TÉCNICAS DE PROGRAMAÇÃO

UNIDADE 4 - ARQUIVOS

Fernando Cortez Sica

EAD.br

Introdução

Prezado cursista! Chegamos ao nosso quarto e último encontro dentro do contexto das Técnicas de Programação. Conversaremos, agora, sobre arquivos. Em muitos momentos, creio que vocês já se depararam com a necessidade de armazenar informações de forma não volátil, não é mesmo? Então, veremos nesta unidade, como podemos fazer isso.

Inicialmente, você poderá perguntar: Todo tipo de informação poderá ser salvo em arquivos? Sim, poderemos salvar, em arquivos, informações declaradas como tipos básicos (tais como *char*, *int* e *float*), assim como poderemos gravar, por exemplo, estruturas de dados (registros). Ponteiros também poderão ser salvos? Ponteiros exatamente não, pois eles apenas representam uma posição dentro da memória principal; e essa posição poderá ser alterada, a cada alocação ou a cada execução do programa, em função do gerenciamento de memória realizada pelo sistema operacional. Uma outra questão que poderia vir à sua cabeça é: Quando se abre um arquivo em algum editor de texto, existem arquivos que são possíveis de ler o seu conteúdo e, outros, que aparecem uns caracteres estranhos. Existem diferenças entre esses dois arquivos? Veremos a respeito deste assunto também, quando falarmos as diferenças entre um arquivo do tipo "texto" e um arquivo "binário". Falar sobre arquivos é o mesmo que falar sobre banco de dados? Não, um banco de dados consiste em vários arquivos gerenciados por um sistema integrador. Esse sistema, chamado de Sistema de Gerenciamento de Bancos de Dados (SGBD) permite que haja segurança, coesão, consistência, sistematização em sua organização e outros fundamentos. Sendo assim, os SGBD usam, em seu nível mais baixo, arquivos.

Vamos, então, conversar sobre arquivos?

4.1 Conceitos básicos sobre arquivos

Podemos falar que um arquivo é uma sequência de registros armazenados na memória secundária (por exemplo, no *HD – Hard Disk*); a estes registros poderão estar vinculados um número que os identificam, por exemplo, caso formos armazenar registros que representam pessoas, podemos definir, como chave de acesso (ou identificador do registro) o próprio CPF da pessoa ou algum outro código atribuído conforme a necessidade do sistema.

VOCÊ QUER VER?

Em algumas ocasiões, surge a necessidade de realizar ordenações dos valores contidos em arquivos. Uma ordenação pode resultar, posteriormente, em uma manipulação do arquivo mais eficiente, facilitando, por exemplo, o processo de busca. Alguns conceitos sobre ordenação externa poderão ser vistos neste link <https://www.youtube.com/watch?v=sVGbj1zgvWQ>.

Mas, como esses registros estão organizados no arquivo? Existem várias formas para armazená-los, a mais comum é através do armazenamento sequencial. Os arquivos representam a unidade básica de uma base de dados, neste caso, existem vários arquivos que podem ser inter-relacionados cujo gerenciamento é feito pelo SGDB (Sistema de Gerenciamento de Banco de Dados). (PUGA, 2016). Mas, vamos focar nos arquivos propriamente ditos.

VOCÊ O CONHECE?

Os arquivos são a base dos bancos de dados. Portanto, para se conhecer a fundo os SGDB – Sistema de Gerenciamento de Branco de Dados, é interessante saber como os arquivos são manipulados. Com esse conhecimento, consegue-se aproveitar melhor os seus recursos e criar consequentemente, sistemas mais eficientes. Por falar em bancos de dados, um dos modelos mais usados é o modelo relacional, criado por Edgar Frank Codd, da IBM- International Business Machines Corporation, em meados de 1970. Para conhecer um pouco mais sobre a contribuição dele, você poderá ler a dissertação de mestrado intitulada "Edgar Frank Codd and the Relational Database: a contribution to the History of Computing",disponível neste link< https://sapientia.pucsp.br/handle/handle/13305?mode=full>.

Como mencionado, um arquivo consiste em uma sequência de registros ou, de forma mais genérica, uma sequência de *bytes*. Mas, como que um sistema ou um programa sabe quantos registros estão armazenados ou qual é o último registro? Assim como as *strings* têm o seu caractere finalizador ("\x0"), os arquivos também possuem o seu delimitador que determina o seu final: *EOF* (*End-Of-File* – Final do Arquivo). A figura a seguir, retirada de Deitel (2011), ilustra esse delimitador.

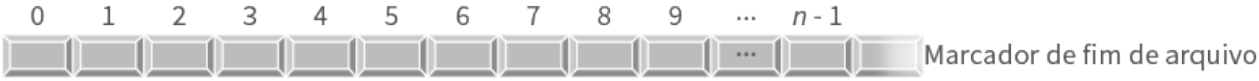

Figura 1 - Posicionamento do delimitador de arquivo EOF. Nota-se que um arquivo é formado por N registros: da posição 0 até a posição n-1.
Fonte: DEITEL, 2011. p. 352.

A figura acima mostra que, em um arquivo composto por ***N*** registros, no último campo sempre estará presente o símbolo *EOF*. Mas como usar o *EOF*? Antes de respondermos a esse questionamento, vamos falar um pouco sobre descritor de arquivo.

