Manipulação de Arquivos

Estrutura de Dados

Prof. Msc. Felipe Leivas Teixeira

Versão 1.0

Arquivos

- **Arquivos** são conjuntos de dados identificados por um nome e que podem ser acessados a partir de programas

Arquivos

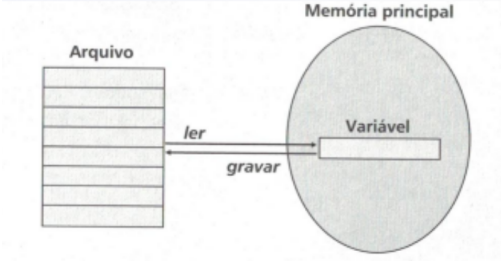
- **Arquivos** são conjuntos de dados identificados por um nome e que podem ser acessados a partir de programas
- Arquivos são armazendos fora da memória principal, de forma que continuam a existir após a finalização dos programas que os utilizam, garantindo a persistência de dados

Arquivos

- **Arquivos** são conjuntos de dados identificados por um nome e que podem ser acessados a partir de programas
- Arquivos são armazendos fora da memória principal, de forma que continuam a existir após a finalização dos programas que os utilizam, garantindo a persistência de dados
- Nos programas, o acesso aos dados dos arqquivos acontece por meio de variáveis armazenadas na memória principal

Arquivos

- **Arquivos** são conjuntos de dados identificados por um nome e que podem ser acessados a partir de programas
- Arquivos são armazendos fora da memória principal, de forma que continuam a existir após a finalização dos programas que os utilizam, garantindo a persistência de dados
- Nos programas, o acesso aos dados dos arqquivos acontece por meio de variáveis armazenadas na memória principal



Arquivos

- Os arquivos podem ser de dois tipos:

Arquivos

- Os arquivos podem ser de dois tipos:
  - **arquivos texto** - armazenam caracteres organizados em linhas. A representação binária dos caracteres segue algum código de representação de caracteres, como ASCII

Arquivos

- Os arquivos podem ser de dois tipos:
  - **arquivos texto** - armazenam caracteres organizados em linhas. A representação binária dos caracteres segue algum código de representação de caracteres, como ASCII
  - **arquivos binários** - armazenam unidades de informação em binário puro. Se o conteúdo de um arquivo binário é listado, em geralmente é incompreensível

Arquivos em C

- Para que um programa possa criar ou utilizar um arquivo preexistente, esse arquivo deve primeiro existir dentro do programa, na forma de um ponteiro do tipo FILE

Arquivos em C

- Para que um programa possa criar ou utilizar um arquivo preexistente, esse arquivo deve primeiro existir dentro do programa, na forma de um ponteiro do tipo FILE
- Um ponteiro desse tipo deve ser declarado para cada arquivo. A sintaxe da declaração é a seguinte: **FILE *<nome_ponteiro>;**

Arquivos em C

- Para que um arquivo possa ser acessado, após sua declaração, ele deve ser aberto

Arquivos em C

- Para que um arquivo possa ser acessado, após sua declaração, ele deve ser aberto
- A abertura de um arquivo em C dá-se pela associação de um ponteiro do tipo FILE, interno ao programa, com o arquivo existente fisicamente no sistema de arquivos, externo ao programa

Arquivos em C

- Para que um arquivo possa ser acessado, após sua declaração, ele deve ser aberto
- A abertura de um arquivo em C dá-se pela associação de um ponteiro do tipo FILE, interno ao programa, com o arquivo existente fisicamente no sistema de arquivos, externo ao programa
- O número máximo de arquivos abertos ao mesmo tempo no programa é definido pela constante **FOPEN_MAX** (definida na biblioteca stdio.h)

Arquivos em C

- Para que um arquivo possa ser acessado, após sua declaração, ele deve ser aberto
- A abertura de um arquivo em C dá-se pela associação de um ponteiro do tipo FILE, interno ao programa, com o arquivo existente fisicamente no sistema de arquivos, externo ao programa
- O número máximo de arquivos abertos ao mesmo tempo no programa é definido pela constante **FOPEN_MAX** (definida na biblioteca stdio.h)
- A função usada para abrir um arquivo é **fopen**. seu protótipo é:

