Introdução à Computação II

Arquivos

Profa.: *Adriane Beatriz de Souza Serapião*

adriane@rc.unesp.br

Arquivos

- **Um arquivo é um conjunto de registros armazenados em um dispositivo de memória (disco, fita, disquete, etc.). Portanto, uma estrutura de dados, no qual cada registro não ocupa uma posição fixa dentro da estrutura, não possuindo tamanho preestabelecido. Pode ser criado, consultado, processado e removido por algoritmos distintos.**

Arquivos

- **Arquivos são importantes por três motivos:**
  - **como via de comunicação: a única maneira de um processo se comunicar com o seu ambiente externo é por meio de arquivos;**
  - **como memória permanente: um processo normalmente tem vida curta: um programa é carregado na memória, executado e tão logo termina, a memória é usada por outro programa. Se o programa não altera um arquivo durante sua execução, seus resultados se perdem tão logo ele termina;**
  - **como memória complementar: pode-se armazenar um volume muito maior de dados num arquivo do que na memória principal.**

Arquivos

- **As operações básicas que podem ser feitas em um arquivo através de um algoritmo são:**
  - **obtenção de um registro (Leitura);**
  - **inserção de um novo registro (Gravação);**
  - **modificação de um registro (Alteração);**
  - **exclusão de um registro (Exclusão).**

- **Os arquivos podem ser organizados de duas formas, quanto à sua criação ou acesso. São elas:**
  - **Sequencial, onde os registros são obtidos ou inseridos no arquivo um após o outro;**
  - **Direta, onde o acesso ao registro é feito em ordem aleatória.**

Arquivos

- **Um arquivo é interpretado pela Linguagem C como qualquer dispositivo, desde um arquivo em disco até um terminal ou uma impressora.**
- **Tipos de arquivos:**
  - **de dispositivo (I/O) – teclado (CON), tela (CON), porta de impressora (LPT, LST, PRN), porta de comunicação (COM), modem.**
  - **de disco – ficam em unidades de disco (armazenamento) do computador.**

Conceito

- **Arquivos não são acessados diretamente.**
- **Para utilizá-los, é preciso associá-los a uma *stream* e, então, manipular a *stream*.**
- **Um arquivo é associado à uma stream através de uma operação de abertura.**
- **A dissociação é realizada por meio de uma operação de fechamento.**

Características dos arquivos

- Podem armazenar grande quantidade de informação.
- Dados são persistentes (gravados em disco).
- Acesso aos dados pode não ser sequencial (acesso direto à registro de um banco de dados).
- Acesso à informação pode ser concorrente (mais de um programa ao mesmo tempo).

Nomes e extensões

- Arquivos são identificados por um nome.
- O nome de um arquivo pode ter uma extensão que indica o tipo do conteúdo do arquivo.

```
arquivo.ext
```

Tipos de arquivos em C

- **Arquivo bufferizado (*formatado ou de alto nível*): definido pelo padrão ANSI e UNIX.**
- **Arquivo não-bufferizado (*não-formatado ou de baixo nível*): definido apenas pelo UNIX.**

Tipos de arquivos

- **Arquivo texto: Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por editores de texto simples.**
  - **Exemplos: código C, texto simples, páginas HTML.**
- **Arquivo binário: Sequência de bits sujeita às convenções dos programas que o gerou, não legíveis diretamente.**
  - **Exemplos: arquivos executáveis ou compactados, documentos do Word.**

Entrada e saída em disco

- **Filas de bytes: é um dispositivo lógico de entrada ou saída de dados, independente do dispositivo real. Um arquivo (dispositivo externo) deve ser associado a uma fila de bytes.**
  - **C utiliza 5 filas de bytes pré-definidas, são elas:**

Fila	Função da fila
stdin	Entrada padrão (*standard input*)
stdout	Saída padrão (*standard output*)
stderr	Erro padrão (*standard error*)
stdprn	Saída para impressora (*standard printer*)
stdaux	Saída auxiliar (*standard auxiliary*)

Entrada e saída em disco

- **Filas de texto: Uma fila de texto é uma sequência de caracteres organizada em linhas. Cada linha é finalizada por um caracter '\n '. Pode ocorrer certas traduções de caracteres, tal como: '\n' ser convertida em CR (13) + LF (10). Dessa forma, pode não haver correspondência de 1 para 1 entre os caracteres que o computador escreve (lê) e aqueles no dispositivo externo.**
  - **CR – Carriage Return (retorno do carro - cursor)**
  - **LF – Line Feed (avanço de linha)**
  - **CR + LF = <enter>**

Entrada e saída em disco

- **Filas binárias: possui correspondência unívoca (1 para 1) com os bytes do dispositivo externo. Portanto nenhuma tradução de caracteres ocorrerá.**

- **Um arquivo binário pode conter tipos de dados diferentes, como por exemplo: char, int, float, vetores, ponteiros, strings, etc. ...**

Caminhos absolutos ou relativos

- **O nome de um arquivo pode conter o seu diretório, ou seja, o caminho para encontrar tal arquivo.**
- **Os caminhos podem ser especificados de duas formas:**
  - **Caminho absoluto: descrição desde o diretório raiz.**

```
/bin/emacs
/home/usr1/arq.txt
```

  - **Caminho relativo: descrição desde o diretório corrente.**

```
arq.txt
mc102/lab.c
```

ARQUIVO BUFFERIZADO

Sistema de arquivo bufferizado

- **A ligação comum que mantém unido o sistema de entrada e saída bufferizado é um ponteiro que aponta para o arquivo. O ponteiro do arquivo identifica um determinado arquivo em disco e é usado pela fila associada a ele para direcionar cada uma das funções de entrada e saída bufferizada para o lugar em que elas operam. Um ponteiro de arquivo é uma variável de ponteiro do tipo FILE *. O tipo FILE é pré-definido em "stdio.h".**

Ponteiro de arquivo

- Como já comentado, a associação de um arquivo a uma stream é realizada pela operação de abertura.
- A abertura de um arquivo retorna um ponteiro especial para o início do arquivo, conhecido como ponteiro de arquivo.
- Basicamente, o ponteiro de arquivo identifica um arquivo específico em disco e é utilizado pela stream associada para direcionar as operações de entrada/saída (E/S).

