

# Estruturas ou Registros

---

DCC 119 – Algoritmos



# *Estruturas heterogêneas*



- Até agora vimos as estruturas de dados homogêneas: **vetores**, **matrizes** e **strings**.
- Nestas estruturas todos os elementos da estrutura são de tipos de dados primitivos: **inteiro**, **real**, **caractere**.

- No entanto, em muitos casos, necessitamos armazenar conjuntos de informações relacionadas, formados por diversos tipos de dados primitivos.
- Exemplos:
  - Endereço;
  - Fichas com dados pessoais de um cliente;
  - Fichas com dados de um produto.

# Variáveis compostas

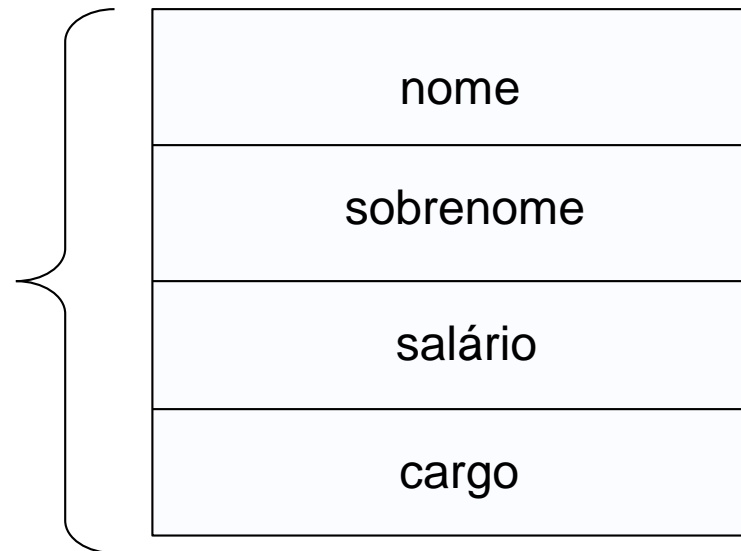


- Quando uma determinada estrutura de dados for composta por **diversos tipos diferentes**, primitivos ou não, temos um conjunto heterogêneo de dados.
- Essas variáveis são chamadas de **variáveis compostas heterogêneas**.
- Estas variáveis compostas são chamadas de **estruturas** (ou **structs** na linguagem C).

# Definição de estrutura

- Uma estrutura pode ser definida como uma coleção de uma ou mais variáveis relacionadas (campos), **onde cada variável pode ser de um tipo distinto.**

- Exemplo:  
  
empregado  
(4 campos)



# Estruturas - Definição



- Sintaxe para definir uma estrutura com **n** campos em C:

```
struct NomeDaEstrutura  
{  
    tipo1  identificador1;  
    tipo2  identificador2;  
    . . .  
    tipoN  identificadorN;  
};
```

# Estruturas - Declaração



- No entanto, dessa forma, é necessário usar a palavra-chave **struct** toda vez que uma variável é declarada:

```
struct <nomeEstrutura> var1, ..., varN;
```

- Exemplo:

```
// Definicao da estrutura
struct Aluno
{
    char nome[100];
    int matricula;
    int idade;
    char cidade[50];
    char sexo;
};
```

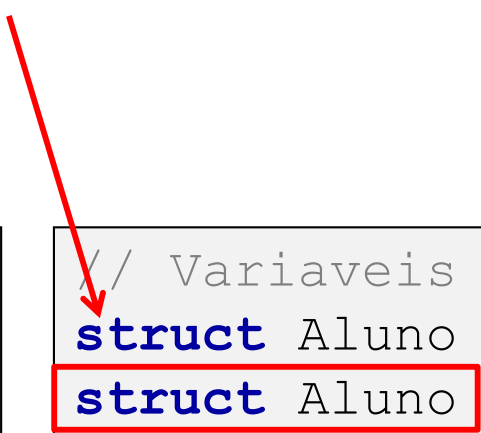
```
// Declaracao de variaveis
struct Aluno fulano;
struct Aluno a1, a2, a3;
```

# Estruturas - Declaração

- Note que para variáveis do tipo estrutura a palavra chave **struct** deve vir antes do nome da estrutura.

```
// Tipos primitivos  
int valor;  
float x, y, z;  
char letra;
```

```
// Variaveis estruturas  
struct Aluno fulano;  
struct Aluno a1, a2, a3;
```



- Existe uma outra possibilidade, mais simples, para declarar variáveis de estruturas.



# Comando *typedef*



- O comando **typedef**, pode ser usado para dar um novo nome para algum tipo de dados.
- Uso: **typedef** <nomeAntigo> <nomeNovo>;
- Exemplos:

```
typedef int Inteiro;  
typedef float Real;
```

- Declarando variáveis

```
Inteiro n;  
Real x;
```

# Estruturas - Declaração

- Assim, com o comando **typedef**, pode-se definir estruturas e declarar variáveis da seguinte forma:

**typedef** **struct** est\_Aluno Aluno;

Nome antigo

Nome novo

- Exemplo

```
// Definicao da estrutura
struct est_Aluno
{
    char nome[100];
    int matricula;
    int idade;
    char cidade[50];
    char sexo;
};
```

```
typedef struct est_Aluno Aluno;

// Declaracao de variaveis
Aluno a1, a2, a3;
```

# Estruturas - Declaração



- A seguinte forma também pode ser utilizada:

```
// Definicao da estrutura
typedef struct
{
    char nome[100];
    int matricula;
    int idade;
    char cidade[50];
    char sexo;
} Aluno;

// Declaracao de variaveis
Aluno a1, a2, a3;
```

Esta será a forma adotada na disciplina.

