Standard Input and Output

Modelo de ficheiro

Visualização como sequência de linhas de texto.

```
bom dia\n
abcd 1234\n
PSC-19/20v\n
```

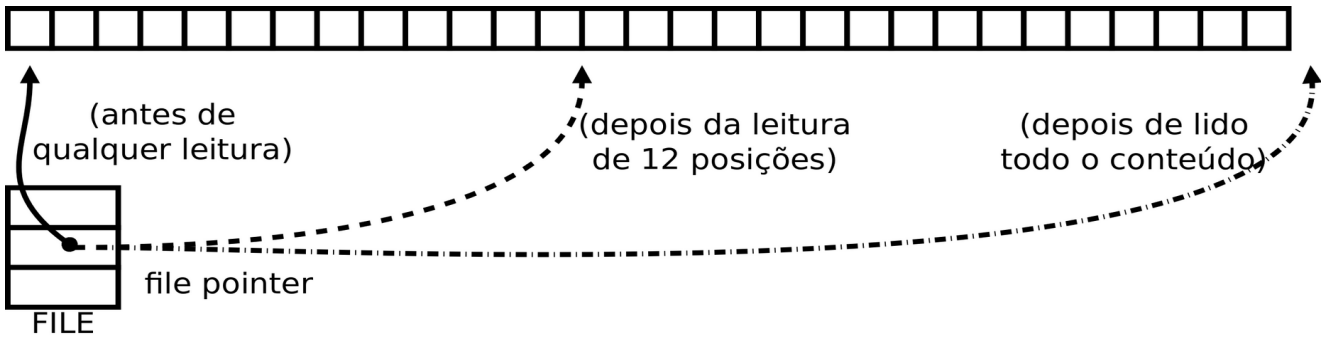
Visualização com sequência de caracteres.

'b'	'o'	'm'	' '	'd'	'i'	'a'	\n	'a'	'b'	'c'	'd'	' '	'1'	'2'	'3'	'4'	\n	'P'	'S'	'C'	'-'	'1'	'9'	'/'	'2'	'0'	'v'	\n

Visualização como sequência de bytes.

62	6f	6d	20	64	69	61	0a	61	62	63	64	20	31	32	33	34	0a	50	53	43	2d	31	39	2f	32	30	76	0a

Modelo de acesso a ficheiros



O conteúdo de um ficheiro pode ser interpretado em modo texto "**t**" ou em modo binário "**b**". Em modo binário o ficheiro é encarado como uma sequência de *bytes* indiferenciados. Em modo texto é encarado como uma sequência de linhas de texto, em que cada linha é formada por uma sequência de caracteres imprimíveis e terminadas por um marcador de fim de linha. No Unix o marcador é o carácter **\n**, no Windows o marcador é a sequência **\n\r**.

Todos os ficheiros, incluindo os que contêm texto, podem ser interpretados em binário.

As transferências de dados entre a memória e o ficheiro processam-se a partir de um indicador de posição associado ao ficheiro (*file pointer*). O indicador de posição avança automaticamente após cada operação de transferência de um número igual ao número de bytes transferidos.

Na operação de abertura, o indicador de posição é colocado no início (opções "r" e "w") ou para além

do fim (opção “a”).

Existem funções para modificar o indicador de posição de um ficheiro aberto.

Suportes físicos

Os dispositivos a considerar são: ecrã, teclado e ficheiros em disco.

Os dispositivos são representados por variáveis do tipo ponteiro para FILE (*file descriptor*). No início da execução de um programa, existem disponíveis três destas variáveis que representam o teclado e o ecrã.

```
FILE *stdin = &struct_stdin;
FILE *stdout = &struct_stdout;
FILE *stderr = &struct_stderr;
```

Modo texto

O texto é enviado ou recolhido dos dispositivos como uma sequência de linhas. As linhas são formadas por caracteres terminadas por um carácter marcador de fim de linha - **`\n`**.

No ecrã, a escrita do carácter **`\n`** provoca uma mudança de linha. No teclado, a tecla ENTER produz o carácter **`\n`**.

