Aula 27

MC 102 - Algoritmos e Programação de Computadores

Arquivos Binários.

Arquivos Binários

Arquivos textos são mais simples de serem manipulados e permitem fácil acesso ao seu conteúdo. Porém, possuem as seguintes desvantagens:

- Os campos devem estar separados por algum caracter
- O acesso aos elementos é seqüencial
- Caracteres numéricos são armazenados como ASCII, ocupando mais bytes que o necessário

Arquivos Binários

- Variáveis int ou float têm tamanho fixo na memória. Por exemplo, um int ocupa 4 bytes.
- Representação em texto precisa de um número variável de dígitos (10, 5.673, 100.340), logo de um tamanho variável.
- Armazenar dados em arquivos de forma análoga à utilizada em memória permite:
  - Reduzir o tamanho do arquivo.
  - Realizar busca não seqüencial

Arquivos Binários

Arquivos binários têm seu conteúdo armazenado no formato binário e, por isso, ocupam menos bytes que os arquivos textos. Além disso, não precisam de separadores entre os campos nem entre os registros.

Para trabalharmos com arquivos binários na linguagem C, da mesma forma que com arquivo texto, devemos usar a biblioteca de funções **`stdlib.h`**.

Obs: *arquivos binários usam normalmente a extensão* ***.dat***

Abrindo Arquivos Binários

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

aloca espaço para a estrutura FILE, atribui informações do arquivo para ela e retorna o endereço desta estrutura na memória; deixa o arquivo aberto para as operações requeridas

Parâmetros:

nome_arq: String que contêm o caminho (completo ou relativo) e o nome do arquivo que será aberto

modo: String que define o modo de abertura do arquivo, que pode ser:

rb	somente para leitura (**o arquivo deve existir**)
wb	somente para escrita a partir do início do arquivo (se o arquivo não existir, cria um novo. Se já existir, apaga seu conteúdo antes da escrita)
ab	adiciona a um arquivo. A escrita é feita a partir do final do arquivo (se o arquivo não existir, cria um novo)
r+b	leitura e escrita no início do arquivo (**o arquivo deve existir**)
w+b	leitura e escrita no início do arquivo (apaga o conteúdo se o arquivo existir ou cria um novo arquivo caso contrário)
a+b	abre para leitura no início do arquivo e escrita no seu final (se o arquivo não existir, cria um novo)

Usando Arquivos Binários

Exemplo:

```
FILE *arq;
arq = fopen("teste.dat", "rb");
```

Como saber se o arquivo foi relmente aberto?

O resultado de fopen pode ser um apontador nulo (NULL). Neste caso, basta verificar se (farq != NULL).

```
FILE *arq;
arq = fopen("teste.dat", "rb");
if (arq == NULL) {
   printf("Erro ao abrir arquivo.\n");
   return 0;
}
```

Escrevendo em um Arquivo Binário

Escreve dados para um arquivo binário:

```
int fwrite(void *pt, int tam, int num, FILE *arq)
```

Parâmetros:

fmt: Um apontador para a variável que comtém os dados

tam: Tamanho de cada dado a ser escrito (pode ser um registro)

num: Número de elementos (registros) que serão escritos

arq: Apontador para a estrutura arquivo (FILE)

Retorno:

int que indica o número de itens escritos

Exemplo:

```
FILE *arq; int i[10] = {1,2,3,4,5,6,7,8,9,10};
arq = fopen("teste.dat", "wb");
if (arq != NULL)
 fwrite(i, sizeof(int), 10, arq);
...
```

Lendo de um Arquivo Binário

Lê dados de um arquivo binário:

```
int fread(void *pt, int tam, int num, FILE *arq)
```

Parâmetros:

fmt: Um apontador para a variável que receberá os dados

tam: Tamanho de cada dado a ser lido (pode ser um registro)

num: Número de elementos (registros) que serão lidos

arq: Apontador para a estrutura arquivo (FILE)

Retorno:

int que indica o número de itens lidos ou **EOF** (0) quando atinge o final do arquivo

