



INSTITUTO FEDERAL

Norte de Minas Gerais

Campus Januária

Estruturas de Dados I

- Structs -



Structs

- **Imagine a solução para o seguinte problema...**
- Faça um programa que armazene o cadastro de até 100 pessoas.
- Cada cadastro deve armazenar: NOME, CPF, IDADE, ALTURA e PESO.
- Imprima o relatório de todas as pessoas, ordenadas pelo NOME.

SOLUÇÕES?



Structs

■ Imagine a solução para o seguinte problema...

- Faça um programa que registre até 100 pessoas.
- Cada cadastro deve conter: NOME, CPF, IDADE, ALTURA e PESO.
- Imprima o registro ordenado pelo NOME.

```
int main(){  
    char nome[100][100];  
    char cpf[12];  
    int idade[100];  
    float altura[100];  
    float peso[100];  
  
    for (int i; i<100; i++){  
        scanf(" %s", nome[i]);  
        scanf(" %s", cpf[i]);  
        scanf(" %d", &idade[i]);  
        (...)  
    }  
}
```

registro de até

NOME, CPF, IDADE,


coordenadas



Structs

■ Imagine a solução para o seguinte problema...

- Faça um programa que leia um vetor de até 100 pessoas.
- Cada cadastro deve conter: NOME, CPF, IDADE, ALTURA e PESO.
- Imprima o vetor de pessoas ordenadas pelo NOME.



```
int main(){
    struct Pessoa p[100][100];
    // ...
    float altura;
    float peso;

    for (i = 0; i < 100; i++){
        // ...
        printf("%d", &1);
        // ...
    }
}
```



Struct / Registro

- **STRUCT (registro)** é uma **Estrutura de Dados**:
 - **Composta**: Permite a **agregação** de um conjunto de **valores** sob um mesmo identificador.
 - **Heterogênea**: Estes valores podem ser de um mesmo tipo **ou não**.
- Geralmente, a definição de uma **struct** é feita através da **criação de um novo tipo abstrato de dados (TAD)** com uso do recurso **typedef**.



Declaração typedef

- A declaração **typedef** permite a definição de novos tipos de dados.

```
#include <stdio.h>
    (...)
typedef int inteiro;
typedef char string[100];
    (...)
int main(){
    inteiro x,y,z;
    string nome;
}
```



Declaração typedef

- A declaração **typedef** permite criar novos tipos de dados.

A declaração de um novo tipo de dados tem que ser realizada no escopo global do programa, ou seja, fora de qualquer função ou procedimento.

```
#include <stdio.h>
...
typedef int inteiro;
typedef char string[100];
...

int main(){
    inteiro x,y,z;
    string nome;
}
```



Declarando uma Struct

- A declaração de um **novo tipo struct** segue o modelo...

```
typedef struct{  
    char nome[100];  
    char cpf[12];  
    int idade;  
    float peso, altura;  
}Pessoa;
```




Declarando uma Struct

- A declaração de um **novo tipo struct** segue o modelo...

```
typedef struct{  
    char nome[100];  
    char cpf[12];  
    int idade;  
    float peso, altura;  
}Pessoa;
```

Declaração de um novo tipo de dados (struct), chamado *"Pessoa"*.



Declarando uma Struct

- A declaração de um **novo tipo struct** segue o modelo...

```
typedef struct{  
    char nome[100];  
    char cpf[12];  
    int idade;  
    float peso, altura;  
}Pessoa;
```

Variáveis que compõem
a estrutura *"Pessoa"*.



Declarando uma Struct

```
#include <stdio.h>

typedef struct{
    char nome[100];
    char cpf[12];
    int idade;
    float peso, altura;
}Pessoa;

int main{
    Pessoa p1;
}
```

Aqui você está declarando
uma variável do tipo Pessoa.



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

ATENÇÃO!

- Imagine que **typedef struct** seja o molde de um carimbo...
 - *Você não escreve em um carimbo em si.*



Nome: _____
CPF: _____ Idade: _____
Altura: _____ Peso: _____

**Mas escreve em um
espaço definido por ele.**



ATENÇÃO!

