CURSO SUPERIOR DE ADS

Arquivos

Prof. Fernando Marlon Soares Figueiredo

Disciplina: Programação Estruturada

SECTRAS

ADS

Aula de Hoje

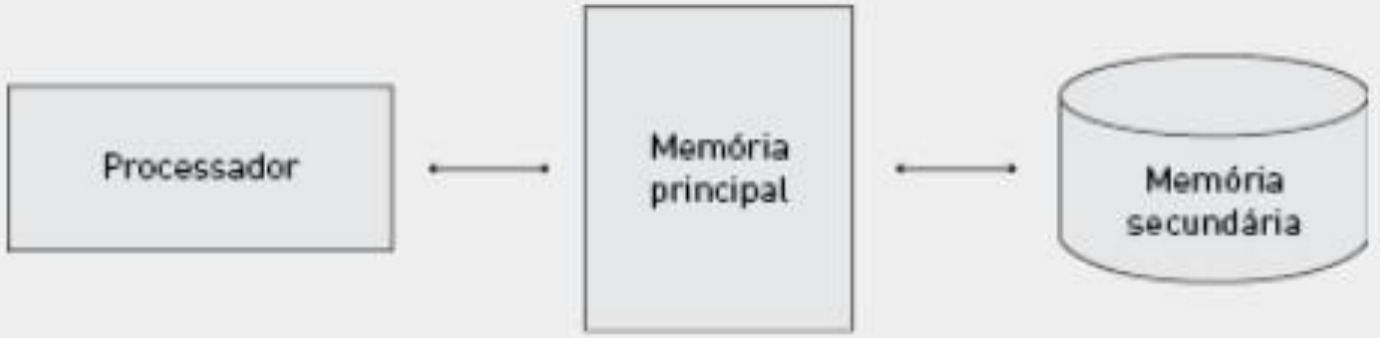
- Por que utilizar arquivos?
- Sintaxe para manipulação de arquivos.
- Arquivos de texto x Arquivos binários.
- Mais exercícios.

Introdução

- Com o conteúdo estudado até então, todos os dados manipulados durante os programas só existem enquanto eles estão rodando na máquina. No momento em que o programa é encerrado, as informações são perdidas.
- **Arquivos** possibilitam que o programador salve os dados para que sejam usados posteriormente, seja pelo próprio ou por outro programa.

Conceitos Básicos

- Arquivos possibilitam o **armazenamento permanente de dados**, os quais são alocados na memória secundária (HDs, SSDs, Pendrives). Dessa forma, programas conseguem ler e escrever dados armazenados entre as memórias primárias e secundárias.
- Um dado pode ser lido de um arquivo e carregado para a memória principal ou ser escrito da memória principal para um arquivo.



Conceitos Básicos

- Observe que a relação que havia entre o processador e a memória principal continua, mas agora a memória principal também se comunica com a secundária para acesso aos arquivos.
- Arquivos são construídos por uma sequência de bytes contendo toda informação necessária.
- Por exemplo, um arquivo que armazena 2 inteiros possui ao todo 8 bytes sequenciais (supondo que cada inteiro ocupa 4 bytes).

Conceitos Básicos

- Entretanto, a menos que se conheça informações prévias sobre este arquivo, não há como saber que de fato ele armazena dois inteiros, 8 caracteres ou 1 double (8 bytes).
- Este fato ocorre porque em arquivos **não há separador de informações entre os dados**.
- Existem dois tipos principais de arquivos: de **texto** e **binários**.
- **Arquivos de texto** interpretam cada byte como caractere, enquanto **arquivos binários** podem conter qualquer combinação de tipos de dados.

