



# TÉCNICAS DE PROGRAMAÇÃO

Aula 09.2 – Registros do tipo struct (prática)



# Objetivos de Aprendizagem

1. Identificar os princípios básicos das formas de construção de programas para pesquisa e alteração em um registro;
2. Desenvolver programas com registro.



# Registros

- Nós já sabemos que um conjunto homogêneo de dados é composto por variáveis do mesmo tipo (vetores).
- Mas, e se tivermos um conjunto em que os elementos não são do mesmo tipo? Teremos, então, um conjunto **heterogêneo de dados, que são chamados de registros**.
- O registro é uma das principais formas de estruturar os dados no programa.
- Visa facilitar o agrupamento de variáveis de tipos diferentes, mas que possuem uma relação lógica.



# Registros

- Um registro é um conjunto de uma ou mais variáveis, que podem ser de tipos diferentes, agrupadas sobre um único nome.
- O fato de variáveis agrupadas em um registro poderem ser referenciadas por um único nome, facilita a manipulação dos dados armazenados nestas estruturas.



# Exemplo

- Imaginem uma estrutura que armazene as diversas informações do boletim de um aluno.
- O boletim é formado por um conjunto de informações logicamente relacionadas, porém de tipos diferentes, tais como:
  - número de matrícula (inteiro),
  - nome do aluno (caractere),
  - nome da disciplina (caractere),
  - média (real)
  - situação (caractere)
- Que são subdivisões do registro (elementos de conjunto), também chamadas de **campos**. Logo, um **registro** é composto por campos que são partes que especificam cada uma das informações.



## Boletim de Notas

Matricula...: 12345  
Nome.....: Michel  
Disciplina...: Matemática  
Média.....: 10.0  
Situação....: Aprovado

Notem que o boletim é composto por informações de diferentes tipos. No entanto, todas as informações do boletim estão relacionadas ao mesmo aluno.

O agrupamento de informações de tipos diferentes, que tem uma relação lógica, facilitará a manipulação de dados.



# Declaração de um Registro

- Para declarar uma variável, precisamos informar o seu tipo e dar um nome à mesma.
- Mas um registro é formado por várias variáveis de tipos diferentes.
- Como iremos declarar um registro?
- Para declarar um registro, é necessário informar quais variáveis, e seus respectivos tipos, fazem parte do registro.
- Dessa forma, precisamos declarar cada campo do registro, agrupando-os em um novo tipo de dado.
- A declaração de um registro passa por duas fases:
  - Definição de um novo tipo de dado
  - Declaração do registro propriamente dito.



# Declaração de um Registro

- Primeiramente, precisamos definir quais campos fazem parte do registro e criar um novo tipo de dado para o nosso programa.
- Precisamos criar um novo tipo de dado porque não conseguiríamos representar o tipo de informação que o registro armazena, utilizando os tipos primitivos disponíveis na linguagem: ***int***, ***float***, ***char***, *etc.*
- *Uma* vez que o registro agrupa variáveis de tipos de dados diferentes.



# Declaração de um Registro

## Sintaxe

```
typedef struct { declaração das variáveis;  
                } nome_do_tipo;
```

- **typedef:** Indica que um novo tipo de dado será definido.
- **struct:** Indica que o tipo de dado que será definido é um registro, ou seja, um agrupamento de variáveis de tipos de dados diferentes.
- **declaração das variáveis:** São as variáveis que fazem parte do registro. Neste local, precisamos especificar quais as variáveis irão compor o registro, além do tipo das mesmas. As variáveis são colocadas entre chaves.
- **nome\_do\_tipo:** é dado um nome ao novo tipo de dado que está sendo criado. Só depois que o novo tipo de dado é criado, é que o registro poderá ser declarado.

# Declaração de um Registro

## Sintaxe

```
nome_do_tipo    nome_do_registro;
```

- Onde:
  - **nome\_do\_tipo:** É o nome do tipo de dado que definimos no nosso programa, formado pelo agrupamento de várias variáveis.
  - **nome\_do\_registro:** É o nome da variável registro **que** **está** sendo declarada. O nome de um registro segue as regras dos identificadores.



