



# Estruturas de Dados 1

## 04 – Estruturas Heterogêneas - structs

Antonio Angelo de Souza Tartaglia  
angelot@ifsp.edu.br

# Estrutura de Dados 1

## Tipos Primitivos (básicos) de Variáveis



- Variável composta homogênea:
  - Estrutura de dados composta por elementos de um **mesmo tipo**:
    - Matrizes e vetores;
    - Tipos char, int, double, float, etc.
- Variável composta heterogênea ou estrutura:
  - Permite agrupar uma coleção de um ou mais elementos de **tipos diferentes**:
    - char, int, double, array, etc.

Dependendo da situação estes tipos não são suficientes!

# Estrutura de Dados 1

## Tipos Básicos de Variáveis

- A linguagem C permite criar tipos de dados definíveis pelo usuário de cinco formas diferentes.
- O primeiro é a **estrutura**, que é um agrupamento de variáveis sob um nome e é chamado tipo de dado agregado;
- O segundo tipo definido pelo usuário é o **campo de bit**, que é uma variação da estrutura e permite o fácil acesso aos bits dentro de uma palavra;
- O terceiro é a **união**, que permite que um mesmo trecho de memória seja definido e utilizado por dois ou mais tipo diferentes de variáveis;
- O quarto tipo de dado definível pelo usuário é a **enumeração**, que é uma lista de símbolos;
- O quinto e último tipo definido pelo usuário é criado através do uso de **typedef** e define um novo nome para um tipo já existente.



# Estrutura de Dados 1

## Estrutura ou Variáveis de Registro

- Em C, uma estrutura (struct) é uma coleção de variáveis referenciadas por um único nome, fornecendo uma maneira conveniente de se ter informações que são relacionadas, agrupadas sob um mesmo nome;
- Permite organizar um grupo de variáveis como uma única variável;
- Estas variáveis que compõem o registro ou estrutura, são chamadas de **elementos**, **campos** ou **membros**, e cada uma delas pode ser de **qualquer tipo inclusive de outras Estruturas criadas**;
- Sintaxe da declaração:

Registro {

```
struct nome_do_tipo_do_registro{  
    tipo1 campo1;  
    tipo2 campo2;  
    ...  
    tipon campon;  
};
```

Elementos, campos ou membros }

Ideia:

Criar apenas um tipo de dado que contenha vários membros, ou seja, uma variável que contém outras variáveis que por sua vez estão logicamente relacionadas.



# Estrutura de Dados 1

## Estrutura ou Variáveis de Registro

- As estruturas também são chamadas de **Variáveis de Registro** e podem ser declaradas em qualquer escopo do programa: elas podem ser **globais** ou **locais**;
- Porém, a maioria das estruturas **são declaradas no escopo global**. Por se tratar de um novo tipo de dado, muitas vezes é interessante que todo o programa tenha acesso à estrutura;
- Exemplo de declaração:

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

```
struct funcionario{  
    int ID;  
    char nome[30];  
    int idade;  
    float salario;  
};
```

Note que nesta declaração de estrutura, a variável propriamente dita **ainda não existe...**

Neste momento, esta declaração descreve apenas o formato que a variável terá.

Observe também que os campos da estrutura são definidos da mesma forma que definimos as variáveis.

A definição termina com ponto e vírgula. Isso porque uma definição de estrutura **é um comando**.



# Estrutura de Dados 1

## Estrutura ou Variáveis de Registro

- Por ser um tipo declarado pelo programador, usa-se a palavra **struct** antes do tipo da nova variável;
- Sintaxe da declaração de variáveis do tipo registro:

```
struct nome_do_tipo_do_registro variavel_reg;  
  
struct nome_do_tipo_do_registro variavel_reg =  
{lista de valores};
```

Quando uma variável do tipo estrutura é declarada, como **func1** ou **func2**, o compilador aloca automaticamente memória suficiente para todos os seus membros.

Neste instante a variável ou registro "func1", passa a existir com a estrutura "struct funcionário"

- Exemplo:

```
struct funcionario func1;  
struct funcionario func2={10,"Paulo",20, 1550.80};
```

```
struct funcionario{  
    int ID;  
    char nome[30];  
    int idade;  
    float salario;  
};
```



# Estrutura de Dados 1

## Estrutura ou Variáveis de Registro

- Por questões de simplificação e por se tratar de um novo tipo, é possível logo na definição da struct, declarar algumas variáveis desse novo tipo. Basta adicionar o nome das variáveis que se quer declarar, após a chave de fechamento da estrutura e antes do ponto e vírgula “;”.