O sistema operacional (SO) pode manter vários arquivos em uso simultaneamente (inclusive arquivos do próprio SO. O SO, neste caso, manipula um vetor de *handles* -manipuladores). No caso da linguagem C, para instanciar um manipulador, deveremos usar um tipo de dado específico para arquivo: o tipo "*FILE*" (definido no arquivo header "*stdio.h*"). Porém, como acessaremos uma estrutura do SO, deveremos associar ao tipo "*FILE*" um ponteiro que será o nosso descritor de arquivo:

FILE *arquivo;

Esse descritor conterá, dentre outras informações inerentes ao arquivo, a posição corrente que está sendo manipulada, como se fosse o índice de um vetor. Já que mencionamos o descritor de arquivo e o delimitador *EOF*, vamos já mostrar, a seguir, algumas funções e manipulações úteis que você poderá realizar sobre os arquivos.

4.1.1 Posicionamento em um ponto do arquivo

Para efetuar o posicionamento em um ponto do arquivo pode-se utilizar a função "**fseek()**". Tal função permite posicionar no ponto do arquivo cuja posição corresponde ao deslocamento a partir de uma referência. Essa referência poderá ser:

- SEEK_SET: início do arquivo
- SEEK_CUR: posição corrente
- SEEK_END: final do arquivo

Protótipo da função:
int fseek(arq,deslocamento,referência_origem_para_o_deslocamento)
Exemplo de uso:

```
#include <stdio.h>

int main()
{
  FILE *arq;                    //descritor do arquivo
  long pos=1;                     //posicao requerida para o
posicionamento
  int ret;                       //para o teste do retorno da
funcao fseek()
  arq=fopen("teste.txt","a");   //abertura do arquivo - a ser
visto oportunamente
  if(arq==NULL)
    {
      printf("Erro de abertura do arquivo.\n");
      return 0;
    }
  ret=fseek(arq,pos,SEEK_SET);   //retorna  0  em  caso  de
sucesso de posicionamento
  if(ret!=0)
    printf("Erro de posicionamento.\n");
  fclose(arq);                  //fechamento do arquivo - a ser
visto oportunamente
  return 0;
}
```

No exemplo acima, o ponteiro de arquivo foi posicionado na primeira posição após o início do arquivo em função do uso do "**SEEK_SET**". Caso, por exemplo, tivesse sido utilizado "**SEEK_END**", o ponteiro seria posicionado na penúltima posição do arquivo.
O posicionamento na posição inicial do arquivo, pode ser realizada de duas formas:
fseek(arq,0,SEEK_SET);
rewind(arq);
Vimos como posicionar o ponteiro do arquivo, mas teria como obter a sua posição atual? Veremos isso a seguir.

4.1.2 Obtenção da posição corrente do ponteiro do arquivo

Além de posicionarmos o ponteiro do arquivo para uma posição, podemos, também, obter qual a sua posição atual. Para tanto, utiliza-se a função "**ftell()**".
Protótipo da função:
long int ftell(arq)
Exemplo de uso

```
#include <stdio.h>

int main()
{
  FILE *arq;                    //descritor do arquivo
  arq=fopen("teste.txt","r");   //abertura do arquivo - a ser
visto oportunamente
  if(arq==NULL)
    {
      printf("Erro de abertura do arquivo.\n");
      return 0;
    }
  printf("Posicao corrente: %ld.",ftell(arq));
  fclose(arq);                  //fechamento do arquivo - a ser
visto oportunamente
  return 0;
}
```

No código de exemplo acima, o retorno da função "**ftell()**" (do tipo "**long int**") é impresso na tela através da função "**printf()**".

4.1.3 Verificação de final de arquivo

Como mencionamos, a finalização do arquivo pode ser realizada mediante o teste com o delimitador "EOF". Porém, pode-se ainda, usar a função "**feof()**". A função "**feof()**" retorna 0 caso não se tenha chegado ao final de arquivo. Caso haja a marcação de final de arquivo, a função retornará um valor não nulo.
Protótipo da função:
int feof(arq)
Exemplo de uso:

```c
#include <stdio.h>

int main()
{
  FILE *arq;                    //descritor do arquivo
  char carac;
  arq=fopen("teste.txt","r");   //abertura do arquivo - a ser
visto oportunamente
  if(arq==NULL)
    {
      printf("Erro de abertura do arquivo.\n");
      return 0;
    }
  while(!feof(arq))    //retorna 0 caso não seja final de
arquivo
    {
      carac=fgetc(arq);   //fgetc() lê um caracter do arquivo
textual
      printf("%c",carac);
    }
  fclose(arq);                  //fechamento do arquivo - a ser
visto oportunamente
  return 0;
}
```

Sobre o exemplo acima, devido ao fato de estar sendo manipulado um arquivo textual, o laço formado pelo comando "**while**" poderia ser substituído por:

```c
. . .
do
  {
    carac=fgetc(arq);
    printf("%c",carac);
  } while(carac!=EOF);
. . .
```

No trecho acima, o próprio caractere lido a partir do arquivo poderá receber o código do finalizador de arquivo EOF e, desta forma, testá-lo como elemento da condição da permanência do laço de repetição.
Ainda sobre o exemplo acima, utilizou-se a função "**fgetc()**" para realizar a leitura de um caractere do arquivo. A mesma ação poderia ser feita com o uso da função "**fscanf()**" – que se comporta de forma análoga à função "**scanf()**". As funções "**fgetc()**" e "**fscanf()**" seguem os seguintes modos de utilização:

int fgetc(arq);
int fscanf(arq,"lista_de_formatações_com_%",lista_de_vars_por_referência);
O exemplo a seguir mostra melhor a utilização das funções "**fgetc()**" e "**fscanf()**":

```
#include <stdio.h>

int main()
{
  FILE *arq;
  char carac,str[15];
  int valor;
  arq=fopen("exemplo.txt","r");
  if(arq==NULL)
    {
      printf("Erro de abertura do arquivo.\n");
      return 0;
    }
  carac=fgetc(arq);  //coleta um caracter do arquivo
  fscanf(arq,"%d%s",&valor,str);  //coleta um inteiro e uma
string do arquivo
  printf("Valores lidos:  %c %d %s.\n",carac,valor,str);
  fclose(arq);
  return 0;
}
```

