Programação de Computadores

Aula #04

Acesso a Arquivos

O que são arquivos? O que é um arquivo texto? Quais são as funções de manipulação de arquivos texto?

Ciência da Computação - BCC e IBM - 2024/01
Prof. Vinícius Fülber Garcia

Arquivos

De maneira geral, um arquivo armazena uma sequência de bytes, cuja interpretação é determinada pela aplicação.

Os arquivos são utilizados para armazenar informações de forma persistente, geralmente em um disco magnético ou SSD.

Em termos gerais, existem duas categorias de arquivos:

ARQUIVOS TEXTO E ARQUIVOS BINÁRIOS

Arquivos Texto

Podemos compreender um arquivo de texto como aquele com conteúdo que pode ser diretamente reconhecido por humanos como **texto simples, também conhecido como texto plano**.

Esses arquivos podem armazenar diversos tipos de dados, como texto sem formatação, código-fonte de programas, configurações de sistema e dados estruturados em formato de texto, como CSV (valores separados por vírgula) ou JSON (notação de objetos JavaScript), entre outros.

Arquivos Texto

Sendo assim, os arquivos de texto são tipicamente associados a caracteres digitáveis e caracteres de controle de texto, estes definidos por meio de um modelo de codificação, como ASCII, UTF-8 e ISO-8859-1.

Na tabela ASCII padrão, os **caracteres digitáveis estão na faixa de 32 a 127**, enquanto os **caracteres de controle incluem** o caractere nulo (**\0**), o caractere de tabulação (**\t**), o caractere de nova linha (**\n**), o caractere de retorno de carro (**\r**), entre outros.

Manipulação de Arquivos em C

A linguagem C fornece uma variedade de funções para a manipulação de arquivos por meio da biblioteca padrão *stdio*:

```
include <stdio.h>
```

O processamento de arquivos em C pode ser realizado usando *streams* (FILE*) ou descritores (fornecidos pelo sistema operacional). Embora menos eficiente em termos de desempenho, os *streams* são mais abstratos e portáteis, sendo a ferramenta de manipulação de arquivos geralmente preferida.

Manipulação de Arquivos em C

Além disso, é importante mencionar que a biblioteca *stdio* inclui alguns arquivos padronizados que possuem funcionalidades específicas:

- **stdin**: entrada padrão, normalmente associada ao teclado [scanf()]
- **stdout**: saída padrão, geralmente associada à tela [printf()]
- **stderr**: saída de erros, também associada à tela [perror()]

Esses arquivos padronizados são gerenciados pelo sistema operacional e podem ser usados diretamente em seu programa.

Mas... e para manipular um arquivo específico através do meu programa?

Abrindo e Fechando Arquivos

Antes de podermos manipular os dados de um arquivo, é necessário abri-lo.

Ao abrir um arquivo, estamos **estabelecendo uma "conexão" entre o programa e um arquivo** específico armazenado no sistema de arquivos. Essa conexão permite que o programa acesse e manipule o conteúdo do arquivo.

A manipulação de um arquivo pode ocorrer tanto em termos de escrita quanto de leitura.

Abrindo e Fechando Arquivos

A função fopen da linguagem C é utilizada para abrir arquivos e possui a seguinte assinatura:

```
FILE* fopen (const char *filename, const char *mode)
```

Ela define dois parâmetros: *filename* e *mode*. O parâmetro *filename* é uma *string* que contém o nome do arquivo a ser aberto, incluindo o caminho até o mesmo, se necessário. Já o parâmetro *mode* indica o modo de abertura do arquivo.

Modo de abertura?

Abrindo e Fechando Arquivos

Os modos de abertura de um arquivo indicam as operações que podem ser executadas no arquivo e a localização dos ponteiros de leitura e escrita em relação aos dados existentes.

Objetivamente, os modos são:
LEITURA, ESCRITA E CONCATENAÇÃO

Abrindo e Fechando Arquivos

MODO DE LEITURA (*READ*)

O modo de leitura é utilizado para abrir um arquivo exclusivamente para leitura, onde nenhuma operação de escrita é permitida.

Quando o arquivo não existe, a função *fopen* retorna um ponteiro nulo.

O argumento correspondente ao parâmetro *mode* da função *fopen* é **“r”**:

```
fopen(“MeuArquivo.txt”, “r”);
```

Abrindo e Fechando Arquivos

MODO DE ESCRITA (*WRITE*)

O modo de escrita é utilizado para abrir um arquivo exclusivamente para escrita, onde nenhuma operação de leitura é permitida.

Quando o arquivo não existe, a função *fopen* cria o arquivo e retorna o seu ponteiro. Caso o arquivo exista, este tem seu conteúdo resetado.