Neste instante, foram declaradas as variáveis do tipo **struct endereço**, “end1” e “end2”.

```
struct endereco{  
    char nome[30];  
    char endereco[40];  
    char cidade[20];  
    char estado[2];  
    int cep;  
}end1, end2;
```

Se o programa só precisa de uma estrutura, o nome não é necessário e pode ser suprimido, criando-se a variável diretamente

```
struct{  
    char nome[30];  
    char endereco[40];  
    char cidade[20];  
    char estado[2];  
    int cep;  
}endereco;
```



# Estrutura de Dados 1

## Estrutura ou Variáveis de Registro

- O uso de estruturas facilita em muito a vida do programador na manipulação dos dados do programa. Imagine ter que declarar quatro cadastros para quatro funcionários diferentes:

```
int id1, id2, id3, id4;  
char nome1[30], nome2[30], nome3[30], nome4[30];  
int idade1, idade2, idade3, idade4;  
float salario1, salario2, salario3, salario4;
```

- Utilizando uma estrutura, o mesmo pode ser feito da seguinte maneira:

```
struct funcionario func1, func2, func3, func4;
```





# Estrutura de Dados 1

## Estrutura ou Variáveis de Registro

- Uma vez definida uma variável do tipo estrutura, é preciso poder acessar seus campos para se trabalhar, e assim, referenciar seus campos dentro do registro:
  - Os campos internos são acessados por meio do operador (•) → **variavel\_reg • campo**

```
struct cadastro{  
    int conta;  
    char nome[30];  
    int idade;  
    float saldo_conta;  
};
```

```
struct cadastro correntista;
```

```
correntista = {845762, "Marcelo da silva", 35, 3485.49};
```

```
char cliente[30];
```

```
float saldo;
```

```
strcpy(cliente, correntista.nome);
```

```
saldo = correntista.saldo_conta;
```

- Cada variável dentro da estrutura pode ser acessada como se apenas ela existisse, não sofrendo nenhuma interferência das outras. Como cada campo é independente dos demais, outros operadores podem ser aplicados a cada campo. Por exemplo, na estrutura acima podemos comparar o campo idade entre dois cadastros.

```
if(correntista1.saldo_conta > correntista2.saldo_conta)
```



# Estrutura de Dados 1

## Estrutura ou Variáveis de Registro

- Atribuição de valores:

```
variavel_reg.campo = novo_valor;
```

- Exemplo:

```
struct funcionario func1, func2;  
  
//atribuição de campos ou elementos  
strcpy(func1.nome, "Maria da Silva");  
func1.id = 123;  
func1.idade = 37;  
func1.salario = 2390,35;  
  
//atribuição de registro ou estruturas  
func2 = func1;
```

```
struct funcionario{  
    int    id;  
    char   nome[30];  
    int    idade;  
    float  salario;  
};
```



# Estrutura de Dados 1

## Estrutura ou Variáveis de Registro

- Atribuição entre estruturas:

- As únicas operações possíveis em uma estrutura, são as de acesso aos membros da estrutura, por meio do operador ponto (.), e as cópias de atribuição por meio do operador de atribuição (=).
- A atribuição entre duas variáveis de estrutura faz com que o conteúdo das variáveis contidos dentro de uma estrutura sejam copiados para a outra, exemplo:

```
struct cadastro{  
    char nome[50];  
    int idade;  
    char rua[50];  
    int numero;  
};  
  
struct cliente{  
    char nome[50];  
    int idade;  
    char rua[50];  
    int numero;  
};
```

```
struct cadastro cad1, cad2;  
struct cliente cli;  
  
cad1 = {"Onofre", 25, "Rua do Porto", 352};  
cad2 = cad1 //todo o conteúdo é copiado para cad2  
  
cli = cad1;//ERRO! estruturas diferentes
```

- No exemplo acima, apesar das estruturas “cadastro” e “cliente” possuírem os mesmos campos, são estruturas diferentes, com nomes diferentes...



# Estrutura de Dados 1

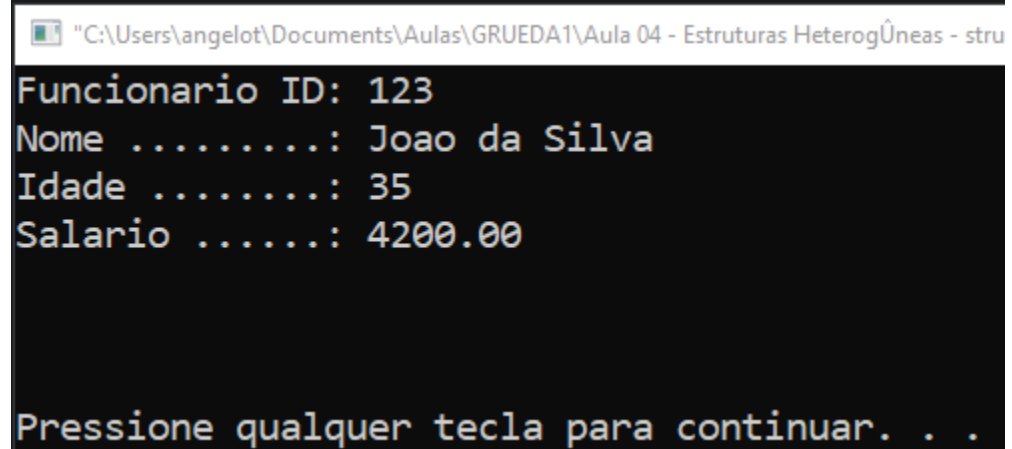
## Estrutura ou Variáveis de Registro – programa exemplo

