Arquivos (Parte 2)

Módulo 9 Aula 1

Linguagem C, o Curso Definitivo WR Kits

Autor: Dr. Eng. Wagner Rambo

Os arquivos binários são utilizados para guardar informações de nossos programas, como dados diversos, conteúdo de variáveis, entre outros. Diferentemente dos arquivos de texto, aqui escrevemos os bytes através de um vetor e poderemos utilizar funções de acesso direto aos elementos: a *fread* e a *fwrite*. Confira o protótipo da função *fwrite* no Box 1, que permite a escrita de blocos de bytes em um arquivo binário.

```
int fwrite(const void *ptr, int size, int n, FILE *arq);
```

Box 1 - Protótipo da função fwrite.

O parâmetro *ptr* é um ponteiro para *void*, pois poderá apontar para qualquer tipo e conterá o endereço do que desejamos armazenar em arquivo. *size* é o tamanho em bytes de cada elemento a ser armazenado, e poderemos utilizar a função *sizeof* para obter o tamanho adequado. O n consiste no número de elementos que iremos escrever. E *arq* é o ponteiro para arquivo, que devemos passar como parâmetro para *fwrite*. No Box 2 apresentamos um programa para escrever 10 bytes em um arquivo binário.

```
main()
{
  FILE  *arq;

  char vec[10] = {8,4,3,1,2,0,3,4,9,4};

  arq = fopen("info.dat","wb"); /* cria para escrita binária */

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

  }

  fwrite(vec, sizeof(char),10,arq);

  fclose(arq);

}
```

Box 2 - Escrevendo 10 bytes em um arquivo binário.

Substituindo o tipo *char* por *int* ou *float*, você conseguirá ver que o arquivo binário preenche mais bytes para reservar espaço para estes tipos de dados, no caso, 4 bytes. Para testar, só altere o vetor vec para o tipo desejado, e mude também o parâmetro de *sizeof* na função *fwrite* para o mesmo tipo do vetor.

No Box 3 apresentamos o protótipo da função *fread*.

```
int fread(const void *ptr, int size, int n, FILE *arq);
```

Box 3 - Protótipo da função fread.

Os parâmetros de *fread* são semelhantes aos parâmetros de *fwrite*, portanto, podemos realizar a leitura do arquivo *info.dat* que geramos anteriormente, utilizando o código do Box 4.

```
main()
{
  FILE  *arq;
  register int i;
  char vec[10];

  arq = fopen("info.dat","rb");

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

  }

  fread(vec, sizeof(char),10,arq);

  for(i=0;i<10;i++)
    printf("%d ",vec[i]);

  fclose(arq);
}
```

Box 4 - Faz a leitura de 10 elementos do tipo char em um arquivo binário.

Quando desejamos encontrar o final de um arquivo binário, não podemos recorrer à constante EOF, pois o seu valor pode ser lido pelo programa e interpretado como um byte de dado. Por este motivo, precisamos recorrer à função *feof*, que determina quando o arquivo binário chegou no final. O programa no Box 5 a seguir lê o arquivo info.dat e copia para arch.bin, lembrando que arquivos no formato .bin são largamente utilizados para gravarmos dados em memória do tipo EPROM, EEPROM, etc.

```
main()
{
  FILE *arq_dat, *arq_bin;
  int ch;

  arq_dat = fopen("info.dat","rb");

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

  arq_bin = fopen("arch.bin","wb");

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

  while(!feof(arq_dat))
  {
    ch = fgetc(arq_dat);

    if(!feof(arq_dat))
      fputc(ch,arq_bin);

  }

  fclose(arq_dat);
  fclose(arq_bin);

  printf("Arquivo copiado com sucesso\n");

}
```

Box 5 - Realiza a cópia de arquivos binários.

No programa do Box 5, utilizamos as funções *fgetc* e *fputc* para leitura e escrita de cada byte do arquivo binário, para salientar que também podemos utilizá-las neste tipo de arquivo.

