Manipulação de Arquivos

Estrutura de Dados II
Prof. Guilherme Tavares de Assis

Universidade Federal de Ouro Preto – UFOP
Instituto de Ciências Exatas e Biológicas – ICEB
Departamento de Computação – DECOM

Tipo Estruturado Arquivo

- Arquivos correspondem a unidades de armazenamento, tipicamente gravados em disco magnético.
- Sistemas operacionais, como *Linux* ou *Windows,* permitem que arquivos sejam criados e recuperados por um nome e pela posição em uma hierarquia de diretórios.
- Devem ser utilizados em programas quando:
  - não existe espaço em memória principal para o armazenamento de um grande volume de dados;
  - há necessidade do armazenamento de dados por um período de tempo indeterminado.

Tipo Estruturado Arquivo

- Um arquivo é interpretado pela linguagem C/C++ como qualquer dispositivo de entrada e saída (E/S), desde um arquivo em disco até uma impressora.
- Dentre os arquivos em disco, existem dois tipos:
  - Texto (caracteres) e Binário (*bytes*).
- Para utilizar um arquivo, é preciso associá-lo a uma variável lógica (*stream*) e, então, manipulá-la.
  - A associação ocorre na operação de abertura do arquivo.
- Operações básicas em um arquivo:
  - Leitura (consulta) de um dado.
  - Gravação (inclusão) de um dado.
  - Alteração ou exclusão de um dado.

Tipo Estruturado Arquivo

- Passos para a manipulação dos dados de um arquivo no próprio arquivo:
  1) Abrir ou criar o arquivo, associando o nome físico do arquivo ao seu nome lógico.
  2) Manipular os dados do arquivo: consulta, inclusão, exclusão, alteração.
  3) Fechar o arquivo.

Tipo Estruturado Arquivo

- Passos para a manipulação dos dados de um arquivo em memória principal:
  1) Abrir o arquivo, caso exista, associando o nome físico do arquivo ao seu nome lógico; senão ir para o passo 4.
  2) Ler os dados do arquivo, armazenando-os em memória principal.
  3) Fechar o arquivo.
  4) Manipular os dados em memória principal: consulta, inclusão, exclusão, alteração.
  5) Criar o arquivo, associando o nome físico do arquivo ao seu nome lógico.
  6) Gravar os dados da memória principal para o arquivo.
  7) Fechar o arquivo.

Ponteiro de Arquivo

- O ponteiro de arquivo serve para referenciar o arquivo a ser tratado pelo programa.
  - O ponteiro não aponta diretamente para o arquivo; contém as seguintes informações sobre o mesmo: nome, situação (aberto ou fechado) e posição atual sobre o arquivo.
- Para se definir uma variável ponteiro de arquivo, usa-se a seguinte declaração:

FILE *Arquivo;

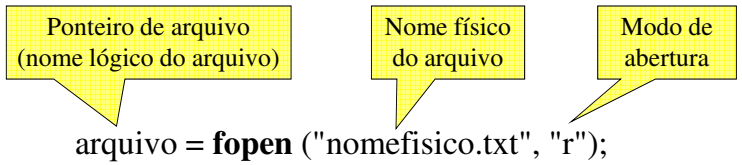
Desta forma, passa a existir uma variável de nome Arquivo que será o ponteiro de um arquivo a ser criado ou aberto.

Funções Comuns no Sistema de E/S ANSI

Função	Ação
fopen()	Abre um arquivo
fclose()	Fecha um arquivo
feof()	Verifica o final de um arquivo
putc() e fputc()	Escreve um caractere em um arquivo texto
getc() e fgetc()	Lê um caractere de um arquivo texto
fprintf()	Permite impressão formatada em um arquivo texto
fscanf()	Permite leitura formatada de um arquivo texto
fseek()	Posiciona em um item (registro) de um arquivo binário
fwrite()	Escreve tipos maiores que 1 *byte* em um arquivo binário
fread()	Lê tipos maiores que 1 *byte* de um arquivo binário

Em C/C++, as funções para manipulação de arquivos encontram-se na biblioteca *stdio.h* ou *stdlib.h*.

Abertura de Arquivo



- A função **fopen**() abre um arquivo. Para tanto, devem ser passados o nome físico do arquivo e o modo de abertura. Caso o arquivo possa ser aberto, retorna um ponteiro referente; caso contrário, retorna NULL (nulo).

