ARQUIVOS

O sistema de entrada e saída do ANSI C é composto por uma série de funções, cujos protótipos estão reunidos em **`stdio.h`**. Estas funções trabalham com o conceito de "ponteiro de arquivo".

Declara-se um ponteiro de arquivo da seguinte maneira:

```
FILE *arq;
```

em que *`arq`* é um ponteiro para um arquivo. Usando este tipo de ponteiro manipula-se arquivos na linguagem C.

Abrindo um arquivo: `fopen()`

Esta é a função de abertura de arquivos.

Sintaxe: `FILE *fopen (char *nome_do_arquivo,char *modo);`

em que `nome_do_arquivo` determina qual arquivo deverá ser aberto. Este nome deve ser válido no sistema operacional que estiver sendo utilizado. O modo de abertura diz à função *`fopen()`* que tipo de uso fará do arquivo.

A tabela abaixo mostra os valores de modo válidos:

Modo	Significado
`"r"` `"rt"`	Abre um arquivo texto para leitura. O arquivo deve existir antes de ser aberto. `"t"` é opcional.
`"w"` `"wt"`	Abre um arquivo texto para gravação. Se o arquivo não existir, ele será criado. Se já existir, o conteúdo anterior será destruído.
`"a"` `"at"`	Abre um arquivo texto para gravação. Os dados serão adicionados no fim do arquivo ("*append*"), se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.
`"rb"`	Abre um arquivo binário para leitura. Igual ao modo `"r"` anterior, só que o arquivo é binário.
`"wb"`	Cria um arquivo binário para escrita, como no modo `"w"` anterior, só que o arquivo é binário.
`"ab"`	Acrescenta dados binários no fim do arquivo, como no modo `"a"` anterior, só que o arquivo é binário.
`"r+"` `"r+t"`	Abre um arquivo texto para leitura e gravação. O arquivo deve existir e pode ser modificado.
`"w+"` `"w+t"`	Cria um arquivo texto para leitura e gravação. Se o arquivo existir, o conteúdo anterior será destruído. Se não existir, será criado.
`"a+"` `"a+t"`	Abre um arquivo texto para gravação e leitura. Os dados serão adicionados no fim do arquivo se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.

`"r+b"`	Abre um arquivo binário para leitura e escrita. O mesmo que `"r+"` acima, só que o arquivo é binário.
`"w+b"`	Cria um arquivo binário para leitura e escrita. O mesmo que `"w+"` acima, só que o arquivo é binário.
`"a+b"`	Acrescenta dados ou cria uma arquivo binário para leitura e escrita. O mesmo que `"a+"` acima, só que o arquivo é binário.

Obs: É necessário verificar se o arquivo foi realmente aberto ou criado. As demais funções de manipulação de arquivo não funcionarão se a função *`fopen()`* não funcionou corretamente. Para saber se a função *`fopen()`* realmente criou ou abriu o arquivo solicitado, deve-se verificar se a variável, que representa o arquivo, possui algum valor. Se ela não possuir um valor, ou seja se for `NULL`, o arquivo não foi aberto ou criado:

Exemplo: Abrir um arquivo binário para escrita

```
/* Declaração da estrutura */
FILE *arquivo;

/* o arquivo exemplo.dat está localizado no diretório corrente */
arquivo = fopen ("exemplo.dat","wb");
if (!arquivo) {
  printf ("Erro na abertura do arquivo.");
  exit (1);
}
```

ou

```
FILE *arquivo;
if ((arquivo= fopen ("exemplo.dat","wb")) == NULL) {
  printf ("Erro na abertura do arquivo.");
  exit (1);
}
```

Toda vez que se trabalha com arquivos, há uma espécie de posição atual no arquivo. Esta é a posição de onde será lido ou escrito a próxima informação. Normalmente, num acesso sequencial a um arquivo, não tem que alterar esta posição pois quando lê-se uma informação a posição no arquivo é automaticamente atualizada. Num acesso randômico é necessário mexer nesta posição (*`fseek()`*).

Fechando um arquivo: `fclose()`

A função `fclose()` é utilizada para fechamento de arquivos.

Sintaxe: `int fclose (FILE *arquivo);`

O ponteiro arquivo passado à função *`fclose()`* determina o arquivo a ser fechado. A função retorna zero no caso de sucesso.

