5950257 - Programação de Computadores

Aula 13 - Arquivos

Prof. Dr. Denis M. L. Martins
DCM | FFCLRP | USP

Objetivos de Aprendizagem

- Compreender o conceito de arquivo em C
- Diferenciar memória principal e memória secundária
- Abrir, ler, escrever e fechar arquivos
- Identificar erros comuns na manipulação de arquivos

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

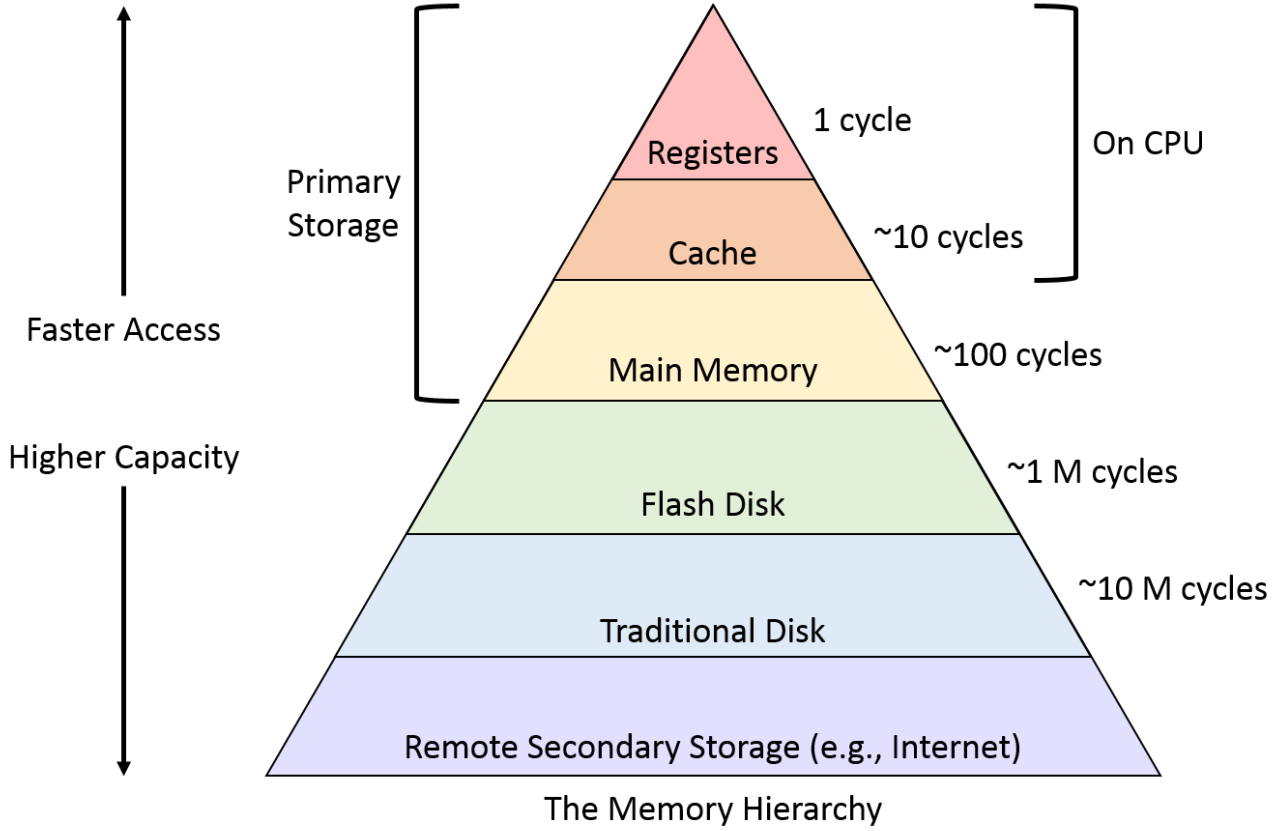
Persistência de Dados em Arquivos

Capacidade de armazenar dados de forma que eles sobrevivam ao término da execução de um programa ou sistema.

- Arquivos são uma das formas mais comuns e fundamentais de alcançar essa persistência.
- **Memória principal (RAM):** Volátil. Os dados armazenados nela são perdidos quando a energia é desligada. Arquivos, por outro lado, permanecem no dispositivo de armazenamento mesmo sem energia.
- **Memória Secundária:** Quando você salva dados em um arquivo (CSV, JSON, Excel, etc.), esses dados são gravados em um dispositivo de armazenamento físico, como Disco Rígido (HDD), Unidade de Estado Sólido (SSD) ou Pendrive/Cartão SD.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Hierarquia de memória



The Memory Hierarchy

Fonte da Imagem: CS31. Veja também: Latency by Collin Scott

Arquivos em C

Uma coleção de dados armazenados em um dispositivo não volátil (HD, SSD).

