

Arquivos Textos

Programação de Computadores 1



Prof. Daniel Saad Nogueira
Nunes

IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

- 1 Introdução
- 2 Conceitos
- 3 Arquivos textos
- 4 Exemplos



Sumário

1 Introdução



Introdução

- Variáveis e vetores são armazenados na memória primária. Ou seja, após a finalização do programa não é possível manipulá-los.
- Caso o objetivo seja conservar as informações, devemos gravar os dados em um dispositivo de memória secundária como: HDs, SSDs, fitas magnéticas. . .
- Utilizamos **arquivos** para gravar as informações nestes dispositivos.



Sumário

2 Conceitos



Sumário

2 Conceitos

- Hardware
- Arquivos e Diretórios
- Caminhos



Memória

- A hierarquia de organização da memória em um computador pode ser dividida em vários níveis, no formato de uma pirâmide, cada um contemplando uma categoria.
- As categorias no topo são mais caras, com menos capacidade e mais rápidas.
- As categorias na base são mais baratas, com maior capacidade, e mais lentas.



Memória

- Ainda as memórias são subdivididas em primária e secundária.
- Memórias primárias **não são** persistentes e podem ser acessíveis diretamente pelo processador.
- Memórias secundárias **são** persistentes e **não** podem ser acessíveis diretamente.
- **Persistência**: capacidade de manter os dados após a finalização dos programas que os tenham gerado.



Memória

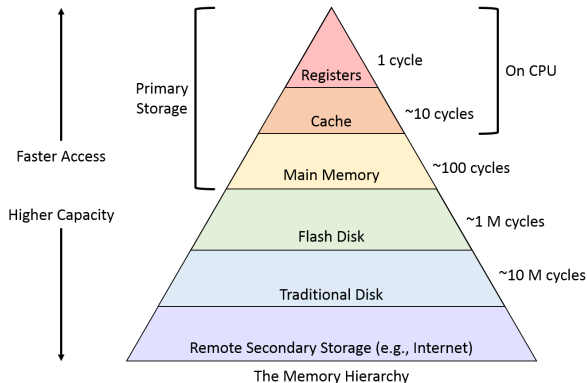


Figura: fonte https://www.cs.swarthmore.edu/~kwebb/cs31/f18/memhierarchy/mem_hierarchy.html



Hierarquia de Memória

- Categorias de memória primária: RAM, cache, registradores, . . .
- Categorias de memória secundária: SSDs, HDs, fitas magnéticas, . . .



Persistência

- Se o objetivo é manter os dados acessíveis, mesmo que o programa tenha finalizado, devemos gravá-los em memória secundária.
- Utilizaremos o conceito de **arquivos** para isso.



Sumário

2 Conceitos

- Hardware
- Arquivos e Diretórios
- Caminhos



Arquivo

- Arquivos são abstrações criadas pelo Sistema Operacional para armazenamento de informações em dispositivos.
- Os dados escritos em um arquivo são armazenados posteriormente em um dispositivo físico como um HD e o SSD.
- Facilita a vida do programador, pois em vez de lidar com endereços físicos e precisar conhecer as características do dispositivo em que se quer armazenar os dados, só precisamos saber manipular o arquivo.
- O **Sistema de Arquivos** é o módulo do Sistema Operacional responsável por realizar esse diálogo entre a entidade lógica (arquivo) e a entidade física (dispositivo).



Nomes e Extensões

- Arquivos são identificados pelo seu caminho.
- Também podem ser acompanhados de uma extensão, que serve para guiar o Sistema Operacional a utilizar o programa correto na hora de abri-lo.



Nomes e Extensões

| Nome | Extensão | Programas Típicos |
|-------------|-----------------|----------------------------------|
| arq.txt | txt | bloco de notas, gedit, ... |
| arq.c | c | vscode, codeblocks, clion, ... |
| arq.pdf | pdf | evince, adobe reader, okular,... |
| arq.exe | exe | executável (Windows) |
| arq | | executável (Linux) |



Diretórios

- Diretórios, fornecem mecanismos para organizar arquivos em uma divisão lógica para o usuário.
- O usuário ou sistema normalmente agrupam arquivos que fazem sentido junto sob um único diretório.



