Operações usando streams

Aqui serão descritas algumas das funções mais usuais para operações de entrada/saída usando streams. A relação completa e mais detalhada pode ser obtida na seção Input/Output on Streams do manual da GLibC. A maioria das funções aqui descritas está declarada no arquivo de cabeçalho `stdio.h`, enquanto suas equivalentes para caracteres de 16 bits estão declaradas no arquivo de cabeçalho `wchar.h`. Algumas funções adicionais estão definidas em `stdio_ext.h`.

Abrindo e fechando streams

```
#include <stdio.h>
FILE * fopen (const char *filename, const char *opentype)
```

Abre um arquivo indicado por `filename` e retorna um ponteiro para o *stream*. A *string* `opentype` define o modo de abertura do arquivo:

- `r` : abre um arquivo existente para leitura (*read*).
- `w` : abre um arquivo para escrita (*write*). Se o arquivo já existe, seu conteúdo é descartado. Senão, um novo arquivo vazio é criado.
- `a` : abre um arquivo para concatenação (*append*). Se o arquivo já existe, seu conteúdo é preservado e as escritas serão concatenadas no final do arquivo. Senão, um novo arquivo vazio é criado.
- `r+` : abre um arquivo existente para leitura e escrita. O conteúdo anterior do arquivo é preservado e o ponteiro é posicionado no início do arquivo.
- `w+` : abre um arquivo para leitura e escrita. Se o arquivo já existe, seu conteúdo é descartado. Senão, um novo arquivo vazio é criado.
- `a+` : abre um arquivo para escrita e concatenação. Se o arquivo já existe, seu conteúdo é preservado e as escritas serão concatenadas no final do arquivo. Senão, um novo arquivo vazio é criado. O ponteiro de leitura é posicionado no início do arquivo, enquanto as escritas são efetuadas no seu final.

```
#include <stdio.h>
FILE * fdopen (int filedes, const char *opentype)
```

Cria um novo *stream* usando o descritor de arquivo `filedes`. O parâmetro `opentype` é similar ao da função `fopen`.

```
#include <stdio.h>
FILE * freopen (const char *filename, const char *opentype, FILE *stream)
```

Fecha e abre novamente um *stream*, permitindo alterar o arquivo e/ou modo de abertura.

```
#include <stdio.h>
int fclose (FILE *stream)
```

Fecha um *stream*. Dados de saída em buffer são escritos, enquanto dados de entrada são descartados.

```
#include <stdio.h>
int fcloseall (void)
```

Fecha todos os *streams* abertos, de forma similar a `fclose`. Isso também é efetuado quando a função `main` termina ou quando a função `exit` é invocada.

```
#include <stdio.h>
```

```
FILE * fdopen (int filedes, const char *opentype)
```

Cria um novo *stream* usando o descritor de arquivo `filedes`. O parâmetro `opentype` é similar ao da função `fopen`.

```
#include <stdio.h>
int fileno (FILE *stream)
```

Retorna o descritor de arquivo associado ao *stream* indicado.

Exemplo: abrindo o arquivo x em leitura, através de um *stream*:

```
#include <stdio.h>

int main (int argc, char *argv[])
{
   FILE *arq ;

   arq = fopen ("x", "r") ;

   if ( ! arq )
   {
      perror ("Erro ao abrir arquivo x") ;
      exit (1) ;
   }

   fclose (arq) ;
   exit (0) ;
}
```

Exemplo: abre o arquivo x em leitura/escrita, criando-o se não existir:

```
#include <stdio.h>

int main (int argc, char *argv[])
{
   FILE *arq ;

   arq = fopen ("x", "w+") ;

   if ( ! arq )
   {
      perror ("Erro ao abrir/criar arquivo x") ;
      exit (1) ;
   }

   fclose (arq) ;
   exit (0) ;
}
```

Saída simples

Estas funções permitem gravar caracteres ou strings simples em streams.

```
#include <stdio.h>
int fputc (int c, FILE *stream)
```

Converte o inteiro `c` no tipo `unsigned char` e o escreve no stream indicado.

```
#include <stdio.h>
int putc (int c, FILE *stream)
```

Equivalente a `fputc`, mas implementado como macro (mais rápido).

```
#include <stdio.h>
int putchar (int c)
```

Equivale a `putc` sobre o stream `stdout`.

```
#include <stdio.h>
int fputs (const char *s, FILE *stream)
```

Escreve a string `s` no stream indicado. Caracteres de controle, como `\n`, são aceitos.

```
#include <stdio.h>
int puts (const char *s)
```

Escreve a string `s` no stream `stdout`, seguida por um *newline*.