# Definição e Declaração



- A **definição** de um tipo estrutura deve ficar **fora** do programa principal (**main**) e de qualquer função.
- A **declaração** de uma variável do tipo estrutura deve ficar **dentro** de alguma função (seja ela a **main** ou outra qualquer).

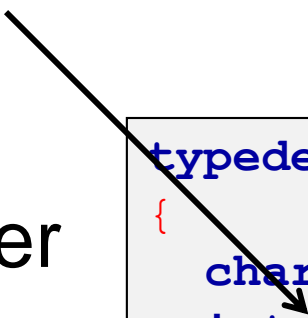
```
#include <stdio.h>

typedef struct
{
    char nome[100];
    int matricula;
    int idade;
    char cidade[50];
    char sexo;
} Aluno;

int main()
{
    Aluno a1, a2, a3;
    // ...
    // processamento
    // ...
    return 0;
}
```

# Estruturas: Manipulação

- Cada campo de uma estrutura pode armazenar um valor (como uma variável). Seu valor pode ser acessado, modificado, impresso, etc.
- Campos são acessados usando o operador de acesso **ponto (.)** entre o **nome da variável** declarada como estrutura e o **nome do campo**.



```
typedef struct
{
    char nome[100];
    int matricula;
    int idade;
    char cidade[50];
    char sexo;
} Aluno;
```

# Estruturas: Manipulação



- No programa são criadas duas variáveis **Aluno**.
- Cada variável **Aluno** (**a1** e **a2**) tem nome, matricula, idade, cidade e sexo.
- O uso de **a1.idade** permite acessar ou modificar o campo **idade** da variável **a1**.

```
typedef struct
{
    char nome[100];
    int matricula;
    int idade;
    char cidade[50];
    char sexo;
} Aluno;
```

```
int main()
{
    Aluno a1, a2;

    gets(a1.nome);
    a1.idade = 20;
    printf("Matricula %d", a2.matricula);
    // ...
    return 0;
}
```

# Estruturas: Manipulação

- Neste exemplo, serão lidos matrícula e nome de um funcionário.

```
#include <stdio.h>
typedef struct {
    char matricula[15];
    char nome[100];
    int setor;
} Funcionario;

int main()
{
    Funcionario f1;
    gets(f1.matricula);
    gets(f1.nome);
    scanf("%d", &f1.setor);
    puts("Informacoes lidas:");
    puts(f1.matricula);
    puts(f1.nome);
    printf("%d\n", f1.setor);
    return 0;
}
```

- 1) Defina uma estrutura para representar as informações de um cartão de crédito. Você precisa apenas criar a estrutura com os dados e tipos apropriados. Não é necessário criar um programa para utilizá-la.
- 2) Defina uma estrutura chamada `medidas` para representar o peso a altura de uma pessoa.
- 3) Considerando a estrutura do exercício (2) e a declaração:  

```
medidas joao, maria;
```

  - Escreva instruções para atribuir: 1.78 à altura de `joao`; 1.64 à altura de `maria`; 75 ao peso de `joao`; 59 ao peso de `maria`.
  - Escreva um conjunto de instruções para imprimir a média das alturas e a média dos pesos de `joao` e `maria`.



# Estruturas aninhadas



- **Definição** de uma estrutura onde um de seus campos é outra estrutura:

```
typedef struct
{
    char nome[50];
    int idade;
    endereco end;
} ficha_pessoal;
```

Repare que um campo do tipo **endereco** é declarado dentro de **ficha\_pessoal**.

- **Declaração** de variáveis do tipo definido acima:

```
ficha_pessoal ficha1, ficha2;
```

# Estruturas aninhadas



```
typedef struct {  
    int dia,mes,ano;  
} Data;  
  
typedef struct {  
    char nome[150];  
    Data nascimento;  
} Aluno;  
  
int main() {  
    Aluno aluno1;  
    ...  
}
```

**aluno1**

The diagram illustrates the memory layout of the `aluno1` variable. It is represented by a large light blue rectangle. Inside this rectangle, at the top, are the labels `nome` and `nascimento`. Below `nascimento`, there is a smaller green rectangle. Inside this green rectangle are the labels `dia`, `mes`, and `ano`, representing the nested `Data` structure.

**nome**  
**nascimento**  
    **dia**  
    **mes**  
    **ano**

# Estruturas aninhadas



```
typedef struct {  
    int dia,mes,ano;  
} Data;  
  
typedef struct {  
    char nome[150];  
    Data nascimento;  
} Aluno;  
  
int main() {  
    Aluno aluno1;  
    gets(aluno1.nome);  
    ...  
}
```

**aluno1**

The diagram illustrates the memory layout of the `aluno1` variable. It is represented as a large light blue rectangle. Inside this rectangle, at the top, are the labels `nome` and `nascimento`. Below `nascimento`, there is a smaller green rectangle. Inside this green rectangle are the labels `dia`, `mes`, and `ano`, representing the nested `Data` structure.

# Estruturas aninhadas



```
typedef struct {  
    int dia,mes,ano;  
} Data;  
  
typedef struct {  
    char nome[150];  
    Data nascimento;  
} Aluno;  
  
int main() {  
    Aluno aluno1;  
    gets(aluno1.nome);  
    aluno1.nascimento.dia = 2;  
    ...  
}
```

**aluno1**

The diagram illustrates the memory layout of the `aluno1` variable. It is represented as a large light blue rectangle. Inside this rectangle, at the top, are the labels `nome` and `nascimento`. Below `nascimento`, there is a smaller green rectangle. Inside this green rectangle are the labels `dia`, `mes`, and `ano`, representing the nested structure.