Sumário

2 Conceitos

- Hardware
- Arquivos e Diretórios
- Caminhos



Caminhos

- Os arquivos e diretórios estão organizados hierarquicamente. Eles podem estar inseridos em outros diretórios.
- Estão organizados sob uma estrutura com formato de **árvore**.
- **Caminhos** são responsáveis por identificar cada arquivo: são o endereço de um arquivo no sistema de arquivos.
- Diretório raiz: (/).



Estrutura em Árvore

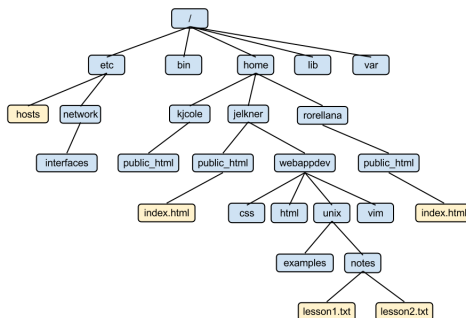


Figura: Fonte: <https://www.openbookproject.net/tutorials/getdown/unix/lesson2.html>

[//www.openbookproject.net/tutorials/getdown/unix/lesson2.html](https://www.openbookproject.net/tutorials/getdown/unix/lesson2.html)



Caminhos

- Os caminhos de cada arquivo ou diretório são simplesmente a concatenação dos nomes no percurso do diretório raiz até este arquivo.
- Utilizando a figura anterior:
 - ▶ `/bin`: diretório `bin` sob a pasta raiz.
 - ▶ `/home/jelkner/public_html/index.html`, arquivo `index.html` organizado sob `public_html`, que está sob o diretório `jelkner`, incluído está sob o diretório `home`, inserido sob o diretório raiz.
 - ▶ `/home/rorellana/public_html/index.html`, arquivo `index.html` organizado sob `public_html`, que está sob o diretório `rorellana`, incluído está sob o diretório `home`, inserido sob o diretório raiz.



Caminhos Relativos

- Caminhos **relativos** também podem ser utilizados para identificar arquivos.
- Eles levam em consideração o **diretório atual**.
- Exemplos:
 - ▶ `../arq.txt`: estamos nos referindo ao arquivo `arq.txt` que está no diretório acima do diretório atual.
 - ▶ `arq2.txt`: estamos nos referindo ao arquivo `arq2.txt` que está no diretório atual. Sinônimo de `./arq2.txt`.
 - ▶ `../dir/arq3.txt`: estamos nos referindo ao arquivo `arq3.txt` que está sob o diretório `dir`, encontrado no diretório acima do atual.



Caminhos

Caminho absoluto vs caminho relativo

- **Caminho absoluto:** descreve o percurso da raiz até o arquivo ou diretório. Sempre é único.
- **Caminho relativo:** descreve o percurso do diretório atual até o arquivo ou diretório pretendido. Depende do **diretório atual**.



Sumário

3 Arquivos textos



Arquivos Textos

- Arquivos textos são utilizados para armazenar informações no formato de texto.
- O conteúdo desses arquivos é legível por uma pessoa qualquer.
- Qualquer editor simples de texto consegue abri-los, permitir visualização e manipulá-los.
- Exemplos de arquivos textos: códigos-fonte, html, arquivos de extensão `.txt`.



Sumário

3 Arquivos textos

- Abertura e fechamento
- Leitura
- Escrita



Abertura

- Antes de realizar qualquer operação em arquivos, é necessário abri-lo antes.
- Após a abertura, qualquer alteração no arquivo é feito através de uma variável do tipo `FILE*`, isto é, uma variável de tipo ponteiro para arquivo.
- A abertura de um arquivo é realizado através da função `fopen`.
- Retorno:



Abertura

- A variável do tipo `FILE*` mantém as informações sobre o arquivo.
- Dentre muitas informações, ela armazena a posição do **indicador de posição**, que indica a próxima posição no arquivo em que se poderá ler ou escrever.
- A cada operação de leitura ou escrita esse indicador de posição é modificado automaticamente.
- Manipulações sobre arquivos requerem o uso do cabeçalho `stdio.h`



Abertura

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

- `filename`: o caminho absoluto ou relativo do arquivo que se quer abrir.
- `mode`: modo de abertura (somente leitura, somente escrita, *etc.*)
- Retorno: um ponteiro para o arquivo ou **NULL** em caso de falha.



