Manipulação de arquivos em linguagem C

Docente: Gil Jader – gil.jader@gmail.com

Agenda

1 - CONCEITO

2 - Vantagens de utilizar arquivos

3 - Tipos de arquivos

4 - Observações

5 – Passos para a manipulação dos dados de um arquivo no próprio arquivo

6 – Ponteiro para arquivo

7 – Operações com Arquivos do tipo texto

8 – Tipos de abertura de arquivos

9 – Problemas na abertura de arquivos

10 – Gravando dados em arquivo Texto

11 – Leitura de arquivo Texto

Docente: Gil Jader – gil.jader@gmail.com

1. CONCEITO

Os arquivos permitem gravar os dados de um programa de forma permanente em mídia digital.

2. Vantagens de utilizar arquivos

- **Armazenamento permanente de dados:** as informações permanecem disponíveis mesmo que o programa que as gravou tenha sido encerrado, ou seja, podem ser consultadas a qualquer momento.

- **Grande quantidade dados pode ser armazenada:** A quantidade de dados que pode ser armazenada depende apenas da capacidade disponível da mídia de armazenamento. Normalmente a capacidade da mídia é muito maior do que a capacidade disponível na memória RAM.

- **Acesso concorrente:** Vários programas podem acessar um arquivo de forma concorrente.

3. Tipos de arquivos

Em Linguagem C trabalhamos com dois tipos de arquivos:

1) Arquivo texto: Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de texto.

> **Exemplos de arquivos texto:** documentos de texto, código fonte C, páginas XHTML.

2) Arquivo binário é uma sequência de bits que obedece regras do programa que o gerou.

> **Exemplos**: Executáveis, documentos do Word, arquivos compactados.

4. Observações:

- 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

5. Passos para a manipulação dos dados de um arquivo texto:

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.

6. Ponteiro para arquivo

- O ponteiro de arquivo serve para referenciar o arquivo a ser tratado pelo programa.
    - O ponteiro 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 < *ponteiro >

O tipo FILE está definido na biblioteca stdio.h.

- Exemplo de declaração de um ponteiro para arquivo em C:

FILE *_pont_arq_;

*Lembrando que **FILE** deve ser escrito em letras maiúsculas.*

7. Operações com Arquivos do tipo texto

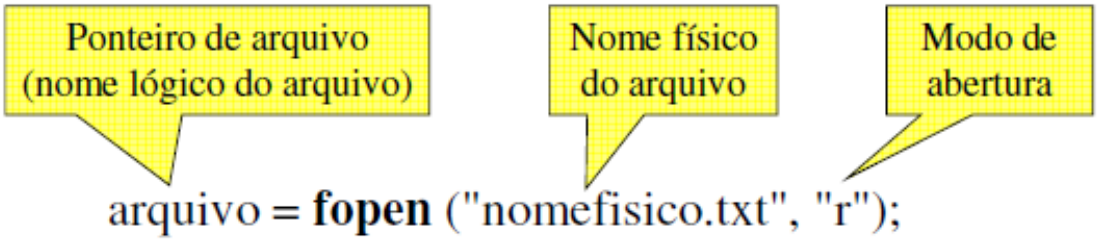
- **Abertura de arquivos**

Para trabalhar com um arquivo, a primeira operação necessária é abrir este arquivo.

- Sintaxe de abertura de arquivo:

< ponteiro > = fopen(“nome do arquivo”,”tipo de abertura”);

- Exemplo:

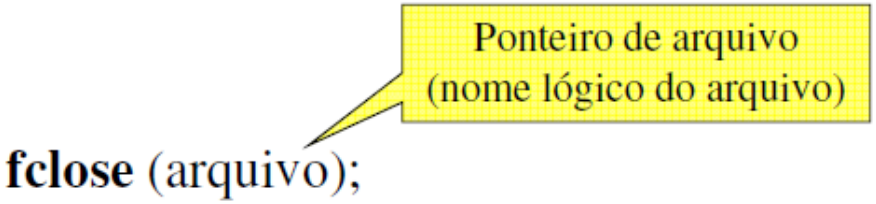


*A função **fopen**() abre um arquivo. Para tanto, devem ser passados o nome físico do arquivo e o modo de abertura. Caso o arquivo possa ser aberto, retorna um ponteiro referente; caso contrário, retorna NULL (nulo).*