Manipulando arquivos

- Arquivos são manipulados em linguagem C por meio de um **ponteiro** para uma estrutura específica do tipo **FILE.**
- A sintaxe para criação desse tipo de dado é:

FILE *nome_da_variável;

- Onde **nome** corresponde ao identificador da variável associada ao arquivo.
- Para ter acesso à estrutura FILE, é necessário incluir a biblioteca de entrada e saída de dados **stdio.h**

Abrindo um arquivo

- A partir do momento em que a variável de controle do arquivo é criada, antes de manipular o arquivo diretamente, é necessário que ele seja aberto pelo comando **fopen**.

```
pArq = fopen(nome_do_arquivo, modo_de_leitura);
```

Onde:

- **pArq** corresponde ao nome da **variável ponteiro de arquivo**.
- **nome do arquivo** corresponde à string com o **caminho de leitura e sua extensão do arquivo** a ser aberto, o caminho pode ser relativo ou absoluto.
- **modo de leitura** indica como o arquivo deve ser aberto

Caminho Relativo ou Absoluto

pArq = fopen(**nome_do_arquivo**, modo_de_leitura);

- Ainda sobre o nome do arquivo, o caminho relativo é quando ele é acessado a partir do executável. O caminho absoluto é quando todo caminho de acesso é fornecido.
- Além disso, o retorno da função deve ser recebido por uma variável do tipo criado na declaração de arquivos. Caso o endereço de retorno seja uma constante chamada NULL, ocorreu algum tipo de erro na abertura do arquivo.

O modo de leitura

pArq = fopen(nome_do_arquivo, **modo_de_leitura**);

O modo de leitura controla como o programa pode ter acesso ao arquivo, podendo ser manipulado **apenas para escrita**, **apenas para leitura** ou **ambos**.

Modos de Leitura

- **r** - Leitura (read): arquivo aberto só pode ser lido. O arquivo precisa existir.
- **w** - Escrita (write): arquivo aberto só pode ser escrito. Caso o arquivo exista, ele será sobrescrito (arquivo prévio de mesmo nome é apagado)
- **a** - Anexação (append): arquivo aberto só pode ser escrito. Caso o arquivo exista, o cursor é posicionado ao final do arquivo, mantendo o conteúdo existente. Se o arquivo não existir, ele é criado.

Modos de leitura

- **r+** - Leitura e atualização: arquivo aberto pode ser lido e escrito. O arquivo precisa existir.
- **w+** - Escrita e atualização: arquivo aberto pode ser escrito e lido. Caso o arquivo exista, ele será sobrescrito (arquivo prévio de mesmo nome é apagado).
- **a+**- Anexação e atualização: arquivo aberto pode ser escrito e lido. Caso o arquivo exista, o cursor é posicionado ao final do arquivo, mantendo o conteúdo existente. Se o arquivo não existir, ele é criado.

Modos de leitura para arquivos binários

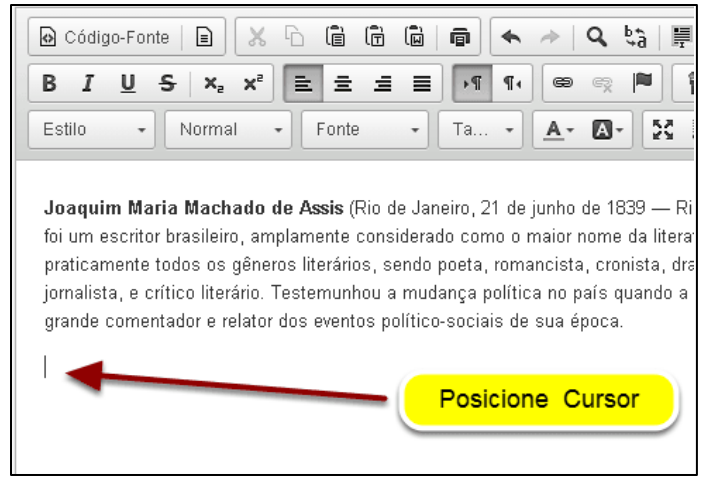
- Caso o programador deseje abrir um arquivo binário, basta colocar o sufixo b em cada um dos modos, ou seja, utilizar os modos: “rb”, “wb”, “ab”, “r+b”, “w+b”, “a+b”.
- Lembrando que em caso de erro na abertura (como, por exemplo, o arquivo não existir no modo de leitura) é retornado o valor NULL.

