UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Arquivos

Aula 10

Marcos Silvano Almeida
Departamento de Computação
UTFPR Campo Mourão

Tipos de arquivos

- Na linguagem C, considera-se arquivo **texto** ou **binário**.
- Diferença ocorre no momento de ler/escrever dados no arquivo.
  - Não é uma boa prática misturar ASCII com dados binários.

- Arquivo Texto (ASCII)
  - Armazena dados em formato de **texto puro**.
  - Editável em editores de texto.
  - Fácil de compreendermos, não seguros e ocupam mais espaço.
    - 2147483647 (int, 4 bytes) VS “2147483647” (string, 10 x 1 byte)
  - Ex: documentos de office (docx, xlsx, pptx)

- Arquivo Binário
  - Armazena dados como **números binários** (0/1)
  - Não são de fácil compreensão, mais seguros e ocupam menos espaço.
  - Ex: executáveis, dados de programas, bibliotecas, etc…

Abrindo e fechando arquivo

- Para acessar arquivo:
  - fopen(): abre arquivo no modo informado
  - fclose(): fecha arquivo
- Quando abrimos o arquivo, um cursor é posicionado
  - Cursor de leitura/escrita ⇒ como em um editor de textos

```
FILE* file;
file = fopen("arquivo.txt","r"); // read
if (file == NULL) {
    printf("ERRO: Arquivo nao existe.\n");
    return 1;
}
// faz alguma coisa...

fclose(file);
```

É aconselhável sempre fechar o arquivo após utilizá-lo, mesmo que, em teste, o Sistemas Operacional o faça quando o programa terminar.

Modos de abertura de arquivo

Modo	Significado	Flags
r	Leitura. Cursor no início.	O_RDONLY
r+	Leitura e escrita. Cursor no início.	O_RDWR
w	Escrita. Trunca para zero OU cria se não existir. Cursor no início. **OBS:** escreve zeros até a posição do cursor	O_WRONLY | O_CREAT | O_TRUNC
w+	Leitura e escrita. Trunca para zero OU cria se não existir. Cursor no início. **OBS:** escreve zeros até a posição do cursor	O_RDWR | O_CREAT | O_TRUNC
a	Leitura. Acrescentar conteúdo ao final. Cria se não existir. Cursor no final do arquivo.	O_WRONLY | O_CREAT | O_APPEND
a+	Leitura e escrita. Cria se não existir. Cursor no início (se leitura) ou final (se escrita).	O_RDWR | O_CREAT | O_APPEND

Arquivo texto
Leitura e escrita de caracteres

Lendo arquivo: caractere a caractere

```
void readFileChar() {
    FILE* file;
    file = fopen("arquivo.txt","r"); // read
    if (file == NULL) {
        printf("ERRO: Arquivo nao existe.\n");
        return;
    }
    // leitura char-a-char
    char ch = fgetc(file);
    while ( ch != EOF) {
        printf("%c", ch);
        ch = fgetc(file);
    }
    fclose(file);
}
```

fgetc() faz a leitura de um char do arquivo e o retorna (ou EOF).

Lendo arquivo: caractere a caractere (versão compacta)

```
void readFileChar() {
    FILE* file;
    file = fopen("arquivo.txt","r"); // read
    if (file == NULL) {
        printf("ERRO: Arquivo nao existe.\n");
        return;
    }
    // leitura char-a-char
    char ch;
    while ( (ch = fgetc(file)) != EOF ) {
        printf("%c", ch);
    }

    fclose(file);
}
```

fgetc() faz a leitura de um char do arquivo e o retorna (ou EOF).

Escrevendo arquivo: caractere a caractere

```c
void writeFileChar() {
    FILE* file = fopen("arquivo.txt","w"); // write + truncate/create

    char text[] = "Texto de teste 1.\nTexto de teste 2.\n";
    for (int i = 0; text[i] != '\0'; i++) {
        fputc(text[i], file);
    }

    fclose(file);
}
```

Escreve char a char no arquivo. Função fputc() retorna EOF se houver algum problema.

Arquivo texto
Leitura e escrita de strings

Lendo arquivo: string

```
void readFileString() {
    FILE* file =fopen("arquivo.txt","r"); // read
    if (file == NULL) {
        printf("ERRO: Arquivo nao existe.\n");
        return;
    }
    int n = 200; // 199 chars + '\0'
    char buffer[n];
    while (fgets(buffer, n, file) != NULL) {
        printf("%s", buffer);
    }

    fclose(file);
}
```

Retorna a string lida ou EOF. fgets() lê n-1 chars da entrada e acrescenta '\0' ao final.