Arquivos em C

- Para que um arquivo possa ser acessado, após sua declaração, ele deve ser aberto
- A abertura de um arquivo em C dá-se pela associação de um ponteiro do tipo FILE, interno ao programa, com o arquivo existente fisicamente no sistema de arquivos, externo ao programa
- O número máximo de arquivos abertos ao mesmo tempo no programa é definido pela constante **FOPEN_MAX** (definida na biblioteca stdio.h)
- A função usada para abrir um arquivo é **fopen**. seu protótipo é:

FILE * fopen(char * <nome_do_arquivo>, char * <modo_de_abertura>)

Arquivos em C

- Para que um arquivo possa ser acessado, após sua declaração, ele deve ser aberto
- A abertura de um arquivo em C dá-se pela associação de um ponteiro do tipo FILE, interno ao programa, com o arquivo existente fisicamente no sistema de arquivos, externo ao programa
- O número máximo de arquivos abertos ao mesmo tempo no programa é definido pela constante **FOPEN_MAX** (definida na biblioteca stdio.h)
- A função usada para abrir um arquivo é **fopen**. seu protótipo é:

  FILE * fopen(char * <nome_do_arquivo>, char * <modo_de_abertura>)

- Essa função devolve um ponteiro do tipo FILE associado ao arquivo, se a operação foi bem sucedida, ou NULL, se ocorrer um erro e o arquivo não puder ser aberto, então deve sempre ser testado se o arquivo foi aberto ou não

Arquivos em C

- Para que um arquivo possa ser acessado, após sua declaração, ele deve ser aberto
- A abertura de um arquivo em C dá-se pela associação de um ponteiro do tipo FILE, interno ao programa, com o arquivo existente fisicamente no sistema de arquivos, externo ao programa
- O número máximo de arquivos abertos ao mesmo tempo no programa é definido pela constante **FOPEN_MAX** (definida na biblioteca stdio.h)
- A função usada para abrir um arquivo é **fopen**. seu protótipo é:

FILE * fopen(char * <nome_do_arquivo>, char * <modo_de_abertura>)

- Essa função devolve um ponteiro do tipo FILE associado ao arquivo, se a operação foi bem sucedida, ou NULL, se ocorrer um erro e o arquivo não puder ser aberto, então deve sempre ser testado se o arquivo foi aberto ou não
- O nome do arquivo, deve ser o caminho completo, em qual drive e pasta ele se encontra, mas é dispensável se o arquivo estiver no mesmo drive e na mesma pasta que contém o programa

Arquivos em C

- O modo de abertura de um arquivo, define:

Arquivos em C

- O modo de abertura de um arquivo, define:
  - o tipo do arquivo, se binário ou texto;

Arquivos em C

- O modo de abertura de um arquivo, define:
  - o tipo do arquivo, se binário ou texto;
  - as operações que podem ser realizadas sobre o arquivo: só leitura, só escrita ou leitura e escrita;

Arquivos em C

- O modo de abertura de um arquivo, define:
  - o tipo do arquivo, se binário ou texto;
  - as operações que podem ser realizadas sobre o arquivo: só leitura, só escrita ou leitura e escrita;
  - se for encontrado um arquivo com mesmo nome, se esse deve ser eliminado e um novo arquivo deve ser criado em seu lugar;

Arquivos em C

- O modo de abertura de um arquivo, define:
  - o tipo do arquivo, se binário ou texto;
  - as operações que podem ser realizadas sobre o arquivo: só leitura, só escrita ou leitura e escrita;
  - se for encontrado um arquivo com mesmo nome, se esse deve ser eliminado e um novo arquivo deve ser criado em seu lugar;
  - a partir de onde os dados devem ser gravados no arquivo: se a partir do início, ou se a partir do fim válido no momento da abertura.

