

Entrada e Saída com Arquivos

PE-09 – v1.0

Prof. Paulo Joia Filho

- 1 Introdução
- 2 Entrada e Saída com Arquivos
- 3 Conclusões

- 1 **Introdução**
 - Objetivos
- 2 Entrada e Saída com Arquivos
- 3 Conclusões

Objetivos

Os principais objetivos desta aula são:

- Entender o conceito de arquivos em C.
- Apresentar as principais funções de manipulação de arquivos.
- Explorar alguns exemplos de manipulação de arquivos.

Objetivos

Os principais objetivos desta aula são:

- Entender o conceito de arquivos em C.
- Apresentar as principais funções de manipulação de arquivos.
- Explorar alguns exemplos de manipulação de arquivos.

Objetivos

Os principais objetivos desta aula são:

- Entender o conceito de arquivos em C.
- Apresentar as principais funções de manipulação de arquivos.
- Explorar alguns exemplos de manipulação de arquivos.

1 Introdução

2 Entrada e Saída com Arquivos

- Funções de E/S
- Abertura de um arquivo
- Fechamento de arquivo
- Retrocedendo ao início de um arquivo
- Leitura e escrita em arquivos
- E/S formatada

3 Conclusões

Arquivos

Funções de E/S

- As funções que compõem o sistema de entrada/saída do C ANSI podem ser agrupadas em duas principais categorias:
 - E/S pelo **console**
 - E/S com **arquivo**
- A linguagem C não contém nenhum comando de E/S. Ao contrário, todas as operações de E/S ocorrem mediante chamadas de funções da biblioteca C padrão (**`stdio.h`**).

Arquivos

Stream × Arquivo

- Antes de nos aprofundarmos nas funções de E/S, é importante diferenciar *stream* e *arquivo*.
- O sistema de E/S de C fornece uma interface consistente ao programador, independente do dispositivo real que é acessado.
- Essa abstração é chamada *stream* e o dispositivo real é chamado *arquivo*.
- Existem dois tipos principais de *stream*:
 - *stream* de texto
 - *stream* binário

Arquivos

Stream de texto

- Basicamente, um *stream* de texto é uma sequência de caracteres.
- Normalmente, um *stream* de texto é organizado em linhas terminadas por um caractere de nova linha.

Exemplo:

todo.txt

1. Buscar jornal
2. Ir ao supermercado
3. Marcar dentista

Arquivos

Stream de texto

- É importante mencionar que a representação de **nova linha** é diferente em sistemas operacionais.

Por exemplo:

- Em sistemas operacionais baseados no UNIX é representado por:
 - `\n` - ASCII 10
 - isto é, somente a quebra de linha (*line feed*)
- No Windows é representado por:
 - `\r\n` - ASCII 13 e 10
 - isto é, retorno de carro (*carrier return*) seguido de quebra de linha (*line feed*)

Arquivos

Stream de texto

Até agora, temos trabalhado com as *streams* padrões:

- **`stdin`** → para entrada de dados.
- **`stdout`** → para saída de dados.

Arquivos

Stream binário

- Basicamente, um *stream* binário é uma sequência de bytes diretamente correspondida com o dispositivo externo, isto é, sem tradução dos caracteres.

Exemplo:

todo.bin

```
100101010100101010111010101001010101110101010010101  
101010010101010010101011101010100101010111010101001  
111110101001010101110101010010101011101010100101011  
101001010101110101010010101011101010100101011001010
```

Arquivos

- Arquivo pode ser qualquer dispositivo de E/S. Por exemplo:
 - impressora, teclado, disquete, disco rígido, etc.
- Os aplicativos C ANSI manipulam **arquivos** através de *streams*.
- Para manipular o **stream** de um **arquivo**, é necessário que o arquivo seja aberto.

👉 **Importante:**

Após manipular o arquivo, esse **deve** ser fechado.

Arquivos

Operações comuns em arquivos:

- abrir e fechar
- apagar
- ler e escrever caracteres ou dados binários
- verificar se chegou ao fim
- posicionar

Observações:

- Obviamente algumas dessas funções não se aplicam a todos os tipos de dispositivos. Por exemplo:
 - não é possível reposicionar uma impressora no início
 - um arquivo em disco permite acesso aleatório ao contrário de um teclado.

