

Elementos de entrada e saída com arquivos

David Deharbe



Conceitos

- Por enquanto vimos como realizar
 - entradas (leitura): do teclado e
 - saídas (impressão): na tela.
- Em geral, pode-se usar rotinas muito similares para impressão e leitura em arquivos de computadores.
- Um arquivo de computador é uma entidade gerenciada pelo sistema operacional e que armazena informações.
 - há arquivos de diversas naturezas:
 - ✓ tipo de informação armazenado: texto simples, texto formatado, som, vídeo, programas, etc.
 - ✓ formato de representação da informação texto,

Conceitos

- Um arquivo de computador é uma entidade gerenciada pelo sistema operacional e que armazena informações.
- Um arquivo geralmente é uma sequência de bytes que codificam a informação representada.
- Existe arquivos de diversas naturezas:
 - tipo de informação armazenado: texto simples, texto formatado, gráficos, gráficos vetoriais, gráficos 3D, som, música, vídeo, programas, etc.
 - formato de representação da informação. Gráficos vetoriais: CDR, ODG, SVG, WMF etc.

Algumas operações básicas sobre arquivos

- **Criar um arquivo (para armazenar informação)**
- **Abrir um arquivo**
 - **leitura do conteúdo**
 - **atualização do conteúdo**
- **Fechar um arquivo**

Objetivos

- Manipular arquivos textos (ASCII)
 - um arquivo é uma sequência de caracteres.
- Criar arquivos
- Abrir arquivo (leitura, escrita)
- Ler de um arquivo
- Escrever em um arquivo
- Fechar arquivo

Princípios

- Toda a comunicação com arquivos é realizada através de canais de comunicação.
- Declarar uma variável para armazenar o canal de comunicação.
- Abrir o arquivo
 - o resultado da chamada à rotina é o canal de comunicação
 - atribuir à variável declarada
- Ler ou escrever no arquivo através do canal de comunicação.
- Fechar o canal de comunicação.

- *Obs.* Em geral a comunicação é buferizada: o comando printf não imprime diretamente no arquivo e sim numa zona intermediária chamada buffer. Quando o buffer é cheio, ou o canal fechado, o seu conteúdo é descarregado no arquivo.

Tipos

- Os tipos e sub-rotinas para acessar arquivos são declarados no arquivo cabeçalho `stdio.h`
`#include <stdio.h>`
- Canais de comunicação são registros do tipo `FILE`.
- Todas as rotinas de acesso a arquivos usam endereços de registros `FILE`.
`FILE * canal1;`
`FILE * canal2;`

```

typedef struct __sFILE {
    unsigned char *_p;          /* current position in (some) buffer */
    int _r;                    /* read space left for getc() */
    int _w;                    /* write space left for putc() */
    short _flags;              /* flags, below; this FILE is free if 0 */
    short _file;               /* fileno, if Unix descriptor, else -1 */
    struct __sbuf _bf;         /* the buffer (at least 1 byte, if !NULL) */
    int _lbfsz;                /* 0 or -_bf._size, for inline putc */

    /* operations */
    void *_cookie;             /* cookie passed to io functions */
    int (*_close)(void *);
    int (*_read)(void *, char *, int);
    fpos_t (*_seek)(void *, fpos_t, int);
    int (*_write)(void *, const char *, int);

    /* separate buffer for long sequences of ungetc() */
    struct __sbuf _ub;         /* ungetc buffer */
    struct __sFILEX *_extra;   /* additions to FILE to not break ABI */
    int _ur;                   /* saved _r when _r is counting ungetc data */

    /* tricks to meet minimum requirements even when malloc() fails */
    unsigned char _ubuf[3];    /* guarantee an ungetc() buffer */
    unsigned char _nbuf[1];    /* guarantee a getc() buffer */

    /* separate buffer for fgetln() when line crosses buffer boundary */
    struct __sbuf _lb;         /* buffer for fgetln() */

    /* Unix stdio files get aligned to block boundaries on fseek() */
    int _blksize;              /* stat.st_blksize (may be != _bf._size) */
    fpos_t _offset;            /* current lseek offset (see WARNING) */
} FILE;