Output

A escrita é realizada no dispositivo indicado no parâmetro **`stream`**. Se for **`stdout`** será no ecrã.

A função **`fputs`** escreve a *string* indicada por **`s`** e acrescenta o carácter **`\n`**, o que provoca uma mudança de linha.

```
int fputs(const char *s, FILE *stream);
```

A funçãp **`printf`** escreve o texto indicada em **`format`**, substituindo os campos de formatação `%` pela representação dos valores passados nos restantes parâmetros.

```
int fprintf(FILE *stream, const char *format, ...);
```

A função **`fputch`** escreve um carácter.

```
int fputc(int c, FILE *stream);
```

As três funções seguintes equivalem às anteriores com **`stdout`** como argumento no parâmetro **`stream`**.

```
int puts(const char *s);
int printf(const char *format, ...);
int putchar(int c);
```

Para atualizar o ficheiro em disco com dados que estejam em *buffers* intermédios, resultantes de operações de escrita anteriores.

```
int  fflush(FILE * stream);
```

Especificações de conversão das funções printf:

%<flags><width><.precision><lenght><conversion>

flags – **+** (imprime o sinal), **-** (ajuste à esquerda), **space** (se não for um sinal), **0** (preencher com zeros), **#** (modo de escrita alternativa)

with – dimensão mínima do campo.

.precision – dimensão máxima para uma string ou casas decimais

length – h short, l long, L long double .

conversion – d, i, o, x, X, u, c, s, f, e, E, g, G, p, n, %

Input

E leitura é realizada do dispositivo indicado no parâmetro **stream**. Se for **stdin** será do teclado.

```
char *  fgets(char * s, int n, FILE * stream);
```

Lê uma linha de texto. Espera pelo terminador de linha (**\n**). O parâmetro **n** indica a dimensão de memória disponível para receber o texto.

```
int fscanf(FILE *stream, const char *format, ... );
```

Aplica a conversão de texto indicada em **format** à medida que lê os caracteres do dispositivo.

```
int fgetc(FILE *stream);
```

Ler um carácter.

As três funções seguintes equivalem às anteriores com **stdin** como argumento no parâmetro **stream**.

Notar que a função **gets** não possui o parâmetro que indica a dimensão do *buffer* disponível.

```
char * gets(char * s);
int scanf(const char *format, ... );
int getchar();
```

Especificações de conversão das funções scanf:

%*<width><.precision><lenght><conversion>

***** – interpreta o campo sem afetar a variável e salta para o próximo.

with – dimensão máxima do campo.

length – **h** short, **l** long, **L** long double .

conversion – d, i, o, x, u, c, s, f, e, g, p, n, [...], [^...], %

A definição da função getchar em stdio.h é equivalente a:

```
int getchar() { return fgetc(stdin); }
```

Redireccionamento

As variáveis **stdin** e **stdout** que representam normalmente o teclado e o ecrã podem representar ficheiros em disco.

Essa substituição pode ser feita na invocação do programa na linha de comando do interpretador de comandos (shell).

O sinal > substitui, em **stdout**, o *file descriptor* do ecrã pelo do ficheiro que se indicar.

O sinal < substitui, em **stdin**, o *file descriptor* do teclado pelo do ficheiro que se indicar.

Exemplos:

$ program < myfile o programa **program** ao ler de **stdin** está efectivamente a ler do ficheiro **myfile**.

$ program2 < text1 > text2 *o programa **program2** lê de **text1** e escreve em **text2**, ao usar, respectivamente, os ponteiros **stdin** e **stdout**.*

Ficheiros

Para que as funções anteriores acedam a um dado ficheiro em disco é necessário que o argumento passado no parâmetro **stream** esteja associado a esse ficheiro. Essa associação é realizada pela função **fopen**.

```
FILE *fopen(const char *filename, const char *mode);
```

Esta função procura, no sistema de ficheiros, pelo ficheiro indica na parâmetro **filename**, e cria uma representação interna desse ficheiro (**FILE**) .

