ARQUIVOS EM C

Abrindo/criando arquivos em C

Abrir um arquivo em C significa criar uma stream e conectá-la ao arquivo em disco. Em C, fopen( ) é a função que abre (ou cria) arquivos, seu protótipo encontra-se abaixo e seu uso necessita da inclusão de stdio.h.
FILE *fopen(const char *filename, const char *mode);

fopen abre o arquivo nomeado por *filename* em um modo especificado por *mode* que o associa a uma *stream*.

Se a operação de abertura transcorre sem problemas *fopen( )* devolve um ponteiro de referência que será usado para manipular o arquivo, caso haja algum erro um NULL será devolvido.

Os modos de abertura do arquivo **TEXTO** podem ser:

Valor	Descrição
"r" ou "rt"	Abre arquivo texto apenas para leitura e posiciona o ponteiro no início do arquivo. Se o arquivo não existe (previamente), *fopen* devolverá NULL.
"w" ou "wt"	Abre arquivo texto apenas para escrita. Se o arquivo não existe ele será criado. **Se o arquivo já existe ele será aberto para escrita, porém seu conteúdo será apagado**. Posiciona o ponteiro no início do arquivo.
"a" ou "at"	Abre arquivo texto para anexação, nesse caso **só será possível acrescentar dados a partir do ponto de abertura que sempre ocorrerá no final do arquivo**, portanto não há possibilidade de perda dos que já existam no arquivo. Se o arquivo já existe ele será aberto para anexação e o ponteiro de arquivo será posicionado no final deste (EOF – End Of File). Se não existe, ele será criado e o ponteiro posicionado no inicio deste.
"r+" ou "r+t"	Abre arquivo texto para leitura e escrita e posiciona o ponteiro no início do arquivo. Se o arquivo não existe (previamente), *fopen* devolverá NULL.
"w+" ou "w+t"	Abre arquivo texto para escrita e leitura. Se o arquivo não existe ele será criado. Se já existe ele será aberto, porém seu conteúdo será apagado. Posiciona o ponteiro no início do arquivo.
"a+" ou "a+t"	Abre arquivo texto para anexação (será possível acrescentar dados , ou seja, escrever dados sem perda dos que já existam no arquivo) e para leitura. Será possível retornar o ponteiro inclusive para posições anteriores à da abertura, porém para estas posições só será possível e efetuar operações de leitura. Se o arquivo já existe ele será aberto para anexação e o ponteiro de arquivo será posicionado no final deste (EOF). Se não existe, ele será criado e o ponteiro posicionado no inicio deste.

OBS: PARA ABRIR ARQUIVOS **BINÁRIOS** BASTA SUBSTITUIR O SÍMBOLO "t" NOS CASOS ACIMA POR "b". AS DESCRIÇÕES CONTINUAM VÁLIDAS PORÉM OS DADOS SERÃO ARMAZENADOS NA FORMA BINÁRIA SEM TRADUÇÃO.

Em *b* foi utilizado um caminho de diretório para a abertura/criação do arquivo, perceba o uso de duas contra-barras, essa é a maneira que o *C* usa para representar um única barra, de outra forma os caracteres \t, na seqüência ...\teste... seriam interpretados como sinal de tabulação. Observe que o uso das duas contra-barras só tem sentido se o nome do arquivo for fornecido em uma string dentro do programa em C, se for feita uma passagem de parâmetros na linha de comando não devemos utilizar as duas contra-barras, como no exemplo seguinte.

No exemplo *c* utiliza-se a entrada do nome do arquivo pela linha de comando, passando argumentos para a função main( ), supondo que o programa executável chama-se cria_arq.exe e deseja-se abrir um arquivo chamado teste.txt, na linha de comandos pode-se entrar com: C:> cria teste.txt

a)

```
#include "stdio.h"
....
main(void)
{ FILE *fp;
  char filename[ ]=”arq.txt”;

  if ((fp = fopen(filename,"w+"))== NULL)
    {printf("erro na abertura do arquivo");
     exit(0);
    }

 .....
}
```

b)

```
#include "stdio.h"
....
main(void)
{ FILE *fp;
  char filename[ ] = ”c:\\teste\\arq.txt”,
                        modo[ ]=”w+”;

  if ((fp = fopen(filename, modo))= = NULL)
    {printf("erro na abertura do arquivo");
     exit(0);
    }
 .....
}
```

c)

```
#include "stdio.h"
#define MSG_ERR “quantidade invalida de parametros”
....
main(int argc, char *argv[])
{ FILE *fp;
  if(argc < 2)
   {printf(MSG_ERR);
    exit(0);
   }
  if ((fp = fopen(argv[ 1],"w+"))== NULL)
    {printf("erro na abertura do arquivo");
     exit(0);
    }
 .....
}
```

buffer / stream

streams e buffers são elementos lógicos criados para servir de interface entre o dispositivo de memória externa (disco) e a memória interna onde reside o programa de usuário.

