Prof. Neucimar J. Leite

Este resumo contém alguns exemplos e comentários sobre algumas das principais funções da linguagem C para manipulação de arquivos de dados.

1 Funções de alto nível

Estas funções de I/O são definidas no arquivo *header* `stdio.h` e não estão diretamente relacionadas ao sistema operacional. Elas manipulam arquivos de dados padrões que podem ser de dois tipos: *formatados* (dados formados por caracteres consecutivos que podem ser interpretados como dados individuais ou componentes de strings ou números), e *não-formatados* (dados organizados em blocos de bytes contíguos de informação).

Exemplo 1 *Estrutura básica de um arquivo para gravação:*

```
#include <stdio.h>
main() {
FILE *fpt ;
fpt = fopen("mc326.dat", "w") ;
...
fclose(fpt) ;
}
```

`FILE` estabelece um *buffer* onde o conteúdo do arquivo de dados será armazenado no computador. `*fpt` é o apontador para este *buffer*.

`fopen` abre o arquivo de dados `mc326.dat` para escrita (`"w"`). Estas possibilidades são:

Constante	Significado
"r"	abre um arquivo existente somente para leitura
"w"	abre um novo arquivo somente para gravação. Apaga arquivo especificado caso o mesmo já exista
"a"	abre um arquivo para adição de informações no final deste
"r+"	abre um arquivo existente para leitura e escrita
"w+"	abre um **novo** arquivo para leitura e escrita
"a+"	abre um arquivo existente tanto para leitura quanto para adição de informações no seu final. O arquivo será criado caso o mesmo não existe.

Exemplo 2 *Acesso a arquivos com testes condicionais:*

```
#include <stdio.h>
#define NULL 0
main() {
  FILE *fpt ;
  if((fpt = fopen("mc326.dat", "r")) == NULL)
    printf("\nErro - O arquivo nao pode ser aberto\n") ;
  else {
    ...
  }
}
```

No exemplo acima, uma mensagem de erro será emitida caso o arquivo `mc326.dat`, a ser aberto para leitura, não exista.

1.1 Leitura e escrita em arquivos de dados

Um arquivo de dados dever ser criado antes de ser processado. Para tanto, pode-se, por exemplo, empregar um editor de texto ou implementar um programa que leia dados do teclado e grave-os no arquivo de dados que está sendo criado.

Exemplo 3 *Conversão de minúsculo para maiúsculo de uma linha de texto lida de um teclado (emprego das funções* **putc**, **getchar** *e* **toupper***):*

```
#include <stdio.h>
main() {
FILE *fpt ;
char c ;
fpt = fopen("mc326.dat", "w") ;
/* ler os caracteres e converte-los gravando no arquivo de dados */
printf("digite sequencia de caracteres e tecle ENTER:\n\n") ;
do
    putc(toupper(c = getchar()), fpt) ;
while (c != '\n') ;
fclose(fpt) ;
}
```

Exemplo 4 *Leitura de uma linha de um arquivo de dados e exibição da mesma na tela (emprego das funções* **putc** *e* **getchar***).*

```
#include <stdio.h>
#define NULL 0
main() {
FILE *fpt ;
char c ;
if ((fpt = fopen("mc326.dat", "r")) == NULL)
    printf("\nErro - O arquivo nao pode ser aberto\n") ;
else
    do
       putchar(c= getc(fpt)) ;
    while(c != '\n') ;
fclose(fpt) ;
}
```

1.2 Arquivo de dados não-formatados

As funções `fread(...)` e `fwrite(...)` ilustradas a seguir manipulam arquivos não-formatados, gerando arquivos binários quando do armazenamento de blocos de dados representando, por exemplo, uma estrutura complexa ou uma matriz. Ao invés de ler ou gravar os seus componentes individualmente, estas funções possibilitam a gravação ou leitura destes blocos de uma só vez.

Exemplo 5 *Gravação e leitura de um arquivo de dados não-formatado contendo registros.*

```
#include <stdio.h>
#include <string.h>
#define OK 1
typedef struct {
  int dia ;
  int mes ;
  int ano ;
} data ;
typedef struct {
  char nome[80] ;
  int num_conta ;
  data ultpag ;
} registro ;
FILE *fpt ;
main() {
int flag = OK ;
registro cliente ;
registro ler_tela(registro cliente) ;
void mostra_tela(registro cliente) ;
```

```
fpt = fopen("dados.bin", "w") ;
/* inicializar valores */
printf("\ndigite a data atual\n") ;
scanf("%d%d%d", &cliente.ultpag.dia, &cliente.ultpag.mes, &cliente.ultpag.ano) ;
while (flag) {
   /* ler dados do cliente */
   cliente = ler_tela(cliente) ;
   if (strcmp(cliente.nome, "FIM") == 0)
     break ;
   /* gravar dados no arquivo dados.bin */
   fwrite( &cliente, sizeof(registro), 1, fpt) ; /* 1 indica transferência de um
único bloco de dados */
 }
fclose(fpt) ;
fpt = fopen("dados.bin", "r") ;
fread(&cliente, sizeof(registro), 1, fpt) ;
 while(!feof(fpt)) {
   mostra_tela(cliente) ;
   fread(&cliente, sizeof(registro), 1, fpt) ;
  }
fclose(fpt) ;
}

registro ler_tela(registro cliente) {     /* ler dados do cleinte */
printf("Nome (digite FIM para terminar):  ") ;
scanf(" %[^\n]", cliente.nome) ;
  if (strcmp(cliente.nome, "FIM") != 0) {
   printf("Numero da Conta:  ") ;
   scanf("%d", cliente.num_conta) ;
  }
return(cliente) ;
}
void mostra_tela(registro cliente) { /* mostra conteúdo do arquivo binário na tela*/
printf("Data:%d%d%d\n", cliente.ultpag.dia, cliente.ultpag.mes, cliente.ultpag.ano);
printf("Nome cliente:  %s\n", cliente.nome) ;
printf("Conta:  %d\n", cliente.num_conta) ;
}
```