Modos de abertura

| Modo | Permissão | Indicador de posição |
|-------------|-----------------------|-----------------------------|
| r | leitura | início do arquivo |
| r+ | leitura e atualização | início do arquivo |
| w | escrita | início do arquivo |
| w+ | escrita e atualização | início do arquivo |
| a | escrita | final do arquivo |
| a+ | escrita e atualização | final do arquivo |



Modos de abertura

- **r**: somente leitura. O arquivo precisa existir, caso contrário `fopen` retornará **NULL**.
- **w**: somente escrita. Se o arquivo não existir, ele é criado. Se o arquivo existir, ele é completamente sobrescrito.
- **r+**: leitura e atualização. O arquivo precisa existir, caso contrário `fopen` retornará **NULL**. Também é possível realizar operações de escrita no arquivo.
- **w+**: escrita e atualização. Se o arquivo não existir, ele é criado. Se o arquivo existir, ele é completamente sobrescrito. Permite operações de leitura.



Modos de abertura

- a: somente escrita. Se o arquivo existir, ele não é sobrescrito. Se ele não existir, ele é criado. Qualquer operação de escrita é feita a partir do **final** do arquivo.
- a+: escrita e atualização. Se o arquivo existir, ele não é sobrescrito. Se ele não existir, ele é criado. Qualquer operação de escrita é feita a partir do **final** do arquivo.



Modos de abertura

Recomendação

- Utilize apenas o modo de abertura estritamente necessário, nunca utilize um modo que forneça mais possibilidades do que você precisa.
- Por exemplo, se o objetivo é apenas ler um arquivo, sem modificá-lo, prefira o `r` ao `r+`.



Abertura: exemplo

```
1  #include <stdio.h>
2
3  int main(void) {
4      FILE *arq = fopen("arq.txt", "r"); // read-only
5      if (arq == NULL) {
6          printf("Impossível abrir o arquivo.\n");
7          return 0;
8      }
9      // ... operações
10     return 0;
11 }
```

- Usamos o caminho relativo `arq.txt`, estamos pedindo para abrir o arquivo `arq.txt` que está na pasta do executável!



Fechamento

- Após realizar todas as operações sobre um arquivo, é importante fechá-lo.
- Usamos a função `fclose` sobre o ponteiro do arquivo aberto.
- Assinatura: `void fclose(FILE* fp);`
- Esquecer de fechar arquivos pode levar a alguns erros, como a não escrita de todos os dados no arquivo, visto que as operações de leitura e escrita são *bufferizadas*.
- Ao fechar os arquivos, estamos assegurando que todos os dados armazenados em *buffers* sejam efetivamente escritos no arquivo.



Sumário

3 Arquivos textos

- Abertura e fechamento
- **Leitura**
- Escrita



Leitura: fscanf

- Para ler um arquivo, utilizamos a função `fscanf`.
- Ela funciona como a função `scanf`, mas recebe um parâmetro extra: um ponteiro para arquivo.
- Na verdade `scanf` pode ser vista como um `fscanf` sobre o arquivo `stdin`.



Leitura: fscanf

fscanf

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

- `fp`: ponteiro para o arquivo.
- `format`: especificadores de formato, como no `scanf`.
- Os demais parâmetros são as variáveis nas quais os valores serão armazenados.
- Retorno: o número de itens lidos ou **EOF** caso atingido o fim de arquivo.



Exemplo: impressão de arquivo

Problema

Imprimir o conteúdo de um arquivo na tela.



Exemplo: impressão de arquivo

- O programa a seguir lê o conteúdo de um arquivo, e o imprime na tela, similar ao comando `cat`.
- O caminho do arquivo é obtido através de parâmetros de linha de comando.
- Funções auxiliares são utilizadas para encerrar o programa caso o caminho não tenha sido passado por parâmetro ou caso não tenha sido possível abrir o arquivo.