Fechar um arquivo faz com que qualquer informação que tenha permanecido no "*buffer*" associado ao fluxo de saída seja gravado. Mas, o que é este "*buffer*"? Quando você envia informações para serem gravadas em um arquivo, estas informações são armazenadas temporariamente em uma área de memória (o "*buffer*") em vez de serem escritos em disco imediatamente. Quando o "*buffer*" estiver cheio, seu conteúdo é escrito no disco de uma vez. A razão para se fazer isto tem a ver com a eficiência nas leituras e gravações de arquivos. Se, para cada informação que fossemos gravar, tivéssemos que posicionar a cabeça de gravação em um ponto específico do disco, apenas para gravar aquela informação, as gravações seriam muito lentas. Assim estas gravações serão efetuadas somente quando houver um volume razoável de informações a serem gravadas ou quando o arquivo for fechado.

A função *exit()* fecha todos os arquivos que um programa tiver aberto.

Gravando estruturas com a função `fwrite()`

Sintaxe:

```
unsigned fwrite (void *buffer,int numero_de_bytes,int qte, FILE *arq);
```

O primeiro argumento *buffer* é um ponteiro do tipo *void* que aponta para a localização na memória do lado a ser gravado. O argumento *numero_de_bytes* é um número inteiro que indica o tamanho do tipo de dado a ser gravado. *qte* é um número inteiro que informa a *fwrite()* quantos itens do mesmo tipo serão gravados. O argumento *arq* é um ponteiro para a estrutura FILE do arquivo que será gravado.

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>
struct LIVROS {
       char titulo[30];
       int regnum;
       double preco;
};
int main() {
  LIVROS livro;
  char numstr[81], resp;
  FILE *arquivo;
  if ((arquivo = fopen("livros.dat", "wb")) == NULL) {
     printf("\n Erro de abertura de arquivo");
     exit(1);
  }
  do {
     printf("\n\n Titulo: ");
     gets(livro.titulo);
     printf("\n Registro: ");
     gets(numstr);
     livro.regnum = atoi(numstr);
     printf("\n Preco: ");
```

```
      gets(numstr);
      livro.preco = atof(numstr);
      // obs: sizeof retorna o tamanho da estrutura livro
      fwrite(&livro, sizeof(livro), 1, arquivo);
      printf("\n Adicionar outro livro (s/n)?: ");
      do {
          resp = toupper(getch());
      } while (resp != 'S' && resp != 'N');
  } while (resp == 'S');
  fclose (arquivo);
  return 0;
}
```

Obs: A função *sizeof(x)* retorna o tamanho em bytes da variável ou do tipo de dados, dado pelo parâmetro `x`.

Lendo estruturas com a função `fread()`

Sintaxe:

```
unsigned fread (void *buffer,int numero_de_bytes, int qte, FILE *arq);
```

O primeiro argumento, *buffer*, é um ponteiro do tipo *void* que aponta para a localização na memória onde serão armazenados os dados lidos. O segundo argumento, *numero_de_bytes* é um número inteiro que indica o tamanho do tipo de dado a ser lido. O terceiro argumento (*qte*) é um número inteiro que informa a *fread()* quantos itens do mesmo tipo serão lidos a cada chamada. O argumento *arq* é um ponteiro para a estrutura `FILE` do arquivo que se deseja ler.

A função *fread()* retorna o número de itens lidos. Normalmente, este número deve ser igual ao terceiro argumento (*qte*). Se for encontrado o fim do arquivo, o número será menor, neste caso `0`.

No modo registro, dados numéricos são gravados em modo binário. Então um inteiro sempre ocupa 4 bytes, um número em ponto flutuante 4 bytes, etc. Como funções de leitura e gravação no modo registro usam formato binário, elas são mais eficientes para guardar dados numéricos do que funções que usam formato texto.

Exemplo 1:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  struct LIVROS {
      char titulo[30];
      int regnum;
      double preco;
  } livro;
  FILE *arquivo;
  if ((arquivo = fopen("livros.dat", "rb")) == NULL) {
    printf ("\n Erro de abertura de arquivo");
```

```
    exit(1);
  }
  while (fread (&livro, sizeof(livro), 1, arquivo) == 1) {
    printf ("\n Titulo: %s ", livro.titulo);
    printf ("\n Registro: %d", livro.regnum);
    printf ("\n Preco: %.2f\n",livro.preco);
  }
  fclose (arquivo);
  return 0;
}
```