Arquivos

Fechar arquivos

- **Sempre** após manipular um arquivo, ele **deve** ser fechado.
- Isso faz com que os *buffers* sejam descarregados para o dispositivo externo, o que ocorre automaticamente quando o aplicativo é encerrado.
- Entretanto, caso o aplicativo encerre de forma não esperada, o *buffer* do arquivo pode não ser descarregado gerando uma inconsistência.

Arquivos

Funções de Entrada e Saída (E/S)

- As funções de entrada e saída encontram-se na biblioteca `stdio.h`.
- Por padrão, a maioria das funções iniciam-se com o prefixo “**f**”.
- No próximo slide veremos as principais funções da linguagem C para manipulação de arquivos.

Arquivos: Funções de Entrada e Saída (E/S)

Função	Descrição
<code>fopen()</code>	Abre um arquivo
<code>fclose()</code>	Fecha um arquivo
<code>fputc()</code> , <code>putc()</code>	Escreve um caractere em um arquivo
<code>fgetc()</code> , <code>getc()</code>	Lê um caractere de um arquivo
<code>fprintf()</code>	Equivalente a <code>printf()</code> , utilizando <i>stream</i>
<code>fscanf()</code>	Equivalente a <code>scanf()</code> , utilizando <i>stream</i>
<code>fgets()</code>	Lê uma cadeia de caracteres (<i>string</i>)
<code>fputs()</code>	Escreve uma cadeia de caracteres (<i>string</i>)
<code>fseek()</code>	Posiciona o arquivo em um ponto específico
<code>rewind()</code>	Posiciona o arquivo no início
<code>feof()</code>	Verifica se chegou ao final do arquivo
<code>ferror()</code>	Verifica se ocorreu um erro
<code>fflush()</code>	Descarrega o <i>buffer</i> associado ao arquivo
<code>fread()</code>	Leitura binária de dados
<code>fwrite()</code>	Escrita binária de dados
<code>remove()</code>	Remove um arquivo

Arquivos

Ponteiro de arquivo

- Para ter acesso aos dados em um arquivo é necessário definir um ponteiro do tipo especial **FILE** – definido na biblioteca **stdio.h**.
- Esse ponteiro permite que o aplicativo tenha acesso a uma estrutura que armazena informações importantes sobre o arquivo.
- Declaração:

```
FILE * fp;
```

fp → ponteiro utilizado para operações no arquivo.

Arquivos

Abertura de um arquivo

- A primeira operação a ser realizada em um arquivo é a sua abertura. Essa operação associa um *stream* (fluxo de dados) ao arquivo.
- Um arquivo pode ser aberto para diversas finalidades:
 - leitura, escrita, leitura/escrita, adição (*append*) de texto etc

Arquivos

Abertura de um arquivo

- A função **fopen** é utilizada para abrir o arquivo.

- Protótipo:

```
FILE * fopen (const char *arq, const char *modo);
```

- **arq** → nome do arquivo (caminho relativo ou absoluto).
 - **modo** → determina como o arquivo será aberto.
- Assim como na função **malloc**, o ponteiro retornado não deve ser modificado. Além disso, caso não consiga abrir o arquivo, é retornado um ponteiro nulo (**NULL**).
- No próximo slide, serão vistos os diversos modos de abertura de arquivos.

Arquivos: Modos de abertura

Modo	Descrição
r	Abre um arquivo texto para leitura
w	Cria um arquivo texto para escrita
a	Adiciona texto ao fim de um arquivo texto
rb	Abre um arquivo binário para leitura
wb	Abre a um arquivo binário para escrita
ab	Anexa a um arquivo binário
r+	Abre um arquivo texto para leitura/escrita
w+	Cria um arquivo texto para leitura/escrita
a+	Anexa ou cria um arquivo texto para leitura/escrita
r+b	Abre um arquivo binário para leitura/escrita
w+b	Cria um arquivo binário para leitura/escrita
a+b	Anexa a um arquivo binário para leitura/escrita

Importante:

Se um arquivo **existente** for aberto para **escrita**, será apagado e um novo será criado. Se for aberto para **leitura/escrita**, ele não será apagado. E, caso ele não exista, então será criado.