Ponteiro de arquivo

- Um ponteiro de arquivo deve ser declarado como sendo do tipo FILE.

```
FILE *arq
```

- O tipo FILE está definido na biblioteca `stdio.h`
- As funções para manipular um arquivo utilizam um ponteiro do tipo FILE.

Funções para manipulação de arquivos bufferizados

Função	Finalidade
`fopen()`	Abre uma stream (arquivo)
`fclose()`	Fecha uma stream (arquivo)
`putc(),fputc()`	Escreve um caractere em um arquivo
`getc(),fgetc()`	Lê um caractere de um arquivo
`fputs()`	Escreve uma string em um arquivo
`fgets()`	Lê uma string de um arquivo
`fputw()`	Escreve um inteiro em um arquivo
`fgetw()`	Lê um inteiro em um arquivo

Funções para manipulação de arquivos bufferizados

Função	Finalidade
`fprintf()`	É para um arquivo o que printf() é para o console
`fscanf()`	É para um arquivo o que scanf() é para o console
`fwrite()`	Escreve tipos de dados maiores que um byte em arquivo
`fread()`	Lê tipos de dados maiores que um byte em arquivo
`feof()`	Devolve verdadeiro se o fim de arquivo for atingido
`fseek()`	Posiciona o arquivo em um byte específico
`ftell()`	Informa a posição atual de um arquivo

Funções para manipulação de arquivos bufferizados

Função	Finalidade
`fflush()`	Descarrega um arquivo
`rewind()`	Reposiciona o localizador no início do arquivo
`ferror()`	Devolve verdadeiro se ocorreu um erro
`remove()`	Apaga um arquivo
`freopen()`	Associa uma stream existente com um novo arquivo

Sintaxe das funções para manipulação de arquivos

Função	Sintaxe
`fopen()`	FILE *fopen(const char *filename, const char *mode);
`fclose()`	int fclose(FILE *fp);
`putc(),` `fputc()`	int putc(int ch, FILE *stream); int fputc(int ch, FILE *stream);
`getc(),` `fgetc()`	int getc(FILE *stream); int getc(FILE *stream);
`fputs()`	int fputs(const char *str, FILE *stream);
`fgets()`	char *fgets(char *str, int length, FILE *stream);
`fputw()`	int fputw(int w, FILE *stream);
`fgetw()`	int fgetw(FILE *stream);

Sintaxe das funções para manipulação de arquivos

Função	Sintaxe
fprintf()	int fprintf(FILE *stream, const char *format, arg1, arg2,...);
fscanf()	int fscanf(FILE *stream, const char *format, &arg1, &arg2,...);
fwrite()	size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream):
fread()	size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream):
feof()	int feof(FILE *stream);
fseek()	int fseek(FILE *stream, long int numbytes, int origem);
ftell()	int ftell(FILE *stream);

Sintaxe das funções para manipulação de arquivos

Função	Sintaxe
fflush()	int fflush(FILE *stream);
rewind()	void rewind(FILE *stream);
ferror()	int ferror(FILE *stream);
remove()	int remove(char *filename);
freopen()	FILE *freopen(const char *filename, const char *mode, FILE *stream);

Abrindo um arquivo – Função fopen( )

- Abre um arquivo para leitura e escrita.
- A função devolve um ponteiro para o arquivo.
- Nunca se deve alterar o valor desse ponteiro.
- O parâmetro <modo> determina como o arquivo será aberto.

```
fopen (<nome do arquivo>, <modo>)
```

Abrindo um arquivo – Modo

fopen() FILE *fopen(const char *filename, const char *mode);

Modo	Significado
r	Abre um arquivo texto para leitura
w	Cria/sobrescreve um arquivo texto para escrita
a	Anexa a um arquivo texto existente
rb	Abre um arquivo binário para leitura
wb	Cria/sobrescreve um arquivo binário para escrita
ab	Anexa a um arquivo binário existente
r+	Abre um arquivo texto para leitura e escrita
w+	Cria/sobrescreve um arquivo texto para leitura e escrita
rb+	Abre um arquivo binário para leitura e escrita
wb+	Cria/sobrescreve um arquivo binário para leitura e escrita

Abrindo um arquivo – Função `fopen( )`

- **Caso ocorra um erro na abertura, retornará NULL.**
- **Deve-se sempre testar o sucesso de `fopen( )` antes de tentar qualquer outra operação sobre o arquivo.**
- **O número máximo de arquivos que podem estar abertos simultaneamente é dado pela macro FOPEN_MAX, da biblioteca `stdio.h`.**

Abrindo um arquivo – Função `fopen( )`

- ***fopen()***

```
FILE  *fp;
fp = fopen("test.txt", "w");
```

Forma usual:

```
FILE *fp;
if ((fp=fopen("teste.txt", "w")) == NULL)
{
      puts( "Não posso abrir o arquivo\n");
      exit (1);
}
```

Fechando um arquivo – Função `fclose()`

- Em caso de sucesso, `fclose()` retorna `0` (zero). Qualquer outro valor indica erro.

```
fclose (<pt_arquivo>);
```

```
fclose(fp);
```

Escrevendo um caractere – Função `putc()`

- O padrão C ANSI define duas funções equivalentes para escrever caracteres em um arquivo: `putc()` e `fputc()`.
- Ambas escrevem caracteres em um arquivo que foi previamente aberto por `fopen()`.

```
putc (<caractere>, <pt_arquivo>);
fputc (<caractere>, <pt_arquivo>);
```

Escrevendo caracteres com putc()

```
#include “stdio.h”
#include “stdlib.h”

void main(int argc, char *argv[])
{
  FILE *fp;
  char ch;
  if(argc !=2){
    printf(“Voce esqueceu de entrar o nome do arquivo \n”);
    exit(1);
  }
  if((fp=fopen(argv[1],”w”))==NULL){
    printf(“Arquivo nao pode ser aberto\n”);
    exit(1);
  }
  do {
    ch = getchar();
    putc(ch,fp);
  } while(ch!=‘$’);
  fclose(fp);
}
```

Lendo um caractere – Função getc()

- Para ler um caractere em um arquivo aberto por fopen(), pode-se usar as funções getc() ou fgetc().

```
fgetc (<pt_arquivo>);
```

- A função devolve EOF quando o final do arquivo é alcançado.

