

Controle de Acesso Acadêmico de Alunos

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

#define MAX_ALUNOS 300
#define MAX_OCORRENCIAS 300
#define MAX_CHAMADAS 300
```

```
typedef char dataRef[11];
```

```
typedef struct {
    char matricula[20];
    char nome[50];
    char periodo[90];
    char tipoOcorrencia[10];
    char datahora[20];
} Acesso;
```

```
typedef struct {
    char matricula[20];
    char nome[50];
    char periodo[20];
} Aluno;
```

```
typedef struct {
    char matricula[20];
    char tipoOcorrencia[10];
} Ocorrencia;
```

```
// Variáveis Globais
```

D	S	T	O	O	S	S
D	L	M	M	J	V	S

Aluno listaAlunos [MAX_ALUNOS];

Correncia listaCorrencias [MAX_CORRENCIAS];

Acesso filaChamada [MAX_CHAMADA];

int tamanhoLista = 0;

int tamanhoCorrencias = 0;

int inicioChamada = 0;

int fimChamada = 0;

bool validarData (const char *data) {

int anos, meses, dias;

if (bscanf (data, "%d-%d-%d", &anos, &meses, &dias) != 3) {

return false;

}

if (meses <= 0 || meses > 12 || dias <= 0 || dias > 31) {

return false;

}

return true;

}

void mostrarListaAlunos () {

printf ("Lista de Alunos:\n");

for (int i = 0; i < TamanhoLista; i++) {

printf ("Matrícula: %d, Nome: %s, Período: %d\n", listaAlunos[i].matricula, listaAlunos[i].nome, listaAlunos[i].periodo);

}

}

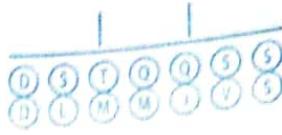
void obterData (char *data) {

do {

printf ("Digite a data de hoje (aaaa-mm-dd): ");

scanf ("%s", data);





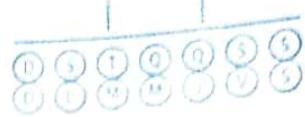
```
    } while (!validarData(data));  
}
```

```
int contarAlunos (const Aluno lista[], int tamanhoLista) {  
    int contador = 0;  
    for (int i = 0; i < tamanhoLista; i++) {  
        if (lista[i].matricula[0] != 'V') {  
            contador++;  
        }  
    }  
    return contador;  
}
```

```
void adicionarNaChamada (const Aluno *aluno) {  
    if ((fimChamada + 1) > MAX_CHAMADA == inicioChamada) {  
        printf ("A chamada este feita. Não é possível adicionar mais  
alunos.\n");  
    } else {  
        strcpy (filaChamada[fimChamada].matricula, aluno->matricula);  
        strcpy (filaChamada[fimChamada].nome, aluno->nome);  
        strcpy (filaChamada[fimChamada].periodo, aluno->periodo);  
        fimChamada = (fimChamada + 1) > MAX_CHAMADA;  
    }  
}
```

```
void mostrarChamada (const char *data) {  
    printf ("Chamada para a data: %s\n", data);  
  
    int i = inicioChamada;  
    while (i != fimChamada) {  
        printf ("Matrícula: %s, Nome: %s, Período: %s\n", filaChamada[i].  
matricula, filaChamada[i].nome, filaChamada[i].periodo);  
        i++;  
    }  
}
```

Jandaia



```
i = (i + 1) % MAX_CHAMADA;  
}
```

```
void registrarOcorrencia (const char *matricula, const char *tipoOcorrencia,  
                           const char *data) {  
    if (tamanhoOcorrencias < MAX_OCORRENCIAS) {  
        strcpy (listaoocorrencias [tamanhoOcorrencias].matricula, matricula);  
        strcpy (listaoocorrencias [tamanhoOcorrencias].tipoOcorrencia, tipoOcorrencia);  
        tamanhoOcorrencias++;  
        printf ("Ocorrência registrada com sucesso para a matrícula  
                %s.\n", matricula);  
    } else {  
        printf ("A lista de ocorrências está cheia. Não é possível adicionar  
               mais ocorrências.\n");  
    }  
}
```

```
void alterarTipoOcorrencia (char tipoOcorrencia) {  
    printf ("Selecione o tipo de ocorrência: \n");  
    printf ("1. Esqueceu \n");  
    printf ("2. Perdeu \n");  
    printf ("3. Não Possui \n");  
    printf ("4. Outros \n");  
    printf ("Escolha uma opção: ");  
    int tipoEscolhido;  
    scanf ("%d", &tipoEscolhido);
```