Entrada de caracteres

Estas funções permitem ler caracteres isolados de um stream. O valor lido é um `int` indicando o caractere lido ou então o valor especial `EOF` (*End-Of-File*, geralmente -1).

```
#include <stdio.h>
int fgetc (FILE *stream)
```

Lê o próximo caractere do stream indicado.

```
#include <stdio.h>
int getc (FILE *stream)
```

Idem, mas implementado como macro (mais rápido).

```
#include <stdio.h>
int getchar (void)
```

Idem, sobre o stream padrão `stdin`.

```
#include <stdio.h>
int ungetc (int c, FILE *stream)
```

Coloca um caractere de volta no buffer de entrada do stream indicado (o arquivo não é alterado). Útil em programas de *parsing*.

Entrada de linhas

```
#include <stdio.h>
char * gets (char *s)
```

Lê caracteres de `stdin` até encontrar um *newline* e os armazena na string `s`. O caractere *newline* é descartado. Atenção: esta função é insegura, pois não provê segurança contra *overflow* na string `s`. Sempre que possível, deve ser usada a função `fgets` ou `getline`.

```
#include <stdio.h>
char * fgets (char *s, int count, FILE *stream)
```

Lê uma linha de caracteres do stream e a deposita na string `s`. O tamanho da linha é limitado em `count-1` caracteres, aos quais é adicionado o `'\0'` que marca o fim da string. O *newline* é incluso.

Observação: a biblioteca GNU LibC recomenda usar a função `getline` no caso de entradas contendo o caractere *null* (`\0`).

Entrada/saída de blocos de dados

A funções a seguir permitem ler/escrever arquivos com blocos de dados binários. Essa abordagem é mais eficiente para manipular grandes volumes de dados, mas os arquivos gerados não podem ser tratados por utilitários do UNIX como `grep`, `sort`, etc, e não são portáveis entre compiladores e/ou sistemas com diferentes arquiteturas.

```
#include <stdio.h>
size_t fread (void *data, size_t size, size_t count, FILE *stream)
```

Lê até `count` blocos de tamanho `size` e os deposita no vetor `data`, a partir do stream indicado. Retorna o número de blocos lidos.

```
#include <stdio.h>
size_t fwrite (const void *data, size_t size, size_t count, FILE *stream)
```

Escreve até `count` blocos de tamanho `size` do vetor `data` no stream indicado. Retorna o número de blocos escritos.

Saída formatada

As operações de entrada e saída formatada usam padrões para formatação dos diversos tipos de dados descritos em livros de programação em C e no manual da GLibC.

```
#include <stdio.h>
int printf (const char *template, ...)
```

Escreve dados usando a formatação definida em `template` no stream de saída padrão `stdout`.

```
#include <stdio.h>
int fprintf (FILE *stream, const char *template, ...)
```

Idêntico a `printf`, usando o stream indicado.

```
#include <stdio.h>
int sprintf (char *s, const char *template, ...)
```

Similar a `printf`, mas a saída é depositada na string `s`. Atenção: o programador deve garantir que s tenha tamanho suficiente para receber a saída; caso contrário, pode ocorrer um *overflow* com conseqüências imprevisíveis. As funções `snprintf` e `asprintf` são mais seguras e evitam esse problema.

Entrada formatada

```
#include <stdio.h>
int scanf (const char *template, ...)
```

Lê dados do stream `stdin` de acordo com a formatação definida na string `template`. Os demais argumentos são ponteiros para as variáveis onde os dados lidos são depositados. Retorna o número de dados lidos ou EOF.

```
#include <stdio.h>
int fscanf (FILE *stream, const char *template, ...)
```

Similar a `scanf`, mas usando como entrada o stream indicado.

```
#include <stdio.h>
int sscanf (const char *s, const char *template, ...)
```

Similar a `scanf`, mas usando como entrada a string `s`.