Exemplo 2: Lendo e escrevendo vetores em um arquivo binário

```
#include <stdio.h>
#include <stdlib.h>
int main () {
  int i, v[20] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19};
  int v2[20];
  FILE *arq; //ponteiro para arquivo
  int n = 20;

  // Gravação do arquivo
  //criação do arquivo para gravação
  arq = fopen ("vetor.dat", "wb");
  //gravação de "n" na 1ª posição do arquivo
  fwrite (&n, sizeof(int), 1, arq);
  //gravação do vetor
  fwrite (v, sizeof(int), n, arq);
  //fechamento do arquivo
  fclose (arq);

  // Leitura do arquivo
  // abertura novamente do arquivo para leitura
  arq = fopen ("vetor.dat", "rb");
  // leitura da 1ª posição que contem a quantidade de elementos do vetor
  fread (&n,sizeof(int),1,arq);
  // impressão da quantidade na tela
  printf ("%d\n",n);
  // leitura de cada posição do vetor e armazeno em v2 na tela
  fread (v2,sizeof(int),n,arq);
  // impressão de cada posição de v2
  for (i=0; i<n; i++)
    printf("%d ",v2[i]);
  //fechamento do arquivo
  fclose (arq);
  return 0;
}
```

Função `rewind ()`

A função `rewind()` retorna a posição corrente do arquivo para o início.

Sintaxe: `void rewind (FILE *arq);`

Exemplo 1: Lendo e escrevendo vetores em um arquivo binário

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  int i, v[20] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19};
  int v2[20];
  FILE *arq; // ponteiro para arquivo
  int n = 20;
  // criação do arquivo para gravação e leitura
  arq = fopen ("vetor2.dat", "w+b");
  // gravação do "n" na primeira posição do arquivo
  fwrite (&n, sizeof(int), 1, arq);
  // gravação do vetor todo de uma vez
  fwrite (&v, sizeof(v), 1, arq);
  // retorno do ponteiro para a primeira posição do arquivo
  rewind (arq);
  // leitura da primeira posição com a quantidade de elementos do vetor
  fread (&n, sizeof(int),1,arq);
  // impressão da quantidade na tela
  printf ("%d\n",n);
  fread (&v2, sizeof(v), 1, arq);
  // leitura do vetor todo e armazeno em v2
  for (i=0; i<n; i++)
    printf("%d ",v2[i]);
  return 0;
}
```

Exemplo 2: Lendo e escrevendo matrizes em um arquivo binário

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  int i, j, m[4][5] = {{0,1,2,3,4},
                       {5,6,7,8,9},
                       {10,11,12,13,14},
                       {15,16,17,18,19}};
  int m2[4][5];
  FILE *arq; // ponteiro para arquivo
  int l = 4;
  int c = 5;
  // criação do arquivo para gravação e leitura
  arq = fopen ("matriz.dat", "w+b");
  // gravação de l na primeira posição do arquivo
  fwrite (&l, sizeof(int), 1, arq);
  // gravação de c na segunda posição do arquivo
  fwrite (&c, sizeof(int), 1, arq);
  // gravação da matriz toda de uma vez
  fwrite (m, sizeof(m), 1, arq);
  // retorno do ponteiro para a primeira posição do arquivo
  rewind (arq);
```

```
  // leitura de "l" da primeira posição do arquivo
  fread (&l, sizeof(int), 1, arq);
  // leitura de "c" da segunda posição do arquivo
  fread (&c,sizeof(int),1,arq);
  // impressão do cabeçalho da matriz na tela
  printf ("Matriz m[%d][%d] =\n",l,c);
  // leitura da matriz toda e armazeno em m2
  fread (m2, sizeof(m), 1, arq);
  for (i=0; i<l; i++) {
    for (j=0;j<c;j++)
      printf("%3d ",m2[i][j]); //imprime cada posição de m2 na tela
      printf("\n");
  }
  return 0;
}
```

