Instituto Federal do Amazonas - IFAM
Curso: Tecnologia em Análise e Desenvolvimento de Sistema
Professor: Sérgio Augusto C. Bezerra
Disciplina: Estrutura de Dados

Arquivos de Dados

1. Introdução

Muitas pesquisas precisam ler ou escrever informações de um dispositivo de memória auxiliar. Essas informações são armazenadas no dispositivo na forma de um **arquivo de dados**. Os arquivos de dados nos permitem armazenar informações permanentemente e acessar e alterar essas informações sempre que necessário.

Ao contrário de outras linguagens de programação, C não distingue arquivos de dados de acesso seqüencial de arquivos de dados de acesso direto (acesso randômico). Há, contudo, dois tipos diferentes de arquivos de dados, chamados **stream oriented (ou padrão*)*** e **system oriented (ou baixo nível*)***. Os arquivos de dados stream são, normalmente, mais fáceis de trabalhar que os arquivos system e, portanto, são mais comumente usados.

Os arquivos de dados stream podem ser divididos em duas categorias. Na primeira, estão os ***arquivos de dados formados por caracteres consecutivos***. Esses caracteres podem ser interpretados como itens de dados individuais ou como componentes de strings ou números. A maneira como esses caracteres são interpretados é determinada pelas funções de biblioteca usadas para transferir a informação ou pelas especificações de formato nas funções de biblioteca, como, por exemplo, nas funções *scanf* e *printf*.

A segunda categoria de arquivos de dados stream é, normalmente, chamada de ***arquivos de dados não formatados***, e os dados são organizados em blocos contendo bytes contíguos de informação. Esses blocos representam estruturas de dados mais complexas, como matrizes ou estruturas. Um conjunto diferente de funções de biblioteca está disponível para o processamento de arquivos de dados stream desse tipo. Essas funções de biblioteca fornecem instruções individuais que podem transferir matrizes ou estruturas inteiras de/ou para os arquivos de dados.

Os arquivos de dados system são mais intimamente relacionados ao sistema operacional do computador que os arquivos stream. Eles são um pouco mais difíceis de trabalhar, apesar de poderem ser mais eficientes em certos tipos de aplicações. É necessário um conjunto de procedimentos separados, em determinadas funções de bibliotecas, para processar arquivos de dados system.

2. Abrindo e Fechando um arquivo de dados

Para trabalhar com um arquivo de dados stream, o primeiro passo é estabelecer um buffer, onde as informações são armazenadas temporariamente durante a transferência entre a memória do computador e o arquivo de dados. Esse buffer permite que informações sejam lidas de/ou escritas em arquivo de dados mais rapidamente. O buffer é estabelecido escrevendo-se

FILE *varpont;

Onde:

- **FILE**: é um tipo de estrutura especial que estabelece o buffer, e definida no arquivo stdio.h.
- **varpont**: é uma variável ponteiro que indica o início do buffer, e normalmente referenciado como um ponteiro stream ou simplesmente um stream.

Um arquivo de dados deve, então, ser aberto antes de poder ser criado ou processado, o que associa o nome do arquivo ao buffer (isto é, à stream). A abertura do arquivo também especifica como o arquivo de dados será utilizado, ou seja, se o arquivo será aberto apenas para leitura (ready-only), apenas para gravação (write-only) ou para leitura e gravação (ready/write), em que as duas operações são permitidas. Para abrir um arquivo escreve-se

varpont = fopen(nome_arquivo, tipo_abertura);

Onde:

- **fopen**: a função de biblioteca usada para abrir um arquivo.
- **nome_arquivo**: o nome do arquivo de dados.
- **tipo_abertura**: é a maneira como o arquivo de dados será utilizado.

A função **fopen** retorna um ponteiro para o início do buffer associado ao arquivo. Um valor NULL é retornado se o arquivo não puder ser aberto, como, por exemplo, quando um arquivo de dados existente não puder ser encontrado.

fclose(varpont);

Onde:

- **fclose**: a função de biblioteca usada para fechar um arquivo.

Tipo	Descrição
r	Abre um arquivo já existente somente para leitura
w	Abre um novo arquivo somente para gravação. Se um arquivo com conteúdo em *nome_arquivo* especificado já existir, ele será destruído e o novo arquivo será criado em seu lugar.
a	Abre um arquivo para alterações. Se o arquivo não existir é criado um novo.
rw	Abre um arquivo para leitura e gravação. Se o arquivo não existir é criado um novo.