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

```
struct funcionario{
    int id;
    char nome[31];
    int idade;
    float salario;
};
```

```
int main(){
    struct funcionario func;
    func.id = 123;
    strcpy(func.nome, "Joao da Silva");
    func.idade = 35;
    func.salario = 4200.00;

    printf("Funcionario ID: %d \n", func.id);
    printf("Nome .....: %s \n", func.nome);
    printf("Idade .....: %d \n", func.idade);
    printf("Salario .....: %.2f \n", func.salario);
    printf("\n\n\n");
    system("pause");
}
```



```
"C:\Users\angelot\Documents\Aulas\GRUEDA1\Aula 04 - Estruturas Heterogêneas - stru"
Funcionario ID: 123
Nome .....: Joao da Silva
Idade .....: 35
Salario .....: 4200.00

Pressione qualquer tecla para continuar. . .
```



# Estrutura de Dados 1

## Matrizes de Registros

- Para declarar uma matriz ou vetor de estruturas, primeiro é necessário que a estrutura do registro esteja definida;
- Em seguida pode-se declarar a matriz ou vetor deste tipo de estrutura.

```
struct nome_do_tipo_do_registro matriz_reg[tam];
```

- Exemplo:

```
struct funcionario func[100];
```

```
struct funcionario{  
    int    id;  
    char   nome[30];  
    int    idade;  
    float  salario;  
};
```

Id	Id	Id	Id	Id	Id	Id	Id
Nome	Nome	Nome	Nome	Nome	Nome	Nome	Nome
Idade	Idade	Idade	Idade	Idade	Idade	Idade	Idade
Salário	Salário	Salário	Salário	Salário	Salário	Salário	Salário
0	1	2	3	4	5	6	n



# Estrutura de Dados 1

## Matrizes de Registros

- Para acessar um registro específico devemos indexar o nome do registro:

```
matriz_reg[i].campo;
```

- Exemplo:

```
struct funcionario{  
    int    id;  
    char   nome[30];  
    int    idade;  
    float  salario;  
};
```

```
printf("ID do funcionario 3: %d", func[2].id);
```

Observe a posição  
do índice!!

func

Id	Id	Id	Id	Id	Id	Id	Id
Nome	Nome	Nome	Nome	Nome	Nome	Nome	Nome
Idade	Idade	Idade	Idade	Idade	Idade	Idade	Idade
Salário	Salário	Salário	Salário	Salário	Salário	Salário	Salário
0	1	2	3	4	5	6	n



# Estrutura de Dados 1

## Matrizes de Registros



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

```
struct pessoa{
    char nome[31];
    int idade;
};
```

```
int main(){
    struct pessoa p[2];
    strcpy(p[0].nome, "Joao da Silva");
    p[0].idade = 35;
    strcpy(p[1].nome, "Maria Eduarda");
    p[1].idade = 27;

    printf("Dados pessoa 1: \n");
    printf("Nome ...: %s \n", p[0].nome);
    printf("Idade ..: %d \n", p[0].idade);
    printf("\n\n");
    printf("Dados pessoa 2: \n");
    printf("Nome ...: %s \n", p[1].nome);
    printf("Idade ..: %d \n", p[1].idade);
    printf("\n\n\n");
    system("pause");
}
```

P

Joao da Silva 35	Maria Eduarda 27
0	1

"C:\Users\angelot\Documents\Aulas\GRUEDA1\Aula 04 - Estruturas Heterogêneas - stru

```
Dados pessoa 1:
Nome ...: Joao da Silva
Idade ..: 35
```

```
Dados pessoa 2:
Nome ...: Maria Eduarda
Idade ..: 27
```

```
Pressione qualquer tecla para continuar. . .
```

# Estrutura de Dados 1

## Atividade 1

- Reescreva o programa do exemplo anterior substituindo a estrutura “pessoa” pela estrutura “funcionário” ao lado;
- Modifique o programa para que o usuário possa fornecer os dados necessários relativos aos funcionários de uma empresa fictícia. Você deve gerar entradas para 5 funcionários;
- Imprima todo o vetor com todos os dados funcionários na tela no estilo de um relatório;
- Entregue no Moodle como Atividade 1.

```
struct funcionario{  
    int ID;  
    char nome[30];  
    int idade;  
    float salario;  
};
```

Ao coletar o nome do funcionário utilize a função “fgets”, porém este comando capturará o “enter” do passo anterior gerado pelo scanf(). Para sanar este problema utilize a função “getchar( )” na linha anterior à linha que contém o “fgets”. Dessa forma, “getchar( )” capturará o “enter” que ficou preso no buffer no passo anterior. Também é possível utilizar o comando “fflush(stdin)”, que limpará completamente o buffer de teclado.





# Estrutura de Dados 1

## Composição de Registros

- Uma estrutura pode agrupar um número arbitrário (qualquer quantidade) de variáveis de tipos diferentes.
- Uma estrutura também é um tipo de dado, com a diferença de que se trata de um tipo de dado criado pelo programador. Portanto é possível a declaração de uma estrutura que contenha outra estrutura **previamente definida**. A esta configuração damos o nome de **estruturas aninhadas**. Sintaxe:

```
struct nome_do_tipo_do_registro{  
    struct nome_de_outra_estrutura campo1;  
    tipo2 campo2;  
    ...  
    tipon campon;  
};
```



# Estrutura de Dados 1

## Composição de Registros

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

```
struct data{
    int dia, mes, ano;
};

struct cadastro_aluno{
    int prontuario;
    char nome[31];
    struct data nascimento;
};
```

```
int main(){
    struct cadastro_aluno aluno;
    aluno.prontuario = 123456;
    strcpy(aluno.nome, "Jose da Silva");
    aluno.nascimento.dia = 30;
    aluno.nascimento.mes = 02;
    aluno.nascimento.ano = 2015;
}
```

Observe que a estrutura "struct data", foi declarada antes da estrutura "struct cadastro\_aluno". Ela deve existir antes de ser declarada em outra estrutura!

