Capítulo 5

Arquivos

5.1 Introdução

Todos os programas codificados até o momento podem armazenar dados de maneira temporária. Isto é, utilizamos a memória principal (RAM) como unidade de armazenamento. Desta forma, nossos programas não são capazes de recuperar as informações armazenadas pelos usuários após o reinício do computador ou mesmo do próprio programa.

Isto se deve ao fato de que os programas não utilizam o recurso de armazenamento secundário (ex.: disco rígido). Ao utilizar este componente, os programas são capazes de armazenar informações que serão salvas mesmo após o desligamento da máquina.

Para acessar este recurso os programas devem implementar as chamadas de sistema relativas a manipulação de arquivos. Um arquivo é uma unidade lógica de armazenamento de dados reconhecida pelos programas de computador. Cada linguagem possui suas próprias funções que realizam acessos em arquivos. Veremos na sequência como utilizar arquivos com a linguagem C.

DICA: Preste bastante atenção nas funções de manipulação de arquivos e verás que você já utiliza arquivos desde a primeira aula.

5.2 Conceito de Arquivo

Na linguagem C um arquivo é visto como uma seqüência de bytes. É também dito que um arquivo é um fluxo (stream) de caracteres (lembre-se que um caractere é um byte). Cada arquivo possui um caractere especial que indica o final do arquivo chamado ***EOF: End Of File***.

A figura a 5.1 ilustra um arquivo sob a ótica da linguagem C.

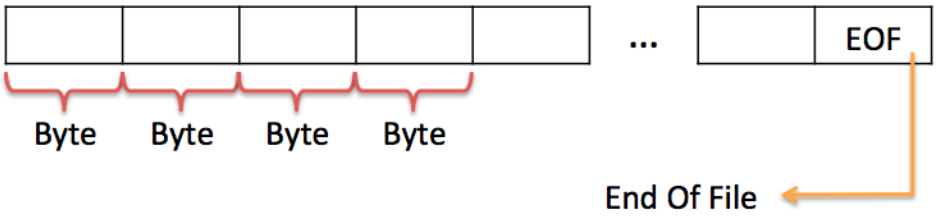

Figura 5.1: Representação de um arquivo: Fluxo (stream) de bytes.

A linguagem C trata os arquivos de duas formas básicas:

- **Arquivo Sequencial (texto)**; e,
- **Arquivo Aleatório (binário)**.

Esta diferença esta relacionada a forma pela qual um arquivo será manipulado. Para cada tipo de arquivo um conjunto específico de funções deve ser utilizado.

5.3 Operações Sobre Arquivos

Algumas operações básicas podem ser feitas sobre arquivos. Entre elas temos:

- Abertura de arquivo: **fopen();**
- Fechamento de arquivo: **fclose();**
- Leitura em arquivo: **fgetc(); fgets(); fscanf(); fread();**
- Escrita em arquivo: **fputc(); fputs(); fprintf(); fwrite();**
- Posicionamento de arquivo: **rewind(); fseek();**

Antes de efetivamente realizarmos escritas ou leituras em arquivos, devemos abri-lo para que possamos acessar seu conteúdo. As operações de abertura e fechamento de arquivo são mostradas a seguir.

5.3.1 Abertura de Arquivo

Como dito anteriormente, um arquivo é um fluxo de bytes. Logo, para acessar seu conteúdo precisamos de um ponteiro de um tipo especial chamado **FILE**. Um tipo FILE é uma estrutura interna do sistema operacional que mantém informações sobre arquivos abertos e que estão sendo manipulados por algum processo. A sintaxe para abrir um arquivo é mostrada a seguir:

```
1  int main(){
2
3     FILE *arq;
4     arq = fopen("teste.txt", "w");
5
6     if(arq == NULL){
7        printf("\n\tErro ao abrir arquivo.");
8        return 0; }
9     else { printf("\n\tArquivo aberto com sucesso."); }
10
11 fclose(arq);
12 return 0;
13 }
```

Neste exemplo podemos notar que uma variável ponteiro (**arq**) foi declarada sendo do tipo **FILE**. Esta variável recebe o retorno da função **fopen()**. A função fopen() retorna um **ponteiro que aponta para o primeiro byte do arquivo** (também dito **início do arquivo**). A figura a 5.2 ilustra o funcionamento da função fopen().

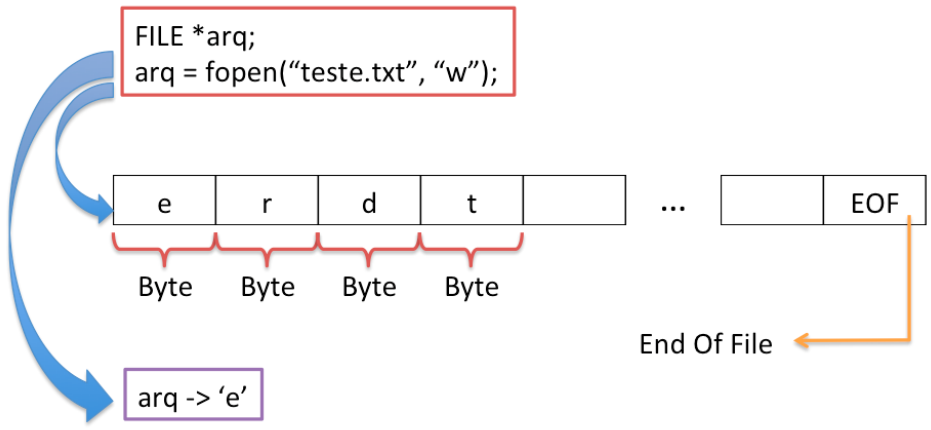

Figura 5.2: Conteúdo do ponteiro após receber o retorna da função fopen().