- Em C, não acessamos o arquivo diretamente como um "objeto" físico, mas sim através de um *fluxo de bytes* (File Stream).
- O tipo **FILE *** é um ponteiro para uma estrutura que contém informações sobre o estado do fluxo (posição atual, buffer, identificador do arquivo no SO).

```
FILE *fp = fopen("dados.txt", "r");
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Abertura de Arquivo

```
FILE *fp = fopen("dados.txt", "r");
```

Função `fopen()`: Abre um arquivo ou cria um novo.

Modos de Abertura:

- `"r"`: Leitura (Read). O arquivo deve existir.
- `"w"`: Escrita (Write). Cria um novo ou sobrescreve o existente.
- `"a"`: Anexar (Append). Escreve no final do arquivo sem apagar o conteúdo anterior.
- `"r+"`, `"w+"`, `"a+"`: Modos mistos de leitura e escrita.

Abertura de Arquivo (cont.)

Sempre verifique se o ponteiro é **NULL** após o **fopen**. Se for **NULL**, a abertura falhou (arquivo não encontrado, falta de permissão, etc.).

```c
FILE *fp = fopen("dados.txt", "r");

if (fp == NULL) {
    perror("Erro ao abrir arquivo");
    return 1; // Saída segura
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Fechando Arquivos: `fclose()`

O sistema operacional mantém um limite de arquivos abertos.

- Não fechar o arquivo pode causar **Memory Leak** (no buffer) e perda de dados.

Exemplo:

```c
FILE *arq = fopen("dados.txt", "w");
if (arq == NULL) {
    printf("Erro ao abrir o arquivo!\n");
    return 1;
}

// Operações com o arquivo aberto

fclose(arq); // Fecha o arquivo para liberar recursos.
```

> Sempre feche os arquivos que abrir. Use `fclose()` antes de sair do escopo ou ao detectar erro.

Exemplo: Leitura de Arquivo

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

int main() {
    char caractere;

    FILE *arq = fopen("meu_texto.txt", "r");
    if (arq == NULL) {
        printf("\nERRO: Não foi possível abrir o arquivo.");
        printf("\nVerifique se o arquivo existe.\n");
        return 1;
    }
    // O loop continua enquanto o caractere lido for diferente de EOF.
    while ((caractere = fgetc(arq)) != EOF) {
        printf("%c", caractere); // Imprime o caractere na tela
    }
    fclose(arq); // Libera o recurso do sistema operacional.

    return 0;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo: Escrevendo um Dado num Arquivo

```c
#include <stdio.h>

int main() {
    float dose = 1.75;
    FILE *arquivo = fopen("dose.txt", "w");

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

    fprintf(arquivo, "Dose medida: %.2f mSv\n", dose);
    fclose(arquivo);

    return 0;
}
```

Exemplo: Lendo um Dado de um Arquivo

Considere que o arquivo **dose.txt** possui apenas o valor **1.75** (escrito anteriormente).

```c
#include <stdio.h>

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

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

    fscanf(arquivo, "%f", &dose);
    printf("Dose lida: %.2f mSv\n", dose);
    fclose(arquivo);
    return 0;
}
```

Exemplo: Lendo Valores Linha por Linha

Considere que o arquivo **dose.txt** possui os seguintes valores separados por linha:

```
1.20
1.40
1.10
1.60
1.30
```

Podemos iterar pelo arquivo para ler os valores:

```c
#include <stdio.h>

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

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

    while (fscanf(arquivo, "%f", &dose) == 1) {
        printf("Dose: %.2f mSv\n", dose);
    }

    fclose(arquivo);

    return 0;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Formatos Comuns

TXT (Text Files):

- Arquivos de texto simples, podem conter dados formatados ou não.
- Requer análise customizada para extrair informações relevantes.

CSV (Comma Separated Values):

- Simples, amplamente utilizado para dados tabulares.
- Delimitador (vírgula, ponto e vírgula, etc.) define a separação entre colunas.

JSON (JavaScript Object Notation):

- Formato flexível para dados semiestruturados.
- Representa dados como pares chave-valor e listas.

Escrita em Arquivos: `fputc()`

A função **`fputc()`** escreve **um único caractere** em um arquivo.

Sintaxe:

```
fputc(caractere, arquivo);
```

Exemplo:

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

fputc('A', arquivo);

fclose(arquivo);
```

Escrita em Arquivos: `fputs()`

A função **`fputs()`** escreve uma **string** em um arquivo.