Abertura de Arquivo

Modo	Ação
r	Abre um arquivo texto existente para leitura
w	Cria um arquivo texto para escrita
a	Abre um arquivo texto para inserção no final
r+	Abre um arquivo texto existente para leitura e escrita
w+	Cria um arquivo texto para leitura e escrita
a+	Abre um arquivo texto para leitura e inserção no final
rb	Abre um arquivo binário existente para leitura
wb	Cria um arquivo binário para escrita
ab	Abre um arquivo binário para inserção no final
r+b	Abre um arquivo binário existente para leitura e escrita
w+b	Cria um arquivo binário para leitura e escrita
a+b	Abre um arquivo binário para leitura e inserção no final

Abertura de Arquivo

```
// Verificação de abertura de arquivo

if ((arquivo = fopen("teste.txt","r")) == NULL)
{
      puts ("Arquivo nao pode ser aberto...");
      exit (1);
}
```


Fechamento de Arquivo

Ponteiro de arquivo (nome lógico do arquivo)

fclose (arquivo);

- O comando **fclose**() fecha um arquivo em nível de sistema operacional. Para tanto, deve ser passado o nome lógico do arquivo a ser fechado.
- Terminar um programa, sem fechar um arquivo aberto, pode provocar perda de dados no arquivo ou corrompê-lo.
- Como, normalmente, há limite do sistema operacional para o número de arquivos abertos ao mesmo tempo, pode ser necessário fechar um arquivo antes de abrir outro.

Leitura em Arquivo Texto

Caractere lido do arquivo

Ponteiro de arquivo (nome lógico do arquivo)

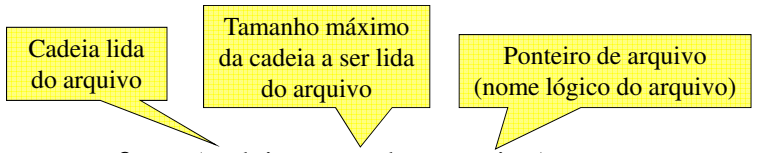
caractere = **getc** (arquivo);

- A função **getc**() ou **fgetc**() lê um caractere de um arquivo texto, retornando-o. Para tanto, deve ser passado o nome lógico do arquivo aberto.
  - Se o ponteiro do arquivo estiver no final do mesmo ou ocorrer um erro na leitura, a função retorna EOF.
- Existem duas funções para preservar a compatibilidade com versões mais antigas de C/C++.

Leitura em Arquivo Texto

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main ( ) {
   FILE *arq; //declaração do arquivo
   char ch;
   if ((arq = fopen("teste.txt","r")) == NULL) {
      cout << "Erro na abertura do arquivo\n"; return 0;
   }
   ch = getc(arq);   //lê o 1o caractere do arquivo
   while (ch != EOF) {  //varre o arquivo
      putchar(ch);   //imprime na tela o caractere lido
      ch = getc(arq);   //lê o próximo caractere
   }
   fclose (arq); //fecha o arquivo
   return 0;
}
```


Leitura em Arquivo Texto



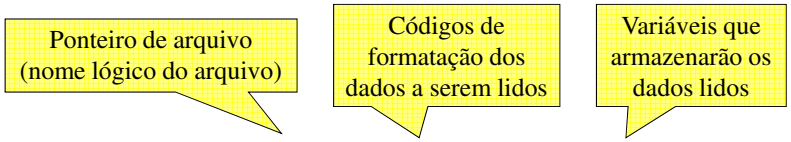
fgets (cadeia, tamanho, arquivo);

- A função **fgets()** lê uma cadeia de um arquivo texto. Para tanto, devem ser passados a cadeia a ser lida (dado de saída), o tamanho máximo da cadeia e o nome lógico do arquivo aberto.
  - A função lê a cadeia até que um caractere de nova linha seja alcançado ou (tamanho - 1) caracteres tenham sido lidos.
  - A função inclui os caracteres "\n" e NULL ao final da cadeia.
  - Se a leitura ocorrer devidamente, a função retorna um ponteiro para a cadeia lida; caso contrário, retorna nulo.

Leitura em Arquivo Texto

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
using namespace std;
main (int argc, char *argv[]) {
   FILE *arq;
   char cadeia[81];
   if (argc != 2)  {
      cout << "Formato: executável arquivo\n"; exit(1);
   }
   if ((arq = fopen(argv[1],"r")) == NULL) {
      cout << "Erro na abertura do arquivo\n"; exit(1);
   }
   while (fgets(cadeia, 80, arq) != NULL)  {
      cout << cadeia;
   }
   fclose(arq);
}
```


