

Flávio Augusto de Freitas

Algoritmos e Estruturas de Dados III usando C/C++

<http://flavioaf.blogspot.com>

# C/C++

**Tutorial 1 (usando Dev-C++ versão 4.9.9.2)**

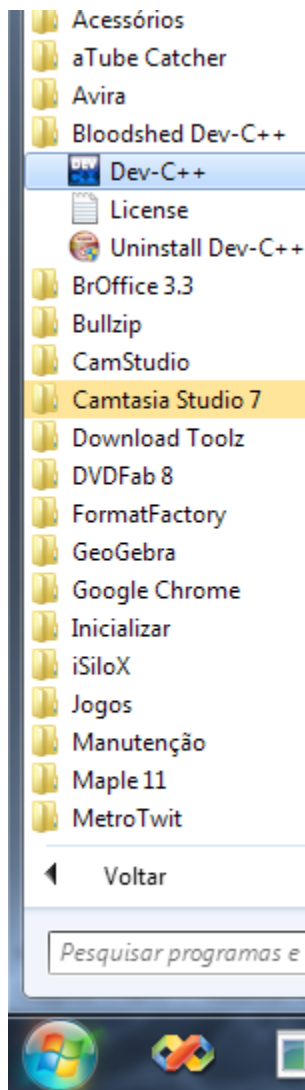


2012

# 1 INTRODUÇÃO

Esta série de tutoriais foi escrita usando o **Microsoft Windows 7 Ultimate** e o **Bloodshed Dev-C++** versão 4.9.9.2, que pode ser baixada em <http://www.bloodshed.net>.

## 2 EXECUTANDO O DEV-C++



Clique no botão Iniciar, depois clique em Todos os programas; em seguida clique em Bloodshed Dev-C++; finalmente clique em Dev-C++.

## 3 O AMBIENTE DO DEV-C++

### 3.1 CONFIGURANDO A LINGUAGEM

**Meu Dev-C++ está em inglês, o que faço?**


Clique no menu Tools, depois clique em Environment Options. Na janela que se abre, clique na aba Interface. Onde está escrito English (Original), mude para Portuguese (Brazil) e clique no botão Ok.

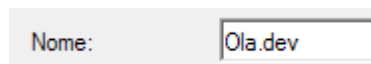
### 3.2 CRIANDO SEU PRIMEIRO APLICATIVO

Depois de executar o programa, clique no menu Arquivo, depois aponte o mouse para a opção Novo e clique em Projeto... . Na janela que se abre, clique em

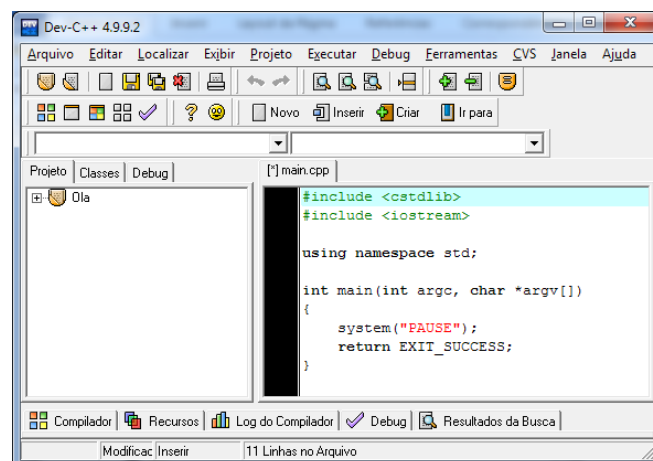


Ainda na mesma janela, onde está escrito Projeto1, mude para **Ola**. Clique no botão Ok. Agora, vamos salvar previamente nosso projeto. Na janela que se abriu, no lado esquerdo, clique no botão Área de Trabalho (ou Desktop, se estiver escrito).

Clique no botão  para criar uma nova pasta na Área de Trabalho. Digite o nome ProjetoOla e pressione a tecla Enter. Dê dois cliques na pasta ProjetoOla, recém criada. Em seguida, na mesma janela, observe que o projeto já tem nome (Ola.dev).



Clique no botão Salvar. A seguinte tela é mostrada:

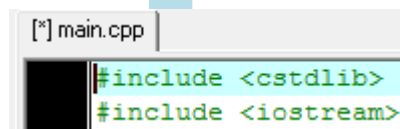


### 3.1 ENTENDENDO O AMBIENTE DO DEV-C++

Observe na tela que há um menu principal e algumas barras de ferramentas. Logo abaixo a janela é dividida em duas partes.

A parte da esquerda é o navegador do projeto. Clique no sinal de + ao lado de Ola. Veja que o Dev-C++ mostra os arquivos que compõem o projeto. Em nosso caso, há apenas o main.cpp.

A parte da direita mostra o arquivo selecionado. Observe que a aba mostra o nome do arquivo e um sinal de (\*).



