11. - ARQUIVOS

11.1 - Características básicas

- Um arquivo é uma coleção de *bytes* referenciados por um nome único.
- Arquivos são usados para armazenamento permanente de uma grande quantidade de dados
- Um arquivo pode ser uma impressora, um teclado, um terminal ou um arquivo em disco; para este contexto, vamos considerar os arquivos em disco.
- Arquivos em disco podem permitir acesso aleatório, o que não ocorre por exemplo com um teclado.
- Uma *stream* é o que vai ser colocado no arquivo, existem dois tipos de *strems*: binários e de texto.
- Dependendo do tipo da *stream*, os arquivos podem ser abertos de dois modos:
    - Modo texto: para conter *streams* do tipo texto.
    - Modo binário: para conter *streams* binárias.
- Modo texto:
    - Seqüência de caracteres agrupados em linhas.
    - As linhas são separadas por um caracter de nova linha.
    - Existe também um caracter indicando final de arquivo (EOF).
    - Números são convertidos para seqüências de caracteres (cada dígito ocupa 1 *byte*).

- Modo binário:

  Elementos são guardados da mesma maneira que estavam na memória, ou seja: 2 *bytes* para inteiros, 4 para *float*, 1 para caracter, etc.

- Para um arquivo ser utilizado por um programa, é necessário que informações específicas sobre o arquivo estejam disponíveis.
- Essas informações são guardadas em uma estrutura pré-definida chamada FILE, ou seja, um arquivo possui o tipo FILE.
- A maior parte das funções e macros para manipulaçào de arquivos estão em <stdio.h>

11.2 - Criação de arquivos

- Três arquivos e seus respectivos streams são abertos/criados automaticamente quando um programa inicia sua execução: a entrada padrão, a saída padrão e a saída padrão de erros.

- A menos que redirecionados, estão normalmente associados ao terminal, isto é , ao teclado, ao monitor e monitor respectivamente. A cada um corresponde uma constante, definida em stdio.h que o identifica:

stdin 0
stdout 1
stderr 2

- Para você criar seu próprio arquivo é necessário criar uma variável do tipo FILE para guardar as informações sobre o mesmo.

  Declaração: FILE *fp;

- Um arquivo possui “dois nomes”:

  - nome interno: variável do programa. Ex.: fp
  - nome externo: arquivo em disco onde os dados são lidos e armazenados. Ex.: saida.dat

- Cada arquivo tem um <u>modo</u>: cadeia de caracteres que indica a finalidade do arquivo

  “**r**” : leitura (o arquivo já existe, abre só para consulta)

  “**w**” : gravação (o arquivo ainda não existe)

  “**a**” : anexação (o arquivo já existe e a gravação é feita no final do mesmo)

- Dois modificadores podem ser adicionados ao modo:

  + : arquivo já existe e se quer atualizá-lo

  b : dados binários

  Variações: r+, w+, a+, rb, wb, ab, rb+, wb+, ab+

Exemplos:

`r` : abre um arquivo existente para leitura (*read*).

`w` : abre um arquivo para escrita. Se o arquivo já existe, seu conteúdo é descartado. Senão, um novo arquivo vazio é criado.

`a` : abre um arquivo para concatenação . Se o arquivo já existe, seu conteúdo é preservado e as escritas serão concatenadas no final do arquivo. Senão, um novo arquivo vazio é criado.

`r+` : abre um arquivo existente para leitura e escrita. O conteúdo anterior do arquivo é preservado e o ponteiro é posicionado no início do arquivo.

`w+` : abre um arquivo para leitura e escrita. Se o arquivo já existe, seu conteúdo é descartado. Senão, um novo arquivo vazio é criado.

`a+` : abre um arquivo para escrita e concatenação. Se o arquivo já existe, seu conteúdo é preservado e as escritas serão concatenadas no final do arquivo. Senão, um novo arquivo vazio é criado. O ponteiro de leitura é posicionado no início do arquivo, enquanto as escritas são efetuadas no seu final.