O operador "." é utilizado para acessar a variável "nascimento". Note que "nascimento" também é uma estrutura, e para acessar seus elementos também utilizamos o operador "." de forma encadeada.



# Estrutura de Dados 1

## Passagem de registros para funções

- Passando campos de registros para funções, exemplo:

```
#includes ....
struct pessoa{
    char nome[30];
    int idade;
};

int main(){
    struct pessoa p;
    strcpy(p.nome, "Jose da Silva");
    p.idade = 34;

    funcao1(p.nome);
    funcao2(p.idade);

    funcao3(&p.nome);
    funcao4(&p.idade);
}
```

```
int funcao1(char vetor_nome[]){
    //comandos
}

int funcao2(int idade){
    //comandos
}

int funcao3(char *ptr_nome){
    //comandos
}

int funcao4(int *ptr_idade){
    //comandos
}
```

Passagem por  
cópia simples.  
Modo normal de  
passagem de  
parâmetros

Passagem do endereço inicial  
da estrutura p



## Passagem de registros inteiros para funções

```
#includes ....
struct pessoa{
    char nome[30];
    int idade;
};

void imprimeTela(struct pessoa p_exibir);

int main(){
    struct pessoa p;
    strcpy(p.nome, "Jose da Silva");
    p.idade = 34;

    imprimeTela(p);
}

void imprimeTelaTela(struct pessoa p_exibir){
    printf("Nome: %s, idade %d\n", p_exibir.nome, p_exibir.idade);
}
```

- ▶ Quando um registro é usado como argumento para uma função, o registro inteiro é passado usando o método padrão de passagem por valor (cópia). Lembre-se que o tipo de argumento, que é passado na chamada da função, deve coincidir com o tipo de parâmetro que é esperado pela função:



# Estrutura de Dados 1

## Retornando um registro

- Quando a função retornar um registro:

```
#includes ....
struct pessoa{
    char nome[30];
    int idade;
};

struct pessoa coletaDados();

int main(){
    struct pessoa p;
    p = coletaDados();
    printf("Nome: %s, idade %d\n", p.nome, p.idade);
}
```

```
struct pessoa coletaDados(){
    struct pessoa pDataIn;
    printf("Digite o nome da pessoa:");
    fgets(pDataIn.nome, 29, stdin); //Jose da Silva
    printf("Digite a idade da pessoa:\n");
    scanf(" %d", &pDataIn.idade);    //34

    return pDataIn;
}
```

Função coletaDados(), declara uma nova instância do tipo struct pessoa (pDataIn) da estrutura ou registro do tipo pessoa e em seguida atribui valores coletados via teclado aos seus elementos ou campos.



# Estrutura de Dados 1

## Retornando um registro e passando para uma função

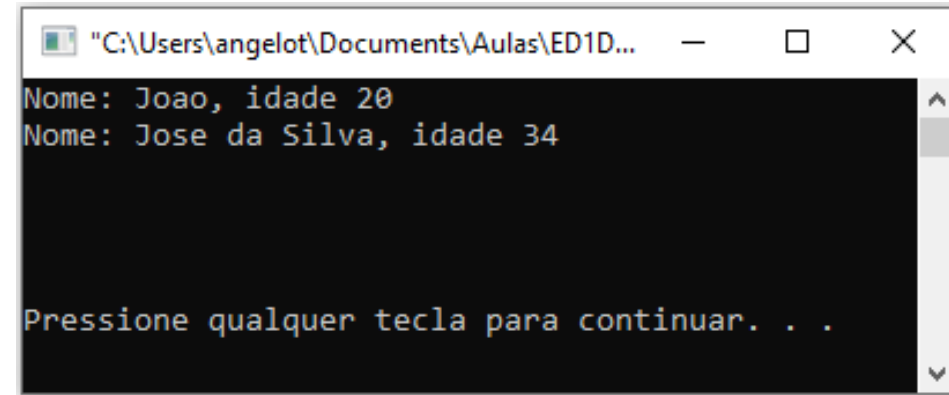
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
struct pessoa{
    char nome[30];
    int idade;
};
```

```
void printTela(struct pessoa vetor_p);
struct pessoa coletaDados();
```

```
int main(){
    struct pessoa p1, p2;
    strcpy(p1.nome, "Joao");
    p1.idade = 20;
    p2 = coletaDados();

    printTela(p1);
    printTela(p2);
    printf("\n\n\n\n");
    system("pause");
}
```