Repare nos parâmetros da função fopen(). Ela recebe 2 argumentos: o **primeiro** é uma **string** que contém o **nome do arquivo**. Este nome é o mesmo que aparecerá quando você visualizar o conteúdo do diretório. Veja na figura 5.3. o resultado após a execução do programa exemplo.

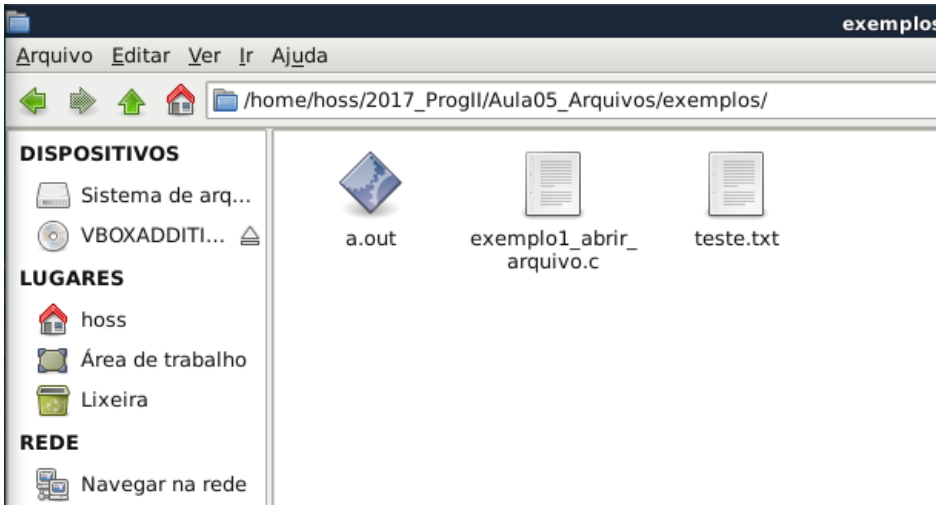

Figura 5.3: Arquivo (teste.txt) criado pelo programa exemplo é listado no diretório.

O **segundo** argumento é o **modo de abertura** de um arquivo. A tabela a seguir resume os principais modos de abertura de um arquivo. Este argumento deve sempre ser indicado na chamada da função.

Modo	Descrição
r	Abre um arquivo para leitura
w	Cria um arquivo para gravação. Se o arquivo já existir, elimina o conteúdo atual
a	Anexa: abre ou cria um arquivo para gravação no final do arquivo
r+	Abre um arquivo para atualização (leitura e gravação)
w+	Cria um arquivo para atualização. Se o arquivo já existir, elimina o conteúdo atual.
a+	Anexa: abre ou cria um arquivo para atualização; a gravação é feita no final do arquivo

Figura 5.4: Resumo dos modos de abertura de um arquivo.

Nota: A função fopen() retorna **NULL** se algum erro ocorrer durante a abertura do arquivo. Logo, é sempre importante testar se o processo de abertura foi realizado com sucesso. Ao contrário, o programa deve avisar o usuário e impedir seu prosseguimento.

5.3.2 Fechamento de Arquivo

Após finalizar o uso de um arquivo é sempre recomendado fechar explicitamente o mesmo. Isto é feito por meio da função **fclose();**. Veja no programa anterior que ela é adicionada ao final do código. Repare também que ela recebe como argumento o ponteiro utilizado na abertura do arquivo.

Se o seu programa abrir mais que um arquivo, ele deve fechar todos os arquivos abertos sendo utilizada a função fclose() para cada ponteiro de arquivo.

5.4 Importante

Antes de estudarmos as operações de leitura e escrita em arquivos, cabe aqui uma nota que ajudará você, aluno, a entender melhor a manipulação de arquivos.

Sabemos que qualquer manipulação de dados em nossos programas deve ser feita por meio de variáveis. Desta forma, devemos relacionar um arquivo criado por meio da função fopen() com os arquivos padrão do sistema (stdin, stdout e stderr).

Isto é, antes de somarmos duas variáveis nós capturamos seus valores com o scanf(). Da mesma forma, ao utilizarmos arquivos devemos extrair seu conteúdo e colocá-lo em variáveis equivalentes. Isto significa que se temos um arquivo com números inteiros, nosso programa deve declarar variáveis inteiras para receber estes números do arquivo e SOMENTE depois realizar operações sobre eles.

Este conceito vai ser esclarecido ao decorrer da leitura. Contudo, lembre-se sempre da sequência:

- **Leitura:** Arquivo -> RAM (variáveis) -> Processador
- **Escrita:** Processador -> RAM (variáveis) -> Arquivo

5.5 Arquivo Sequencial

Nesta seção iremos discutir a leitura e escrita sobre arquivos texto (acesso sequencial).

Após a abertura do arquivo, nosso programa está apto a realizar operações de escrita e leitura sobre o mesmo. A maneira pela qual iremos manipular nosso arquivo vai indicar se este arquivo se trata de um binário (acesso aleatório) ou um arquivo texto (acesso sequencial).