Modos de abertura do ficheiro: "r" - só ler; "w" - só escrever; "a" - escrever no fim . Sinal + significa abrir em modo atualização; "r+" - ler e escrever; "w+" - ler e escrever começa vazio; "a+" escrever no fim; ler em qualquer lado.

Quando um ficheiro é aberto em modo de atualização deve-se usar **fflush**, **fseek**, **fsetpos** entre as escritas e as leituras para posicionar o indicador de posição.

A função **fclose** garante atualização do ficheiro no sistema de ficheiros com eventuais dados em trânsito, e elimina a representação interna do ficheiro. A partir desse momento o ficheiro deixa de estar acessível. Ao terminar um processo, o sistema operativo executa esta função para todos os ficheiros abertos.

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

A função remove serve para eliminar um ficheiro.

```
int remove(const char * filename);
```

A função rename permite alterar o nome de um ficheiro.

```
int rename(const char * oldname, const char * newname);
```

A função **tmpfile** cria um ficheiro temporário anónimo. A função **fclose** elimina-o do sistema de ficheiros.

```
FILE *tmpfile(void);
```

A função **tmpnam** cria um nome de ficheiro diferente de qualquer outro existente no sistema de ficheiros.

```
char *  tmpnam(char S[L_tmpnam]);
```

Posicionamento

As funções seguintes permitem manipular o indicador de posição.

```
int fseek(FILE *stream, long offset, int whence);
```

SEEK_SET - posiciona na posição indicada. **SEEK_CUR** - posiciona em relação à posição corrente; **SEEK_END** - posiciona em relação ao fim.

```
long ftell(FILE *stream);
int fsetpos(FILE *stream, const fpos_t *pos);
int fgetpos(FILE *restrict stream, fpos_t *restrict pos);

void rewind(FILE * stream);
```

Binário

Em formato binário, um ficheiro é encarado como uma sequência de bytes.

Output

Escrever no ficheiro representado por stream, uma sequência de items com dimensão nitens, tendo cada item a dimensão size em bytes. Esta operação transfere um bloco com a dimensão nitens * size byte para a memória, a partir da posição ptr.

```
size_t fwrite(const void * ptr, size_t size, size_t nitems, FILE * stream);

int fputc(int c, FILE *stream);
```

Atualiza o ficheiro com dados que estejam em *buffers* intermédios, resultantes de operações de escrita anteriores.

```
int  fflush(FILE * stream);
```

Input

Ler do ficheiro representado por stream, uma sequência de items com dimensão nitens, tendo cada item a dimensão size em bytes. Esta operação transfere um bloco com a dimensão nitens * size byte da memória, a partir da posição ptr.

```
size_t fread(void * ptr, size_t size, size_t nitems, FILE * stream);

int fgetc(FILE *stream);
```

Recua de uma posição o indicador de posição do ficheiro e insere o valor do parâmetro c nessa posição. Útil na construção de interpretadores.

```
int ungetc(int c, FILE * stream);
```

Erros

Em todas as funções é retornada a indicação sobre eventual ocorrência de erro. Essa indicação, do tipo "sim ou não", é indicada na forma de um ponteiro NULL ou de um valor negativo.

Essa informação pode ser obtida posteriormente com

```
int ferror(FILE * stream);
```

ou obtida uma indicação mais precisa através da variável **errno.**

A função **perror** imprime, em **stderr,** uma mensagem relativa ao erro registado em **errno.**

```
void perror ( const char * str);
```

```
void clearerr(FILE * stream);
```

Elimina a indicação de erro ocorrido em operação anterior.

```
int feof(FILE * stream);
```

Informa se foi tentado aceder para além do fim do ficheiro.

```
char * strerror(int errnum);
```

Traduz um código de erro para uma mensagem legível.

Referências

The C Programming Language

cplusplus.com - http://www.cplusplus.com/reference/clibrary/clibrary/cstdio/

Exercícios

1. Copiar ficheiros. Primeira versão - carácter a carácter; segunda versão - bloco a bloco.
2. Concatenar ficheiros.
3. Imprimir o conteúdo de um ficheiro em hexadecimal (tpo **hexdump**).