Exemplo 3: Escrita de vetor de estrutura

```
/* ......... */
typedef struct {
            char nome[20];
            float media;
        } tipo_aluno;

tipo_aluno a;
tipo_aluno vet[100];

fwrite (&a, sizeof(tipo_aluno), 1, arq); // escrita de um registro;
fwrite (vet, sizeof(tipo_aluno), 100, arq); // escrita do vetor todo
fwrite (vet, sizeof(vet), 1, arq); // escrita do vetor todo;
```

Exemplo 4: Leitura de vetor de estrutura

```
/* ......... */
typedef struct {
            char nome[20];
            float media;
        } tipo_aluno;

tipo_aluno a;
tipo_aluno vet[100];

fread (&a, sizeof(tipo_aluno), 1, arq); // leitura de um registro;
fread (vet, sizeof(tipo_aluno), 100, arq); // leitura do vetor todo
fread (vet, sizeof(vet), 1, arq); // leitura do vetor todo
```

Função `fseek()`

Utiliza-se a função *`fseek()`* para fazer acessos randômicos em arquivos. Esta função move a posição corrente de leitura ou escrita no arquivo de um valor especificado, a partir de um ponto especificado.

Sintaxe: `int fseek (FILE *arq, long numbytes, int origem);`

O parâmetro *origem* determina a partir de onde os *numbytes* de movimentação serão contados. Os valores possíveis são definidos por macros em `stdio.h` e são:

Nome	Valor	Significado
`SEEK_SET`	0	Início do arquivo
`SEEK_CUR`	1	Posição corrente no arquivo
`SEEK_END`	2	Fim do arquivo

Tendo-se definido a partir de onde irá se contar, *numbytes* determina quantos bytes de deslocamento serão dados na posição atual.

Exemplos:

```
// posiciona-se no final do arquivo
fseek (arquivo, 0, SEEK_END);
// gravação
fwrite (&dado, sizeof(dado), 1, arquivo);

// posiciona-se no início do arquivo (idêntico ao rewind())
fseek (arquivo, 0, SEEK_SET);
// leitura do registro
fread (&dado, sizeof(dado), 1, arquivo);

// posiciona-se no segundo registro:
fseek (arquivo, 2*sizeof(dado), SEEK_SET);
// leitura do registro
fread (&dado, sizeof(dado), 1, arquivo);
```

Função `remove()`

A função `remove()` apaga um determinado arquivo. Se o arquivo for excluído com sucesso, a função retorna o valor 0, caso contrário será devolvido outro valor. É importante fechar o arquivo antes de removê-lo.

Sintaxe: `int remove (char *nome_do_arquivo);`

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main () {
  // nome do arquivo a ser excluído
  char *arquivo = "c:\\testes.txt";
  // exclusão do arquivo
  if (remove(arquivo) == 0)
    printf ("Arquivo foi excluido com sucesso.");
```

```
  else
    printf ("Nao foi possivel excluir o arquivo.");
  return 0;
}
```

Função `rename()`

A função `rename()` troca o nome de um arquivo.

Sintaxe: `rename (char *nome_atual, char *nome_novo);`

em que `nome_atual` indica o nome físico atual do arquivo, podendo ser incluído o caminho (*path*) e, `nome_novo` indica o novo nome físico que se pretende dar ao arquivo, podendo ser incluído o caminho (*path*).

Fim de Arquivo (EOF)

O `EOF` (*End Of File*) indica que o ponteiro está posicionado no fim do arquivo. É importante entender que `EOF` é um valor enviado ao programa pelo sistema operacional e definido no arquivo `stdio.h`. Esse valor pode ser diferente para diferentes sistemas operacionais, por isso, deve-se usar sempre a constante `EOF`.

Exemplo: Leitura de um arquivo pode-se fazer:

```
#include <stdio.h>
#include <stdlib.h>
int main () {
  FILE *arq;
  char caracter;
  arq = fopen ("teste.txt", "r");
  do {
    caracter = getc(arq);
    printf ("%c",caracter);
  } while (caracter != EOF);
  return 0;
}
```

Função de Fim de Arquivo: `feof()`

Na manipulação dos dados binários, na leitura de um arquivo binário, um valor inteiro igual a `EOF` pode ser lido por engano. Isso pode fazer com que fosse indicado o fim de arquivo antes deste ter chegado. Para resolver este problema a linguagem C inclui a função *`feof()`* que determina quando o final de um arquivo foi atingido.

Sintaxe: `feof (FILE *arquivo);`

onde *`arquivo`* é um ponteiro de arquivo. Esta função faz parte de `stdio.h` e devolve verdadeiro caso o final de arquivo seja atingido; caso contrário ela devolve 0.

Exemplo: Leitura de um arquivo pode-se fazer:

```
#include <stdio.h>
#include <stdlib.h>
int main () {
  FILE *arq;
  char caracter;
  arq = fopen ("teste.txt", "r");
  while (!feof(arq)) {
    caracter = getc(arq);
    printf ("%c",caracter);
  };
  return 0;
}
```

Escrevendo e lendo caractere

Para a escrita e leitura de caractere a caractere em arquivo utiliza-se duas funções básicas: *putc()* e *getc()*.

Escrevendo no arquivo com a função `putc()` e `fputc()`

A função `putc()`, equivalente a `fputc()`, escreve um caractere em um arquivo que foi previamente aberto para escrita.

Sintaxe: `int putc (char* ch, FILE *arq);`

Em que `arq` é um ponteiro de arquivo devolvido por `fopen()` e `ch` é o caractere a ser escrito.

Exemplo: Gravação de uma *string*, caractere por caractere em um arquivo texto.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  FILE *arquivo;
  char texto[100];
  int i;
  /* Arquivo ASCII, para escrita */
  arquivo = fopen ("arquivo.txt","w");
  if (!arquivo) {
    printf( "Erro na abertura do arquivo");
    exit(1);
  }
  printf ("Digite uma string a ser gravada no arquivo:");
  gets (texto);
  for (i=0; texto[i]; i++)
    putc (texto[i], arquivo); // Grava a string, caractere a caractere
  fclose(arquivo);
```

```
  return 0;
}
```

Lendo o arquivo com a função `getc()` e `fgetc()`

A função `getc()`, equivalente a `fgetc()`, lê um caractere em um arquivo que foi previamente aberto no modo leitura.

Sintaxe: `int getc (FILE *arq);`

Em que `arq` é um ponteiro de arquivo devolvido por `fopen()`.

Exemplo 1: Leitura de um arquivo texto, caractere por caractere, que conta o número de caracteres lido.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  FILE *arquivo;
  int cont = 0;
  char ch;
  if ((arquivo = fopen (“arquivo.txt”,”r”)) == NULL) {
    printf (“\n Arquivo não pode ser aberto.”);
    exit (0);
  }
  while ((ch = getc(arquivo)) != EOF)
    cont++;
  fclose (arquivo);
  printf (“\n O arquivo contém %d caracteres.”,cont);
  return 0;
}
```

Exemplo 2: Copia de arquivo

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  FILE *original,*copia;
  char caracter, nome[13], nome_novo[13];
  printf("\nDigite o nome do arquivo: ");
  gets (nome);
  printf("\nDigite o nome do arquivo copia: ");
  gets (nome_novo);
  if ((original = fopen(nome,"r")) == NULL) {
    printf("\nErro ao abrir o arquivo original.\n\n");
    exit(1);
  }
  if ((copia = fopen(nome_novo,"w")) == NULL) {
    printf("\nErro ao abrir o arquivo cópia.\n\n");
    exit(1);
  }
```

```
  while(!feof(original)) {
    caracter = getc (original);
    if (!feof(original))
      putc (caracter,copia);
  }
  fclose(original);
  fclose(copia);
  printf("\n%s copiado com sucesso com o nome de %s.\n\n",nome,nome_novo);
  return 0;
}
```

O programa acima é um exemplo de utilização do `getc()` e do `putc()` destinado apenas para fins didáticos, para realizar cópias de arquivos, deve-se usar `fread()` e `fwrite()` transferindo uma grande quantidade de dados por vez.

Gravando e Lendo um Arquivo Linha a Linha

Ler e gravar cadeias de caracteres (*string*) em arquivo é mais fácil que ler e gravar caractere a caractere. Para gravar uma *string* utiliza-se a função `fputs()` e para ler utiliza-se a função `fgets()`.