Lendo um caractere – Função getc()

- **No código abaixo, o arquivo é lido até que a marca de final de arquivo (EOF – *End of File*) seja alcançada.**

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

Lendo caracteres com getc()

```
#include “stdio.h”
#include “stdlib.h”

void main(int argc, char *argv[])
{
  FILE *fp;
  char ch;
  if(argc !=2){
    printf(“Voce esqueceu de entrar o nome do arquivo \n”);
    exit(1);
  }
  if((fp=fopen(argv[1],”r”))==NULL){
    printf(“Arquivo nao pode ser aberto\n”);
    exit(1);
  }
  ch = getc(fp);
  while (ch!=EOF){
    putchar(ch);
    ch = getc(fp);
  }
  fclose(fp);
}
```

Escrevendo uma string – Função `fputs()`

- **Grava string de caracteres na stream especificada.**
- **Devolve EOF se ocorrer erro.**

```
fputs (string, pt_arquivo);
```

Escrevendo strings com `fputs()`

```
#include “stdio.h”
#include “stdlib.h”
#include “string.h”

void main(void)
{
  char str[80];
  FILE *fp;
  if((fp=fopen(“frase.dat”,”w”))==NULL)
  {
    printf(“Arquivo nao pode ser aberto\n”);
    exit(1);
  }
  do {
    printf(“entre com uma string (CR para sair): \n”);
    gets(str);
    strcat(str,”\n”);
    fputs(str,fp);
  } while (*str != ‘\n”);
  fclose(fp);
}
```

Lendo uma string – Função `fgets()`

- Lê uma string de caracteres da stream especificada até que um caractere de nova linha seja lido ou que length-1 caracteres sejam lidos.
- Se lido, o caractere de nova linha (\n) faz parte da string.
- A string resultante é terminada por um caractere nulo (\0).

```
fgets (string, lenght, pt_arquivo);
```

Lendo strings com `fgets()`

```
#include “stdio.h”
#include “stdlib.h”
#include “string.h”

void main(void)
{
  char str[80];
  FILE *fp;
  if((fp=fopen(“frase.dat”,”r”))==NULL)
  {
    printf(“Arquivo nao pode ser aberto\n”);
    exit(1);
  }
  while(!feof(fp))
  {
    fgets(str, 79, fp);
    printf(str);
  }
  fclose(fp);
}
```

Lendo e escrevendo estruturas de dados – Funções `fread()` e `fwrite()`

- **Permitem a leitura e escrita de blocos de qualquer tipo de dado.**

```
fread(buffer, no_bytes, no_itens, pt_arq);
fwrite(buffer, no_bytes, no_itens, pt_arq);
```

- **Buffer é um ponteiro para que receberá/ fornecerá os dados lidos/escritos no arquivo.**
- **no_bytes é o número de bytes a ler/escrever.**
- **no_itens determina quantos itens serão lidos/ escritos, cada um de comprimento no_bytes.**

Lendo e escrevendo estruturas de dados – Funções `fread()` e `fwrite()`

- **A função `fread()` devolve o número de itens lido e a função `fwrite()` devolve o número de itens escritos.**
- **Se tais valores forem menores que o campo no_itens, é porque o final do arquivo (EOF) foi atingido o ocorreu um erro.**
- **Uma das aplicações mais úteis dessas funções envolve ler e escrever tipos de dados definidos pelo usuário, especialmente estruturas.**

Lendo e escrevendo em structs – Exemplo `fread()`, `fwrite()`

```
#include <stdio.h>
#include <string .h>
typedef struct _PESSOA {
  char nome [40]; int ano;
} PESSOA ;
int main ()
{
  FILE *arq;
  char nome [40] , linha [80];
  PESSOA turma [4], back [4];
  int i;
  for (i=0; i <4; i++)
  {
    puts ("Nome ? ");
    fgets( turma[i].nome , 40, stdin );
    turma[i].nome  [  strlen(  turma[i].
     nome ) -1]= '\0 ';
    puts  ("Ano  ?  ");  fgets(linha,  80,
     stdin);
    sscanf (linha, "%d", & turma[i].ano);
  }
  puts ("\ nGravando \n");
  puts ("Qual o nome do arquivo ?");
  fgets(nome, 40, stdin);
  nome [strlen ( nome ) -1]= '\0 ';
  if ((arq = fopen(nome, "w+")) == NULL )
  {
    puts("Arquivo não pode ser aberto ");
    return 1;
  }
  for (i=0; i <4; i++) {
    if  (  fwrite(  &turma[i],  sizeof(
          PESSOA), 1, arq) != 1)
    puts ("Erro na escrita .");
  }
  rewind (arq);
  for (i=0; i <4; i++)
  {
    if  (  fread(  &  back  [i],  sizeof(
          PESSOA), 1, arq) != 1) {
      if ( feof (arq) ) break;
      puts (" Erro na leitura .");
    }
  }
  for (i=0; i <4; i++) {
    printf("Nome = %s\n", back[i].nome);
    printf("Ano = %d\n\n", back [i].ano);
  }
  return 0;
};
```

Lendo e escrevendo dados formatados – Funções `fscanf()` e `fprintf()`

- **Essas funções funcionam exatamente como `printf()` e `scanf()`, exceto por operarem com arquivos.**

```
fprintf (pt_arq, string_controle, argumentos);
fscanf (pt_arq, string_controle, argumentos);
```

- **Note que `fprintf()` e `scanf()` direcionam suas operações de entrada e saída formatadas para o arquivo apontado por `pt_arq`.**

Lendo e escrevendo dados formatados – Funções `fscanf()` e `fprintf()`

- Embora essas duas funções sejam a maneira mais fácil de escrever e ler dados em arquivos de disco, nem sempre são a escolha mais apropriada.
- Como os dados são escritos em ASCII e formatados como apareceriam na tela (e não em binário), um tempo extra é perdido a cada chamada.
- Portanto, se há preocupação com velocidade ou tamanho de arquivo, deve-se utilizar as funções `fread()` e `fwrite()`.