- Antes de um arquivo ser utilizado, ele deve ser colocado “a disposição”, ou ainda “ser aberto”.
- *fopen*( ): função que torna o arquivo disponível para leitura ou gravação, isto implica em:
  - Preencher a variável do tipo FILE com as informações sobre o arquivo.
  - Retornar um ponteiro para a localização de memória do inicio do arquivo (indicador de posição).

Sintaxe:

< nome interno > = fopen (< nome externo > , < modo >);

Exemplo:

```
FILE  *fp;                              /* declaração  de fp*/
fp      =     fopen( “saída.dat”,   “w”);

↓                       ↓              ↓

nome interno        nome externo    modo
```

Obs.: Se houver algum problema como por exemplo, tentar abrir um arquivo que não existe ou que não tem permissão para leitura, então *fopen( )* retorna a constante NULL

(“ if (fp ! = NULL) está tudo bem! “)

- Depois de um arquivo ser utilizado ele deve ser “fechado”.
- *fclose*( ): função que “fecha” o arquivo, isto implica em:
  - Finalizar a gravação do arquivo (esvaziar o buffer)
  - Liberar o apontador (acesso ao arquivo).

  Sintaxe:

  fclose (< nome interno >);

  Exemplo:

  fclose(fp);

- *exit*( ): função que também fecha arquivos. Difere de *fclose*( ) no seguinte:
  - Fecha todos os arquivos abertos.
  - Termina o programa e devolve o controle ao S.O..

11.2 – Algumas funções e macros

- EOF: constante (geralmente igual a -1), indica final de arquivo.
- OPEN_ MAX: constante inteira que determina o número máximo de arquivos que podem ser abertos ao mesmo tempo em um programa. Esse valor geralmente é maior que oito, mas depende do compilador e do sistema operacional.
- *feof*( ): função que devolve um valor verdadeiro (inteiro diferente de zero) quando o EOF é lido por um comando de leitura executado anteriormente.

  Sintaxe:

  feof(< nome interno >);

  Exemplo:

  ```
  if  ( ! ( feof(fp) ) )
      printf("Não  chegou ao final do arquivo ");
  ```

- *rewind*( ): função que reposiciona o indicador de posição para o inicio do arquivo. Útil para consultar um arquivo récem-criado.

  Sintaxe:

  rewind(< nome interno >);

- *ferror*( ): determina se uma operação sobre um arquivo produziu algum erro. Se ocorrer erro, a função retorna verdadeiro, senão retoran falso.

  Sintaxe:

  ferror(< nome interno >);

  Ex.:
  ```
  if  ( ! ferror(fp) )
          printf( "Esta tudo OK com o arquivo! ");
  ```

- *remove*( ): apaga o arquivo especificado. Devolve zero (falso) caso a operação for bem sucedida e verdadeiro caso contrário.

  Sintaxe: *remove*(< nome interno >);

- *fflush*( ): escreve o conteúdo dos dados do buffer para o arquivo especificado. Devolve 0 se a operação for bem sucedida, caso contrário, devolve EOF.

  Sintaxe: *fflush*(< nome interno >);

- *fseek*( ): modifica o indicador de posição de arquivos

  Sintaxe:

  fseek(<nome interno>, <número de bytes>, <origem>);

  - O indicador de posição do arquivo "<nome interno>" será movido segundo o "<número de bytes>", a partir da origem.

  - A origem é definida através de uma Macro.

origem	**valor**	**nome da Macro**
inicio do arquivo	0	SEEK_SET
posição corrente	1	SEEK_CUR
final do arquivo	2	SEEK_END

  - *fseek*( ) devolve zero quando bem sucedida e algo diferente de zero, caso contrário.

