Arquivos (Parte 1)

Módulo 9 Aula 0

Linguagem C, o Curso Definitivo WR Kits

Autor: Dr. Eng. Wagner Rambo

A Linguagem C apresenta poderosos recursos e funções para manipulação de arquivos, onde podemos armazenar dados de nosso programa de forma definitiva, ao invés de perde-los ao final da execução. Outra grande vantagem é o fato de podermos ler dados de arquivos e adapta-los das mais diversas formas. Do ponto de vista da Linguagem C, um arquivo é simplesmente um conjunto sequencial de bytes – que também pode ser chamado de *stream* – onde até mesmo uma nova linha é interpretada como um caractere.

Para trabalhar com arquivos em um programa em C, precisamos a priori abrir um arquivo, associando-o a uma variável em nosso código. Podemos realizar todo o tipo de operação com os dados do arquivo e então devemos desvincular o mesmo de nosso programa, realizando o fechamento do arquivo.

A abertura do arquivo é realizada através de um ponteiro para arquivo e utilizamos para isso o tipo FILE. A escrita é toda com letras maiúsculas mesmo para destacar, pois FILE é um tipo definido na biblioteca stdio.h e não consiste em um tipo de dado básico ou em uma palavra reservada do C. O Box 1 demonstra como declarar um ponteiro para arquivo.

```
FILE   *nome;        /* declarando um ponteiro para arquivo */
```

Box 1 - O tipo FILE é usado para declarar ponteiros para arquivos.

Observe o operador de indireção (*) antes do nome do ponteiro para arquivo. E o nome segue as mesmas regras conhecidas de declaração de variáveis em C. A função *fopen* é utilizada para a abertura do arquivo e está contida também na biblioteca stdio.h. No Box 1 conheça o protótipo da função *fopen*.

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

Box 2 - Protótipo da função fopen.

O parâmetro *filename* é uma *string* contendo o nome do arquivo a ser aberto, com respectiva extensão, como por exemplo “info.txt”. E *mode* é a *string* que conterá o modo de abertura do arquivo.

Os modos possíveis para abertura estão explícitos na Tabela 1.

Modo	Significado
r	Abre um arquivo-texto para leitura (read)
w	Cria um arquivo-texto para escrita (write)
a	Acrescenta dados a um arquivo-texto (append)
rb	Abre um arquivo binário para leitura (read binary)
wb	Cria um arquivo binário para escrita (write binary)
ab	Acrescenta dados a um arquivo binário (append binary)
r+	Abre um arquivo-texto para leitura/escrita
w+	Cria um arquivo-texto para leitura/escrita
a+	Acrescenta ou cria um arquivo-texto para leitura/escrita
r+b	Abre um arquivo binário para leitura/escrita
w+b	Cria um arquivo binário para leitura/escrita
a+b	Acrescenta a um arquivo binário para leitura/escrita

Tabela 1 - Modos de abertura de um arquivo.

Antes do fechamento do arquivo, todos os dados são devidamente gravados. Após, é necessário liberarmos a memória alocada pela função *fopen*. Para isso, utilizamos a função *fclose*, cujo protótipo está evidenciado no Box 3.

```
int fclose(FILE *arq);
```

Box 3 - Protótipo da função fclose.

A função retorna 0 em caso de sucesso ou EOF (*end of file*) em caso de erro.

O código do Box 4 ilustra o processo de criação de um arquivo txt para escrita, a verificação do ponteiro para arquivo e o fechamento do mesmo.

```
main()
{
  FILE *arq;

  if((arq=fopen("teste.txt","w"))==NULL)
  {
    printf("ERRO\n");
    system("pause");
    exit(0);

  }


  fclose(arq);

  printf("Criou o arquivo com sucesso!\n");

}
```

Box 4 - Criando um arquivo .txt para escrita.