```

```

typedef struct __sFILE {
    unsigned char *_p;          /* current position in (some) buffer */
    int _r;                    /* read space left for getc() */
    int _w;                    /* write space left for putc() */
    short _flags;              /* flags, below; this FILE is free if 0 */
    short _file;               /* fileno, if Unix descriptor, else -1 */
    struct __sbuf _bf;         /* the buffer (at least 1 byte, if !NULL) */
    int _lbfsize;              /* 0 or -_bf._size, for inline putc */

    /* operations */
    void *_cookie;             /* cookie passed to io functions */
    int (*_close)(void *);
    int (*_read)(void *, char *, int);
    fpos_t (*_seek)(void *, int, int);
    int (*_write)(void *, const char *, int);

    /* separate buffer for long sequences of ungetc() */
    struct __sbuf _ub;         /* ungetc buffer */
    struct __sFILEX *_extra;   /* additions to FILE to not break ABI */
    int _ur;                   /* saved _r when _r is counting ungetc data */

    /* tricks to meet minimum requirements even when malloc() fails */
    unsigned char _ubuf[3];    /* guarantee an ungetc() buffer */
    unsigned char _nbuf[1];    /* guarantee a getc() buffer */

    /* separate buffer for fgetln() when line crosses buffer boundary */
    struct __sbuf _lb;         /* buffer for fgetln() */

    /* Unix stdio files get aligned to block boundaries on fseek() */
    int _blksize;              /* stat.st_blksize (may be != _bf._size) */
    fpos_t _offset;            /* current lseek offset (see WARNING) */
} FILE;

```

Não precisa saber nada disso para trabalhar com arquivos!

Tipos

- Os tipos e sub-rotinas para acessar arquivos são declarados no arquivo cabeçalho `stdio.h`
`#include <stdio.h>`
- Canais de comunicação são registros do tipo `FILE`.
- Todas as rotinas de acesso a arquivos usam endereços de registros `FILE`.
`FILE * canal1;`
`FILE * canal2;`

Abertura de um arquivo

- A rotina para abrir um arquivo tem a seguinte interface:
FILE * fopen (char * name, char * mode);
- O parâmetro **name** é o nome do arquivo a abrir.
- O parâmetro **mode** indica se o arquivo vai ser acessado
 - "r" para leitura;
 - "w" para escrita no início do arquivo;
 - "a" para escrita no final do arquivo.
- O resultado é
 - o canal de comunicação com o arquivo
 - ou **NULL** caso não seja possível realizar esta operação

Exemplo

```
#include <stdio.h>

int main (void)
{
    FILE * fp;
    fp = fopen("afile.txt", "r");
    if (fp == NULL)
        printf("Arquivo \"afile.txt\" não encontrado.\n");
    return 0;
}
```

Exemplo

```
#include <stdio.h>

int main (void)
{
    FILE *fp = fopen("afile.txt", "r");
    if (fp == NULL)
        printf("Arquivo \"afile.txt\" não encontrado.\n");
    return 0;
}
```

localhost:aula27 david\$./ex1
Arquivo "afile.txt" não encontrado.
Arquivo "ex1.c" aberto em leitura.

Condições de falha de abertura

- Leitura de um arquivo inexistente
- Leitura de um arquivo protegido em leitura
- Escrita de um arquivo protegido em escrita
- Falha na alocação de memória para armazenar o canal de comunicação.
- O nome do arquivo é grande demais
- O programa já abriu o número máximo de canais autorizado
- etc.

Fechamento de um arquivo

- Abrir um arquivo aloca recursos:
 - memória usada para representar o canal de comunicação
 - usa a "quota" que o sistema operacional atribui a cada programa ou usuário.
- Quando não é mais necessário o acesso ao arquivo, esses recursos devem ser liberados.
- A rotina para abrir um arquivo tem a seguinte interface:

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

 - O parâmetro `canal` contém o endereço do registro armazenando o canal de comunicação.
 - O resultado é 0 se a operação foi bem-sucedida, EOF caso contrário.