Fechando um arquivo

- Da mesma forma que um arquivo é aberto, ele deve ser sempre fechado após seu uso.
- Para fechar um arquivo já aberto, utiliza-se da função **fclose**:

```
fclose(pArq);
```

O cursor interno



- Existe um cursor interno na variável contendo sua **posição atual**, que costuma começar na posição zero, ou seja, no início do arquivo.
- A cada vez que um conteúdo é lido ou escrito nesse arquivo, este cursor é incrementado para que fique na posição correta.
- Existe 3 funções relevantes associadas ao cursor: ftell, fseek e rewind
- Todas elas pedem que a variável passada seja do tipo ponteiro de FILE.

1. Posição do cursor: ftell

- A primeira função recebe a variável e retorna um inteiro contendo a posição atual do cursor e possui a seguinte sintaxe:

```
pos_cursor = ftell(pArq);
```

2. Alterando a posição do cursor: fseek

- A segunda função permite alterar a posição do cursor conforme os parâmetros especificados. Sua sintaxe é:

fseek (pArq, deslocamento, **origem**);

- Onde **deslocamento** indica o número de bytes que se deseja movimentar o cursor; e **origem** indica em qual ponto do arquivo o cursor deve se localizar para que seja movimentado com o deslocamento.
- Para controlar a origem, podem ser utilizadas três constantes.
  - SEEK_SET: cursor parte do início do arquivo.
  - SEEK_CUR: cursor parte de onde já se encontra.
  - SEEK_END: cursor parte do final do arquivo.

O deslocamento

fseek (pArq, **deslocamento**, origem);

- O deslocamento pode ser também negativo, tornando possível começar do final do arquivo e ir decrementando para o início, por exemplo.

3. Voltando o cursor para o início

- A função rewind serve para voltar o cursor par ao início do arquivo e possui a seguinte sintaxe.

```
rewind(pArq);
```

Exemplo

```c
#include <stdio.h>

int main()
{
    FILE *arq;
    int posCursor;

    arq = fopen("nomes.txt", "r");

    if(arq == NULL)
    printf("Erro ao abrir o arquivo!\n");
    else
    {
        posCursor = ftell(arq);
        printf("Posicao do cursor: %d\n", posCursor);
        fseek(arq, 3, SEEK_SET);
        posCursor = ftell(arq);
        printf("Posicao do cursor: %d\n", posCursor);
        rewind(arq);
        posCursor = ftell(arq);
        printf("Posicao do cursor: %d\n", posCursor);

        fclose(arq);
    }

    return 0;
}
```

Explorando o Código

- O código do slide anterior abre um arquivo de texto chamado “nomes.txt” no modo leitura e modifica a posição do cursor pelas funções estudadas.
- É necessário que o arquivo de texto que iremos utilizar esteja na mesma pasta do código fonte.

feof

- Outra função relevante é a feof, que verifica o cursor para ver se o arquivo chegou ao final, ou seja, contém o símbolo especial de final de arquivo chamado EOF.
- A função possui a seguinte sintaxe:

```
chegouFinal = feof(pArq);
```

- A variável chegouFinal recebe valor zero enquanto o caractere a ser lido não é o caractere de final de arquivo.

feof

- Uma forma comum de utilizar esta função é junto com o while, quando precisamos ler um arquivo e não sabemos previamente quantas linhas ele tem.
- Utilizamos comumente o seguinte trecho de código a seguir para este fim, indicando que o laço while continuará sendo executado até que o final do arquivo não seja atingido:

```
...
while ( !feof(pArq) ) {
...
}
```

Arquivos de Texto
&
Arquivos Binários

Apesar de arquivos serem compostos por uma sequência de bytes, o modo como estes bytes são interpretador pode ser de duas formas, classificando-os como arquivos de texto ou binário.