Leitura em Arquivo Texto



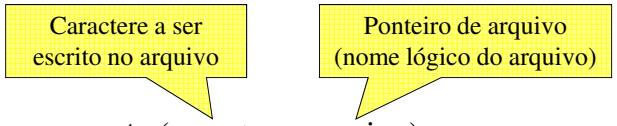
fscanf (arquivo, "formatação", variáveis);

- A função **fscanf()** lê dados formatados de um arquivo texto. Para tanto, devem ser passados o nome lógico do arquivo aberto, os códigos de formatação referentes aos dados a serem lidos e as variáveis que receberão os dados lidos.
  - Se a leitura ocorrer devidamente, a função retorna a quantidade de dados lidos com sucesso; caso contrário, retorna 0.
  - Se a função tenta ler o fim de arquivo, retorna EOF.

Leitura em Arquivo Texto

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main ( ) {
   FILE *arq;
   char enter, titulo[31];
   int codlivro;
   float preco;
   if ((arq = fopen("teste.txt","r")) == NULL) {
      cout << "Erro na abertura do arquivo\n"; return 0;
   }
   while (fscanf(arq,"%[A-Z a-z] %d %f %[\n]",
      titulo, &codlivro, &preco, enter) != EOF)  {
      printf("%s %3d %.2f\n", titulo, codlivro, preco);
   }
   fclose (arq);
   return 0;
}
```


Escrita em Arquivo Texto



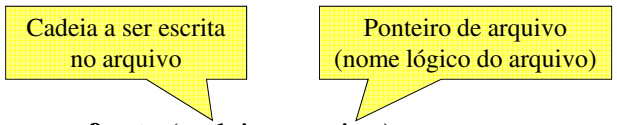
putc (caractere, arquivo);

- A função **putc()** ou **fputc()** escreve um caractere em um arquivo texto. Para tanto, devem ser passados o caractere a ser escrito e o nome lógico do arquivo aberto.
  - Se a escrita ocorrer devidamente, a função retorna o caractere escrito; caso contrário, retorna EOF.
- Existem duas funções para preservar a compatibilidade com versões mais antigas de C/C++.

Escrita em Arquivo Texto

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main ( ) {
   FILE *arqE, *arqS;    char ch;
   if ((arqE = fopen("entrada.txt","r")) == NULL) {
      cout << "Erro na abertura do arquivo\n"; return 0;
   }
   if ((arqS = fopen("saida.txt","w")) == NULL) {
      cout << "Erro na criação do arquivo\n"; return 0;
   }
   ch = getc(arqE);
   while (ch != EOF) {
      putc(ch,arqS);    ch = getc(arqE);
   }
   fclose (arqE);    fclose (arqS);
   return 0;
}
```


Escrita em Arquivo Texto



fputs (cadeia, arquivo);

- A função **fputs()** escreve uma cadeia em um arquivo texto. Para tanto, devem ser passados a cadeia a ser escrita e o nome lógico do arquivo aberto.
  - Se a escrita ocorrer devidamente, a função retorna um valor não negativo; caso contrário, retorna EOF.

Escrita em Arquivo Texto

```
#include <iostream>   #include <stdio.h>
#include <stdlib.h>   #include <string.h>
using namespace std;
main (int argc, char *argv[]) {
   FILE *arq;
   char cadeia[81];
   if (argc != 2)  {
      cout << "Formato: executável arquivo\n"; exit(1);
   }
   if ((arq = fopen(argv[1],"w")) == NULL) {
      cout << "Erro na criação do arquivo\n"; exit(1);
   }
   while (strlen(gets(cadeia)) > 0) {
      fputs (cadeia, arq);
      fputs ("\n", arq);
   }
   fclose(arq);
}
```


Escrita em Arquivo Texto

Ponteiro de arquivo (nome lógico do arquivo)

Códigos de formatação dos dados a serem escritos

Dados a serem escritos no arquivo

fprintf (arquivo, "formatação", variáveis);

- A função **fprintf()** escreve dados formatados em um arquivo texto. Para tanto, devem ser passados o nome lógico do arquivo aberto, os códigos de formatação e as variáveis referentes aos dados a serem escritos no arquivo.
  - Se a escrita ocorrer devidamente, a função retorna a quantidade de *bytes* escritos com sucesso no arquivo; caso contrário, retorna 0.