Os arquivos sequenciais são vistos pelos programas da mesma forma que o usuário os enxerga. Isto é, se o usuário abrir o arquivo diretamente pelo editor de texto vai conseguir ler seu conteúdo. Geralmente temos em cada linha do arquivo um conjunto de caracteres que representa alguma informação. A indicação de onde termina um dado e começa outro é dado pela quebra de linha (new line = '\n').

A figura 5.5 apresenta o conteúdo de um arquivo texto (sequencial).

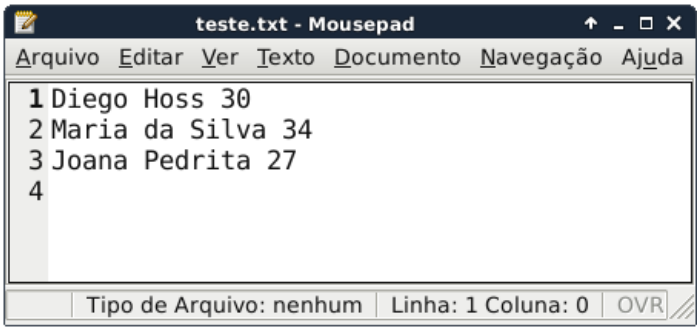

Figura 5.5: Conteúdo de um arquivo texto.

5.5.1 Escrita em Arquivo Sequencial

Uma vez aberto o arquivo em modo texto podemos agora escrever algum conteúdo neste arquivo. Para realizar esta tarefa, pode-se utilizar 3 (três) funções básicas:

- **fputc():** escreve um caractere no arquivo
- **fputs():** escreve uma frase no arquivo
- **fprintf():** escreve um conteúdo formatado no arquivo

Repare que estas funções possuem o mesmo nome das funções tradicionais (putc, puts e printf) acrescido da letra 'f' que indica "file". Isto significa que o conceito utilizado nas funções tradicionais é aplicado para arquivos. Vejamos um exemplo.

fputc()

```
1  #include <stdio.h>
2
3  int main(){
4
5          FILE *arq;
6          char ch;
7
8          //abre o arquivo
9          arq = fopen("teste.txt", "w");
10
11         //testa se a operacao foi realizada com sucesso
12         if(arq == NULL){
13                 printf("\n\tArquivo nao pode ser aberto."
                       );
14                 return 0;
15         }
16
17         //obtem o caracatere
18         printf("\n\tDigite um caractere: ");
19         scanf("%c", &ch);
20
21         //escreve o caractere contido na variavel 'ch'
               dentro do arquivo 'arq'
22         fputc(ch, arq);
23
24         //fecha o arquivo
25         fclose(arq);
26 return 0;
27 }
```

Lembra que anteriormente dissemos que se você prestasse bem atenção nas funções de escrita e leitura em arquivos verias que já as utilizava desde o início? Então repare bem na função **fputc()**. Note que sua utilização é praticamente idêntica à tradicional acrescida do arquivo para o qual se deseja escrever.

A figura 5.6 mostra a execução do programa.

```
hoss@debian8vm:~/2017_ProgII/Aula05_Arquivos/exemplos$ ./a.out

        Digite um caractere: d
hoss@debian8vm:~/2017_ProgII/Aula05_Arquivos/exemplos$
```

Figura 5.6: Execução do programa.

Veja na figura 5.7 o conteúdo do arquivo "teste.txt"após a execução do programa. Note que abrimos o arquivo em modo 'w'. Isto fez com que o conteúdo anterior fosse apagado.

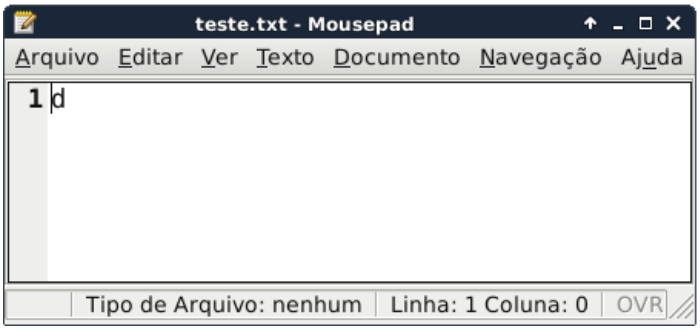

Figura 5.7: Caractere armazenado no arquivo teste.txt após a execução do programa.

fputs()

Do mesmo modo que o exemplo anterior, podemos também escrever frases completas nos arquivos. Assuma que o usuário digitou a frase: "Hoje o tempo está nublado.". Veja o exemplo.

```
1  #include <stdio.h>
2
3  int main(){
4
5          FILE *arq;
6          char frase[100];
7
8          //abre o arquivo
9          arq = fopen("teste.txt", "w");
10
11         //testa se a operacao foi realizada com sucesso
12         if(arq == NULL){
```

```
13                  printf("\n\tArquivo nao pode ser aberto."
                        );
14                  return 0;
15          }
16
17          //obtem a frase
18          printf("\n\tDigite uma frase: ");
19          __fpurge(stdin);
20          gets(frase);
21
22          //escreve a frase dentro do arquivo 'arq'
23          fputs(frase, arq);
24
25          //fecha o arquivo
26          fclose(arq);
27  return 0;
28  }
```

A figura 5.8 mostra o conteúdo do arquivo "teste.txt"após a execução do programa.