Arquivos de Texto

- Arquivos cuja interpretação dos dados é feita byte a byte, correspondendo a caracteres.
- Podem ser lidos e modificados por programas de edição de texto.
- Alguns exemplos de arquivos do tipo texto são: texto, código fonte e tabelas no formato Comma-separated Values (CSV).

fprintf

- A função de escrita principal para este tipo de arquivo é muito semelhante à utilizada para saída de dados em linha de comando, possuindo a seguinte sintaxe:

fprintf(pArq, “caracteres e tipos”, var 1, ..., varN);

- A função fprintf é análoga a printf, exceto pelo primeiro parâmetro, que recebe a variável do tipo ponteiro para o arquivo.

Funções de Entrada

Para as funções de entrada, entretanto, temos três funções importantes, dependendo da forma como os dados desejam ser lidos:

- caract = fgetc(pArq); //por caractere
- fgets(nome, tamanho, pArq); //linha
- fscanf(pArq, “caracteres e tipos”, &var1, ..., &varN); //formatada

fgetc

- caract = fgetc(pArq); //por caractere
- A operação de leitura por caractere retorna cada um dos caracteres lidos para a variável caract, atualizando o cursor em uma posição.

fgets

- fgets(nome, tamanho, pArq); //linha
- A função fgets retorna uma leitura por linha. Note que aqui estamos alterando o terceiro parâmetro para o ponteiro do arquivo a ser lido (em vez da constante stdin, como visto anteriormente). A função lê toda uma linha do arquivo ou até tamanho-1 (o que ocorrer primeiro) e retorna NULL em caso de erro.

fscanf

- fscanf(pArq, “caracteres e tipos”, &var1, ..., &varN); //formatada
- A leitura formatada inclui uma variação da função scanf, agora requerendo que o ponteiro do arquivo seja passado como argumento durante sua chamada.

Exemplo

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define TAMANHO_LINHA 100
5
6 int main( )
7 {
8    FILE *entrada, *saida;
9    char texto[TAMANHO_LINHA];
10
11    entrada = fopen("dados.txt", "r");
12
13    if(entrada == NULL)
14    printf("Erro ao abrir arquivo de entrada.\n");
15    else
16    {
17    saida = fopen("resultados.txt", "w");
18    if(saida == NULL)
19    printf("Erro ao abrir arquivo de saida.\n");
20    else
21    {
22     while(fgets(texto, TAMANHO_LINHA, entrada) != NULL)
23     {
24      printf("%s", texto);
25      fprintf(saida, "%s", texto);
26     }
27     fclose(saida);
28   }
29   fclose(entrada);
30 }
31
32 return 0;
33 }
```

Explorando o código

- O programa anterior trabalha com 2 tipos de arquivos, um somente para leitura (entrada) e outro somente para escrita (saída), e consiste em copiar o conteúdo do arquivo de entrada para o de saída, além de imprimir esse conteúdo lido na tela.
- Observe que a cada vez que um arquivo é aberto, deve-se testar se o ponteiro retornado é NULL, indicando erro. É importante também fechar todos os arquivos abertos com a função fclose.
- As linhas 22-26 correspondem à parte de cópia efetivamente.
- A linha 20 inicia um iterador utilizando o retorno da função fgets como critério de parada, ou seja, enquanto o retorno ao ler uma linha é diferente de NULL (quando ele for nulo, indica o fim do arquivo).

Explorando o código

- Cada linha lida é armazenada na variável texto, a qual é impressa na tela (linha 24) e escrita no arquivo (linha 25).
- Para que não ocorra erro durante a execução do programa, é fundamental que o usuário tenha criado um arquivo chamado dados.txt na mesma pasta do programa.
- Ao final da execução, deverá existir também na mesma pasta um arquivo chamado “resultado.txt”.
- Em geral, costuma-se nomear os arquivos texto com o sufixo “txt”, indicando que se trata de um arquivo texto personalizado.

Arquivos Binários

- Arquivos cuja interpretação dos dados não é conhecida, necessitando de um conhecimento prévio dos dados armazenados e sua respectiva ordem para que seja corretamente interpretado.
- Consistem na maioria dos tipos de arquivos salvos por programas, incluindo arquivos de imagem (formatos jpg, png, gif), áudio e vídeo (mp3, mp4, avi), edição de texto (doc, odt, pdf), savegames de jogos eletrônicos, etc.

Arquivos Binários

- Diferente dos arquivos de texto, em que há formas alternativas para escrita e leitura de caracteres, arquivos binários possuem apenas uma função para escrita e outra para leitura, pois consistem em escrever dados (conjunto de bytes) em determinada ordem.
- A função de escrita é dada pela seguinte sintaxe:

```
fwrite(&dados, sizeof(tipo), qtd_elementos, pArq);
```

fwrite(&dados, sizeof(tipo), qtd_elementos, pArq);

- &dados - O primeiro parâmetro consiste no endereço da variável que contém os dados a serem colocados no arquivo binário. Esta variável pode ser de qualquer tipo (inclusive estruturas, por exemplo), dependendo do tipo de dado que o programador quer armazenar.
- sizeof(tipo) - O segundo parâmetro pede quantos bytes o tipo dessa variável ocupa. Como esse número pode variar dependendo de fatores externos do ambiente em que o programa está rodando, recomenda-se o uso dessa função que retorna automaticamente o tamanho em bytes do tipo de dados da variável dados.

fwrite(&dados, sizeof(tipo), qtd_elementos, pArq);

- qtd_elementos – O terceiro pede quantos elementos serão armazenados desse tipo, sendo útil para armazenar uma quantidade específica de dados que estejam em um vetor, por exemplo. Neste caso também pode-se utilizar a função sizeof passando a variável como parâmetro para descobrir quantos bytes o seu tipo ocupa.
- pArq – É a variável do ponteiro de arquivo com o arquivo já aberto para que o conteúdo seja escrito.

Função de Leitura

- A função de leitura possui sintaxes análoga à de escrita

```
fread(&dados, sizeof(tipo), qtd_elementos, pArq);
```

- Neste caso, o endereço da variável dados ao final da operação de leitura conterá os dados lidos.

Exemplo

Linha 27 é o vetor leAposta[i];

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int aposta[6] = {8, 14, 25, 32, 47, 59};
7     int le Aposta[6] = {0};
8     int i;
9     FILE *arq;
10
11      arq = fopen("aposta.bin", "w+b");
```

```
13     if(arq == NULL)
14     printf("Erro ao abrir arquivo de entrada.\n");
15     else
16     {
17       fwrite(aposta, size of(int), 6, arq);
18       printf("Aposta escrita:");
19       for(i =0; i <6; i++)
20           printf("%d", aposta[i]);
21
22       rewind(arq);
23
24       fread(le Aposta, size of(int), 6, arq);
```

```
26       for(i =0; i <6; i++)
27           printf("%d", aposta[i]);
28
29       fclose(arq);
30     }
31
32     return 0;
33 }
```

Explorando o Código

- O programa mostra a escrita e leitura de uma aposta de loteria.
- Dois vetores do tipo inteiro são criados, um contendo as apostas, especificadas diretamente no código, e outro de mesmo tamanho inicializado com zero, que serve para ler as apostas escritas no arquivo.
- A linha 17 contém o código referente à escrita, enquanto as linhas seguintes imprimem na tela os valores apostados. Como o cursor após a operação de escrita se encontra ao final do arquivo, a função rewind é utilizada para voltar o cursor ao início do código, pois agora tudo que foi escrito é lido com a função fread (linha 24).

Explorando o Código

- As linhas seguintes novamente imprimem na tela a aposta, mostrando os mesmos valores na tela, agora ocupados pelo vetor leAposta (antes zerado).
- Por fim, o arquivo é fechado e o programa é encerrado.
- O arquivo chamado “aposta.bin” deve estar localizado na mesma pasta do programa executável. Em geral, costuma-se nomear as variáveis binárias co,m a extensão “.bin”, indicando que o arquivo é um binário customizado.