Arquivos em C

- Os modos de abertura para arquivos de texto são: **r, w, r+, w+, a e a+**

Arquivos em C

- Os modos de abertura para arquivos de texto são: **r, w, r+, w+, a e a+**
- Os modos de abertura para arquivos binários são os mesmos, acrescidos do sufixo b: **rb, wb, r+b(rb+), w+b(wb+), ab, a+b(ab+)**

Arquivos em C

- Os modos de abertura para arquivos de texto são: **r, w, r+, w+, a e a+**
- Os modos de abertura para arquivos binários são os mesmos, acrescidos do sufixo b: **rb, wb, r+b(rb+), w+b(wb+), ab, a+b(ab+)**
- Os modos só com **r** permitem apenas leitura, já os modos só com **a** ou **w** permitem apenas escrita e os modos com **+** permitem tanto leitura quanto escrita

Arquivos em C

- Os modos de abertura para arquivos de texto são: **r, w, r+, w+, a e a+**
- Os modos de abertura para arquivos binários são os mesmos, acrescidos do sufixo b: **rb, wb, r+b(rb+), w+b(wb+), ab, a+b(ab+)**
- Os modos só com **r** permitem apenas leitura, já os modos só com **a** ou **w** permitem apenas escrita e os modos com **+** permitem tanto leitura quanto escrita
- A tabela abaixo apresenta um quadro resumo dos modos de abertura

Modo de abertura	Arquivo existe	Arquivo não existe	Permissão de leitura	Permissão de escrita
r / rb	ok	erro	X	-
w / wb	cria novo, antigo eliminado	cria	-	X
r+ / r+b / rb+	ok	erro	X	X
w+ / w+b / wb+	cria novo antigo eliminado	cria	X	X
a / ab	novos dados acrescentados ao final	cria	-	X
a+ / a+b / ab+	novos dados acrescentados ao final	cria	X	X

Arquivos em C

- No processamento de arquivos de texto linha por linha, são utilizadas as funções **fgets** e **fputs**

Arquivos em C

- No processamento de arquivos de texto linha por linha, são utilizadas as funções **fgets** e **fputs**
  - **fgets** - A função fgets pode ser utilizada para a leitura de linhas, pois lê de um arquivo conjunto de caracteres até um tamanho máximo informado e armazena os caracteres lidos em uma variável string (vetor de caracteres). O protótipo da função fgets é:

Arquivos em C

- No processamento de arquivos de texto linha por linha, são utilizadas as funções **fgets** e **fputs**
  - **fgets** - A função fgets pode ser utilizada para a leitura de linhas, pois lê de um arquivo conjunto de caracteres até um tamanho máximo informado e armazena os caracteres lidos em uma variável string (vetor de caracteres). O protótipo da função fgets é:

char * fgets(char *str, int tamanho, FILE *fp)

Arquivos em C

- No processamento de arquivos de texto linha por linha, são utilizadas as funções **fgets** e **fputs**
  - **fgets** - A função fgets pode ser utilizada para a leitura de linhas, pois lê de um arquivo conjunto de caracteres até um tamanho máximo informado e armazena os caracteres lidos em uma variável string (vetor de caracteres). O protótipo da função fgets é:

    char * fgets(char *str, int tamanho, FILE *fp)
  - **fputs** - A função fputs escreve uma string (ou linha) em um arquivo. O protótipo da função fputs é:

Arquivos em C

- No processamento de arquivos de texto linha por linha, são utilizadas as funções **fgets** e **fputs**
  - **fgets** - A função fgets pode ser utilizada para a leitura de linhas, pois lê de um arquivo conjunto de caracteres até um tamanho máximo informado e armazena os caracteres lidos em uma variável string (vetor de caracteres). O protótipo da função fgets é:

    char * fgets(char *str, int tamanho, FILE *fp)
  - **fputs** - A função fputs escreve uma string (ou linha) em um arquivo. O protótipo da função fputs é:

    char * fputs(char *str, FILE *fp)