```
switch (tipoEscolhido) {  
    case 1:  
        strcpy (tipoOcorrencia, "Esqueceu");
```



break;

case 2:

strcpy (tipoOcorrência, "Reclui");

break;

case 3:

strcpy (tipoOcorrência, "Não Possui");

break;

case 4:

strcpy (tipoOcorrência, "Outros");

break;

default:

printf ("Opção Inválida. A ocorrência será registrada como 'Outros'. \n");

strcpy (tipoOcorrência, "Outros");

break;

}

}

void reiniciarOcorrências () {

// Define o tamanho de ocorrências de volta para zero

tamanhoOcorrências = 0;

printf ("Ocorrências reiniciadas com sucesso. \n");

}

void reiniciarPrograma () {

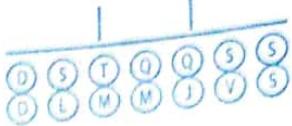
// Reinicia todas as variáveis globais

tamanhoLista = 0;

tamanhoOcorrências = 0;

inícioChamada = 0;

fimChamada = 0;



```
// Re inicialize suas listas ou estruturas conforme necessário
for (int i = 0; i < MAX_ALUNOS; i++) {
    listaAlunos[i].matricula[0] = '10'
    listaAlunos[i].nome[0] = '10'
    listaAlunos[i].periodo[0] = '10'
}
```

```
for (int i = 0; i < MAX_OCORRENCIAS; i++) {
    listaOcorrencias[i].matricula[0] = '10';
    listaOcorrencias[i].tipoOcorrencia[0] = '10';
}
```

```
for (int i = 0; i < MAX_CHAMADA; i++) {
    filaChamada[i].matricula[0] = '10';
    filaChamada[i].nome[0] = '10';
    filaChamada[i].periodo[0] = '10';
}
```

```
printf("Programa reiniciado com sucesso.\n");
```

```
bool validarPeriodo (char *periodo) {
    if (strcmp(periodo, "M") == 0 || strcmp(periodo, "V") == 0 ||
        strcmp(periodo, "N") == 0) {
        return true;
    }
    return false;
}
```

```
void obterInformacaoAluno (char *matricula, char *nome, char
    *periodo) {
    printf("Digite a matrícula de novo aluno: ");
    scanf ("%s", matricula);
```



D S T Q A S
I L M H J V S

printf("Digite o nome

```
getchar(); // Limpar o buffer de entrada antes de usar fgets.  
fgets(nome, sizeof(listaAlunos[0].nome), stdin);  
nome[strlen(nome) - 1] = '\0';  
do {  
    printf("Digite o período do novo aluno: \n");  
    printf("Digite M para matutino \n");  
    printf("Digite V para vespertino \n");  
    printf("Digite N para noturno \n");  
    scanf("%s", periodo);  
} while (!validarPeriodo(periodo));  
}
```

```
void cadastrarAluno(const char *matricula, const char *nome, const  
char *periodo) {  
    if (tamanhoLista < MAX_ALUNOS) {  
        strcpy(listaAlunos[tamanhoLista].matricula, matricula);  
        strcpy(listaAlunos[tamanhoLista].nome, nome);  
        strcpy(listaAlunos[tamanhoLista].periodo, periodo);  
        tamanhoLista++;  
        printf("Aluno cadastrado com sucesso! \n");  
    } else {  
        printf("A lista de alunos está cheia. Não é possível  
cadastrar mais alunos. \n");  
    }  
}
```

```
bool buscarAlunoPorMatricula(const char *matricula, Aluno *alunoEncontrado)  
{  
    for (int i = 0; i < tamanhoLista; i++) {  
        if (strcmp(listaAlunos[i].matricula, matricula) == 0) {  
            *alunoEncontrado = listaAlunos[i];  
        }  
    }  
}
```

Jandaia

return true;

}

return false;

}

void removerAluno (const char *matricula) {

int indiceAluno = -1; // Índice do aluno a ser removido.

// Procura o índice do aluno na lista

for (int i = 0; i < tamanhoLista; i++) {

if (strcmp (listaAlunos[i].matricula, matricula) == 0) {

indiceAluno = i;

break;

}

}

if (indiceAluno != -1) {

// Move os elementos para preencher o espaço do aluno removido.

for (int i = indiceAluno; i < tamanhoLista - 1; i++) {

strcpy (listaAlunos[i].matricula, listaAlunos[i + 1].matricula);

strcpy (listaAlunos[i].nome, listaAlunos[i + 1].nome);

strcpy (listaAlunos[i].periodo, listaAlunos[i + 1].periodo);

}

// Decrementa o tamanho da lista.

tamanhoLista --;

printf ("Aluno com matrícula %s não encontrado.\n", matricula);

}

}

