

# Algoritmos e Estruturas de Dados I

## Aula 12 - Struct

Prof. Felipe Lara



# Introdução

- Até o momento, usamos apenas os tipos de dados simples, ou seja, que já estão predefinidos no compilador
- **Problema:** agrupar em um único nome um conjunto de tipos de dados não similares

```
char nome[100];  
int idade;  
float nota[3];
```

# Introdução

- Até o momento, usamos apenas os tipos de dados simples, ou seja, que já estão predefinidos no compilador
- **Problema:** agrupar em um único nome um conjunto de tipos de dados não similares
  - O problema de agrupar dados diferentes em C é resolvido com estruturas (Struct)

ALUNO



```
char nome[100];  
int idade;  
float nota[3];
```

# Introdução

- Até o momento, usamos apenas os tipos de dados simples, ou seja, que já estão predefinidos no compilador
- **Problema:** agrupar em um único nome um conjunto de tipos de dados não similares
  - O problema de agrupar dados diferentes em C é resolvido com estruturas (Struct)

**Estruturas heterogêneas:** tipos de variáveis que agrupam dados geralmente diferentes

**ALUNO**



```
char nome[100];  
int idade;  
float nota[3];
```

# Introdução

- Até o momento, usamos apenas os tipos de dados simples, ou seja, que já estão predefinidos no compilador
- **Problema:** agrupar em um único nome um conjunto de tipos de dados não similares
  - O problema de agrupar dados diferentes em C é resolvido com estruturas (Struct)

**Estruturas heterogêneas:** tipos de variáveis que agrupam dados geralmente diferentes

- Já as matrizes agrupam dados similares, por isso, são ditas homogêneas
- Os itens de dados da estrutura são chamados de membros, e os da matriz, de elementos

# Introdução

- Com uma estrutura (struct) é possível criar um novo tipo de dados definido pelo desenvolvedor, ou seja, que inclui o que quisermos
- Por exemplo, podemos criar uma estrutura Folha\_de\_Pagamento, que irá incluir:
  - o nome do funcionário (string),
  - o número do departamento (inteiro)
  - o salário (float)

# Introdução

- Quando usamos:
  - São usadas para declarar registros a serem armazenados em arquivo
  - Estruturas de dados mais complexas: listas, filas, pilhas e árvores.
  - Ponteiros e estruturas facilitam a formação dessas estruturas.
  - Tipos de dados derivados (e não primitivos)

# Novos Tipos

- Por meio da palavra chave struct definimos um novo tipo de dado
- Definir um novo tipo de dado, significa informar ao compilador seu nome, tamanho em bytes e forma como deve ser armazenado e recuperado da memória
- Após ter sido definido, o novo tipo de dados existe e pode ser utilizado para criar variáveis de modo similar a qualquer tipo simples

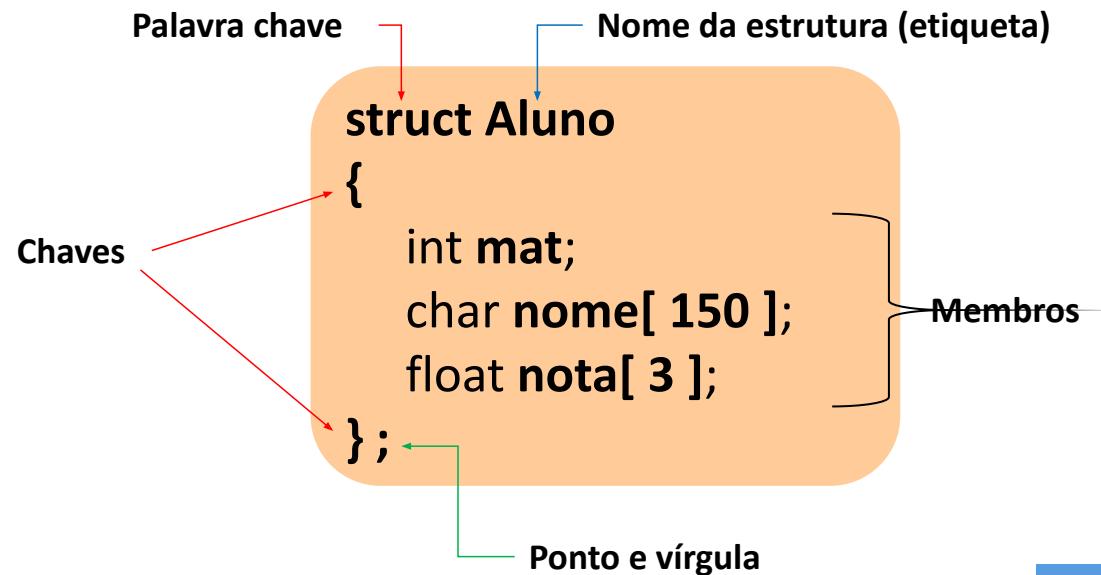
- Exemplo – Aluno (“tag” da estrutura)

```
struct Aluno
{
    int mat;
    char nome[ 150 ];
    float nota[ 3 ];
};
```