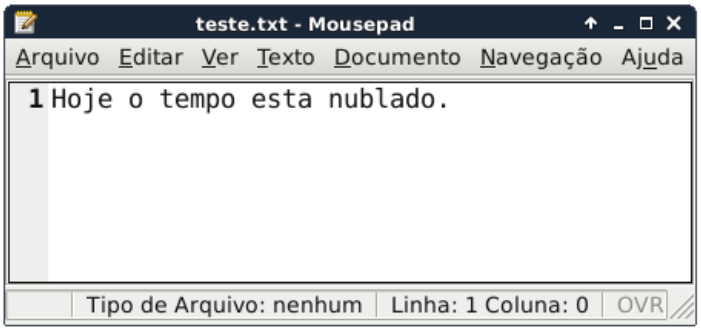

Figura 5.8: Frase armazenada no arquivo teste.txt após a execução do programa.

fprintf()

A função **fprintf()** é a mais elaborada entre as três disponíveis para arquivos texto. Com ela podemos enviar dados formatados para os arquivos. Veja o exemplo.

```
1  #include <stdio.h>
2
3  int main(){
```

```
4
5           FILE *arq;
6           char nome[100];
7           int idade, i;
8           float altura;
9
10          //abre o arquivo
11          arq = fopen("teste.txt", "w");
12
13          //testa se a operacao foi realizada com sucesso
14          if(arq == NULL){
15                  printf("\n\tArquivo nao pode ser aberto."
                        );
16                  return 0;
17          }
18
19          for(i = 0; i < 3; i++){
20                  printf("\n\tDigite o Nome: ");
21                  __fpurge(stdin);
22                  gets(nome);
23
24                  printf("\n\tDigite a idade: ");
25                  scanf("%d", &idade);
26
27                  printf("\n\tDigite a altura: ");
28                  scanf("%f", &altura);
29
30                  //escreve os campos formatados dentro do
                        arquivo 'arq'
31                  fprintf(arq, "%s - %d - %.2f\n", nome,
                        idade, altura);
32          }
33
34          //fecha o arquivo
35          fclose(arq);
36  return 0;
37  }
```

A figura 5.9 mostra a execução do programa.

```
hoss@debian8vm:~/2017_ProgII/Aula05_Arquivos/exemplos$ ./a.out

        Digite o Nome: Diego

        Digite a idade: 30

        Digite a altura: 1.78

        Digite o Nome: Joao

        Digite a idade: 81

        Digite a altura: 1.71

        Digite o Nome: Maria

        Digite a idade: 78

        Digite a altura: 1.54
hoss@debian8vm:~/2017_ProgII/Aula05_Arquivos/exemplos$
```

Figura 5.9: Execução do programa.

A figura 5.10 mostra o conteúdo do arquivo "teste.txt"após a execução do programa.

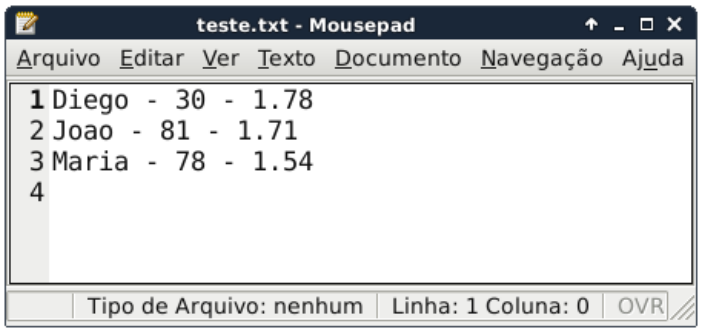

Figura 5.10: Conteúdo armazenado no arquivo teste.txt após a execução do programa.

Repare na sintaxe das funções **fputc(), fputs() e fprintf()**. Note que elas possuem apenas o **ponteiro para o arquivo** como parâmetro adicional em relação as funções tradicionais.

5.5.2 Leitura em Arquivo Sequencial

A leitura de arquivos texto segue a mesma lógica da escrita. Para ser mais exato, é a lógica inversa. Isto é, troca-se as funções fputc(), fputs() e fprintf() por: **fgetc(), fgets() e fscanf()**.

Deve-se também atentar para 3 itens importantes durante a leitura de arquivos.

- I) Inverte-se os argumentos durante a leitura. Isto é, o ponteiro para arquivo é o último parâmetro da função;
- II) Deve-se abrir o arquivo em modo leitura para não apagar seu conteúdo ('r');
- III) Deve-se utilizar a função feof() para saber quando chegamos ao final do arquivo;

A ideia de utilizar a função feof() é fazer com que a leitura ocorra enquanto o final do arquivo não é encontrado. A função feof() retorna um valor diferente de zero se o final do arquivo for encontrado. Vamos ver o exemplo:

fgetc()

```
1  #include <stdio.h>
2
3  int main(){
4
5          FILE *arq;
6          char ch;
7
8          //abre o arquivo
9          arq = fopen("teste.txt", "r");
10
11         //testa se a operacao foi realizada com sucesso
12         if(arq == NULL){
13                 printf("\n\tArquivo nao pode ser aberto."
                       );
14                 return 0;
15         }
16
17         while(!feof(arq)){
18
19                 //fgetc retorna o caracter lido
20                 ch = fgetc(arq);
```

```
21
22                  //escreve na tela
23                  printf("%c", ch);
24          }
25
26          //fecha o arquivo
27          fclose(arq);
28  return 0;
29  }
```

A figura 5.11 mostra o conteúdo do arquivo "teste.txt".