2 Funções de baixo nível

As funções de baixo nível estão mais diretamente relacionadas ao sistema operacional empregado do que as de alto nível vistas anteriormente.

2.1 A função `creat`

Esta função permite a criação de um novo arquivo binário.

`int creat(char *nome, int permissão)`

retorna um descritor de arquivo (um inteiro), se for possível a sua criação, e -1 em caso contrário. Um arquivo já existente aberto com esta função terá o seu conteúdo apagado.

Para um novo arquivo, a `permissão` corresponde àqueles nove bits de proteção que controlam a permissão de leitura, gravação e execução para o dono do arquivo, grupo e para todos os outros usuários. Por exemplo, o número octal de três dígitos 0755 especifica permissão de leitura, gravação e execução para o dono, de leitura e execução para o grupo e para os demais usuários do sistema.

No caso de um arquivo existente, o tratamento depende do seu parâmetro de permissão. Se existe permissão de gravação, então o arquivo é reinicializado com seu conteúdo original perdido. Em caso contrário, o arquivo permanece inalterado e a função retorna o valor -1.

Exemplo 6 *Criação de um arquivo com a função* `creat`*.*

```
#include <stdio.h>
main () {
int arquivo ;
if((arquivo = creat("exemplo.bin", 0644)) == -1) ;
   printf("\nArquivo nao pode ser criado\n") ;
else
   printf("\nArquivo criado com sucesso\n") ;
}
```

2.2 A função `open`

Esta função permite a abertura de um arquivo existente e é semelhante à função `fopen`, exceto que ao invés de retornar um apontador, ela retorna um descritor de arquivo na forma de um inteiro.

`int open(char *nom, int acesso) ;`

`nom` é o nome do arquivo e `acesso` é o tipo da operação realizada. Algumas destas operações, representadas pelas constantes descritas no arquivo *header* FCNTL.H, são as seguintes:

Constante	Tipo de acesso
O_RDONLY	leitura
O_WRONLY	gravação
O_RDWR	leitura e gravação
O_APPEND	gravação em fim de arquivo
O_CREAT	criação de um arquivo inexistente
O_TRUNC	reinicialização do arquivo

Os valores acima podem ser combinados através do operador | . A função retorna o valor -1 caso o arquivo não possa ser aberto adequadamente.

Uma outra forma de declaração da função `open` é a seguinte:

```
int open(char *nom, int acesso, int permissão) ;
```

em que `permissão`, como visto anteriormente, corresponde ao tipo de permissão atribuído a um arquivo a ser criado. Naturalmente, este parâmetro só é útil quando se deseja abrir um arquivo que não existe ainda (neste caso, o parâmetro `acesso` deve ser do tipo O_CREAT).

Exemplo 7 *Criação de arquivos com a função* `open`.

```
#include <stdio.h>
#include <fcntl.h>
main () {
int arquivo1, arquivo2 ;
arquivo1 = open("exemplo.bin", O_RDONLY) ;
/* abrir o arquivo para leitura apenas ou criá-lo de acordo
com o parâmetro permissão */
arquivo2 = open("exemplo.txt", O_WRONLY | O_CREAT, 0777) ;
...
close(arquivo1) ;
close(arquivo2) ;
}
```

Como ilustrado acima, após o processamento de um arquivo, o mesmo deve ser fechado com a função:

```
int close(int descritor_arquivo) ;
```

2.3 Leitura de um arquivo: a função `read`

Para as operações de I/O de baixo nível, um arquivo é constituído de uma seqüência de bytes. Sua leitura consiste, assim, do acesso a um número especificado destes bytes.

A função `read` que carrega para a memória do computador o número de bytes do arquivo de dados é declarada como:

```
int read(int descritor_arquivo, void *buffer, unsigned numero_bytes) ;
```

`buffer` representa o endereço da área da memória que conterá os bytes lidos. `numero_bytes` corresponde ao número destes bytes.

