

Estruturas de Dados II

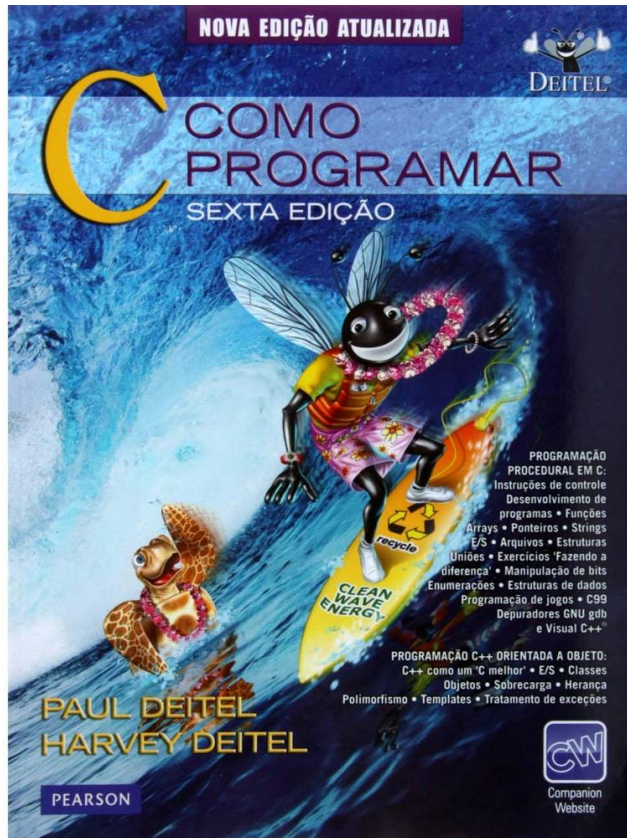
Manipulação de Arquivos

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

Bibliografia



Título: Como Programa C – Sexta Edição

Editora: Pearson

Autor: Paul Deitel

Capítulo 11.

Manipulação de Arquivo

- O armazenamento de dados em variáveis e arrays é temporário; **todos os dados são perdidos quando um programa termina.**
- Os arquivos são usados para conservação permanente de grandes quantidades de dados.
- Os computadores armazenam arquivos em dispositivos secundários de armazenamento, especialmente dispositivos de armazenamento em disco.

Manipulação de Arquivo

- A linguagem C visualiza cada arquivo simplesmente como um fluxo sequencial de bytes.
- Cada arquivo termina ou com um marcador de final de arquivo (**end-of-file marker**),

Manipulação de Arquivo

- Abrir um arquivo retorna um **ponteiro** para uma estrutura FILE (**definida em <stdio.h>**) que contém informações usadas para processar o arquivo.
- O padrão de entrada, o padrão de saída e o padrão de erros são manipulados por meio dos ponteiros de arquivos **stdin**, **stdout** e **stderr**.

Modos de Abertura

- Para criar um arquivo ou eliminar o conteúdo de um arquivos antes da gravação dos dados, abra o arquivo para gravação ("w").
- Para ler um arquivo existente, abra-o para leitura ("r").
- Para adicionar registros ao final de um arquivo existente, abra o archive append("a").
- Para abrir um arquivo de forma que ele possa ser gravado e lido, abra o arquivo com um dos três modos de atualização "r+", "w+" ou "a+".
 - O modo "r+" abre um arquivo para leitura e gravação.
 - O modo "w+" cria um arquivo para leitura e gravação. Se o arquivo já existir, o arquivo é aberto e o conteúdo atual é eliminado.
 - O modo "a+" abre um arquivo para leitura e gravação. Toda gravação é feita no final do arquivo. Se o arquivo não existir, é criado.

Modos de Abertura (resumo)

"r"

- Abre um arquivo texto para leitura. O arquivo deve existir antes de ser aberto.

"w"

- Abrir um arquivo texto para gravação. Se o arquivo não existir, ele será criado. Se já existir, o conteúdo anterior será destruído.

"a"

- Abrir um arquivo texto para gravação. Os dados serão adicionados no fim do arquivo ("append"), se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.

"r+"

- Abre um arquivo texto para leitura e gravação. O arquivo deve existir e pode ser modificado.

"w+"

- Cria um arquivo texto para leitura e gravação. Se o arquivo existir, o conteúdo anterior será destruído. Se não existir, será criado.

"a+"

- Abre um arquivo texto para gravação e leitura. Os dados serão adicionados no fim do arquivo se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.