O exemplo acima realiza a leitura de informações a partir de um arquivo cuja linha lida corresponde a:
a 1234 bc
Podemos perceber, então, que a função "**fscanf()**" realiza a entrada formatada de dados como se estivéssemos inserindo informações a partir do teclado, sendo assim, o fluxo de entrada deixa de ser o teclado, para ser o arquivo. A função "**fgetc()**", que permite a leitura de um caractere do arquivo, poderia ser substituída por "**fscanf (arq,"%c",&carac)**" ou, ainda, ter adicionado essa operação em: "**fscanf(arq,"%c%d%s",&carac,&valor,str)**".
Todas essas funções e manipulações serão exemplificadas adiante, após falarmos de algumas outras questões importantes. Para iniciar, podemos mencionar que, para a manipulação de um arquivo, teremos que seguir uma sequência fixa. Clique nos *cards* e confira mais sobre o tema.

Abertura

A abertura permite definir o nome do arquivo e as formas de abertura; para que possamos manipular um arquivo, devemos sempre abri-lo anteriormente.

Manipulação

A própria manipulação para gravar, buscar, apagar, modificar, criar registros.

Fechamento

Encerra a manipulação do arquivo.

Veremos, a seguir, essas manipulações de forma aprofundada.

4.2 Abertura de um arquivo

Como já foi adiantado, todo arquivo, para ser manipulado, necessita ser aberto. A abertura consiste em instanciar o descritor de arquivo e se dá através da função "**fopen()**". Para tanto, a referida função tem o seguinte protótipo:

FILE *fopen("nome_do_arquivo","tipo_de_abertura");

Caso o arquivo não possa ser aberto, a função "**fopen()**" retorna um ponteiro nulo ("**NULL**"). Em operações normais, o descritor de arquivo será corretamente instanciado para que seja utilizado no decorrer do programa.

Para reforçar a utilização da função "**fopen()**", vamos verificar o código a seguir:

```
#include <stdio.h>

int main()
{
  FILE *arq;                    //descritor do arquivo
  arq=fopen("teste.txt","r");   //abertura do arquivo
  if(arq==NULL)
    {
      printf("Erro de abertura do arquivo.\n");
      return 0;
    }
  fclose(arq);                  //fechamento do arquivo
  return 0;
}
```

No código de exemplificação acima, podemos destacar dois fatos. Clique e confira quais são eles.

"fclose()" cujo protótipo é: "int fclose(ponteiro_arquivo)". A refererida função retorna 0 caso o fechamento tenha sido bem-sucedido. Caso contrário, retornará um código de erro.

No caso do exemplo acima, a função "fopen()" retorna um ponteiro para a manipulação do arquivo intitulado "teste.txt" que foi aberto no modo "r".

Mas, o que vem a ser esse modo "**r**" de abertura? Veremos a seguir os modos nos quais poderemos efetuar a abertura de um arquivo.

4.3 Modos de abertura

Um arquivo, na linguagem C, pode ser aberto de três formas distintas: "**r**", "**w**" e "**a**". Essas letras são as iniciais de "*read*" (ler), "*write*" (escrever) e "*append*" (anexar). Mas, então, quais as diferenças entre eles? Vamos sumarizar na sequência. Clique para conferir.

- **Modo "r":**

  O arquivo, nesse caso, é aberto para o modo somente leitura. Nesse modo, o arquivo já deve existir.

- **Modo "w"**

  No modo "**w**", é criado um novo arquivo, sendo aberto para a escrita. Caso um arquivo com o mesmo nome já exista na pasta, ele é destruído e aberto um novo vazio.

- **Modo "a"**

  Neste modo, um arquivo é aberto (quando já existir) ou criado (na inexistência prévia do arquivo) para a escrita. Quando usado esse modo, o ponteiro é posicionado ao final do arquivo, permitindo-se assim, a anexação de outros registros após o último previamente gravado.

Além de "**r**", "**w**" e "**a**", poderemos acrescentar o sinal "**+**" e a letra "**b**". Ao acrescentarmos o sinal "**+**", estamos liberando a permissão tanto de escrita quanto de leitura. Por sua vez, a letra "**b**" denota a utilização de arquivos binários, os quais conversaremos adiante.
O quadro a seguir, extraída de Ascencio (2012), sumariza os modos de abertura de um arquivo utilizando-se a linguagem C de programação.

Formato	Descrição
r	Abre um arquivo de texto onde poderão ser realizadas apenas leituras.
w	Cria um arquivo de texto onde poderão ser realizadas apenas operações de escrita.
a	Anexa novos dados a um arquivo de texto.
rb	Abre um arquivo binário onde poderão ser realizadas apenas leituras.
wb	Cria um arquivo binário onde poderão ser realizadas apenas operações de escrita.
ab	Anexa novos dados a um arquivo binário.
r+	Abre um arquivo de texto onde poderão ser realizadas operações de leitura e de escrita.
w+	Cria um arquivo de texto onde poderão ser realizadas apenas operações de leitura e de escrita.
a+	Anexa novos dados ou cria um arquivo de texto para operações de leitura e de escrita.
rb+	Abre um arquivo binário onde poderão ser realizadas operações de leitura e de escrita.
wb+	Cria um arquivo binário onde poderão ser realizadas apenas operações de leitura e de escrita.
ab+	Anexa novos dados a um arquivo binário para operações de leitura e de escrita.