# Exemplo

## Definição de tipo e declaração de registro

```
typedef struct {  int matricula;  
                  char nome[20], disciplina[20], situação[10];  
                  float media;  
  
                  } Tipo_Aluno;
```

```
Tipo_Aluno aluno;
```

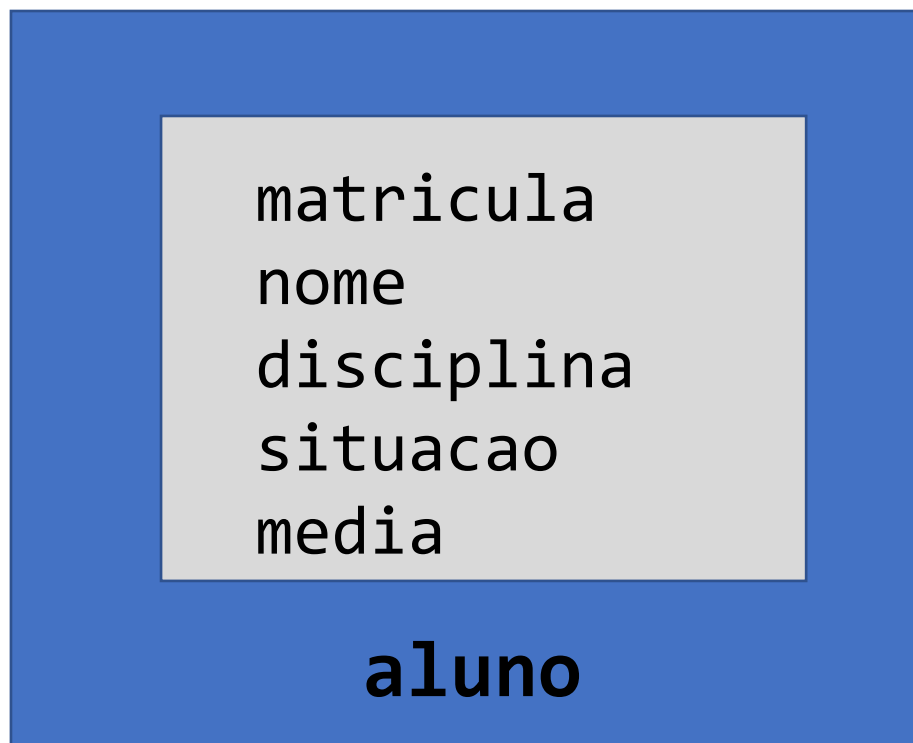


# Explicação do Boletim

- **Linha 1:** Com o typedef estamos informando que será definido um novo tipo de dado. O struct indica que este tipo é um agrupamento de variáveis de tipos diferentes, ou seja, um registro. Colocamos o abre chaves e começamos a declarar as variáveis que irão compor o registro. Começamos com a declaração da variável inteira matricula. Se houvesse mais variáveis do tipo int, poderiam ser declaradas nesta mesma linha.
- **Linha 2:** Declaração das variáveis do tipo char: nome, disciplina e situação.
- **Linha 3:** Declaração da variável media que é do tipo float.
- **Linha 4:** Após a declaração de todas as variáveis que compõe o registro, podemos fechar a chaves e, em seguida, dar um nome a esse tipo de agrupamento, que acabamos de definir. Neste caso, o tipo foi chamado de Tipo\_Aluno. Ao definirmos um tipo de dado no nosso programa, significa que: podemos declarar variáveis dos tipos de dados primitivos (int, float, char, etc), além de variáveis do tipo de dado que nós definimos, neste caso, Tipo\_Aluno.
- **Linha 5:** Declaração de uma variável chamada aluno, e o tipo de dado que ela armazena é Tipo\_Aluno, ou seja, armazena: matricula, nome, disciplina, situacao e media.



# Representação gráfica do registro *aluno*



Notem que, a variável *aluno* (que é um registro) é formada pelos campos definidos no *Tipo\_Aluno*.





# Representação gráfica do registro *aluno*

- Como uma variável registro é formada por vários campos, precisamos utilizar uma forma diferenciada para informar qual campo do registro nós estamos querendo acessar.
- Lembram dos vetores que precisávamos dizer qual elemento do vetor seria acessado?
- Com os registros vai acontecer algo parecido.