O argumento correspondente ao parâmetro *mode* da função *fopen* é **“w”**:

```
fopen(“MeuArquivo.txt”, “w”);
```

Abrindo e Fechando Arquivos

MODO DE CONCATENAÇÃO (*APPEND*)

O modo de concatenação é utilizado para abrir um arquivo para leitura e concatenação de dados (isto é, escrita no final do arquivo).

Quando o arquivo não existe, a função *fopen* cria o arquivo e retorna o seu ponteiro. Caso o arquivo exista, o seu conteúdo é preservado.

O argumento correspondente ao parâmetro *mode* da função *fopen* é **“a”**:

```
fopen(“MeuArquivo.txt”, “a”);
```

Abrindo e Fechando Arquivos

Também existem modos derivados dos mencionados anteriormente, que envolvem a inclusão do símbolo "+" ao lado do modo de abertura.

Nesses casos, independentemente do modo, o arquivo é aberto para leitura e escrita, e caso não exista, um novo arquivo é criado. Além disso:

- “r+”: o conteúdo é preservado; ponteiros são alocados no início do arquivo
- “w+”: o conteúdo é apagado; ponteiros alocados no início do arquivo
- “a+”: o conteúdo existente é preservado; o ponteiro de leitura é alocado no início do arquivo, e o ponteiro de escrita é alocado no final do arquivo

Abrindo e Fechando Arquivos

Após manipular os dados de um arquivo, é importante fechá-lo usando a função *fclose*. A assinatura de tal função é a seguinte:

```
int fclose (FILE* stream)
```

A função retorna o valor zero (0) caso o fechamento seja bem sucedido.

O fechamento do arquivo encerra o processo de manipulação de seus dados, libera os recursos computacionais alocados para tais processos e esvazia os *buffers* de saída.

Abrindo e Fechando Arquivos

Note que a manipulação de um arquivo é feita usando um ponteiro do tipo **FILE***.

Um ponteiro FILE* aponta para uma estrutura de controle de arquivo que contém informações importantes, como:

- Nome do arquivo
- Modo de abertura (leitura, gravação, etc.)
- Posição atual do ponteiro de leitura
- Posição atual do ponteiro de gravação

Escrita em Arquivos

As operações de escrita permitem adicionar dados em arquivos. Essa adição se dá byte a byte a partir do endereço apontado pelo ponteiro de escrita de um arquivo aberto.

Existem duas categorias principais de operações de escrita em arquivos:

SIMPLES E FORMATADA

Escrita Simples em Arquivos

A função *fputc* permite **escrever um único byte** em um arquivo especificado. Sua assinatura é a seguinte:

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

A função *fputc* possui dois parâmetros: um inteiro *c*, que representa o byte a ser gravado, e o *stream s*, indicando em qual arquivo o byte deve ser gravado. Em caso de sucesso, a função retorna o próprio byte escrito.

Opção via macro: `int putc(int c, FILE* s)`

Opção para o *stream stdout*: `int putchar(int c)`

Escrita Simples em Arquivos

Porém, em arquivos de texto, uma operação bastante conveniente é a **escrita de *strings***, em vez de escrever cada um de seus caracteres individualmente. A função fputs disponibiliza essa funcionalidade:

```
int fputs(const char *str, FILE* s)
```

A função *fputs* recebe a *string str* a ser escrita e o *stream s* que indica o arquivo onde a *string* será escrita. Ela retorna um valor positivo em caso de sucesso na escrita.

Opção para o *stream stdout*: `int puts(const char *str)`

Escrita Formatada em Arquivos

A função *fprintf*, no entanto, permite a escrita de arquivos seguindo um **padrão de formatação** (o mesmo do *printf*):

```
int fprintf (FILE* s, const char* f, ...)
```

A função *fprintf* possui n parâmetros: um *stream s*, indicando em qual arquivo os dados devem ser gravados; uma *string f* determinando o formato de gravação e os dados a serem inseridos nas posições previstas no formato. Em caso de sucesso, a função retorna o número de bytes escritos no arquivo.

Opção para *string*: `int sprintf (char* str, const char* f, ...)`

Opção para o *stream stdout*: `int printf (const char* f, ...)`

Leitura de Arquivos

Se a intenção de escrever em um arquivo é inserir dados nele, de forma intuitiva podemos entender que ler de um arquivo é recuperar dados do mesmo, utilizando um arquivo previamente aberto.

Além disso, as operações de leitura de arquivos são categorizadas em **LEITURAS SIMPLES** e **LEITURAS FORMATADAS**.