Exemplo:

```
FILE *arq; int i[10];
arq = fopen("teste.dat", "rb");
if (arq != NULL)
  fread(i, sizeof(int), 10, arq);
...
```

Fechando um Arquivo Binário

Fecha um arquivo aberto e libera a estrutura da memória:

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

Parâmetros:

arq: Apontador para a estrutura arquivo (FILE)

Exemplo:

```
FILE *arq; char s[101];
arq = fopen("teste.dat", "rb");
if (arq != NULL) {
  fread(s, sizeof(char), 100, arq);
}
fclose(arq);
```

Fim de arquivo

Informa se chegou ao fim do arquivo:

```
int feof(FILE *arq)
```

Parâmetros:

arq: Apontador para a estrutura arquivo (FILE)

Exemplo:

```
FILE *arq; char s[101];

arq = fopen("teste.dat", "rb");
if (arq != NULL)
  while (!feof(arq)) {
    fread(s, sizeof(char), 100, arq);
    printf("%s\n", s);
  }
fclose(arq);
```

Obs: *Todo arquivo possui um marcador de final de arquivo, para facilitar a leitura e identificar o seu fim, assim como uma string possui o '\0'.*

Lendo e Escrevendo

Tanto **`fwrite`** quanto **`fread`** retornam um número inteiro. O valor retornado representa o número de bytes escritos / lidos pelo comando. Se o retorno for 0, significa que nenhum byte foi escrito / lido, ou porque o arquivo não pode ser escrito / lido, ou porque o disco encontra-se cheio (em caso de escrita), ou porque o final do arquivo foi atingido (em caso de leitura).

Como ler registros não seqüenciais?

Algumas vezes é interessante lermos determinados registros do arquivo, acessando-o de forma não seqüêncial. Para fazer isso, devemos utilizar o **cursor** que marca a posição de leitura/escrita atual do arquivo, semelhante ao cursor de texto que marca a posição na tela onde o texto será escrito.

Movendo o cursor do Arquivo

Mover o cursor para um determinada posição do arquivo:

```
int fseek(FILE *arq, int deslocamento, int modo)
```

Parâmetros:

arq: Apontador para a estrutura arquivo (FILE)

deslocamento: Quantidade (em bytes) a ser deslocada para o posicionamento do cursor (pode ser negativa)

modo: Modo de deslocamento do cursor, sendo:

- SEEK_CUR: a partir da posição corrente
- SEEK_SET: a partir do início do arquivo
- SEEK_END: a partir do fim do arquivo (usado para deslocamentos de trás para frente)

Retorno:

Zero (0) caso o posicionamento do cursor tenha sido possível. Em caso de erro, retorna um valor diferente de zero.

Movendo o cursor do Arquivo

```
int i, e, n;
FILE *arq;

if ((arq = fopen("vet.dat", "a+b")) == NULL)
  return 0;
do {
  printf("Digite:\n\t1) gravar\n\t2) ler\n\t0) sair\nOp: ");
  scanf("%d", &i);
  if (i == 1) {
    printf("digite o numero: ");
    scanf("%d", &n);
    fseek(arq, 0, SEEK_END); /* Vai para o final */
    fwrite(&n, sizeof(int), 1, arq);
  } else if (i == 2) {
    printf("digite o elemento: ");
    scanf("%d", &e);
    if (fseek(arq, (e-1)*sizeof(int), SEEK_SET)==0) &&
       (fread(&n, sizeof(int), 1, arq)>0)
      printf("Elemento %d = %d\n\n", e, n);
    else
      printf("Elemento Inexistente\n\n");
  }
} while ((i == 1)||(i == 2));
```

Exercício

Escreva um programa que leia dois arquivos de inteiros ordenados e escreva um arquivo cuja saída é um único arquivo ordenado. Vale a pena colocar o conteúdo dos arquivos de entrada em dois vetores?

Escreva duas versões deste programa, uma para arquivos texto e outra para arquivos binários.