Escrevendo arquivo: string

```
void writeFileString() {
    FILE* file = fopen("arquivo.txt","w"); // write truncate+create

    char text[] = "Texto de teste 1.\nTexto de teste 2.\n";

    fputs(text, file);

    fclose(file);
}
```

Escreve a string no arquivo sem o '\0' ao final.

Arquivo texto
Leitura e escrita formatada

Escrevendo e lendo arquivo com fprintf() e fscanf()

```c
void writeReadFileFormatted() {
    FILE* file = fopen("arquivo.txt","w+"); // write-read trucate/create
    // Escreve com formato: deve iniciar com char
    for (int i = 0; i < 10; i++) {
        fprintf(file, " %d - %s - %f", i+1, "texto-teste", (i+1)/2.0f);
    }
    rewind(file); // mesmo que fseek(file, 0, SEEK_SET);

    // Leitura com o mesmo formato usado para escrita
    int a; float b; char str[20];
    while (fscanf(file, " %d - %s - %f", &a, str, &b) != EOF) {
        printf("Dados escritos: %2d, %s, %.2f\n", a, str, b);
    }
    fclose(file);
}
```

String “interna” não pode conter espaços

Posiciona cursor no início do arquivo.

Arquivo binário
Leitura e escrita

Escrevendo e lendo binário com fwrite() e fread()

- Vamos considerar a seguinte estrutura e o vetor de teste inicializado:

```
struct Person {
   int id;
   char name[51];
   char email[41];
};

int n = 5;
struct Person v[] = {
   {1, "Carlos Antonio", "carlos.antonio@gmail.com"},
   {2, "Maria Clara", "maria.clara@gmail.com"},
   {3, "Jose Eduardo", "jose.eduardo@gmail.com"},
   {4, "Marcos Cillo", "marcos.cillo@gmail.com"},
   {5, "Marcia Marcozo", "marcos.marcozo@gmail.com"}
};
```

Escrevendo e lendo binário com fwrite() e fread()

```c
FILE* file = fopen("arquivo.bin", "w+");
fwrite(v, sizeof(struct Person), n, file);

rewind(file); // mesmo que fseek(file, 0, SEEK_SET);

struct Person buff[n];
fread(buffer, sizeof(struct Person), n, file);

for (int i = 0; i < n; i++) {
    printf("%d, %s, %s\n",
            buffer[i].id, buffer[i].name, buffer[i].email);
}

fclose(file);
```

Escreve o vetor de estruturas Person inteiro no arquivo

Posiciona cursor no início do arquivo.

Lê o vetor de estruturas Person inteiro do arquivo

UTFPR

Considerando o arquivo.bin criado

```
FILE* file = fopen("arquivo.bin", "r+");
// substitui 4o reg no arquivo
fseek(file, sizeof(struct Person) * 3, SEEK_SET);
struct Person newPerson = {9, "Daniel Resina", "dresina@gmail.com"};
fwrite(&newPerson, sizeof(struct Person), 1, file);

struct Person read;
fseek(file, sizeof(struct Person) * 0, SEEK_SET); // lê 1o reg do arquivo
fread(&read, sizeof(struct Person), 1, file);
printf("%d, %s, %s\n", read.id, read.name, read.email);

fseek(file, sizeof(struct Person) * 3, SEEK_SET); // lê 4o reg do arquivo
fread(&read, sizeof(struct Person), 1, file);
printf("%d, %s, %s\n", read.id, read.name, read.email);
fclose(file);
```

fseek() posiciona cursor em N byte a partir do início do arquivo (SEEK_SET)

Quadro geral
Recapitulando todas as funções mencionadas

Operação	Função	Descrição
open	fopen(caminho, modo) → ponteiro arquivo	Abre arquivo (ver **modo**)
close	fclose(ponteiro)	Fecha arquivo
read txt	fgetc(ponteiro) → int	Lê char
write txt	fputc(char, ponteiro) → int	Escreve char
read txt	fgets(string, n, ponteiro) → string	Lê string
write txt	fputs(string, ponteiro) → string	Escreve string
read txt	fscanf(file, string_formatada, ...)	Lê string formatada
write txt	fprintf(file, string_formatada, ...)	Escreve string formatada
read bin	fread(end_buffer, tamanho_dado, qtde_dado, ponteiro)	Lê bytes
write bin	fwrite(end_buffer, tamanho_dado, qtde_dado, ponteiro)	Escreve bytes
get cursor	ftell(ponteiro) → posição (bytes)	Obtém posição do cursor em bytes
set cursor	fseek(ponteiro, posição_bytes, referência) referência: SEEK_SET, SEEK_END, SEEK_CUR	Posiciona cursor em bytes