Exemplo: leitura

```
void testa_parametros(int argc) {  
    if (argc != 2) {  
        printf("Uso: ./executavel <arquivo>");  
        exit(0);  
    }  
}
```




Exemplo: leitura

```
void testa_abertura(FILE *fp, const char *path, const char *mode) {  
    if (fp == NULL) {  
        printf("Falha em abrir o arquivo %s com o modo %s\n", path, mode);  
        exit(0);  
    }  
}
```



Exemplo: leitura

```
void imprime_arquivo(FILE *fp) {  
    char c;  
    while (fscanf(fp, "%c", &c) != EOF)  
        printf("%c", c);  
}
```



Exemplo: leitura

```
int main(int argc, char *argv[]) {
    testa_parametros(argc);
    FILE *fp = fopen(argv[1], "r");
    testa_abertura(fp, argv[1], "r");
    imprime_arquivo(fp);
    fclose(fp);
    return 0;
}
```



Exemplo: leitura

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void testa_parametros(int argc) {
5      if (argc != 2) {
6          printf("Uso: ./executavel <arquivo>");
7          exit(0);
8      }
9  }
10
11 void testa_abertura(FILE *fp, const char *path, const char *mode) {
12     if (fp == NULL) {
13         printf("Falha em abrir o arquivo %s com o modo %s\n", path, mode);
14         exit(0);
15     }
16 }
17
18 void imprime_arquivo(FILE *fp) {
19     char c;
20     while (fscanf(fp, "%c", &c) != EOF)
21         printf("%c", c);
22 }
```



Exemplo: leitura

```
23
24 int main(int argc, char *argv[]) {
25     testa_parametros(argc);
26     FILE *fp = fopen(argv[1], "r");
27     testa_abertura(fp, argv[1], "r");
28     imprime_arquivo(fp);
29     fclose(fp);
30     return 0;
31 }
```



Leitura: fgets

fgets

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

- A função `fgets` pode ser utilizada para ler uma linha de um arquivo.
- `str` a string que armazenará a linha lida.
- `n`: o tamanho de `str`.
- `fp`: o ponteiro para arquivo.
- Retorno: um ponteiro para a string lida em caso de sucesso ou **EOF** caso esteja no fim do arquivo.
- Já utilizamos esta função para ler uma linha da entrada padrão ao colocar **stdin** no terceiro argumento.



Leitura: fgets

- Podemos adaptar o programa anterior para imprimir o conteúdo de um arquivo.
- Lemos uma linha inteira, em vez de um caractere.



Leitura: fgets

```
void imprime_arquivo(FILE *fp) {  
    char c;  
    char linha[82];  
    while (fgets(linha, 82, fp) != NULL)  
        printf("%s", linha);  
}
```




Sumário

3 Arquivos textos

- Abertura e fechamento
- Leitura
- Escrita



Escrita: fprintf

fprintf

- Para escrever em um arquivo, utilizamos a função `fprintf`.
- Ela funciona como a função `printf`, mas recebe um parâmetro extra: um ponteiro para arquivo.
- Na verdade `printf` pode ser vista como um `fprintf` sobre o arquivo `stdout`.



Escrita: fprintf

fprintf

- `fp` : ponteiro para o arquivo.
- `format` : especificadores de formato, como no `printf` .
- Os demais parâmetros são as variáveis nas quais os valores serão armazenados.
- Retorno: o número de bytes escritos ou um número negativo em caso de erro.



Exemplo: cópia de arquivo

Problema

Criar uma cópia de um arquivo texto.



Exemplo: escrita

- O programa a seguir copia o conteúdo de um arquivo para o outro, similar ao comando `cp`.
- Os caminhos dos arquivos são obtidos através da linha de comando.
- Funções auxiliares são utilizadas para encerrar o programa caso os caminhos não tenham sido passado pela linha de comando ou caso não tenha sido possível abrir os arquivos.



Exemplo: escrita

```
void testa_parametros(int argc) {  
    if (argc != 3) {  
        printf("Uso: ./executavel <arquivo_entrada> <arquivo_saida>");  
        exit(0);  
    }  
}
```



Exemplo: escrita