Quadro 1 - Modos de abertura de um arquivo utilizando-se a linguagem C. Nota-se que é possível fazer combinações dos modos "r", "w" e "a" com os símbolos "+" e "b".
Fonte: ASCENCIO, 2012. p. 423.

O código a seguir ilustra a utilização do modo de abertura:

```c
#include <stdio.h>

int main()
{
  FILE *arq;
  char palavra[10];
  arq=fopen("teste.txt","a+");
  if(arq==NULL)
    {
      printf("Erro de abertura do arquivo.\n");
      return 0;
    }
  printf("Palavra a ser gravada no arquivo:  ");
  scanf("%s",palavra);
  fprintf(arq,"%s",palavra);
  if(ferror(arq))
    printf("Erro na gravacao.");
  fclose(arq);
  return 0;
}
```

No exemplo acima, temos um arquivo denominado "**teste.txt**" sendo aberto para permitir as operações de leitura e escrita ao seu final (modo "**a+**"). Neste caso, a palavra fornecida pelo usuário através da função "**scanf()** " é gravado, ao final do arquivo, por intermédio da função "**fprintf()**".

Nesse exemplo, introduzimos uma nova função: a "**ferror()**"; cujo protótipo é:

int ferror(arquivo)

pode ser colocada após cada operação de manipulação de arquivo para retornar um código de erro (caso a operação tenha sido malsucedida) ou retorna 0 caso a última operação tenha sido executada normalmente.

Mas se o modo de abertura fosse "**r+**" ao invés de "**a+**"? Neste caso, a gravação da palavra ocorreria no início do arquivo, sobrescrevendo ao conteúdo das posições iniciais, caso o arquivo não estivesse vazio.

Ainda no exemplo acima, foi utilizada a função "**fprintf()**". Essa função funciona da mesma forma que a função " **printf()**" com a diferença que a sua saída é direcionada para o arquivo passado como primeiro parâmetro, no caso, o ponteiro para o descritor de arquivo "**arq**". (MIZRAHI, 2008). Para o seu uso, usa-se:

int fprintf(arq,"lista_de_formatações_com_%",lista_de_variáveis);

No exemplo a seguir, poderemos perceber melhor a sua utilização.

```c
#include <stdio.h>

int main()
{
   FILE *arq;
   char carac='a',str[20]="Apenas um teste";
   int valor=54778;
   arq=fopen("exemplo1.txt","w");
   if(arq==NULL)
     {
       printf("Erro de abertura do arquivo.\n");
       return 0;
     }
   fprintf(arq,"%c %d %s\n",carac,valor,str);
   fclose(arq);
   return 0;
}
```

Nota-se, no exemplo acima, que a utilização da função "**fprintf()**" é análoga à função "**printf()**" com a diferença do direcionamento do fluxo para o arquivo "**arq**". A saída obtida por esse programa é apresentada abaixo, onde poderemos notar que, apesar de ser utilizada, por exemplo, uma variável inteira, a gravação no arquivo foi realizada no formato de texto (*string*):

a 54778 Apenas um teste

Para permitir a manipulação de *strings* e caracteres, além das funções "**fprintf()**" e "**fscanf()**" podemos usar, também, as funções "**fputs()**", "**fgets()**", "**fputc()**" e "**fgetc()**". Tais funções estão sumarizadas no quadro abaixo.

Função	Objetivo	Protótipo
fputs()	Escreve, no arquivo, a *string* str. Em caso de sucesso, retorna um valor inteiro não negativo. Em caso de erro, retorna *EOF*.	int fputs(char *str,FILE *arq)
fgets()	Leitura de uma *string* de tamanho tam. Em caso de sucesso, a função retorna a própria *string* str lida.	char *fgets(char *str,int tam,FILE *arq)
fputc()	Grava o caractere "carac" no arquivo. Em caso de sucesso, retorna o próprio caractere "carac". Em caso de falha, retorna EOF.	int fputc(int carac,FILE *arq)
fgetc()	Retorna o caractere lido. No caso de ter encontrado o término do arquivo ou na ocorrência de uma condição de erro, a função retorno *EOF*.	int fgetc(FILE *arq)

Quadro 2 - Funções básicas para a escrita e leitura em arquivos do tipo texto.
Fonte: Elaborada pelo autor, 2019.

Até o momento, apenas efetuamos a leitura ou a gravação de caracteres ou *strings* no arquivo. Seria possível trabalharmos com manipulação de outros tipos de dados em qualquer posição do arquivo? Responderemos a esse questionamento a seguir.

4.4 Acesso direto a arquivo: localizar, alterar, excluir e incluir registros

Antes de falarmos sobre a localização, alteração, exclusão e inclusão de registros, vamos apresentar duas novas funções: a função "**fread()**" e a função "**fwrite()**". Ambas as funções permitem a manipulação (leitura e escrita, respectivamente) de informações de tipos distintos (tais como valores do tipo "**float**", dados organizados em estruturas – "**struct**"). Essas funções seguem os seguintes protótipos:
unsigned int fread(&dados_lidos, tamanho_de_um_item, qtd_itens,arq)
unsigned int fwrite(&dados_para_gravar, tamanho_de_um_item, qtd_itens,arq)
Para exemplificar o uso das funções "**fread()**" e "**fwrite()**", vamos supor que exista um arquivo onde cada linha contém idades de pessoas identificadas pelo seu CPF. Essa mesma estruturação de arquivo será utilizada para demais exemplos de manipulação de registros. A figura a seguir ilustra um possível conteúdo do arquivo citado.

