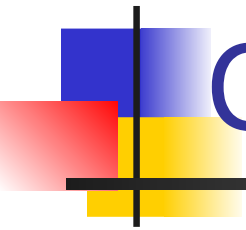
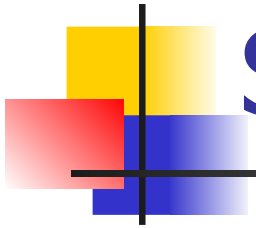


SCC0221 – Introdução à Ciência da Computação I

Prof.: Dr. Rudinei Goularte

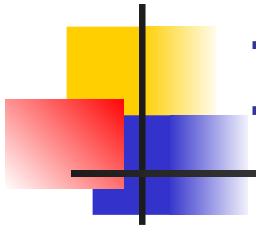
Estruturas (Structs)

Instituto de Ciências Matemáticas e de Computação – ICMC
Sala 4-229



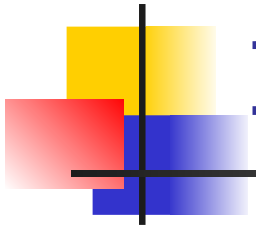
Sumário

- 1. Typedef
- 2. Estruturas
- 3. Estruturas na Heap



1. typedef

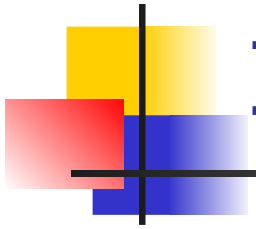
- Permite que se associe explicitamente um tipo a um identificador.
  - Composição de novos tipos de dados a partir de tipos pré-existentes.
  - Não cria um novo tipo de dado.



1. typedef

- Forma geral:
  - **typedef** *tipo identificador*;
  - *tipo*:qualquer tipo válido em C.
  - *identificador*: um identificador válido em C.

- Exemplos:
  - typedef char letra;
  - typedef int polegadas, METROS;

