



TÉCNICAS DE PROGRAMAÇÃO

Aula 10 – Manipulação arquivos em C/C++





Objetivos de Aprendizagem

1. Identificar as formas de aberturas de um arquivo, bem como sua manipulação;
2. Identificar os princípios básicos das formas de construção de programas para excluir e listar usando arquivo;
3. Identificar os princípios básicos das formas de construção de programas para localizar e alterar usando arquivo;
4. Desenvolver programas com arquivo.




O que são arquivos?

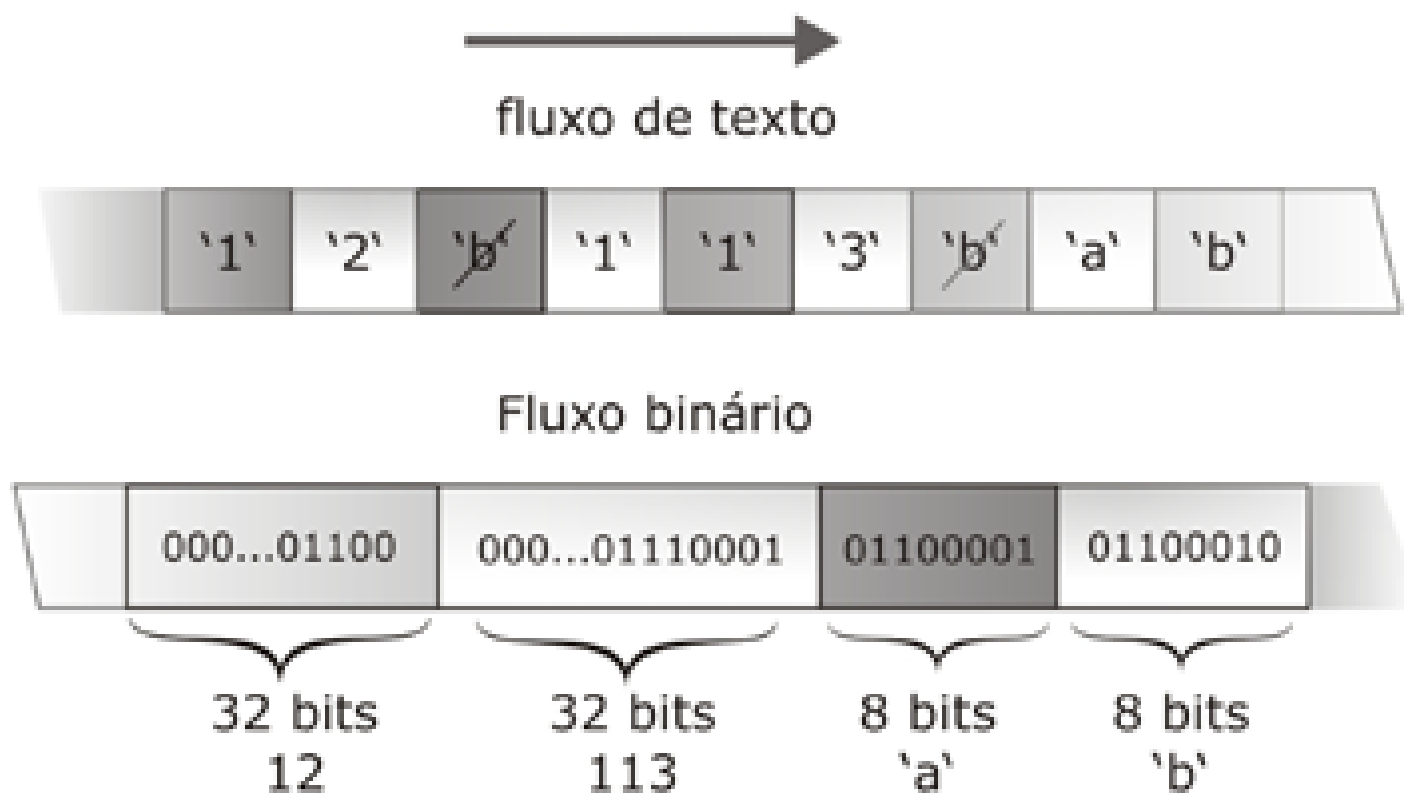
- Os arquivos são estruturas de dados manipuladas fora do ambiente do programa.
 - Considera-se como ambiente do programa a memória principal, onde nem sempre é conveniente manter certas estruturas de dados.
 - De modo geral, os arquivos são armazenados na memória secundária, como, por exemplo: disco rígido (HD - *hard disk*), *CD* e *pendrive*.
- 