Gravando arquivo linha a linha

A função de gravação de cadeias de caracteres em arquivo tem a seguinte sintaxe:

Sintaxe: `char *fputs (char *str, FILE *arq);`

onde `arq` é o ponteiro do arquivo e str é o conjunto de caracteres que será gravado no arquivo.

Exemplo: Escreve cadeias de caracteres em um arquivo.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
  FILE *arquivo;
  char texto[81];
  if ((arquivo = fopen("arquivo.txt","w")) == NULL) {
    printf ("\n Arquivo não pode ser aberto");
    exit(1);
  }
  while (strlen(gets(texto))>0) {
    fputs (texto,arquivo); // grava o conjunto de caracteres
    putc ('\n',arquivo); // grava caractere de mudança de linha
  }
  fclose (arquivo);
  return 0;
}
```

Lendo arquivo linha a linha

Para se ler uma *string* num arquivo podemos usar `fgets()`:

Sintaxe: `char *fgets (char *str, int tamanho,FILE *arq);`

A função `fgets()` possui três argumentos: a *string* a ser lida (`str`), o limite máximo de caracteres a serem lidos (`tamanho`) e o ponteiro para `FILE`, que está associado ao arquivo de onde a *string* será lida.

A função lê a *string* até que um caractere de nova linha seja lido ou `tamanho-1` caracteres tenham sido lidos. Se o caractere de nova linha ('\n') for lido, ele fará parte da *string*, o que não acontecia com `gets()`. A *string* resultante sempre terminará com '\0' (por isto somente `tamanho-1` caracteres, no máximo, serão lidos).

A função `fgets()` é semelhante à função `gets()`, porém, além dela poder fazer a leitura a partir de um arquivo de dados e incluir o caractere de nova linha na *string*, ela ainda especifica o tamanho máximo da *string* de entrada. A função `gets()` não tem este controle, o que pode acarretar erros de "estouro de *buffer*".

Exemplo: Escreve cadeias de caracteres em um arquivo.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  FILE *arquivo;
  char texto[81];
  if ((arquivo = fopen("arquivo.txt","r")) == NULL) {
    printf ("\n Arquivo não pode ser aberto");
    exit(1);
  }
  while (fgets(texto,80,arquivo) != NULL)
    printf ("%s", texto);
  fclose (arquivo);
  return 0;
}
```

Gravando e lendo um arquivo de forma formatada

As funções de gravação e leitura formatada são: `fprintf()` e `fscanf()`.

Gravando um arquivo de forma formatada

A função `fprintf()` escreve os valores formatados de várias maneiras; todas as possibilidades de formato de `printf()` operam com `fprintf()`.

Sintaxe: `fprintf (arquivo, "formatos", lista de variáveis);`

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
  FILE *arquivo;
  char titulo[30];
  int regnum;
  float preco;
  if ((arquivo = fopen("dados.txt","w")) == NULL) {
    printf ("\n Arquivo não pode ser aberto.");
    exit(1);
  }
  do {
    printf ("\n Digite titulo, registro, preco: ");
    scanf ("%s %d %f", titulo,&regnum,&preco);
    fprintf (arquivo,"\n%s %d %f", titulo, regnum, preco);
  } while (strlen(titulo) > 1);
  fclose (arquivo);
  return 0;
}
```

Obs: É possível especificar o número de casas (bytes) que será ocupada pelos números que serão gravados.

Exemplo: `fprintf (arquivo, "%5d", regnum);`

Lendo arquivo de forma formatada

Para ler um arquivo de forma formatada utiliza-se a função `fscanf()` que é similar a função `scanf()` e aceita todos os seus formatos.

Sintaxe: `fscanf (arquivo, "formatos", lista de variáveis);`

Exemplo: Lê os dados formatados e termina quando encontra o fim de arquivo

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  FILE *arquivo;
  char titulo[30];
  int regnum;
  float preco;
  if ((arquivo = fopen ("texto.txt","r")) == NULL) {
    printf ("\n Arquivo não pode ser aberto.");
    exit(1);
  }
```

```
  while (fscanf (arquivo, "%s %d %f", titulo,&regnum,&preco) != EOF)
    printf ("%30s %5d %.2f\n", titulo,regnum, preco);
  fclose (arquivo);
  return 0;
}
```