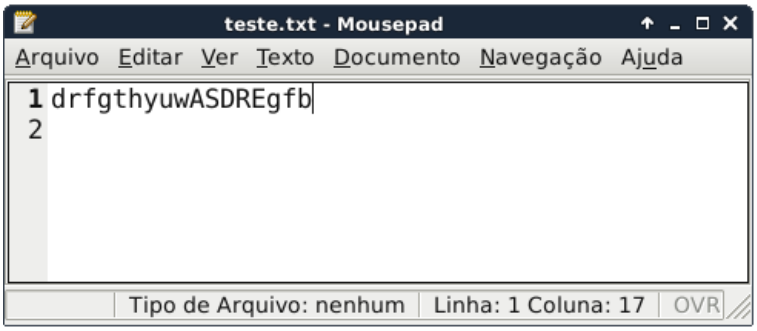

Figura 5.11: Conteúdo do arquivo teste.txt.

Note que para a leitura de arquivos utiliza-se a função feof() juntamente com um laço de repetição. Isto é, enquanto houver caracteres continue lendo-os.

fgets()

Em outro exemplo podemos ver a utilização da função fgets().

```
1  #include <stdio.h>
2
3  int main(){
4
5          FILE *arq;
6          char frase[100];
7
8          //abre o arquivo
9          arq = fopen("teste.txt", "r");
10
11         //testa se a operacao foi realizada com sucesso
12         if(arq == NULL){
```

```
13                  printf("\n\tArquivo nao pode ser aberto."
                        );
14                  return 0;
15          }
16
17          //obtem a frase
18          while((fgets(frase, sizeof(frase), arq))!=NULL ){
19                  printf("\n\tfrase: %s", frase);
20          }
21          //fecha o arquivo
22          fclose(arq);
23  return 0;
24  }
```

Esta função faz a leitura frase a frase. Isto é, lê todos os caracteres até encontrar o caractere newline ('\n'). Ela possui três parâmetros:

- **frase** -> vetor de char onde será armazenada a frase lida do arquivo;
- **sizeof(frase)** -> Indica quantos bytes deverão ser lidos; Utilizamos o **sizeof(frase)** porque queremos ler a quantidade de caracteres suportados pelo vetor; Isto é, se o vetor tem 100 caracteres este valor é informado na função; Se tiver 80, então 80 é passado para a função. Utilizando o sizeof evitamos o problema de não haver espaços para armazenar a frase.
- **arq** -> nome do ponteiro que aponta para o arquivo aberto;

Note a condição presente no while(). Para o caso de leituras de frases a condição de parada é diferente. Não precisamos chegar até final do arquivo com o feof(). Testamos se o retorno da função **fgets()** é diferente de NULL.

fscanf()

A função fscanf() funciona de maneira semelhante a função scanf(). A diferença fica por conta do primeiro argumento que deve ser o ponteiro do arquivo aberto. Veja o exemplo que é a resposta para o exemplo do fprintf().

```
1  #include <stdio.h>
2
3  int main(){
4
5          FILE *arq;
6          char nome[100];
7          int idade, i;
```

```
8           float altura;
9
10          //abre o arquivo
11          arq = fopen("dados.txt", "r");
12
13          //testa se a operacao foi realizada com sucesso
14          if(arq == NULL){
15                  printf("\n\tArquivo nao pode ser aberto."
                        );
16                  return 0;
17          }
18
19          for(i = 0; i < 3; i++){
20                  //le os campos formatados dentro do
                        arquivo 'arq'
21                  fscanf(arq, "%s - %d - %f", nome, &idade,
                         &altura);
22
23                  printf("\n\tNome: %s", nome);
24
25                  printf("\n\tIdade: %d", idade);
26
27                  printf("\n\tAltura: %.2f", altura);
28          }
29
30          //fecha o arquivo
31          fclose(arq);
32  return 0;
33  }
```

Neste exemplo é importante chamar a atenção para o segundo parâmetro da função. Note que ele possui os hífens dispostos entre os especificadores de conversão. Isto deve ser feito se o arquivo foi gravado utilizando estes caracteres. Veja a figura 5.10 e perceba que o conteúdo foi gravado com esta formatação. Logo ela deve ser respeitada na leitura.

5.6 Arquivo Aleatório (binário)

Como estudado até o presente momento, os arquivos criados com as funções fputc(), fputs() e fprintf() não possuem necessariamente o mesmo tamanho. Entretanto, normalmente os dados de um arquivo de acesso aleatório possuem o mesmo tamanho e podem ser acessados diretamente (e, dessa forma,

rapidamente) sem passar por outros registros.

Por conta disso, nós podemos ver o arquivo de acesso aleatório (binário) como um grande vetor sendo que em cada uma das posições do vetor está armazenado um tipo de dado (sempre o mesmo tipo de dado). Isto é, se desejarmos salvar valores inteiros usando arquivos binários, este arquivo deve armazenar somente dados do tipo **int**. Se a regra não for respeitada poderão ocorrer erros em tempo de execução e de inconsistência de dados.

A figura 5.12 mostra como um programa enxerga um arquivo binário durante a manipulação dos dados (leitura e escrita).