"...b"

- Executa a mesma operação, só que o arquivo é binário.. Ex. a+b, wb, r+b, etc

fopen

- **Fopen**
- Esta é a função de abertura de arquivos. Seu protótipo é:

```
FILE *fopen (char *nome_do_arquivo, char *modo);
```


fclose

- **fclose**
- Quando acabamos de usar um arquivo que abrimos, devemos fechá-lo. Para tanto usa-se a função **fclose()**:

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

eof

- **feof**
- EOF ("End of file") indica o fim de um arquivo.
- Ela retorna não-zero se o arquivo chegou ao EOF, caso contrário retorna zero. Seu protótipo é:

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

fscan

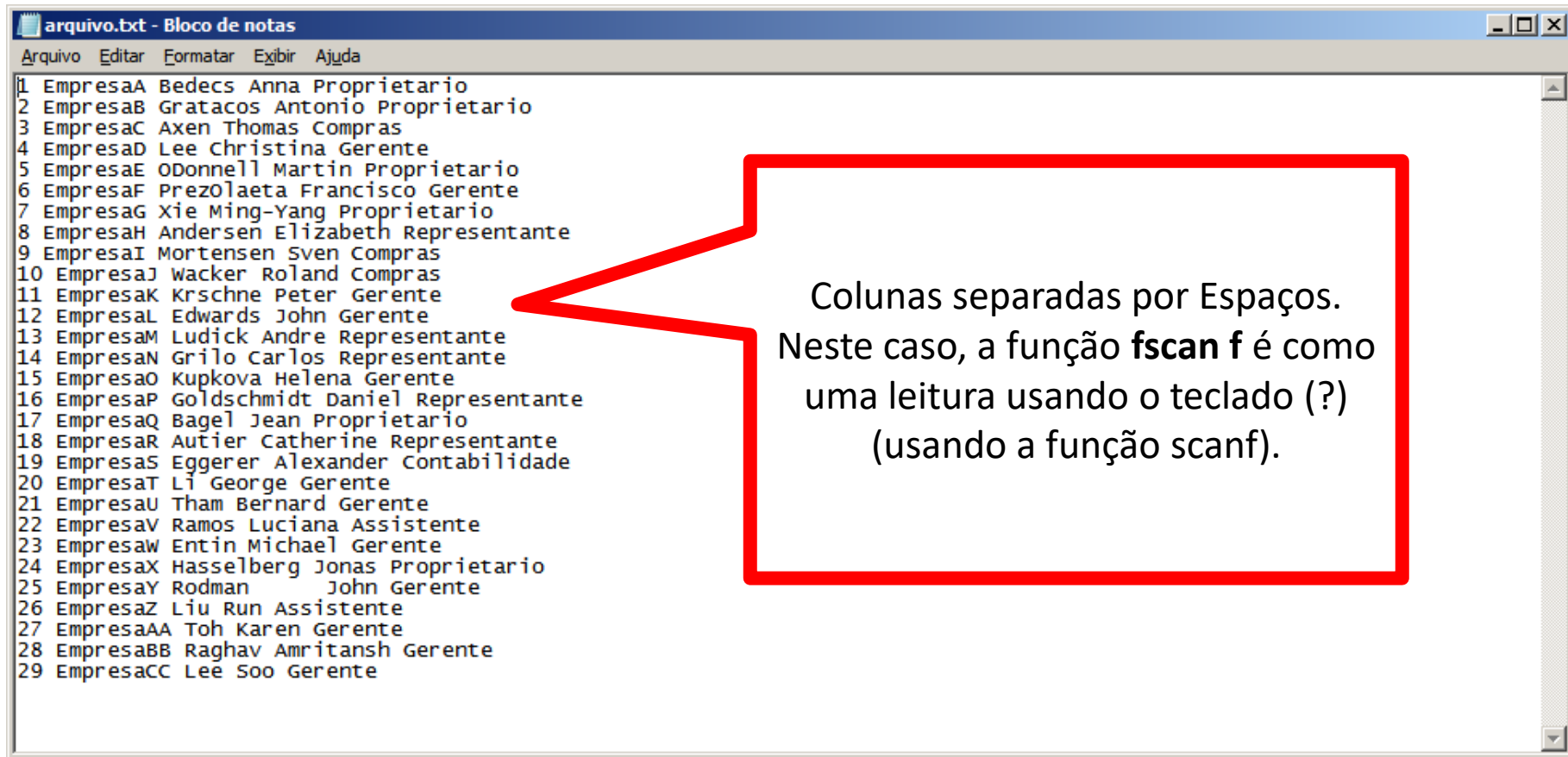
- **fscanf**
- A função **fscanf()** funciona como a função **scanf()**. A diferença é que **fscanf()** lê de um arquivo e não do teclado do computador.
- Protótipo:

```
int fscanf (FILE *fp, char *str, variável) ;
```

EXEMPLO 01 – Lendo Arquivo Exemplo

(Note, o arquivo exemplo tem 6 colunas)