Obs: É importante saber como os dados são guardados no disco por `fprintf()`. Texto e caracteres são guardados um caractere por byte, como acontece na memória. Os números não são guardados como na memória (4 bytes para inteiro, 4 para float etc.) e sim como cadeias de caracteres. Então, o inteiro 222 é armazenado em 4 bytes na memória mas requer 3 bytes em arquivo em disco, um para cada caractere ´2´. O número 57.89 requer 4 bytes quando guardado na memória, mas cinco em disco. Portanto, arquivos com maior número de dígitos ocupam um espaço substancialmente maior em disco que na memória. Assim, se um grande número de dados numéricos devem ser armazenados em disco, o uso de formato texto torna-se ineficiente. A solução é usar funções que guardam números em formato binário.

Funções `ferror()` e `perror()`

Sintaxe: `int ferror (FILE *arquivo);`

A função retorna zero, se nenhum erro ocorreu e um número diferente de zero se algum erro ocorreu durante o acesso ao arquivo.

`ferror()` se torna muito útil quando queremos verificar se cada acesso a um arquivo teve sucesso, de modo que consigamos garantir a integridade dos nossos dados. Na maioria dos casos, se um arquivo pode ser aberto, ele pode ser lido ou gravado. Porém, existem situações em que isto não ocorre. Por exemplo, pode acabar o espaço em disco enquanto gravamos, ou o disco pode estar com problemas e não conseguimos ler, etc.

Uma função que pode ser usada em conjunto com *`ferror()`* é a função *`perror()`* **(print error)**, cujo argumento é uma *string* que normalmente indica em que parte do programa o problema ocorreu.

No exemplo a seguir, fazemos uso de `ferror()`, `perror()` e `fputs()`.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
  FILE *arq;
  char string[100];
  if((arq = fopen("arquivo.txt","w")) == NULL) {
    printf ("\nNao consigo abrir o arquivo ! ");
    exit(1);
  }
  do {
    printf ("\nDigite uma nova string. Para terminar, digite <enter>: ");
    gets (string);
    fputs (string, arq);
```

```
    putc ('\n', arq);
    if (ferror(arq)) {
      perror ("Erro na gravacao");
      fclose (arq);
      exit(1);
    }
  } while (strlen(string) > 0);
  fclose(arq);
  return 0;
}
```

Função `fflush()`

Sintaxe: `int fflush (FILE *arquivo);`

A função `fflush()` esvazia o buffer do arquivo referenciado pelo ponteiro passado como argumento, se o arquivo passado for um arquivo de saída o conteúdo do buffer é gravado no mesmo. Se a função for chamada sem parâmetros, os buffers de todos os arquivos abertos para saída serão descarregados. A função devolve 0 para indicar sucesso, caso contrário devolve `EOF`.

Arquivos Padrões

Arquivos definidos automaticamente abertos quando o programa é iniciado. Esses arquivos são:

- `stdin` dispositivo de entrada padrão (teclado)
- `stdout` dispositivo de saída padrão (vídeo)
- `stderr` dispositivo de saída padrão de erros (vídeo)
- `stdaux` dispositivo auxiliar padrão
- `stdprn` impressora padrão

Esses arquivos estão declarados no **`stdio.h`**.

Alguns compiladores indicam uma falha de segurança na utilização do `gets()` para ler uma cadeia de caracteres. Isto se deve à possibilidade do usuário digitar mais dados do que o vetor de caracteres que receberá a cadeia suporta. Veja o exemplo:

```
char nome[10];
gets(nome);
```

No exemplo, se o usuário digitar um nome com mais que 9 caracteres, uma área de memória não disponível para a variável nome será invadida. Para evitar, substitui-se a função `gets()` pelo `fgets()`, como no exemplo:

```
char nome[10];
fgets(nome, 10, stdin);
```

No exemplo, se o usuário digitar um nome com mais que 10 caracteres ele será truncado.

Outra aplicação interessante dos arquivos padrões é a possibilidade de enviar textos para uma impressora, veja o exemplo:

```
fprintf(stdprn, "Texto na impressora\n\f");
```