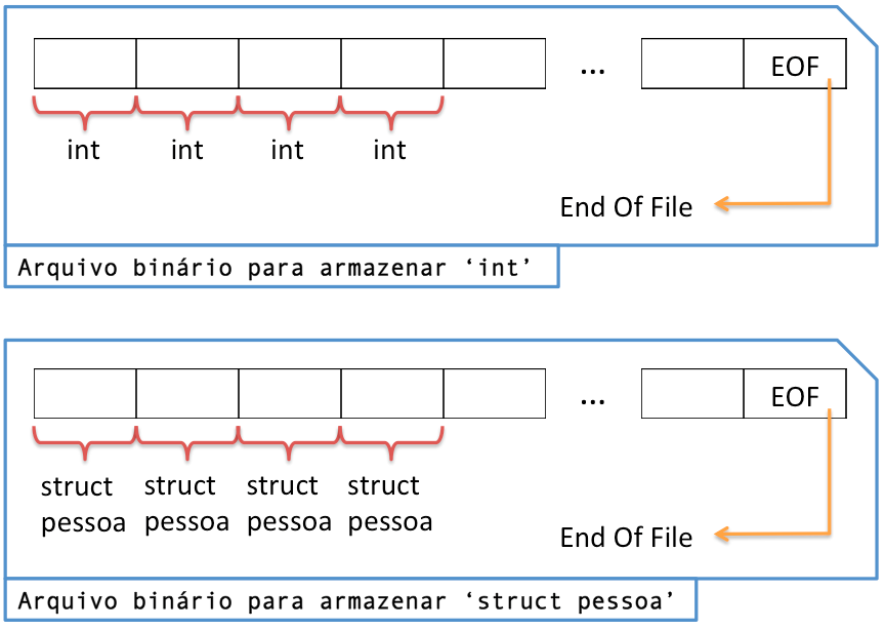

Figura 5.12: Representação de um arquivo binário considerando sua manipulação.

Nota: Para manipular arquivos binários devemos acrescentar a letra **'b'** no modo de abertura dos arquivos.

5.6.1 Escrita em Arquivos Binários

A inserção de dados em arquivos binários é feita pela função **fwrite()**. A função fwrite() transfere para um arquivo um número especificado de bytes, iniciando em um determinado local da memória. Os dados são gravados

iniciando no local do arquivo indicado pelo ponteiro de posição do arquivo. Vejamos um exemplo:

```
1  #include <stdio.h>
2  int main(){
3
4          FILE *arq;
5          int numero = 15;
6
7          //abre o arquivo em modo binario 'wb'
8          arq = fopen("teste.bin", "wb");
9
10         //testa se a operacao foi realizada com sucesso
11         if(arq == NULL){
12                 printf("\n\tArquivo nao pode ser aberto."
                       );
13                 return 0;
14         }
15
16         fwrite(&numero, sizeof(int), 1, arq);
17
18         //fecha o arquivo
19         fclose(arq);
20 return 0;
21 }
```

A função fwrite() recebe 4 argumentos:

- **&numero** -> os bytes representados pela variável número. Isto é, copia literalmente os bytes contidos no endereço de memória apontado por numero.
- **sizeof(int)** -> determina o tamanho em bytes que deve ser copiado. Como se trata de uma variável do tipo 'int' nós pegamos o tamanho do int por meio do sizeof().
- **1** -> indica o número de vezes que a cópia será feita. Isto é, quantas vezes queremos gravar o conteúdo da variável número?
- **arq** -> arquivo onde será armazenado o conteúdo.

O uso dos arquivos binários está intimamente relacionado ao conceito de estruturas (struct). Isto porque o uso do arquivo binário somado às estruturas permitem que programas de controle de estoque, reserva de voos e cadastro de usuários, por exemplo, sejam facilmente implementados.

Veja o próximo exemplo utilizando estruturas e arquivos binários.

```c
#include <stdio.h>

struct pessoa{
        char nome[30];
        int idade;
        float altura, peso;
};

int main(){

        struct pessoa aluno;
        FILE *arq;

        //abre o arquivo em modo binario 'wb'
        arq = fopen("teste.bin", "wb");

        //testa se a operacao foi realizada com sucesso
        if(arq == NULL){
                printf("\n\tArquivo nao pode ser aberto.");
                return 0;
        }

        printf("\n\tDigite seu nome: ");
        __fpurge(stdin);
        gets(aluno.nome);

        printf("\n\tDigite sua idade: ");
        scanf("%d", &aluno.idade);

        printf("\n\tDigite seu peso: ");
        scanf("%f", &aluno.peso);

        printf("\n\tDigite sua altura: ");
        scanf("%f", &aluno.altura);

        //insere a struct no arquivo
        fwrite(&aluno, sizeof(struct pessoa), 1, arq);

        fclose(arq);
printf("\n\n");
return 0;
}
```

Perceba que os 2 primeiros argumentos foram alterados. Agora nós queremos gravar no arquivo uma estrutura inteira - representada pela variável **aluno** do tipo **struct pessoa** (&aluno). Precisamos também indicar a quantidade de bytes que queremos gravar, ou seja, o tamanho (sizeof) da estrutura (**sizeof(struct pessoa)**).

5.6.2 Leitura em Arquivos Binários

Como já mencionamos, um arquivo binário não pode ser lido diretamente como é feito com um arquivo texto. Isto porque os dados são escritos no arquivo utilizando qualquer caractere da tabela ASCII, inclusive os não legíveis por humanos.

Para ter acesso ao conteúdo de um arquivo binário nós precisamos de um programa que faça a leitura deste arquivo e converta os dados em informações úteis. Isto é, a linguagem C não faz diferença entre uma **struct** e um **int** quando se trata de escrever em arquivos binários.