# Estruturas aninhadas



```
typedef struct {  
    int dia,mes,ano;  
} Data;  
  
typedef struct {  
    char nome[150],cpf[12];  
    Data nascimento;  
} Responsavel;  
  
typedef struct {  
    char nome[150];  
    Data nascimento;  
    Responsavel mae,pai;  
} Aluno;  
  
int main() {  
    Aluno aluno1;  
    ...  
}
```

aluno1

nome  
nascimento

dia  
mes  
ano

mae

nome  
cpf  
nascimento

dia  
mes  
ano

pai

nome  
cpf  
nascimento

dia  
mes  
ano

# Estruturas aninhadas



```
typedef struct {
    int dia,mes,ano;
} Data;

typedef struct {
    char nome[150],cpf[12];
    Data nascimento;
} Responsavel;

typedef struct {
    char nome[150];
    Data nascimento;
    Responsavel mae,pai;
} Aluno;

int main() {
    Aluno aluno1;
    gets(aluno1.nome);
    aluno1.nascimento.dia = 2;
    ...
}
```

aluno1

nome  
nascimento

dia  
mes  
ano

mae

nome  
cpf  
nascimento

dia  
mes  
ano

pai

nome  
cpf  
nascimento

dia  
mes  
ano

# Estruturas aninhadas



```
typedef struct {
    int dia,mes,ano;
} Data;

typedef struct {
    char nome[150],cpf[12];
    Data nascimento;
} Responsavel;

typedef struct {
    char nome[150];
    Data nascimento;
    Responsavel mae,pai;
} Aluno;

int main() {
    Aluno aluno1;
    gets(aluno1.nome);
    aluno1.nascimento.dia = 2;
    aluno1.mae.nascimento.ano = 1975;
    printf("%s", aluno1.pai.cpf); ...
```

aluno1

nome  
nascimento

dia  
mes  
ano

mae

nome  
cpf  
nascimento

dia  
mes  
ano

pai

nome  
cpf  
nascimento

dia  
mes  
ano

# Estruturas com vetor



- Em alguns casos o tipo estrutura possui vetores como um dos seus campos.

```
typedef struct
{
    int vetorA[4];
    char vetorB[10];
} Dados;
```

- O acesso a estes campos é feito da mesma maneira como acesso direto a um vetor.

```
Dados registro;
registro.vetorA[2] = 100;
gets(registro.vetorB);
```



- 4) Faça um programa (função principal) para leitura, via teclado, dos dados de um atleta.  
Os dados a serem guardados na estrutura atleta são os seguintes: nome, medidas (estrutura definida no exercício 2), esporte praticado e idade.  
Ao final, imprima estas informações na tela.

# Atribuição de estruturas

- Uma das vantagens ao utilizarmos estruturas é a possibilidade de copiarmos toda a informação de uma estrutura para outra do mesmo tipo com uma atribuição simples:

```
typedef struct {  
    int X;  
    int Y;  
} Coordenadas;  
  
int main ()  
{  
    Coordenadas primeira, segunda;  
    primeira.X = 20;  
    primeira.Y = 30;  
    segunda = primeira;  
    printf("%d %d", segunda.X, segunda.Y);  
    return 0;  
}
```

# Exemplo completo

- Considere as informações de um aluno que tem NOME e 4 notas como campos de uma estrutura; veja *layout* abaixo.

Cadastro de notas escolares

Nome: \_\_\_\_\_

Notas			
1	2	3	4

- Desenvolver um algoritmo para ler e imprimir o nome e as notas de um aluno.

# *Exemplo completo*



- Ideia básica do algoritmo :
  1. Definir estrutura
  2. Declarar variáveis
  3. Ler os dados de um aluno
  4. Imprimir os dados do aluno

# Exemplo completo



## 1. Definir estrutura

```
typedef struct
{
    char nome[40];
    float notas[4];
} Aluno;
```

# Exemplo completo



1. Definir estrutura
2. Declarar variáveis

```
typedef struct
{
    char nome[40];
    float notas[4];
} Aluno;
```

```
int main()
{
    Aluno a; // dados do aluno
    int i; // contador para percorrer o vetor de notas
    ...
}
```

# Exemplo completo



1. Definir estrutura
2. Declarar variáveis
3. Ler dados de um aluno

```
typedef struct
{
    char nome[40];
    float notas[4];
} Aluno;

int main()
{
    Aluno a;
    int i;
    ...
}
```

```
gets(a.nome);
for (i = 0 ; i <= 3; i++ )
{
    scanf("%f", &a.notas[i]);
}
```

# Exemplo completo



1. Definir estrutura
2. Declarar variáveis
3. Ler dados de um aluno
4. Imprimir os dados

```
typedef struct
{
    char nome[40];
    float notas[4];
} Aluno;

int main()
{
    Aluno a;
    int i;
    gets(a.nome);
    for (i = 0 ; i <= 3; i++ )
    {
        scanf("%f", &a.notas[i]);
    }
```

```
puts(a.nome);
for (i = 0 ; i <= 3; i++ )
{
    printf("%f", a.notas[i]);
}
```



# Exemplo completo



```
typedef struct {
    char nome[40];
    float notas[4];
} Aluno;

int main()
{
    Aluno a;
    int i;
    gets(a.nome);
    for (i = 0 ; i <= 3; i++ )
    {
        scanf("%f", &a.notas[i]);
    }

    puts(a.nome);
    for (i = 0 ; i <= 3; i++ )
    {
        printf("%f", a.notas[i]);
    }
    return 0;
}
```