7. Operações com Arquivos do tipo texto

- **Fechamento de arquivos**
  - A função **fopen** recebe como parâmetros o nome do arquivo a ser aberto e o tipo de abertura a ser realizado.
  - Depois de aberto, realizamos as operações necessárias e fechamos o arquivo.
  - Para fechar o arquivo usamos a função **fclose**.

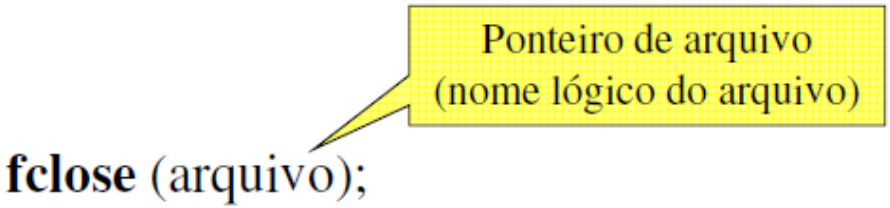
Exemplo:



***OBS**.: Lembrando que o ponteiro deve ser a mesma variável ponteiro associada ao comando de abertura de arquivo.*

7. Operações com Arquivos do tipo texto

- **Fechamento de arquivos**



- 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.
- É recomendável fechar um arquivo antes de abrir outro.

7. Operações com Arquivos do tipo texto

- **Abertura e Fechamento de arquivos**

Funções Comuns no Sistema de E/S do ANSI C

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

8. Tipos de abertura de arquivos

- **r:** Permissão de abertura somente para leitura. É necessário que o arquivo já esteja presente no disco.

- **w:** Permissão de abertura para escrita (gravação). Este código cria o arquivo caso ele não exista, e caso o mesmo exista ele recria o arquivo novamente fazendo com que o conteúdo seja perdido. Portanto devemos tomar muito cuidado ao usar esse tipo de abertura.

- **a:** Permissão para abrir um arquivo texto para escrita(gravação), permite acrescentar novos dados ao final do arquivo. Caso não exista, ele será criado.

8. Tipos de abertura de arquivos

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

8. Tipos de abertura de arquivos

Exemplo de criação de arquivos.

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

int main(void)
{
  // criando a variável ponteiro para o arquivo
  FILE *pont_arq;

  //abrindo o arquivo
  pont_arq = fopen("arquivo.txt", "a");

  // fechando arquivo
  fclose(pont_arq);

  //mensagem para o usuário
  printf("O arquivo foi  criado com sucesso!");