Lendo e escrevendo formatado – Exemplo `fscanf()`, `fprintf()`

```
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
int main(void)
{
  FILE *fp;
  char s[80];
  int t;
  if((fp=fopen("test",   "w"))   ==
    NULL) {
    printf("O arquivo não pode ser
    aberto.\n");
    exit(1);
  }
  printf("Entre com uma string e um
    número: ");
  /* ler do teclado */
  fscanf(stdin, "%s%d", s, &t);
  /* escrever no arquivo */
  fprintf(fp, "%s %d", s, t);
  fclose(fp);
   if((fp  =  fopen("test","r"))  ==
    NULL) {
    printf("O arquivo não pode ser
    aberto.\n");
    exit(1);
  }
  /* ler do arquivo */
  fscanf(fp, "%s%d", s, &t);
  /* escrever na tela */
  fprintf(stdout, "%s %d", s, t);
  return 0;
}
```

Exemplo

- **Escreva um programa que leia um arquivo do tipo base de dados em .txt, retire e apresente informações importantes na tela. O arquivo pode ser do tipo:**
- **Nome: Joao da Silva Telefone: +55 00 000-0000 Endereco: Rua do Joao, 168 $ Nome: Maria da Silva Telefone: +55 99 999-9999 Endereco: Rua da Vila, 093 $ ... O símbolo $** (dólar) entre os dados é só para informar que acabou uma sequência de dados, e pode ser usado para restabelecer o sincronismo, caso o programa se perca em algum ponto. Você pode usar um flag para encerrar, mas e melhor que use **feof**() (é mais elegante...).

Solução

```
include <stdio.h>  #define NOME_TAM 26  #define ENDER_TAM 30
 #define TELE_TAM 20
void main()  {        FILE *arq;        char nome[NOME_TAM+1],
ender[ENDER_TAM+1], tele[TELE_TAM+1], arq_nome[30];     char c;
      printf("\n\nArquivo de entrada: ");        gets(arq_nome);
        if ((arq = fopen(arq_nome, "r")) == NULL)         {
            printf("\nErro ao abrir o arquivo %s!",arq_nome);
       exit(1);     }
     printf("\n\nImprimindo dados do arquivo: %s\n", arq_nome);
```

Solução

```
    while (!feof(arq))/* Enquanto nao encontra o fim do arquivo */
       {           c =0;           /* Leitura sincronizada.. */
        fscanf(arq, "%s", nome);  /* Salta a palavra: 'Nome:' */
         fgets(nome, NOME_TAM, arq);          fscanf(arq, "%s",
tele);  /* Salta a palavra: 'Telefone:' */         fgets(tele,
TELE_TAM, arq);          fscanf(arq, "\n%s", ender);/* Salta a
palavra: 'Endereco:'*/           fgets(ender, ENDER_TAM, arq);
      printf("\nNome: %s\nEndereco: %sTelefone: %s", nome, ender,
tele);
     /* Caso alguma parte se perca, e o ponteiro no arquivo fique parado em
alguma parte do registro, arotina fara a sincronizacao. */        while(c
!= '$')        {          c = getc(arq);          if (feof(arq))
break;       }     }     fclose(arq);  }
```

Biblioteca `stdio.h`

- O arquivo de cabeçalho `stdio.h` também define várias macros como: NULL, EOF, FOPEN_MAX, SEEK_SET, SEEK_CUR e SEEK_END.
  - A macro NULL define um ponteiro nulo.
  - EOF define -1 e é usada quando uma função tenta ler além do final do arquivo.
  - As outras macros são utilizadas pela função fseek().

Acesso aleatório – Função `fseek()`

- **Operações de leitura e escrita aleatórias podem ser realizadas com a ajuda da função fseek(), que modifica o indicador de posição de arquivo.**

```
fseek (pt_arq, no_bytes, origem);
```

- **no_bytes é o número de bytes, a partir de origem, que se tornará a nova posição corrente.**
- **origem é uma das seguintes macros:**

Acesso aleatório – Função `fseek()`

Origem	Macro
Início do arquivo	SEEK_SET
Posição atual	SEEK_CUR
Final do arquivo	SEEK_END

- **SEEK_SET - Parte do início do arquivo e avança <n> bytes**
- **SEEK_END - Parte do final do arquivo e retrocede <n> bytes**
- **SEEK_CUR - Parte do local atual e avança <n> bytes**
- **A função `fseek()` pode ser utilizada para efetuar movimentações em múltiplos de qualquer tipo de dado, simplesmente utilizando-se o comando `sizeof()`.**

Acesso aleatório – Função `fseek()`

```
fseek(<pt_arquivo>, <numbytes>, <origem>);
```

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
  FILE *fp;
  if(argc!=3) {
    printf("Uso: SEEK nomearq byte\n");
    exit(1);  }
  if((fp = fopen(argv[1], "rb"))==NULL) {
    printf("O arquivo não pode ser aberto.\n");
    exit(1);  }
  if(fseek(fp, atol(argv[2]), SEEK_SET)) {
    printf("Erro na busca.\n");
    exit(1);  }
  printf("o byte em %ld é %c.\n", atol(argv[2]), getc(fp));
  fclose(fp);
  return 0;
}
```

Retornando ao início do arquivo – Função `rewind()`

```
rewind(<pt_arquivo>);
```

```
#include <stdio.h>
#include <stdlib .h>
int main (void )
{
  int c;
  FILE *arq;
  char *nome = " texto.txt";
  /* Abre o arquivo para leitura
    e escrita */
  if (( arq = fopen(nome , "w+"))
    == NULL ) {
    printf("\n\nNao  foi  possivel
          abrir o arquivo .\n");
    exit (1);
  }
  /* Cada caractere digitado sera
    gravado no arquivo */
  c = getchar ();
  while (! feof ( stdin ))
  {
    fputc(c, arq );
    c = getchar ();
  }
  /* volta ao inicio do arquivo*/
  rewind (arq);
  printf("\ nTerminei de escrever
    , agora vou ler .\n");
  c = fgetc(arq);
  while (! feof (arq))
  {
    putchar (c);
    c = fgetc(arq);
  }
  fclose (arq);
  return 0;
}
```

Uso da Função ferror()

```
ferror(<pt_arquivo>);
```

```
#include <stdio.h>
int main (void)
{
  FILE *Arq ;
  Arq =fopen(" MeusDados.txt","r");
  if (Arq == NULL ){
    printf (" Erro abrindo arquivo .");
    return 1;
  }
  else  {
    fputc ('x', Arq );
    if ( ferror ( Arq )) {
      printf ("Erro escrevendo arquivo \n");
      fclose (Arq );
      return 1;
    }
  }
  return 0;
};
```

Apagando um arquivo – Função remove()

