Manipulação de Arquivos

- Existem dois tipos de streams (seqüência de bytes de dados – caracteres) em C:

  - *Texto* – composto apenas de caracteres organizados em linhas de no máximo 255 caracteres. Cada linha é terminada com um caractere de fim de linha (ex.: CR-LF no DOS);

  - *Binário* – qualquer tipo de dado, incluindo texto. Bytes de dados em um stream binário não são traduzidos ou interpretados de forma diferente. Por exemplo, no modo *texto*, a combinação dos caracteres correspondentes ao *carriage return* (CR) e *line feed* (LF) em um arquivo são interpretados como um "`\n`" ao imprimir na tela. Em modo binário, não recebem tratamento específico, sendo impresso os caracteres "especiais" como parte da mesma linha.

Tipo de Arquivo	**Vantagens**	**Desvantagens**
Texto	- *Facilidade de leitura:* os dados podem ser lidos por qualquer programa, caractere por caractere.	- *Maior gasto de memória:* gasta 1 byte por caractere, exigindo, por exemplo, 10 bytes para armazenar o número 123456.789 que gastaria 4 bytes em um `float`); - *Maior gasto de tempo em buscas:* para acessar o *n*-ésimo elemento, exige uma busca seqüencial acessando todos os elementos anteriores no arquivo.
Binário	- *Menor gasto de memória*: similar à forma como é armazenado em memória RAM, onde o número 123456.789 gastaria 4 bytes equivalente a um `float`; - *Menor gasto de tempo em buscas*: para saber a posição do *n*-ésimo número fracionário de uma lista de números fracionários, bastaria localizar a posição `n*sizeof(float)` movendo o cursor do arquivo.	*Dificuldade de leitura:* apenas o criador do arquivo sabe como manipulá-lo.

Nomes de arquivos

- Todo arquivo em memória secundária (disco rígido, disquete, CD, etc.) possui um nome.
- Regras para a composição desses nomes variam de um sistema operacional para outro.
  - *DOS e Windows 3.x* – 8 caracteres seguido opcionalmente de ponto e extensão (3 caracteres).
  - *Windows 9x, 2000, ME, XP, NT e maioria dos sistemas UNIX – 256 caracteres.*

- Os seguintes caracteres não são permitidos na composição do nome do arquivo no Windows 9x (cada sistema operacional possui suas próprias regras de composição de caracteres para nome de arquivos):

  `/ \ : * ? " < >`

- Um nome de arquivo em C pode conter a informação do seu caminho (drive e/ou diretório onde o arquivo se encontra no disco). Exemplo: o arquivo `info.txt` no diretório `c:\dados\rodrigo` pode ser referenciado em C da seguinte maneira:
  - *no Windows:*
    ```
    char *arquivo = "c:\\dados\\rodrigo\\info.txt";
    ```
  - *no UNIX:*
    ```
    char *arquivo = "c:\/dados\/\/rodrigo\/info.txt";
    ```

Abrindo arquivos

- Para abrir um arquivo, usa-se a função `fopen()` que retorna um ponteiro para o tipo `FILE`, uma estrutura declarada na biblioteca `stdio.h`.
- *Sempre* que abrir um arquivo com `fopen()`, antes de terminar o programa, fechá-lo com `fclose()`.
- Para cada arquivo que se deseja abrir, deve-se declarar um ponteiro para `FILE`. Ao chamar a função `fopen()`, ela cria uma instância da estrutura `FILE` e retorna um ponteiro para ela. Eis a função:

  ```
  FILE *fopen(const char *<nome arquivo>, const char *<modo>);
  ```

  - `<nome arquivo>` – nome do arquivo (com ou sem o caminho) a ser aberto.
  - `<modo>` – modo de abertura do arquivo, controlando se é arquivo texto ou binário, para leitura, escrita ou ambos. Os possíveis modos de abertura de um arquivo são mostrados na tabela a seguir:

Modo	***Significado***	***Se o arquivo NÃO existe...***	***Se o arquivo existe...***
r	Abre arquivo para leitura.	O `fopen()` retorna `NULL`.	
w	Abre o arquivo para escrita.	O arquivo é criado.	O arquivo é apagado sem qualquer aviso e um novo arquivo vazio é criado em seu lugar.
a	Abre arquivo para adicionar novos caracteres.	O arquivo é criado.	O arquivo é aberto para adição de caracteres no fim.
r+	Abre o arquivo para leitura e escrita.	O `fopen()` retorna `NULL`.	O arquivo é aberto para adição de caracteres no início, sobrescrevendo caracteres já existentes.
w+	Abre o arquivo para leitura e escrita.	O arquivo é criado.	O arquivo é aberto para adição de caracteres no início, sobrescrevendo o arquivo inteiro.
a+	Abre o arquivo para leitura e para adicionar novos caracteres.	O arquivo é criado.	O arquivo é aberto para adição de caracteres no fim.