- Obs.: dois tipos de estruturas diferentes podem ter membros com o mesmo nome

# Struct

- A definição da estrutura informa como ela é organizada e quais são seus membros
- Declarações de estruturas não criam espaço na memória



# Struct

- A definição de uma estrutura **não cria nenhuma variável**, somente informa ao compilador as características de um novo tipo de dado
  - **Não há nenhuma reserva de memória**
- A palavra struct indica que um novo tipo de dado está sendo definido Aluno (no exemplo) será o nome da estrutura ou etiqueta (“tag”)
  - O nome do nosso novo tipo de dados é struct Aluno

Vamos definir nossa estrutura antes da main(), o que permite um acesso global a todas as funções definidas no programa

```
struct Aluno {  
    ....  
};  
  
int main( ) {  
    struct Aluno alu;  
    ....  
}
```

## Variável do Novo Tipo

Para declarar uma variável do tipo de dado definido, vamos usar a seguinte instrução:

```
int main()
{
    struct NomeEstrutura variável;
}
```

# Variável do Novo Tipo

---

Para declarar uma variável do tipo de dado definido, vamos usar a seguinte instrução:

```
int main()
{
    struct NomeEstrutura variável;
}
```

No nosso exemplo:

```
struct Aluno alu1, alu2;
```

Aqui foram criadas duas variáveis (alu1 e alu2) do tipo de dados Aluno

---

# Operações

---

- Atribuição de variáveis do tipo da estrutura a variáveis do mesmo tipo da estrutura
  - Indicação do endereço de uma variável do tipo da estrutura (operador &)
  - Acesso aos membros de uma variável do tipo da estrutura
- 
- Uso do operador sizeof para determinar o tamanho de uma variável do tipo da estrutura
  - NÃO PODEMOS:  
Comparar estruturas usando == e !=

# Inicialização

- Similar a vetores e matrizes na declaração

```
typedef struct Aluno
{
    int mat;
    char nome[250];
    double nota[3], media;
} Aluno;
```

```
Aluno variavel = {33, "exemplo", {5.2, 6.1, 7.5}, 34.5};
```

# Inicialização

- Similar a vetores e matrizes na declaração

```
typedef struct Aluno
{
    int mat;
    char nome[250];
    double nota[3], media;
} Aluno;
```

```
Aluno variavel = {33, "exemplo", {5.2, 6.1, 7.5}, 34.5};
```

- Atenção: se o número de inicializadores na lista for menor do que os membros na estrutura, os membros restantes serão automaticamente inicializados em zero, ou NULL se o membro for um ponteiro.

# TypeDef

---

- Declarações com `typedef` não produzem novos tipos de dados
  - Criam apenas novos nomes (sinônimos) para os tipos de dados existentes, podendo ser uma estrutura

- Sintaxe

```
typedef tipo-existente sinônimo;
```

- Exemplos

```
typedef char Byte;  
typedef int uint;
```

---

Foi definido outro nome para o tipo `char` e para o tipo `int`

# Typedef

---

- É possível definir outro nome para uma estrutura, ou então definir o mesmo nome usando o typedef

```
typedef struct NomeEstrutura
{
    tipo de dado nomeMembro;
    tipo de dado nomeMembro;
    tipo de dado N nomeMembroN;
} NomeEstruturaNovo ;
```

- Mas porque alterar o nome de uma estrutura?
  - A declaração de uma variável para esse novo tipo ficará mais simplificada

# Variável com Typedef

---

- Definindo um novo nome para a estrutura:

```
typedef struct Aluno
{
    int mat;
    char nome[ 150 ];
    float nota[ 3 ];
} AlunoNovo ;
```

