Programação 2

Aula 6

Thiago Cavalcante – thiago.cavalcante@penedo.ufal.br

22 de janeiro de 2020

Universidade Federal de Alagoas – UFAL
Campus Arapiraca
Unidade de Ensino de Penedo

Arquivos

Arquivos são coleções de bytes armazenados em um dispositivo de armazenamento (como um disco rídigo, por exemplo)

Um arquivo pode ser interpretado como um texto, uma imagem, um vídeo etc.

Arquivos são coleções de bytes armazenados em um dispositivo de armazenamento (como um disco rídigo, por exemplo)

Um arquivo pode ser interpretado como um texto, uma imagem, um vídeo etc.

Vantagens

- Armazenamento durável
- Grandes quantidades de informação
- Acesso sequencial ou não
- Acesso concorrente (vários programas)

Arquivos

arquivos texto $\times$ arquivos binários

Arquivos texto

- Podem ser mostrados **diretamente na tela** ou modificados em um editor de texto como o **bloco de notas**
- Dados são gravados como **caracteres de 8 bits da tabela ASCII** (é feita uma conversão)

Arquivos binários

- Armazenam uma **sequência de bits** sujeita às convenções dos programas que a geraram
- São gravados **exatamente como estão organizados na memória**, sem conversão
- Exemplos: arquivos executáveis, arquivos compactados etc.

Leitura e escrita

Funções da biblioteca **`stdio.h`**

- Abrir/fechar
- Leitura/escrita de caracteres/bytes

Ponteiro para arquivo

```
FILE *p;
```

Aponta para uma área na memória chamada de **buffer**, que contém informações sobre o arquivo aberto

Abrindo um arquivo: `fopen`

```
FILE *fopen(char *nome_do_arquivo, char *modo)
```

`nome_do_arquivo` pode ser um caminho **absoluto** ou **relativo**, com relação à localização do programa

Retorna **`NULL`** caso ocorra um erro

Arquivos

Localização do programa

```
"C:\Programas\programa.c"
```

Localização do arquivo (absoluta)

```
"C:\Programas\Arquivos\arquivo.txt"
```

Localização do arquivo (relativa)

```
".\Arquivos\arquivo.txt"
```

Arquivos

Modo	Função
`"r"`	Leitura. Arquivo deve existir.
`"w"`	Escrita. Cria arquivo, se não houver. Apaga o anterior, se ele existir.
`"a"`	Escrita. Os dados serão adicionados no fim do arquivo (*append*).
`"r+"`	Leitura/escrita. O arquivo deve existir e pode ser modificado.
`"w+"`	Leitura/escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
`"a+"`	Leitura/escrita. Os dados serão adicionados no fim do arquivo (*append*).

Para arquivos binários, adiciona-se um **b** ao modo

Saindo de um programa em caso de erro

```
void exit(int codigo_de_retorno)
```

Convenção: **`codigo_de_retorno`** é **`0`** para um término normal e outro número caso ocorra um problema

Fechando um arquivo: `fclose`

```
int fclose(FILE *fp)
```

Retorna **`0`** no caso de sucesso e outro número, caso contrário

Ao fechar um arquivo, todo caractere que tenha permanecido no buffer é **gravado** (também ocorre quando o buffer fica cheio)

A função **`exit`** fecha todos os arquivos abertos

Escrevendo um char: `fputc`

```
int fputc(int c, FILE *fp)
```

Retorna a constante **EOF**, se houver erro na escrita, ou o **próprio caractere**, se ele foi escrito com sucesso

Após a escrita, o **indicador de posição interna** é avançado em um caractere (pode ser visto como o **cursor**)

Pode-se escrever na tela com o arquivo **`stdout`**

Lendo um char: `fgetc`

```
int fgetc(FILE *fp)
```

Retorna a constante **EOF**, se houver erro na leitura, ou o **próprio caractere**, se ele foi lido com sucesso

Após a leitura, o **indicador de posição interna** é avançado em um caractere

Pode-se ler do teclado com o arquivo **`stdin`**

Fim do arquivo: `feof`

```
int feof(FILE *fp)
```

Retorna **0** caso o fim ainda não tenha sido atingido e outro número, caso contrário

Posição atual: `ftell`

```
long int ftell(FILE *fp)
```

Para **arquivos binários**, o valor retornado indica o **número de bytes lidos** a partir do início do arquivo

Para arquivos **texto**, **não existe garantia** de que o valor retornado seja o número exato de bytes lidos a partir do início do arquivo (mas esse valor pode ser usado para se retornar ao mesmo ponto, de qualquer forma)

Em caso de erro, retorna -1

Escrevendo uma string: `fputs`

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

Retorna a constante **EOF**, se houver erro na escrita, ou um valor diferente de **0**, caso contrário

Não insere uma nova linha '\n' (responsabilidade do programador)

Lendo uma string: `fgets`

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

Retorna **`NULL`**, se houver erro na leitura, ou o ponteiro para o primeiro caractere da string, caso contrário

Lê até encontrar uma nova linha `'\n'` (que fará parte da string) ou até receber **`tamanho - 1`** caracteres (o último é sempre `'\0'`)

Escrevendo blocos de bytes: `fwrite`

```
int fwrite(void *buffer, int nro_de_bytes,
           int count, FILE *fp)
```

Retorna o número de unidades escritas com sucesso (pode ser menor que **`count`** ⚠)

Lendo blocos de bytes: `fread`

```
int fread(void *buffer, int nro_de_bytes,
          int count, FILE *fp)
```

Retorna o número de unidades escritas com sucesso (pode ser menor que **`count`** ⚠)

Boa prática: escrever no arquivo o tamanho da string/array antes do conteúdo (facilita na hora de ler, depois)

Escrevendo dados formatados: `fprintf`

```
int fprintf(FILE *fp, "tipos de saída",
            lista de variáveis)
```

Retorna o total de caracteres escritos, em caso de sucesso, ou um número negativo, caso contrário

Lendo dados formatados: `fscanf`

```
int fscanf(FILE *fp, "tipos de entrada",
           lista de variáveis)
```

Retorna o total de caracteres lidos, em caso de sucesso, ou a constante **EOF**, caso contrário

As funções `fprintf` e `fscanf` convertem os dados para caracteres, o que faz com que os arquivos sejam maiores e as operações levem mais tempo para serem executadas

Movendo-se dentro do arquivo: `fseek`

```
int fseek(FILE *fp, long numbytes, int origem)
```

Retorna **0**, caso a movimentação seja bem sucedida, ou outro valor, caso contrário

origem pode ser **SEEK_SET**, **SEEK_CUR** ou **SEEK_END**

numbytes pode ser negativo

Voltando ao início do arquivo: `rewind`

```
void rewind(FILE *fp)
```

Excluindo um arquivo: `remove`

```
int remove(char *nome_do_arquivo)
```

Recebe o **caminho para o arquivo** e não o ponteiro para FILE

Retorna **0**, caso o arquivo seja removido com sucesso, ou outro valor, caso contrário

Erro ao acessar o arquivo: `ferror` e `perror`

```
int ferror(FILE *fp)
void perror(char *str)
```

`ferror` detecta se a última operação realizada no arquivo produziu algum erro e retorna **`0`**, no caso negativo

`perror` imprime na tela a string **`str`** seguida da mensagem de erro do sistema