- O programa a seguir verifica se foi possível abrir o arquivo texto desejado:

```
#include <stdio.h>
void main (void) {
   FILE *arquivo;
   arquivo = fopen ("aula.txt", "r");
   if (arquivo != NULL) {
      printf ("Arquivo aberto com sucesso");
      fclose (arquivo);
   }
   else
      printf ("Não foi possível abrir o arquivo");
}
```

- O modo padrão de abertura de arquivo é texto. Para abrir um arquivo no modo binário, basta adicionar um "b" no argumento `<modo>`. Exemplo: `"rb", "wb", "ab", "r+b",` etc.

Funções principais para escrita e leitura de arquivos

fprintf() – para arquivos texto

- Funciona como o `printf()`, exceto pelo fato de que envia a saída para o *stream* e não para o monitor.
- Não funciona para arquivos binários pois converte os dados em texto. É necessário usar a função `fwrite`.
- Sintaxe da função: `int fprintf(FILE *arquivo, char *formato_caracteres, ...);`

```
FILE *arq;
arq = fopen("a.txt", "w");
fprintf(arq, "%d/%d/%d", 2, 5, 2006); //escreve no arquivo texto:"2/5/2006"
fclose(arq);
```

fwrite() – para arquivos binários

- Recebe 4 parâmetros: um ponteiro para o vetor em memória, o tamanho de cada elemento do vetor, o número de elementos do vetor e a variável associada ao arquivo.
- Sintaxe da função: `int fwrite(char *vet, int tam, int num, FILE *arquivo);`

```
FILE *arq;
int vet[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
arq = fopen ("a.dat", "wb");
fwrite(vet, sizeof (int), 10, arq);
fclose (arq);
```

fscanf() – para arquivos texto

- Funciona como o `scanf()`, exceto pelo fato de que lê do arquivo e não do teclado.
- Sintaxe da função: `int fscanf(FILE *arquivo, char *formato_caracteres, ...);`

```
int dia, mes, ano;
FILE *arq;
arq = fopen("a.txt", "r");
fscanf(arq, "%d/%d/%d", &dia, &mes, &ano); // armazena data nas variaveis
fclose(arq);
```

fread() – para arquivos binários

- Sintaxe da função: `int fread(char *vet, int tam, int num, FILE *arquivo);`

```
int i, vet[10];
FILE *arq;
arq = fopen ("a.dat", "rb");
fread(vet, sizeof (int), 10, arq);  // copia os dez inteiros do arquivo para o vetor
fclose (arq);
```

getc() e fgetc() – para arquivos texto

- Funções idênticas. Devolvem o caracter apontado pelo ponteiro `FILE` ou `EOF` (*end of file*) em um erro.
- Sintaxe da função: `char fgetc(FILE *arquivo);`

```
FILE *arq;
arq = fopen("a.txt", "r");
char c = fgetc(arq);
fclose(arq);
```

fgets() – para arquivos texto

- Para ler uma linha de caracteres do arquivo.
- Sintaxe da função: `char *fgets(char *string, int tamanho, FILE *arquivo);`

```
char str[255];
FILE *arq;
arq = fopen("a.txt", "r");
fgets(str, 255, arq);
fclose(arq);
```

feof() – para arquivos texto e binários

- Retorna zero se o final do arquivo foi atingido e um valor não nulo caso contrário.
- Sintaxe da função: `int feof(FILE *arquivo);`

```
char str[255];
FILE *arq;
arq = fopen("a.txt", "r");
while ( !feof(arq) ) {      /* imprime todo o arquivo no monitor */
   fgets(str, 255, arq);
   printf("%s", str);
}
fclose(arq);
```

rewind() – para arquivos texto e binários

- Retorna o indicador de posição do ponteiro `FILE` em relação ao arquivo para o seu início.
- Sintaxe da função: `void rewind(FILE *arquivo);`

```
char str[255];
FILE *arq;
arq = fopen("a.txt", "r");
for (int i = 0; i < 5; i++) {  /* imprime o arquivo 5 vezes no monitor */
   while ( !feof(arq) ) {
      fgets(str, 255, arq);
      printf("%s", str);
   }
   rewind(arq);
}
fclose(arq);
```