# Estruturas e funções



- Como qualquer outra variável, uma variável do tipo estrutura pode ser usada como **parâmetro** ou pode ser o **valor de retorno** de uma função.

```
typedef struct
{
    int valorA;
    int valorB;
} MinhaEstrutura;
```

```
void imprime(MinhaEstrutura e1)
{
    printf("\n%d", e1.valorA);
    printf("\n%d", e1.valorB);
    e1.valorB = e1.valorA;
}

MinhaEstrutura novaEstrutura( )
{
    MinhaEstrutura novaVar;
    scanf("%d", &novaVar.valorA);
    scanf("%d", &novaVar.valorB);
    return novaVar;
}

int main()
{
    MinhaEstrutura var;
    var = novaEstrutura();
    imprime(var);
    return 0;
}
```

Tipo de  
retorno  
da função

# Estruturas e funções

- Como qualquer outra variável, uma variável do tipo estrutura pode ser usada como **parâmetro** ou pode ser o **valor de retorno** de uma função.

```
typedef struct
{
    int valorA;
    int valorB;
} MinhaEstrutura;
```

A variável **e1** é alterada.  
Observe que **e1** é uma  
cópia de **var**.  
Assim, **var** não é alterada.

```
void imprime(MinhaEstrutura e1)
{
    printf("\n%d", e1.valorA);
    printf("\n%d", e1.valorB);
    e1.valorB = e1.valorA;
}

MinhaEstrutura novaEstrutura( )
{
    MinhaEstrutura novaVar;
    scanf("%d", &novaVar.valorA);
    scanf("%d", &novaVar.valorB);
    return novaVar;
}

int main()
{
    MinhaEstrutura var;
    var = novaEstrutura();
    imprime(var);
    return 0;
}
```

O valor de **var** é  
copiado para **e1**

A variável **var**  
não é alterada na  
função **imprime**

- Pode-se criar vetores de estruturas como se criam vetores de tipos primitivos.
- Os programas apresentados até o momento só fizeram menção a uma única instância da estrutura.
- É necessário possuir uma definição da estrutura antes de declarar um vetor de estrutura.

- Suponha que deseja-se manter um registro de informações relativas a passagens rodoviárias de todos lugares (poltronas) de um ônibus.
- Pode-se utilizar uma estrutura referente a cada poltrona (`passagem`) e para agrupar todas elas utiliza-se um vetor de estruturas.

# Vetores de estruturas

- Um ônibus possui 44 lugares numerados de 0 a 43, passa por várias cidades e precisa armazenar as informações abaixo:

0	
1	
⋮	
43	

Nome:\_\_\_\_\_ Número:\_\_\_\_\_

De:\_\_\_\_\_ Para:\_\_\_\_\_

Data:\_\_\_\_/\_\_\_\_/\_\_\_\_ Horário:\_\_\_\_\_

Poltrona:\_\_\_\_\_ Distância:\_\_\_\_\_

# Vetores de estruturas

- Definição da estrutura:

```
typedef struct
{
    char nome[50];
    int numero;
    char origem[20];
    char destino[20];
    char data[8];
    char horario[5];
    int poltrona;
    float distancia;
} Passagem;
```

- Declaração do vetor de estruturas:

```
Passagem dadosPassagem[44];
```

- Declaração do vetor de estruturas:

```
Passagem dadosPassagem[44];
```

- Observe que cada posição do vetor de estruturas pode ser vista como uma variável do tipo Passagem.

```
Passagem p, dadosPassagem[44];  
p = dadosPassagem[1];
```



- Em cada posição do vetor, os campos de Passagem podem ser acessados:

```
Passagem dadosPassagem[44];
```

```
dadosPassagem[2].numero = 2;  
scanf("%f", &dadosPassagem[2].distancia);  
gets(dadosPassagem[2].origem);  
gets(dadosPassagem[2].destino);  
if (dadosPassagem[2].distancia > 50.0)  
    puts(dadosPassagem[2].nome);
```

- Em cada posição do vetor, os campos de Passagem podem ser acessados:

```
int i;  
Passagem dadosPassagem[44];  
for ( i=0; i<44; i++ ) {  
    dadosPassagem[i].numero = i;  
    scanf ("%f", &dadosPassagem[i].distancia);  
    gets (dadosPassagem[i].origem);  
    gets (dadosPassagem[i].destino);  
    if (dadosPassagem[i].distancia > 50.0)  
        puts (dadosPassagem[i].nome);  
}
```

# Inicialização de estruturas

```
typedef struct
{
    int  codigo;
    char descricao[120];
} produto;
```

- Pode-se inicializar um vetor de estruturas no momento de sua criação como o código abaixo:

```
produto estoque[6] = {{235, "Teclado USB"},
                      {245},
                      {515, "Memoria DDR"} };
```

# Inicialização de estruturas

```
typedef struct
{
    int  codigo;
    char descricao[120];
} produto;
```

- Se o número de elementos inicializados for menor que os alocados em memória, o restante do vetor ficará “zerado” (se numérico) ou vazio (se *strings*).