  system("pause");
  return(0);
}
```

9. Problemas na abertura de arquivos

- Na prática, nem sempre é possível abrir um arquivo. Podem ocorrer algumas situações que impedem essa abertura, por exemplo:
  - Você está tentando abrir um arquivo no modo de leitura, mas o arquivo não existe;
  - Você não tem permissão para ler ou gravar no arquivo;
  - O arquivo está bloqueado por estar sendo usado por outro programa.

*Lembrando que quando o arquivo não pode ser aberto a função **fopen** retorna o valor NULL.*

9. Problemas na abertura de arquivos

É recomendável criar um trecho de código a fim de verificar se a abertura ocorreu com sucesso ou não.

Exemplo:

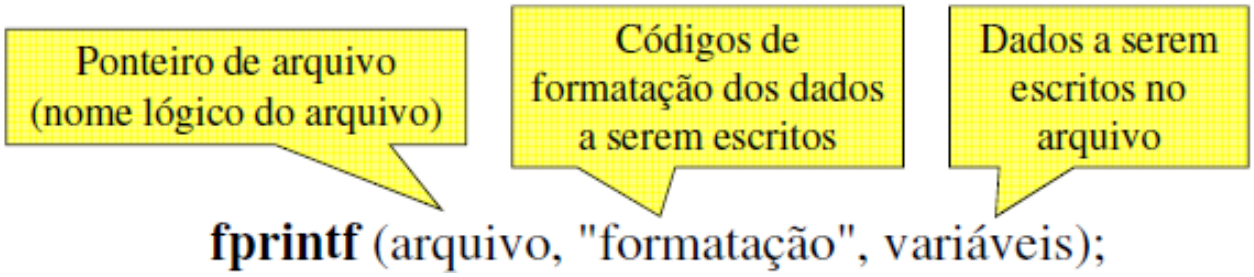
```
if (pont_arq == NULL)
{
    printf("ERRO! O arquivo não foi aberto!\n");
}
else
   {
     printf("O arquivo foi aberto com sucesso!");
   }
```

10. Gravando dados em arquivo Texto

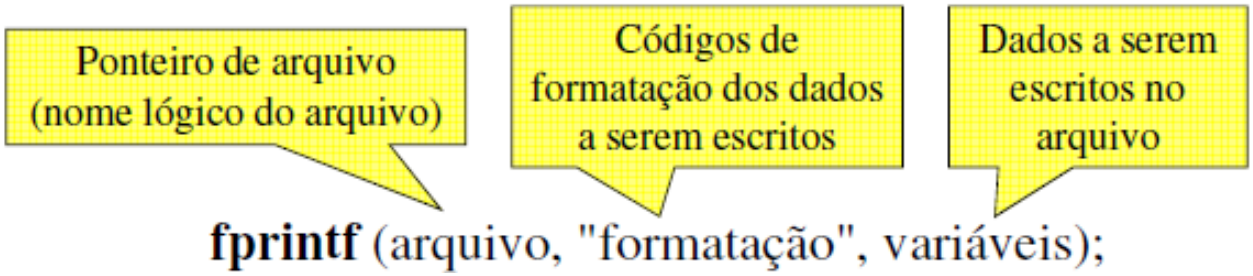
A função **fprintf** armazena dados em um arquivo. Seu funcionamento é muito semelhante ao **printf**, a diferença principal é a existência de um parâmetro para informar o arquivo onde os dados serão armazenados.

Sintaxe:

fprintf(nome_do_ponteiro_para_o_arquivo, “%s”,variavel_string)



10. Gravando dados em arquivo Texto



A função **fprintf()** escreve dados formatados em um arquivo texto. Para tanto, devem ser passados o **nome lógico** do arquivo aberto, os **códigos de formataçã**o e as **variáveis** referentes aos dados a serem escritos no arquivo.

- A escrita ocorrer devidamente, a função retorna a quantidade de ***Bytes*** escritos com sucesso no arquivo; caso contrário, retorna 0.

10. Gravando dados em arquivo Texto

Funções em *C* para a operação de gravação dados em arquivos:

- **putc()** ou **fputc()**: Grava um único caractere no arquivo;
- **fprintf()**: Grava dados formatados no arquivo, de acordo com o tipo de dados (float, int, ...). Similar ao printf, porém ao invés de imprimir na tela, grava em arquivo
- **fwrite()**: Grava um conjunto de dados heterogêneos (struct) no arquivo

10. Gravando dados em arquivo Texto

Sintaxe das funções para gravação:

- Grava o conteúdo da variável caractere no arquivo

  putc (caracter, arquivo);

- Grava dados formatados no arquivo, de acordo com o tipo de dados (float, int, ...)

  fprintf(arquivo, "formatos", var1, var2 ...);

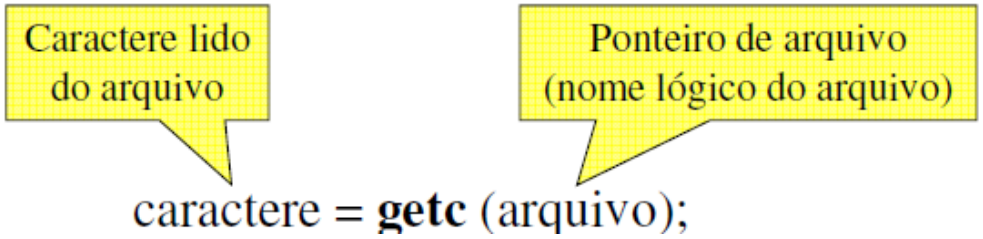
- Grava um conjunto de dados heterogêneos (struct) no arquivo

  fwrite (buffer, tamanhoembytes, quantidade, ponteirodearquivo);

Linguagem de Programação Estruturada – 2016.1