```
remove(<pt_arquivo>);
```

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
int main(int argc, char *argv[])
{
  char str[80];
  if(argc!=2) {
    printf("Uso: xerase <nomearq>\n");
    exit(1);
  }
  printf("Apagar %s? (S/N): ", argv[1]);
  gets(str);
  if(toupper(*str)=='S')
    if(remove(argv[1])) {
      printf("O arquivo não pode ser apagado.\n");
      exit(1);
    }
  return 0;
}
```

Redirecionando um arquivo – Função `freopen()`

```
freopen(<pt_arquivo>);
```

```
#include <stdio.h>
int main(void)
{
  char str[80];
  freopen("SAIDA", "w", stdout);
  printf("Entre com uma string: ");
  gets(str);
  printf(str);
  return 0;
}
```

Em geral, redirecionar streams padrões usando freopen( ) é útil em situações especiais, tais como debugging. Entretanto, executar E/S em disco usando stdin e stdout redirecionada não é tão eficiente quanto usar funções como fread( ) ou fwrite( ).

Arquivo binário – Exercícios

- EP.1: **Faça um programa que crie uma base de dados com 10 alunos. O arquivo em disco deve conter:**
  - **matrícula do aluno;**
  - **nome do aluno;**
  - **nota Prova1;**
  - **nota Prova2.**

- EP.2: **Faça um programa que, dado a matrícula do aluno, procure-o no arquivo e retorne seu nome e média.**

- EP.3: **Faça um programa que, dada a matrícula do aluno, altere suas notas no arquivo.**

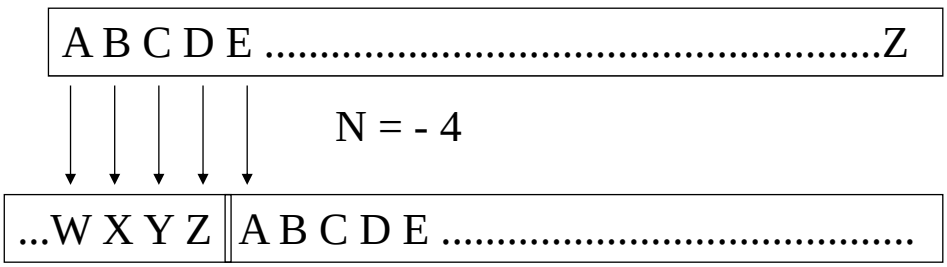
Arquivo texto – Exercícios

- EP.4: **Faça um programa que lê, via teclado, um valor inteiro N. Em seguida lê, também via teclado, e grava, em um arquivo do tipo texto, de nome interno Texto e nome externo 'DADOS.TXT', o Nome e a Idade em anos de N pessoas.**

- EP.5: **Faça um programa que mostra no vídeo todos os valores de Nome e Idade que foram gravados pelo exemplo anterior, no arquivo texto de nome interno Texto e de nome externo 'DADOS.TXT'.**

Arquivo texto – Exercícios

- EP.6: **Implemente um sistema criptográfico que leia um arquivo texto escrito por você e embaralhe as letras de acordo com uma senha *N*. Essa senha irá deslocar o alfabeto de *N* posições. Use A-Z, a-z e 0-9. Grave então o texto criptografado em um novo arquivo.**



Arquivo texto – Exercícios

- EP.7: Implementar um "*Sistema de Agenda*"

- **Requisitos do sistema:**
  - **o sistema deverá ter um menu de inserção, alteração e remoção de cadastros;**
  - **deverá permitir também consulta aos dados por nome, cidade ou estado;**
  - **os dados ficarão gravados em disco;**
  - **os cadastros deverão conter: nome, endereço, bairro, cidade, estado, cep, telefone e celular.**

Solução EP6 – Criptografia (v.1)

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

char *criptografa(char s[80], int cifra) {
  int i, n = strlen(s);
  char saux[80];
  strcpy(saux, s);
  for (i=0; i<n; i++) {
    saux[i] = saux[i] + cifra;
  }
  return(saux);
}

char *descriptografa(char s[80], int cifra) {
  int i, n = strlen(s);
  char saux[80];
  strcpy(saux, s);
  for (i=0; i<n; i++) {
    saux[i] = saux[i] - cifra;
  }
  return(saux);
}
```

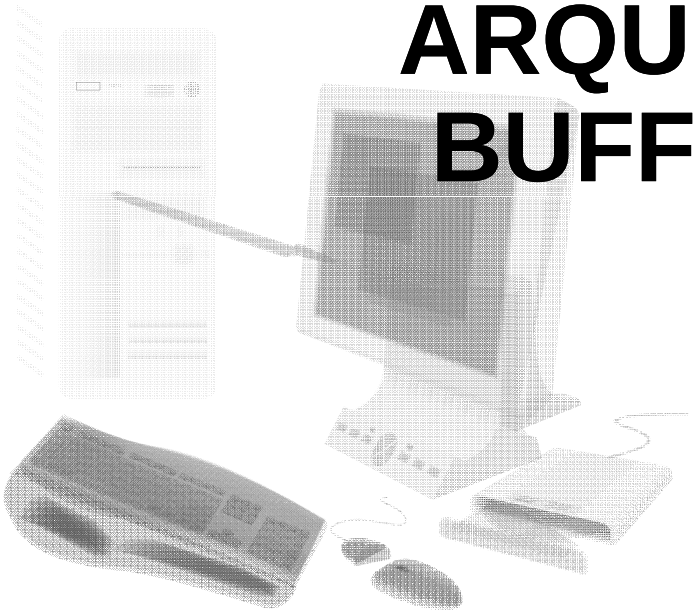
Solução EP6 – Criptografia (v.1)

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

void main() {
  char s[80], sc[80], sd[80];
  int cifra;
  printf("Informe um texto:\n");
  gets(s);
  printf("\nInforme o valor da cifra:\n");
  scanf("%d", &cifra);
  strcpy(sc, criptografa(s, cifra));
  strcpy(sd, descriptografa(sc, cifra));
  printf("\nTexto criptografado:\n");
  printf("%s", sc);
  printf("\n\nTexto descriptografado:\n");
  printf("%s", sd);
}
```