Sintaxe:

```
fputs(string, arquivo);
```

Exemplo:

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

fputs("Controle de qualidade\n", arquivo);

fclose(arquivo);
```

> **Observação**: **`fputs()`** não adiciona **`\n`** automaticamente.

Escrita em Arquivos: `fwrite()`

A função **`fwrite()`** escreve dados em formato **binário** no arquivo.

Sintaxe:

```
fwrite(&variavel, tamanho, quantidade, arquivo);
```

Exemplo:

```
FILE *arquivo = fopen("dados.bin", "wb");
float dose = 1.75;

fwrite(&dose, sizeof(float), 1, arquivo);

fclose(arquivo);
```

> **Uso típico**: Escrever **`struct`** e arquivos binários.

Argumentos da função **`fwrite`**:

- **`&variavel`**: nome da variável em que a informação do arquivo será guardada.
- **`tamanho`**: número de bytes do tipo da variável
- **`quantidade`**: quantos dados do tipo especificado serão utilizados
- **`arquivo`**: ponteiro para arquivo.

Escrita em Arquivos: `fprintf()`

A função **`fprintf()`** escreve dados **formatados** em um arquivo.

Sintaxe:

```
fprintf(arquivo, "formato", valores);
```

Exemplo:

```
float dose = 1.75;
FILE *arquivo = fopen("relatorio.txt", "w");
fprintf(arquivo, "Dose medida: %.2f mSv\n", dose);
fclose(arquivo);
```

> **Uso típico**: Relatórios textuais, escrita com variáveis, saídas organizadas e legíveis.

Conclusão

Manipulação de arquivos é essencial para criar programas que armazenam dados de forma permanente.

Em C, arquivos permitem:

- registrar resultados;
- salvar medições;
- criar relatórios;
- reutilizar dados;
- construir sistemas mais completos.

Dúvidas e Discussão

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercícios e Questões

Exercício 1

Crie um programa que leia 8 temperaturas de um sensor e grave:

- todas as temperaturas;
- a maior temperatura;
- a menor temperatura;
- a média.

Arquivo de saída: **temperaturas.txt**.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício 2

Crie um programa que cadastre em um arquivo **pacientes.txt** os dados de 3 pacientes:

- nome;
- idade;
- código do exame.

Cada paciente deve ser salvo em uma linha no arquivo.

Exercício 3

Crie uma estrutura Aluno com **nome**, **matricula** e **nota**.

- O programa deve permitir ao usuário cadastrar 3 alunos e salvar os dados em um arquivo chamado **alunos.txt**.
- Em seguida, o programa deve ler esse mesmo arquivo e exibir na tela a lista de todos os alunos que possuem nota maior ou igual a 7.0.

Exercício 4

Escreva um programa que peça ao usuário o nome de um arquivo **.txt**. O programa deve abrir esse arquivo, contar quantos caracteres existem nele (incluindo espaços e quebras de linha) e exibir o resultado no console.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Questão 1

Por que é obrigatório verificar se o ponteiro retornado por **fopen()** é **NULL** antes de realizar operações?

- (A) Porque o compilador não permite a execução caso seja NULL.
- (B) Para evitar que o programa tente acessar um endereço de memória inválido, causando um "Segmentation Fault".
- (C) Porque o sistema operacional exige essa verificação para liberar o arquivo.

Questão 2

Qual a principal diferença entre os modos **"w"** e **"a"** na função **fopen()**?

- (A) O modo **"w"** apaga o conteúdo existente, enquanto o **"a"** adiciona ao final do arquivo.
- (B) O modo **"w"** só permite escrita de números, enquanto o **"a"** permite texto.
- (C) Não há diferença; ambos abrem o arquivo para escrita.

Questão 3

Em um cenário onde você precisa salvar uma lista de 100 registros de "Produtos" (cada um sendo uma **struct**), qual método é mais eficiente: gravar cada campo individualmente com **fprintf** ou usar **fwrite** para a estrutura completa?

Justifique sua resposta.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br