```c
//Exemplo: Abrindo, gravando e fechando arquivo
#include <stdio.h>
#include <conio.h>

int main(void)
{
  FILE *pont_arq; // cria variável ponteiro para o arquivo
  char palavra[20]; // variável do tipo string

  //abrindo o arquivo com tipo de abertura w
  pont_arq = fopen("arquivo_palavra.txt", "w");

  //testando se o arquivo foi realmente criado
  if(pont_arq == NULL)
  {
     printf("Erro na abertura do arquivo!");
     return 1;
  }

  printf("Escreva uma palavra para testar gravacao de arquivo: ");
  scanf("%s", palavra);

  //usando fprintf para armazenar a string no arquivo
  fprintf(pont_arq, "%s", palavra);

  //usando fclose para fechar o arquivo
  fclose(pont_arq);

  printf("Dados gravados com sucesso!");

  getch();
  return(0);
}
```

11. Leitura de arquivo Texto



- 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++.

11. Leitura de arquivo Texto

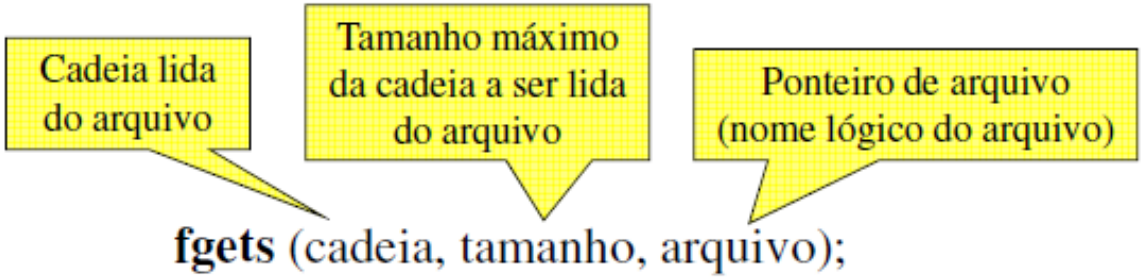
Então... Para realizar a leitura de um arquivo inteiro **caracter** por **caracter** podemos usar **getc** dentro de um laço de repetição.

Exemplo:

```
do
{
    //faz a leitura do caracter no arquivo apontado por pont_arq
    c = getc(pont_arq);

    //exibe o caracter lido na tela
    printf("%c" , c);

}while (c != EOF);
```

11. Leitura de arquivo Texto

Leitura de Strings



- 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.

11. Leitura de arquivo Texto

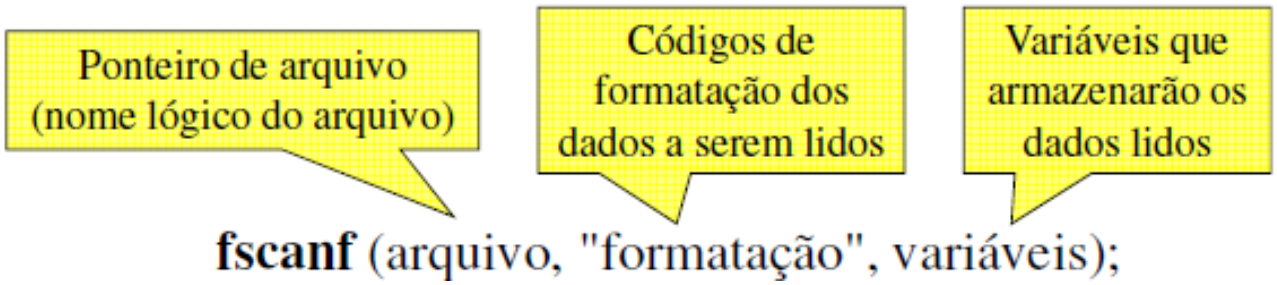
Exemplo:

```c
//Leitura de arquivo
#include <stdio.h>
#include <conio.h>

int main(void)
{
  FILE *pont_arq;
  char texto_str[20];

  //abrindo o arquivo_frase em modo "somente leitura"
  pont_arq = fopen("arquivo_palavra.txt", "r");

  //enquanto não for fim de arquivo o looping será executado
  //e será impresso o texto
  while(fgets(texto_str, 20, pont_arq) != NULL)
    printf("%s", texto_str);

  //fechando o arquivo
  fclose(pont_arq);

  getch();
  return(0);
}
```

11. Leitura de arquivo Texto

fscanf(): retorna a quantidade variáveis lidas com sucesso

Sintaxe:



- 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.

11. Leitura de arquivo Texto

fscanf(): retorna a quantidade variáveis lidas com sucesso

- Isso é interessante se tivermos um arquivo com um determinado formato.

Exemplo

Suponha que tenhamos um arquivo de texto chamado "arquivo.txt" com o seguinte conteúdo:

a b c
d e f
g h i
j k l

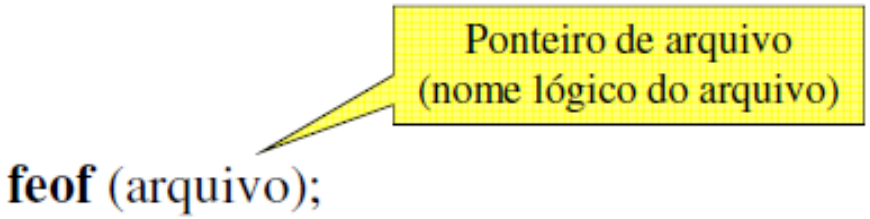
Como usar a fscanf para lê-lo?

Basta notar que o formato desse arquivo é: caractere, espaço, caractere, espaço, caractere e enter Ou seja: "%c %c %c\n"

11. Leitura de arquivo Texto

```
#include <stdio.h>

int main(void)
{
        char url[]="arquivo.txt",
             ch1, ch2, ch3;
        FILE *arq;

        arq = fopen(url, "r");
        if(arq == NULL)
                        printf("Erro, nao foi possivel abrir o arquivo\n");
        else
                while( (fscanf(arq,"%c %c %c\n", &ch1, &ch2, &ch3))!=EOF )
                        printf("%c %c %c\n", ch1, ch2, ch3);

        fclose(arq);

        return 0;
}
```

12. Final de 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.

12. Final de Arquivo

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

int main () {

    FILE *arq;
    char ch;
    int cont = 0;

    if ((arq = fopen("teste.txt","r")) == NULL) {
       printf("Erro na abertura do arquivo\n");
       getch();
    }

    while (!feof(arq)) {
          ch = getc(arq);
          cont++;
    }

    fclose (arq);
    printf ("Quantidade: %d", cont);

    getch();
}
```