Arquivos

Abertura de um arquivo

Exemplo:

```
FILE *fp; char ch;  
fp = fopen("arquivo.txt", "r");  
if (!fp) {  
    printf("O arquivo nao pode ser aberto");  
    return -3;  
}
```

Importante:

Usualmente, testa-se o retorno da função **fopen** para verificar se houve erro na abertura de um arquivo, tal como arquivo não existente, arquivo sem permissão de leitura ou escrita etc.

Arquivos

Fechamento de um arquivo

- A função **fclose** fecha *stream* aberto pela função **fopen**.
- Protótipo:

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

- **fp** → ponteiro para o arquivo a ser fechado.

Arquivos

Fechamento de um arquivo

- Caso a função retorne 0 indica que o arquivo foi fechado com sucesso. Qualquer outro valor indica que ocorreu algum erro.
- A chamada da função **fclose** implica na escrita de qualquer dado que ainda não tenha sido **efetivamente** escrito no arquivo.
 - Isso é um ponto importante a mencionar, pois no UNIX, por exemplo, uma operação de escrita em arquivo não ocorre imediatamente à emissão da ordem de escrita.
 - Normalmente, o sistema operacional executa a escrita no momento que achar mais conveniente.

Arquivos

Verificando o final de um Arquivo

- A função **feof** indica se chegou ao final do arquivo.

- Protótipo:

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

- **fp** → ponteiro para o arquivo a ser verificado se atingiu o final.
- O uso mais comum de **feof** é em um laço de repetição, por exemplo, para a leitura completa de um arquivo.

Exemplo:

```
while ( !feof(fp) ) { /* Poderia ser while( feof(fp) == 0 ) */  
    ch = fgetc(fp); /* A função fgetc ou getc lê um caractere */  
    ...  
}
```

Arquivos

Retrocedendo um arquivo ao início

- A função **rewind** reposiciona o arquivo em seu início.
 - Operação semelhante a de rebobinar um VHS.

- Protótipo:

```
void rewind (FILE *fp);
```

- **fp** → ponteiro para o arquivo a ser retrocedido.
- É importante observar o modo de abertura do arquivo.
 - Por exemplo, um arquivo aberto somente para escrita, se retrocedido ao início, não irá permitir outra operação que não seja escrita.

Arquivos

Leitura e escrita de caracteres

- As operações mais simples em arquivos são leitura e escrita de caracteres.
- Para efetuar a **leitura** do caractere de um arquivo pode-se utilizar as funções **getc** e **fgetc** que são equivalentes.

- Protótipo:

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

- **fp** → ponteiro para o arquivo cujo caractere será lido.
- A função retorna o caractere como um **int** (padrão).
- Caso tente-se ler de um arquivo cujo final já foi alcançado, a função devolve o código **EOF** (-1).

Arquivos

Leitura e escrita de caracteres

- Para efetuar a **escrita** de um caractere em um arquivo pode-se utilizar as funções **putc** e **fputc** que são equivalentes.
- Protótipo:

```
int fputc (int ch, FILE *fp);
```

- **fp** → ponteiro para o arquivo cujo caractere será escrito.
- **ch** → caractere a ser escrito.

Arquivos

[exibe_conteudo.c]

Exemplo 1 (Exibindo o conteúdo de um arquivo)

```
1  #include <stdio.h>

3  int main() {
    FILE *fp; char ch;

5

    fp = fopen("lista_compras.csv", "r");
7    if (!fp) {
        printf("O arquivo não pode ser aberto.\n");
9        return -3;
    }

11

    while (!feof(fp)) {
13        ch = fgetc(fp);
        if (ch != EOF) {
15            printf("%c", ch);
        }

17    }

19    fclose(fp);
    return 0;

21 }
```

O que será impresso?