Exemplo

```
#include <stdio.h>
int main (void)
{
    FILE * fp1;
    FILE * fp2;
    fp1 = fopen("afile.txt", "r");
    fp2 = fopen("ex1.c", "r");
    if (fp1 == NULL) printf("Arquivo \"afile.txt\" não encontrado.\n");
    if (fp2 != NULL) printf("Arquivo \"ex1.c\" aberto em leitura.\n");
    if (fclose(fp1) == 0) printf("Primeiro canal fechado.\n");
    else printf("Não foi possível fechar o primeiro canal.\n");
    if (fclose(fp2) == 0) printf("Segundo canal fechado.\n");
    else printf("Não foi possível fechar o segundo canal.\n");
    return 0;
}
```

Exemplo

```
localhost:aula27 david$ ./ex2
Arquivo "afile.txt" não encontrado.
Arquivo "ex1.c" aberto em leitura.
Não foi possível fechar o primeiro canal.
Segundo canal fechado.
```

```
#include <stdio.h>
int main (void)
{
    FILE * fp1;
    FILE * fp2;
    fp1 = fopen("afile.txt", "r");
    fp2 = fopen("ex1.c", "r");
    if (fp1 == NULL) printf("Arquivo \"afile.txt\" não encontrado.\n");
    if (fp2 != NULL) printf("Arquivo \"ex1.c\" aberto em leitura.\n");
    if (fclose(fp1) == 0) printf("Primeiro canal fechado.\n");
    else printf("Não foi possível fechar o primeiro canal.\n");
    if (fclose(fp2) == 0) printf("Segundo canal fechado.\n");
    else printf("Não foi possível fechar o segundo canal.\n");
    return 0;
}
```

Saída formatada

- Para impressão formatada em arquivo, a biblioteca padrão fornece a rotina **fprintf**.
- O primeiro parâmetro é o endereço de um canal de comunicação (aberto para escrita).
- **int fprintf(FILE * canal, char * formato, ...);**
- O retorno do comando é
 - o número de caracteres (bytes) impressos se a impressão foi bem sucedida,
 - um número negativo, caso contrário.

Exemplo

```
#include <stdio.h>

int main (void)
{
    FILE * canal;

    canal = fopen("afile2.txt", "w");
    if (canal != NULL)
    {
        printf("Arquivo \"afile2.txt\" aberto em escrita.\n");
        fprintf(canal, "Hello world!\n");
        fprintf(canal, "%i\n", 40+2);
        if (fclose(canal) == 0)
            printf("Arquivo \"afile2.txt\" fechado.\n");
    }
    return 0;
}
```

Exemplo

```
#include <stdio.h>

int main (void)
{
    FILE * canal;

    canal = fopen("afile2.txt", "w");
    if (canal == NULL)
    {
        printf("Arquivo \"afile2.txt\" aberto em escrita.\n");
        fclose(canal);
        printf("Arquivo \"afile2.txt\" fechado.\n");
    }
    canal = fopen("afile2.txt", "w");
    fprintf(canal, "Hello world!\n");
    fclose(canal);
    printf("Arquivo \"afile2.txt\" fechado.\n");
}

return 0;
}
```

```
localhost:aula27 david$ cat afile2.txt
cat: afile2.txt: No such file or directory
localhost:aula27 david$ ./ex3
Arquivo "afile2.txt" aberto em escrita.
Arquivo "afile2.txt" fechado.
localhost:aula27 david$ cat afile2.txt
Hello world!
localhost:aula27 david$
```

Princípios

- Em geral, a impressão é buferizada:
 - o comando `printf` não imprime diretamente no arquivo e sim numa zona intermediária chamada *buffer* (um arranjo de bytes).
 - Quando o buffer é cheio, ou o canal fechado, o seu conteúdo é descarregado no arquivo.