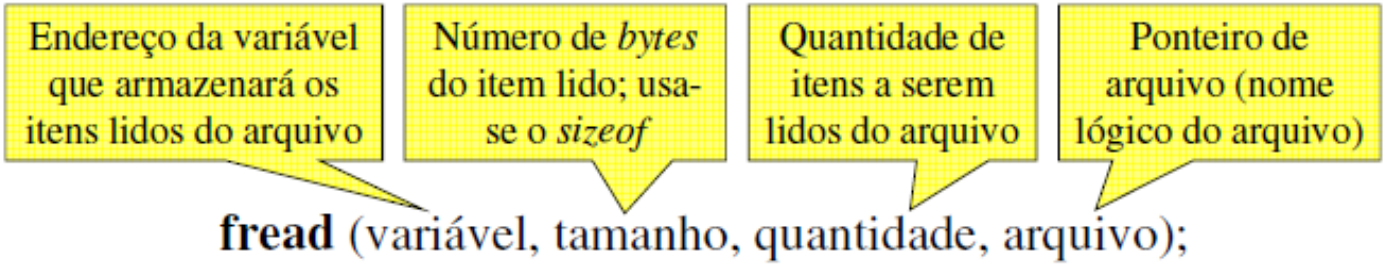
13. Lendo e Gravando Estruturas

Além da manipulação de arquivos do tipo texto, pode-se ler e escrever estruturas maiores, usando as funções **fread()** e **fwrite()**, conforme as sintaxes a seguir:

fread (**buffer**, tamanhoembytes, quantidade, ponteirodearquivo)

fwrite(**buffer**, tamanhoembytes, quantidade, ponteirodearquivo)

14. Leitura em Arquivo Binário



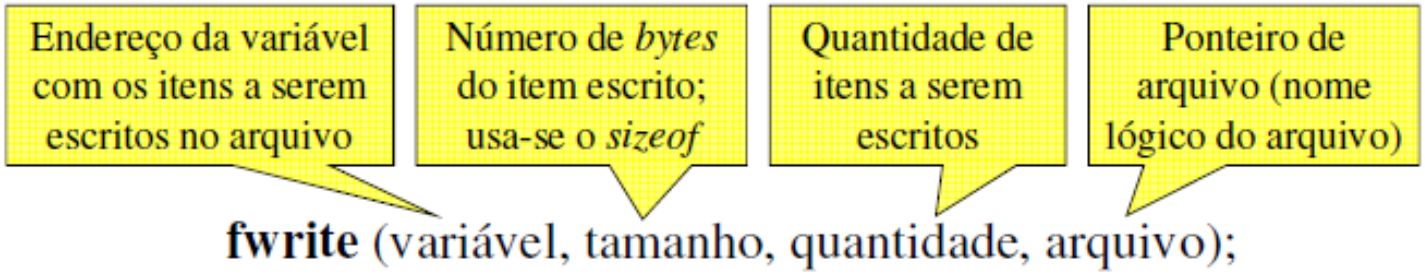
- A função **fread()** lê itens de um arquivo binário. Para tanto, devem ser passados a variável que receberá os itens lidos, o tamanho em *bytes* do item a ser lido, a quantidade de itens a serem lidos (cada item do tamanho especificado) e o nome lógico do arquivo aberto.

- 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.

14. Leitura em Arquivo Binário

```
#include <stdio.h>
#include<conio.h>
#include <stdio.h>

struct acervo{
       char titulo[30];
       int codlivro;
       float preco;
};

int main(){
  FILE *arq;
  struct acervo livro;

  if ((arq = fopen("livros.bin","rb")) == NULL) {
     printf ("Erro na abertura do arquivo\n");
     getch();
  }
  while (fread(&livro, sizeof(livro), 1, arq) == 1) {
        printf ("Título: %s", livro.titulo);
        printf ("Codigo: %d", livro.codlivro);
        printf ("Preco: %f",  livro.preco);
  }
  fclose (arq);
  getch();
}
```

15. Escrita em Arquivo Binário



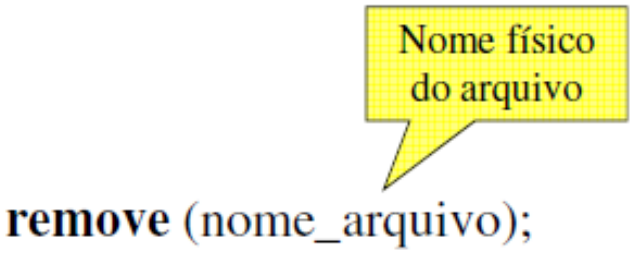
A função **fwrite()** escreve itens em um arquivo binário. Para tanto, devem ser passados a variável que contém os itens a serem escritos, o tamanho em bytes do item a ser escrito, a quantidade de itens a serem escritos (cada item do tamanho especificado) e o nome lógico do arquivo aberto.