Esvaziando ou fechando uma stream e buffer associado:

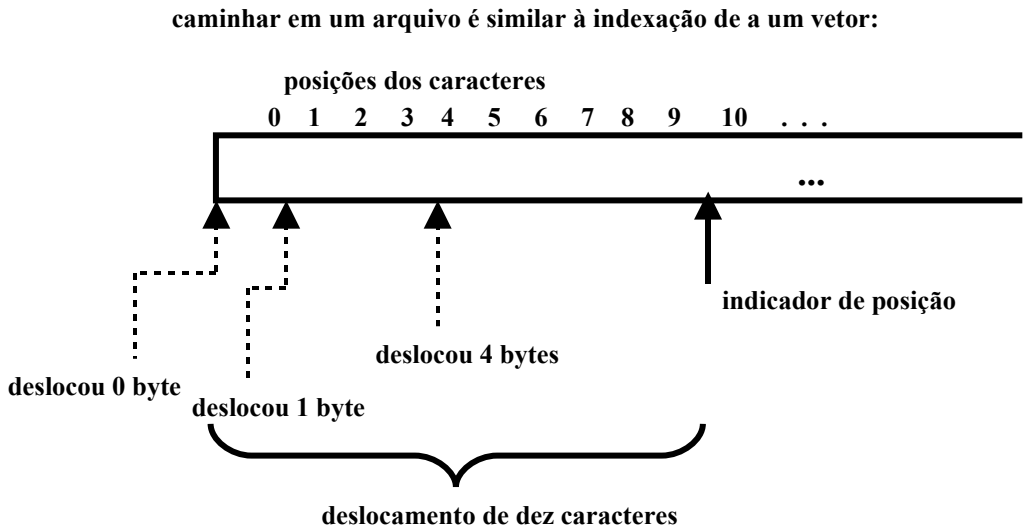
1) int fflush(FILE *fp)
2) int fclose(FILE *fp)

Acesso seqüencial / aleatório

Um arquivo em C funciona como uma espécie de fita com um indicador de posição que revela a qualquer momento, em que ponto desta fita o arquivo será acessado. O deslocamento da posição é sempre especificada em bytes e toma como referência o início do arquivo, ou seja, a posição zero !
Se um arquivo é recém criado seu comprimento é zero e o indicador de posição assim indicará. Se o arquivo pré-existe e é aberto o indicador de posição se posicionará a depender do modo de abertura (a, r,w,...)

As funções de leitura ou escrita utilizam o indicador de posição para executar suas ações. Cada operação atualiza esse indicador de posição em função do número de bytes lidos ou escritos.

Exemplo: ao abrir um arquivo **texto** para leitura, solicitou-se a leitura de 10 caracteres, considerando que cada caracter tenha tamanho 1 byte, tal operação implica no posicionamento do indicador de posição dez bytes à frente. Nessa posição, havendo uma nova leitura, o próximo caracter a ser lido será aquele que está na posição do indicador, nesse caso, aquele ocupa a décima posição:



Para quebrar a forma seqüencial de andamento sobre um arquivo (a fita) pode-se lançar mão de alguns recursos tais como:

a) *void rewind(FILE *fp)* que reposiciona o indicador de posição do arquivo no inicio deste.

b) A função *int fseek(FILE *fp, long numbytes, int origin)* permite forçar o indicador de posição de forma que este "aponte" para qualquer ponto do arquivo. Os parâmetros de fseek() são: o ponteiro do arquivo, um valor referente a distância em bytes que o indicador de posição deve se deslocar e o ponto de referencia para o deslocamento, esse deve ser expresso em um dentre os três valores na tabela abaixo:

Macro constante	valor	Descrição
SEEK_SET	0	Mover *numbytes* a partir do início do arquivo
SEEK_CUR	1	Mover a partir da posição atual
SEEK_END	2	Mover a partir do final do arquivo

c) *long ftell(FILE *fp)* preferencialmente aplicada a arquivos binários, fornece o deslocamento em bytes, do indicador de posição, tendo como referência o inicio do arquivo (lembre-se que antes do primeiro byte, temos a posição zero).

OBSERVAÇÃO:
Se você abrir um arquivo para anexação e de imediato executar *ftell( )* ela devolverá zero, apesar de (normalmente) haver algum conteúdo previamente gravado no arquivo e do indicador de posição estar automaticamente no final do arquivo (modo de anexação). Para que *ftell( )* retorne o valor correto, execute um *fseek(fp,0L,SEEK_END)* antes do f*tell( )*. A função *ftell( )* retorna –1 se houver algum problema na sua execução.