Escrita em Arquivo Texto

```
#include <iostream>   #include <stdio.h>
#include <string.h>   using namespace std;
int main ( ) {
   FILE *arq;
   char titulo[31];  int codlivro;  float preco;
   if ((arq = fopen("teste.txt","w")) == NULL) {
      cout << "Erro na criação do arquivo\n"; return 0;
   }
   do  {
      cout << "Título:";   cin.getline(titulo, 30);
      if (strlen(titulo) > 1) {
         cout << "Código:";   cin >> codlivro;
         cout << "Preco:";    cin >> preco;
         fprintf (arq, "%-30s%-4d %.2f\n",
                        titulo, codlivro, preco);
      }
   }  while (strlen(titulo) > 1); fclose(arq); return 0;
}
```


Final de Arquivo

Ponteiro de arquivo (nome lógico do arquivo)

feof (arquivo);

- A função lógica **feof()** serve para indicar que o final de um arquivo binário (preferencialmente) ou texto foi encontrado. Para tanto, deve ser passado o nome lógico do arquivo aberto.
  - Se o fim de arquivo não tiver sido atingido, a função retorna 0; caso contrário, retorna um valor diferente de 0.
  - É geralmente usada em leitura de arquivos binários, pois um valor inteiro pode ser lido, erroneamente, como sendo o EOF e não como parte do arquivo, finalizando a leitura.

Final de Arquivo

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main ( ) {
   FILE *arq;
   char ch;
   int cont = 0;
   if ((arq = fopen("teste.txt","r")) == NULL) {
      cout << "Erro na abertura do arquivo\n"; return 0;
   }
   while (!feof(arq)) {
      ch = getc(arq);
      cont++;
   }
   fclose (arq);
   cout << "Quantidade: " << cont << endl;
   return 0;
}
```


Leitura em Arquivo Binário

Endereço da variável que armazenará os itens lidos do arquivo | Número de *bytes* do item lido; usa-se o *sizeof* | Quantidade de itens a serem lidos do arquivo | Ponteiro de arquivo (nome lógico do arquivo)

fread (variável, tamanho, quantidade, arquivo);

- A função **fread()** lê itens de um arquivo binário. Para tanto, devem ser passados a variável que receberá os itens lidos, o tamanho em *bytes* do item a ser lido, a quantidade de itens a serem lidos (cada item do tamanho especificado) e o nome lógico do arquivo aberto.
  - Se a leitura ocorrer devidamente, a função retorna o número de itens lidos que, normalmente, é igual ao terceiro argumento.
    - Se for encontrado o fim de arquivo, o número retornado é menor.

Leitura em Arquivo Binário

```
#include <iostream>   #include <stdio.h>
using namespace std;
typedef struct {
   char titulo[30];  int codlivro;  float preco;
} reglivro;
int main ( ) {
   FILE *arq;   reglivro livro;
   if ((arq = fopen("livros.bin","rb")) == NULL) {
      cout << "Erro na abertura do arquivo\n"; return 0;
   }
   while (fread(&livro, sizeof(livro), 1, arq) == 1) {
      cout << "Título: " << livro.titulo << endl;
      cout << "Código: " << livro.codlivro << endl;
      cout << "Preço: " << livro.preco << endl;
   }
   fclose (arq);
   return 0;
}
```


Escrita em Arquivo Binário

Endereço da variável com os itens a serem escritos no arquivo | Número de *bytes* do item escrito; usa-se o *sizeof* | Quantidade de itens a serem escritos | Ponteiro de arquivo (nome lógico do arquivo)

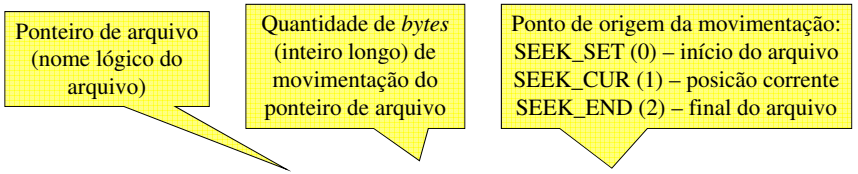
fwrite (variável, tamanho, quantidade, arquivo);

- A função **fwrite()** escreve itens em um arquivo binário. Para tanto, devem ser passados a variável que contém os itens a serem escritos, o tamanho em *bytes* do item a ser escrito, a quantidade de itens a serem escritos (cada item do tamanho especificado) e o nome lógico do arquivo aberto.
  - Se a escrita ocorrer devidamente, a função retorna o número de itens escritos, ou seja, o tamanho especificado.
    - Se ocorrer um erro, o número retornado é diferente do tamanho.

Escrita em Arquivo Binário