# Acessando os campos do registro

- A variável ***aluno*** é um registro formado por vários campos. Para acessar um campo de um registro, devemos usar a seguinte sintaxe:

## Sintaxe

nome\_do\_registro.campo

- Onde:
  - **nome\_do\_registro:** É o nome da variável registro que queremos acessar. Após o nome\_do\_registro devemos colocar um ponto, que irá separar o nome do registro, do campo que vem logo em seguida.
  - **campo:** É o campo do registro que será acessado.




# Exemplo: Acesso aos campos de um registro

```
aluno.matricula = 12345;  
scanf("%f", &aluno.media);  
gets(aluno.nome);  
printf("Situacao do aluno: %s", aluno.situacao);
```

Notem que, se tivermos vários alunos em uma turma, precisaremos de várias variáveis registro do tipo *Tipo\_Aluno*, uma para cada aluno. Para fazer isso de forma mais simplificada, devemos juntar os conceitos de vetores e registros e criar um **vetor de registro**.





# Exemplo: Acesso aos campos de um registro

- Na **linha 1**, estamos acessando o campo matricula do registro aluno. Assim, colocamos o nome do registro, o ponto e o campo que queremos acessar. Com acesso ao campo, fizemos uma atribuição.
- Na **linha 2**, estamos acessando o campo media do registro aluno. Neste caso, ao invés de atribuir um valor ao campo de registro, estamos fazendo uma leitura via teclado e armazenando o valor digitado no campo media. Como o campo media é do tipo float, colocamos o %f no scanf, indicando que será lido um número real.
- Na **linha 3**, temos o comando de leitura gets, responsável por ler variáveis do tipo char. Neste gets, estamos fazendo uma leitura via teclado e armazenando o valor digitado no campo nome do registro aluno.
- Na **linha 4**, temos um printf que apresenta a situação do aluno. Para isso, acessamos o campo situação do registro aluno.



# Vetor de registro

- Os vetores são formados por um conjunto de dados do mesmo tipo.
- Na aula anterior, utilizamos vetores que armazenavam dados de tipos primitivos, ou seja, tipos disponíveis na linguagem – int, float, char, etc.
- Veremos que podemos utilizar como elemento do vetor não apenas um tipo primitivo, mas também os tipos construídos (tipos definido pelo programador), neste caso, os registros.
- Imaginem que queremos armazenar os boletins dos 50 alunos de uma turma.
- Para isso, será necessário um registro diferente para cada aluno.
- Para agrupar todos estes registros, iremos definir um vetor de registro.
- Como possuímos 50 alunos, podemos criar um vetor no qual cada posição armazena um Tipo\_Aluno.



# Vetor de registro

- Para declarar um vetor de registro, precisamos antes definir os elementos do registro, utilizando o *typedef struct*, e assim, definir um novo tipo de dado que será utilizado no programa.
- Após definirmos o novo tipo de dado, o vetor poderá ser declarado.
- Para declararmos um vetor, precisamos informar o tipo de dado que o vetor armazena, damos um nome ao vetor e informamos, entre colchetes, o tamanho do vetor.

# Definição do registro e declaração do vetor de registro

```
typedef struct { int matricula;  
                char nome[20], disciplina[20], situação[10];  
                float media;  
            } Tipo_Aluno;  
Tipo_Aluno alunos[50];
```

alunos	matricula	matricula		matricula	matricula
	nome	nome		nome	nome
	disciplina	disciplina		disciplina	disciplina
	media	media		media	media
	situacao	situacao		situacao	situacao
	0	1	...	48	49

**Se apenas com os vetores, já havíamos adquirido uma facilidade na declaração e manipulação de um grande conjunto de variáveis, com os vetores de registros, esta facilidade será aumentada.**

# Acessando os campos do vetor de registro

- Quando precisamos acessar um elemento do vetor, informamos, entre colchetes, o índice do elemento do vetor que será acessado.
- Quando queremos acessar um campo do registro, informamos o nome da variável registro, colocamos um ponto e o nome do campo que queremos acessar.

## Sintaxe

```
nome_do_vetor[indice].campo
```

- Onde:
  - **nome\_do\_vetor**: nome do vetor que queremos acessar.
  - **[índice]**: índice do vetor que será acessado.
  - **.campo**: campo do registro que será acessado.