Uma outra vantagem de trabalharmos com arquivos binários é a possibilidade de acessarmos diretamente pontos intermediários no arquivo. A função *ftell* retorna um inteiro longo e pode ser utilizada para lermos a posição atual do arquivo binário. O Box 6 apresenta um exemplo de aplicação.

```
main()
{

  FILE *arq_dat, *arq_bin;
  int ch;



  arq_bin = fopen("arch2.bin","wb");

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

                          /* inicia no endereço 0 */
  fputc(0xAA,arq_bin);    /* estará no endereço 1 */
  fputc(0xAB,arq_bin);    /* estará no endereço 2 */
  fputc(0xAC,arq_bin);    /* estará no endereço 3 */
  fputc(0xAD,arq_bin);    /* estará no endereço 4 */

  /* imprimirá 4 como resultado: */
  printf("End.Atual: %ld\n",ftell(arq_bin));

}
```

Box 6 - Código para teste da função ftell.

No exemplo do Box 6, abrimos um arquivo binário para escrita e inserimos 4 bytes no mesmo, através da função *fputc*. Após, imprimimos o valor retornado por *ftell*, que será 4, pois o ponto atual do arquivo é 4. Experimente inserir a função *rewind* após o *printf* do código do Box 6 e imprimir novamente o retorno de *ftell*. Veja no Box 7 as linhas a serem inseridas abaixo do *printf* do código do Box 6.

```
  rewind(arq_bin);  /* volta para a posição 0 */

  /* imprimirá 0 como resultado: */
  printf("End.Atual: %ld\n",ftell(arq_bin));
```

Box 7 - Utilizando rewind.

Como pode ser visto, teremos o valor 0 retornado por *ftell* uma vez que a função *rewind* devolve o ponteiro para o início do arquivo. Esta função pode ser utilizada quando você deseja reler um arquivo binário no código, sem a necessidade de reabri-lo.

A função que permite apontar o ponteiro para qualquer posição do arquivo é a *fseek*, que tem o seu protótipo explícito no Box 8.

```
int fseek(FILE *arq, long offset, int org);
```

Box 8 - Protótipo da função fseek.

A função *fseek* retornará falso caso o movimento no arquivo tenha sido realizado com sucesso, do contrário, devolverá verdadeiro. O parâmetro *arq* é o ponteiro para arquivo em que desejamos posicionar, *offset* é o número de bytes que queremos avançar (valor positivo) ou retroceder (valor negativo) no arquivo. Já o *org* indicará o local a partir de onde o salto no arquivo será realizado e são possíveis 3 constantes já pré-definidas para este parâmetro:

SEEK_SET, ou 0, realiza o salto a partir do início do arquivo;

SEEK_CUR, ou 1, realiza o salto a partir da posição corrente do arquivo;

SEEK_END, ou 2, realiza o salto a partir da posição final do arquivo.

O programa do Box 9 lê o tamanho do arquivo sem precisar percorrê-lo por completo.

```
main()
{

  FILE *arq_bin;

  arq_bin = fopen("arch.bin","rb");

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

  fseek(arq_bin,0,SEEK_END);

  printf("%ld bytes\n",ftell(arq_bin));

  fclose(arq_bin);

}
```

Box 9 - Lê o tamanho de um arquivo binário em bytes, com acesso direto.

Podemos aproveitar *fseek* para criar espécies de partições em arquivos binários, determinando o endereço em que desejamos escrever os dados. Vamos supor que desejamos gravar os dados BCh e 3Ch nos endereços 00h e 01h respectivamente, porém criaremos uma partição que escreverá os dados A1h e 73h a partir do endereço 0Bh. O código do Box 10 demonstra como realizar essa tarefa utilizando *fseek*.

```
main()
{
  FILE *arq_bin;

  arq_bin = fopen("arch3.bin","wb");

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

  fputc(0xBC,arq_bin);
  fputc(0x3C,arq_bin);

  fseek(arq_bin,0x0B,SEEK_SET);

  fputc(0xA1,arq_bin);
  fputc(0x73,arq_bin);

  fclose(arq_bin);
}
```

Box 10 - Criando uma partição simples em um arquivo binário.

Exercício proposto: uma prática comum na aplicação de arquivos em sistemas embarcados consiste em salvar dados de leitura de um sensor, como o de temperatura por exemplo. Com isso, pode-se desenvolver um *datalogger* que irá registrar a temperatura ambiente (ou de uma máquina por exemplo) em um período conhecido.

Desenvolva o projeto de um programa em C que leia o arquivo “celsius.dat” e mostra 10 valores de temperatura em °C na tela. Em seguida, solicite ao usuário para entrar com 10 novos valores, atualizando o conteúdo do arquivo. O arquivo deve ser gerado pelo próprio programa, quando este for executado pela primeira vez.

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