Solução EP6 – Criptografia (v.2)

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

#define caesar(x) rot(13, x)
#define decaesar(x) rot(13, x)
#define decrypt_rot(x, y) rot((26-x), y)

void rot(int c, char *str)
{
    int l = strlen(str);
    const     char     *alpha[2]     =     {     "abcdefghijklmnopqrstuvwxyz",
    "ABCDEFGHIJKLMNOPQRSTUVWXYZ"};

    int i;
    for (i = 0; i < l; i++)
    {
          if (!isalpha(str[i]))
                    continue;

          str[i] = alpha[isupper(str[i])][((int)(tolower(str[i])-'a')+c)%26];
    }
}
```

ARQUIVO NÃO-BUFFERIZADO



Sistema de arquivo não-bufferizado

- **Ao contrário do sistema de entrada e saída de alto nível (sistema bufferizado), o sistema de baixo nível (sistema não-bufferizado) não utiliza ponteiros de arquivo do tipo FILE, mas sim descritores de arquivos chamados *handles* do tipo int, ou seja, uma variável inteira.**

- **Utiliza as bibliotecas `stdio.h` e `fcntl.h`.**

Funções para manipulação de arquivos não-bufferizados

Função	Finalidade
`open()`	Permite abrir/criar arquivo em disco
`creat()`	Cria um arquivo para operação de escrita e leitura
`close()`	Fecha um arquivo
`write()`	Grava caracteres no arquivo
`read()`	Copia os dados lidos no buffer
`unlink()`	Elimina um arquivo do disco
`lseek()`	Devolve o número de bytes se tiver sucesso
`eof()`	Devolve: (1) fim do arquivo, (0) não é o fim do arquivo, (-1) erro
`tell()`	Devolve a posição corrente do descritor

Sintaxe das funções para manipulação de arquivos

Função	Sintaxe
`open()`	int open (char *nomearquivo, int acesso);
`creat()`	int creat (const char *nomearquivo, int modo);
`close()`	int close (int fd);
`write()`	int write (int fd, void *buf, int tamanho);
`read()`	int read (int fd, void *buf, int tamanho);
`unlink()`	int unlink (const char *nomearquivo);
`lseek()`	long int lseek (int fd, long int num_bytes, int origem);
`eof()`	int int eof (int fd);
`tell()`	long int tell (int fd);

Tipos de acesso em arquivos não-bufferizados

open() **int open(char *filename, int acesso);**

Acesso	Significado
O_RDONLY	Abre um arquivo apenas para leitura
O_WRONLY	Abre um arquivo apenas para escrita
O_RDRW	Abre um arquivo para leitura e escrita
O_CREAT	Cria e abre
O_TRUNC	Abre com '*truncation*'
O_EXCL	Abre exclusiva
O_APPEND	Abre um arquivo texto para incluir no fim
O_TEXT	Translação CR para LF
O_BYNARY	Sem translação

Modos de acesso em arquivos não-bufferizados

creat() **int creat(char *filename, int modo);**

Modo	Significado
O_CREAT	Criar um arquivo se ele não existe
O_TRUNC	Truncar o arquivo se ele existe
O_APPEND	Anexar no fim antes de cada escrita
O_EXCL	Excluir. Força a criação se o arquivo existe
O_RDONLY	Acesso somente para leitura
O_WRONLY	Acesso somente para escrita
O_RDWR	Acesso para leitura e escrita

Abertura de um arquivo Unix

```
#include <io.h>
#include <fnctl.h>
main()
{
  int i; /* identificador do arquivo */
  char buff[512];/* definição do buffer */
  if((i=open(“teste.txt”,O_RDONLY | O_BINARY)) < 0)
      perror(“Não posso abrir arquivo:”);
  ...
}
```

- No exemplo acima, é mostrada a abertura de um arquivo somente para leitura (O_RDONLY) e binário (O_BINARY). O comando open() devolve um inteiro para i como descritor do arquivo para futuras operações no mesmo.

Programa criar um arquivo de palavras permitindo a gravação destas palavras

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

int main (void)
{
 int fd;
 char reg[40];
 char nome_do_arquivo[14];
 unsigned int n;
 char ch;

 printf("Nome do arquivo: ");
 scanf(“%13s”, nome_do_arquivo);
 if  ((fd  =  open(  nome_do_arquivo,
    O_RDONLY,"r")) == EOF)
    if      ((fd      =      creat(
    nome_do_arquivo,O_RDWR)) == EOF)
       printf("Erro  Fatal:  Problema
    no disco\n");
    else  {
       n = 0;
       do {
         printf("%d: Palavra: ", n);
         scanf("%s", reg);
         write(fd, reg, sizeof(reg));
         n++;
         printf("Continua  [S]im  ou
    [N]ão ?");
         do {
            ch = getchar();
         }while(!strchr("SsNn", ch));
       } while (strchr("Ss",ch));
       close(fd);
    }
 else {
       printf("ERRO: Arquivo já EXISTE
     \n");
       close(fd);
     }
}
```

Programa permite abrir o arquivo de palavras e exibe-os na tela

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

int main (void)
{
 int fd;
 char reg[40];
 char nome_do_arquivo[14];
 unsigned int n;
 int error;

 printf("Nome do Arquivo: ");
 scanf(“%13s”, nome_do_arquivo);
 if  ((fd  =  open(  nome_do_arquivo,
    O_RDONLY,"r")) == EOF)
     printf("ERRO:    Arquivo    NÃO
    existe ...\n");
else
       {
          n = 0;
          do {
              error  =  read(fd,  reg,
    sizeof(reg));
              printf("%d:    Palavra:
    %s\n", n, reg);
              n++;
            } while (error);
// error = 0 – fim do arquivo
          close(fd);
          printf(         “         %d
    palavra(s)\n”,n);
        }
}
```

Programa permite alterar o arquivo de palavras

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

int main (void)
{
 int fd;
 char reg[40];
 char nome_do_arquivo[14];
 unsigned int n;
 long int posicao;
 char ch;

 printf("Nome do arquivo: ");
 scanf(“%s”, nome_do_arquivo);
 if  ((fd  =  open(  nome_do_arquivo,
    O_RDWR,"w")) == EOF)
     printf("ERRO:    Arquivo    NÃO
    existe ...\n");
 else {
      do {
        printf("Registro: ");
    scanf("%d", &n);
    posicao = n * sizeof(reg);
    lseek(fd, posicao, SEEK_SET);
    if (read(fd, reg, sizeof(reg))) {
       printf("%d: Palavra: %s\n", n,
    reg);
       printf("NOVA PALAVRA: ");
       scanf(“%s”, reg);
       lseek(fd, posicao, SEEK_SET);
       write(fd, reg, sizeof(reg));
     }
     else
       printf("ERRO:   Registro   não
    existe\n");
     printf("Continua [S] ou [N]? ");
     do {
          ch = getchar();
     } while (!strchr("SsNn",ch));
  } while (strchr("Ss",ch));
  close(fd);
 }
}
```