Por conta disso, é responsabilidade do programador fazer com que os dados sejam gravados e lidos de tal modo que possam ser recuperados e interpretados pelo programa. Para fazer isto, devemos utilizar o mesmo tipo de dado tanto na escrita quanto na leitura.

Para o exemplo anterior no qual gravamos uma estrutura chamada "pessoa", devemos agora ler os dados do arquivo com este mesmo tipo. Isto é, devemos recuperar os dados e armazenar em variáveis do tipo **struct pessoa**.

Vejamos o exemplo:

```
1  #include <stdio.h>
2
3  struct pessoa{
4          char nome[30];
5          int idade;
6          float altura, peso;
7  };
8
9  int main(){
10
11         struct pessoa aluno;
12         FILE *arq;
13
14         //abre o arquivo em modo binario 'rb'
15         arq = fopen("teste.bin", "rb");
16
17         //testa se a operacao foi realizada com sucesso
```

```
18          if(arq == NULL){
19                  printf("\n\tArquivo nao pode ser aberto."
                        );
20                  return 0;
21          }
22
23          //lê os registros do arquivo e armazena na
                variável aluno
24          fread(&aluno, sizeof(struct pessoa), 1, arq);
25
26          printf("\n\tDados cadastrais: ");
27          printf("\n\tNome: %s", aluno.nome);
28          printf("\n\tIdade: %d", aluno.idade);
29          printf("\n\tPeso: %.2f", aluno.peso);
30          printf("\n\taltura: %.2f", aluno.altura);
31
32  fclose(arq);
33  printf("\n\n");
34  return 0;
35  }
```

A função que faz a leitura de arquivos binário é a **fread()**. Perceba que a sintaxe da função **fread()** é a mesma da fwrite(). A Diferença é a ordem da operação. Para a fwrite() nós lemos da variável &aluno e enviamos para o arquivo 'arq'.

Na função fread() a ordem dos argumentos é a mesma, contudo, os dados são lidos do arquivo 'arq' e escritos na variável &aluno.

A função fread() por padrão faz a leitura um-a-um dos elementos. Ela lê o primeiro elemento e "anda"para o próximo. Para ilustrar, podemos imaginar que a função fread() é como um laço de repetição em um vetor, acessa elemento a elemento até o seu fim.

Nota: Lembre-se de abrir o arquivo em modo leitura+binário ('rb').

Dica: Tanto a função fwrite() quanto a fread() retornam 1 para sucesso e 0 para final de arquivo ou erro.

5.7 Posicionamento do Ponteiro do Arquivo

Para melhor compreensão deste tópico, procure sempre imaginar o arquivo binário como um vetor.

5.7.1 fseek()

Muitas vezes é necessário fazer o posicionamento do ponteiro do arquivo para um local específico a fim de ler um dado sem ter que passar por todos os outros. Isto traz muita agilidade para um programa que possui muitos registros armazenados. Imagine um banco que possui milhares de clientes ordenados pelo número da conta-corrente. Se quiséssemos ler o extrato do correntista que possui a conta 1000. Sem este recursos teríamos que passar pelas 999 antes de chegarmos neste cliente.

A linguagem C oferece uma função que posiciona o ponteiro do arquivo em um local específico dentro do arquivo. A função é **fseek()**. Sua sintaxe é:

fseek(FILE *ptrArq, long int OFFSET, int POSICAO)
Os argumentos da função são:

- **ptrArq** -> este argumento é a variável ponteiro que contém o fluxo do arquivo aberto
- **OFFSET** -> indica a quantidade de bytes a ser deslocadas a partir de uma posição no arquivo
- **POSICAO** -> indica a posição a partir da onde será deslocado o ponteiro do arquivo. Ela pode assumir apenas um dos 3 valores possíveis: **SEEK_SET**: inicio do arquivo; **SEEK_CUR**: posição corrente do arquivo; e **SEEK_END**: final do arquivo.

5.7.2 rewind()

Há também uma outra função para posicionamento de ponteiro de arquivo muito util. Esta função é a **rewind()**. Ela recebe como argumento o ponteiro do arquivo e faz com que ele volte para o início do arquivo.

5.7.3 Exemplos

Veja um exemplo onde são aplicadas ambas as funções (fseek() e rewind()):

```
1 #include <stdio.h>
2 #define TAM 3
3
4 struct pessoa{
5         char nome[30];
6         int idade;
```

```
7  };
8
9  int main(){
10
11         struct pessoa aluno[TAM], alu;
12         FILE *arq;
13         int i;
14
15         //abre o arquivo em modo binario 'w+b' para
               permitir a leitura
16         arq = fopen("teste.bin", "w+b");
17
18         //testa se a operacao foi realizada com sucesso
19         if(arq == NULL){
20                 printf("\n\tArquivo nao pode ser aberto."
                       );
21                 return 0;
22         }
23
24         for(i = 0; i < TAM; i++){
25                 printf("\n\tDigite seu nome: ");
26                 __fpurge(stdin);
27                 gets(aluno[i].nome);
28
29                 printf("\n\tDigite sua idade: ");
30                 scanf("%d", &aluno[i].idade);
31
32                 //insere a struct no arquivo
33                 fwrite(&aluno[i], sizeof(struct pessoa),
                       1, arq);
34         }
35
36         //rebobina o ponteiro para o inicio do arquivo
37         rewind(arq);
38
39         //ler o terceiro elemento (posicao 2)
40         fseek(arq, 2 * sizeof(struct pessoa), SEEK_SET);
41
42         //ler o conteudo da posicao 2 (terceiro elemento)
43         fread(&alu, sizeof(struct pessoa), 1, arq);
44
45         printf("\n\tNome: %s\n\tIdade: %d", alu.nome, alu
               .idade);
```

```
46
47          fclose(arq);
48  printf("\n\n");
49  return 0;
50  }
```

5.8 Exemplos Extras

Nesta seção são apresentados alguns códigos-fonte adicionais utilizados como exemplos dos conceitos estudados até o momento.