```
#include <iostream>   #include <stdio.h>
using namespace std;
typedef struct {
   char titulo[30];  int codlivro;  float preco;
} reglivro;
int main ( ) {
   FILE *arq;   reglivro livro;   char opcao;
   if ((arq = fopen("livros.bin","wb")) == NULL) {
      cout << "Erro na criação do arquivo\n"; return 0;
   }
   do  {
      cout << "Título:";  cin.getline(livro.titulo, 30);
      cout << "Código:";  cin >> livro.codlivro;
      cout << "Preco:";   cin >> livro.preco;
      fwrite (&livro, sizeof(livro), 1, arq);
      cout << "Adiciona outro livro (S/N)? ";
      cin >> opcao;
   }  while (toupper(opcao) == 'S');
   fclose (arq);   return 0;
}
```


Movimentação do Ponteiro de Arquivo



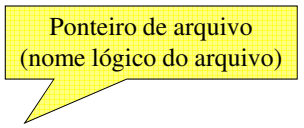
fseek (arquivo, deslocamento, origem);

- A função **fseek()** altera o endereço do ponteiro de um arquivo binário. Para tanto, devem ser passados o nome lógico do arquivo aberto, o deslocamento do ponteiro em termos de *bytes* e o ponto de origem do deslocamento.
  - Se a movimentação ocorrer devidamente, a função retorna o valor 0; caso contrário, retorna um valor diferente de 0.

Movimentação do Ponteiro de Arquivo

```
#include <iostream>   #include <stdio.h>   #include <stdlib.h>
using namespace std;
typedef struct {
   char titulo[30];  int codlivro;  float preco;
} reglivro;
main (int argc, char *argv[]) {
   FILE *arq;   reglivro livro;   long desloc;
   if (argc != 3)  {
      cout << "Formato: executável arquivo registro\n"; exit(1);
   }
   if ((arq = fopen(argv[1],"rb")) == NULL) {
      cout << "Erro na abertura do arquivo\n"; exit(1);
   }
   desloc = atoi(argv[2]) * sizeof(livro);
   if (fseek(arq, desloc, 0) != 0) {
      cout << "Movimentação não realizada";  exit(1);
   }
   if (fread(&livro, sizeof(livro), 1, arq) != 1) {
      cout << "Leitura não realizada";  exit(1);
   }
   cout << livro.titulo << livro.codlivro << livro.preco << endl;
   fclose (arq);
}
```


Movimentação do Ponteiro de Arquivo



rewind (arquivo);

- O comando **rewind()** faz com que o ponteiro de um arquivo aponte para o início do mesmo. Para tanto, deve ser passado o nome lógico do arquivo aberto.

Movimentação do Ponteiro de Arquivo

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
using namespace std;

typedef struct {
   char nome[30];    long fone;
} Registro;

int main ( ) {
   Registro cliente;
   FILE *arq;    char auxiliar[9];

   if ((arq = fopen("clientes.bin","w+b")) == NULL) {
      cout << "Erro na abertura do arquivo\n"; return 0;
   }

// continua ...
```


Movimentação do Ponteiro de Arquivo

```
// continuando programa anterior ...

   do {
      cout << "Nome do cliente: ";
      cin.getline (cliente.nome, 30);
      if (strcmp (cliente.nome,"") != 0){
         cout << "Telefone: ";  cin.getline (auxiliar, 9);
         cliente.fone = atol(auxiliar);
         fwrite (&cliente, sizeof(cliente), 1, arq);
      }
   } while (strcmp(cliente.nome,"") != 0);

   rewind (arq);
   printf ("%-20s  %10s\n", "NOME", "TELEFONE");
   while (fread (&cliente, sizeof(Registro), 1, arq) == 1) {
      printf("%-20s %10ld\n", cliente.nome, cliente.fone);
   }
   fclose (arq);
   return 0;
}
```


Posição Corrente do Ponteiro de Arquivo

Posição do ponteiro do arquivo em *bytes* (inteiro longo) em relação ao seu início

Ponteiro de arquivo (nome lógico do arquivo)

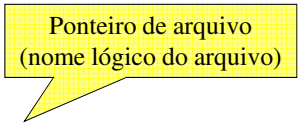
posicao = **ftell** (arquivo);

- A função **ftell()** retorna a posição corrente do ponteiro de um arquivo binário, ou seja, o número de *bytes* desde o início do arquivo. Para tanto, deve ser passado o nome lógico do arquivo aberto.