Programa-exemplo de escritas em arquivos

```
#include <stdio.h>  /* all I/O functions */
#include <fcntl.h>  /* all UNIX low level functions*/
#include <string.h> /* all string manipulation functions */
#include <stdlib.h> /* permission modes for UNIX low level */
#include <io.h>
#include <sys/types.h>
#include <sys/stat.h>

extern int errno;   /* needed to go with perror() */
/*   describe a structure template, no variable of declared */
typedef struct tagPERSON {
    char name[30];
    char street[20];
    char city[20];
    char state[3];
    char zip[6];
    char ssn[13];
    int age;
    int height;
    int weight;
} PERSON;
```

Programa-exemplo de escritas em arquivos

```
/*  function prototypes  */
int getdata( PERSON * );
int showdata(PERSON * );
int puts_gets( void );
int fprnt_fscan( void );
int fread_fwrite( void );
int read_write( void );
int err_handler(FILE *, char *, int );
/*  S T A R T   O F   P R O G R A M  */
int which;
    /*    which functions are to be executed     */
    do {
        printf("\nWhich set of I/O functions are to be tested?");
        printf("\n    1. fputs and fgets");
        printf("\n    2. fprintf and fscanf");
        printf("\n    3. fread and fwrite");
        printf("\n    4. read and write");
        printf("\n    5. quit this program");
        printf("\nEnter your selection: ");
        gets(ans);
        which = atoi(ans);
```

Programa-exemplo de escritas em arquivos

```
        switch(which) {
            case 1:
                puts_gets();
                break;
            case 2:
                fprnt_fscan();
                break;
            case 3:
                fread_fwrite();
                break;
            case 4:
                read_write();
                break;
            case 5:
                return(0);
            default:
                printf("\n\nInvalid selection . . . try again!");
                break;
        }
    } while(1);
}
```

Programa-exemplo de escritas em arquivos

```
 /*   read data from screen into structure elements   */
int getdata( PERSON *ptr)
{
    int result;

    printf("\nEnter your name: ");      gets(ptr->name);
    printf("\nEnter your street: ");    gets(ptr->street);
    printf("\nEnter your city: ");      gets(ptr->city);
    printf("\nEnter your state: ");     gets(ptr->state);
    printf("\nEnter your zip code: ");  gets(ptr->zip);
    printf("\nEnter your ssn: ");       gets(ptr->ssn);
    printf("\nEnter your age: ");       scanf("%d",&ptr->age);
    printf("\nEnter your height: ");    scanf("%d",&ptr->height);
    printf("\nEnter your weight: ");    scanf("%d",&ptr->weight);
    /*   flush the input data stream so no newlines are left    */
    if((result = fflush(stdin)) == EOF)
        err_handler(stdin,"stdin",1);
    return 0;
}
```

Programa-exemplo de escritas em arquivos

```
int showdata(PERSON *ptr)
{
    printf("\nPERSON: %s",ptr->name);
    printf("\n      : %s",ptr->street);
    printf("\n      : %s",ptr->city);
    printf("\n      : %s",ptr->state);
    printf("\n      : %s",ptr->zip);
    printf("\n      : %s",ptr->ssn);
    printf("\n      : %d",ptr->age);
    printf("\n      : %d",ptr->height);
    printf("\n      : %d",ptr->weight);
    return 0;
}
```

Programa-exemplo de escritas em arquivos

```
int puts_gets() {
  FILE *fp;
  PERSON my;
  char *val;
  char ans[2],filename[16],text[80];
  int   rtnval, linecnt, lgth;
  strcpy(filename,"testfil1.dat");
  if((fp = fopen(filename,"w")) == NULL)
    err_handler(fp,filename,1);
    do {
        printf("\nEnter Text:");
        gets(text);
        lgth = strlen(text);
        text[lgth] = '\n';      /* replace NULL terminator */
        text[lgth + 1] = '\0';  /* place NULL terminator  */
        if((rtnval = fputs(text,fp)) == EOF)
            err_handler(fp,filename,2);
        strcpy(ans," ");
        printf("\nContinue(Y/N)? ");
        gets(ans);
    } while(!strcmp(ans,"y"));
```

Programa-exemplo de escritas em arquivos

```
    if((rtnval = fclose(fp)) == EOF)   {
        err_handler(fp,filename,3);
    }
    if((fp = fopen(filename,"r")) == NULL)
        err_handler(fp,filename,3);
    linecnt = 0;
    do {
        if((val = fgets(text,sizeof(text),fp)) == NULL)
            if(err_handler(fp,filename,3))
                break;
        printf("\nLine %d:%s",linecnt,text);
        ++linecnt;
        strcpy(ans," ");
        printf("\nContinue(Y/N)? ");
        gets(ans);
    } while(!strcmp(ans,"y"));
    if((rtnval = fclose(fp)) == EOF)   {
        err_handler(fp,filename,3);
    }
    return 0;
}
```

Programa-exemplo de escritas em arquivos

```
int fprnt_fscan()
{
  FILE *fp;
  PERSON my;
  char filename[16],lname[20],tstreet[20],ans[2];
  int rtnval;
  strcpy(filename,"testfil2.dat");
  if((fp = fopen(filename,"w")) == NULL)    {
        perror("FPRNT_FSCAN(): cannot open file for write");
        exit(3);
  }
  do
  {
     printf("\nEnter only a single string ");
     printf("for name, street and city");
     getdata(&my);
     if((rtnval  =  fprintf(fp,"%s  %s  %s  %s  %s  %s  %d  %d  %d\n“,
         my.name,    my.street,  my.city,  my.state,  my.zip,  my.ssn,
         my.age, my.height, my.weight)) == EOF)  {
            perror("FPRNT_FSCAN(): cannot fprintf to file");
            exit(4);          }
```