Layout do Arquivo



```
arquivo.txt - Bloco de notas
Arquivo  Editar  Formatar  Exibir  Ajuda
1 EmpresaA Bedecs Anna Proprietario
2 EmpresaB Gratacos Antonio Proprietario
3 EmpresaC Axen Thomas Compras
4 EmpresaD Lee Christina Gerente
5 EmpresaE ODonnell Martin Proprietario
6 EmpresaF Prezolaeta Francisco Gerente
7 EmpresaG Xie Ming-Yang Proprietario
8 EmpresaH Andersen Elizabeth Representante
9 EmpresaI Mortensen Sven Compras
10 EmpresaJ Wacker Roland Compras
11 EmpresaK Krschne Peter Gerente
12 EmpresaL Edwards John Gerente
13 EmpresaM Ludick Andre Representante
14 EmpresaN Grilo Carlos Representante
15 EmpresaO Kupkova Helena Gerente
16 EmpresaP Goldschmidt Daniel Representante
17 EmpresaQ Bagel Jean Proprietario
18 EmpresaR Autier Catherine Representante
19 EmpresaS Eggerer Alexander Contabilidade
20 EmpresaT Li George Gerente
21 EmpresaU Tham Bernard Gerente
22 EmpresaV Ramos Luciana Assistente
23 EmpresaW Entin Michael Gerente
24 EmpresaX Hasselberg Jonas Proprietario
25 EmpresaY Rodman John Gerente
26 EmpresaZ Liu Run Assistente
27 EmpresaAA Toh Karen Gerente
28 EmpresaBB Raghav Amritansh Gerente
29 EmpresaCC Lee Soo Gerente
```

Colunas separadas por Espaços.
Neste caso, a função **fscanf** é como
uma leitura usando o teclado (?)
(usando a função scanf).

Lendo Arquivo Sequencial (fscanf)

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

main(){
    int codigo;
    char empresa[10];
    char sobre[12];
    char nome[10];
    char funcao[10];

    FILE *txt;

    if((txt = fopen("arquivo.colunas","r")) == NULL)
        {
            printf("Erro ao abrir arquivo");
        }
    else
    {
        while (!feof(txt)) {
            fscanf(txt, "%d %s %s %s %s ", &codigo, empresa, sobre, nome, funcao);
            printf("%d \t %-10s %-12s %-10s %-10s \n", codigo, empresa, sobre, nome, funcao);
        }
        fclose(txt);
    }
    system("pause");
}
```

EXEMPLO 02 – Gravando um Arquivo

Escrevendo Formatado: fprintf

- **fprintf**
 - A função **fprintf()** funciona como a função **printf()**. A diferença é que a saída de **fprintf()** é um arquivo e não a tela do computador.
- Protótipo:

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


Gravando Arquivo Sequencial

```
#include <stdio.h>

main(){
    int i;
    FILE *txt;

    if((txt = fopen("Arquivo.txt", "a+")) == NULL)    {
        printf("Erro ao abrir arquivo");
    }
    else {
        i = 50;
        while(i < 100)
        {
            fprintf(txt, "%d\n", i);
            ++i;
        }
        fclose(txt);
    }
}
```

EXEMPLO 04 – Lendo um Arquivo – caractere a caractere.

Leitura Byte a Byte: fgetc

- **Fgetc**

- Lê o caractere presente na posição atual do fluxo interno. Após a leitura, a posição atual é avançada para o próximo caractere.

- Protótipo:

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

Lendo Arquivo Sequencial Não Formatado

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

main()
{
    char caractere;
    FILE *txt;
    if((txt = fopen("c:\\arquivo.txt", "r")) == NULL)
    {
        printf("Erro ao abrir arquivo");
    }
    else
    {
        while (!feof(txt)) {
            caractere = fgetc(txt);
            printf("%c", caractere);
        }
        fclose(txt);
    }
    system("pause");
}
```



Lê um caractere por vez.

Inserindo Barra “|”entre as palavras.

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

main()
{
    char caractere;

    FILE *txt;

    if((txt = fopen("c:\\arquivo.txt","r")) == NULL)
    {
        printf("Erro ao abrir arquivo");
    }
    else
    {
        while (!feof(txt)) {
            caractere = fgetc(txt);
            printf("%c", caractere);
        }
        fclose(txt);
    }
    system("pause");
}
```

Lê um caractere por vez.