TIPO != VARIÁVEL

```
#include <stdio.h>

int main(){
    int n;
    scanf(" %d", &int);
    return 0;
}
```





Acessando uma Struct

■ Observe...

```
int main(){  
    Pessoa p1;  
    scanf(" %[^\n]s", p1.nome);  
    scanf(" %s",    p1.cpf);  
    scanf(" %d",    &p1.idade);  
    scanf(" %f",    &p1.peso);  
    scanf(" %f",    &p1.altura);  
}
```



Comparação vs. Atribuição

```
int main(){  
    Pessoa a,b;  
    scanf(" %[^\n]s", a.nome);  
    b = a;  
    printf("%s", b.nome);  
}
```

Atribuição de Structs



```
int main(){  
    (...)  
    if (a == b)  
        printf("A e B são iguais");  
}
```

Comparação de Structs





Bora CODAR!!!



1. Faça um programa que define o tipo de dados **Pessoa**, contendo: nome, cpf, altura, peso e idade.

Leia do usuário os dados de uma Pessoa, e após isso:

- a) Imprima todas as informações lidas.
- b) Calcule o IMC (Índice de Massa Corpórea) dessa pessoa:

$$\text{IMC} = \text{Peso} / \text{Altura}^2$$

- c) Informe o resultado do IMC, conforme tabela abaixo:

< 18.5	Abaixo do Peso
18.5 – 24.9	Saudável
25.0 – 29.9	Acima do Peso
> 30	Obesidade



Trabalhando com N variáveis

- Criar um **novo tipo de dados** para armazenar **apenas 1 variável** não faz muito sentido...
- Normalmente, precisaremos armazenar **muitas variáveis** de um mesmo **tipo struct**...

Qual estrutura de dados permite armazenar vários elementos de um mesmo tipo?



Vetor + Struct == Solução!

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

```
typedef struct{  
    char nome[100];  
    int idade;  
    float peso, altura;  
    char sexo;  
}Pessoa;
```

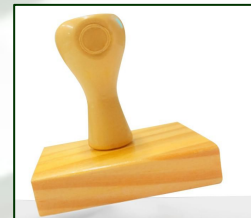
```
int main(){  
    Pessoa cadastro[100];  
}
```

Declaramos um Array com espaço para 100 elementos do tipo Pessoa



Vetor + Struct == Solução!

```
int main(){
    Pessoa cadastro[100];
}
```



Nome: _____ CPF: _____ Idade: _____ Altura: _____ Peso: _____	Nome: _____ CPF: _____ Idade: _____ Altura: _____ Peso: _____	Nome: _____ CPF: _____ Idade: _____ Altura: _____ Peso: _____	Nome: _____ CPF: _____ Idade: _____ Altura: _____ Peso: _____	Nome: _____ CPF: _____ Idade: _____ Altura: _____ Peso: _____
0	1	2	...	99



Vetor + Struct == Solução!

```
int main(){
    Pessoa cadastro[100];
}
```



Nome: _____ CPF: _____ Idade: _____ Altura: _____ Peso: _____	Nome: _____ CPF: _____ Idade: _____ Altura: _____ Peso: _____	Nome: _____ CPF: _____ Idade: _____ Altura: _____ Peso: _____	Nome: _____ CPF: _____ Idade: _____ Altura: _____ Peso: _____	Nome: _____ CPF: _____ Idade: _____ Altura: _____ Peso: _____
0	1	2	...	99

mas quantas pessoas DE FATO estão cadastradas?



Acessando um Vetor de Struct

```
int main(){
    Pessoa cadastro[100];    //repositório
    int contP = 0;           //qtde itens no repositório
    do{
        scanf(" %[^\n]s", cadastro[contP].nome);
        scanf(" %d", &cadastro[contP].idade);
        scanf(" %f", &cadastro[contP].peso);
        scanf(" %f", &cadastro[contP].altura);
        scanf(" %c", &cadastro[contP].sexo);
        contP++;
        scanf(" %c", &continua)
    }while(continua == 's' && contP<100);
}
```



Acessando um Vetor de Struct

```
int main(){
    Pessoa cadastro[100];    //repositório
    int contP = 0;           //qtde itens no repositório
    do{
        scanf(" %[^\n]s", cadastro[contP].nome);
        scanf(" %d", &cadastro[contP].idade);
        scanf(" %f", &cadastro[contP].peso);
        scanf(" %f", &cadastro[contP].altura);
        scanf(" %c", &cadastro[contP].sexo);
        contP++;
        scanf(" %c", &continua)
    }while(continua == 's' && contP<100);
}
```



Bora CODAR!!!