Como os arquivos são organizados?


- ▶ A linguagem C utiliza o conceito de fluxo de dados (***stream***) para manipular os vários tipos de dispositivos de armazenamento e seus diferentes formatos.
 - ▶ Os dados podem ser manipulados em dois diferentes tipos de fluxos: fluxos de texto e fluxos binários.
 - ▶ Um fluxo de texto (*text stream*) é composto por uma *sequência* de caracteres, que pode ou não ser dividida em linhas, terminadas por um caractere de final de linha.
 - ▶ Um fluxo binário (*binary stream*) é composto por uma sequência de *bytes*, que são lidos, sem tradução, diretamente do dispositivo externo.
- 

Fluxo de Dados






Fluxo de Dados

- No fluxo de texto, os dados são armazenados como caracteres sem conversão para a representação binária.
 - Cada um dos caracteres ocupa um *byte*. *O número 12 ocupa dois bytes e o número 113 ocupa 3 bytes.*
 - Um caractere em branco foi inserido entre cada um dos números para separá-los, de modo que a função de entrada e saída possa descobrir que são dois números inteiros (12 e 113) e não o número 12113.
 - No fluxo binário, cada número inteiro ocupa *32 bits (4 bytes)* e é armazenado na forma binária. Observem que, em arquivos binários, não há necessidade de separar os números já que eles sempre ocupam *32 bits*.
 - Os arquivos binários são utilizados quando queremos armazenar registros completos.
 - Com estes arquivos, poderemos acessar qualquer registro de forma mais rápida.
 - No caso dos arquivos texto, quando queremos encontrar alguma informação, temos que fazer uma varredura seqüencial no arquivo, tornando a busca pouco eficiente.
- 



Ponteiros

- Um ponteiro é um tipo de variável que armazena um endereço de memória.
 - Nós já vimos variáveis que armazenam números inteiros, números reais e caracteres.
 - Ao trabalhar com arquivos, precisamos saber em qual endereço de memória o arquivo está armazenado.
 - O endereço de memória onde o arquivo está armazenado será colocado em uma variável que armazena endereço de memória (ponteiro).
- 




Ponteiros

- Para declarar uma variável que é capaz de armazenar um endereço de memória, usamos a seguinte sintaxe:

Sintaxe

```
tipo *nome_do_ponteiro;
```

- **tipo:** Tipo de variável que o ponteiro armazena endereço. Podemos dizer que nosso ponteiro armazena o endereço de uma variável inteira, real, caractere, etc. Para este capítulo, estaremos utilizando um ponteiro que vai armazenar o endereço de um arquivo.
 - *****: o asterisco na frente do nome de uma variável simboliza que a variável que está sendo declarada é um ponteiro.
 - **nome_do_ponteiro:** daremos um nome à nossa variável que armazena endereços.
- 

Declaração de ponteiros para arquivos

```
FILE *p aluno;  
FILE *p produto;
```

- Na linha 1, temos a declaração do ponteiro *p aluno*, que irá armazenar o endereço de um arquivo (FILE).
- Devemos colocar o FILE em maiúsculo. Na linha 2, temos a declaração de outro ponteiro, *p produto*, que armazena o endereço de um *FILE*.
- Quando queremos inicializar uma variável real ou inteira, atribuímos zero às mesmas.
- Quando precisarmos inicializar um ponteiro, devemos atribuir: NULL. Quando um ponteiro armazena NULL, quer dizer que ele não está armazenando um endereço no momento.
- Tenham calma, daqui a pouco o conceito de ponteiros ficará mais claro.
- Vamos começar a conhecer os comandos de manipulação de arquivos binários e entenderemos melhor onde o conceito de ponteiro será aplicado.