Repare que o ponteiro para arquivo arq apontará para a abertura com *fopen* e escrevemos a *string* “teste.txt” para abrir um arquivo de texto para escrita “w”. Nesse caso, o arquivo será criado, caso não exista. Verificamos se o ponteiro não contém a constante simbólica NULL, que nesse caso indicará a falha na abertura do arquivo. Portanto, o teste de condição é realizado e caso verdadeiro, geramos mensagem de erro e encerremos o programa. Do contrário

obtivemos sucesso na abertura e fechamos o arquivo com *fclose*, além de informar o fato com o *printf* final. Rodando o programa do Box 4, você irá ver que um arquivo de teste com a extensão txt surgirá na pasta do seu projeto. No entanto, o arquivo está vazio, pois não escrevemos nada nele. Uma forma de escrever caracteres em arquivos é utilizando a função *fputc*, conforme sintaxe do Box 5.

```
main()
{
  FILE *arq;

  if((arq=fopen("teste.txt","w"))==NULL)
  {
    printf("ERRO\n");
    system("pause");
    exit(0);

  }

  fputc('H',arq);
  fputc('e',arq);
  fputc('l',arq);
  fputc('l',arq);
  fputc('o',arq);

  fclose(arq);

  printf("Criou o arquivo com sucesso!\n");

}
```

Box 5 - Escrevendo "Hello" em um arquivo de texto.

Como pode-se perceber, o programa é muito semelhante ao do Box 4, com a adição da escrita do arquivo, caractere por caractere. Agora vamos entender como ler um arquivo existente, para isso, experimente o programa do Box 6.

```
main()
{
  FILE *arq;

  if((arq=fopen("readme.txt","r"))==NULL)
  {
    printf("ERRO\n");
    system("pause");
    exit(0);

  }

  printf("%c\n", fgetc(arq));
  printf("%c\n", fgetc(arq));
  printf("%c\n", fgetc(arq));

  fclose(arq);

  printf("Arquivo lido com sucesso!\n");

}
```

Box 6 - Lendo um arquivo de texto.

A função utilizada para ler caractere por caractere consiste na *fgetc* e podemos imprimi-los no console para verificar sua correta leitura. Por exemplo, se você criar um arquivo de texto chamado readme na mesma pasta do projeto C e escrever nele “abc”, você verá “abc” escrito no console. Se atualizar o arquivo para “abdc” você ainda assim verá “abc”, pois no programa do Box 6 estamos lendo apenas os três primeiros caracteres do arquivo com *fgetc*.

A função *fgetc* na realidade retorna um inteiro, conforme podemos observar em seu protótipo no Box 7.

```
int fgetc(FILE *arq);
```

Box 7 - Protótipo da função fgetc.

Isso é necessário devido à constante EOF (*End Of File*) que indica quando um arquivo terminou. Pois na leitura de caracteres, devemos ter espaço disponível para os 256 caracteres da tabela ASCII. Logo, o EOF deverá ser reconhecido fora dessa faixa. Por este motivo, historicamente utilizamos *int* ao invés de *char* para receber os caracteres lidos em um arquivo.

Então, para não precisarmos repedir a função *fgetc* por muitas e muitas vezes para ler um arquivo completo, podemos simplesmente lê-lo até que a constante EOF seja detectada. O código do Box 8 apresenta um dos métodos utilizados.

```
main()
{
  FILE *arq;
  int ch;

  if((arq=fopen("readme.txt","r"))==NULL)
  {
    printf("ERRO\n");
    system("pause");
    exit(0);

  }

  while((ch=fgetc(arq))!=EOF)
    printf("%c",ch);

  putchar('\n');

  fclose(arq);

  printf("Arquivo lido com sucesso!\n");

}
```

Box 8 - Código para leitura completa de um arquivo.

Podemos combinar os modos de leitura e escrita em um mesmo programa, confira no Box 9 um exemplo de leitura de arquivo e cópia do seu conteúdo.

```
main()
{
  FILE  *fr, *fw;
  int ch;

  fr = fopen("readme.txt","r");

  if(fr == NULL)
  {
    printf("ERRO1\n");
    exit(1);
  }

  fw = fopen("output.txt","w");

  if(fw == NULL)
  {
    printf("ERRO2\n");
    exit(2);
  }

  while((ch=fgetc(fr))!=EOF)   /* lê todos os caracteres */
    fputc(ch,fw);              /* copia para output */

  fclose(fr);
  fclose(fw);

}
```

Box 9 - Lendo um arquivo de texto e copiando o seu conteúdo.