# Acesso aos elementos de um vetor de registro

```
alunos[2].media = 7.5;  
alunos[3].matricula = 12345;  
gets(alunos[0].nome);  
scanf("%d",&alunos[2].matricula);
```

- Na **linha 1**, acessamos o elemento de índice 2 do vetor alunos e atribuímos 7.5 ao campo media.
- Na **linha 2**, acessamos o elemento de índice 3 do vetor alunos e atribuímos 12345 ao campo matricula.
- Nas **linhas 3 e 4**, temos acessos através de comandos de entrada de dados.
- Na **linha 3**, temos um gets, que obtém o nome do aluno, via teclado, e armazena no índice 0 do vetor alunos, no campo nome.
- Na **linha 4**, temos o scanf que lê a matrícula do aluno, armazenando no índice 2 do vetor alunos, no campo matricula.



# Usando vetor de registro

- Faça um programa que cadastre e apresente os dados dos alunos de uma escola. Os dados dos alunos devem ser armazenados em um vetor de registro, com a capacidade de armazenar até 20 alunos.
- A quantidade de alunos que será cadastrada é desconhecida.
- No momento do cadastro serão informados os seguintes dados para cada aluno: matrícula, nome, série(1-4) e se tem irmão na escola (1-sim/0-nao).
- O programa irá calcular o valor da mensalidade do aluno, que depende da série do aluno e se o mesmo tem irmão na escola. Valor da Mensalidade: 1ª Serie: R\$110, 2ª Serie: R\$130, 3ª Serie: R\$160, 4ª Serie: R\$170.
- O aluno que tiver irmão na escola, receberá 20% de desconto no valor da mensalidade.
- Quando o usuário decidir que não deseja mais cadastrar, o programa apresentará os dados de todos os alunos, em forma de tabela. Para facilitar o entendimento deste enunciado, vejam os exemplos das telas que são apresentadas durante a execução do programa:

## Colégio Legal

Cadastro de Aluno

Matricula..:

Nome.....:

Serie(1-4):

Irmão na escola (1-sim/0-nao):

Cadastrar outro aluno (1-sim/0-nao)?

## Colégio Legal

### Relatório Geral

Matricula	Nome	Serie	Irmão
Mensalidade			
xx	xxxxxx	x	x
xx.xx			
xx	xxxxxx	x	x
xx.xx			
xx	xxxxxx	x	x
xx.xx			
xx	xxxxxx	x	x
xx.xx			
xx	xxxxxx	x	x
xx.xx			

Tecla enter para sair...





# Explicação do Problema

- A primeira tela representa a operação de cadastramento dos dados dos alunos. Nesta tela, é que serão fornecidos os dados para o cadastramento dos alunos. Como a mensalidade é um dado calculado pelo programa, o usuário não fornecerá este dado no momento do cadastro.
- A segunda tela representa a apresentação dos dados de todos os alunos, em forma de tabela (listagem ou relatório). Cada linha da tabela apresentará os dados de um aluno.

# Código (I)

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
main(){
    typedef struct { int mat, serie,
                     irmao;
                     char nome[20];
                     float mens;
                     } Tipo_Aluno;
    Tipo_Aluno alunos[20];
    int qa, i, resp;
    qa =0;
```

```
do{
    system("cls");
    printf("Colégio Legal\n");
    printf("\n\nCadastro de
Alunos\n\n");
    printf("\nMatricula.: ");
    scanf("%d",&alunos[qa].mat);
    printf("\nNome.....: ");
    fflush(stdin);
    gets(alunos[qa].nome);
    printf("\nSerie(1-4): ");
    scanf("%d",&alunos[qa].serie);
    printf("\nIrmão na escola(1-
sim/0-nao):");
    scanf("%d",&alunos[qa].irmao);
```

# Código (II)

```
switch(alunos[qa].serie){
    case 1: alunos[qa].mens =
110; break;
    case 2: alunos[qa].mens =
130; break;
    case 3: alunos[qa].mens =
160; break;
    case 4: alunos[qa].mens =
170; break;
}

if (alunos[qa].irmao == 1)
    alunos[qa].mens =
alunos[qa].mens*0.8;
    qa++;
    printf("\n\nDeseja
cadastrar outro    aluno(1-sim/0-
nao)? ");
    scanf("%d",&resp);
```