```
void testa_abertura(FILE *fp, const char *path, const char *mode) {  
    if (fp == NULL) {  
        printf("Falha em abrir o arquivo %s com o modo %s\n", path, mode);  
        exit(0);  
    }  
}
```



Exemplo: escrita

```
void copia_arquivo(FILE *fin, FILE *fout) {  
    char c;  
    while (fscanf(fin, "%c", &c) != EOF)  
        fprintf(fout, "%c", c);  
}
```




Exemplo: escrita

```
int main(int argc, char *argv[]) {
    testa_parametros(argc);
    FILE *fin = fopen(argv[1], "r");
    FILE *fout = fopen(argv[2], "w");
    testa_abertura(fin, argv[1], "r");
    testa_abertura(fout, argv[2], "w");
    copia_arquivo(fin, fout);
    fclose(fin);
    fclose(fout);
    return 0;
}
```



Exemplo: escrita

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void testa_parametros(int argc) {
5      if (argc != 3) {
6          printf("Uso: ./executavel <arquivo_entrada> <arquivo_saida>");
7          exit(0);
8      }
9  }
10
11 void testa_abertura(FILE *fp, const char *path, const char *mode) {
12     if (fp == NULL) {
13         printf("Falha em abrir o arquivo %s com o modo %s\n", path, mode);
14         exit(0);
15     }
16 }
17
18 void copia_arquivo(FILE *fin, FILE *fout) {
19     char c;
20     while (fscanf(fin, "%c", &c) != EOF)
21         fprintf(fout, "%c", c);
22 }
```



Exemplo: escrita

```
23
24 int main(int argc, char *argv[]) {
25     testa_parametros(argc);
26     FILE *fin = fopen(argv[1], "r");
27     FILE *fout = fopen(argv[2], "w");
28     testa_abertura(fin, argv[1], "r");
29     testa_abertura(fout, argv[2], "w");
30     copia_arquivo(fin, fout);
31     fclose(fin);
32     fclose(fout);
33     return 0;
34 }
```



Sumário

4 Exemplos



Exemplo: tamanho do arquivo

Problema

Trocar todas as vogais minúsculas por maiúsculas em um arquivo texto.



Exemplo: troca de minúsculas por maiúsculas

- O programa a seguir troca as vogais minúsculas por maiúsculas.
- A estratégia é ler cada caractere e verificar se é uma vogal.
- Em caso afirmativo, devemos retroceder o indicador de posição de uma unidade e sobrescrever a vogal minúscula pela maiúscula correspondente.
- Para ler um caractere, podemos utilizar a função `fgetc` e para devolver um caractere, retrocedendo uma posição do indicador, usamos a função `ungetc`. Além disso, podemos utilizar a função `fputc`.
- Também poderíamos utilizar `fprintf` e `fscanf` para ler e escrever os caracteres.



Exemplo: troca de minúsculas por maiúsculas

- O caminho do arquivo a ser modificado será obtido pela linha de comando.
- Utilizaremos o modo `r+`, já que queremos conservar o seu conteúdo e modificar apenas as vogais.
- Utilizaremos a função `toupper` do cabeçalho `ctype.h` para converter uma vogal na sua versão maiúscula.



Exemplo: troca de minúsculas por maiúsculas

```
void testa_parametros(int argc) {  
    if (argc != 2) {  
        printf("Uso: ./executavel <arquivo>");  
        exit(0);  
    }  
}
```




Exemplo: troca de minúsculas por maiúsculas

```
void testa_abertura(FILE *fp, const char *path, const char *mode) {  
    if (fp == NULL) {  
        printf("Falha em abrir o arquivo %s com o modo %s\n", path, mode);  
        exit(0);  
    }  
}
```



Exemplo: troca de minúsculas por maiúsculas

```
int is_lowercase_vowel(int c) {  
    return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';  
}
```



Exemplo: troca de minúsculas por maiúsculas

```
void troca_vogal(FILE *fp) {  
    int c;  
    while ((c = fgetc(fp)) != EOF) {  
        if (is_lowercase_vowel(c)) {  
            ungetc(c, fp);  
            fputc(toupper(c), fp);  
        }  
    }  
}
```