Exemplo 1: Um programa C para abrir e fechar um arquivo

```
//abrearq.cpp
#include <stdio.h>

void main(){

  FILE *pontarq;                    /* define um ponteiro para a estrutura FILE */

  /* abre o arquivo e faz o ponteiro apontar p/ o buffer */
  pontarq = fopen("teste.doc", "r");
```

```
 if(pontarq == NULL)
   cout << "\nErro ao abrir o arquivo";

 else{
   cout << "\nO arquivo foi aberto com sucesso!";
   /* fecha o arquivo */
   fclose(pontarq);
 }
 cout("\nPressione qualquer teclar para sair! >>>";
 getchar();

} /* fecha o programa principal */
```

Exemplo 2: Um programa C para trabalhar com a linha de comando

```
//linhacom.cpp
#include <stdio.h>
#include <iostream.h>
#include <conio.h>

void main(int argc, char *argv[]){

 int i;
 clrscr();
 cout << "\n Trabalhando com numero de argumentos na linha de comando";

 /* Exibindo a linha de comando */
 for (i=0; i<argc; ++i)
   cout <<"\n argv[" << i << "] aponta para: " << argv[i];

 cout << "\n Pressione ENTER para sair! >>>";
 getch();

} /* fecha o programa principal */
```

Exemplo 3: Criando um arquivo utilizando a linha de comando.

```
//converte.cpp
/* programa para ler uma linha de texto em minúsculo e armazenar
   a linha equivalente em maiúsculo em um arquivo de dados */
#include <conio.h>
#include <stdio.h>
#include <ctype.h>

void main(int argc, char *argv[]){

 int i;
 char c;

 FILE *pontarq;  /* ponteiro do arquivo */

/* cria um novo arquivo de dados para gravacao apenas */
 if(( pontarq = fopen(argv[1], "w")) = = NULL)
   printf("Erro ao tentar criar o arquivo: %s \n", argv[1]);

 else{
```

```
   /* ler cada caractere e grava seu equivalente maiúsculo no arquivo */
  printf("\n ********** CONVERTE O TEXTO PARA MAIUSCULO**********\n");

   do
     putc(toupper(c = getchar()), pontarq);
   while(c != '\n');

   fclose(pontarq);

 }
 printf("\nPressione qualquer tecla para sair! >>>");
 getchar();

} /* fecha o programa principal */
```

Exemplo 4:

```
//lerarq.cpp
/* programa para exibir o conteúdo de um arquivo a
   partir da linha de comando */

#include <stdio.h>

void main(int argc, char *argv[]){
  int i;
  FILE *pontarq;  /* ponteiro do arquivo */

  char linha[255]; /* linha do arquivo */

  clrscr();
  if(( pontarq = fopen(argv[1], "r")) = = NULL)
    printf("Erro ao abrir o arquivo: %s \n", argv[1]);

  else{
    /* le e exibe cada linha do arquivo */
    while(fgets(linha, sizeof(linha), pontarq))
      fputs(linha, stdout);

    fclose(pontarq);  /* fechar o arquivo */
  }
  printf("\nPressione qualquer tecla para sair! >>>");
  getchar();

} /* fecha o programa principal */
```

Exemplo 5:

```
//pot2arq.cpp
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

int pot2(int num){
   return num*num;
}
```

```
void main(int argc, char *argv[]){
  int i, num;

  FILE *pontarq;  /* ponteiro do arquivo */

  char linha[255]; /* linha do arquivo */

  clrscr();
  if(( pontarq = fopen(argv[1], "r")) == NULL)
    printf("Erro ao abrir o arquivo: %s \n", argv[1]);

  else{
    /* le e exibe cada linha do arquivo */
    printf("\nElevando o numero ao quadrado .....\n");
    while(fgets(linha, sizeof(linha), pontarq)){
      //fputs(linha, stdout);
            num = atoi(linha);
            printf("%d -> %d\n", num, pot2(num));
    }
    fclose(pontarq);  /* fechar o arquivo */
  }
  printf("\nPressione qualquer tecla para sair! >>>");
  getchar();

} /* fecha o programa principal */
```