```
}
```

12/11/2020

```
while ((resp == 1) && (qa <20));
```

```
system("cls");
printf("Colégio Legal\n");
printf("\n\nRelatorio Geral\n");
printf("\n_____
");
printf("\nMatricula    Nome
Serie    Irmão    Mensalidade");
printf("\n_____
");
```

```
for(i = 0; i < qa; i++)
printf("\n%9d    %-20s    %5d    %5d
%11.2f",
alunos[i].mat, alunos[i].nome,
alunos[i].serie,
alunos[i].irmao, alunos[i].mens);
printf("\n_____
");
```

```
printf("\nTecle enter para sair...");
getche();
```

```
}
```

# Ponteiros para structs

- Assim como ponteiros para variáveis, ponteiros para structs também podem ser definidos.
- `struct myStruct * struct_ptr;`
  - define um ponteiro para a estrutura `myStruct`.
- `struct_ptr = & struct_var;`
  - armazena o endereço da variável de estrutura `struct_var` no ponteiro `struct_ptr`.
- `struct_ptr -> struct_mem;`
  - acessa o valor do membro da estrutura `struct_mem`.

# Exemplo com structs de struct

```
struct aluno {  
    char nome[50];  
    int numero;  
    int idade;  
};  
// um ponteiro para struct como parâmetro de uma função  
void mostrarDadosAluno(struct aluno *al) {  
    printf("\nAluno:\n");  
    printf("Nome: %s\n", st->nome);  
    printf("Numero: %d\n", st->numero);  
    printf("Idade: %d\n", st->idade);  
}
```

# Exemplo com structs de struct


```
struct aluno a11 = {"Maria", 5, 21};  
mostrarDadosAluno(&a11);
```

- O operador **->** permite acessar membros da estrutura através do ponteiro.
- **(\*a1).idade** é o mesmo que **a1->idade**.
- Além disso, quando um **typedef** foi usado para nomear a estrutura, um ponteiro é declarado usando apenas o nome **typedef** junto com **\*** e o nome do ponteiro.



# Structs como parâmetros de função

- Uma função pode ter parâmetros de estrutura que aceitam argumentos por valor quando uma cópia da variável struct é tudo o que é necessário.
- Para uma função alterar os valores reais em uma variável struct, são necessários parâmetros de ponteiro.



# Exemplo de structs como parâmetro (I)

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

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

```
typedef struct {
```

```
    int id;
```

```
    char titulo[40];
```


```
    float carga_horaria;
```

```
} curso;
```

```
void atualizar_curso(curso *aula);
```

```
void mostrar_curso(curso aula);
```





# Exemplo de structs como parâmetro (II)

```
int main() {  
    curso c2;  
    atualizar_curso(&c2);  
    mostrar_curso(c2);  
    return 0;  
}
```

# Exemplo de structs como parâmetro (III)

```
void atualizar_curso(curso *aula) {  
    strcpy(class->title, "Intro ao C");  
    aula->id = 111;  
    aula->carga_horaria = 12.30;  
}
```

```
void mostrar_curso(curso aula) {  
    printf("%d\t%s\t%3.2f\n", aula.id, aula.title,  
    aula.carga_horaria);  
}
```



# Structs como parâmetro de função

- Como você pode ver, **atualizar\_curso()** assume um ponteiro como parâmetro, enquanto **mostrar\_curso()** assume a struct por valor.



# Vetores de struct

- Um vetor pode armazenar elementos de qualquer tipo de dado, incluindo structs.
- Depois de declarar um vetor de structs, um elemento é acessível com o número do índice.
- O operador de ponto é então usado para acessar membros do elemento, como no programa a seguir...



# Exemplo de vetor de structs (I)

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

```
typedef struct {  
    int altura;  
    int comprimento;  
    int largura;  
} caixa;
```



# Exemplo de vetor de structs (II)

```
int main() {  
    caixa caixas[3] = {{2, 6, 8}, {4, 6, 6}, {2, 6, 9}};  
    int k, volume;  
  
    for (k = 0; k < 3; k++) {  
        volume = caixas[k].altura * caixas[k].comprimento  
* caixas[k].largura;  
        printf("caixa% d volume% d \ n", k, volume);  
    }  
    return 0;  
}
```