Arquivos

[exibe_conteudo.c]

Exemplo 1 (Exibindo o conteúdo de um arquivo)

```
1 #include <stdio.h>

3 int main() {
    FILE *fp; char ch;

5
    fp = fopen("lista_compras.csv", "r");
7    if (!fp) {
        printf("O arquivo não pode ser aberto.\n");
9        return -3;
    }

11
    while (!feof(fp)) {
13        ch = fgetc(fp);
        if (ch != EOF) {
15            printf("%c", ch);
        }

17    }

19    fclose(fp);
    return 0;

21 }
```

O que será impresso?

```
Produto;Fabricante;Qtd.;Medida;Preco;Data
Suco;ValeSuco;1;1;3,00;12/05/2011
Suco;Flash;1;1;4,50;12/05/2011
Tomate; - ;1;kg;3,50;14/05/2011
Arroz;Tio Jose;5;kg;8,64;14/05/2011
Arroz;Sem Broto;5;kg;9,99;12/05/2011
Feijao;Sem Broto;1;kg;4,00;12/05/2011
Tomate; - ;1;kg;2,99;14/05/2011
Ovo;A Granja;12;u;3,19;12/05/2011
```

Arquivos

[cat_simples.c]

Exemplo 2 (Criando um cat simplificado)

```
1  #include <stdio.h>

3  int main (int argc, char * argv[]) {
    FILE *fp; char ch;

5      if (argc != 2){
6          printf ("Uso: cat_simples <nome-do-arquivo>");
7          return -2;
8      }

11     fp = fopen(argv[1], "r");

13     if (!fp) {
14         printf("O arquivo não pode ser aberto.\n");
15         return -3;
16     }

17     while (!feof(fp)) {
18         ch = fgetc(fp);
19         if (ch != EOF) {
20             printf("%c", ch);
21         }
22     }

23     fclose(fp);
24     return 0;
25 }
27 }
```

O que será impresso?

Arquivos

[cat_simples.c]

Exemplo 2 (Criando um cat simplificado)

```
1  #include <stdio.h>

3  int main (int argc, char * argv[]) {
    FILE *fp; char ch;

5      if (argc != 2){
6          printf ("Uso: cat_simples <nome-do-arquivo>");
7          return -2;
8      }

11     fp = fopen(argv[1], "r");

13     if (!fp) {
14         printf("O arquivo não pode ser aberto.\n");
15         return -3;
16     }

17     while (!feof(fp)) {
18         ch = fgetc(fp);
19         if (ch != EOF) {
20             printf("%c", ch);
21         }
22     }

23     fclose(fp);
24     return 0;
25 }
```

O que será impresso?

Uso: cat_simples <nome-do-arquivo>

Arquivos

[cat_simples.c]

Exemplo 2 (Criando um cat simplificado)

```
1  #include <stdio.h>

3  int main (int argc, char * argv[]) {
    FILE *fp; char ch;

5      if (argc != 2){
6          printf ("Uso: cat_simples <nome-do-arquivo>");
7          return -2;
8      }

11     fp = fopen(argv[1], "r");

13     if (!fp) {
14         printf("O arquivo não pode ser aberto.\n");
15         return -3;
16     }

17     while (!feof(fp)) {
18         ch = fgetc(fp);
19         if (ch != EOF) {
20             printf("%c", ch);
21         }
22     }

23     fclose(fp);
24     return 0;
25 }
26
27 }
```

O que será impresso?

Uso: cat_simples <nome-do-arquivo>

```
$ ./cat_simples lista_compras.csv
Produto;Fabricante;Qtd.;Medida;Preco;Data
Suco;ValeSuco;1;1;3,00;12/05/2011
...
```

Arquivos

[escrita1.c]

Exemplo 3 (Escrita, Retrocesso e Leitura)

```
1  #include <stdio.h>

3  int main() {
    FILE *fp; char ch;

5
    fp = fopen("alfabeto.txt", "w+"); /* Abre para leitura/escrita */
7    if (!fp) {
        printf("Erro na abertura do arquivo.\n");
9        return -3;
    }

11
    for (int i = 65; i <= 90; i++) { /* Grava de A a Z */
13        fputc(i, fp);
    }

15
    rewind(fp); /* Rebobina o arquivo */

17
    while (!feof(fp)) { /* Leitura completa */
19        ch = fgetc(fp);
        if (ch != EOF) {
21            printf("%c", ch);
        }
23    }

25    fclose(fp); /* Fecha o arquivo */
    return 0;
27 }
```

O que será impresso?