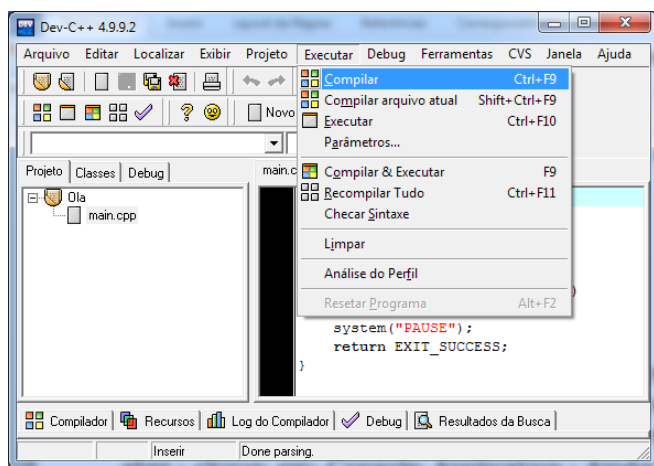
Isto significa que ainda não salvamos o arquivo corrente. Vamos salvá-lo agora. Clique no menu Arquivo, depois clique em Salvar. Na caixa que aparece, não modifique nada, apenas clique no botão Salvar. Observe que o (\*) sumiu. Sempre que alterarmos o código o Dev-C++ mostra o (\*), indicando que precisamos salvar o arquivo.

## 3.2 EDITANDO O TEXTO

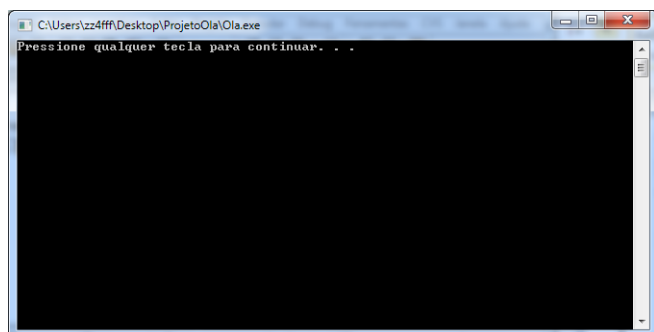
Notou a linha destacada (com fundo azulado). Esta é a linha corrente de edição do texto. Use as teclas de setas do teclado para mover-se pelo texto. Experimente agora. Para inserir texto basta deslocar o cursor usando as teclas de setas e digitar. Mas não faça nada ainda. Veja também que várias palavras aparecem ou coloridas, ou em **negrito**. O Dev-C++ destaca as palavras reservadas da linguagem com diversas cores e negritos ou *itálicos*. Assim você não se perde com muita frequência. Continue lendo.

## 3.3 EXECUTANDO O PROGRAMA

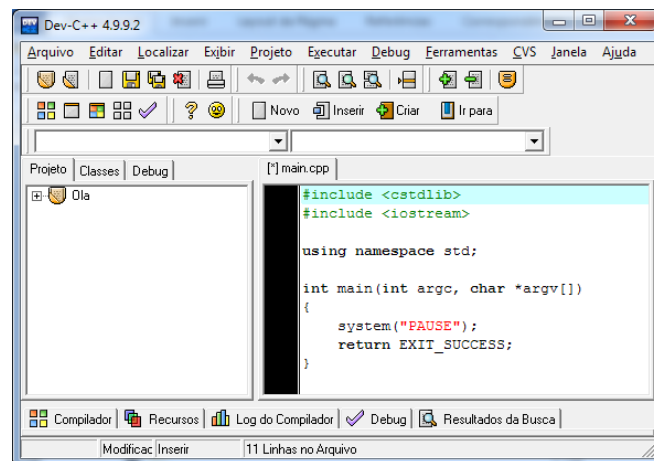
Vamos executar o programa e ver o que acontece. Clique no menu Executar e depois em Compilar & Executar, como abaixo:



Ou pressione F9, que é a tecla de atalho. Aliás, vários menus têm alguma tecla de atalho. Aprenda as principais: Ctrl-S salva o arquivo selecionado; Ctrl-F localiza texto; Para compilar e executar o programa, basta pressionar F9, sem ter de ir até o menu Executar. Após a execução, a seguinte tela se apresenta:



Pressione qualquer tecla e o programa termina, voltando ao Dev-C++.



Mas o que está escrito? O que são estes códigos?

```
#include <cstdlib>
#include <iostream>
```

Estas duas linhas incluem as bibliotecas de códigos necessárias para que o programa execute. Cada programa terá tantas bibliotecas quantas forem preciso. Há centenas delas.

```
using namespace std;
```

Isto é padrão no Dev-C++. Todo programa deste tutorial terá esta linha de código.

```
int main(int argc, char *argv[])
```

Esta é a declaração da função main. Todo programa em C/C++ tem pelo menos uma função com o nome main. Isto é obrigatório.

Vamos analisar esta linha:

O **int**, diz que a função main deve retornar um valor inteiro ao final da execução. Isto é feito pela linha de código **return EXIT\_SUCCESS;**. Entre parênteses existem dois argumentos, que são valores que podem ser passados para o programa, quando ele é executado. Aqui **int argc** indica quantos argumentos foram passados e **char \*argv[]** é a lista de argumentos que foram passados.

Finalmente **system("PAUSE");** indica ao compilador para mostrar uma mensagem padronizada do sistema operacional. É aquela mensagem "**Pressione qualquer tecla para continuar. . .**" que apareceu quando você executou o programa.