```
if (caractere == ' ')
    printf(" | ");
else
    printf("%c", caractere);
```

EXEMPLO 05 – Lendo um arquivo - conjunto de
de caracteres ou parágrafo.

Lendo conjunto de bytes: fgets

•fgets

- Lê do *fluxo* para a cadeia de caracteres *string* até a quantidade de caracteres (*tamanho* - 1) ser lida ou até uma nova linha (`\n`) ou EOF ser encontrado. Após a leitura, a posição atual do *fluxo* é avançada para o próximo caractere não lido.
 - Para ler da entrada padrão (stdin) de forma segura, é necessário o uso desta função. Como ela limita o número de caracteres lidos pelo parâmetro *tamanho*, ela previne buffer overflows que possam causar erros de segurança ou *crashes* na aplicação.
- A função fgets **pára** caso encontre uma nova linha (`\n`), incluindo-a na *string*.
 - A função lê até o (*tamanho* - 1) pois devemos contar o espaço para o NULL (`'\0'`) no final da *string*.
- Protótipo:

```
char * fgets (char * string, int tamanho, FILE * fluxo);
```

Lendo Arquivo Sequencial (fgets)

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

main()
{
    char linha[1024];

    FILE *txt;

    if((txt = fopen("arquivo_texto.txt","r")) == NULL)
    {
        printf("Erro ao abrir arquivo");
    }
    else
    {
        while (!feof(txt)) {
            fgets(linha, 1024, txt);
            printf("%s", linha);
        }
        fclose(txt);
    }
    system("pause");
}
```

Lê uma string até o **\n** ou **1023** caracteres;

Lendo Arquivo Sequencial (fgets)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main() {
    char linha[1024];
    char * ultima;
    FILE *txt;
    if((txt = fopen("c:\\arquivo.txt","r")) == NULL)
    {
        printf("Erro ao abrir arquivo");
    }
    else {
        while (!feof(txt)) {
            fgets(linha, 1024, txt);
            ultima = strtok (linha, " ");
            while (ultima != NULL)
            {
                printf ("%s ",ultima);
                ultima = strtok (NULL, " ");
            }
            fclose(txt);}
        system("pause"); }
```

strtok quebra a string no delimitador. (neste caso, separa em palavras)

Arquivos Binários

EXEMPLO 06 – Escrevendo Arquivos Binários.

fwrite

FWRITE

- fwrite tenta escrever para o *fluxo* *numero_itens* elementos, com *tamanho* bytes cada.
- Em caso de sucesso, ou seja, todos os elementos tenham sido escritos com sucesso, fwrite escreveu (*tamanho* * *numero_itens*) bytes do parâmetro *dados* para o *fluxo*.
- fwrite funciona como se [fputc](#) fosse chamada *tamanho* vezes para cada objeto.
- O ponteiro interno de posição do *fluxo* é avançado pelo número de bytes escritos com sucesso.
- A marca temporal de última modificação do arquivo é atualizada.

Protótipo:

```
size_t fwrite(void * dados, size_t tamanho, size_t numero_itens, FILE * fluxo);
```

```
#include <stdio.h>
#include <string.h>
typedef struct Cpessoa
{   char nome[20];
    int idade; };

int main(void)
{
    char condicao = 's';
    Cpessoa aluno;
    FILE *bin;

    if((bin = fopen("arquivo_binario.txt","ab")) == NULL)
    {
        printf("Erro ao abrir arquivo");    }
    else
    {
        while (condicao == 's' || condicao == 'S')
        {
            printf("Informe o nome:");
            scanf("%s", aluno.nome);

            printf("Informe a idade:");
            scanf("%d", &aluno.idade);

            printf("Continuar S/N?:");

            fwrite(&aluno, 1, sizeof(aluno), bin);

            fflush(stdin);
            condicao = getchar();
        }
    }
}
```

EXEMPLO 07 – Lendo Arquivos Binários.

fread

FREAD

- fread tenta ler do *fluxo* *numero_itens* elementos, com *tamanho* bytes cada. Em caso de sucesso, ou seja, todos os elementos tenham sido lidos com sucesso, fread lê (*tamanho* * *numero_itens*) bytes do *fluxo* para o parâmetro *dados*.
- fread funciona como se [fgetc](#) fosse chamada *tamanho* vezes para cada objeto. Note que fread apenas *funciona como* chamadas sucessivas a fgetc. Na realidade, fread não faz uso da função fgetc.
- O ponteiro interno de posição do *fluxo* é avançado pelo número de bytes lidos.

Protótipo:

```
size_t fread(void * dados, size_t tamanho, size_t numero_itens,  
             FILE * fluxo);
```

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

typedef struct Cessoa
{
    char nome[20];
    int idade;};

int main(void)
{
    char condicao = 's';
    Cessoa aluno[200];
    FILE *bin;
    int tamanho = 0;

    if((bin = fopen("arquivo_binario.txt","rb")) == NULL)
    {
        printf("Erro ao abrir arquivo");
    }
    else
    {
        //Lendo o Arquivo
        while (!feof(bin))
        {
            fread(&aluno[tamanho] ,1, sizeof(Cessoa), bin);
            tamanho++;
        }
        //Mostrando na tela
        for (int i = 0; i<tamanho-1 ; i++)
        {
            printf("%s \t %d \n", aluno[i].nome, aluno[i].idade);
        }
        getchar();
    }
}
```