```
void editarAlunoRemoverAluno (Aluno* lista, int  
tamanhoLista) {  
    char matricula [20];  
    printf ("Digite a matrícula do aluno a ser  
editado: ");  
    scanf ("%s", matricula);
```

```
int encontrado = -1; // índice do aluno encon-  
troado
```

```
for (int i = 0; i < tamanhoLista; i++) {  
    if (strcmp (lista [i].matricula, matricula)  
        == 0) {
```

encontrado = i;

```
printf ("Aluno encontrado: Matrícula:  
%s, Período: %dm"; lista [i].matricula, lista  
[i].nome, lista [i].periodo);
```

```
int opcao;  
printf ("Escolha a opção: \n");  
printf ("1. Editar nome\n");  
printf ("2. Editar turma\n");  
printf ("3. Remover aluno\n");  
printf ("Opção: ");  
scanf ("%d", &opcao);
```

Switch (opção) {

Case 1:

```
    printf("Digite o nome: ");
    scanf("%s", listaEncontrado.name);
    break;
```

Case 2:

do {

```
    printf("Digite o novo periodo da tur-
```

```
: \n");
```

```
    printf("Digite M para matutino\n");
    printf("Digite V para vespertino\n");
    printf("Digite N para noturno\n");
    scanf("%s", listaEncontrado.periodo);
```

```
    while (!ValidarPeriodo(listaEncontrado.periodo));
    scanf("%s", listaEncontrado.T.periodo);
    break;
```

Case 3:

```
rememberAluno(matricula);
```

```
break;
```

default:

```
    printf("Opção invalida.\n");
```

3 }
3 }

if (encontrado == -1) {

```
    printf("Aluno com matricula %s não encon-  
trado.\n", matricula);
```

3 }

3 }

Vaid matr^{er}(Correncia, PorMatricula(estrutura
*matricula)) {
 printf("Correncias do aluno com matrícula
%s: \n", matricula);
 const int i = 0; i < tomento(Correncias); i++) {
 if (strcmp(listaCorrencias[i].matricula
matricula) == 0) {
 printf("Tipo de Ocorrência: %s, Data:
%s \n", listaCorrencias[i].tipoOcorrencia);
 }
 }
}

Vaid adicionarCorrencia(estrutura
*matricula,
const char *tipoOcorrencia, const char *data) {
 if (tomento(Correncias) < MAX_OCORRENCIAS) {
 // Adiciona na lista de Ocorrências chamada
 Aluno alunoEncontrado;
 if (buscaAlunoPorMatricula(matricula, &
alunoEncontrado)) {
 adicionarNaChamada(&alunoEncontrado);
 // Adiciona na lista de Ocorrências
 registerOcorrencia(matricula, tipoOcorrencia,
data);
 }
 }
}

Estrutura
printf ("A lista de Ocorrências está
cheia. Não é possível adicionar mais Ocorrências.
%s\n",);

void chamadaGeral(const char* data) {
 if (tamanhoLista == 0) {
 printf("Vc não possui alunos cadastrados.\n");
 return;
}

3

char tipo(Carrenca[10]);
for (int i = 0; i < tamanhoLista; i++) {
 printf("Matrícula: %.3s, Nome: %.15s, Período: %.5s\n", listaAlunos[i].matricula, listaAlunos[i].name, listaAlunos[i].periodo);
}
printf("Gostaria de adicionar uma crença para este aluno? (1 para sim, 0 para não): ");
int escolha;
scanf("%d", &escolha);
if (escolha == 1) {
 alterarTipoCarrenca(tipo(Carrenca),
 adicionarCrenca(&listaAlunos[i].matricula, tipo(Carrenca), data));
}

3

3

3

```
int main() {  
    int escolha;  
    int app = 1;  
    char matricula[20];  
    char nome[30];  
    char tipoAvalencia[20];  
    char data[20];  
    char periodo[20];
```

Alema alternativa;

While (app == 1) {