Posicionamento

Estas funções permitem conhecer/modificar o ponteiro de posição corrente de leitura/escrita em streams.

```
#include <stdio.h>
long int ftell (FILE *stream)
```

Retorna a posição corrente de leitura/escrita em `stream`. Ver também `ftello` e `ftello64`.

```
#include <stdio.h>
int fseek (FILE *stream, long int offset, int whence)
```

Ajusta posição do ponteiro no stream indicado. O ajuste é definido por `offset`, enquanto o valor de `whence` indica se o ajuste é relativo ao início do arquivo (SEEK_SET), à posição corrente (SEEK_CUR) ou ao final do arquivo (SEEK_END). Ver também `fseeko` e `fseeko64`.

```
#include <stdio.h>
void rewind (FILE *stream)
```

Reposiciona o ponteiro no início (posição 0) do stream indicado.

Controle de travas

Todos os acessos a um stream são efetuados de forma atômica, com sincronização através de uma trava

implícita (automática). Todavia, algumas funções são providas para controlar explicitamente a trava associada a um stream, caso isso seja necessário.

```
#include <stdio.h>
void flockfile (FILE *stream)
```

Obtém a trava associada ao stream. O processo ou thread fica bloqueado até obter a trava.

```
#include <stdio.h>
int ftrylockfile (FILE *stream)
```

Tenta obter a trava associada ao stream. Retorna imediatamente, com erro se não conseguir.

```
#include <stdio.h>
void funlockfile (FILE *stream)
```

Libera a trava obtida explicitamente com `flockfile` ou `ftrylockfile`.

```
#include <stdio.h>
#include <stdio_ext.h>
int __fsetlocking (FILE *stream, int type)
```

Permite alterar a forma de travamento do stream. Os valores possíveis para type são: `FSETLOCKING_INTERNAL` (uso implícito da trava por todas as funções que acessam o stream), `FSETLOCKING_BYCALLER` (a trava somente é usada de forma explícita) e `FSETLOCKING_QUERY` (consulta a forma atual de travamento usada).

Erros e fim de stream

```
int EOF
```

Macro com valor retornado por algumas funções, para indicar fim do arquivo ou erro.

```
#include <stdio.h>
int feof (FILE *stream)
```

Retorna valor não nulo se o stream chegou ao seu fim. Ver também `feof_unlocked`.

```
#include <stdio.h>
int ferror (FILE *stream)
```

Retorna um valor não nulo se ocorreu um erro em algum acesso anterior ao stream. Ver também `ferror_unlocked`.

Além de ajustar o indicador de erro do stream, as funções de acesso a streams também ajustam a variável `errno`.

Controle de buffers

A descarga (*flushing*) do buffer de entrada/saída de um stream ocorre de forma automática quando:

- Quando o buffer está cheio.

- Quando o stream é fechado.
- Quando o programa termina (fim de `main` ou chamada a `exit`)
- Quando um *newline* é escrito em um stream com buffer de linha.
- Quando ocorre uma leitura de dados do arquivo associado ao stream.

```
#include <stdio.h>
int fflush (FILE *stream)
```

Descarrega o buffer do stream indicado, ou de todos os streams abertos, caso `stream*` seja nulo. Ver também `fflush_unlocked`.

Após abrir um stream (mas antes de usá-lo), é possivel definir sua política de buffer.

```
#include <stdio.h>
int setvbuf (FILE *stream, char *buf, int mode, size_t size)
```

Especifica a política de buffer a ser usada no stream indicado. O valor mode pode ser `_IOFBF` (buffering completo), `_IOLBF` (buffering de linha) ou `_IONBF` (sem uso de buffer). Também é possível especificar um buffer externo (`buf`) e o tamanho do buffer a ser usado (`size`). Caso o buffer não seja indicado, um buffer será alocado e liberado automaticamente quando o stream for fechado.

```
int BUFSIZ
```

Macro que indica o tamanho mais adequado de buffer para o sistema. Depende de vários parâmetros do sistema, mas seu valor mínimo é 256.

From:
https://wiki.inf.ufpr.br/maziero/ - **Prof. Carlos Maziero**

Permanent link:
https://wiki.inf.ufpr.br/maziero/doku.php?id=pua:operacoes_usando_streams

Last update: **2016/03/22 10:49**