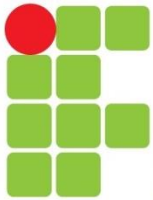

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
MATO GROSSO DO SUL

Algoritmos II

Arquivos

Prof. Rogério Alves dos Santos Antoniassi
Prof. Alex Fernando de Araujo

Introdução

- Com a manipulação de arquivos é possível efetuar a persistência de dados de forma não volátil;

- Dois tipos principais:
  - Texto: Texto simples (ASCII, UTF-8);
  - Binário: Dados em formato binário (bytes);

- Manipulação com ponteiros do tipo FILE*.

Biblioteca necessária

- #include <stdio.h>

- Contém funções:
  - fopen, fclose, fprintf, fscanf;
  - fread, fwrite, fseek, ftell, rewind.

Funções para manipulação de arquivos

- **fopen():** Abre um arquivo. Retorna um ponteiro para o arquivo (FILE*).
  - Sintaxe: FILE *fopen(const char *filename, const char *mode);
  - mode especifica o modo de abertura (leitura, escrita, etc.).
- **fclose():** Fecha um arquivo. Libera os recursos alocados pelo arquivo.
  - Sintaxe: int fclose(FILE *fp);
- **fprintf():** Escreve dados formatados no arquivo. Similar ao printf(), mas escreve no arquivo.
  - Sintaxe: int fprintf(FILE *fp, const char *format, ...);
- **fscanf():** Lê dados formatados do arquivo. Similar ao scanf(), mas lê do arquivo.
  - Sintaxe: int fscanf(FILE *fp, const char *format, ...);
- **fgetc():** Lê um caractere do arquivo. Sintaxe: int fgetc(FILE *fp);
- **fputc():** Escreve um caractere no arquivo. Sintaxe: int fputc(int c, FILE *fp);
- **fgets():** Lê uma string do arquivo. Sintaxe: char *fgets(char *str, int n, FILE *fp);
- **fputs():** Escreve uma string no arquivo. Sintaxe: int fputs(const char *str, FILE *fp);
- **fread():** Lê um bloco de dados do arquivo. Sintaxe: size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
- **fwrite():** Escreve um bloco de dados no arquivo. Sintaxe: size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);

Modos de abertura de arquivos fopen()

- **r:** Abre para leitura. Obs: O arquivo deve existir.
- **w:** Abre para escrita. Obs: Se o arquivo existir, seu conteúdo é apagado; se não existir, é criado.
- **a:** Abre para adição (escrita no final do arquivo). Obs: Se o arquivo não existir, é criado.
- **r+:** Abre para leitura e escrita. Obs: O arquivo deve existir.
- **w+:** Abre para leitura e escrita. Obs: Se o arquivo existir, seu conteúdo é apagado; se não existir, é criado."
- **a+:** "Abre para adição e leitura. Obs: Se o arquivo não existir, é criado.
- **"b":** Pode ser combinado com os modos anteriores para abrir arquivos binários (ex: rb, wb).

Abrindo e fechando arquivos

```
FILE *fp;
fp = fopen("exemplo.txt", "r");
if (fp == NULL) {
    /* erro */
}
fclose(fp);
```

Obs: fopen() retorna NULL no caso de falha na abertura.

Escrita em arquivo

```c
#include<stdio.h>

 int main() {

  FILE *arquivo;
  arquivo = fopen("dados.txt", "w");


  if (arquivo == NULL) {
    printf("Erro ao abrir o arquivo.\n");
    return 1;
  }

  fprintf(arquivo, "Nome: Alex\n");
  fprintf(arquivo, "Idade: 30\n");

  fclose(arquivo);

  return 0;

}
```

Obs: poderia ser:
fprintf(fp, "Nome: %s\n", nome);

Leitura em arquivo

```c
#include <stdio.h>
 int main() {
  FILE *arquivo;
  char linha[100];

  arquivo = fopen("dados.txt", "r");

  if (arquivo == NULL) {
    printf("Erro ao abrir o arquivo.\n");
    return 1;
  }

  while (fgets(linha, sizeof(linha), arquivo) != NULL) {
    printf("%s", linha);
  }

  fclose(arquivo);
  return 0;
 }
```

Obs: Poderia ser:
fscanf(fp, "%s", nome);

Escrita e leitura em arquivo binário

```c
#include <stdio.h>

 struct Pessoa {
  char nome[50];
  int idade;
  float peso;
 };

 int main() {
  FILE *arquivo;
  struct Pessoa p1 = {"Alex", 30, 70.5};
  struct Pessoa p2;

  // Escrita
  arquivo = fopen("pessoa.dat", "wb");
  fwrite(&p1, sizeof(struct Pessoa), 1, arquivo);
  fclose(arquivo);

  // Leitura
  arquivo = fopen("pessoa.dat", "rb");
  fread(&p2, sizeof(struct Pessoa), 1, arquivo);
  fclose(arquivo);

  printf("Nome: %s\n", p2.nome);
  printf("Idade: %d\n", p2.idade);
  printf("Peso: %.2f\n", p2.peso);

  return 0;
}
```

Movimentando no arquivo: fseek()

- Função que permite que você defina a posição de leitura/escrita em um arquivo.

- Útil para acessar dados em locais específicos do arquivo
  - Sintaxe: int fseek(FILE *stream, long int offset, int whence);
    - stream: Ponteiro para o arquivo.
    - offset: Deslocamento em bytes a partir de uma posição de referência.
    - whence: Ponto de referência para o deslocamento.

  - Valores para whence:
    - SEEK_SET (ou 0): Início do arquivo.
    - SEEK_CUR (ou 1): Posição atual.
    - SEEK_END (ou 2): Fim do arquivo.