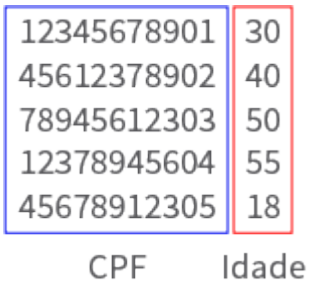

Figura 2 - Exemplo do conteúdo do arquivo que será usado para exemplificar o uso das funções "fread()" e "fwrite()" assim como as demais manipulações desta seção.
Fonte: Elaborada pelo autor, 2019.

Para o caso, o campo de CPF será manipulado como uma *string* de onze posições e, o campo de idade, como uma informação do tipo "**int**".
A partir da descrição de como será o arquivo, vamos falar como podem ser implementadas as funcionalidades de criação, localização, alteração, remoção e inclusão de novos registros.

4.4.1 Criação dos registros iniciais

O primeiro exemplo que abordaremos será a criação do arquivo para a inclusão de dois registros iniciais. Para certificação, serão realizadas as suas leituras.

```c
#include <stdio.h>
#include <string.h>

typedef struct
{
  char cpf[12];
  int idade;
} IPessoa;

int IncluirRegistro(char cpf[12],int idade, FILE *arq)
{
  IPessoa pessoa;
  strcpy(pessoa.cpf,cpf);
  pessoa.idade=idade;
  if(!fwrite(&pessoa,sizeof(IPessoa),1,arq))
    return -1;
  return 0;
}

int main()
{
  FILE *arq;
  IPessoa pessoa;
  arq=fopen("registros.dat","w+");
  if(arq==NULL)
    {
      printf("Erro de abertura do arquivo.\n");
      return 0;
    }
  if(IncluirRegistro("12345678901",30,arq)==-1)
    {
      printf("Erro de gravacao do registro.");
      return 0;
    }
  if(IncluirRegistro("45612378902",40,arq)==-1)
    {
      printf("Erro de gravacao do registro.");
```

No exemplo acima, tivemos a inclusão dos dois primeiros registros formados pelos campos "**cpf**" e "**idade**". Nota-se que a função "**fwrite()**" requer a utilização de passagem de parâmetro por referência para o item a ser gravado. Esse fato é motivado pela necessidade de tornar a função compatível para qualquer tipo de dado, inclusive se o item a ser salvo for uma "**struct**".
Mas, como localizar, alterar e incluir novos registros? Vamos ver essas funcionalidades a seguir.

4.4.2 Localização de um registro

Para exemplificar a operação de localização de registros, vamos tomar como exemplo o mesmo arquivo de dados usado na etapa de criação do arquivo e inclusão de dois registros iniciais. O código a seguir ilustra a operação de localização de registros:

```c
#include <stdio.h>
#include <string.h>

typedef struct
{
  char cpf[12];
  int idade;
} TPessoa;

long LocalizaRegistro(char cpf[12], TPessoa *pessoa,FILE
*arq)
{
  long pos;
  rewind(arq);
  while(1)
    {
      pos=ftell(arq);
      fread(pessoa,sizeof(TPessoa),1,arq);
      if(feof(arq))
        break;
      if(!strcmp(pessoa->cpf,cpf))
        return pos;
    }
  return -1;   //nao achado
}

int main()
{
  FILE *arq;
  TPessoa pessoa;
  long pos;
  char cpf[12];
  arq=fopen("registros.dat","r");
  if(arq==NULL)
    {
      printf("Erro de abertura do arquivo.\n");
      return 0;
    }
  printf("CPF a ser localizado: ");
  scanf("%s",cpf);
  pos = LocalizaRegistro(cpf,&pessoa,arq);
  if(pos==-1)
    printf("Registro nao encontrado.\n");
  else
    printf("Encontrado posicao = %ld\tCPF = %s\tidade = %d",
           pos,pessoa.cpf,pessoa.idade);
  fclose(arq);
  return 0;
}
```

No código acima, temos a funcionalidade de localização de um registro. Nota-se que a instanciação da variável " pos" é realizada antes da leitura. Isso é devido ao fato de que, após a leitura, a posição do ponteiro de arquivo já apontará para o próximo registro independentemente se o registro buscado foi ou não encontrado.
Essa função poderá ser utilizada na funcionalidade de alteração de um registro, como veremos a seguir.

4.4.3 Alteração de um registro

Para se alterar um registro, temos que, antes, localizá-lo. Sendo assim, usaremos a função de localização já implementada. Com essa função, já se sabe qual a posição será sobrescrita com o valor devidamente modificado.
A funcionalidade de alteração de registro pode ser visualizada no exemplo a seguir.

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

typedef struct
{
  char cpf[12];
  int idade;
} TPessoa;


long LocalizaRegistro(char cpf[12], TPessoa *pessoa,FILE *arq)
{
  long pos;
  rewind(arq);
  while(1)
    {
      pos=ftell(arq);
      fread(pessoa,sizeof(TPessoa),1,arq);
      if(feof(arq))
        break;
      if(!strcmp(pessoa->cpf,cpf))
     return pos;
    }
  return -1;   //nao achado
}


int AlteraRegistro(long pos, TPessoa pessoa, FILE *arq)
{
  fseek(arq,pos,SEEK_SET);
  if(!fwrite(&pessoa,sizeof(TPessoa),1,arq))
    return -1;   //erro de escrita
  return 0;
}

int main()
{
  FILE *arq;
  TPessoa pessoa;
  long pos;
  char cpf[12],resp;
  int ret;
  arq=fopen("registros.dat","r+");
  if(arq==NULL)
    {
      printf("Erro de abertura do arquivo.\n");
      return 0;
    }
  printf("CPF a ser alterado: ");
  scanf("%s",cpf);
```