Exemplos: supondo *fp* como um ponteiro para arquivo binário

a) O seguinte comando faz o indicador de posição passar para o nono registro adiante do atual: *fseek(fp, 9 * sizeof(struct list_type), SEEK_SET);*

b)Agora suponha um arquivo aberto (não no modo anexação) um comando: *rewind(fp)*, equivaleria aos comandos:

```
long desloc;
desloc = ftell(fp);
fseek(fp, - desloc,1);
/*o sinal negativo e a definição de origin como 1, faz com que o deslocamento seja
feito em direção ao início do arquivo, partindo da posição atual do indicador de
posição. */
```

c) posicionando no final do arquivo:

```
fseek(fp,0L,SEEK_END);      /* SEEK_END é definido como valor 2, referencia a
                               partir do final do arq. */
```

e) posicionando no inicio do arquivo: *fseek(fp,0L,SEEK_SET);*

Funções para leitura/escrita em arquivos (modo texto)

1) *int putc(int ch, FILE *fp)*, escreve um caractere (contido no byte menos significativo do inteiro *ch*) em um arquivo cuja referência é *fp*. Se bem sucedida devolve o caractere escrito, senão devolve o caractere EOF (end-of-file).

2) *int getc(FILE *fp)*, lê um caractere de um arquivo referenciado por *fp*. Devolve EOF se o final do arquivo for encontrado.

3) *int ungetc(int char, FILE *stream)* devolve o caractere *char* para o arquivo *stream* de maneira que este caractere fique disponível para a próxima leitura.

4) *int fputs(const char *str, FILE *fp)* escreve uma string apontada por *str* em um arquivo referenciado por *fp*, fputs( ) devolve EOF se ocorrer algum erro na operação. A função também traduz o caracter '\n' é para um par CRLF carriage return - line feed, sinalizando que a próxima string (linha) escrita no arquivo será visualizada abaixo da anterior.

5) *int fgets(char *str, int tam, FILE *fp)* lê uma cadeia de caracteres do arquivo referenciado por *fp* e atribui essa string à *str*. A leitura de cada cadeia de caracteres no arquivo se dará até que sejam lidos (**tam** −1) caracteres ou que seja encontrado um caractere de nova linha, o que ocorrer primeiro. Cada CRLF é reconvertido em '\n' pela função *fgets( )* que também acrescenta '\0'ao final da string de destino, *str*.
   fgets( ) devolve NULL se ocorrer erro na operação ou se EOF for encontrado.

6) fprintf( ) e fscanf( ):
   Funções para leitura e escrita formatada, são para arquivos em disco o que printf( ) e scanf( ) são para os dispositivos de saída e entrada padrão (vídeo e teclado).
   Protótipos:

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

Encontrando o final de arquivo

Em arquivos modo texto pode-se lançar mão de getc( ) pois ela devolve EOF quando encontra o final de arquivo:

```
while(( c = getc(fp)) != EOF)
{ .....................
}
```

No entanto se estivermos trabalhando em modo binário não há como detectar o final de arquivo procurando pelo EOF.

Para detecção de final de arquivo em modo binário deve-se lançar mão da função *int feof(FILE *fp)*). Esta devolve zero enquanto não encontra o final do arquivo, e algo diferente de zero quando o final é detectado.

Atenção: FEOF( ) É UMA FUNÇÃO ASSOCIADA A OPERAÇÕES DE LEITURA
Ao usar *feof( )* para controlar um laço, tenha certeza de que imediatamente antes foi executada uma operação de leitura (fread, fgets, etc) isso deve ser feito para evitar a repetição da iteração sobre o último registro lido. Então, para detecção de final de arquivo, imediatamente antes do *feof( )* certifique-se que foi feita uma operação de leitura.

.....
for (operação de leitura; !feof(fp) ; operação de leitura)
{ .....
}
.....
	.....
	While(Operação leitura && ! feof( ))
	{ .....
	.....
	}
	.....