Arquivos

[escrita1.c]

Exemplo 3 (Escrita, Retrocesso e Leitura)

```
1  #include <stdio.h>

3  int main() {
    FILE *fp; char ch;

5
    fp = fopen("alfabeto.txt", "w+"); /* Abre para leitura/escrita */
7    if (!fp) {
        printf("Erro na abertura do arquivo.\n");
9        return -3;
    }

11
    for (int i = 65; i <= 90; i++) { /* Grava de A a Z */
13        fputc(i, fp);
    }

15
    rewind(fp); /* Rebobina o arquivo */

17
    while (!feof(fp)) { /* Leitura completa */
19        ch = fgetc(fp);
        if (ch != EOF) {
21            printf("%c", ch);
        }

23    }

25    fclose(fp); /* Fecha o arquivo */
    return 0;

27 }
```

O que será impresso?

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Arquivos

Leitura e escrita de cadeia caracteres (*strings*)

- As funções **fgets** e **fputs** realizam a leitura e escrita de cadeias de caracteres em arquivos.

- Protótipo:

```
int fputs(char *str, FILE *fp);
```

- str** → *string* a ser gravada.
- fp** → ponteiro para o arquivo cujo *string* será escrito.
- A função **fputs** escreve o *string* **str** no *stream* **fp**.
- Em caso de erro, a função **fputs** retorna **EOF**.

Arquivos

Leitura e escrita de cadeia caracteres (*strings*)

- Protótipo `fgets`:

```
int fgets(char *str, int comp, FILE *fp);
```

- **str** → *string* a ser preenchida pela leitura.
 - **comp** → tamanho do *string*.
 - **fp** → ponteiro para o arquivo cujo *string* será lido.
- A função **fgets** lê uma cadeia de caracteres do *stream* **fp** para a variável **str**.
 - Até que um caractere de nova linha seja encontrado ou **comp-1** caracteres sejam lidos.
 - Em caso de erro, a variável **str** retorna **NULL**.

Arquivos

[escrita2.c]

Exemplo 4 (Escrita, Retrocesso e Leitura utilizando fgets e fputs)

```
1  #include <stdio.h>
   #define STR_TAM 100
3
4  int main() {
5      char linguagens[5][20] = { "PASCAL", "C", "C++", "SmallTalk", "Java" };
   FILE *fp; int i; char str[STR_TAM];
7
8      fp = fopen("linguagens.txt", "w+"); /* Abre para leitura/escrita */
9      if (!fp) {
10         printf("Erro na abertura do arquivo.\n");
11         return -3;
12     }
13
14     for (i = 0; i < sizeof(linguagens)/sizeof(linguagens[i]); i++) {
15         fputs(linguagens[i], fp);
16         fputc('\n', fp);
17     }
18
19     rewind(fp); /* Rebobina o arquivo */
20
21     while (!feof(fp)) { /* Leitura completa */
22         if (fgets(str, STR_TAM, fp) != NULL) {
23             printf("%s", str);
24         }
25     }
26
27     fclose(fp); /* Fecha o arquivo */
28     return 0;
29 }
```

O que será impresso?

Arquivos

[escrita2.c]

Exemplo 4 (Escrita, Retrocesso e Leitura utilizando fgets e fputs)

```
1  #include <stdio.h>
   #define STR_TAM 100
3
4  int main() {
5      char linguagens[5][20] = { "PASCAL", "C", "C++", "SmallTalk", "Java" };
   FILE *fp; int i; char str[STR_TAM];
7
8      fp = fopen("linguagens.txt", "w+"); /* Abre para leitura/escrita */
9      if (!fp) {
10         printf("Erro na abertura do arquivo.\n");
11         return -3;
12     }
13
14     for (i = 0; i < sizeof(linguagens)/sizeof(linguagens[i]); i++) {
15         fputs(linguagens[i], fp);
16         fputc('\n', fp);
17     }
18
19     rewind(fp); /* Rebobina o arquivo */
20
21     while (!feof(fp)) { /* Leitura completa */
22         if (fgets(str, STR_TAM, fp) != NULL) {
23             printf("%s", str);
24         }
25     }
26
27     fclose(fp); /* Fecha o arquivo */
28     return 0;
29 }
```

O que será impresso?

