

Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

SIN 110

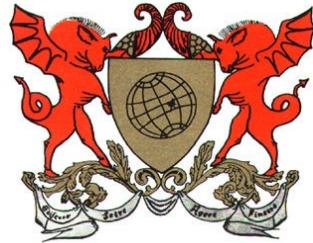
Programação

Sistemas de Informação

~~Prof. Guilherme C. Pena~~

~~guilherme.pena@ufv.br~~

Rodrigo Smarzano :)



Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

Aula de Hoje

Estruturas (Structs)

Estruturas de Dados

A linguagem C permite criar **tipos de dados definidos pelo próprio programador.**

Muitas vezes precisamos **compor** os dados para formar estruturas de dados **complexas**.



Estruturas de Dados

Variáveis compostas **homogêneas** (Array):
Conjunto de variáveis do **mesmo tipo**.

Variáveis compostas **heterogêneas** (Struct):
Conjunto de variáveis de **tipos diferentes**.



Estruturas de Dados

Em C, uma **estrutura (struct)** é uma coleção de variáveis referenciadas por um **nome**.

Isso fornece uma maneira conveniente de se ter informações relacionadas agrupadas.



Exemplo

Vimos em aulas anteriores programas que coletavam as notas dos alunos de uma sala e calculava a média da turma.

As únicas informações que podíamos tirar no final do programa eram **quantos!**

- **Quantos** alunos ficaram acima de média?
- **Quantos** alunos ficaram abaixo de média?



Exemplo

Agora, se ao invés de **QUANTOS**, nós quiséssemos saber **QUAIS** alunos ficaram acima ou abaixo da média??

Qual seria a solução?



Exemplo

Agora, se ao invés de **QUANTOS**, nós quiséssemos saber **QUAIS** alunos ficaram acima ou abaixo da média??

Qual seria a solução?

Uma possibilidade seria a utilização de um vetor para as notas e um outro vetor de strings para os nomes dos alunos onde índices identicos representam o mesmo aluno.



Exemplo

No caso desse programa, além das notas, quais outras informações dos alunos seriam relevantes para o nosso propósito?

- Nota
- Nome
- Matrícula

Se a gente tem outros dados para armazenar, o uso de vários vetores não fica muito prático!!!



Exemplo

Com o tipo estruturado (**struct**) nós podemos criar um novo tipo “Aluno” composto das informações que precisamos para o programa.



Definição de uma Estrutura em C

```
typedef struct rotulo_do_novo_tipo
{
    tipo1 nome1;
    tipo2 nome2;
    ...
    tipoN nomeN;
} identificador_do_novo_tipo;
```

A palavra-chave **struct** informa ao compilador que um novo rótulo de estrutura está sendo definido.



Definição de uma Estrutura em C

```
typedef struct rotulo_do_novo_tipo
{
    tipo1 nome1;
    tipo2 nome2;
    ...
    tipoN nomeN;
} identificador_do_novo_tipo;
```

A palavra-chave **typedef** informa ao compilador o identificador do novo tipo que representa a estrutura que está sendo definida.



Definição de uma Estrutura em C

```
typedef struct rotulo_do_novo_tipo
{
    tipo1 nome1;
    tipo2 nome2;
    ...
    tipoN nomeN;
} identificador_do_novo_tipo;
```

The diagram shows a C code snippet defining a new type. A red bracket on the right side of the code is labeled 'definição de tipo'. Two red arrows point from the word 'tipo' to the word 'tipo' in the bracket and from the word 'definição' to the word 'definição' in the bracket.

A palavra-chave **typedef** informa ao compilador o identificador do novo tipo que representa a estrutura que está sendo definida.



Definição de uma Estrutura em C

```
typedef struct rotulo_do_novo_tipo
{
    tipo1 nome1;
    tipo2 nome2;
    ...
    tipoN nomeN;
}identificador_do_novo_tipo;
```

As variáveis que compreendem a estrutura são chamadas de membros da estrutura. (Comumente chamados de elementos ou campos).



Definição de uma Estrutura em C

Geralmente todos os elementos na estrutura são logicamente relacionados.

Voltando ao exemplo:

```
typedef struct rotuloAluno
{
    char nome[30];
    int matricula;
    float nota;
}Aluno;
```



Definição de uma Estrutura em C

A definição de um tipo estruturado deve vir **antes** dos locais em que ele será usado e além disso, deve vir **antes da função int main()**.



Há variações

Em geral, tipos estruturados são definidos após a inclusão de bibliotecas.