Posição Corrente do Ponteiro de Arquivo

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main (int argc, char *argv[]) {
   FILE *arq;
   long posicao;
   if (argc != 2)  {
      cout << "Formato: executável arquivo\n"; return 0;
   }
   if ((arq = fopen(argv[1],"rb")) == NULL) {
      cout << "Erro na abertura do arquivo\n"; return 0;
   }
   fseek (arq, 0, SEEK_END);
   posicao = ftell (arq);
   fclose (arq);
   cout << "Tamanho do arquivo: " << posicao << endl;
   return 0;
}
```

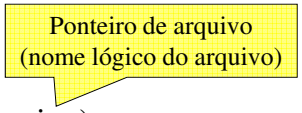
Verificação de Erro



ferror (arquivo);

- A função **ferror()** retorna verdadeiro se ocorreu um erro durante a última operação com o arquivo; caso contrário, retorna falso. Para tanto, deve ser passado o nome lógico do arquivo aberto.
  - Como cada "operação de arquivo" modifica a condição de erro, a função deve ser chamada logo após a operação a ser avaliada.

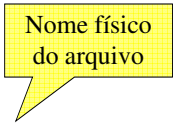
Limpeza de *Buffer*



fflush (arquivo);

- A função **fflush()** escreve o conteúdo do *buffer* em um arquivo, esvaziando-o. Para tanto, deve ser passado o nome lógico do arquivo aberto para escrita.
  - Se a limpeza do *buffer* ocorrer devidamente, a função retorna o valor 0; caso contrário, retorna EOF.
- Pode ser usada para limpar *buffer* da entrada padrão – **fflush (stdin)** - ou liberar *buffer* de escrita para a saída padrão - **fflush (stdout)** .

Remoção de Arquivo



remove (nome_arquivo);

- A função **remove()** remove um arquivo. Para tanto, deve ser passado o nome físico do arquivo fechado.
  - Se a remoção do arquivo ocorrer devidamente, a função retorna o valor 0; caso contrário, retorna um valor diferente de zero.

Remoção de Arquivo

```
#include <iostream>   #include <stdio.h>
#include <stdlib.h>
using namespace std;
main (int argc, char *argv[]) {
   char opcao;
   if (argc != 2)  {
      printf ("Formato: executável arquivo\n"); exit(1);
   }
   printf("Deseja apagar o arquivo %s (S/N)?", argv[1]);
   fflush (stdout); // liberação do buffer, quando preciso
   opcao = getchar();
   if (toupper(opcao) == 'S')
      if (remove(argv[1])) {
          printf ("Erro ao tentar apagar arquivo.\n");
          exit(1);
      }
      else   printf ("Arquivo apagado com sucesso.\n");
}
```

Renomeação de Arquivo

Nome físico do arquivo a ser renomeado

Novo nome físico do arquivo

rename (nome_atual, nome_novo);

- A função **rename()** renomeia um arquivo. Para tanto, devem ser passados o nome físico atual do arquivo fechado e o novo nome físico do mesmo.
  - Se a renomeação do arquivo ocorrer devidamente, a função retorna o valor 0; caso contrário, retorna um valor diferente de zero.
- A função também serve para mover um arquivo.
  - É preciso incluir a nova localização como parte do novo nome do arquivo.

Renomeação de Arquivo

```
#include <stdio.h>   #include <stdlib.h>
using namespace std;
main ( ) {
   char atual[30], novo[30];
   printf ("Nome atual do arquivo original: ");
   fflush (stdout); // liberação do buffer, quando preciso
   gets (atual);
   printf ("Novo nome para o arquivo: ");
   fflush (stdout); // liberação do buffer, quando preciso
   gets (novo);

   if (rename(atual, novo)) {
      printf("Erro ao renomear arquivo!\n");
      exit (1);
   }
   else
      printf ("Arquivo renomeado com sucesso.\n");
}
```