Comandos para manipular arquivos Binários

- Como já mencionado, os arquivos binários são utilizados quando queremos armazenar registros.
- É como se tivéssemos um vetor de registro, só que os dados não são perdidos ao terminar a execução do programa.




Representação gráfica de um arquivo binário




Comandos para manipular arquivos Binários

- O arquivo binário é formado por um conjunto de registros, armazenados um após o outro. As operações realizadas em um arquivo binário dependerão do local onde se encontrar o **leitor**.
- Pense no leitor como se fosse a agulha de uma vitrola.
- Para tocar uma música, a agulha passa sobre o disco, fazendo a leitura da música.
- Dependendo de onde a agulha seja posicionada, é tocada uma música do disco.





Comandos para manipular arquivos Binários

- ▶ O leitor do arquivo passará sobre os registros, fazendo a leitura dos mesmos.
 - ▶ Nós podemos colocar o leitor sobre qualquer registro e executar uma operação sobre o mesmo (leitura e/ou gravação).
 - ▶ O arquivo tem uma marcação indicando onde ele termina (*end of file*).
 - ▶ As operações para a manipulação de arquivo estão na biblioteca *stdio.h*. Com isso, ao trabalhar com arquivos, devemos incluir esta biblioteca nos nossos programas.
- 



Declaração de um ponteiro para arquivo

- Na linguagem C, as funções que manipulam arquivos trabalham com o conceito de ponteiros para arquivo.
- Com isso, teremos uma variável que armazenará o endereço de memória onde está armazenado nosso arquivo.
- Devemos declarar um ponteiro para cada arquivo que formos manipular no nosso programa.



Comando para abrir um arquivo

- A maior parte das operações sobre um arquivo (leitura, gravação, etc) só pode ser executada com o arquivo aberto. O comando de abertura do arquivo (*fopen*):

Sintaxe


```
ponteiro_arquivo = fopen("nome_do_arquivo", "modo_de_abertura");
```

- ponteiro_arquivo:** Ao abrir um arquivo, a função *fopen* retorna o endereço de memória do arquivo. Por conta disso, devemos atribuir o endereço de memória do arquivo, para um ponteiro. Após o arquivo ser aberto, usaremos este endereço de memória para acessá-lo.
- nome_do_arquivo:** Determina qual arquivo deverá ser aberto. Neste espaço é colocado o nome do arquivo, da forma que foi salvo no diretório. O comando *fopen* procura o arquivo que desejamos abrir, no mesmo diretório onde está armazenado o programa executável.
- modo_de_abertura:** Informa que tipo de uso você vai fazer do arquivo. O modo de abertura indica o que faremos no arquivo: leitura, gravações e alterações. Além disso, o modo de abertura também vai indicar se queremos que um novo arquivo seja criado.



Modos de Abertura


Modo	Significado
"r+b"	Abre um arquivo binário para leitura e escrita. O arquivo deve existir antes de ser aberto. Dessa forma, este modo de abertura só pode ser usado se o arquivo que estamos querendo abrir já existe no nosso computador.
"w+b"	Cria um arquivo binário para leitura e escrita. Se o arquivo não existir, ele será criado. Se já existir, o conteúdo anterior será destruído. Este modo de abertura tem a capacidade de criar novos arquivos. Mas, se solicitarmos que seja aberto um arquivo que já existe, o conteúdo do arquivo será apagado.
"a+b"	Acrescenta dados ou cria um arquivo binário para leitura e escrita. Caso o arquivo já exista, novos dados podem ser adicionados, apenas, no final do arquivo. Se o arquivo não existir, um novo arquivo será criado.