```
  pos = LocalizaRegistro(cpf,&pessoa,arq);
  if(pos==-1)
    {
      printf("Registro nao encontrado.\n");
      return 0;
    }
  printf("CPF = %s\tIdade = %d.\n",pessoa.cpf,pessoa.idade);
  do
    {
      printf("Manter a idade <s><n>?");
      scanf("%c",&resp);
      resp=toupper(resp);
    }while((resp!='S')&&(resp!='N'));
  if(resp=='N')
    {
      printf("Digite a nova idade: ");
      scanf("%d",&pessoa.idade);
      ret = AlteraRegistro(pos,pessoa,arq);
      if(ret==-1)
        printf("Erro de atualizacao do registro.\n");
    }
  printf("Testando alteracao...\n");
  fseek(arq,pos,SEEK_SET);
  fread(&pessoa,sizeof(TPessoa),1,arq);
  printf("CPF = %s idade = %d\n",pessoa.cpf,pessoa.idade);
  fclose(arq);
  return 0;
}
```

No código acima, temos um exemplo para a modificação de um registro previamente gravado. Como a função de localização já retorna a posição do registro a ser modificado, a função de alteração realiza apenas o posicionamento no registro a ser alterado e efetua a sua gravação. Na gravação todo o registro é sobrescrito, modificando consequentemente o seu valor.

Nota-se que, no código acima, foi usada a forma "**r+**" para a abertura de arquivo para permitir tanto a leitura quanto a gravação de registros. Caso, por exemplo, fosse aberto com "**a+**", a gravação ocorreria sempre após o último registro mesmo com a execução da função "**fseek()**".

Por fim, como realizar a remoção de um item? Veremos esse assunto a seguir.

4.4.4 Remoção de um registro

Para a remoção de um registro, pode-se, simplesmente, alterar o conteúdo do registro a ser excluído para um conteúdo vazio ou não válido. No caso do código anterior, aproveitando-se as funções de localização e alteração de registro, o programa principal ficaria como listado a seguir:

```
. . .
. . .

int main()
{
  FILE *arq;
  TPessoa pessoa;
  long pos;
  char cpf[12],resp;
  int ret;
  arq=fopen("registros.dat","r+");
  if(arq==NULL)
    {
      printf("Erro de abertura do arquivo.\n");
      return 0;
    }
  printf("CPF a ser removido: ");
  scanf("%s",cpf);
  pos = LocalizaRegistro(cpf,&pessoa,arq);
  if(pos==-1)
    {
      printf("Registro nao encontrado.\n");
      return 0;
    }
  printf("CPF = %s\tIdade = %d.\n",pessoa.cpf,pessoa.idade);
  do
    {
      printf("Remover o registro <s><n>?");
      scanf("%c",&resp);
      resp=toupper(resp);
    }while((resp!='S')&&(resp!='N'));
  if(resp=='S')
    {
      strcpy(pessoa.cpf,"00000000000");
      pessoa.idade=0;
      ret = AlteraRegistro(pos,pessoa,arq);
      if(ret==-1)
        printf("Erro de atualizacao do registro.\n");
    }
  printf("Testando alteracao...\n");
  fseek(arq,pos,SEEK_SET);
  fread(&pessoa,sizeof(TPessoa),1,arq);
  printf("CPF = %s idade = %d\n",pessoa.cpf,pessoa.idade);
  fclose(arq);
  return 0;
}
```

Pode-se então, aproveitar todas essas funções para que seja implementada a funcionalidade de inclusão de novos registros? Veremos a seguir.

4.4.5 Inclusão de um novo registro

Caso a remoção tenha sido implementada através da gravação de um valor nulo sobre o registro removido, então, para a inserção de um novo registro, pode-se procurar pelos espaços vagos, ou seja, procurar por registros cujos valores de CPF sejam "**00000000000**". Em tais espaços, realiza-se a modificação incluindo-se valores válidos relativos aos novos registros.
Sendo assim, o código para a inserção de um novo registro ficaria:

```
int InsereNovoRegistro(TPessoa pessoa, FILE *arq)
{
  long pos;
  TPessoa p;
  pos = LocalizaRegistro("00000000000",&p,arq);
  if(pos!=-1)
    return AlteraRegistro(pos,pessoa,arq);
  fseek(arq,0,SEEK_END);
  if(!fwrite(&pessoa,sizeof(TPessoa),1,arq))
    return -1;
  return 0;
}
```

Como foi mencionado anteriormente, a inserção de novos registros ocorre nos espaços dos registros anteriormente apagados (aqui representados pelos registros com CPF igual a "**00000000000**"). Caso não existam registros nulos, a inserção ocorre após o último registro do arquivo.

VOCÊ QUER LER?

Em muitas ocasiões, faz-se necessário saber o tamanho de um arquivo; porém, a linguagem C não tem funções com esse objetivo. Para saber calcular, em C, o tamanho de um arquivo, você poderá usar a dica postada por Lima (2006), disponível neste link <https://allanlima.wordpress.com/2006/07/15/calculando-o-tamanho-de-um-arquivo-em-c-2/>.

Anteriormente, mencionamos os temos "*arquivo de texto*" e "*arquivo binário*"; inclusive, falamos quando estávamos abordando os tipos de abertura de arquivo que, ao colocar o símbolo "b" estamos a manipular arquivos binários. Vamos, então, conversar sobre o que vem a ser esses dois tipos de arquivos.

4.5 Arquivo de texto

Apesar de termos mencionado, quando falávamos sobre os modos de abertura de um arquivo, que existem os arquivos texto e binário, até o momento, não fizemos distinção entre esses dois formatos. Mas, qual a diferença entre eles?

VOCÊ SABIA?

Você sabia que os arquivos ".CSV" (*Comma Separated Values* – Valores Separados por Vírgula) são arquivos textuais em podem ser manipulados usando a linguagem C? Para ver um exemplo de código para a manipulação de arquivos ".CSV" você poderá acessar este link <https://www.vivaolinux.com.br/script/Manipulacao-de-arquivos-CSV-Estruturado>.

No arquivo em formato texto, como o próprio nome diz, seu conteúdo consiste em um texto puro, que poderá ser aberto em qualquer editor de texto. Sendo assim, temos que usar, em tais arquivos, apenas as funções que manipulam caracteres e *strings*.
O exemplo, abaixo ilustra um caso onde um arquivo textual é criado.