11.3 - Funções para manipular dados de entrada e saída em arquivos

As funções para manipular dados de entrada e saída em arquivos são muito parecidas com as funções para entrada e saída padrão.

11.3.1 - Leitura e escrita de caracteres

- *getc*( ): função que lê (retorna) um caracter do arquivo (nome interno). Essa função devolve EOF quando chega no final do arquivo.

  Sintaxe :

  getc (< nome interno >)

  Ex. 1: *getc* (fp);

- *putc*( ): função que insere um caracter em um arquivo. Se essa operação for bem sucedida, a função devolve o caracter inserido, caso contrário devolve EOF.

  Sintaxe:

  putc (< car > , < nome interno >);

Ex. 2: Copia um arquivo em outro.

```
while ( ! feof(fp1) )
{
   ch = getc(fp1);
    putc(ch, fp2);
 }
```

Ex. 3: Programa para ler e duplicar um arquivo de caracteres.

```
#include <stdio.h>
main( )
{
    char c;
    FILE   *arqIn,   *arqOut;
    arqIn = fopen (“entrada.dat”, “r”);        /*  r : leitura  */
    arqOut = fopen (“saída.dat”, “w”);         /*  w: escrita  */
    while  ( ( c = getc(arqIn) )  !=  EOF )
          putc ( c, arqOut );
     fclose (arqIn);
     fclose (arqOut);
}
```

11.3.2 - Leitura e escrita de strings em arquivos

- fgets( ):
  - Lê uma linha por vez de um arquivo texto.
  - Caracteres são lidos até que:
    - apareça ‘\n’ (final de linha)
    - apareça EOF (final de arquivo)
    - o limite especificado seja atingido.
  - Sintaxe:

    fgets (<string> , < n_max > , < arquivo>)
    - < string> ⇒ vetor no qual a linha será armazenada
    - < n_max > ⇒ no. máx. de caracteres a serem lidos
    - < arquivo > ⇒ nome interno do arquivo de leitura
  - n_max -1 caracteres são lidos.

- fgets retorna NULL quando não encontra caracteres (ou encontrar EOF).

- O '\n' é incluído em string. Diferente do gets! O valor n_max deve considerar isto. Por fim é adiconado um '\0'.

Ex1:

```
main( )
 {
    FILE  *arqIn;
    char frase[81];
    arqIn = fopen ( “entrada.txt “, “r”);
    while (fgets (frase, 80, arqIn) ! = NULL)
       printf ( “% s”, frase);
     fclose (arqIn);
 }
```

- A função gets não limita o número de caracteres, o que pode acarretar erros de "estouro de buffer", quando o usuário digita mais caracteres que o limite do array especificado. Portanto, levando em conta que o ponteiro fp pode ser substitído por stdin, uma alternativa ao uso de gets é usar a seguinte construção:

```
fgets (str, tamanho, stdin);
```

onde str é a string que se está lendo e tamanho deve ser igual ao tamanho alocado para a string subtraído de 1, por causa do '\0'.

Mas lembre-se que o '\n' irá fazer parte da string!

Ex2: Lê uma linha do teclado e imprime em arquivo, finalizar com <ctrl D> e enter.

```
#include <stdio.h>
main()
{   FILE *fp;
  char fnome[13];
  char linha[81];

  printf("Entre um nome de arquivo: ");
```

```
  scanf("%s", fnome);

  fp = fopen(fnome, "w" );    /* abre arquivo. Conteúdo anterior é
                                 perdido.*/
   if (fp == NULL)
      printf("Erro ao abrir %s\n", fnome);
  else   {
      printf("Arquivo aberto com sucesso");

     /* lê linha do teclado, armazena em uma string linha e
        salva em arquivo */
      while(fgets(linha, 80, stdin) != NULL)
        fputs(linha, fp);
      fclose(fp); /* fecha arquivo */
   }
}
```

- fputs( ) : Escreve uma string em um arquivo.
  - Sintaxe: fputs (< string >, <arquivo>);

Ex1:

Seja o seguinte trecho de programa:

```
.
.
while (fgets (frase, 80, arqin) ! = NULL)
 {
       fputs (frase, arqOut);
       fputs ( “\n “, arqOut);
 }
```

  - fputs não insere o caracter de nova linha (\n) após escrever a string (puts coloca).

