных. Если при выполнении функции происходит ошибка, она возвращает значение EOF, иначе — нуль.

Ниже приведены примеры открытия и закрытия обычных текстовых файлов. Предусмотрены обработка ошибок и печать диагностики; возвращаемые значения – такие же, как у функций fopen и fclose.

```
#include<errno.h>
#include<stdio.h>
FILE *open_input(const char *filename)
    /* Открытие файла filename; в случае ошибки, возврат NULL */
{
    FILE * f;
    errno = 0;
    /* Значение NULL аргумента filename недопустимо для следующей функции */
    if (filename==NULL) filename = "\0";
    f = fopen(filename, "r");  /* режим "w" указывается для open_output */
    if (f==NULL)
        fprintf(stderr,
            "open_input(\"%s\" с  ошибкой: %s\n",
              filename, strerror(errno));
    return f;
}

int close_file(FILE *f)
/* Закрыть файл f */
{
    int s=0;
    if (f == NULL) return 0; /* Игнорировать это условие */
    errno=0;
    s=fclose(f);
    if (s==EOF) perror("Закрытие завершилось ошибкой");
    return s;
}
```