1. Faça um programa que define um novo tipo de dados chamado **Aluno**. Cada registro de Aluno deve conter: Nome do Estudante (s), Número de Matrícula (i), Nome do Curso (s), Média de Notas (f).
 - a) Leia os dados de vários alunos (até o nome informado for “**exit**”).
 - b) Após a fase de cadastro, pergunte ao usuário do sistema algum **Número de Matrícula** para ser pesquisado, e encontrando o registro, imprima todas as informações deste aluno.
 - c) Repita a operação da letra B acima, até que o usuário informe um N° de matrícula negativo (para encerrar o programa).
2. Refatore o problema anterior. Agora, toda vez que um aluno for pesquisado, deve-se perguntar se o usuário deseja lançar uma nota para este aluno. Sabe-se que todo aluno pode ter até 10 notas.
Todas as notas individuais devem ser salvas, e a Média de Notas deve ser recalculada automaticamente pelo sistema.
3. Analise! Ainda faz sentido manter a “Média de Notas” como atributo da struct?



A Biblioteca `datetime.h`

- A biblioteca customizada **`datetime.h`** foi criada para facilitar o uso de informações de **data e hora** nos problemas da disciplina.

Não existe portanto de forma nativa na linguagem.

- Está disponível no git da disciplina.
- Pode ser estendida com outras funções relacionadas ao seu objetivo e que julgar pertinente.



A Biblioteca `datetime.h`

- Novo tipo abstrato definido:

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

int main(){
    Datetime d;
}
```

Cria uma variável (struct) do tipo `Datetime`



A Biblioteca `datetime.h`

- Novo tipo abstrato definido:

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

int main(){
    Datetime d = dt_now();
}
```

A função `dt_now()` retorna os dados de data e hora atuais do sistema.



A Biblioteca `datetime.h`

Informações Disponíveis na struct `Datetime`

```
int main(){
    Datetime d = dt_now();
    printf("Dia: %d\n", d.day);
    printf("Mês: %d\n", d.mon);
    printf("Ano: %d\n", d.year);
    printf("Hora: %d\n", d.hour);
    printf("Min: %d\n", d.min);
    printf("Seg: %d\n", d.sec);
    printf("Dia Semana: %d\n", d.wday);
    printf("Dia do Ano: %d\n", d.yday);
}
```



A Biblioteca `datetime.h`

Cria uma data (dia e mês) aleatória no ano atual

```
Datetime d = dt_rand_date();
```

Cria um horário aleatório no dia atual

```
Datetime d = dt_rand_time();
```

Cria uma data e horário aleatórios dos últimos 50 anos

```
Datetime d = dt_rand_birth(50);
```

Imprime uma estrutura do tipo `datetime`

```
dt_print(d);
```




A Biblioteca `datetime.h`

Cria uma data com base nos argumentos enviados...

```
Datetime d = dt_create(char str[],...);
```

str pode conter os seguintes caracteres e representações:

d	: dia	h	: horas
m	: mes	n	: minutos
y	: ano	s	: segundos

exemplos...

```
Datetime d = dt_create("dmy",4,3,1986);  
Datetime d = dt_create("dmhn",31,12,18,30);  
Datetime d = dt_create("ymdhn",2025,6,1,10,15);
```



Bora CODAR!!!



1. **Que dia da semana você nasceu?** Utilize a função `dt_create()` para criar um datetime, e com isso, consulte e imprima o dia da semana correspondente à data criada. Repita essa operação para várias datas informadas pelo usuário, até que se informe valores negativos (para encerrar a execução).
2. **Refatore o problema anterior (do cadastro de alunos), adicionando agora:**
A apresentação inicial do programa, que deve ser:

{Bom dia/Boa tarde/Boa noite}. Hoje é DD/MM/YYYY. Tenha uma ótima {???-feira}.

Cada Aluno deve conter sua **Data de Nascimento** cadastrada (use a função de geração aleatória de datas definida na lib `datetime.h`).

Após cadastrar a data de nascimento, calcule a idade atual do aluno (considerando se já completou aniversário no ano ou não). Se ele for menor de idade, o sistema deverá gerar um alerta para o usuário, avisando-o que deve estar acompanhado do pai ou responsável.