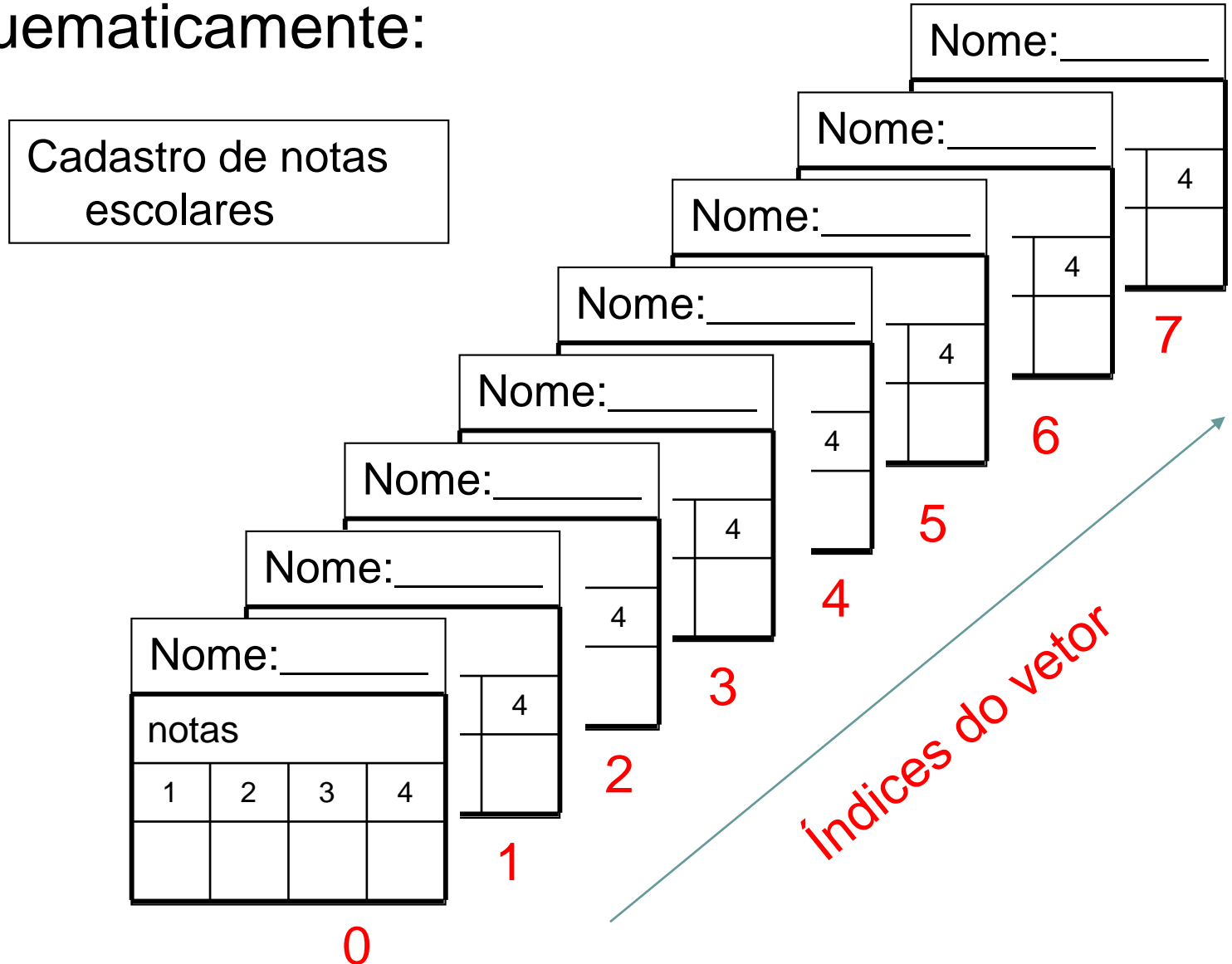
```
produto estoque[6] = {{235, "Teclado USB"},
                      {245},
                      {515, "Memoria DDR"} };
```

- No exemplo, o índice 1 do vetor terá descrição vazia e os índices 3, 4 e 5 terão código 0 e descrição vazia.

- Outro exemplo mais completo....
- Considere que você está fazendo um programa que leia o nome e as 4 notas escolares de 8 alunos.