- Declarando a variável:

```
int main()
{
    AlunoNovo variável;
```

Observe que aqui basta colocar o novo nome da estrutura, não é necessário usar a palavra chave **struct**

---

## Acesso aos Membros

Para acessar os membros de uma estrutura, usamos o operador . (ponto) e a **variável** do tipo da estrutura, já declarada:

```
int main()
{
    NomeEstruturaNovo variável;
    variável.nomeMembro1 = valor;
    scanf("%tipo_de_dado", &variável. nomeMembro2);
}
```

# Exemplo

```
int main()
{
    //struct Aluno alu; é possível usar assim sem typedef
    AlunoNovo alu; // ou assim com typedef
    float soma=0;
    novoint i;
    alu.mat = rand()%100; //numero aleatorio de 0 a 99 para a matricula
    printf("Digite o nome do aluno: ");
    gets(alu.nome);
    for(i=0;i<3;i++)
    {
        printf("Digite a nota %d: ", i+1);
        scanf("%f", &alu.notas[ i ]);
        soma += alu.notas[ i ];
    }
    alu.media = soma/3;
    printf("O aluno %s matricula %d teve media %.2f!", alu.nome, alu.mat, alu.media);
    return 0;
}
```

```
typedef struct Aluno
{
    int mat;
    char nome[ 250 ];
    float notas[ 3 ], media;
} AlunoNovo;

typedef int novoint;
//usando typedef
```

# Vetor e Matriz

---

- É possível declarar um vetor e/ou matriz do tipo da estrutura definida
- A declaração é semelhante a usada para os tipos de dados usados até o momento:

**Estrutura nomeVetor [ tamanho ] ;**

- E para o caso de matrizes:

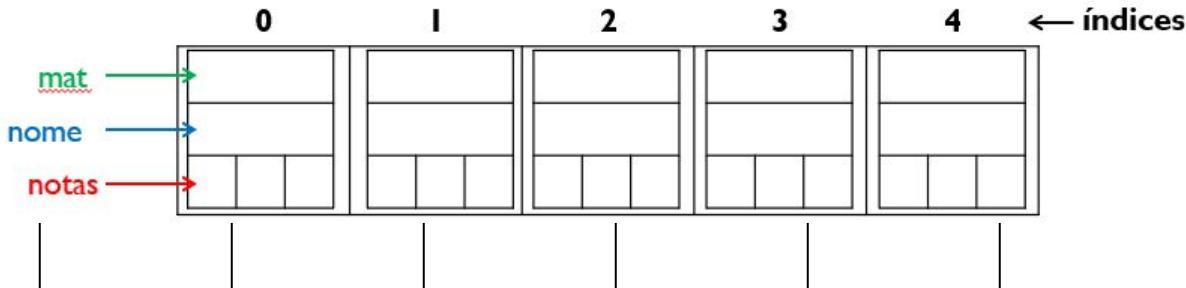
**Estrutura nomeMatriz [ linhas ] [ colunas ] ;**

# Vetor e Matriz

---

- Cada posição do vetor ou da matriz irá conter todos os membros definidos na estrutura, observe o exemplo, considerando um vetor:

```
int main()
{
    AlunoNovo vet[ 5 ];
    ...
}
```



```
struct Aluno
{
    int mat;
    char nome[ 250 ];
    float nota[ 3 ]
};
```

# Vetor e Matriz

---

- Para acessar os membros da estrutura com vetor também usaremos o operador ponto ( . )
- A principal diferença é que precisamos indicar em qual posição do vetor (índice) será armazenado cada dado, isso no caso de entrada de dados

Continuamos precisando de repetição para percorrer o vetor e ir armazenando os dados em cada índice

```
for ( i = 0; i < tamanho; i ++)
{
    scanf( "%tipoDeDadosDoMembro", &nomeVetor[ i ].membroEstrutura);
}
```