Abertura do Arquivo

```
FILE *paluno, *pprofessor;  
paluno = fopen("alunos.bin", "w+b");  
pprofessor = fopen("professores.bin", "r+b");  
if (pprofessor == NULL)  
    printf("Arquivo de professores não existe");
```

- Para saber se um arquivo foi aberto corretamente, podemos fazer um teste como mostra as linhas 4 e 5.
 - Após executar um *fopen*, verificamos se o ponteiro está com *NULL*. Caso afirmativo, é porque a abertura não foi executada corretamente.
 - Uma vez aberto, o arquivo fica disponível para leituras e gravações através da utilização de funções adequadas.
- 

Comando para fechar um arquivo

- Após terminar de usar um arquivo, devemos fechá-lo. Para isso, usamos a função *fclose*, que tem a seguinte sintaxe.

Sintaxe

```
fclose(ponteiro_arquivo);
```


- **ponteiro_arquivo:** É o ponteiro que tem armazenado o endereço de memória do arquivo que queremos fechar.


Só podemos fechar arquivos que estão abertos!




Fechamento de arquivo

```
fclose(palunos);  
fclose(pprofessores);
```

- Na linha 1, fechamos o arquivo que está armazenado no endereço de memória *palunos* (que é o ponteiro que armazena o endereço do arquivo).
 - Nós só utilizamos o nome do arquivo (o nome que está salvo no diretório), no momento da abertura. Depois de aberto, só utilizamos seu endereço de memória, certo?
- 



Comando para ler um registro armazenado no arquivo


- A leitura é feita a partir do ponto onde o leitor se encontra.
 - Ao abrir o arquivo, o leitor é posicionado no início do primeiro registro.
 - À medida que vamos executando leituras, o leitor vai se deslocando.
 - Dessa forma, precisamos ter noção da posição onde o leitor se encontra.
 - Nos arquivos binários, SEMPRE fazemos a leitura de um registro completo, independente de quantos campos ele tenha.
 - Um arquivo deve armazenar registro do mesmo tipo. Se os registros são do mesmo tipo, cada um deles ocupará o mesmo espaço de memória.
 - Dessa forma, o leitor sempre se deslocará em intervalos regulares.
- 



Leitura do arquivo

sintaxe

```
fread (&registro, numero_de_bytes, quantidade,  
ponteiro_arquivo);
```

- **®istro:** É o registro que armazenará os dados do registro lido do arquivo. É assim, nós pegamos um registro que está no arquivo, e armazenamos no registro passado como parâmetro.
 - **numero_de_bytes:** O que o comando de leitura faz é informar que o leitor precisa se deslocar, fazendo a leitura de uma certa quantidade de bytes. A quantidade de bytes que o leitor deve se deslocar é exatamente quantos bytes de memória o registro ocupa.
- 

Leitura do arquivo

sintaxe

```
fread (&registro, numero_de_bytes, quantidade,  
ponteiro_arquivo);
```

- **quantidade:** Neste terceiro parâmetro, usaremos **SEMPRE 1**. Este parâmetro significa quantas vezes a leitura será executada. Nós sempre queremos que seja feita a leitura de um registro por vez. Por isso, sempre utilizaremos 1.
- **ponteiro_arquivo:** Precisamos informar em qual arquivo a leitura será feita. Com isso, passamos o endereço de memória onde o arquivo que será lido está armazenado. Assim, utilizamos o ponteiro que tem o endereço do arquivo.



Exemplo

```
typedef { int matricula;  
          float nota1, nota2, media;  
        } TAluno  
  
TAluno aluno;  
  
FILE *paluno;  
  
fread(&aluno, sizeof(TAluno), 1, paluno);
```



Arquivo Binário



Posição do leitor antes de executar o fread



Posição do leitor após de execução do fread



Comando para gravar um registro no arquivo

- Para gravar um registro em um arquivo binário, nós utilizamos o comando *fwrite*.
- *Este comando é muito parecido com o fread.*
- A diferença é que o *fread* lê uma sequência de bytes no arquivo, e armazena em um registro (primeiro parâmetro do *fread*).
- E o *fwrite*, pega um registro, e armazena suas informações no arquivo.