- Se a escrita ocorrer devidamente, a função retorna o número de itens escritos, ou seja, o tamanho especificado.

15. Escrita em Arquivo Binário

```
#include <stdio.h>
#include<conio.h>
#include <ctype.h>
struct acervo{
       char titulo[30];
       int codlivro;
       float preco;
};
int main(){
    FILE *arq;
    struct acervo livro;
    char opcao;

    if ((arq = fopen("livros.bin","wb")) == NULL) {
       printf ("Erro na criação do arquivo\n");
       return 0;
    }
do {
   printf("Titulo: ");
   scanf ("%s", livro.titulo);
   printf("Codigo: ");
   scanf ("%d", livro.codlivro);
   printf("Preco: ");
   scanf ("%f", livro.preco);
   fwrite (&livro, sizeof(livro), 1, arq);
   printf ("Adiciona outro livro (S/N)? %c");
} while (toupper(opcao) == 'S');
fclose (arq);
  getch();
}
```

16. Remoção de Arquivo

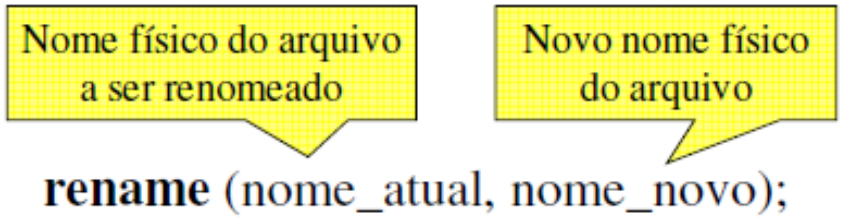


- A função remove() remove um arquivo. Para tanto deve ser passado o nome físico do arquivo fechado.
- Se a remoção do arquivo ocorrer devidamente, a função retorna o valor 0; caso contrário, retorna um valor diferente de zero.

16. Remoção de Arquivo

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> // inclui apenas para usar o sleep

int main(void) {
    FILE *fp;
    char * file_name;

    file_name = "testeremove";

    printf("Criando arquivo\n");

    fp = fopen(file_name, "w");
    if (fp == NULL)
    {
        printf("erro ao criar o arquivo para escrita\n");
    }
    else
    {
        printf("Colocando um conteudo no arquivo\n");

        fprintf(fp, "Colocando um conteudo qualquer");
        fclose(fp);

        sleep(10); // aguarda 10 segundos antes de apagar o arquivo
        int ret;
        ret = remove(file_name);
        printf("%d\n",ret);
    }

    return EXIT_SUCCESS;
}
```

17. Renomear Arquivo



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.

17. Renomear Arquivo

```
#include <stdio.h>
#include <stdlib.h>

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");
      }
      system("pause");
}
```

14. REFERÊNCIAS

- Márcio Alexandre Marques. Algoritmos - Lógica para Desenvolvimento de Programação de Computadores. 1ª Ed. Editora Érica, 2010.

- Sandra Rita. TREINAMENTO EM LOGICA DE PROGRAMAÇAO, Digerati Books, 1 ed. 2009.

- SIMÃO, DANIEL HAIASHIDA; REIS, WELLINGTON JOSÉ DOS. LOGICA DE PROGRAMAÇAO. São Paulo : EDITORA VIENA, 2015. 176p.

- Souza, Marco Antonio Furlan de *et. all*, Algoritimos e Lógica de Programação. 2 ed. São Paulo : Nobel, 2011.