11.3.3 - Entrada e saída formatadas

- *fscanf*( ): similar a *scanf*( ), porém o primeiro parâmetro é sempre o nome interno do arquivo. Essa função retorna EOF quando chegar ao final do arquivo, ou o número de itens lidos.

  Sintaxe:

  fscanf(< nome interno >, <formato dos elementos>, < lista de elementos >);

- *fprintf*( ) : usado da mesma maneira que *printf*( ), porém o primeiro parâmetro é sempre o nome interno do arquivo.

  Sintaxe:

  fprintf(< nome interno >, <formato dos elementos>, < lista de elementos >);

Exemplo: Programa para ler um arquivo contendo nome, código e preço de livros e duplicar o mesmo, formatando os dados de saída.

```
#include <stdio.h>
main( )
{
    char titulo[30];
    int codigo; double preço;
    FILE  *arqIn,  *arqOut;
    arqIn = fopen (“entrada.txt”, “r“);
    arqOut = fopen (“saida.txt” , “w”);
    while (fscanf (arqIn, “% s   % d  % lf “, titulo, &codigo,
        &preço) != EOF)
            /* “título” é vetor (sem &) */
            fprintf (arqout , “%s  % 03d  % .2f \n “, título,
                    código, preço);
    fclose (arqIn);
    fclose (arqOut);
 }
```

Para o programa anterior:

Helena 102 12.503	Helena 102 12.50
Iracema 321 13.251	Iracema 321 13.25
Macunaima 56 9.3	Macunaima 056 9.30
X 0 0	X 000 0.00
entrada.txt	saida.txt

11.4 Arquivos Binários

São arquivos cujo o conteúdo não representa um texto e são usados para armazenar dados cujo formato necessita ser conhecido a priori, interpretados como sequências de bytes.

Leitura e escrita:

unsigned fread (void *memoria, int tamanho, int quantidade, FILE *arquivo)
void *memória : apontador para a região na qual serão armazenados os dados lidos.
int tamanho: é o número de bytes da unidade a ser lida.
int quantidade: indica quantas unidades devem ser lidas.
arquivo: arquivo de onde deve ser lida ou escrita uma unidade.

Retorna o número de unidades lidas, porém poderá ser menor que int quantidade se for encontrado o fim do arquivo ou se algum erro ocorrer.

unsigned fwrite(void *memoria, int tamanho, int quantidade, FILE *arquivo);
Retorna o número de itens escritos e este valor será igual a int quantidade, exceto se houver algum erro.

Exemplo:

```
#include <stdio.h>
int main()
{
   FILE *pf;
   float pi = 3.1415;
   float pi2;
   if((pf=fopen("arquivo.bin","wb"))== NULL)
    /* Abre arquivo binario para escrita */
   {
     printf("Erro na abertura do arquivo");
     exit(1);    // finaliza programa
   }
   if(fwrite(&pi,sizeof(float),1,pf)!=1)
     /* Escreve a variavel pi */
     printf("Erro na escrita do arquivo");
   fclose(pf); /* Fecha o arquivo */

   if((pf =fopen("arquivo.bin","rb"))==NULL)
   /* Abre arquivo novamente para leitura*/
   {
     printf("Erro na abertura do arquivo");
     exit(1);
   }
   if(fread(&pi2, sizeof(float), 1,pf)!=1)
   /*Atribui a pi2 o valor armazenado no
    arquivo */
     printf("Erro na leitura do arquivo");

   printf("\nValor em arquivo.bin:%f", pi2);
 fclose(pf);
}
```