```
struct pessoa coletaDados(){
    struct pessoa pDataIn;
    printf("Digite o nome da pessoa:");
    fgets(pDataIn.nome, 29, stdin); //Jose da Silva
    //abaixo, substitui \n por \0 na string
    pDataIn.nome[strlen(pDataIn.nome) - 1] = '\0';
    printf("Digite a idade da pessoa:");
    scanf(" %d", &pDataIn.idade); //34
    system("cls"); //apaga a tela
    return pDataIn;
}

void printTela(struct pessoa p){
    printf("Nome: %s, idade %d\n", p.nome, p.idade);
}
```



# Estrutura de Dados 1

## Passando um vetor de structs para uma função

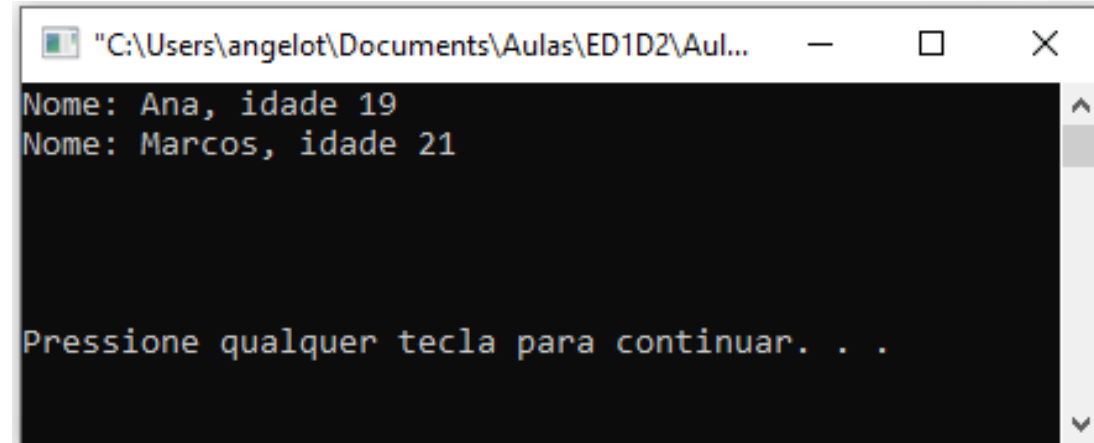
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct pessoa{
    char nome[30];
    int idade;
};

void printTela(struct pessoa vetor_p[], int tam);

int main() {
    struct pessoa vetor_pessoa[2];
    strcpy(vetor_pessoa[0].nome, "Ana");
    vetor_pessoa[0].idade = 19;
    strcpy(vetor_pessoa[1].nome, "Marcos");
    vetor_pessoa[1].idade = 21;

    printTela(vetor_pessoa, 2);
    printf("\n\n\n\n");
    system("pause");
}
```



```
"C:\Users\angelot\Documents\Aulas\ED1D2\Aul...
Nome: Ana, idade 19
Nome: Marcos, idade 21

Pressione qualquer tecla para continuar. . .
```

```
printTela(struct pessoa vetor_p[], int tam){
    int i;
    for(i = 0; i < tam; i++){
        printf("Nome: %s, idade %d\n",
            vetor_p[i].nome, vetor_p[i].idade);
    }
}
```



# Estrutura de Dados 1

## Estruturas – passagem de parâmetros por referência

- Em C existe apenas a passagem de parâmetros para funções por valor.
- Se é absolutamente necessário que uma função que recebe uma estrutura, modifique o seu conteúdo sem devolvê-la ao final de sua execução, **é necessário a passagem de um ponteiro para a estrutura**, devendo receber esse endereço em um parâmetro do tipo **ponteiro**, podendo assim modificar o conteúdo dos campos dessa estrutura em seu local de origem.







## Estruturas – passagem de parâmetros por referência

- Ao recebermos um ponteiro para um tipo estrutura, devemos ter alguns cuidados:
  - No exemplo anterior se tivéssemos passado um ponteiro para a estrutura nos parâmetros da função `coletaDados()` para que esta executasse um processamento qualquer, por exemplo, coletar os dados do usuário; teríamos um problema de precedência;

```
struct pessoa p1, p2;    void coletaDados(struct pessoa *pDatain){
coletaDados(&p1);        //comandos
coletadatos(&p2);        fgets(*pDatain.nome, 30, stdin);
                          //comandos
                          }
```

- Para o compilador, o operador ponto (.) tem precedência sobre o operador asterisco (\*), com isso a execução do programa tentaria encontrar primeiro o campo “idade” do elemento passado, mas isso não seria possível já que o que foi passado é apenas um ponteiro, ou seja o endereço, e não a estrutura completa;
- Isso é resolvido com a utilização de parênteses “()”, forçando a resolução do ponteiro, encontrando assim a estrutura na memória, antes do acesso ao campo da estrutura.

# Estrutura de Dados 1

## Estruturas – passagem de parâmetros por referência

- Assim teríamos que acessar o campo da seguinte forma:

```
void coletaDados(struct pessoa *pDataain){  
    //comandos  
    fgets((*pDataain).nome, 30, stdin);  
    //comandos  
}
```

- Isso torna a programação um tanto confusa e pode ser simplificado.
- A Linguagem C coloca a disposição o operador “->” (sinal de menos seguido do sinal de maior que), que permite simplificar a expressão anterior:

```
void coletaDados(struct pessoa *pDataain){  
    //comandos  
    fgets(pDataain->nome, 30, stdin);  
    //comandos  
}
```