Esta função retorna o número de bytes lidos efetivamente. Se este número for inferior a `numero_bytes`, então o arquivo encontra-se no seu fim. O valor -1 é retornado em caso de problemas na abertura do arquivo.

O ponteiro correspondente à posição atual no arquivo é atualizado a cada etapa de leitura, de acordo com o número de bytes lidos.

Exemplo 8 *Leitura de blocos de bytes de um arquivo de dados.*

```
#include <stdio.h>
#include <fcntl.h>
main() {
int arquivo1, tamanho ;
char *buffer ;
scanf("%d", &tamanho) ;
 if((arquivo1 = open("dados.bin", O_RDONLY)) != -1)
 {
    buffer = malloc(tamanho) ;
    read(arquivo1, buffer, tamanho) ;
    /* ...  */
 }
 else printf("\nProblemas na abertura do arquivo\n") ;
close(arquivo1) ;
}
```

2.4 Gravação em um arquivo: a função `write`

A função de gravação de baixo nível envia um certo número de bytes da memória do computador para um arquivo. Ela é especificada como:

```
int write(int descritor_arquivo, void *buffer, unsigned numero_bytes) ;
```

O valor retornado por esta função corresponde ao número de bytes efetivamente gravados no arquivo. O valor -1 é retornado em caso de problemas na gravação.

Os bytes são escritos a partir da posição corrente que é atualizada de acordo com o número de bytes transferidos.

Exemplo 9 *Gravação de blocos de bytes em um arquivo*

```
#include <stdio.h>
#include <fcntl.h>
main() {
int arquivo1, tamanho ;
char *buffer ;
scanf("%d", &tamanho) ;
if((arquivo1 = open("dados.bin", O_APPEND)) != -1)
  {
     buffer = malloc(tamanho) ;
     /* ...  */
     write(arquivo1, buffer, tamanho) ;
     close(arquivo1) ;
  }
else printf("\nProblemas na abertura do arquivo\n") ;
}
```

Exemplo 10 *Programa que copia o conteúdo de um arquivo em outro qualquer no disco.*

```
#include <stdio.h>
#include <fcntl.h>
#define NULL 0
#define TAMBUF 512
#define MODOP 0644
main(argc, argv)
int argc ;
char *argv[] ;
{
int f1, f2, n ;
char buf[TAMBUF] ;
if(argc != 3) printf("\nSintaxe:  copiar de -- para\n") ;
if((f1 = open(argv[1],0)) == -1) printf("\nerro:  nao pode abrir%s\n", argv[1])
;
if ((f2 = creat(argv[2], MODOP)) == -1) printf("\nerro:  nao pode criar
while ((n = read(f1, buf, TAMBUF)) > 0)
    if(write(f2,buf,n) != n) printf("\nerro:  erro de gravacao\n") ;
}
```

2.5 A função `eof`

A função

`int eof(int descritor_arquivo) ;`

permite verificar a ocorrência de um final de arquivo. O valor retornado é o seguinte:

1	se final de arquivo
0	em caso contrário
-1	em caso de erro

2.6 Acesso direto: `lseek`

As funções acima modificam automaticamente a posição corrente de leitura e gravação seqüenciais num arquivo de dados. Um acesso direto aos dados deste arquivo, sem leitura ou gravação, pode ser realizado pela função

`long lseek(int descritor_numero, long numero_bytes, int origem) ;`

`numero_bytes` indica o deslocamento relativo em bytes e `origem` especifica a origem deste deslocamento. Esta constante pode assumir os seguintes valores:

Constante	Deslocamento relativo
SEEK_SET ou 0	a partir do início do arquivo
SEEK_CUR ou 1	a partir da posição atual
SEEK_END ou 2	a partir do final do arquivo

O valor retornado por esta função é a nova posição atual que também pode ser determinada por

```
long tell(int descritor_numero) ;
```

Exemplo 11 *Emprego da função* `lseek`

```
#include <stdio.h>
#include <fcntl.h>
main() {
int arq1, tamanho ;
int nBytes = 256 ;
int teste ;
 if((arq1 = open("dados.bin", O_RDWR)) == -1)
   printf("\nerro no acesso ao arquivo\n") ;
 else
 {
    teste = lseek(arq1, nBytes,SEEK_SET) ; /* avanco de 256 bytes no arquivo */
    /* imprime teste = 256 e tell = 256 */
    printf("\nteste = %d, tell = %d\n", teste, tell(arq1)) ;
    /* ...  */
    close(arq1) ;
 }
}
```

2.7 Supressão de um arquivo

A função empregada para apagar um determinado arquivo de um diretório é a seguinte:

```
int unlink(char *nom) ;
```

Esta função retorna o valor 0, se devidamente executada , ou -1 em caso de erro.

Exemplo 12 *Supressão de um arquivo de dados.*

```
#include <stdio.h>
main() {
char arquivo[30] ;
puts("\nDigite arquivo a ser apagado\n");
gets(arquivo) ;
if(unlink(arquivo))
   puts("\nProblemas!!!\n");
else
   puts("\nArquivo apagado\n") ;
}
```