```c
#include <stdio.h>
#include <string.h>

int main()
{
  char nome[30], aniversario[9];
  FILE *arq;
  arq=fopen("aniversarios.txt","w");
  if(arq==NULL)
    {
      printf("Erro de abertura de arquivo.");
      return 0;
    }
  while(1)
    {
      printf("Nome (<digite 'fim' para finalizar>): ");
      gets(nome);
      if(!strcmp(nome,"fim"))
        break;
      else
     {
       printf("Data aniversario (dd/mm/aa): ");
       gets(aniversario);
       fprintf(arq,"%s - %s\n",nome,aniversario);
     }
    }
  fclose(arq);
  return 0;
}
```

Nota-se, no exemplo anterior, a utilização para a gravação no arquivo apenas de funções manipulando variáveis do tipo *strings*. Mas qual foi o motivo de utilizarmos a função "**gets()**" e não "**scanf()**" para a entrada das informações? A função "**scanf()**" divide a entrada digitada em palavras, ou seja, não aceita o espaço em branco como parte de apenas uma informação digitada. Por sua vez, com a função "**gets()**", já é possível incluir os espaços em branco dentro de uma única informação. Para a gravação no arquivo, foi utilizada a função "**fprintf()** " assim como poderia ser utilizada a função "**fputs()**".

Mencionamos, anteriormente, a remoção de um registro de um arquivo. Como remover uma informação de um arquivo texto? No exemplo do registro, realizamos a sobescrita de um registro com um valor não válido para a representar a sua remoção. No caso de um arquivo texto, que pode ser considerado como um arquivo sequencial para a remoção de um item, precisamos criar um arquivo temporário para receber todas as informações exceto aquelas a serem retiradas. O exemplo a seguir realiza a remoção do aniversário de uma pessoa cujo arquivo foi construído com o exemplo anterior.

```c
#include <stdio.h>
#include <string.h>

int main()
{
  char linha[39],nome_ret[30];
  FILE *arq,*arqtmp;
  arq=fopen("aniversarios.txt","r");
  if(arq==NULL)
    {
      printf("Erro de abertura de arquivo.");
      return 0;
    }
  arqtmp=fopen("aniversarios_tmp.txt","w");
  if(arqtmp==NULL)
    {
      printf("Erro de abertura do arquivo temporario.");
      return 0;
    }
  printf("Nome a ser retirado da lista de aniversarios: ");
  gets(nome_ret);
  while(1)
    {
      fgets(linha,39,arq);   //fgets() lê toda a linha até o
<enter>
      if(feof(arq))
        break;
      if(strncmp(linha,nome_ret,strlen(nome_ret)))
        fprintf(arqtmp,"%s",linha);
    }
  fclose(arq);
  fclose(arqtmp);
  if(remove("aniversarios.txt"))  //devolve  0  em  caso  de
sucesso
    printf("Erro na remocao do arquivo.\n");
  else
    if(rename("aniversarios_tmp.txt",   "aniversarios.txt"))
//devolve 0

//em caso de sucesso
       printf("Erro no renomeamento do arquivo.\n");
  return 0;
}
```

No código acima, temos a presença de duas funções que manipulam diretamente o arquivo, mas não os registros. Trata-se das funções "remove**()**" e "**rename()**". Essas duas funções possuem os seguintes modos de uso:
remove(nome_arquivo_a_ser_removido);
rename(nome_antigo , novo_nome);
Esse processo de utilização de um arquivo temporário é típico na manipulação de arquivos sequenciais. Um arquivo sequencial é aquele que necessitamos percorrer por todo o conteúdo até chegarmos à um ponto específico.

4.6 Arquivo binário

A manipulação de um arquivo binário torna-se mais eficiente em função da possibilidade de manipulação de registros. Desta forma, pode-se manipular tais arquivos de forma não sequencial, ou seja, de forma aleatória. Neste caso, o próprio código de acesso de um registro representa o próprio deslocamento a ser utilizado através da função "**fseek()**".

Uma outra vantagem do uso de arquivos binários consiste na possibilidade de redução de espaço armazenado. Por exemplo, caso necessite armazenar um valor inteiro de valor 123.457, no arquivo texto gastaríamos seis *bytes*; um *byte* para cada dígito, ou sete bytes, se considerarmos o ponto. Já nos arquivos binários, o tamanho gasto é exatamente o tamanho do tipo do dado, no caso, gastaríamos "**(sizeof(int))**". Esse gasto a mais impactaria, consequentente, no tempo de leitura ou gravação. Sendo assim, uma outra vantagem dos arquivos binários consiste no fato de que eles demandam de menos tempo computacional para realizar suas operações de leitura e escrita.

CASO

Uma certa empresa familiar manipulava as suas informações totalmente de forma arcaica. Todo gerenciamento do faturamento e débitos era realizado de forma quase manual, através de relatórios textuais. Um profissional da área foi chamado para tentar automatizar o processo; a primeira providência tomada foi de implementar um sistema que, para interpretar os lançamentos, tinha que quebrar as informações dos relatórios em "*tokens*" (*token* representa um pedaço da informação, por exemplo, uma palavra). Porém esse processamento textual demandava muito computacional. Diante da lentidão do sistema, ele resolveu transformar os relatórios em arquivos binários para melhor manipular os registros; os quais representavam as despesas e as entradas de valores. Com os arquivos binários, através da manipulação de seus registros, o processamento tornou-se mais rápido, flexível e ocupava menos espaço de armazenamento.

Para os arquivos binários, abertos com a inserção do "b" no modo de abertura da função "**fopen()**", não podemos usar, por exemplo, as funções "**fputs()**" e "**fgets()**". E, sim, teremos que usar as funções "**fread()**" e "**fwrite()**" como utilizadas em vários exemplos ao longo da unidade.

O exemplo a seguir, ilustra a gravação e recuperação de registros em um arquivo binário onde a primeira posição do arquivo contém a quantidade de registros armazenados. As demais posições são representadas pelos registros, no caso, sequências formadas por um valor inteiro e um valor do tipo *float*.