Questões

Questão 1

Na manipulação de arquivos em linguagem C, a função fopen () abre um arquivo, retornando o ponteiro associado ao arquivo, fazendo uso da seguinte sintaxe:

FILE *p;

p = fopen (nome_do_arquivo, modo_de_abertura);

Observe a instrução:

FILE *prova;

prova = fopen (“arquivo.dat", “wb+");

Pode-se afirmar que se realizou:
A) a criação de um arquivo binário chamado arquivo.dat, em que poderão ser realizadas operações de leitura e de escrita.
B) a criação de um arquivo chamado prova, em que poderão ser realizadas somente as operações de leitura.
C) a criação de um arquivo binário chamado prova, em que poderão ser realizadas somente as operações de escrita.
D) a criação de um arquivo chamado arquivo.dat, em que poderão ser realizadas operações de leitura.

Questão 1

Resposta: A

Na manipulação de arquivos em linguagem C, a função fopen () abre um arquivo, retornando o ponteiro associado ao arquivo, fazendo uso da seguinte sintaxe:

FILE *p;

p = fopen (nome_do_arquivo, modo_de_abertura);

Observe a instrução:

FILE *prova;

prova = fopen (“arquivo.dat", “wb+");

Pode-se afirmar que se realizou:
A) a criação de um arquivo binário chamado arquivo.dat, em que poderão ser realizadas operações de leitura e de escrita.
B) a criação de um arquivo chamado prova, em que poderão ser realizadas somente as operações de leitura.
C) a criação de um arquivo binário chamado prova, em que poderão ser realizadas somente as operações de escrita.
D) a criação de um arquivo chamado arquivo.dat, em que poderão ser realizadas operações de leitura.

Questão 2

Qual código é usado para abrir um arquivo chamado "dados.txt" em modo de leitura em C?

a) fopen("dados.txt", "r");

b) fopen("dados.txt", "read");

c) open("dados.txt", "r");

d) file_open("dados.txt", "read");

Questão 2 Resposta: A

Qual código é usado para abrir um arquivo chamado "dados.txt" em modo de leitura em C?

a) fopen("dados.txt", "r");

b) fopen("dados.txt", "read");

c) open("dados.txt", "r");

d) file_open("dados.txt", "read");

Questão 3

Qual função em C é usada para mover a posição de leitura/escrita dentro de um arquivo?

a) seek

b) move

c) position

d) fseek

Questão 3

Resposta: D

Qual função em C é usada para mover a posição de leitura/escrita dentro de um arquivo?

a) seek

b) move

c) position

d) fseek

Exercícios

Exercício 1

- Faça um programa que conte a quantidade de caracteres de um arquivo texto já existente com nome “teste.txt”.
- Imprima esse valor encontrado na tela.
- Lembre-se de criar o arquivo e escrever algum texto nele para teste.

Exercício 1 - Resposta

```c
#include <stdio.h>

int main()
{
    FILE *arq;
    char caract;
    int cont;

    arq = fopen("teste.txt", "r");
     if(arq == NULL)
     printf("Erro ao abrir o arquivo. \n");
     else
     {
      cont =0;
      while(! feof(arq))
      {
       caract = fgetc(arq);
       cont++;
      }

      fclose(arq);

      printf("Qtd de caracteres do texto: %d\n", cont);
    }

    return 0;
}
```

Exercício 2

- Faça um programa que transcreva diversos valores reais positivos digitados por um usuário para um novo arquivo texto a ser criado, onde cada valor deve ficar em uma linha.
- Na última linha do arquivo, o valor total da soma desses valores deve ser colocado.
- O nome do novo arquivo criado deve ser escolhido pelo usuário no início do programa.

Exercício 2 - Resposta

```c
#include <stdio.h>

int main()
{
   FILE *arq;
   char nome[50];
   float valor =0, soma=0;


    printf("Digite o nome do arquivo texto:");
    fgets(nome, 50, stdin);

    arq = fopen(nome, "w");
    if(arq == NULL)
     printf("Erro ao abrir o arquivo. \n");
    else
    {
     while(valor >= 0)
     {
       printf("Digite um valor (negativo p/ sair):");
       scanf("%f", &valor);

       if(valor >= 0)
       {
         fprintf(arq, "%f \n", valor);
         soma = soma + valor;
       }
     }

       fprintf(arq, "%f", soma);

       fclose(arq);
    }

    return 0;
}
```

Exercício 3

- Fala um programa que armazene um vetor já preenchido de cinco inteiros em um arquivo binário chamado “nums.bin”.
- Em seguida leia este mesmo arquivo de trás pra frente, imprimindo na tela seus valores.