## 4 UM EXEMPLO COMPLETO

Modifique o código do programa para que fique como na imagem abaixo:

```
#include <cstdlib>
#include <iostream>

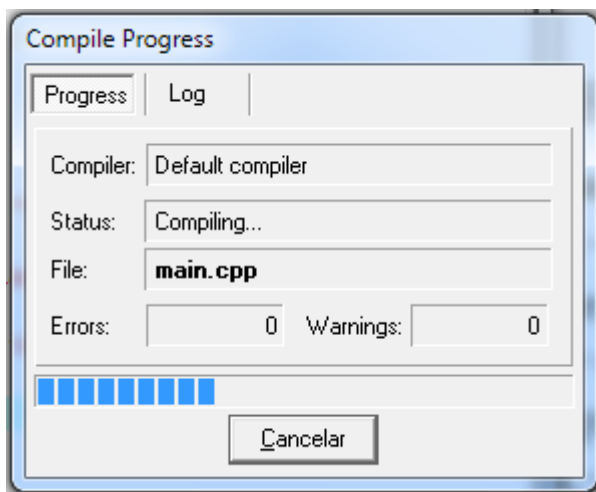
using namespace std;

int main(int argc, char *argv[])
{
    char nome[10];
    int idade;

    printf("Insira o seu nome: ");
    scanf("%s", &nome);
    printf("\n\nInsira a sua idade: ");
    scanf("%d", &idade);
    printf("O seu nome eh %s e tem %d anos.\n", nome, idade);

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

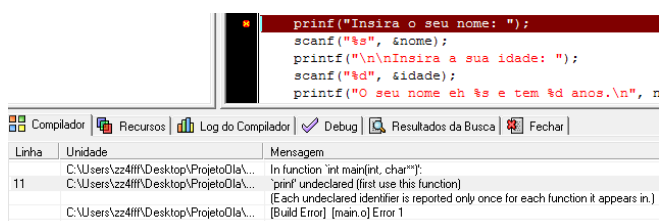
Após a digitação, pressione Ctrl-S para salvar o arquivo. Pressione F9 para executar o programa. A seguinte janela aparece:



Ela mostra o progresso da compilação. É importante você observar as partes

Errors: 0 e Warnings: 0

que mostram se houve erros (*Errors*) ou avisos (*Warnings*), respectivamente. Avisos não precisam ser levados tão a sério, ao menos por enquanto para você, mas erros sim, pois impedem o programa de executar. Caso haja um ou mais erros, após a compilação, o compilador se mostrará assim:



Observe que na parte em que editamos o código, uma linha ficou marcada com fundo vermelho escuro. É a linha onde o Dev-C++ encontrou um erro. Mas que erro? Abaixo da janela de código, observe a linha destacada com fundo cinza. Na coluna Linha, vemos qual linha está o erro encontrado (linha 11). Na coluna Mensagem, podemos ler a mensagem

'printf' undeclared (first use this function)

que indica que o Dev-C++ não encontrou a função 'printf'. Voltando ao código, descobrimos o erro: digitamos incorretamente o nome da função printf (faltando a letra t) e o Dev-C++ nos avisou. Basta corrigir e tentar compilar novamente, pressionando F9. Para ver um erro ocorrendo, experimente trocar a palavra **char** por **chr** e tente executar novamente o programa.

Se o programa compilar e executar corretamente, você deverá ver uma tela preta, com uma mensagem solicitando o seu nome:

Insira o seu nome:

Digite seu nome (Flavio, por exemplo) e pressione Enter.

Em seguida, o programa solicitará sua idade:

Insira a sua idade:

Digite sua idade (42, por exemplo) e pressione Enter.

Em seguida o programa mostra uma mensagem, parecida com o exemplo abaixo.

O seu nome eh Flavio e tem 42 anos.  
Pressione qualquer tecla para continuar. . .

Pressione qualquer tecla, Enter, por exemplo, e o programa termina e retorna ao compilador.

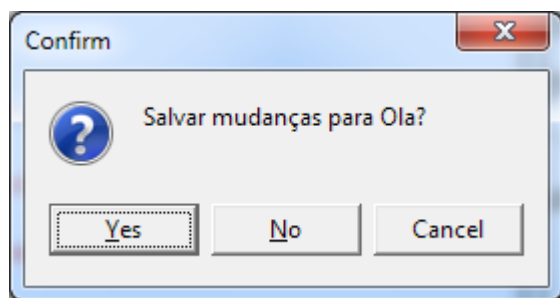
## 5 ENDENTAÇÃO

É o alinhamento do texto. Observe que o código entre { } está deslocado para a direita. Isto é importante para a visualização e entendimento do código escrito. Quanto maior for seu código, ou seja, quanto mais linhas de código tiver seu programa, mais importante será a endentação.

## 6 SAINDO DO DEV-C++

Agora que terminamos este programa, podemos sair do ambiente do Dev-C++. Para isto, clique no menu Arquivo e depois clique na opção Sair.

Caso alguma coisa não esteja salva, o Dev-C++ mostrará a seguinte tela:



Basta clicar no botão Yes para o Dev-C++ salvar o arquivo e terminar. Se desistir de sair do Dev-C++ basta clicar no botão Cancel e você continuará com o Dev-C++ aberto.

## 7 REABRINDO O ÚLTIMO PROJETO

Entre novamente no Dev-C++ (se tiver esquecido, volte ao tópico número 2). Com o Dev-C++ aberto, clique no menu Arquivo e depois aponte o mouse para a opção Reabrir. Clique na opção com o número 0 (zero). Pronto, reabrimos o projeto Ola.

Se não estiver vendo o código do programa, clique no sinal de + ao lado de Ola (lado esquerdo da tela, no navegador de projeto), depois clique em main.cpp, que aparece embaixo de Ola. O código deve aparecer no lado direito.

## 8 ENTRADA E SAÍDA DE DISCO

Agora que já relembramos o uso do DevC++ e um pouquinho de linguagem C/C++, vamos ao que realmente interessa nesse tutorial, ou seja, trabalhar com arquivos em disco.

### 8.1 O PONTEIRO DE ARQUIVO:

A linha comum que une o sistema de E/S de disco do DevC++ é o ponteiro de arquivo. Um ponteiro de arquivo é um ponteiro para uma área na memória (buffer) onde estão contidos vários dados sobre o arquivo a ler ou escrever, tais como o nome do arquivo, estado e posição corrente. O buffer apontado pelo ponteiro de arquivo é a área intermediária entre o arquivo no disco e o programa.

Este buffer intermediário entre arquivo e programa é chamado 'fluxo', e no jargão dos programadores é comum falar em funções que operam fluxos em vez de arquivos. Isto se deve ao fato de que um fluxo é uma entidade lógica genérica, que pode estar associada a uma

unidade de fita magnética, um disco, uma porta serial, etc. Adotaremos esta nomenclatura aqui.

Um ponteiro de arquivo é uma variável-ponteiro do tipo FILE que é definida em stdio.h.

Para ler ou escrever em um arquivo de disco, o programa deve declarar uma (ou mais de uma se formos trabalhar com mais de um arquivo simultaneamente) variável ponteiro de arquivo. Para obter uma variável ponteiro de arquivo, usa-se uma declaração semelhante a esta ao declararmos as demais variáveis do programa:

```
FILE *fp;
```

onde fp é o nome que escolhemos para a variável (podia ser qualquer outro).

### FUNÇÕES MAIS COMUNS DE ARQUIVO

Função	Operação
fopen()	Abre um fluxo
fclose()	Fecha um fluxo
putc()	Escreve um caractere para um fluxo
getc()	Lê um caractere para um fluxo
fseek()	Procura por um byte especificado no fluxo
fprintf()	Ê para um fluxo aquilo que printf() é para o console
fscanf()	Ê para um fluxo aquilo que scanf() é para o console
feof()	Retorna verdadeiro se o fim do arquivo é encontrado
ferror()	Retorna verdadeiro se ocorreu um erro
fread()	Lê um bloco de dados de um fluxo
fwrite()	Escreve um bloco de dados para um fluxo
rewind()	Reposiciona o localizador de posição de arquivo no começo do arquivo
remove()	Apaga um arquivo

### 8.2 ABRINDO UM ARQUIVO

A função fopen() serve a dois propósitos. Primeiro, ela abre um fluxo para uso e liga um arquivo com ele. Segundo, retorna o ponteiro de arquivo associado àquele arquivo. Mais frequentemente, e para o resto desta discussão, o arquivo é um arquivo em disco. A função fopen() tem este protótipo:

```
FILE *fopen(char *nome_de_arquivo, char *modo);
```

onde modo é um string contendo o estado desejado para abertura. O nome do arquivo deve ser um string de caracteres que compreende um nome de arquivo válido para o sistema

operacional e onde possa ser incluída uma especificação de caminho (PATH).

Como determinado, a função `fopen()` retorna um ponteiro de arquivo que não deve ter o valor alterado pelo seu programa. Se um erro ocorre quando se está abrindo um arquivo, `fopen()` retorna um nulo.

Como a Tabela abaixo mostra, um arquivo pode ser aberto ou em modo texto ou em modo binário. No modo texto, as sequências de retorno de carro e alimentação de formulários são transformadas em sequências de novas linhas na entrada. Na saída, ocorre o inverso: novas linhas são transformadas em retorno de carro e alimentação de formulário. Tais transformações não acontecem em um arquivo binário.

### Valores para modo

#### Modo Significado

"r"	Abre um arquivo para leitura
"w"	Cria um arquivo para escrita
"a"	Acrescenta dados para um arquivo existente
"rb"	Abre um arquivo binário para leitura
"wb"	Cria um arquivo binário para escrita
"ab"	Acrescenta dados a um arquivo binário existente
"r+"	Abre um arquivo para leitura/escrita
"w+"	Cria um arquivo para leitura/escrita
"a+"	Acrescenta dados ou cria um arquivo para leitura/escrita
"r+b"	Abre um arquivo binário para leitura/escrita
"w+b"	Cria um arquivo binário para leitura/escrita
"a+b"	Acrescenta ou cria um arquivo binário para leitura/escrita
"rt"	Abre um arquivo texto para leitura
"wt"	Cria um arquivo texto para escrita
"at"	Acrescenta dados a um arquivo texto
"r+t"	Abre um arquivo-texto para leitura/escrita
"w+t"	Cria um arquivo texto para leitura/escrita
"a+t"	Acrescenta dados ou cria um arquivo texto para leitura/escrita

Se você deseja abrir um arquivo para escrita com o nome `test` você deve escrever