Programa-exemplo de escritas em arquivos

```
    strcpy(ans," ");
    printf("\nContinue(Y/N)? ");    gets(ans);
  } while(!strcmp(ans,"y"));
  fclose(fp);
  if((fp = fopen(filename,"r")) == NULL)    {
        perror("FPRNT_FSCAN(): cannot open file for read");
        exit(5);
    }
  do {
    if((rtnval = fscanf(fp, "%s %s %s %s %s %s %s %s %d %d %d",
    my.name,  lname,  my.street,  tstreet,  my.city,  my.state,  my.zip,
    my.ssn, &my.age, &my.height, &my.weight)) == NULL)    {
        perror("FPRNT_FSCAN(): cannot fscanf from file");
        exit(3);
    }
    showdata(&my);     strcpy(ans," ");
    printf("\nContinue(Y/N)? ");    gets(ans);
  } while(!strcmp(ans,"y"));
  fclose(fp);
  return 0;
}
```

Programa-exemplo de escritas em arquivos

```
int fread_fwrite()
{
  FILE *fp;
  PERSON my;
  char filename[16],ans[2];
  int  rtnval;
  strcpy(filename,"testfil3.dat");
  if((fp = fopen(filename,"w+b")) == NULL)  {
        perror("FREAD_FWRITE(): cannot open file for write");
        exit(6);
  }
  do {
    getdata(&my);
    if((rtnval = fwrite(&my,sizeof(my),1,fp)) == EOF)   {
            perror("FREAD_FWRITE(): cannot fwrite to file");
            exit(7);
    }
    strcpy(ans," ");
    printf("\nContinue(Y/N)? ");       gets(ans);
  } while(!strcmp(ans,"y"));
  fclose(fp);
```

Programa-exemplo de escritas em arquivos

```
  if((fp = fopen(filename,"r+b")) == NULL)
  {
        perror("FREAD_FWRITE(): cannot open file for read");
        exit(8);
  }
  do {
    if((rtnval = fread(&my,sizeof(my),1,fp)) < 1)
       if(err_handler(fp,filename,9))
           break;
    showdata(&my);
    strcpy(ans," ");
    printf("\nContinue(Y/N)? ");
    gets(ans);
  }while(!strcmp(ans,"y"));
  fclose(fp);
  return 0;
}
```

Programa-exemplo de escritas em arquivos

```
int read_write()
{
  char buf[512];
  int fp;
  PERSON my;
  char filename[16], ans[2];
  int rtnval;
  strcpy(filename,"testfil4.dat");
  #ifdef PC
    if((fp = open(filename, O_WRONLY|O_CREAT|O_BINARY, S_IWRITE))  ==
    EOF)   {
        perror("READ_WRITE(): cannot open file for write");
        exit(10);
    }
  #else if((fp = open(filename, O_RDWR|O_CREAT, S_IREAD|S_IWRITE)) ==
    EOF)    {
        perror("READ_WRITE(): cannot open file for write");
        exit(10);
    }
  #endif
```

Programa-exemplo de escritas em arquivos

```
do
{
  #ifndef PC  printf("\nEnter only a single string ");
      printf("for name, street and city");
  #endif
  getdata(&my);
  #ifdef PC
      if((rtnval = write(fp,&my,sizeof(my))) == EOF)
      {
          perror("READ_WRITE(): cannot write to file");
          exit(7);
      }
   #else
      sprintf(buf,"%s %s %s %s %s %s %d %d %d", my.name, my.street,
       my.city,   my.state,   my.zip,   my.ssn,   my.age,   my.height,
       my.weight);
      if((rtnval = write(fp,buf,sizeof(my))) == EOF)    {
          perror("READ_WRITE(): cannot write to file");
          exit(7);
      }
   #endif
```

Programa-exemplo de escritas em arquivos

```
  strcpy(ans," ");
  printf("\nContinue(Y/N)? ");
  gets(ans);
} while(!strcmp(ans,"y"));
close(fp);
#ifdef PC
  if((fp = open(filename,O_RDONLY|O_BINARY)) == EOF)
  {
      perror("READ_WRITE(): cannot open file for read");
      exit(11);
  }
#else
  if((fp = open(filename,O_RDONLY)) == EOF)
  {
      perror("READ_WRITE(): cannot open file for read");
      exit(11);
  }
#endif
```

Programa-exemplo de escritas em arquivos

```
  do
  {
    #ifdef PC
        if((rtnval = read(fp,&my,sizeof(my))) < 0)
        {
            perror("READ_WRITE(): cannot read from file");
            exit(12);
        }
        if( rtnval == 0 )
        {
            fprintf(stderr,"\nEnd Of File Reached");
            break;
        }
    #else
        if((rtnval = read(fp,buf,sizeof(my))) <0)
        {
            perror("READ_WRITE(): cannot write to file");
            exit(7);
        }
```

Programa-exemplo de escritas em arquivos

```
      if( rtnval == 0 )
      {
            fprintf(stderr,"\nEnd Of File Reached");
            break;
      }
      sscanf(buf,"%s %s %s %s %s %s %d %d %d“, my.name, my.street,
         my.city, my.state, my.zip, my.ssn, &my.age, &my.height,
         &my.weight);
    #endif
    showdata(&my);
    strcpy(ans," ");
    printf("\nContinue(Y/N)? ");
    gets(ans);
  }  while(!strcmp(ans,"y"));
  close(fp);
  return 0;
}
```

Programa-exemplo de escritas em arquivos

```
int err_handler(FILE *fileptr, char *filename, int exitnum)
{
  char errmsg[80];
  if ferror(fileptr)
  {
      sprintf(errmsg,"ERROR - cannot access file:%s",filename);
      putchar('\n');
      perror(errmsg);
      clearerr(fileptr);
      exit(exitnum);
  }
  if feof(fileptr)
  {
      sprintf(errmsg,"End of File reached on file:%s", filename);
      putchar('\n');
      perror(errmsg);
      clearerr(fileptr);
      return(1);
  }
  return 0;
}
```