

Notas de Aula - AED1 – organização de arquivos  
Prof. Jefferson T. Oliva

O armazenamento de dados na memória RAM é temporário. Esses dados são perdidos quando o programa é fechado.

Arquivos são utilizados para o armazenamento de dados em dispositivos secundários (HD, pendrive, cartão de memória, CD, DVD, etc).

Sistema de arquivos é a forma de organização de dados, geralmente de forma hierárquica, em algum meio de armazenamento de dados em massa, como frequentemente feito em discos magnéticos.

Arquivo é coleção de informações armazenadas em memória secundária, podendo ser dados, programas, etc.

- Arquivos são armazenados em formato texto e binário e são gerenciados pelo sistema operacional.

Atributos de arquivos:

- Nome: o nome simbólico do arquivo é a única informação conservada em forma legível pelas pessoas.
- Identificador: este rótulo único, usualmente um número, identifica o arquivo dentro do sistema de arquivo; é o nome não legível pelas pessoas.
- Tipo: esta informação é necessária para aqueles sistemas que suportam diferentes tipos.
- Posição: esta informação é um ponteiro para um dispositivo e para a posição do arquivo naquele dispositivo.
- Tamanho: o tamanho corrente do arquivo e possivelmente o tamanho máximo permitido estão incluídos neste atributo. Tamanho do arquivo, tamanho em disco.
- Proteção: a informação de controle de acesso determina quem pode ler o arquivo, gravá-lo, executá-lo e assim por diante.
- Hora, data e identificação do usuário: estas informações podem ser conservadas em relação a data da criação, última modificação e última utilização do arquivo. Estes dados podem ser úteis para proteção, segurança e monitoramento de uso do arquivo.

Uma biblioteca da linguagem C define um conjunto completo de funções de entrada/saída: #define <stdio.h>

## **Stream**

O sistema de arquivos C é projetado para possibilitar o uso de vários dispositivos (discos, impressoras, teclados).

O sistema de arquivos transforma os dados carregados em um dispositivo lógico denominado stream, que é uma sequência de dados disponibilizados temporariamente, isto é, quando um arquivo é aberto, um ponteiro é associado a ele na memória primária.

Apesar dos dispositivos de entrada/saída serem distintos, todas as streams comportam-se de forma similar, ou seja, são amplamente independentes de dispositivo.

Existem dois tipos de streams:

- tipo texto: sequência de caracteres.
- binária: sequência de bytes.

## Arquivos

a linguagem C, um arquivo é uma sequência de bytes.

Quando um arquivo é aberto, um ponteiro é associado à sua respectiva stream.

Os arquivos são manipulados por meio de ponteiros (FILE \*):

*FILE \* arquivo = fopen(char \* nome\_arquivo, char \* modo);*

- A função fopen retorna um ponteiro do tipo FILE.
- Caso ocorra um erro na operação, a função retorna NULL.
- nome\_arquivo: é o nome e a localização do arquivo. Exemplos "teste1.txt", "/pasta/teste.txt".
- modo: determina a operação que deverá ser aplicada no arquivo (leitura, escrita, etc).

Modos de processamento de arquivos do tipo texto:

- "r": abre um arquivo texto para leitura.
- "w": abre um arquivo texto para escrita. Caso o arquivo não exista, o mesmo será criado, caso contrário, o conteúdo do mesmo é apagado.
- "a": abre um arquivo texto para escrita no final.
- "r+": abre um arquivo texto para leitura e gravação (o arquivo deve existir e pode ser alterado).
- "w+": abre um arquivo texto para leitura e gravação. Se o arquivo não existir, o mesmo é criado, caso contrário o conteúdo do mesmo é apagado.
- "a+": abre um arquivo texto para leitura e gravação. Os dados são adicionados no final do arquivo.

Após o uso do arquivo, o mesmo deve ser fechado por meio da função fclose: int resultado = fclose(FILE \* arquivo). A função fclose retorna 0, caso a operação seja bem-sucedida, ou 1, caso ocorra um erro na operação.

Funções principais para o processamento de arquivos texto:

- int getc(FILE \* arquivo) ou int fgetc(FILE \* arquivo): lê caracteres no arquivo texto.
- int putc(int ch, FILE \* arquivo) ou int fputc(int ch, FILE \* arquivo): escreve caracteres no arquivo texto. Retorna 0 caso a operação seja bem-sucedida, ou 1, caso ocorra um erro.
- char \* fgets(char \* string, int n, FILE \* arquivo): lê uma string de tamanho n no arquivo.
- char \* fputs(char \* string, FILE \* arquivo): escreve uma string no arquivo texto.
- int feof(FILE \* arquivo): retorna um valor diferente de 0 se o final do arquivo foi atingido.

Exemplo:

```
void print_file_content(char * path){
    FILE * arq = fopen(path, "r");
    char str[100];
    if (arq){
        while (!feof(arq)){
            fgets(str, 100, arq);
            printf("%s", str);
        }
        printf("\n");
        int close = fclose(arq);
        if (close == 0)
            printf("Operacao bem sucedida!\n");
        else
            printf("Falha ao fechar o arquivo.\n");
    }
}
```

```
void write_file(char * path, char * content){
    FILE * arq = fopen(path, "w");

    fputs(content, arq);

    fclose(arq);
}
```

```
void append_file(char * path, char * content, unsigned int times){
    FILE * arq = fopen(path, "a");
    int i;
    if (arq){
        for (i = 0; i < times; i++){
            fputs("\n", arq);
            fputs(content, arq);
        }
        fclose(arq);
    }
}
```

## Arquivos Binários

Arquivos texto são simples de serem manipulados e podem ser facilmente compreendidos por humanos.