Tipos de Estruturas

Existem três formas de se criar estruturas:

- Anônimas
- Rotuladas
- Rotuladas e Nomeadas



Estruturas Anônimas

Uma definição de estrutura anônima possui a seguinte forma:

```
struct
{
    tipo1 nome1;
    tipo2 nome2;
    ...
    tipoN nomeN;
} variavel1, variavel2;
```

Neste exemplo, duas variáveis são declaradas diretamente para a estrutura definida.



Estruturas Anônimas

```
#include <stdio.h>

struct
{
    int hora;
    int minuto;
    int segundo;
} H;

int main(void)
{
    H.hora = 10;
    H.minuto = 15;
    H.segundo = 30;
    printf("%d:%d:%d", H.hora, H.minuto, H.segundo);
    return 0;
}
```



Estruturas Anônimas

As estruturas anônimas não podem ser referenciadas em outras partes do código.

Alguns problemas do uso de estrutura anônima:

- Incentiva a criação de variáveis globais, o que pode não ser ideal para uma boa prática de programação.
- Impede a declaração de variáveis locais nas funções.



Estruturas Rotuladas

Estruturas Rotuladas criam um “rótulo” que pode ser referenciado posteriormente no código.

Criação de rótulos:

```
struct rotulo_da_estrutura
{
    tipo1 nome1;
    tipo2 nome2;
    ...
    tipoN nomeN;
};
```



Estruturas Rotuladas

```
#include <stdio.h>

struct sHora {
    int hora;
    int minuto;
    int segundo;
};

int main(void) {
    struct sHora H;
    H.hora = 10;
    H.minuto = 15;
    H.segundo = 30;
    printf("%d:%d:%d", H.hora, H.minuto, H.segundo);
    return 0;
}
```



Estruturas Rotuladas e Nomeadas

Uma estrutura rotulada e nomeada pode ser definida da seguinte forma:

```
typedef struct rotulo_da_estrutura
{
    tipo1 nome1;
    tipo2 nome2;
    ...
    tipoN nomeN;
} id_tipo_da_estrutura;
```

Essa estrutura pode ser referenciada de acordo com o nome que se dá ao `id_tipo_da_estrutura`.



Estruturas Rotuladas e Nomeadas

```
#include <stdio.h>

typedef struct Hora {
    int hora;
    int minuto;
    int segundo;
} THora;

int main(void) {
    THora H;
    H.hora = 10;
    H.minuto = 15;
    H.segundo = 30;
    printf("%d:%d:%d", H.hora, H.minuto, H.segundo);
    return 0;
}
```



Declarado como novo tipo antes da função main

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <locale.h>
4
5 typedef struct sEndereco ← rótulo
6 {
7     char rua[40];
8     int numero;
9     char cidade[30];
10    char estado[2];
11    int CEP;
12 } Endereco; ← identificador
13
14
15 int main(){
16
17     struct sEndereco ender1, ender2;
18     Endereco ender3;
```

Declarado como novo tipo antes da função main

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <locale.h>
4
5 typedef struct sEndereco
6 {
7     char rua[40];
8     int numero;
9     char cidade[30];
10    char estado[2];
11    int CEP;
12 } Endereco;
13
14
15 int main(){
16
17     struct sEndereco ender1, ender2;
18     Endereco ender3;
```

Declarar os tipos fora da função main mantém o código mais organizado.

Formas de declarar novas variáveis "struct"

usando o rótulo



usando o identificador

Podemos declarar structs sem rótulos, mas...

```
int main(){
    //struct SEM rótulo
    struct
    {
        char a;
        int b;
        float c;
    } struct1, struct2, struct3;
```



...depois não conseguiremos reaproveitar a estrutura da struct em novas declarações

Podemos criar a struct com rótulo e já declarar algumas variáveis

```
int main(){
    //struct COM rótulo
    struct comRotulo
    {
        char a;
        int b;
        float c;
    } struct1, struct2, struct3;
    typedef struct comRotulo minhaStruct;

    struct comRotulo struct4;
    minhaStruct struct5;
```

rótulo

variáveis

Criar identificador para struct com rótulo

Se a struct possui rótulo, podemos reutilizá-la

Se possui identificador, também podemos utilizar na declaração

Representação na Memória

Após sua definição, os elementos de um tipo estruturado ocupam posições consecutivas na memória.

```
typedef struct rotuloAluno
{
    char nome[30];
    int matricula;
    float nota;
}Aluno;
```



Declaração de uma variável Estruturada

Para declararmos uma variável estruturada, segue-se a ideia de declaração de uma variável normal usando o **identificador** que foi colocado na **struct**.

```
identificador_do_novo_tipo nome_da_variavel;
```

Exemplo (Declaração):