Leitura formatada

- Para impressão formatada em arquivo, a biblioteca padrão fornece a rotina **fscanf**.
- O segundo parâmetro é o endereço de um canal de comunicação (aberto para escrita).
- **int fscanf(FILE * canal, char * formato, ...);**
- O retorno do comando é
 - o número de valores que foram lidos e atribuídos na leitura, se a leitura foi bem sucedida,
 - **EOF**, caso contrário.

Exemplo

```
#include <stdio.h>
int main (void)
{
    char arquivo [] = "contas.txt";
    FILE * canal;
    int val;
    int r = 1;

    canal = fopen(arquivo, "r");
    if (canal == NULL)
        val = 0;
    else
    {
        fscanf(canal, "%i", &val);
        fclose(canal);
    }
}
```

```
if (r == 1)
{
    canal = fopen(arquivo, "w");
    if (canal != NULL)
    {
        fprintf(canal, "%i\n", val+1);
        fclose(canal);
    }
}
return 0;
}
```

Exemplo

```
#include <stdio.h>
int main (void)
{
    char arquivo [] = "contas.txt";
    FILE * canal;
    int val;
    int r = 1;

    canal = fopen(arquivo, "w");
    if (canal != NULL)
        val = 1;
    else
    {
        fscanf(canal, "%i", &val);
        fclose(canal);
    }
}
```

```
localhost:aula27 david$ ./ex4
localhost:aula27 david$ cat contas.txt
1
localhost:aula27 david$ ./ex4; ./ex4; ex4
localhost:aula27 david$ cat contas.txt
4
```

```
if (r == 1)
```

```
"w");
```

```
, val+1);
```

```
    }
}
return 0;
}
```

Leitura e escrita por caracter

- Existe outras rotinas de impressão e leitura.
- Por exemplo, rotinas de escrita e leitura caracter por caracter:
 - `int fputc(int c, FILE * canal);`
 - `int fgetc(FILE * canal);`
- O primeiro argumento de `fputc` é convertido para um `unsigned char`
- O resultado de `fgetc` é o próximo byte no arquivo do canal de comunicação, ou `EOF` caso não tenha mais.

Exemplo

```
#include <stdio.h>
int main (void)
{
    char arquivo [] = "contas";
    FILE * canal;
    int val, r = 1;
    canal = fopen(arquivo, "r");
    if (canal == NULL)
        val = 0;
    else
    {
        val = fgetc(canal);
        fclose(canal);
    }
}
```

```
if (r == 1)
{
    canal = fopen(arquivo, "w");
    if (canal != NULL)
    {
        fputc(val+1, canal);
        fclose(canal);
    }
}
return 0;
}
```

Exemplo

```
#include <stdio.h>
int main (void)
{
    char arquivo [] = "contas";
    FILE * canal;
    int val, r = 1;
    canal = fopen(arquivo, "r");
    if (canal != NULL)
        val = fgetc(canal);
    else
    {
        val = fgetc(canal);
        fclose(canal);
    }
}
```

Atenção: o conteúdo do arquivo não é mais texto e sim a codificação binária de um unsigned char.

```
if (r == 1)
{
    canal = fopen(arquivo, "w");
    if (canal != NULL)
        fputc(val+1, canal);
    fclose(canal);
}
return 0;
}
```

Canais de comunicação padrão

- Adicionalmente, os programas C tem três canais de comunicação sempre abertos
 - o canal de saída padrão `stdout` que o SO conecta por default à tela
 - o canal de entrada padrão `stdin` que o SO conecta por default ao teclado
 - o canal de saída de erros padrão `stderr` que o SO conecta por default à tela.
- `printf("oi");` nada mais é que `printf(stdout, "oi");`
- o usuário pode redirecionar esses canais de comunicação usando comandos do terminal:
 - `./programa < entrada.txt > saida.txt`

Conclusões

- A entrada e a saída em arquivos não difere muito daquela que vimos anteriormente.
- A principal diferença reside na necessidade de abrir e fechar canais de comunicação com os arquivos.
- A biblioteca padrão oferece outras formas de acessar arquivos.