Desvantagens dos arquivos texto:

- Os registros devem ser separados por algum caractere ou identificador.
- O acesso ao conteúdo é sequencial.
- Os caracteres numéricos são armazenados em formato ASCII:
  - Variáveis do tipo short, int, long, float e double possuem tamanho fixo na memória.
  - Para representação de números em arquivos texto, cada dígito é representado por um caractere equivalente (exemplo: 300000.123).

Em arquivos binários, o conteúdo é armazenado em formato binário, possibilitando o uso de menor quantidade de espaço em relação aos arquivos do tipo texto.

Não é necessário o uso de caracteres ou identificadores para separar cada registro.

O acesso ao conteúdo em arquivos binários não é sequencial, isto é, basta o conhecimento da posição em que um determinado registro esteja inserido.

Arquivos binários podem ter diversas extensões (de acordo com o programa que os lê), por exemplo ".dat".

Na linguagem C, para o processamento de arquivos binários deve ser utilizada uma biblioteca apropriada: `stdlib.h`

Os arquivos binários também são manipulados por meio de ponteiros (FILE \*): `FILE * arquivo = fopen(char * nome_arquivo, char * modo)`

Modos de processamento de arquivos binários:

- "rb": abre um arquivo binário para leitura.
- "wb": abre um arquivo binário para escrita.
- "ab": abre um arquivo binário para escrita no final.
- "r+b" ou "rb+": abre um arquivo binário para leitura e gravação (o arquivo deve existir e pode ser alterado).
- "w+b" ou "wb+": abre um arquivo binário para leitura e gravação.
- "a+b" ou "ab+": abre um arquivo binário para leitura e gravação. Os dados são adicionados no final do arquivo.

Funções principais para o processamento de arquivos binários:

- int fread(void \* buffer, int qtd\_bytes, int n\_unidades, FILE \* arquivo): lê o conteúdo de um arquivo binário.
  - buffer: região da memória onde os dados são armazenados.
  - qtd\_bytes: número de bytes que deverão ser lidos por unidade.
  - n\_unidade: quantidade de unidades que deverão ser lidas.
  - Essa função retorna a quantidade de unidades lidas efetivamente.
- int fwrite(void \* buffer, int qtd\_bytes, int n\_unidades, FILE \* arquivo): escreve dados em um arquivo binário.
- int fseek(FILE \* arquivo, int qtd\_bytes, int posicao\_origem): move a posição do cursor (qtd\_bytes) a partir de uma localização específica (posicao\_origem). Na variável posicao\_origem pode ser atribuído os seguintes valores:
  - SEEK\_SET: a partir da posição inicial do arquivo (corresponde ao valor 0).
  - SEEK\_CUR: a partir da posição atual (corresponde ao valor 1)
  - SEEK\_END: a partir do final do arquivo (corresponde ao valor 2)
- void rewind(FILE \* arquivo): retorna o cursor ao início do arquivo
- ftell(FILE \* arquivo): Retorna a posição do ponteiro.
- void rewind(FILE \* arquivo): retorna o cursor ao início do arquivo.

Exemplos:

```
typedef struct{
    long RA;
    char nome[70];
    int codigo_curso;
    float coef;
}Aluno;

void write_bin_file(char * path, Aluno content[], int n){
    FILE * arq = fopen(path, "wb");
    int i = 0;

    if (arq){
        fwrite(content, sizeof(Aluno), n, arq);

        fclose(arq);
    }else
        printf("Erro ao gerar o arquivo");
}
```

```
void read_bin_file(char * path, Aluno content[], int n){
    FILE * arq = fopen(path, "rb");

    if (arq){
        fread(content, sizeof(Aluno), n, arq);

        fclose(arq);
    }else
        printf("Erro ao abrir o arquivo");
}

int getPosicaoAluno(Aluno Aluno[], int n, char * nome){
    int i;

    for (i = 0; i < n; i++)
        if (strcmp(Aluno[i].nome, nome) == 0)
            return i;

    return -1;
}

void setModel(char * path, int position, Aluno *aluno){
    FILE * arq = fopen(path, "r+b");

    if (arq){
        fseek(arq, sizeof(Aluno) * (position), SEEK_SET);
        fwrite(aluno, sizeof(Aluno), 1, arq);
        fclose(arq);
    }
}
```

## Arquivos: Outras Funções

Outras funções para processamento de arquivos:

- int fprintf(FILE \* arquivo, char \* string, ...): possui a mesma finalidade que a função printf(), porém escreve em um arquivo texto.
- int fscanf(FILE \* arquivo, char \* string, ...): possui a mesma finalidade que a função scanf(), porém lê de um arquivo texto.
- int ferror(FILE \* arquivo): verifica se ocorreu um erro no arquivo apontado por FILE.
- int remove(FILE \* arquivo): deleta o arquivo.
- ftell(FILE \* arquivo): retorna a posição do ponteiro (em bytes percorridos no arquivo).

## **Fluxos Padrão**

stdin: entrada padrão (exemplo: teclado).

stdout: saída padrão (exemplo: vídeo).

stderr: saída de erro padrão (exemplo: vídeo).

stdaux: dispositivo de saída auxiliar (exemplo: porta serial).

stdprn: dispositivo de impressão padrão (exemplo: porta paralela).

**Ver slide 28 e exercícios no slide 29.**

## **Referências**

Deitel, H. M. e and Deitel, P. J. C: Como Programar. Pearson, 2011.

Schidildt, H. C Completo e Total. Pearson, 2011.