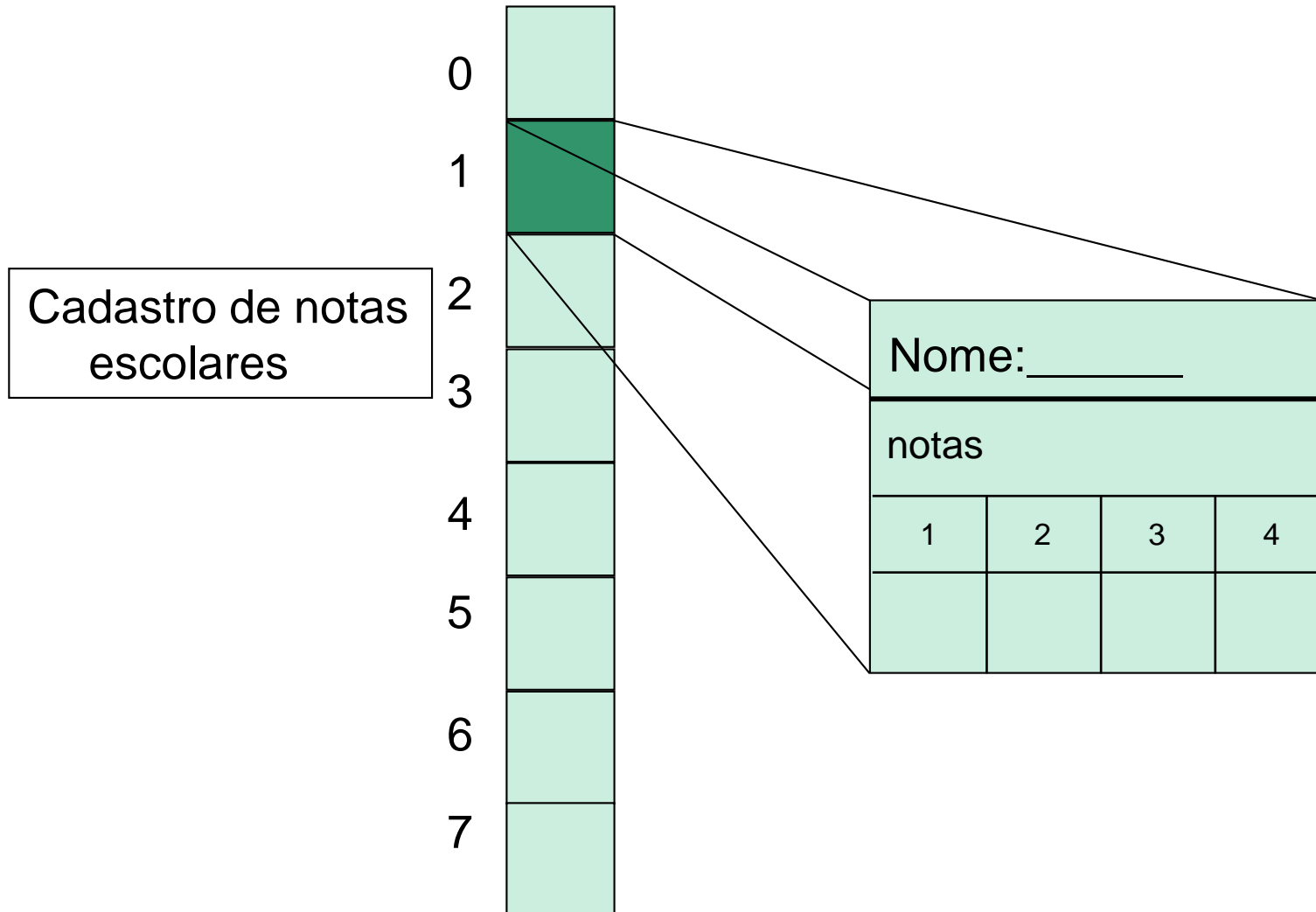
# Vetores de estruturas

- Esquemáticamente:



# Vetores de estruturas

- Visão esquemática do vetor de estruturas:



- Definição da estrutura e do vetor de 8 alunos.

```
typedef struct
{
    char nome[40];
    float notas[4];
} Aluno;

int main()
{
    Aluno notasAlunos[8];
    ...
}
```



- Procedimento para ler os dados dos alunos:

```
void le_vetor_alunos (Aluno A[8])
{
    int i, j;
    for (i = 0; i < 8; i = i + 1)
    {
        gets (A[i].nome);
        for (j = 0; j < 4; j = j + 1)
        {
            scanf ("%f", &A[i].notas[j]);
        }
    }
}
```

# Leitura de vetor de estruturas



- Procedimento para ler os dados dos alunos:

```
void le_vetor_alunos (Aluno A[8])
{
    int i, j;
    for (i = 0; i < 8; i = i + 1)
    {
        gets (A[i].nome);
        for (j = 0; j < 4; j = j + 1)
        {
            scanf ("%f", &A[i].notas[j]);
        }
    }
}
```

Assim como ocorre com tipos básicos:

- Alterações feitas em uma **variável** passada por parâmetro **não têm efeito** fora da função;
- Alterações feitas em um **vetor são mantidas** fora da função.

- Procedimento para ler os dados dos alunos:

```
void le_vetor_alunos (Aluno A[8])
{
    int i, j;
    for (i = 0; i < 8; i = i + 1)
    {
        gets (A[i].nome);
        for (j = 0; j < 4; j = j + 1)
        {
            scanf ("%f", &A[i].notas[j]);
        }
    }
}
```

Laço para ler  
os oito alunos

- Procedimento para ler os dados dos alunos:

```
void le_vetor_alunos (Aluno A[8])
{
    int i, j;
    for (i = 0; i < 8; i = i + 1)
    {
        gets (A[i].nome);
        for (j = 0; j < 4; j = j + 1)
        {
            f", &A[i].notas[j]);
        }
    }
}
```

gets é usado  
para ler o nome  
de cada aluno

Índice i do  
vetor de  
alunos

- Procedimento para ler os dados dos alunos:

```
void le_vetor_alunos (Aluno A[8])
{
    int i, j;
    for (i = 0; i < 8; i = i + 1)
    {
        gets (A[i].nome);
        for (j = 0; j < 4; j = j + 1)
        {
            scanf ("%f", &A[i].notas[j]);
        }
    }
}
```

Para cada aluno, um  
laço para ler as suas  
quatro notas

- Procedimento para ler os dados dos alunos:

```
void le_vetor_alunos (Aluno A[8])
{
    int i, j;
    for (i = 0; i < 8; i = i + 1)
    {
        gets (A[i].nome);
        for (j = 0; j < 4; j = j + 1)
        {
            scanf ("%f", &A[i].notas[j]);
        }
    }
}
```

Índice i do  
vetor de  
alunos

Índice j do  
vetor de notas

# Impressão de vetor de estruturas

- O processo de escrita de um vetor de estrutura é similar aos modos de escrita anteriores já vistos.

```
void imprime_vetor_alunos (Aluno A[8])
{
    int i, j;
    for (i = 0; i < 8; i = i + 1)
    {
        puts (A[i].nome);
        for (j = 0; j < 4; j = j + 1)
        {
            printf(" %f", A[i].notas[j]);
        }
    }
}
```

# Programa completo



```
#include <stdio.h>

typedef struct
{
    char nome[40];
    float notas[4];
} Aluno;

void leVetorAlunos (Aluno a[], int n)
{
    int i, j;
    for( i = 0; i < n; i++)
    {
        printf("Informe o nome do aluno");
        gets(a[i].nome);
        printf("Informe as 4 notas ");
        for( j = 0; j < 4; j++)
            scanf("%f%c", &a[i].notas[j]);
    }
}
```

```
void imprimeVetorAlunos (Aluno a[], int n)
{
    int i, j;
    printf("\nAlunos:");
    for( i = 0; i < n; i++)
    {
        printf("\n%20s ", a[i].nome);
        for( j = 0; j < 4; j++)
            printf("%.2f ", a[i].notas[j]);
    }
}

int main()
{
    Aluno turma[35];

    leVetorAlunos (turma, 35);
    imprimeVetorAlunos (turma, 35);

    return 0;
}
```