```c
#include <stdio.h>

int GravaDados(int vi,float vf,FILE *arq)
{
  if(!fwrite(&vi,sizeof(int),1,arq))
    return -1;
  if(!fwrite(&vf,sizeof(float),1,arq))
    return -1;
  return 0;
}

int main()
{
  FILE *arq;
  int qtd=2,valor_int,i;
  float valor_float;
  arq=fopen("exemplobinario.dat","wb+");
  if(arq==NULL)
    {
      printf("Erro de abertura do arquivo.\n");
      return 0;
    }
  //gravando os dados
  fwrite(&qtd,sizeof(int),1,arq);    //qtd de itens a serem
gravados
  GravaDados(23,45.9,arq);
  GravaDados(1256,500.987,arq);
  //recuperando os dados para testar
  rewind(arq);
  fread(&qtd,sizeof(int),1,arq);
  for(i=0; i<qtd; i++)
    {
      fread(&valor_int,sizeof(int),1,arq);
      fread(&valor_float,sizeof(float),1,arq);
      printf("registro                [%d]:                int=%d
float=%f\n",i,valor_int,valor_float);
    }
  fclose(arq);
  return 0;
}
```

Nota-se, no exemplo acima, que podemos manipular qualquer tipo de dado em qualquer ocasião através das funções "**fwrite()**" e "**fread()**". Para tanto, devemos saber como o arquivo encontra-se estruturado.

Síntese

Chegamos ao fim de nossa quarta e última conversa sobre técnicas de programação. Nesta, pudemos abranger um outro ponto importante da programação que é a manipulação de arquivos. Com os pontos abordados, realizamos operações de gravação e recuperação de informações em memória não volátil, por exemplo, manipulando arquivos gravados no HD – *hard disk*. Desta forma, você poderá, agora, implementar programas com amplas funcionalidades desde a manipulação de estruturas mais complexas até a sua gravação e recuperação em arquivos.

Com os conteúdos apresentados nesta quarta unidade, esperamos que você construa programas mais complexos, envolvendo a amplidão de recursos que a linguagem C proporciona.

Nesta unidade, você teve a oportunidade de:

- ter contato com conceitos inerentes aos arquivos;
- saber identificar e usar as funções para a manipulação de arquivos;
- identificar, sugerir e implementar soluções de *software* através da manipulação de arquivos;
- empregar corretamente as funções e estruturas de programação para a localização, inserção, remoção, alteração de registros em arquivos;
- saber diferenciar e utilizar um arquivo texto de um arquivo binário.

Bibliografia

ASCENCIO, A. F. G. **Fundamentos de Programação de Computadores: Algoritmos, PASCAL, C/C++ (Padrão ANSI) e Java**. 3. ed. São Paulo: Pearson Education do Brasil, 2012. Disponível em: <https://laureatebrasil.blackboard.com/>. Acesso em: 21/07/2019.

BACKES, A. **Ordenação Externa**. Aula 66. 14:31min. Linguagem C Programação Descomplicada, 2014. Disponível em: <https://www.youtube.com/watch?v=sVGbj1zgvWQ>. Acesso em: 30/07/2019.

CANDIDO, F. A. S. Manipulação de Arquivos CSV – Estruturado. 2010. Disponível em: <https://www.vivaolinux.com.br/script/Manipulacao-de-arquivos-CSV-Estruturado>. Acesso em: 30/07/2019.

DEITEL, P. J.; DEITEL, H. **C: Como Programar**. 6. Ed. São Paulo: Pearson Prentice Hall, 2011. Disponível em: <https://laureatebrasil.blackboard.com/>. Acesso em: 21/07/2019.

LIMA, A. D. S. **Calculando o Tamanho de um Arquivo em C**. 2006. Disponível em: <https://allanlima.wordpress.com/2006/07/15/calculando-o-tamanho-de-um-arquivo-em-c-2/>. Acesso em: 30/07/2019.

MIZRAHI, V. V. **Treinamento em Linguagem C**. 2. Ed. São Paulo: Pearson Prentice Hall, 2008. Disponível em: <https://laureatebrasil.blackboard.com/>. Acesso em: 21/07/2019.

PUGA, S.; RISSETTI, G. **Lógica de Programação e Estruturas de Dados – com Aplicações em Java**. 3 ed. São Paulo: Pearson Education do Brasil, 2016. Disponível em: <https://laureatebrasil.blackboard.com/>. Acesso em: 21/07/2019.

SOUZA, O. **Edgar Frank Codd e o Banco de Dados Relacional**: uma contribuição para a História da Computação. 2015. Dissertação (Mestrado em História da Ciência). Pontífica Universidade Católica de Sao Paulo. São Paulo: 2015. Disponível em <https://sapientia.pucsp.br/handle/handle/13305?mode=full>. Acesso em: 30/07/2019.