Exemplo 01

```
1   #include <stdio.h>
2
3   char arquivo[] = "cadastro.dat";
4
5   int main()
6   {
7           FILE *ptrfile;
8           char nome[100];
9           int idade;
10          float salario;
11
12          if((ptrfile = fopen(arquivo, "r")) == NULL){
13                  printf("\n\nArquivo nao pode ser aberto.\
                       n\n");
14                  getchar();
15                  return 1;
16          }
17
18          printf("\n\t%s \t%s \t%s\n", "Nome", "Idade", "
                Salario");
19          fscanf(ptrfile, "%s%i%f", nome, &idade, &salario)
                ;
20
21
22          while(!feof(ptrfile)){
23                  printf("\t%s\t%i\t R$ %.2f\n", nome,
                       idade, salario);
24                  fscanf(ptrfile, "%s%i%f", nome, &idade, &
                       salario);
25          }
```

```
26
27
28 fclose(ptrfile);
29 printf("\n\n");
30 return 0;
31 }
```

Exemplo 02

```
1  #include <stdio.h>
2
3  char arquivo[] = "cadastro.dat";
4
5  int main()
6  {
7          // declaracao da variavel do tipo arquivo
8          FILE *ptrfile;
9
10         char nome[100];
11         int idade;
12         float salario;
13
14         //comando para abrir o arquivo
15         // funcao fopen();
16         if((ptrfile = fopen(arquivo, "a")) == NULL){
17                 printf("\n\nArquivo nao pode ser aberto.\
                       n\n");
18                 getchar();
19                 return 1;
20         }
21
22         printf("\n\tForneca os dados: NOME, IDADE e
               SALARIO:");
23         printf("\n\tDigite <ctrl + d> para encerrar.");
24         printf("\n\t?. ");
25         scanf(" %s %i %f", nome, &idade, &salario);
26
27         while(!feof(stdin)){
28                 //fprintf(ptrfile, "%s %i %.2f\n", nome,
                       idade, salario);
29                 //funcao que escreve uma string em um
                       arquivo aberto
30                 //prototipo da funcao (arquivo, texto)
31                 fputs(nome, ptrfile); fputs(" ", ptrfile)
```

```
                    ;
32
33                  //funcao que escreve n tipo de dados
                       formatados em um arquivo
34                  fprintf(ptrfile, "%i %.2f\n", idade,
                       salario);
35                  printf("\n\t?. ");
36                  scanf(" %s %i %f", nome, &idade, &salario
                       );
37
38
39          }
40
41
42  fclose(ptrfile);
43  printf("\n\n");
44  return 0;
45  }
```

Exemplo 03 (read)

```
1   #include <stdio.h>
2
3   typedef struct{
4           char nome[50];
5           int idade;
6   } pessoa;
7
8   int main()
9   {
10          int retorno = 1;
11          char resp = 's';
12          pessoa aluno;
13          FILE *fp = fopen("alunos.bin", "rb");
14
15          if(fp != NULL){
16                  printf("\n\t--- ALUNOS CADASTRADOS ---\n"
                       );
17                  while(retorno == 1){
18
19                          retorno = fread(&aluno, sizeof(
                               pessoa), 1, fp);
20                          if(retorno == 1){
21                                  printf("\n\t%s\t%i",
```

```
                                        aluno.nome, aluno.
                                        idade);
22                              }
23                      }
24
25
26              }
27      fclose(fp);
28      printf("\n\n");
29      return 0;
30      }
```

Exemplo 03 (write)

```
1   #include <stdio.h>
2
3   typedef struct{
4           char nome[50];
5           int idade;
6   } pessoa;
7
8   int main()
9   {
10          int retorno = 1;
11          char resp = 's';
12          pessoa aluno;
13          FILE *fp = fopen("alunos.bin", "wb");
14
15          if(fp != NULL){
16
17                  while(resp == 's' || resp == 'S'){
18                          printf("\n\tNome: ");
19                          scanf(" %s", aluno.nome);
20                          printf("\n\tIdade: ");
21                          scanf("%i", &aluno.idade);
22
23                          fwrite(&aluno, sizeof(pessoa), 1,
                                fp);
24                          printf("\n\tContinuar? (s/n): ");
25                          __fpurge(stdin);
26                          resp = getchar();
27                  }
28
29
```

```
30          }
31  fclose(fp);
32  printf("\n\n");
33  return 0;
34  }
```

Referências Bibliográficas

[1] Luís Damas. Linguagem c. *10a. Edição, LTC*, 2007.

[2] F. de A. C. Pinheiro. *Elementos de Programação em C*. Bookman, 2009.

[3] Harvey M Deitel e Paul J Deitel. *Como programar em C*. LTC, 1999.

[4] N. Edelweiss e M.A.C. Livi. *Algoritmos e Programação com Exemplos em Pascal e C: Série Livros Didáticos UFRGS - Volume 23*. 23. Bookman Editora, 2014.