fseek() – para arquivos binários

- O fato de podermos guardar facilmente estruturas de tamanho fixo em arquivos binários permite que possamos localizar diretamente um elemento no arquivo sem precisar ler todos os elementos que o precedem, como teria de ser feito em um arquivo texto. Para ler diretamente o elemento desejado, é necessário mover o "cursor"do arquivo até a posição desejada, utilizando a função `fseek`.
- Recebe 3 parâmetros: a variável associada ao arquivo, um deslocamento em bytes (que pode ser tanto positivo quanto negativo) e o ponto relativo ao qual o deslocamento é realizado. Esse último pode assumir 3 valores constantes:
  - SEEK_SET, indicando que o deslocamento deve iniciar no começo do arquivo;
  - SEEK_CUR, indicando que o deslocamento é a partir da posição atual do cursor; e
  - SEEK_END, indicando que o deslocamento deve ser iniciado a partir do fim do arquivo.
- Sintaxe da função: `fseek(FILE *arquivo, int deslocamento, int pontoInicio);`
- O programa abaixo abre um arquivo com um vetor de inteiros, lê a *i*-ésima posição do vetor (*i* indicado pelo usuário), incrementa uma unidade e grava no mesmo ponto do arquivo.

```
#include <stdio.h>

int main (void)
{
   int vet[4] = {5, 10, 15, 20};
   int i, elem;
   FILE *arq;

   arq = fopen ("vet.dat", "w+b");
   if (arq != NULL) {
      fwrite(vet, sizeof (int), 4, arq);
      scanf ("%d", &i);
      if (i >= 0 && i < 4) {
         // Posiciona o cursor no i-esimo elemento
         fseek (arq, i * sizeof (int), SEEK_SET);
         // le um inteiro
         fread (&elem, sizeof (int), 1, arq);

         elem = elem + 1;
         // Volta 1 posicao (reposiciona no i-esimo)
         fseek (arq, -sizeof (int), SEEK_CUR);
         // grava o novo elemento
         fwrite (&elem, sizeof (int), 1, arq);
      }
      rewind(arq);
      fread(vet, sizeof(int), 4, arq); // carrega todo o arquivo no vetor
      for (i = 0; i < 4; i++)          // imprime o arquivo alterado
         printf("%d ", vet[i]);
      fclose (arq);
   }
   else
      printf ("Nao foi possivel abrir o arquivo");

   return 0;
}
```

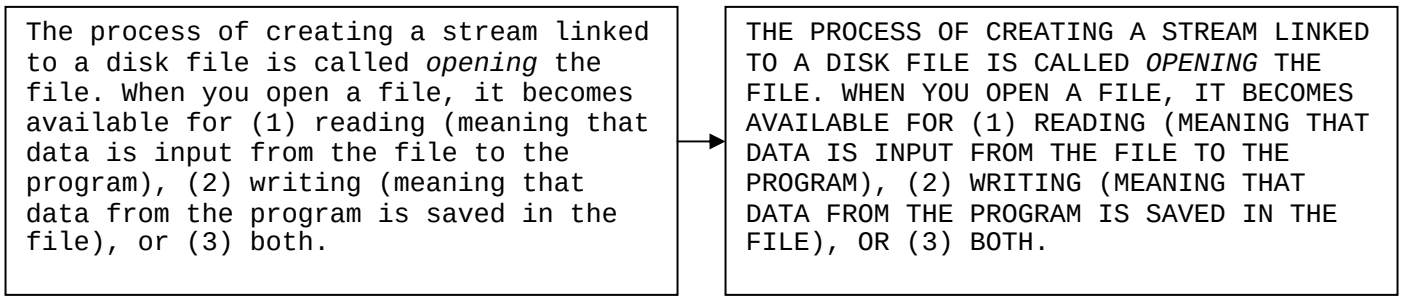
Exercícios:

a) Escreva um programa que receba do usuário 5 números inteiros e o nome do arquivo no qual eles devem ser armazenados. Em seguida, ler do arquivo estes valores armazenados copiando-os para um vetor de inteiros e imprimindo na tela.

```
#include <stdio.h>
void main()
{
   FILE *arq;
   int passo, temp, vet[5];
   char nome_arq[20];

   gets(nome_arq);
   if ( (arq = fopen(nome_arq, "w+")) == NULL) {
      fprintf(stderr, "Erro ao abrir arquivo %s.", nome_arq);
      return(1);
   }
   /* escrevendo números no arquivo */
   for (passo = 0; passo < 5; passo++) {
      scanf("%d", &temp);
      fprintf(arq, "%d", temp);
      if (passo < 4)
         fprintf(arq, "\n");
   }
   /* lendo dados do arquivo, armazenando em um vetor e imprimindo no monitor */
   rewind(arq);
   passo = 0;
   while ( !feof(arq) ) {
      fscanf(arq, "%d", &vet[passo]);
      printf("%d\n", vet[passo]);
        passo++;
   }
   fclose(arq);
}
```

b) *Arquivo caixa alta* – Escreva um programa que lê e modifica um arquivo texto lido trocando cada letra pela sua correspondente maiúscula. Exemplo: o arquivo abaixo deve ser modificado como a seguir:



c) *Tabela de notas* – Escreva um programa que gerencie uma tabela de notas. Ao executar, o usuário terá as seguintes opções: 1-Adicionar mais um aluno (RA e nota); 2-Exibir todas as notas; 3-Calcular a média. O programa deve usar um arquivo texto para armazenar e consultar as informações.

d) *Tabela de notas de acesso rápido* – Modifique o programa anterior para que use um arquivo binário e possa acessar o *i*-ésimo aluno do vetor de `structs` aluno (usando acesso direto).