# Exemplo

```
typedef struct Aluno {  
    int mat;  
    char nome[150];  
    float nota[3];  
} AlunoNovo;  
  
#define tamanho 3  
  
int main()  
{  
    AlunoNovo alu[tamanho];  
    int i, j;  
    for(i=0; i <tamanho; i++)  
    {  
        alu[i].mat=rand()%100;  
        printf("Digite o nome:");  
        gets(alu[i].nome);  
    }
```

```
for(j=0; j<3; j++)  
{  
    printf("Digite a nota:");  
    scanf("%f", &alu[i].nota[j]);  
}  
}// fim do for do i  
printf("\n");  
  
for(i=0; i <tamanho; i++)  
{  
    printf("\nMatricula: %d",alu[i].mat);  
    printf("\nNome: %s",alu[i].nome);  
    for(j=0; j<3; j++)  
    {  
        printf("\nNota %d = ",(j+1),alu[i].nota[j]);  
    }  
}  
return 0;  
}
```

# Uso com Funções

---

- A linguagem C permite que as **funções retornem uma estrutura completa para outra função**
  - Para isso, basta criar uma variável do tipo da estrutura e não esquecer de incluir o return (retornando a variável)

```
Venda TotalVendas(Venda C,Venda D)
{
    Venda T;
    T.pecas = C.pecas + D.pecas;
    T.preco = C.preco + D.preco;
    return T;
}
```

---

# Uso com Funções

---

- Além disso, é possível que os **parâmetros de uma função sejam de um tipo de dados criado, ou seja, do tipo de uma estrutura**
  - Para isso, basta indicar na criação o tipo de dado do parâmetro, assim como fazemos com os tipos de dados simples

```
Venda TotalVendas(Venda C,Venda D)
{
    Venda T;
    T.pecas = C.pecas + D.pecas;
    T.preco = C.preco + D.preco;
    return T;
}
```

---

## Uso com Funções

---

- Quando as estruturas ou membros individuais da estrutura são passados a uma função, eles são **passados por valor**
- Os membros das estruturas passados por valor **não podem ser modificados** pela função utilizada

```
Venda TotalVendas(Venda C,Venda D)
{
    Venda T;
    T.pecas = C.pecas + D.pecas;
    T.preco = C.preco + D.preco;
    return T;
}
```

---

## Uso com Funções

---

- Quando as estruturas ou membros individuais da estrutura são passados a uma função, eles são passados por valor
  - Os membros das estruturas passados por valor não podem ser modificados pela função utilizada
- 
- Para passar uma estrutura por ponteiro:
    - Passe o endereço da variável da estrutura como com os tipos básicos
-

# Exemplo

```
typedef struct Venda
{
    int pecas;
    float preco;
} Venda;
```

```
Venda TotalVendas(Venda C, Venda D)
{
    Venda T;
    T.pecas = C.pecas + D.pecas;
    T.preco = C.preco + D.preco;
    return T;
}
```

```
int main()
{
    Venda A, B, Total;
    printf("\nVenda A");
    printf("\nDigite a quantidade de pecas: ");
    scanf("%d", &A.pecas);
    printf("\nDigite o preco: ");
    scanf("%f", &A.preco);
    printf("\nVenda B");
    printf("\nDigite a quantidade de pecas: ");
    scanf("%d", &B.pecas);
    printf("\nDigite o preco: ");
    scanf("%f", &B.preco);
    Total = TotalVendas(A, B);
    printf("\n\nTotal das vendas: \nPecas: %d\n
          Preco: %.2f", Total.pecas, Total.preco);
    return 0;
}
```

# Estrutura de Estrutura

---

- É possível definir estruturas com membros que sejam outras estruturas

```
typedef struct NomeEstrutura1
{
    tipo de dado nomeMembro;
    tipo de dado N nomeMembroN;
} NomeEstruturaNovo1 ;
```

```
typedef struct NomeEstrutura2
{
    tipo de dado nomeMembro1;
    NomeEstruturaNovo1 nomeMembro2;
} NomeEstruturaNovo2 ;
```

*Observe que o membro2 dessa estrutura é do tipo da estrutura anterior*

---

## Exemplo 1

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

typedef struct Aluno
{
    Data dtanasc;
    int mat;
    char nome[250];
} Aluno;
```