Exercício 3 - Resposta

```c
#include <stdio.h>

int main( )
{
    FILE *arq;
    int nums[5] = {5, 8, 3, 4, 10}, i;
    int aux;

    arq = fopen("nums.bin", "w+b");
     if(arq == NULL)
       printf("Erro ao abrir o arquivo . \ n");
     else
     {
      fwrite(nums, size of(int), 5, arq);

     for(i =0; i <5; i++)
     {
       fseek(arq,– size of(int) *(i+1),SEEK_END);
       fread(&aux, size of(int), 1, arq);
       printf("Valor %d: %d\n", i, aux);
     }

     fclose(arq);
   }

   return 0;
}
```

Exercício 4

- Faça um programa que coloque uma mensagem personalizada em uma nova linha ao final de um arquivo texto por meio de uma função.
- Esta função deve receber o nome do arquivo texto, além da mensagem personalizada e retornar 0 em caso de falha ou 1 em caso de sucesso.
- Peça o nome do arquivo e a mensagem para o usuário no programa principal.

Exercício 4 - Resposta

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

int msgFimArq(char nomeArq[ ], char msg[ ]);

int msgFimArq(char nomeArq[ ], char msg[ ])
{
   FILE *arq;
   int sucesso;

    arq = fopen(nomeArq, "a");
   if(arq == NULL)
     sucesso = 0;
   else
   {
     fprintf(arq, "\n%s", msg);

     fclose(arq);

     sucesso = 1;
   }

   return sucesso;
}

int main( )
{
   char nome[50];
   char msg[200];
   int sucesso;

   printf("Digite o nome do arquivo:");
   fgets(nome, 50, stdin);
   nome[strlen(nome) -1] = '\0';
   printf("Digite a mensagem:");
   fgets(msg, 200, stdin);
   msg[strlen(nome) -1] = '\0';
   sucesso = msgFimArq(nome, msg);

   if(sucesso == 1)
     printf("Arquivo anexado com sucesso! \n");
   else
     printf("Falha ao abrir o arquivo! \n");

   return 0;
}
```

Exercício 5

- Faça um programa que defina um tipo estruturado chamado Atleta, contendo o número de identificação, idade e peso de um atleta.
- As informações de cinco atletas estão armazenadas nesta ordem, com um atleta por linha e entre vírgulas, em um arquivo texto já criado chamado “atletas.txt”.
- O programa deve conter uma função que leia os dados dos atletas desse arquivo texto e coloque-os em um vetor com cinco elementos do tipo atleta.
- No programa principal, armazene os valores recebidos no vetor em um arquivo binário chamado “atletas.bin”. A função não deve possuir retorno.

Exercício 5 - Resposta

```c
#include <stdio.h>

#define MAX_ATLETAS 5

typedef struct
{
    int id;
    int idade;
    float altura;
} Atleta;

void caractAtleta(Atleta atletas[ ]);

void caractAtleta(Atleta atletas[ ])
{
    FILE *arq;
    int i;

    arq = fopen("atletas.txt", "r");
    if(arq == NULL)
      printf("Erro ao abrir o arquivo texto. \n");
    else
    {
      for(i =0; i <MAX_ATLETAS; i++)
        fscanf(arq, "%d,%d,% f \n", &atletas[i]. id,&atletas[i].
       idade,&atletas[i].altura);

      fclose(arq);
    }
}

int main( )
{
    FILE *arq ;
    Atleta atletas[MAX_ATLETAS];

    caractAtleta(atletas);

    arq = fopen("atletas.bin", "wb");
    if(arq == NULL)
      printf("Erro ao abrir o arquivo binario. \n");
    else
{
    fwrite(atletas, sizeof(Atleta),MAX_ATLETAS, arq);

    printf("Arquivo binario criado com sucesso. \n");

    fclose(arq);
  }

  return 0;
}
```