```
FILE *ofp;  
ofp = fopen("test", "w");
```

Entretanto, você verá frequentemente escrito assim:

```
FILE *out;  
  
if((out=fopen("test","w"))==NULL) {  
    puts("não posso abrir o arquivo\n");  
    exit(1);  
}
```

A macro `NULL` é definida em `STDIO.H`. Esse método detecta qualquer erro na abertura do arquivo como um arquivo protegido contra escrita ou disco cheio, antes de tentar escrever nele. Um nulo é usado porque no ponteiro do arquivo nunca haverá aquele valor. Também introduzido por esse fragmento está outra função de biblioteca: `exit()`. Uma chamada à função `exit()` faz com que haja uma terminação imediata do programa, não importando de onde a função `exit()` é chamada. Ela tem este protótipo (encontrado em `STDLIB.H`):

```
void exit(int val);
```

O valor de `val` é retornado para o sistema operacional. Por convenção, um valor de retorno 0 significa término com sucesso para o DOS. Qualquer outro valor indica que o programa terminou por causa de algum problema (retornado para a variável `ERRORLEVEL` em arquivos "batch" do DOS).

Se você usar a função `fopen()` para abrir um arquivo, qualquer arquivo preexistente com o mesmo nome será apagado e o novo arquivo iniciado. Se não existirem arquivos com o nome, então será criado um. Se você quiser acrescentar dados ao final do arquivo, deve usar o modo "a". Para se abrir um arquivo para operações de leitura é necessário que o arquivo já exista. Se ele não existir, será retornado um erro. Finalmente, se o arquivo é aberto para escrita/leitura, as operações feitas não apagarão o arquivo, se ele existir; entretanto, se não existir, será criado.

## 8.3 ESCRREVENDO UM CARACTERE

A função `putc()` é usada para escrever caracteres para um fluxo que foi previamente aberto para escrita pela função `fopen()`. A função é declarada como

```
int putc(int ch, FILE *fp);
```

onde `fp` é o ponteiro de arquivo retornado pela função `fopen()` e `ch`, o caractere a ser escrito. Por razões históricas, `ch` é chamado formalmente de



int, mas somente o caractere de mais baixa ordem é usado.

Se uma operação com a função `putc()` for satisfatória, ela retornará o caractere escrito. Em caso de falha, um EOF é retornado. EOF é uma macrodefinição em `STDIO.H` que significa fim do arquivo.

## 8.4 LENDO UM CARACTERE

A função `getc()` é usada para ler caracteres do fluxo aberto em modo de leitura pela função `fopen()`. A função é declarada como

```
int getc(FILE *fp)
```

onde `fp` é um ponteiro de arquivo do tipo `FILE` retornado pela função `fopen()`.

Por razões históricas, a função `getc()` retorna um inteiro, porém o byte de mais alta ordem é zero.

A função `getc()` retornará uma marca EOF quando o fim do arquivo tiver sido encontrado ou um erro tiver ocorrido. Portanto, para ler um arquivo-texto até que a marca de fim de arquivo seja mostrada, você poderá usar o seguinte código:

```
ch=getc(fp);
while(ch!=EOF) {
    ch=getc(fp);
}
```

## 8.5 USANDO A FUNÇÃO `fEOF()`

O sistema de arquivo do DevC++ também pode operar com dados binários. Quando um arquivo é aberto para entrada binária, é possível encontrar um valor inteiro igual à marca de EOF. Isso pode fazer com que, na rotina anterior, seja indicada uma condição de fim de arquivo, ainda que o fim de arquivo físico não tenha sido encontrado. Para resolver esse problema, foi incluída a função `fEOF()` que é usada para determinar o final de um arquivo quando da leitura de dados binários. A função `fEOF()` tem este protótipo:

```
int fEOF(FILE *fp);
```

O seu protótipo está em `STDIO.H`. Ela retorna verdadeiro se o final do arquivo tiver sido encontrado; caso contrário, é retornado um zero. Portanto, a seguinte rotina lê um arquivo binário até que o final do arquivo seja encontrado.

```
while(!fEOF(fp))ch=getc(fp);
```

Obviamente, o mesmo método pode ser aplicado tanto a arquivos textos como a arquivos binários.

## 8.6 FECHANDO UM ARQUIVO

A função `fclose()` é usada para fechar um fluxo que foi aberto por uma chamada à função `fopen()`. Ela escreve quaisquer dados restantes na área intermediária (fluxo) no arquivo e faz um fechamento formal em nível de sistema operacional do arquivo. Uma falha no fechamento de um arquivo leva a todo tipo de problemas, incluindo-se perda de dados, arquivos destruídos e a possibilidade de erros intermitentes no seu programa. Uma chamada à função `fclose()` também libera os blocos de controle de arquivo (FCB) associados ao fluxo e deixa-os disponíveis para reutilização.

Como você provavelmente sabe, o sistema operacional limita o número de arquivos abertos que você pode ter em um dado momento, de modo que pode ser necessário fechar alguns arquivos antes de abrir outros. (Comando `FILES=` no `autoexec.bat`).

A função `fclose()` é declarada como:

```
int fclose(FILE*fp);
```

onde `fp` é o ponteiro do arquivo retornado pela chamada à função `fopen()`. Um valor de retorno igual a zero significa uma operação de fechamento com sucesso; qualquer outro valor indica um erro. Você pode usar a função padrão `ferror()` (discutida a seguir) para determinar e reportar quaisquer problemas. Geralmente, o único momento em que a função `fclose()` falhará é quando um disquete tiver sido prematuramente removido do drive ou não houver mais espaço no disco.

### **ferror() e rewind()**

A função `ferror()` é usada para determinar se uma operação em um arquivo produziu um erro. Se um arquivo é aberto em modo texto e ocorre um erro na leitura ou na escrita, é retornado EOF. Você usa a função `ferror()` para determinar o que aconteceu. A função `ferror()` tem o seguinte protótipo:

```
int ferror(FILE*fp);
```

onde `fp` é um ponteiro válido para um arquivo. `ferror()` retorna verdadeiro se um erro ocorreu durante a última operação com o arquivo e falso, caso contrário. Uma vez que cada operação em arquivo determina uma condição de erro, a

função `ferror()` deve ser chamada imediatamente após cada operação com o arquivo; caso contrário, um erro pode ser perdido. O protótipo de função `ferror()` está no arquivo `STDIO.H`.

A função `rewind()` recolocará o localizador de posição do arquivo no início do arquivo especificado como seu argumento. O seu protótipo é:

```
void rewind(FILE*fp);
```

onde `fp` é um ponteiro de arquivo. O protótipo para a função `rewind()` está no arquivo `STDIO.H`.

## 8.6 USANDO `FOPEN()`, `GETC()`, `PUTC()` E `FCLOSE()`

```
// Grava em disco dados digitados no teclado
// até teclar $
#include <stdio.h>
#include <stdlib.h>
main(int argc, char *argv[]) {
    FILE *fp;
    char ch;
    if(argc!=3) {
        puts("Você se esqueceu de informar o nome do
arquivo e o modo!");
        exit(1);
    }
    if((fp=fopen(argv[1], argv[2]))==NULL) {
        // experimentar "wb"/"wt" e testar newline
        puts("O arquivo não pode ser aberto!");
        exit(1);
    }

    do {
        // getche() não funcionaria aqui porque
        // getchar() pega o caracteres do teclado e vai
        // armazenando no buffer até o newline.
        ch=getchar();
        if(EOF==putc(ch, fp)) {
            puts("Erro ao escrever no arquivo!");
            break;
        }
    } while (ch!='$');
    fclose(fp);
    return 0;
    // código de retorno OK p/ DOS. OPCIONAL
}
```

```
// Lê arquivos e exibe-os na tela.
```

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

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

```
main(int argc, char *argv[]) {
    FILE *fp;
    char ch;
    if(argc!=3) {
        puts("Você se esqueceu de informar o nome do
arquivo e o modo!");
        exit(1);
    }

    if((fp=fopen(argv[1], argv[2]))==NULL){
        // experimentar "rb"/"rt" e testar newline
        puts("O arquivo não pode ser aberto!");
        exit(1);
    }

    ch= getc(fp);    // lê um caractere
    while(ch!=EOF) {
        putchar(ch);    // imprime na tela
        ch=getc(fp);
    }
    return 0;
}
```

## 8.7 `GETW()` E `PUTW()`

Funcionam exatamente como `putc()` e `getc()` só que em vez de ler ou escrever um único caractere, leem ou escrevem um inteiro.

### Protótipos:

```
int putw(int i, FILE *fp);
```

```
int getw(FILE *fp);
```

**Header:** `stdio.h`

### Exemplo:

```
putw(100, saida);
```

escreve o inteiro 100 no arquivo apontado por `saida`.

## 8.8 `FGETS()` E `FPUTS()`

Funcionam como `gets()` e `puts()` só que leem e escrevem em fluxos.

### Protótipos:

```
char *fputs(char *str, FILE *fp);
```

```
char *fgets(char *str, int tamanho, FILE *fp);
```

### Observação

`fgets()` lê uma string de um fluxo especificado até que um `newline` seja lido OU tamanho-1 caracteres tenham sido lidos. Se um `newline` é



lido, ele fará parte da string (diferente de gets()). A string resultante termina com um nulo.

**Header:** stdio.h

## 8.9 FREAD() E FWRITE()

Leem e escrevem blocos de dados em fluxos.

### Protótipos:

```
unsigned fread(void *buffer, int num_bytes, int count, FILE *fp);
unsigned fwrite(void *buffer, int num_bytes, int count, FILE *fp);
```

**Header:** stdio.h

No caso de fread(), buffer é um ponteiro para uma área da memória que receberá os dados lidos do arquivo.

No caso de fwrite(), buffer é um ponteiro para uma área da memória onde se encontram os dados a serem escritos no arquivo.

Buffer usualmente aponta para uma matriz ou estrutura (a ser visto adiante).

O número de bytes a ser lido ou escrito é especificado por num\_bytes.

O argumento count determina quantos itens (cada um tendo num\_bytes de tamanho) serão lidos ou escritos.

O argumento fp é um ponteiro para um arquivo de um fluxo previamente aberto por fopen().

A função fread() retorna o número de itens lidos, que pode ser menor que count caso o final de arquivo (EOF) seja encontrado ou ocorra um erro.

A função fwrite() retorna o número de itens escritos, que será igual a count exceto na ocorrência de um erro de escrita.

Quando o arquivo for aberto para dados binários, fread() e fwrite() podem ler e escrever qualquer tipo de informação. O programa a seguir escreve um float em um arquivo de disco chamado teste.dat:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
main() {
    FILE *fp;
    float f=M_PI;
    if((fp=fopen("teste.dat","wb"))==NULL) {
        puts("Nao posso abrir arquivo!");
        exit(1);
    }
    if(fwrite(&f, sizeof(float), 1, fp)!=1) puts("Erro
    escrevendo no arquivo!");
    fclose(fp);
    return 0;
}
```

Como o programa anterior ilustra, a área intermediária de armazenamento pode ser (e frequentemente é) simplesmente uma variável.

Uma das aplicações mais úteis de fread() e fwrite() é o armazenamento e leitura rápidos de matrizes e estruturas (estrutura é uma entidade lógica a ser vista adiante) em disco:

```
/* escreve uma matriz em disco */
#include<stdio.h>
#include<stdlib.h>
main() {
    FILE *fp;
    float exemplo[10][10];
    int i,j;

    if((fp=fopen("exemplo.dat","wb"))==NULL) {
        puts("Nao posso abrir arquivo!");
        exit(1);
    }
    for(i=0; i<10; i++) {
        for(j=0; j<10; j++) {
            exemplo[i][j] = (float) i+j; /* lei de formação
            dos elementos da matriz */
        }
    }
    /* o código a seguir grava a matriz inteira em
    um único passo: */
    if(fwrite(exemplo, sizeof(exemplo), 1, fp)!=1) {
        puts("Erro ao escrever arquivo!");exit(1);}
    fclose(fp);
    return 0;
}
```

```

/* lê uma matriz em disco */
#include<stdio.h>
#include<stdlib.h>

main() {
    FILE *fp;
    float exemplo[10][10];
    int i,j;
    if((fp=fopen("exemplo.dat","rb"))==NULL) {
        puts("Nao posso abrir arquivo!");
        exit(1);
    }

    // o código a seguir lê a matriz inteira em
    // um único passo:
    if(fread(exemplo, sizeof(exemplo), 1, fp)!=1) {
        puts("Erro ao ler arquivo!");
        exit(1);
    }

    for(i=0; i<10; i++) {
        for(j=0; j<10; j++) printf("%3.1f ", exemplo[i][j]);
        printf("\n");
    }
    fclose(fp);
    return 0;
}

```

## 8.10 ACESSO RANDÔMICO A

### ARQUIVOS: FSEEK()

A função `fseek()` indica o localizador de posição do arquivo.

#### Protótipo:

```
int fseek(FILE *fp, long numbytes, int origem);
```

#### Header: stdio.h

- **fp** é um ponteiro para o arquivo retornado por uma chamada à `fopen()`.
- **numbytes (long int)** é o número de bytes a partir da origem até a posição corrente.
- **origem** é uma das seguintes macros definidas em `stdio.h`:

		Valor
Origem	Nome da Macro	numérico
começo do arquivo	SEEK_SET	0
posição corrente	SEEK_CUR	1
fim do arquivo	SEEK_END	2

Portanto, para procurar `numbytes` do começo do arquivo, origem deve ser `SEEK_SET`. Para procurar da posição corrente em diante, origem é `SEEK_CUR`. Para procurar `numbytes` do final do arquivo de trás para diante, origem é `SEEK_END`.

O seguinte código lê o 235º byte do arquivo chamado `test`:

```

FILE *fp;
char ch;

if((fp=fopen("teste","rb"))==NULL) {
    puts("Nao posso abrir arquivo!");
    exit(1);
}
fseek(fp,234,SEEK_SET);
ch=getc(fp); /* lê um caractere na posição 235º */

```

A função `fseek()` retorna zero se houve sucesso ou um valor não zero se houve falha no posicionamento do localizador de posição do arquivo.

```

// utilitário de visualização de disco usando
// fseek() visualiza setores de 128 de bytes
// do disco e apresenta em ASCII e hexadecimal
// digite -1 para deixar o programa
#include <stdio.h>
#include <ctype.h> /* isprint() necessita */
#include <stdlib.h>

```

```
#define TAM 128 /* TAM = 128 */
```

```
char buf[TAM]; /* global */
```

```
void display();
```

```

main(int argc, char *argv[]) {
    FILE *fp;
    long int setor, numlido;
    if(argc!=2) {
        puts("Uso: tmp nome_do_arquivo");exit(1);}
    // esqueceu de digitar o nome de arquivo
    if((fp=fopen(argv[1],"rb"))==NULL) {
        puts("Nao posso abrir arquivo!");
        exit(1);
    }
    for(;;) {
        printf("Informe o setor (-1 p/ terminar): ");
        scanf("%ld",&setor);
        if(setor<0) break;
        if(fseek(fp, setor*TAM, SEEK_SET)) {
            puts("Erro de busca!");
            exit(1);
        }
        if((numlido=fread(buf,1,TAM,fp)) != TAM) {
            puts("EOF encontrado!");
        }
        display(numlido);
    }
    return 0;
}