Exemplo: troca de minúsculas por maiúsculas

```
1  #include <ctype.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  void testa_parametros(int argc) {
6      if (argc != 2) {
7          printf("Uso: ./executavel <arquivo>");
8          exit(0);
9      }
10 }
11
12 void testa_abertura(FILE *fp, const char *path, const char *mode) {
13     if (fp == NULL) {
14         printf("Falha em abrir o arquivo %s com o modo %s\n", path, mode);
15         exit(0);
16     }
17 }
18
19 int is_lowercase_vowel(int c) {
20     return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';
21 }
22
```



Exemplo: troca de minúsculas por maiúsculas

```
23 void troca_vogal(FILE *fp) {
24     int c;
25     while ((c = fgetc(fp)) != EOF) {
26         if (is_lowercase_vowel(c)) {
27             ungetc(c, fp);
28             fputc(toupper(c), fp);
29         }
30     }
31 }
32
33 int main(int argc, char *argv[]) {
34     testa_parametros(argc);
35     FILE *fp = fopen(argv[1], "r+");
36     testa_abertura(fp, argv[1], "r+");
37     troca_vogal(fp);
38     fclose(fp);
39     return 0;
40 }
```



Exemplo: tamanho do arquivo

Problema

Recuperar o tamanho total de um arquivo em bytes.



Exemplo: tamanho do arquivo

- Para identificar o tamanho total do arquivo, podemos posicionar o indicador de posição no final do arquivo e perguntar qual o valor daquela posição.
- Utilizaremos as funções `fseek` e `ftell`.



Exemplo: tamanho do arquivo

fseek

```
int fseek(FILE* fp, long int offset, int origin)
```

- `fp` : o ponteiro para arquivo.
- `offset` : o deslocamento a ser realizado. Se positivo, o indicador de posição avança.
- `origin` : de onde o deslocamento irá partir. Temos três opções:
 - ▶ `SEEK_SET` : do início do arquivo.
 - ▶ `SEEK_CUR` : do indicador de posição atual.
 - ▶ `SEEK_END` : do final do arquivo.
- Retorno: zero em caso de sucesso e um número negativo em caso de falha.



Exemplo: tamanho do arquivo

Exemplos:

- `fseek(fp,10,SEEK_SET)` : posiciona o indicador no byte 10 do arquivo.
- `fseek(fp,-10,SEEK_END)` : posiciona o indicador de posição 10 bytes antes do fim do arquivo.
- `fseek(fp,1,SEEK_CUR)` : avança o indicador de posição em 1 byte.



Exemplo: tamanho do arquivo

- Uma vez que o indicador de posição esteja no fim do arquivo, só precisamos obter o valor desse indicador.
- Utilizaremos a função `ftell`.



Exemplo: tamanho de arquivo

ftell

```
long int ftell(FILE* fp);
```

- `fp`: o ponteiro para o arquivo.
- Retorno: a posição atual do indicador de posição do arquivo.



Exemplo: tamanho de arquivo

```
long int tamanho_arquivo(FILE *fp) {  
    fseek(fp, 0, SEEK_END);  
    return ftell(fp);  
}
```



Exemplo: tamanho de arquivo

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void testa_parametros(int argc) {
5      if (argc != 2) {
6          printf("Uso: ./executavel <arquivo>");
7          exit(0);
8      }
9  }
10
11 void testa_abertura(FILE *fp, const char *path, const char *mode) {
12     if (fp == NULL) {
13         printf("Falha em abrir o arquivo %s com o modo %s\n", path, mode);
14         exit(0);
15     }
16 }
17
18 long int tamanho_arquivo(FILE *fp) {
19     fseek(fp, 0, SEEK_END);
20     return ftell(fp);
21 }
22
```



Exemplo: tamanho de arquivo

```
23  int main(int argc, char *argv[]) {  
24      testa_parametros(argc);  
25      FILE *fp = fopen(argv[1], "r");  
26      testa_abertura(fp, argv[1], "r");  
27      long int tam = tamanho_arquivo(fp);  
28      printf("O arquivo tem %ld bytes", tam);  
29      fclose(fp);  
30      return 0;  
31  }
```