Caso você for trabalhar apenas com arquivos de texto, poderá utilizar as funções *fprintf* e *fscanf* que permitem a escrita e leitura de dados em modo formatado. Ambas têm certa semelhança com as funções que já utilizamos largamente para escrita no console e leitura do teclado *printf* e *scanf*, apenas necessitam de um parâmetro adicional, que é o ponteiro para arquivo. O mesmo código do Box 9, por ser utilizado para leitura e escrita de arquivos de texto (formato txt), pode ser desenvolvido com as funções *fprintf* e *fscanf*. Confira no Box 10, você pode alterar apenas a linha de leitura e escrita do novo arquivo.

```
  while(fscanf(fr,"%c",&ch)!=EOF)   /* lê todos os caracteres */
    fprintf(fw,"%c",ch);             /* copia para output */
```

Box 10 - Você pode utilizar fprintf e fscanf para arquivos no formato de txt.

Você também pode achar interessante entrar com o nome do arquivo que deseja ler, ou determinar o nome do arquivo que deseja escrever. Como a função *fopen* apresenta uma *string* para o nome do arquivo, é possível solicitar esse nome via entrada do teclado com a já conhecida função *gets*. Veja um exemplo de leitura de arquivo no Box 11.

```
main()
{
  FILE *arq;
  char s[50];
  int ch;

  printf("Digite o nome do arquivo: ");
  gets(s);

  arq = fopen(s,"r");

  if(arq==NULL)
  {
    printf("ERRO\n");
    exit(1);

  }

  while((ch=fgetc(arq))!=EOF)
    printf("%c",ch);


  fclose(arq);

}
```

Box 11 - Leitura de um arquivo qualquer.

Se o arquivo desejado estiver fora da pasta do seu projeto, você poderá ler ele indicando o caminho no momento da entrada da *string*. Supondo que o arquivo que você deseja ler se chama "teste.txt" e esteja na pasta "arquivo" do disco C, a entrada será: "C:\arquivo\teste.txt".

Para criar um arquivo "teste2.txt" na pasta arquivo localizada no disco C, utilize "C:\\arquivo\\teste2.txt". Repetir a barra invertida é necessário nesse caso para o compilador reconhecer como o caractere de barra invertido e não como um formato de dado.

Você também pode utilizar a função *fputs* para escrever *strings* em arquivos ou *fgets* para ler *strings*. No Box 12 apresentamos um exemplo, onde você digita o nome do arquivo que deseja criar e inserir a *string* e depois ler o mesmo e mostrar no console.

```
main()
{
  FILE *arq;
  char s[50];

  printf("Digite o nome do arquivo: ");
  gets(s);

  arq = fopen(s,"w");

  if(arq==NULL)
  {
    printf("ERRO\n");
    exit(1);

  }

  fputs("Escrita de string no arquivo.", arq);

  fclose(arq);

  arq = fopen(s,"r");

  if(arq==NULL)
  {
    printf("ERRO\n");
    exit(2);

  }

  if(fgets(s,200,arq)) printf(s);

  fclose(arq);

}
```

Box 12 - Escrita e leitura de strings com fputs e fgets.

Exercício proposto: Desenvolva o projeto de um programa para criptografar arquivos de texto, que desloque os caracteres digitados 3 posições para frente.

Exemplo:

Mensagem original: **Curso de C**

Mensagem criptografada: **Fxuvr#gh#F**

Desenvolva um software para remover a criptografia e gerar um novo arquivo com a mensagem original.

Exemplo:

Mensagem criptografada: **Fxuvr#gh#F**

Criptografia removida: **Curso de C**

Bibliografia:

DAMAS, Luís; Linguagem C, décima edição.

Disponível em: https://amzn.to/3nGdIbN

SCHILDT, Herbert; C Completo e Total, terceira edição.

Disponível em: https://amzn.to/3xxi3l8