Leitura Simples em Arquivos

A função *fgetc* permite **ler um único byte** em um arquivo especificado. Sua assinatura é a seguinte:

```
int fgetc(FILE* s)
```

A função *fgetc* possui apenas um parâmetro: o *stream s*, indicando de qual arquivo o byte deve ser lido. Em caso de sucesso, a função retorna o byte lido.
Opção via macro: `int getc(FILE* s)`
Opção para o *stream stdin*: `int getchar()`

Leitura Simples de Arquivos

Já a função *gets* é destinada a ler uma *string* do *stdin* até encontrar uma nova linha (\n), descartando o caractere de nova linha e armazenando os dados na *string str* fornecida como argumento:

```
char* gets (char *str)
```

Porém, essa função é considerada bastante insegura! Por quê?

Leitura Simples de Arquivos

Devido aos potenciais estouros de *buffer*, uma função alternativa de leitura de linhas de arquivos para strings foi implementada:

```
char* fgets (char *str, int c, FILE *s)
```

Nesse caso, um limite de caracteres lidos é definido para *c-1*, e o caractere de fim de linha (\n) é incluído na leitura de uma linha qualquer, esta lida do *stream s* e armazenada na *string str* (sendo *c*, *s* e *str* parâmetros da função).

Note que a função *fgets* também é mais genérica que a *gets*! Por quê?

Leitura Formatada de Arquivos

A função *fscanf*, por sua vez, oferece a leitura de arquivos seguindo um **padrão de formatação** (o mesmo do *scanf*):

```
int fscanf (FILE* s, const char* f, ...)
```

A função *fscanf* possui n parâmetros: um *stream s*, indicando de qual arquivo os dados devem ser lidos; uma *string f* determinando o formato de leitura e as variáveis onde os dados devem ser inseridos considerando as posições previstas no formato. Em caso de sucesso, a função retorna o número de dados lidos.

Opção para *string*: `int sscanf (const char* s, const char* f, ...)`

Opção para o *stream stdin*: `int scanf (const char* f, ...)`

O Fim de Arquivo

Em iterações com arquivos, é crucial identificar o ponto que indica o fim dos dados presentes no arquivo. A partir desse ponto, as operações de leitura não serão mais executadas, uma vez que não há mais dados disponíveis para serem retornados.

Mas a pergunta é: **como eu identifico o final de um arquivo?**

O Fim de Arquivo

A linguagem de programação C oferece recursos para identificar e lidar com o final de um arquivo, e um dos mais comuns é a função *feof* (*found end of file*):

```
int feof(FILE *stream)
```

A função *feof* funciona da seguinte maneira:

- Retorna zero (0) se o fim do arquivo não foi alcançado.
- Retorna um número diferente de zero se o fim do arquivo foi alcançado.

O Fim de Arquivo

Outra opção para verificar o fim de um arquivo é verificar a constante **EOF (E*nd Of File*)**, que é retornada por funções como *getchar* e *fgetc* quando nenhum novo caractere é lido.

Também é possível verificar se ocorreu algum erro na última operação realizada em uma stream de arquivo usando a função *ferror*. Ela retorna zero se nenhuma ocorrência de erro foi registrada:

```
int ferror (FILE* stream)
```

Um Exemplo Simples

```c
#include <stdio.h>
int main() {
   FILE *arquivo;
   char byte_lido, byte_escrito;

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

   byte_lido = fgetc(arquivo);
   printf("Byte lido: %c\n", byte_lido);
   byte_escrito = 'A';
   fputc(byte_escrito, arquivo);
   printf("Byte escrito: %c\n", byte_escrito);

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

O código ao lado apresenta algumas rotinas simples de manipulação de arquivos:

- Qual é o modo de leitura, e o que ele permite?
- O que acontece se não houver dados no arquivo aberto?

Exercício #4

Considere o arquivo fornecido junto com o material desta aula para a realização de testes. Este arquivo contém um texto em inglês.

Desenvolva um programa que receba o nome, abra um arquivo texto e faça uma varredura em todas as suas palavras, satisfazendo o seguinte requisito para apresentar o resultado:

- Identifique e conte as ocorrências da palavra mais frequente
  - Em empates, retorne qualquer uma das palavras mais frequentes
  - Nenhuma palavra terá mais do que 100 caracteres
  - Não se preocupe com pontuações

Exercício #4

Você foi designado para desenvolver um programa que analisa um arquivo de vendas e gera um relatório com informações importantes. O arquivo de vendas contém uma lista de registros (um por linha), cada um representando um tipo de produto vendido:

Nome_do_produto|Valor_unitário|Número_de_vendas

O relatório gerado deve responder às seguintes perguntas:

- Qual é o faturamento total?
- Qual é o produto mais vendido?
- Qual é o produto com maior receita?

Obrigado!

Vinícius Fülber Garcia
inf.ufpr.br/vinicius
vinicius@inf.ufpr.br