# Exemplo 2



```
#include <stdio.h>
#define TAM 4

typedef struct
{
    int codigo;
    char descricao[120];
} Produto;

int main()
{
    Produto estoque[TAM];
    leProdutos(estoque);
    imprimeProdutos(estoque);
    return 0;
}
```

```
void leProdutos (Produto p[TAM])
{
    int i;
    printf("Produtos armazenados:\n");
    for( i = 0; i < TAM; i++)
    {
        printf("Produto %d: ", i);
        printf("Codigo: ");
        scanf("%d%c", &p[i].codigo);
        printf("Descricao: ");
        gets(p[i].descricao);
    }
}

void imprimeProdutos (Produto p[TAM])
{
    int i;
    for( i = 0; i < TAM; i++)
        printf("%3d - %s\n",
            p[i].codigo, p[i].descricao);
}
```

- 5) a) Crie uma estrutura chamada `ponto` contendo apenas as coordenadas  $x$  e  $y$  (inteiros) do ponto.
- b) Faça uma função que receba dois pontos por parâmetro e retorne a distância entre eles.
- c) Faça um programa que declare 2 pontos, leia as coordenadas  $x$  e  $y$  de cada um e chame a função criada para calcular a distância entre eles. Apresente no final a distância entre os dois pontos.
- 6) Utilizando a estrutura `ponto` definida no exercício 5, faça uma função que receba 2 pontos e retorne o ponto mais próximo da origem. Em seguida, modifique o programa principal para ler 4 pontos e imprimir apenas as coordenadas do ponto mais próximo da origem.

- 7) Utilizando a estrutura `medidas` definida no exercício 2, faça um programa que leia a altura e o peso de 6 pessoas e imprima a média da altura e a média do peso do grupo.
- 8) Faça um programa que receba três nomes (de no máximo 100 caracteres cada) e as idades das respectivas pessoas em um vetor de estruturas. Após o recebimento, listar os três nomes armazenados neste vetor por ordem crescente de idades.
- 9) Faça um programa que armazene informações de restaurantes. Cada restaurante é identificado pelo nome, o tipo de comida (brasileira, chinesa, francesa, italiana, japonesa, etc.) e uma nota para a cozinha (entre 0 e 5). O programa deverá ler todas as informações e imprimir a lista de todos os restaurantes e, ao final, o tipo de cozinha do restaurante com maior nota.

10) Faça um programa que permita o cadastro de veículos. A estrutura veículos deverá conter os campos placa, marca, modelo e ano. Faça um menu com as seguintes opções:

Menu:

- 1 - Ler as informações de um veículo
- 2 - Verificar se uma placa está no formato correto (AAADDDDD)
- 3 - Imprimir por ano
- 4 - Pesquisar veículo por placa
- 5 - Imprimir todos os veículos cadastrados

O programa deverá ter as seguintes características:

- No primeiro item, peça inicialmente o índice do vetor que deseja alterar.
- No terceiro item, peça o ano mínimo e máximo e imprima os veículos que estão nesse intervalo.
- Faça funções para realizar as operações de cada um dos itens do menu.

# Estruturas

DCC 120



# Estruturas: definição e declaração

- Definição de estruturas e declaração de variáveis

```
// Definicao da estrutura
typedef struct
{
    char nome[100];
    int matricula;
    int idade;
    char cidade[50];
    char sexo;
} Aluno;

void funcao()
{
    // Declaracao de variaveis
    Aluno fulano;
    Aluno a1, a2, a3 ;
```

# Estruturas: acesso a campos

- Campos ou membros de uma estrutura podem ser usados da mesma forma como as variáveis. São acessados usando o operador **ponto (.)** entre o **nome da variável** declarada como estrutura e o **nome do campo**.

```
typedef struct
{
    char nome[100];
    int matricula;
    int idade;
    char cidade[50];
    char sexo;
} Aluno;
```

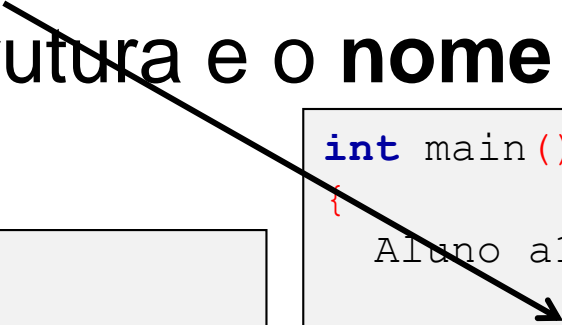
```
int main()
{
    Aluno a1;

    gets(a1.nome);
    a1.idade = 20;

    printf("Matricula %d", a1.matricula);

    // ...

    return 0;
}
```



# Estruturas: acesso a campos


- É possível inicializar os campos da estrutura na declaração da variável da estrutura. Veja o exemplo.

```
typedef struct
{
    int horas;
    int minutos;
    int segundos;
} Horario;

int main()
{
    Horario hnasc = { 8, 45, 0 };

    printf("%d %d %d", hnasc.horas, hnasc.minutos,
               hnasc.segundos);

    return 0;
}
```





# Estruturas com vetor

- Em alguns casos o tipo estrutura possui vetores como um dos seus campos.

```
typedef struct
{
    int vetorA[4];
    char vetorB[10];
} Dados;
```

- O acesso a estes campos é feito da mesma maneira como acesso direto a um vetor.

```
Dados registro;
registro.vetorA[2] = 100;
gets(registro.vetorB);
```