```

```

void display(long int numlido) {
    long int i,j;

    for(i=0; i<=numlido/16 ; i++) {
        // controla a indexação de linhas:
        // 0 a 8 (128/16 = 8)
        for(j=0; j<16; j++) printf("%02X ", buf[i*16+j]);
        // imprime 16 col. em hexa
        printf(" "); /* separa mostrador hexa do ascii */
        for(j=0; j<16; j++) {
            /* imprime 16 col. em ascii: (só imprimíveis) */
            if(isprint(buf[i*16+j])) printf("%c",buf[i*16+j]);
            else printf(".");
        }
        printf("\n");
    }
}

```

## 9 OS FLUXOS-PADRÕES

Toda vez que um programa começa a execução, são abertos 5 fluxos padrões:

- **stdin** (aponta p/ o teclado se não redirecionado pelo DOS. Ex: MORE < FILE.TXT)
- **stdout** (aponta p/ a tela se não redirecionado pelo DOS. Ex : DIR > PRN)
- **stderr** (recebe mensagens de erro - aponta para a tela)
- **stdprn** (aponta p/ a impressora)
- **stdaux** (aponta p/ a porta serial)

Para entender o funcionamento destes fluxos, note que putchar() é definida em stdio.h como:

```
#define putchar(c) putc(c,stdout)
```

e a função getchar() é definida como

```
#define getchar() getc(stdin)
```

Ou seja, estes fluxos permitem serem lidos e escritos como se fossem fluxos de arquivos. Toda entrada de dado de um programa é feita por stdin e toda saída por stdout:

```
/* localiza uma palavra especificada pela linha de
comando em um arquivo redirecionado para
stdin, mostra a linha e seu número */
```

```
/* rodar no prompt do DOS o seguinte comando:
tmp argv < tmp.c */
```

```

#include<string.h> /* strstr() necessita */
#include<stdio.h>
main(int argc, char *argv[]) {
    char string[128];
    int line=0;

    while(fgets(string, sizeof(string),stdin)) {
        ++line;
        if(strstr(string, argv[1]))
            printf("%02d %s",line,string);
    }
}

```

O fluxo stderr é usado para mostrar mensagens de erro na tela, quando a saída do programa esta redirecionada p/ outro dispositivo que não seja a tela:

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

```

main(int argc, char *argv[]) {
    FILE *fp;
    int count;
    char letter;

    if(argc!=2) {
        puts("Digite o nome do arquivo!");exit(1);
    }
    if(!(fp = fopen(argv[1],"w"))) {
        fputs("Erro na abertura do arquivo",stderr);
        // 2ª opção:
        // puts("Erro na abertura do arquivo");
        // -> vai p/ stdout, isto é, tela
        // ou outro dispositivo
        exit(1);
    }
    else
        for(letter='A'; letter<='Z'; letter++)
            putc(letter, fp);
        fclose(fp);
}

```

Se a saída deste programa (stdout) for redirecionada para algum outro arquivo, a mensagem de erro forçosamente aparecerá na tela porque estamos escrevendo em stderr.

## 9.1 FPRINTF() E FSCANF()

Comportam-se como `printf()` e `scanf()` só que escrevem e leem de arquivos de disco. Todos os códigos de formato e modificadores são os mesmos.

### Protótipos:

```
int fprintf(FILE *fp, char *string_de_controle,
lista_de_argumentos);
int fscanf(FILE *fp, char *string_de_controle,
lista_de_argumentos);
```

**Header:** `stdio.h`

Embora `fprintf()` e `fscanf()` sejam a maneira mais fácil de ler e escrever tipos de dados nos mais diversos formatos, elas não são as mais eficientes em termos de tamanho de código resultante e velocidade. Quando o formato de leitura e escrita for de importância secundária, deve-se dar preferência a `fread()` e `fwrite()`.

```
// imprime os quadrados de 0 a 10 no arquivo
// quad.dat no formato número - quadrado
#include<stdio.h>
#include<stdlib.h>
```

```
main() {
    int i;
    FILE *out;
    if((out=fopen("quad.dat","wt"))==NULL) {
        // sempre texto c/ fscanf() e fprintf()
        puts("Nao posso abrir arquivo!");
        exit(1);
    }
    for (i=0; i<=10; i++) {
        fprintf(out,"%d %d\n", i , i*i);
    }
    fclose(out);
}
```

Deixaremos o exemplo de uso de `fscanf()` para o próximo tutorial (Alocação Dinâmica).

Apagando arquivos: `remove()`

```
int remove(char *nome_arquivo);
```

Retorna zero em caso de sucesso e não zero se falhar.

## 10 TERMINAMOS

Terminamos por aqui. Clique no menu Arquivo, depois clique na opção Sair.

Corra para o próximo tutorial.