Comando para gravar um registro no arquivo

- Vamos ver a sintaxe do *fwrite*:

Sintaxe

```
fwrite(&registro, numero_de_bytes, quantidade, ponteiro_arquivo);
```

- **®istro:** é o registro que será armazenado no arquivo.
- **numero_de_bytes:** é a quantidade de *bytes que serão* gravadas no arquivo. Neste parâmetro, também usaremos o *sizeof*.
- **quantidade:** neste terceiro parâmetro, também, usaremos SEMPRE 1. Este parâmetro significa quantas vezes a gravação será executada. Nós sempre queremos que seja feita a gravação de um registro por vez. Por isso, sempre utilizaremos 1.
- **ponteiro_arquivo:** precisamos informar em qual arquivo a gravação será feita. Assim, utilizamos o ponteiro que tem o endereço do arquivo.



Exemplo: Leitura de um registro do arquivo

```
typedef { int matricula;  
          float nota1, nota2, media;  
        } TAluno  
  
TAluno aluno;  
  
FILE *paluno;  
  
fwrite(&aluno, sizeof(TAluno), 1, paluno);
```

Comando para posicionar o leitor em um ponto do arquivo

- As operações de leitura e gravação são feitas na posição onde o leitor se encontra no momento. Podemos mudar a posição do leitor, colocando-o em um ponto específico do arquivo.

Sintaxe


```
fseek(ponteiro_arquivo, numero_de_bytes, origem);
```

- **ponteiro_arquivo:** Precisamos informar em qual arquivo o leitor está sendo posicionado.
- **numero_de_bytes:** é a quantidade de *bytes que o leitor irá se deslocar pelo arquivo, até chegar no local desejado.*
- **origem:** determina a partir de onde os *número_de_bytes de deslocamento do leitor serão contados.*



Origem

Origem	Significado
SEEK_SET	O deslocamento do leitor será contado a partir do início do arquivo.
SEEK_CUR	O deslocamento do leitor será contado a partir da sua posição corrente.
SEEK_END	O deslocamento do leitor será contado a partir do final do arquivo.



Exemplo: Comando *fseek*

```
fseek(paluno, 0, SEEK_END);
```


```
fseek(paluno, 2*sizeof(paluno), SEEK_SET);
```



Posição do leitor após o fseek da linha 1



Posição do leitor após a execução do fseek da linha 2



Comando para posicionar o leitor no início do arquivo

- Nós podemos posicionar o leitor no início do arquivo, utilizando o comando *rewind*.


Sintaxe

```
rewind(ponteiro_arquivo);
```

- Onde:
 - **ponteiro_arquivo:** é o ponteiro que tem o endereço do arquivo que queremos posicionar o leitor.

Exemplo

```
rewind(paluno);
```





Comando para verificar se chegou ao final do arquivo

- Quando fazemos uma varredura no arquivo, não sabemos quantos registros tem armazenados no mesmo.
- No entanto, precisamos saber o momento de parar de executar a leitura dos registros.
- O comando *feof (end of file) informa se o leitor chegou ao final do arquivo ou não.*



Comando para verificar se chegou ao final do arquivo

Sintaxe

```
int feof(ponteiro_arquivo);
```

- **int: é o retorno da função.** A função *feof* retorna um número inteiro, que indica se o arquivo terminou ou não. Quando a função retorna zero, significa que ainda não chegou no final do arquivo. Qualquer valor diferente de zero, indica que chegou ao final do arquivo.
- **ponteiro_arquivo:** é o ponteiro que tem o endereço do arquivo que queremos verificar se chegou ao fim.