- Definição da estrutura

```
typedef struct
{
    int  codigo;
    char descricao[120];
} Produto;
```

- Após a definição da estrutura produto, pode-se definir um vetor de produtos como:

```
Produto estoque[100];

...

estoque[0].codigo = 10;
estoque[1].codigo = 17;
```

# Vetor de Estruturas - Exemplo



```
#include <stdio.h>
#define TAM 100

typedef struct
{
    int codigo;
    char descricao[120];
} Produto;

int main()
{
    int i;
    Produto estoque[TAM];
    for( i = 0; i < TAM; i++)
    {
        printf("Informe o codigo do produto %d: ", i);
        scanf("%d%c", &estoque[i].codigo); // %*c descarta o '\n'
        printf("Informe a descricao do produto %d: ", i);
        gets(estoque[i].descricao);
    }

    printf("Listagem dos produtos armazenados:\n");
    for( i = 0; i < TAM; i++)
        printf("%3d - %s\n", estoque[i].codigo, estoque[i].descricao);
    return 0;
}
```

# Funções e Estruturas



- Vetores de estruturas podem ser passados como argumentos de funções como qualquer outro vetor.

```
...  
Produto estoque[100];  
lerProdutos(estoque);  
...
```

```
void lerProdutos( Produto estoque[100] )  
{  
    for(int i = 0; i < 100; i++)  
    {  
        scanf("%d", &estoque[i].codigo);  
        gets(estoque[i].descricao);  
    }  
}
```

- 1) Um teatro precisa armazenar, para cada evento, as informações sobre os ingressos disponíveis em cada categoria de preço. Assim, defina uma estrutura que contém os dados de uma categoria: o número total de ingressos, o número de ingressos já vendidos e o preço do ingresso.

Faça um programa que declare três categorias distintas (estrutura definida acima): plateia, lateral e balcão. Inicialmente, o programa deve ler o preço e o total de ingressos de cada categoria. Assuma que neste momento, nenhum ingresso foi vendido.

Depois, o programa deve registrar cada venda de ingressos lendo categoria (P, L ou B) e número de ingressos e imprimindo quanto deve ser pago pelo comprador. A leitura de vendas será encerrada quando uma categoria inválida for digitada.

No final, o programa deve imprimir quantos ingressos ainda estão disponíveis em cada categoria e o total arrecadado com as vendas.

- 2) Defina uma estrutura para armazenar dados de uma disciplina: nome, número de alunos, número de avaliações, uma matriz com as notas e um caractere indicando o critério de cálculo da nota final (M=média; S=soma). A matriz deve armazenar até 7 avaliações para, no máximo, 150 alunos.

Faça um programa que leia os dados de uma turma e, em seguida, calcule e imprima a nota final de cada aluno.

# Exercícios



- 3) Faça um programa que leia os dados de  $N$  ingressos (sendo  $N$  definido com a diretiva *define*). As informações que deverão ser lidas de cada ingresso são: preço, local e atração. Ao final, informe os eventos de ingresso mais barato e mais caro.
- 4) Utilizando a estrutura definida no exercício anterior. Faça uma função que receba como parâmetro um vetor de ingressos e o nome de uma atração. A função deve localizar a atração no vetor, ler do teclado o novo preço do ingresso e atualizar o preço da atração. Faça também uma função que receba como parâmetro um único ingresso e imprima suas informações. Modifique o programa principal para atualizar o preço das cinco primeiras atrações e imprimir os dados de todos os ingressos, chamando obrigatoriamente as funções anteriores.
- 5) Faça um programa que permita comparar o preço de um eletrodoméstico em diferentes lojas da cidade. Para cada loja, os seguintes campos devem ser armazenados: nome da loja, endereço (estrutura com rua, número e bairro), telefone e preço do eletrodoméstico. Utilize funções para: (1) ler do teclado todas as informações de todas as lojas; (2) ler do teclado o nome de uma loja já cadastrada, localizar e imprimir suas informações; (3) calcular, imprimir e retornar a média dos preços cadastrados; (4) imprimir nome e telefone das lojas em que o preço está abaixo da média (use a função anterior para obter o valor da média); (5) imprimir todas as informações da loja que tem o menor preço. Faça um programa principal que apresente um menu com as opções acima.

- 6) Elabore um programa que auxilie no controle de uma fazenda de gado que possua um total de 100 cabeças de gado. O programa deverá conter uma estrutura que comporte:
- código: código da cabeça de gado;
  - leite: número de litros de leite produzido por semana;
  - alimento: quantidade de alimento ingerida por semana (kg);
  - mês de nascimento;
  - ano de nascimento;
  - abate: 'N'(não) ou 'S' (sim).

## 6) (*continuação...*)

O seu programa deverá **conter um menu** com as seguintes funcionalidades:

- (a) Ler a base de dados (código, leite, alimento, nascimento) informados pelo usuário e armazenar em um vetor de estruturas.
- (b) Preencher o campo *abate*, considerando que a cabeça de gado irá para o abate caso:
  - tenha mais de 5 anos, ou;
  - produza menos de 40 litros de leite por semana, ou;
  - produza entre 50 e 70 litros de leite por semana e ingira mais de 50 quilos de alimento por semana.
- (c) Imprimir a quantidade total de leite produzida por semana na fazenda.
- (d) Imprimir a quantidade total de alimento consumido por semana na fazenda.
- (e) Imprimir a quantidade total de leite que vai ser produzido por semana na fazenda, após o abate
- (f) Imprimir a quantidade total de alimento que vai ser consumido por semana na fazenda, após o abate
- (g) Imprimir número de cabeças de gado que irão para o abate.
- (h) Inclua uma opção para sair do menu.