Exemplo

```c
#include <stdio.h>

int main() {
    FILE *arquivo;
    arquivo = fopen("teste.txt", "r");

    if (arquivo == NULL) {
      perror("Erro ao abrir o arquivo");
      return 1;
    }

    // Move para o 10º byte do arquivo
    fseek(arquivo, 10, SEEK_SET);

    char caractere;
    caractere = fgetc(arquivo);
    printf("Caractere na posição 10: %c\n", caractere);

    // Move 5 bytes à frente da posição atual
    fseek(arquivo, 5, SEEK_CUR);

    // Move 3 bytes antes do final do arquivo
    fseek(arquivo, -3, SEEK_END);
    fclose(arquivo);
    return 0;
}
```

Obs: perror(): Exibe uma mensagem de erro na saída de erro padrão (stderr).

Obtendo a posição atual: ftell()

- Útil para retornar a posição atual de leitura/escrita em um arquivo, do seu início. Obs: Em bytes.
  - Sintaxe: long int ftell(FILE *stream);
  - Com isso, é possível guardar a posição atual para utilizá-la mais tarde.

Exemplo

```c
#include <stdio.h>

int main() {
    FILE *arquivo;
    arquivo = fopen("teste.txt", "r");

    if (arquivo == NULL) {
        perror("Erro ao abrir o arquivo");
        return 1;
    }

    fseek(arquivo, 10, SEEK_SET);

    long int posicao = ftell(arquivo);
    printf("Posição atual: %ld\n", posicao);  // Saída: Posição atual: 10

    fclose(arquivo);
    return 0;
}
```

Voltando para o início: rewind()

- Responsável por reposicionar o indicador de posição no início.
  Obs: Poderia ser feito com fseek.

```c
#include <stdio.h>

int main() {
    FILE *arquivo;
    arquivo = fopen("teste.txt", "r");

    if (arquivo == NULL) {
        perror("Erro ao abrir o arquivo");
        return 1;
    }

    fseek(arquivo, 10, SEEK_SET);
    printf("Posição antes de rewind: %ld\n", ftell(arquivo));

    rewind(arquivo);
    printf("Posição depois de rewind: %ld\n", ftell(arquivo));  //Posição após o rewind: 0

    fclose(arquivo);
    return 0;
}
```

Excluindo arquivos

- **remove()**
- Sintaxe: int remove(const char *filename);
- Obs: Retorna 0 se tudo correr bem.

Renomeando arquivos

- **rename()**

- rename(”nomearquivo.txt", "novonome.txt");

Verificando o fim do arquivo: feof()

- Verifica se o indicador de fim de arquivo está definido.
- Diferente de zero se estiver no fim do arquivo;
- Caso contrário, zero.
- Sintaxe: int feof(FILE *stream);
- Obs: Não utilize para controlar loops de leitura, utilize o retorno das funções de leitura.

Exemplo

```c
#include <stdio.h>

int main() {
    FILE *arquivo;
    char linha[100];

    arquivo = fopen("teste.txt", "r");
    if (arquivo == NULL) {
        perror("Erro ao abrir o arquivo");
        return 1;
    }

    while (fgets(linha, sizeof(linha), arquivo) != NULL) {
        printf("%s", linha);
    }

    fclose(arquivo);
    return 0;
}
```

Controlando o loop de leitura sem o feof()

Verificando erros: ferror()

- Verifica se ocorreu algum erro durante uma manipulação de arquivo.
- Diferente de zero se ocorreu erro;
- Zero, caso não tenha ocorrido nenhum erro;
- Sintaxe: int ferror(FILE *stream);
  - Importante utilizar após leitura e escrita para verificação de erros.

Exemplo

```c
#include <stdio.h>

int main() {
    FILE *arquivo;
    arquivo = fopen("teste.txt", "r");

    if (arquivo == NULL) {
        perror("Erro ao abrir o arquivo");
        return 1;
    }

    //Tentativa de escrita em um arquivo aberto para leitura (erro)
    fprintf(arquivo, "Algum texto");
    if (ferror(arquivo)) {
        perror("Erro ao escrever no arquivo");
    }

    fclose(arquivo);
    return 0;
}
```

Exemplo de busca de dados em arquivo

```
while(fscanf(fp, "%s", palavra) != EOF) {
  if(strcmp(palavra, "procura") == 0)
    printf("Palavra encontrado!");
}
```

strcmp é uma função definida em <string.h>
Efetua a comparação de duas strings.

Referências

- ASCENCIO, Ana Fernanda Gomes; DE CAMPOS, Edilene Aparecida Veneruchi. **Fundamentos da programação de computadores**. Pearson Educación, 2008.
- BACKES, André. ***Linguagem C: completa e descomplicada***. 2. ed., 4. imp. Rio de Janeiro: LTC, 2023.
- CORMEN, Thomas H. et al. **Algoritmos: teoria e prática**. Rio de Janeiro: Elsevier, c2012.
- DAMAS, Luís. **Linguagem C**. 10. ed. Rio de Janeiro: LTC, 2007.
- EVARISTO, Jaime. **Aprendendo a programar: programando em linguagem C**. Rio de Janeiro: Book Express, 2001. 205 p.
- MEDINA, Marco. **Algoritmos e programação: teoria e prática**. São Paulo: Novatec, 2006.
- MIZRAHI, Victorine Viviane. **Treinamento em Linguagem C**. 2. ed. São Paulo: Pearson, 2008.
- LAUREANO, Marcos. **Estrutura de Dados com Algoritmos e C**. Rio de Janeiro: Brasport, 2008.