Exemplo

- Normalmente, o *feof* é usado como condição de um *while*. Este *while* será executado enquanto não chegar no final do arquivo.

```
while (feof(paluno)==0)
```





Comando para remover um arquivo

- Quando desejamos apagar um arquivo do diretório, podemos utilizar o comando *remove*.
- *Este comando só pode ser utilizado em arquivos fechados.*

Sintaxe

```
remove("nome_do_arquivo");
```

- **nome_do_arquivo:** é o nome do arquivo que queremos apagar. Neste caso, é utilizado o nome do arquivo salvo no diretório do nosso computador.

Exemplo

```
remove("alunos.bin");
```





Comando para renomear um arquivo

- Nós podemos renomear um arquivo no diretório do nosso computador. Para isso, utilizamos o comando *rename*, só pode ser utilizado em arquivos fechados.

Sintaxe

```
rename("nome_do_arquivo", "novo_nome_do_arquivo");
```

- **nome_do_arquivo:** é o nome do arquivo que queremos renomear. Neste caso; é utilizado o nome do arquivo salvo no diretório do nosso computador.
- **novo_nome_do_arquivo:** é o novo nome para o nosso arquivo.

Exemplo

```
rename("alunos.bin", "alunos_temp.bin");
```

Neste exemplo, o arquivo *alunos.bin*, será renomeado para *alunos_temp.bin*.





Implementação das operações básicas em um arquivo

- Mais uma vez, teremos um programa cheio de detalhes, bem maior do que os que fizemos até então.
- Para facilitar o entendimento deste programa, que tem quase 200 linhas, o mesmo foi dividido em 8 partes.






Parte 1

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
typedef struct{
    int mat;
    char nome[20];
    float med;
} TAluno;

FILE *paluno;
TAluno aluno_aux, aluno_nulo;
void linha(){
    int i;
    for (i=1; i<=80; i++)
        printf("_");
}
```

```
void cabec() {
    system("cls");
    printf("Faculdade Joaquim
Nabuco");
    linha();
}

void abre_arquivo(){
    paluno = fopen("aluno.dat",
    "r+b");
    if (paluno == NULL)
        paluno =
        fopen("aluno.dat", "w+b");
}
```



Parte 2

```
void inserir(){
    int resp;
    do {
        cabec();
        printf("\n\nCadastrar
novo aluno\n\n");
        printf("\nMatricula: ");
        scanf("%d",
&aluno_aux.mat);
        printf("\nNome.....: ");
        fflush(stdin);
        gets(aluno_aux.nome);
        printf("\nMedia.....: ");
```

```
        scanf("%f", &aluno_aux.med);
        fseek(paluno, 0,
SEEK_END);
        fwrite(&aluno_aux,
sizeof(TAluno), 1, paluno);
        printf("\n\nAluno
cadastrado com sucesso!\n\n");
        printf("\nDeseja
cadastrar outro: (1-sim/0-nao)?
");
        scanf("%d", &resp);
    } while (resp ==1);
}
```

Parte 3

```
int procura(int matp) { int p;  
    p = 0;  
    rewind(paluno);  
    fread(&aluno_aux, sizeof(TAluno), 1,  
paluno);  
    while (feof(paluno)==0) {  
        if (aluno_aux.mat == matp)  
            return p;  
        else {  
            fread(&aluno_aux, sizeof(TAluno),  
1, paluno);  
            p++;  
        }  
    }  
    return -1;  
}
```


```
}  
}  
return -1;  
}  
  
void mostre(int pos) {  
    fseek(paluno, pos*sizeof(TAluno),  
SEEK_SET);  
    fread(&aluno_aux, sizeof(TAluno), 1,  
paluno);  
    printf("\n\n");  
    linha();  
    printf("Matricula Nome Media\n");  
    linha();  
    printf("%9d %-20s %5.1f\n",  
aluno_aux.mat, aluno_aux.nome,  
aluno_aux.med);  
    linha();  
}
```




Parte 4