PASCAL
C
C++
SmallTalk
Java

Arquivos

E/S formatada

- As funções **fprintf** e **fscanf** são equivalentes as já conhecidas funções **printf** e **scanf**.
- As funções **printf** e **scanf** trabalham **sempre** com os arquivos padrões de E/S
 - **stdin** → entrada padrão, normalmente, teclado.
 - **stdout** → saída padrão, normalmente, monitor.
- A única diferença é que nas funções **fprintf** e **fscanf** deve-se informar qual o arquivo em que se está trabalhando.

Arquivos

E/S formatada

- Protótipo fprintf:

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

- **fp** → ponteiro para o arquivo a ser escrito.
- **format** → sequência de conversão.
- ... → variáveis (varargs).

- Exemplo:

```
printf("%d", num) ↔ fprintf(stdout, "%d", num);
```

Arquivos

E/S formatada

- Protótipo fscanf:

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

- **fp** → ponteiro para o arquivo a ser escrito.
- **format** → sequência de conversão.
- ... → variáveis (varargs).

- Exemplo:

```
scanf("%d", &num) ↔ fscanf(stdin, "%d", &num);
```

Arquivos

E/S formatada

- Embora essas funções, por sua semelhança com `printf` e `scanf`, sejam maneiras convenientes de escrever e ler dados de arquivos, elas têm a desvantagem de serem mais lentas do que uso de arquivos binários.
- A perda de tempo ocorre devido aos dados serem armazenados na codificação ASCII, o que obriga conversão de dados a cada operação.
- Contudo, dados armazenados em ASCII pode ser também uma vantagem, uma vez que são facilmente verificados pelos usuários, o que não ocorre com dados binários.

Arquivos

[grava_matriz.c]

Exemplo 5 (Gravando uma matriz em um arquivo)

```
1  #include <stdio.h>

3  int main() {
    FILE *fp;
5   int i, j, tam, aux;

7   printf("Tamanho da Matriz Quadrada: ");
   scanf("%d", &tam);

9   fp = fopen("matriz.txt", "w"); /* Abre para escrita */
11  if (!fp) {
    printf("Erro na abertura do arquivo.\n");
13     return -3;
   }
15  fprintf(fp, "%d\n", tam); /* Armazena o tamanho da matriz */

17  for (i = 0; i < tam; i++) { /* Lê matriz gravando no arquivo */
    for (j = 0; j < tam; j++) {
19      printf("mat[%d][%d] = ", (i + 1), (j + 1));
      scanf("%d", &aux);
21      fprintf(fp, "%d ", aux);
    }
23    fprintf(fp, "\n");
   }
25

   fclose(fp); /* Fecha o arquivo */
27  return 0;
}
```

O que será impresso?

Arquivos

[grava_matriz.c]

Exemplo 5 (Gravando uma matriz em um arquivo)

```
1  #include <stdio.h>

3  int main() {
    FILE *fp;
5   int i, j, tam, aux;

7   printf("Tamanho da Matriz Quadrada: ");
   scanf("%d", &tam);

9   fp = fopen("matriz.txt", "w"); /* Abre para escrita */
11  if (!fp) {
    printf("Erro na abertura do arquivo.\n");
13     return -3;
   }

15  fprintf(fp, "%d\n", tam); /* Armazena o tamanho da matriz */

17  for (i = 0; i < tam; i++) { /* Lê matriz gravando no arquivo */
    for (j = 0; j < tam; j++) {
19       printf("mat[%d][%d] = ", (i + 1), (j + 1));
       scanf("%d", &aux);
21       fprintf(fp, "%d ", aux);
    }
23     fprintf(fp, "\n");
   }

25  fclose(fp); /* Fecha o arquivo */
27  return 0;
}
```