Arquivos em C

- No processamento de arquivos de texto linha por linha, são utilizadas as funções **fgets** e **fputs**
  - **fgets** - A função fgets pode ser utilizada para a leitura de linhas, pois lê de um arquivo conjunto de caracteres até um tamanho máximo informado e armazena os caracteres lidos em uma variável string (vetor de caracteres). O protótipo da função fgets é:

    char * fgets(char *str, int tamanho, FILE *fp)
  - **fputs** - A função fputs escreve uma string (ou linha) em um arquivo. O protótipo da função fputs é:

    char * fputs(char *str, FILE *fp)
- Existe ainda a possibilidade de processar arquivos de texto caractere por caractere. Para isso são utilizadas as funções **getc** e **putc**

Arquivos em C

- Para detectar o final de um arquivo é necessário efetuar uma tentativa de leitura além do seu final

Arquivos em C

- Para detectar o final de um arquivo é necessário efetuar uma tentativa de leitura além do seu final
- Assim, se um arquivo vazio for aberto, só se descobrirá que ele está vazio após uma primeira tentativa frustrada de leitura sobre o mesmo

Arquivos em C

- Para detectar o final de um arquivo é necessário efetuar uma tentativa de leitura além do seu final
- Assim, se um arquivo vazio for aberto, só se descobrirá que ele está vazio após uma primeira tentativa frustrada de leitura sobre o mesmo
- A condição de final de arquivo pode ser verificada com o auxilio da função **feof**, cujo o protótipo é:

Arquivos em C

- Para detectar o final de um arquivo é necessário efetuar uma tentativa de leitura além do seu final
- Assim, se um arquivo vazio for aberto, só se descobrirá que ele está vazio após uma primeira tentativa frustrada de leitura sobre o mesmo
- A condição de final de arquivo pode ser verificada com o auxilio da função **feof**, cujo o protótipo é:

int feof(FILE *fp)

Arquivos em C

- Para detectar o final de um arquivo é necessário efetuar uma tentativa de leitura além do seu final
- Assim, se um arquivo vazio for aberto, só se descobrirá que ele está vazio após uma primeira tentativa frustrada de leitura sobre o mesmo
- A condição de final de arquivo pode ser verificada com o auxilio da função **feof**, cujo o protótipo é:

int feof(FILE *fp)

- A função **feof** devolve o valor verdadeiro (não zero) caso o final do arquivo tenha sido atingido, ou falso (zero) em caso contrário

Arquivos em C

- Após a abertura e a utilização de um arquivo, o mesmo deve ser fechado

Arquivos em C

- Após a abertura e a utilização de um arquivo, o mesmo deve ser fechado
- A função usada para fechar um arquivo é a **fclose**. Seu protótipo é:

Arquivos em C

- Após a abertura e a utilização de um arquivo, o mesmo deve ser fechado
- A função usada para fechar um arquivo é a **fclose**. Seu protótipo é:

int fclose(<ponteiro_para_um_arquivo>)

Arquivos em C

- Após a abertura e a utilização de um arquivo, o mesmo deve ser fechado
- A função usada para fechar um arquivo é a **fclose**. Seu protótipo é:

  int fclose(<ponteiro_para_um_arquivo>)
- Se a função for executada corretamente, será devolvido zero e, em caso de erro, será devolvida a constante EOF

EXEMPLO ARQUIVOS DE TEXTO

Arquivos em C

- O processamento de arquivos binários acontece com base em número de bytes

Arquivos em C

- O processamento de arquivos binários acontece com base em número de bytes
- No processamento de arquivos binários, são utilizadas as funções **fwrite** e **fread**

Arquivos em C

- O processamento de arquivos binários acontece com base em número de bytes
- No processamento de arquivos binários, são utilizadas as funções **fwrite** e **fread**
  - **fwrite** - A função fwrite grava blocos de bytes em um arquivo a partir do elemento corrente. Seu protótipo é:

Arquivos em C

- O processamento de arquivos binários acontece com base em número de bytes
- No processamento de arquivos binários, são utilizadas as funções **fwrite** e **fread**
  - **fwrite** - A função fwrite grava blocos de bytes em um arquivo a partir do elemento corrente. Seu protótipo é:

    fwrite(&varbuffer, numbytes, quant, FILE *fp)
  - onde **&varbuffer** é de onde os dados devem ser obtidos; **numbytes** é o tamanho, em bytes, da unidade a ser gravada; **quant** são quantas unidades deverão ser gravadas; e **FILE *fp** é o ponteiro para o arquivo que está sendo gravado.

