

Arquivos Texto

Programação de Computadores 1

Prof. Daniel Saad Nogueira Nunes



**INSTITUTO
FEDERAL**

Brasília

Campus
Taguatinga

Sumário

Introdução

Conceitos

Arquivos texto

Exemplos

Sumário

Introdução

Conceitos

Arquivos texto

Exemplos

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

Introdução

Conceitos

Arquivos texto

Exemplos

Sumário

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



figuras/hierarquia-memoria.png

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

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

Conceitos

Hardware

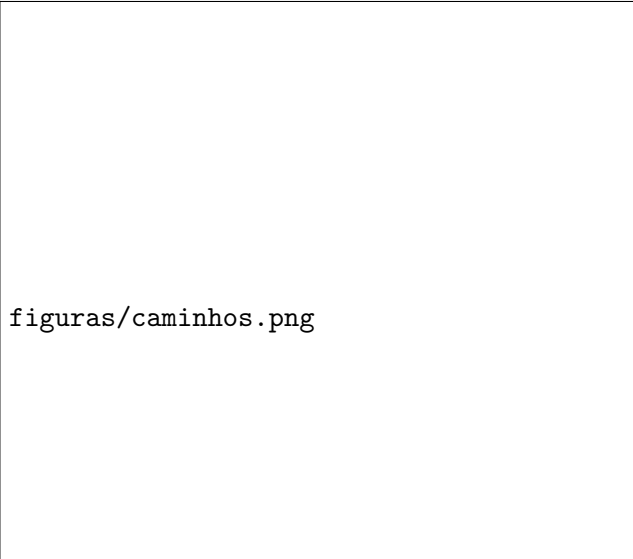
Arquivos e Diretórios

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



`figuras/caminhos.png`

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

Introdução

Conceitos

Arquivos texto

Exemplos

Arquivos Texto

- ▶ Arquivos texto 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 texto: códigos-fonte, html, arquivos de extensão `.txt`.

Sumário

Arquivos texto

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

-
- ▶ 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

Arquivos texto

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

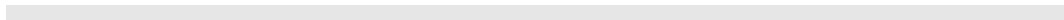
Problema

Imprimir o conteúdo de um arquivo na tela.

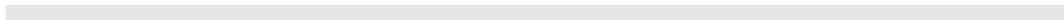
Exemplo: impressão de arquivo

- ▶ 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.

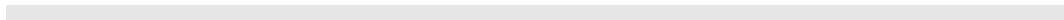
Exemplo: leitura



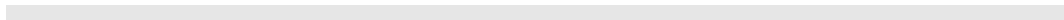
Exemplo: leitura



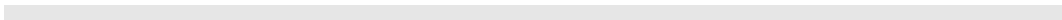
Exemplo: leitura



Exemplo: leitura



Exemplo: leitura



Exemplo: leitura



Leitura: fgets

fgets

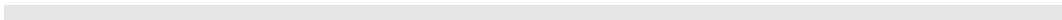
```
char* fgets (char* str, int n, FILE* fp);
```

- ▶ 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



Sumário

Arquivos texto

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

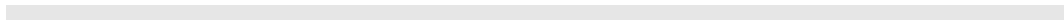
Problema

Criar uma cópia de um arquivo texto.

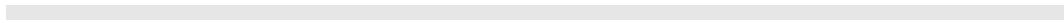
Exemplo: escrita

- ▶ 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.

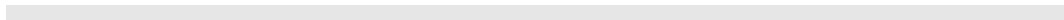
Exemplo: escrita



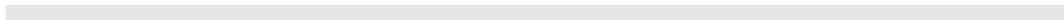
Exemplo: escrita



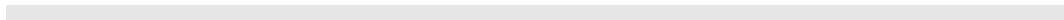
Exemplo: escrita



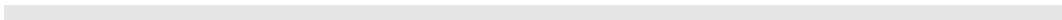
Exemplo: escrita



Exemplo: escrita



Exemplo: escrita



Sumário

Introdução

Conceitos

Arquivos texto

Exemplos

Exemplo: tamanho do arquivo

Problema

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

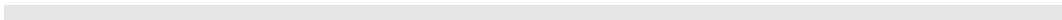
Exemplo: troca de minúsculas por maiúsculas

- ▶ 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.

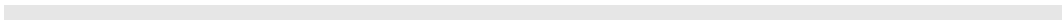
Exemplo: troca de minúsculas por maiúsculas

- ▶ 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.

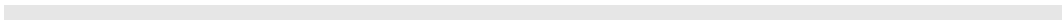
Exemplo: troca de minúsculas por maiúsculas



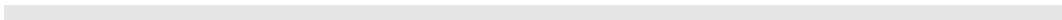
Exemplo: troca de minúsculas por maiúsculas



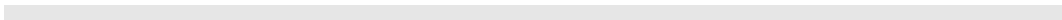
Exemplo: troca de minúsculas por maiúsculas



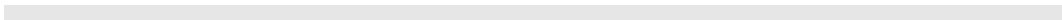
Exemplo: troca de minúsculas por maiúsculas



Exemplo: troca de minúsculas por maiúsculas



Exemplo: troca de minúsculas por maiúsculas



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`.

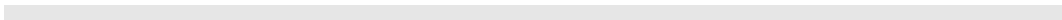
Exemplo: tamanho de arquivo

ftell

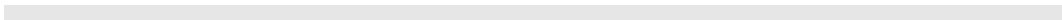
```
long int ftell(FILE* fp);
```

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

Exemplo: tamanho de arquivo



Exemplo: tamanho de arquivo



Exemplo: tamanho de arquivo