# Estrutura de Dados 1

## Estruturas – passagem por referência utilizando o operador seta

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

struct pessoa{
    char nome[30];
    int idade;
};

void printTela(struct pessoa vetor_p);
void coletaDados(struct pessoa *p);

int main() {
    struct pessoa p1, p2;
    strcpy(p1.nome, "Joao");
    p1.idade = 20;
    coletaDados(&p2);
    printf("Agora utilizando ponteiros!\n\n");
    printTela(p1);
    printTela(p2);
    printf("\n\n\n\n");
    system("pause");
}
```

```
void coletaDados(struct pessoa *pDataIn){
    printf("Digite o nome da pessoa:");
    fgets(pDataIn->nome, 29, stdin); //Jose da Silva
    //linha abaixo, substitui \n por \0 na string
    pDataIn->nome[strlen(pDataIn->nome) - 1] = '\0';
    printf("Digite a idade da pessoa:");
    scanf(" %d", &pDataIn->idade); //34
    system("cls"); //apaga a tela
}

void printTela(struct pessoa p){
    printf("Nome: %s, idade %d\n", p.nome, p.idade);
}
```

"C:\Users\angelot\Documents\Aulas\GRUEDA1\Aula 04 - Estruturas Heterogêneas - stru

Agora utilizando ponteiros!

Nome: Joao, idade 20

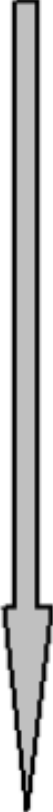
Nome: Jose da Silva, idade 34

Pressione qualquer tecla para continuar. . .



# Estrutura de Dados 1

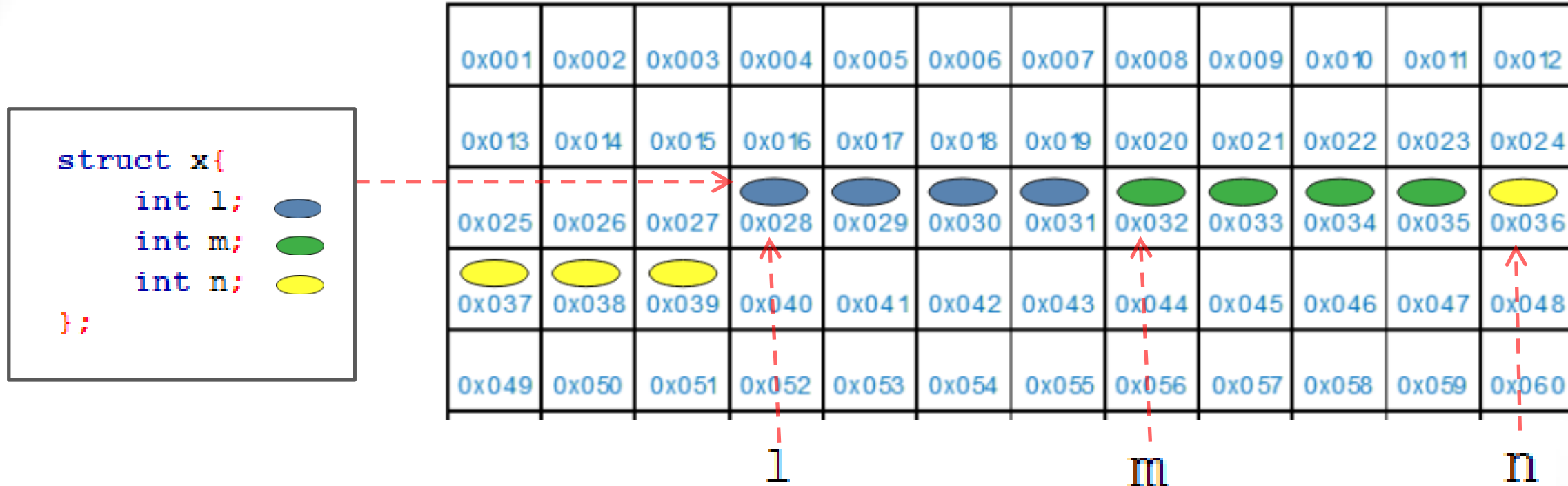
## Tabela de precedência dos operadores

GRUPOS DE OPERADORES	ASSOCIATIVIDADE	PRECEDÊNCIA
() , [] , -> , •	Esquerda para direita	 Alta
( <i>tipo</i> ) [ <i>conversão explícita</i> ], sizeof, & [ <i>endereço</i> ], * [ <i>indireção</i> ], -, +, ~, ++, --, ! [ <i>todos neste grupo são unários</i> ]	Direita para Esquerda	
* [ <i>multiplicação</i> ], /, %	Esquerda para direita	
+, - [ <i>binários</i> ]	Esquerda para direita	
<<, >>	Esquerda para direita	
<, <=, >, >=	Esquerda para direita	
==, !=	Esquerda para direita	
& [ <i>conjunção sobre bits</i> ]	Esquerda para direita	
^	Esquerda para direita	
	Esquerda para direita	
&&	Esquerda para direita	
	Esquerda para direita	
? :	Direita para esquerda	
=, +=, -=, *=, /=, %=, <<=, >>=, &=,  =, ^=	Direita para esquerda	
, [ <i>vírgula</i> ]	Esquerda para direita	
		Baixa



# Estrutura de Dados 1

Estruturas – passagem de parâmetros por referência, considerações:






- Se  $x$  é uma estrutura, então  $\&x$  devolve o endereço da estrutura em memória (0x028), isto é, o **menor** ou o **início** dos endereços ocupados pela estrutura na memória.
- Se  $x$  é uma estrutura e  $m$  um campo dessa estrutura, então  $\&x.m$  devolve o endereço de memória do membro  $m$  da estrutura  $x$  (0x032).



# Estrutura de Dados 1



Estruturas – passagem de parâmetros por referência, considerações:

```
struct x{  
    int l;   
    int m;   
    int n;   
};
```

P	0x001	0x002	0x003	0x004	0x005	0x006	0x007	0x008	0x009	0x010	0x011	0x012
	0x013	0x014	0x015	0x016	0x017	0x018	0x019	0x020	0x021	0x022	0x023	0x024
	0x025	0x026	0x027	0x028	0x029	0x030	0x031	0x032	0x033	0x034	0x035	0x036
	0x037	0x038	0x039	0x040	0x041	0x042	0x043	0x044	0x045	0x046	0x047	0x048
	0x049	0x050	0x051	0x052	0x053	0x054	0x055	0x056	0x057	0x058	0x059	0x060

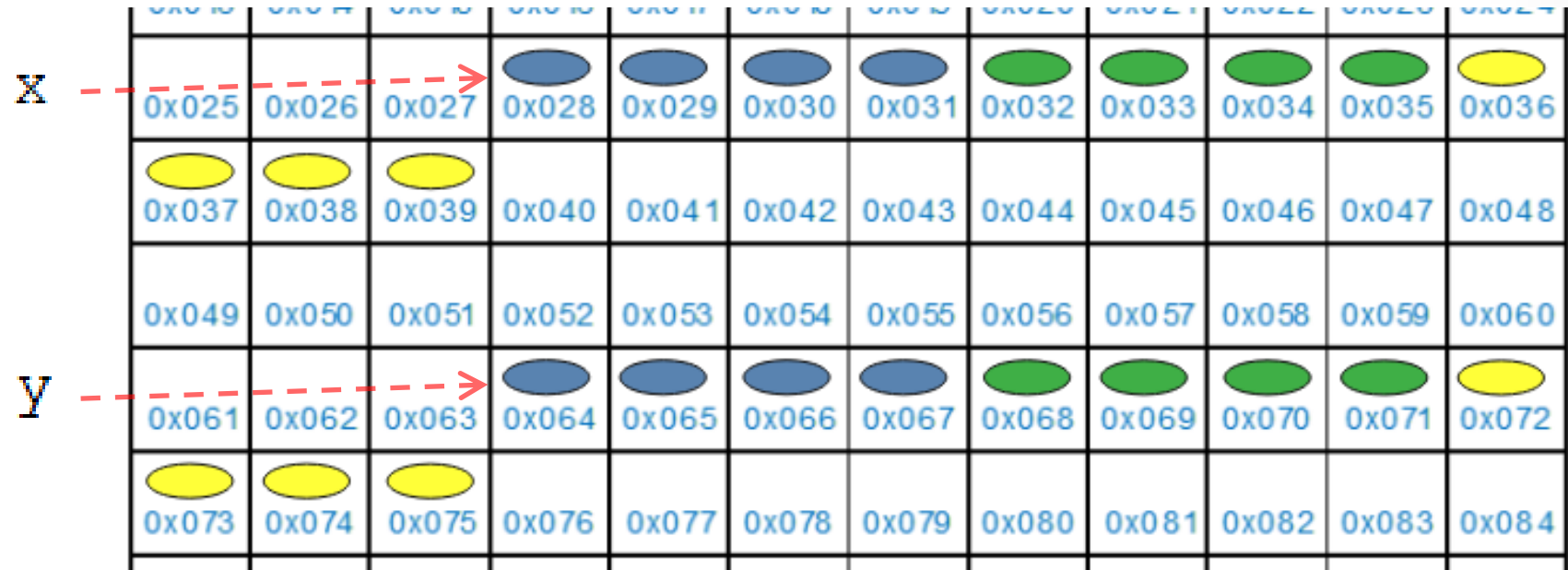
- Se  $p$  é um ponteiro para uma estrutura e  $m$  é um campo dessa estrutura, então o operador ponto “.” permite obter o valor armazenado no campo  $m$ , através de  $(*p).m$ .
- Se  $p$  é um ponteiro para uma estrutura e  $m$  é um campo dessa estrutura, então o operador “->” também permite obter o valor armazenado no campo  $m$  através de  $p->m$ .

# Estrutura de Dados 1



Estruturas – passagem de parâmetros por referência, considerações:

```
struct x{  
    int l;  
    int m;  
    int n;  
};
```



- Se x e y forem duas variáveis com a mesma estrutura, então, para copiar **todos os elementos** ou **campos** de x (0x028) para y (0x064) basta fazer  $y = x$ , isto é, pode-se fazer a atribuição de estruturas, e neste caso, a estrutura é copiada de uma para a outra variável na íntegra.
- Não se pode fazer comparações diretas entre estruturas através dos operadores  $<$ ,  $<=$ ,  $>$ ,  $>=$ ,  $==$  ou  $!=$ . O programador deverá estabelecer qual a relação entre as duas variáveis do tipo estrutura a partir de comparações entre seus campos.



# Estrutura de Dados 1

## Comando `typedef`

- A linguagem C permite que o programador defina os seus próprios tipos com base em outros tipos de dados existentes. Para isso utilizamos o comando `typedef`, sua forma geral é:

```
typedef tipo_existente novo_nome
```

- Onde:
  - **tipo\_existente** – é um tipo básico ou definido pelo programador (**por exemplo, uma struct que você criou...**);
  - **novo\_nome** – é o nome para o novo tipo que estamos definindo.
- O comando `typedef` não cria um novo tipo, ele apenas cria um sinônimo para um tipo já existente. Esse novo nome se torna equivalente ao tipo já existente.





# Estrutura de Dados 1

## Comando `typedef`

- Por exemplo:

```
typedef int banana;  
  
int num = 5;  
banana b = 2;  
  
printf("A soma e: %d", num + b);
```

- As variáveis do tipo `int` e `banana` são usadas de maneira conjunta. Isso é possível porque na realidade elas são do mesmo tipo (`int`). O comando `typedef` apenas disse ao compilador para reconhecer `banana` como outro nome para o tipo `int`.
- O comando `typedef` pode ser usado para simplificar a declaração de um tipo definido pelo programador (`struct`, `union`, etc), ou ainda um ponteiro.



# Estrutura de Dados 1

## Comando `typedef`

- O comando `typedef` pode ser combinado com a declaração de um tipo definido pelo programador em uma única instrução, com a sintaxe:

```
typedef struct nome_do_tipo_do_registro{  
    tipo1 campo1;  
    tipo2 campo2;  
    ...  
    tipon campon;  
}NOME_REGISTRO;
```

Não é obrigatório na linguagem C, porém tornou-se um padrão entre os programadores, a nomeação de novos tipos com todas as letras em maiúsculo.

- Em seguida, novas variáveis deste tipo seriam declaradas neste formato:

```
NOME_REGISTRO reg;
```





## Comando `typedef`

- Tomemos por base a estrutura:

```
struct funcionario{  
    int    id;  
    char   nome[30];  
    int    idade;  
    float  salario;  
};
```

- O comando `typedef` tem como objetivo atribuir nomes alternativos aos tipos já existentes, na maioria das vezes aqueles cujo padrão de declaração é pesado e potencialmente confuso.
- Ele é usado, neste caso, para eliminar a necessidade da palavra-chave `struct` na declaração de variáveis. Assim poderíamos redefinir o nome para a estrutura acima e em seguida declarar uma variável deste novo tipo:

```
typedef struct funcionario FUNCIONARIO;  
FUNCIONARIO func;
```

# Estrutura de Dados 1

## Comando `typedef`

- Dessa forma, utilizando o exemplo anterior, uma estrutura para cadastro de funcionários seria declarada assim:

```
typedef struct funcionario{  
    int    id;  
    char   nome[30];  
    int    idade;  
    float  salario;  
}FUNCIONARIO;
```

- Note que a definição da estrutura está inserida no meio (dentro) do comando `typedef`, formando por tanto uma única instrução. Além disso como estamos associando um novo nome à nossa `struct`, seu nome original pode ser omitido da declaração da `struct`:

```
typedef struct{  
    int    id;  
    char   nome[30];  
    int    idade;  
    float  salario;  
}FUNCIONARIO;
```



# Estrutura de Dados 1

## Estrutura ou Registro



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

```
typedef struct funcionario{
    int    id;
    char   nome[30];
    int    idade;
    float  salario;
}FUNCIONARIO;
```

Toda esta estrutura passa a se chamar FUNCIONARIO

Agora é possível declarar uma variável do tipo FUNCIONARIO

```
int main(){
    FUNCIONARIO func;
    func.id = 123;
    strcpy(func.nome, "Joao");
    func.idade = 21;
    func.salario = 2500;
    printf("Funcionario: %d\nnome: %s\nidade: %d",
           func.id, func.nome, func.idade);
    printf("\n\n\n\n");
    system("pause");
}
```

```
"C:\Users\angelot\Documents\Aulas\GRUEDA1\Aula 04 - Estruturas Heterogêneas - stru
Funcionario: 123
nome: Joao
idade: 21

Pressione qualquer tecla para continuar. . .
```

# Estrutura de Dados 1

## Atividade 2

- Modifique o programa da Atividade 1 para que o registro se torne um “tipo” e dessa forma ser possível a declaração direta do registro, para isso utilize o comando `typedef`;
- Crie duas funções.
  - A primeira para coletar os dados fornecidos via teclado, que não receberá parâmetro algum, mas retornará uma estrutura completa e preenchida, ou seja, retornará um `FUNCIONARIO`;
  - E a segunda, que gerará as impressões na tela. Esta deve receber o vetor[] e a quantidade de elementos desse vetor;
- Crie um procedimento (função `void` que não retorna valor), onde o salário do funcionário sofrerá um reajuste de 10%. Nesta função, **somente o campo salário da estrutura funcionário** deverá ser passado como parâmetro, e terá obrigatoriamente que ser por referência, utilizando-se o endereço do campo (ponteiro) – ver slide 19, funções 3 e 4 como exemplo.
- Gere uma impressão em tela onde será exibido somente o nome do funcionário e o novo valor de salário, a rotina de impressão deverá ficar em outra função chamada `rel_salario_corrigido()`;
- Entregue no Moodle como Atividade 2.