```
int main()
{
    Aluno A;
    printf("Cadastro Aluno: ");
    printf("\nData de nascimento - dia: ");
    scanf("%d", &A.dtanasc.dia);
    printf("\nData de nascimento - mes: ");
    scanf("%d", &A.dtanasc.mes);
    printf("\nData de nascimento - ano: ");
    scanf("%d", &A.dtanasc.ano);
    printf("\nDigite a matricula: ");
    scanf("%d", &A.mat);
    printf("\nDigite o nome: ");
    gets( A.nome);

    printf("\n\n*****");
    printf("\nAluno: %s Matricula:%d", A.nome, A.mat);
    printf("\nData de nascimento: %d / %d / %d",
        A.dtanasc.dia, A.dtanasc.mes, A.dtanasc.ano);
    return 0;
}
```

## Exemplo 2

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

typedef struct Aluno
{
    Data dtanasc;
    int mat;
    char nome[250];
} Aluno;
```

```
Aluno CadastraAluno( )
{
    Aluno A;
    printf("Cadastro Aluno: ");
    printf("\nData de nascimento - dia: ");
    scanf("%d", &A.dtanasc.dia);
    printf("\nData de nascimento - mes: ");
    scanf("%d", &A.dtanasc.mes);
    printf("\nData de nascimento - ano: ");
    scanf("%d", &A.dtanasc.ano);
    printf("\nDigite a matricula: ");
    scanf("%d", &A.mat);
    printf("\nDigite o nome: ");
    gets(A.nome);
    return A;
}
```

## Exemplo 2

```
int main()
{
    Aluno Alu;
    Alu = CadastraAluno( );
    printf("\n\n*****");
    printf("\nAluno: %s Matricula:%d", Alu.nome, Alu.mat);
    printf("\nData de nascimento: %d / %d / %d ",Alu.dtanasc.dia, Alu.dtanasc.mes,
          Alu.dtanasc.ano);
    return 0;
}
```

## Exemplo 3

```
#include <stdio.h>
#include <stdlib.h>
#define TAM 3

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

typedef struct Empregado
{
    Data dtanasc;
    int mat;
    char nome[250];
    float salario;
} Empregado;
```

```
int main()
{
    Empregado Emp[TAM];
    int i;
    for(i=0;i<TAM; i++)
    {
        printf("\nCadastro Empregado: ");
        printf("\nData de nascimento - dia: ");
        scanf("%d", &Emp[i].dtanasc.dia);
        printf("\nData de nascimento - mes: ");
        scanf("%d", &Emp[i].dtanasc.mes);
        printf("\nData de nascimento - ano: ");
        scanf("%d", &Emp[i].dtanasc.ano);
        Emp[i].mat = (i+1);
        printf("\nDigite o nome: ");
        gets(Emp[i].nome);
        printf("\nDigite o salario: ");
        scanf("%f", &Emp[i].salario);
    }
}
```

## Exemplo 3

```
for(i=0;i<TAM; i++)
{
    printf("\n\n*****");
    printf("\nEmpregado: %s Matricula:%d", Emp[i].nome, Emp[i].mat);
    printf("\nData de nascimento: %d / %d / %d ",Emp[i].dtanasc.dia, Emp[i].dtanasc.mes,
          Emp[i].dtanasc.ano);
    printf("\nSalario: %.2f", Emp[i].salario);
}
return 0;
}
```

# Ponteiro para Estrutura

---

- É possível definir ponteiros para estruturas

```
typedef struct Venda  
{  
    int pecas;  
    float preco;  
} Venda;  
  
Venda loja1, *loja2;
```

```
loja1.pecas = 3;  
  
loja2->pecas = 3;  
(*loja2).pecas = 3;
```

- Operador de membro de estrutura ou operador de ponto ( . )
- Operador de ponteiro de estrutura ou operador de seta ( -> )

# Ponteiro para Estrutura

---

- É possível definir ponteiros para estruturas

```
typedef struct Venda
{
    int pecas;
    float preco;
} Venda;

Venda loja1, *loja2;
```

```
loja2 = &loja1;
printf("%d\n", loja2->pecas);
```

## Apontamento para outra struct:

- Operador de membro de estrutura ou operador de ponto ( . )
- Operador de ponteiro de estrutura ou operador de seta ( -> )

# Ponteiro para Estrutura

---

- É possível definir ponteiros para estruturas

```
typedef struct Venda  
{  
    int pecas;  
    float preco;  
} Venda;  
  
Venda loja1, *loja2;
```

```
loja2 = malloc(sizeof(Venda));  
loja2->pecas = 20;  
loja2->preco = 9.99;
```

## Alocando dinamicamente (heap)

- Operador de membro de estrutura ou operador de ponto ( . )
- Operador de ponteiro de estrutura ou operador de seta ( -> )

# Dúvidas?