O seguinte fragmento de código leva a uma duplicação na exibição da última leitura: *while(! feof(fp))* *{ fgets(str,MAX,fp.);* *printf(str);* ..... *}*	Nesse também ocorre problema, o feof( ) não reconhece o avanço definido por fseek: *while(! fseek(fp,sizeof(reg),1) && ! feof(fp))* *{ *nblocos)++;* ..... *}*

Usando putc/getc e fprintf/fscanf

```
main(void)
{       FILE *fp;
        char letra, nome_arq[ ] = "teste.txt";
  if ((fp = fopen(nome_arq,"w+"))==NULL)
    {   printf("erro na abertura do arquivo");
        exit(0);
    }
  puts("entre com uma linha ou CR para encerrar");
  while(1)
  { letra = getc(stdin);
    if (letra == '\n')
        break;
        else
    {   puts("------> escrevendo com putc()");
        fputc(letra,fp);
    }
  }
  rewind(fp);
  puts("-------> lendo com getc");
  while(!feof(fp))
  {  letra =  getc(fp);
    putchar(letra);
  }
  fclose(fp);
}
```

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "conio.h"

void main(void)
{       FILE *fp;
        int i, oct,dec, hexa;
        char  nome_arq[ ] = "teste.txt";
  if ((fp = fopen(nome_arq,"w+"))==NULL)
    {   printf("erro na abertura do arquivo");
        exit(0);
    }

   printf("\n entre com inteiro  octal e hexadecinal\n");
   scanf("%i %o %x",  &i, &oct, &hex);

   puts("gravando com fprintf");
   for(i=0;i < 4;i++)
   {   printf("--->%o\t --->%i\t--->%x\t\n",oct,i,hex);
       fprintf(fp, "--->%o\t --->%i\t--->%x\t\n",i,i,i);
   }

  rewind(fp);

  /* lendo com fscanf */
  puts("lendo com fscanf");
  while(!feof(fp))
  { fscanf(fp, "--->%o\t --->%i\t--->%x\t\n",&oct,&dec,&hexa);
    printf("--->%o\t --->%i\t--->%x\t\n",oct,dec,hexa);
  }
  fclose(fp);
}
```

Usando fgets/fputs

```
main(void)
{ FILE *fp;
.....
 /* escreve usando fputs */
  while(1)
   {      puts("entre com uma linha ou 'fim' para encerrar");
          gets(fr);
          if (strcmp(fr,"fim")== 0)
                break;
          else
              { strcat(fr,"\n" );
                fputs(fr,fp);
              }
    }
/* "rebobinando" o arquivo*/
       rewind(fp);
//lendo com fegts
for( fgets(fr,MAX,fp); !feof(fp); fgets(fr,MAX,fp))
          printf(fr);

}
```

Funções para leitura/escrita em arquivos binários

As funções fread( )/fwrite( ) são utilizadas para ler e escrever blocos de dados de qualquer tipo, através delas é possível escrever ou ler de um arquivo, por exemplo, toda uma matriz , estrutura, ou qualquer outro tipo de dado.
Protótipos:

*int fread(void *buf, int size, int count, FILE *fp);*
int fwrite*(void *buf, int size, int count, FILE *fp);*

O parâmetro *buf* para *fread( )* é um ponteiro para a região de memória que receberá os dados lidos, já para fwrite( ),o mesmo parâmetro se refere a uma região de memória que contém informações a serem enviadas para o arquivo.

O parâmetro *size* corresponde ao número de bytes a serem lidos ou escritos.
O parâmetro *count* determina quantos itens (cada um de comprimento *size* bytes) serão lidos ou escritos (usualmente *count* é igual a 1).
Finalmente *fp* é o ponteiro de arquivo

Exemplo usando fwrite( )/fread( )

```
typedef struct {        int i;
                        char str[30];
        }reg;

main(void)
{       FILE *fp;
        int count, size;
        reg registro, aux;

        char nome_arq[ ] = "teste.bin";
        registro.i = 100;
        strcpy(registro.str,"valor");

        if ((fp = fopen(nome_arq,"w+b"))==NULL)
          {   printf("erro na abertura do arquivo");
              exit(0);
          }
        puts("------> escrevendo com fwrite()");
        size = sizeof(reg);
        count = 1;
        if ( (fwrite(&registro,size,count,fp)) < count)
          {   puts("erro na operacao de escrita");
              exit(0);
          }

        rewind(fp);
        puts("------> lendo com fread( )");
        fread(&aux,sizeof(reg),1,fp);
        printf(" valores lidos: %i, %s ", aux.i,aux.str);

        fclose(fp);
}
```

MAIS DUAS FUNÇÕES ÚTEIS
int rename(const char *oldname, const char *newname);

A função rename( ) altera o nome do arquivo especificado por *oldname* para o valor especificado em *newname*. ATENÇÃO: o arquivo especificado por oldname deve estar fechado !
A função devolve zero se bem sucedida e diferente de zero caso ocorra erro.

int remove(const char *file_name);
A função apaga o arquivo especificado por file_name. Devolve zero se a operação foi bem sucedida caso contrario devolve diferente de zero. ATENÇÃO: o arquivo especificado por file_name deve estar fechado !