O que será impresso?

```
Tamanho da Matriz Quadrada: 2
mat[1][1] = 6
mat[1][2] = 8
mat[2][1] = 1
mat[2][2] = 3
```

Arquivos

[le_matriz.c]

Exemplo 6 (Lendo uma matriz de um arquivo)

```
1  #include <stdio.h>

3  int main() {
    FILE *fp;
5   int i, j, tam, aux;

7   fp = fopen("matriz.txt", "r"); /* Abre para leitura */

9   if (!fp) {
    printf("Erro na abertura do arquivo.\n");
11    return -3;
    }

13    fscanf(fp, "%d", &tam); /* Le o tamanho */

15    for (i = 0; i < tam; i++) { /* Imprime a matriz gravada no arquivo */
17        for (j = 0; j < tam; j++) {
            fscanf(fp, "%d", &aux);
19            fprintf(stdout, "%d ", aux);

21            fprintf(stdout, "\n");
        }

23        fclose(fp); /* Fecha o arquivo */
25        return 0;
    }
```

O que será impresso?

Arquivos

[le_matriz.c]

Exemplo 6 (Lendo uma matriz de um arquivo)

```
1  #include <stdio.h>

3  int main() {
    FILE *fp;
5   int i, j, tam, aux;

7   fp = fopen("matriz.txt", "r"); /* Abre para leitura */

9   if (!fp) {
    printf("Erro na abertura do arquivo.\n");
11    return -3;
    }

13    fscanf(fp, "%d", &tam); /* Le o tamanho */

15    for (i = 0; i < tam; i++) { /* Imprime a matriz gravada no arquivo */
17        for (j = 0; j < tam; j++) {
            fscanf(fp, "%d", &aux);
19            fprintf(stdout, "%d ", aux);

21            fprintf(stdout, "\n");
        }

23        fclose(fp); /* Fecha o arquivo */
25        return 0;
    }
```

O que será impresso?

```
6 8
1 3
```


- 1 Introdução
- 2 Entrada e Saída com Arquivos
- 3 Conclusões**
 - Considerações Finais
 - Referências bibliográficas

Considerações Finais

- ❖ Nesta aula foram apresentados os principais conceitos relacionados a:
 - Manipulação de arquivos na linguagem C.
- ✓ *É importante rever os conceitos apresentados na aula e consultar a bibliografia sugerida sobre o assunto.*

Referências Bibliográficas I



Aguilar, L. J. (2008).

Programação em C++: Algoritmos, Estruturas de Dados e Objetos.
McGraw-Hill, São Paulo.



Cormen, T. H., Leiserson, C. E., Rivest, R. L., e Stein, C. (2002).

Algoritmos: Teoria e Prática.
Elsevier, Rio de Janeiro.



Drozdek, A. (2009).

Estrutura de Dados e Algoritmos em C++.
Cengage Learning, São Paulo.



Forbellone, A. L. V. e Eberspacher, H. F. (2005).

Lógica de Programação: A Construção de Algoritmos e Estrutura de Dados.
Pearson Prentice Hall, São Paulo, 3 edition.



Knuth, D. E. (2005).

The Art of Computer Programming.
Addison-Wesley, Upper Saddle River, NJ, USA.



Pinheiro, F. d. A. C. (2012).

Elementos de Programação em C.
Bookman, Porto Alegre.

Referências Bibliográficas II



Sedgewick, R. (1998).

Algorithms in C: Parts 1-4, Fundamentals, Data Structures, Sorting, Searching.
Addison-Wesley, Boston, 3rd edition.



Szwarcfiter, J. L. e Markenzon, L. (1994).

Estruturas de Dados e Seus Algoritmos.
LTC, Rio de Janeiro.



Tenenbaum, A. A., Langsam, Y., e Augenstein, M. J. (1995).

Estruturas de Dados Usando C.
Makron Books, São Paulo.



Terra, R. (2014).

Linguagem C - Notas de Aula.

Universidade Federal de Lavras (UFLA).

Disponível em:

http://professores.dcc.ufla.br/~terra/public_files/2014_apostila_c_ansi.pdf. Acesso em: 2018-09-16.