Arquivos em C

- O processamento de arquivos binários acontece com base em número de bytes
- No processamento de arquivos binários, são utilizadas as funções **fwrite** e **fread**
  - **fwrite** - A função fwrite grava blocos de bytes em um arquivo a partir do elemento corrente. Seu protótipo é:

    fwrite(&varbuffer, numbytes, quant, FILE *fp)
  - onde **&varbuffer** é de onde os dados devem ser obtidos; **numbytes** é o tamanho, em bytes, da unidade a ser gravada; **quant** são quantas unidades deverão ser gravadas; e **FILE *fp** é o ponteiro para o arquivo que está sendo gravado.
  - **fread** - A função fread lê para a memória principal blocos de bytes de um arquivo a partir do elemento corrente. Seu protótipo é:

Arquivos em C

- O processamento de arquivos binários acontece com base em número de bytes
- No processamento de arquivos binários, são utilizadas as funções **fwrite** e **fread**
  - **fwrite** - A função fwrite grava blocos de bytes em um arquivo a partir do elemento corrente. Seu protótipo é:

    fwrite(&varbuffer, numbytes, quant, FILE *fp)
  - onde **&varbuffer** é de onde os dados devem ser obtidos; **numbytes** é o tamanho, em bytes, da unidade a ser gravada; **quant** são quantas unidades deverão ser gravadas; e **FILE *fp** é o ponteiro para o arquivo que está sendo gravado.
  - **fread** - A função fread lê para a memória principal blocos de bytes de um arquivo a partir do elemento corrente. Seu protótipo é:

    fread(&varbuffer, numbytes, quant, FILE *fp)

Arquivos em C

- O processamento de arquivos binários acontece com base em número de bytes
- No processamento de arquivos binários, são utilizadas as funções **fwrite** e **fread**
  - **fwrite** - A função fwrite grava blocos de bytes em um arquivo a partir do elemento corrente. Seu protótipo é:

    fwrite(&varbuffer, numbytes, quant, FILE *fp)
  - onde **&varbuffer** é de onde os dados devem ser obtidos; **numbytes** é o tamanho, em bytes, da unidade a ser gravada; **quant** são quantas unidades deverão ser gravadas; e **FILE *fp** é o ponteiro para o arquivo que está sendo gravado.
  - **fread** - A função fread lê para a memória principal blocos de bytes de um arquivo a partir do elemento corrente. Seu protótipo é:

    fread(&varbuffer, numbytes, quant, FILE *fp)
  - onde **&varbuffer** referencia o local em que os dados lidos devem ser armazenados; **numbytes** é o tamanho , em bytes, da unidade lida; **quant** especifica quantas unidades de numbytes deverão ser lidas; e **FILE *fp** é o ponteiro para o arquivo que está sendo lido.

EXEMPLO ARQUIVOS BINÁRIOS

Exercício

1. Gere, linha por linha, um arquivo de texto com o seu conteúdo digitado pelo usuário durante a execução. Agravação do arquivo de ser parada quando o usuário digitar FIM. Apresente, ao final do processamento, o total de linhas gravadas, no arquivo.
2. Usando o arquivo de texto produzido no exercício anterior, faça um programa que leia esse arquivo e imprima na tela apenas as linhas pares do arquivo.
3. Usando o arquivo de texto produzido no exercício 1, faça um programa que transforme o arquivo de texto em um arquivo binário.

Manipulação de Arquivos

Estrutura de Dados

Prof. Msc. Felipe Leivas Teixeira

Versão 1.0