Aluno a;

struct rotuloAluno a;

```
typedef struct rotuloAluno  
{  
    char nome[30];  
    int matricula;  
    float nota;  
}Aluno;
```



Acessando membros de uma variável Estruturada

Para acessar os **membros** de uma variável estruturada utilizamos o operador **(.) ponto**. Após isso eles são vistos como variáveis normais.

```
typedef struct rotuloAluno
{
    char nome[30];
    int matricula;
    float nota;
}Aluno;
```

equivalentes

```
int main()
{
    Aluno a;//struct rotuloAluno b;
    gets(a.nome);
    scanf("%d", &a.matricula);
    scanf("%f", &a.nota);
}
```



Outro Exemplo

Uma estrutura de endereço, que possa ser usada como se fosse um tipo de dado posteriormente:

```
typedef struct sEndereco
{
    char rua[40];
    int numero;
    char cidade[30];
    char estado[3];
    char CEP[10];
}Endereco;
```



Outro Exemplo

```
int main(){
    //cria variável ender1 do tipo Endereco
    Endereco ender1;
    strcpy(ender1.rua, "Rua 7 de Setembro");
    ender1.numero = 405;
    strcpy(ender1.cidade, "Goiania");
    strcpy(ender1.estado, "GO");
    ender1.CEP = "38810-000";
}
```



Outro Exemplo

Definindo, declarando, atribuindo e imprimindo.

```
#include <stdio.h>
typedef struct sRetangulo{
    float altura, largura;
} Retangulo;

int main(){
    Retangulo ret1;
    ret1.altura = 10;
    printf("Digite o valor da largura: ");
    scanf("%f", &ret1.largura);
    printf("Altura: %.1f\n", ret1.altura);
    printf("Largura: %.1f\n", ret1.largura);
    return 0;
}
```



Estruturas Aninhadas

São estruturas em que um ou mais de seus membros
também ~~São~~ sejam estruturas:

```
typedef struct rotulo_estrutura1 {  
    tipo1 nome1;  
    tipoN nomeN;  
} id_estrutura1;  
  
typedef struct rotulo_estrutura2 {  
    id_estrutura1 nome;  
    tipoN nomeN;  
} id_estrutura2;
```



Estruturas Aninhadas

Exemplo:

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

typedef struct sHora
{
    int hora;
    int minuto;
    int segundo;
} Hora;

typedef struct sRelogio
{
    Hora H;
    char modelo[10];
} Relogio;
```



Estruturas Aninhadas

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

typedef struct sHora
{
    int hora;
    int minuto;
    int segundo;
} Hora;

typedef struct sRelogio
{
    Hora H;
    char modelo[10];
} Relogio;
```

```
int main(void)
{
    Relogio r1;

    r1.H.hora = 10;
    r1.H.minuto = 15;
    r1.H.segundo = 30;

    strcpy(r1.modelo, "Cassio");

    printf("Modelo: %s\n", r1.modelo);

    printf("%d:%d:%d", r1.H.hora,
           r1.H.minuto,
           r1.H.segundo);

    return 0;
}
```



Arrays e Estruturas

Como vimos, é possível combinar arrays e estruturas para criação de diferentes estruturas de dados.

Podemos ter uma estrutura contendo um membro do tipo array, ou

Criar um array cujo os elementos sejam estruturas



Definindo estruturas com arrays

```
typedef struct slivro
{
    char titulo[30];
    char autor[30];
    int regnum;
    float preco;
} livro;
```

Membros do tipo
array



Declarando arrays de estruturas

```
typedef struct slivro
{
    char titulo[30];
    char autor[30];
    int regnum;
    float preco;
} livro;
```

Membros do tipo
array

```
livro livros[50];
```

Array do tipo
livro



Declarando arrays de estruturas

```
livro livros[50];
```

- **livros** é um vetor de 50 elementos.
- Cada elemento do vetor é uma estrutura do tipo struct **livro**
- O que significa **livros[0]**, **livros[1]**, **livros[2]**, etc?

** Através dessa instrução o compilador providencia espaço de memória para 50 estruturas do tipo **livro**.



Arrays e Estruturas

```
...
typedef struct sEndereco{
    char rua[40];
    int numero;
}Endereco;

int main() {
    Endereco listaend[5];           vetor da struct Endereco
    listaend[0].numero = 100;        cada elemento do vetor é uma struct completa
    strcpy(listaend[3].rua, "Av. Brasil");

    return 0;
}
```



Arrays e Estruturas

```
...
typedef struct sEndereco{
    char rua[40];
    int numero;
}Endereco;
int main(){
    Endereco listaend[5];
    listaend[0].numero = 100;
    strcpy(listaend[3].rua, "Av. Brasil");
    return 0;
}
```

representação na memória

ID da estrutura

listaend:



Arrays e Estruturas

```
#include <stdio.h>
typedef struct sHora{
    int hor, min, seg;
} Hora;

int main() {
    Hora H[5]; //Vetor de horas com 5 posicoes
    //Usando apenas a primeira posicao do vetor.
    H[0].hor = 22;
    H[0].min = 05;
    H[0].seg = 23;
    printf("%d:%d:%d\n", H[0].hor, H[0].min, H[0].seg);

    return 0;
}
```

| | hor | min | seg |
|-----|-----|-----|-----|
| [0] | 22 | 5 | 23 |
| [1] | | | |
| [2] | | | |
| [3] | | | |
| [4] | | | |



Exercícios

- 1) Implemente um programa que leia e imprima os dados de um produto (nome, preço).

- 2) Utilizando estrutura, faça um programa que permita a entrada de nome, endereço e telefone de 5 pessoas e os imprima.



Exercícios

3) Faça um programa que leia um vetor com dados de 5 livros: título (máximo 30 letras), autor (máximo 15 letras) e ano.

- Procure um livro por título, perguntando ao usuário qual título deseja buscar e mostre os dados de autor e ano do livro encontrado.
 - Procure um livro por autor, e mostre os títulos e anos de todos os livros desse autor.
 - Procure um livro por ano e mostre os títulos e autores de todos desse ano.
-



Exercícios

- 4) Codifique um programa que leia e imprima os dados de vários produtos (preço, nome, desconto, disponibilidade(Sim/Não)) até que o usuário digite um preço negativo. Em seguida imprima:
- (a) O produto mais caro e o mais barato.
 - (b) O produto com maior desconto.
 - (c) O produto disponível com maior desconto.
 - (d) O produto disponível com preço final mais baixo.



Solução Exercício 1

```
1 #include<stdio.h>
2
3 typedef struct sProduto{
4     char nome[30];
5     float preco;
6 } Produto;
7
8 int main () {
9
10    Produto produto;
11
12    printf("Digite o nome do produto.: ");
13    fflush(stdin);
14    gets(produto.nome);
15
16    printf("Digite o preco do produto: ");
17    scanf("%f", &produto.preco);
18
19    printf("\nProduto: %s\nPreco...: R$ %0.2f\n", produto.nome, produto.preco);
20
21    return 0;
22 }
```

Solução Exercício 2

```
1 #include<stdio.h>
2
3 typedef struct sEndereco{
4     char rua[30];
5     int numero;
6     char bairro[30];
7     char cidade[30];
8     char estado[3];
9 }Endereco;
10
11 typedef struct sPessoa{
12     char nome[30];
13     Endereco endereco;
14     char telefone[10];
15 }Pessoa;
16
17 int main () {
18
19     Pessoa pessoas[5];
20     int i;
21
22     for (i = 0; i < 5; i++) {
23         printf("Digite o nome da pessoa %d: \n", i+1);
24         gets(pessoas[i].nome);
25         fflush(stdin);
26
27         printf("Digite a rua da pessoa %d: \n", i+1);
28         gets(pessoas[i].endereco.rua);
29         fflush(stdin);
30
31         printf("Digite o numero do endereco da pessoa %d: \n", i+1);
32         scanf(" %d", &pessoas[i].endereco.numero);
33         fflush(stdin);
34
35         printf("Digite o bairro da pessoa %d: \n", i+1);
36         gets(pessoas[i].endereco.bairro);
37         fflush(stdin);
38
39         printf("Digite a cidade da pessoa %d: \n", i+1);
40         gets(pessoas[i].endereco.cidade);
41         fflush(stdin);
42
43         printf("Digite o UF da pessoa %d: \n", i+1);
44         gets(pessoas[i].endereco.estado);
45         fflush(stdin);
46
47         printf("Digite o telefone da pessoa %d: \n", i+1);
48         gets(pessoas[i].telefone);
49         fflush(stdin);
50
51         printf("\n");
52     }
53
54     for (i = 0; i < 5; i++) {
55         printf("\nDados da pessoa %d:", i+1);
56         printf("\nNome....: %s", pessoas[i].nome);
57         printf("\nEndereco: %s", pessoas[i].endereco.rua);
58         printf("\nNumero...: %d", pessoas[i].endereco.numero);
59         printf("\nBairro...: %s", pessoas[i].endereco.bairro);
60         printf("\nCidade...: %s", pessoas[i].endereco.cidade);
61         printf("\nEstado...: %s", pessoas[i].endereco.estado);
62         printf("\nTelefone: %s", pessoas[i].telefone);
63         printf("\n");
64     }
65
66
67 }
```