```
    printf("Início : \n");  
    printf("1. Consultar aluno \n");  
    printf("2. Buscar aluno \n");  
    printf("3. Informar nota aluno \n");  
    printf("4. Inserir acomadaria \n");  
    printf("5. Montar ministro de novo professor");  
    printf("6. Editar / remover aluno \n");  
    printf("7. Montar acomadaria \n");  
    printf("8. Reiniciar acomadaria \n");  
    printf("9. Reiniciar programa \n");  
    printf("0. Sair \n");  
    printf("Escolha uma opção: ");  
    scanf("%d", &escolha);
```

```
Switch (escola) {
```

```
    case 1:
```

```
        if (tamanhoLista == 0) {
```

```
            printf ("Você não possui alunos cadastrados!");
```

```
    } else {
```

```
        obterData(data);
```

```
        lerMatricula (matricula1);
```

```
        mostrarAluno (data);
```

```
}
```

```
    break;
```

```
    case 2:
```

```
        if (tamanhoLista == 0) {
```

```
            printf ("Você não possui alunos cadastrados!");
```

```
    } else {
```

```
        printf ("Digite a matrícula do aluno a ser buscado: ");
```

```
        scanf ("%s", matricula1);
```

```
        if (buscarAlunoPorMatricula (matricula1, alunosEncontrados))
```

```
            printf ("Aluno encontrado! Em Matrícula %s,"
```

```
                nome: %s, periodo: %s.\n",
```

```
                alunosEncontrados.matricula, alunosEncontrados
```

```
.nome, alunosEncontrados.periodo);
```

```
    } else {
```

```
        printf ("Aluno com matrícula %s não  
        encontrado. Em "Matrícula");
```

```
}
```

```
}
```

```
break;
```



Exercício 3:

```
if (tamanhoLista < MAX_ALUNOS) {  
    obterInformacoesAluno (matricula, nome, periodo);  
    cadastrarAluno (matricula, nome, periodo);  
}  
else {  
    printf ("A lista de alunos está cheia. Não é  
    possível cadastrar mais alunos. ln");  
}  
break;
```

Exercício 4:

```
if (tamanhoLista == 0) {  
    printf ("você não possui alunos cadastrados.");  
}  
else {  
    printf ("Digite a matrícula do aluno: ");  
    scanf ("%d", matricula);  
    printf ("Digite o tipo de ocorrência: ");  
    scanf ("%s", tipoOcorrencia);  
    registrarOcorrencia (matricula, tipoOcorrencia, data);  
}  
break;
```

(11)

case 5:

```
    printf ("Número de alunos cadastrados: %d\n";
            *contAlunos (*listAlunos, *contAlunos, *contMatr);
    break;
```

case 6:

```
if (*contAluno == 0) {
    printf ("Você não inseriu aluno cadastrado. \n");
    3 che {
```

```
    char matricula[20];
```

```
    printf ("Digite a matrícula do aluno a ser
            editado: ");
```

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

```
if (*contAluno >= *contMatr) {
    printf ("Aluno com matrícula (%s, %s),",
            nome, periodo);
```

```
    nome = NULL, periodo = NULL, alunos[*contAluno].name,
    alunos[*contAluno].periodo);
```

```
    editarAluno (*listAlunos, *contAlunos, *contMatr);
```

```
    3 che {
```

```
    printf ("Aluno com matrícula %s não encon-
            trado. \n", matricula);
```

```
    3
```

```
    break;
```

Case 7:

```
if (tamanhoList == 0) {
```

printf ("você não possui alunos cadastrados!\n");
} else {

```
mostrarListaAlunos();
```

```
escreverMatricula[20];
```

printf ("Digite a matrícula do aluno para
mostrar a correnteio: ");

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

```
if (buscaMatricula (matricula, &correnteio))
```

```
{ mostrarCorrenteio (matricula, &correnteio);
```

} else *

printf ("Aluno com matrícula %s não
encontrado. In", matricula);

}

}

```
break;
```

case 8:

```
reiniciarCorrenteios();
```

```
break;
```

case 9:

```
reiniciarPrograma();
```

```
break;
```

case 0:

```
printf ("Finalizando o programa. In");
```

```
app = 0;
```

```
break;
```



default:

3 prevent ("Operação inválida. Tente novamente.");

3

Return 0;

3