```
void consultar(){
    int resp, matcon, posicao;
    do{
        cabec();
        printf("\n\nConsultar
Aluno\n\n\n");
        printf("Matricula do aluno: ");
        scanf("%d", &matcon);
        posicao = procura(matcon);
        if (posicao == -1)
            printf("\n\nMatricula não
encontrada!\n\n");

        else
            mostre(posicao);
        printf("\n\nDeseja consultar
outro (1-sim/0-nao)? ");
        scanf("%d", &resp);
    } while (resp == 1);
}
```



Parte 5

```
void remover(){
int matrem, conf, resp, posicao;
aluno_nulo.mat = 0;
aluno_nulo.med = 0;
do{ cabec();
printf("\n\nRemover aluno\n\n\n");
printf("Matricula: ");
scanf("%d", &matrem);
posicao = procura(matrem);
if (posicao == -1)
printf("\nAluno nao encontrado!!\a");
```

```
else { mostre(posicao);
printf("\n\nDeseja remover o aluno
(1-sim/0-nao)? ");
scanf("%d", &conf);
if (conf == 1){
fseek(paluno,posicao*sizeof
(TAluno),SEEK_SET);
f w r i t e ( & a l u n o _ n u l o ,
sizeof(TAluno), 1, paluno);
printf("\n\nAluno removido com
sucesso!");
}
else
printf("\nRemocao cancelada!");
}
printf("\n\n\nDeseja remover outro
(1-sim/0-nao)? ");
scanf("%d", &resp);
} while (resp ==1);
}
```

Parte 6

```
void alterar()
{ int matalt, conf, resp, posicao;
do { cabec();
printf("\n\nAlterar media do aluno\n\n");
printf("Matricula: ");
scanf("%d", &matalt);
posicao = procura(matalt);
if (posicao == -1)
printf("\nAluno,nao encontrado!!\a");
else
{ mostre(posicao);
printf("\n\nAlterar a media do
aluno(1-sim/0-nao)? ");
scanf("%d", &conf);
if (conf == 1)
{ printf("\nNova media: ");
scanf("%f", &aluno_aux.med);
printf("\nMedia alterada com sucesso!
\n\n");
```


```
fseek(paluno,posicao*sizeof(TAluno),
SEEK_SET);
fwrite(&aluno_aux,sizeof(TAluno), 1,
paluno);
mostre(posicao);
printf("\nMedia do aluno alterada com
sucesso!\n");
}
Else
printf("\n\nAlteracao cancelada!\n\n");
}
printf("\n\nDeseja alterar outro (1-
sim/0-nao)? ");
scanf("%d", &resp);
}while (resp ==1);
}
```



Parte 7

```
void listagem(){
cabec();
printf("\n\nListagem
Geral\n\n\n");
linha();
printf("Matricula Nome
Media\n");
linha();
rewind(paluno);
fread(&aluno_aux,
sizeof(TAluno), 1, paluno);
```

```
while (feof(paluno)==0){
if (aluno_aux.mat != 0)
printf("%9d %-20s %5.1f\n",
aluno_aux.
mat,
aluno_aux.nome,
aluno_aux.med);
fread(&aluno_aux,
sizeof(TAluno), 1, paluno);
}
linha();
printf("tecle enter para
voltar ao menu...");
getche();
}
```



Parte 8

```
main(){
int op;
abre_arquivo();
do{
cabec();
printf("\n\nOpcoes: \n\n\n");
printf(" 1- Cadastrar novo aluno\n\n");
printf(" 2- Remover aluno\n\n");
printf(" 3- Consultar aluno por
matricula\n\n");
printf(" 4- Alterar media do
aluno\n\n");
printf(" 5- Listagem geral\n\n");
printf(" 0- Sair\n\n");
linha();
printf("Informe a opcao desejada: ");
scanf("%d", &op);
```

```
switch(op){
case 1: inserir(); break;
case 2: remover(); break;
case 3: consultar(); break;
case 4: alterar(); break;
case 5: listagem(); break;
case 6: limpar(); break;
case 0: fclose(paluno); break;
default: printf("\n\n\taOpcao
invalida!");
break;
}
} while (op != 0);
}
```