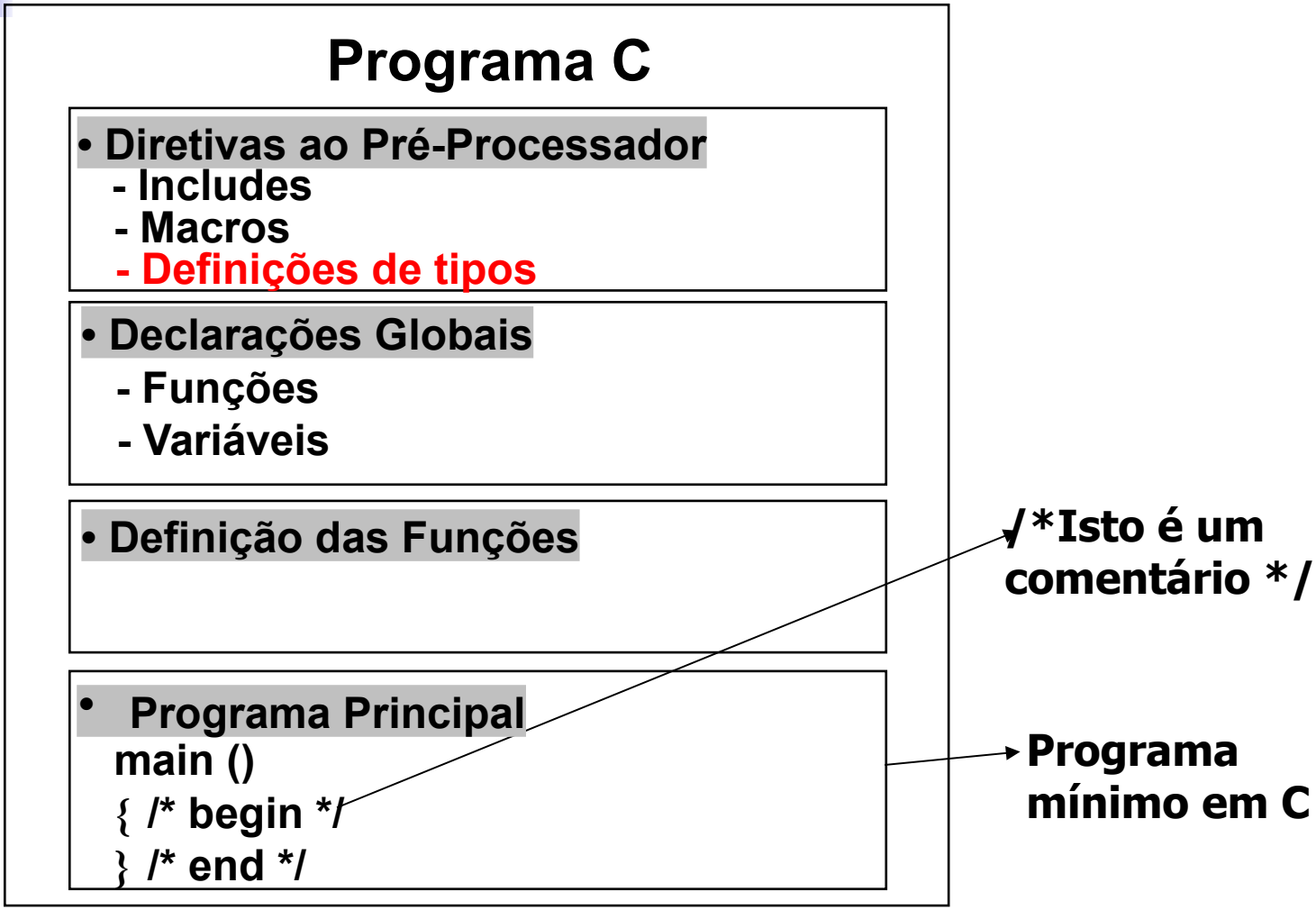


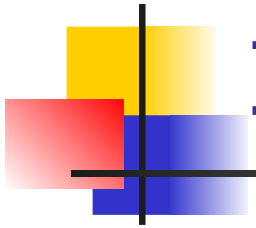
1. typedef

- Os identificadores de cada definição de tipo podem ser usados para declarar variáveis ou funções.

- Exemplos
  - letra u;
  - polegadas altura, largura;

1. typedef

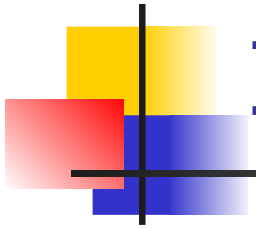

Programa C
• Diretivas ao Pré-Processador
- Includes
- Macros
- Definições de tipos
• Declarações Globais
- Funções
- Variáveis
• Definição das Funções
• Programa Principal
main ()
{ /* begin */
} /* end */
/*Isto é um comentário */
Programa mínimo em C



1. typedef

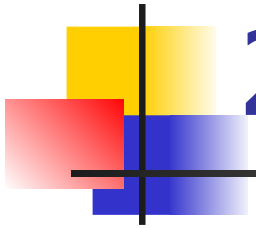
- Exemplos

```
#include <stdio.h>
typedef float num_real;
typedef int medida;
typedef medida altura;
int main (void) {
        altura alt=20;
        int x=4, i;
        i = alt / x;
        return(0);
}
```



1. typedef

- Qual a vantagem de criar uma nova nomenclatura para um tipo existente?
  - Abreviar declarações longas.
  - Possuir nomes de tipos que refletem para que serão utilizados.
  - Facilitar a portabilidade.



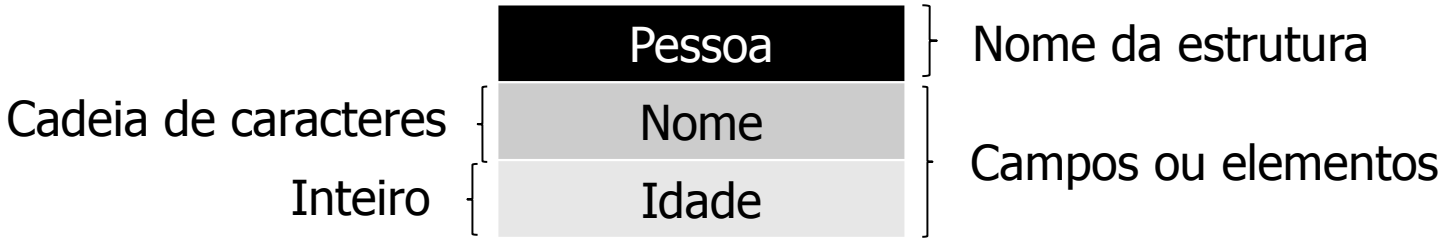
2. Estruturas

- Uma estrutura é uma coleção de variáveis, possivelmente de diferentes tipos, organizadas em um único conjunto.

- As variáveis que compreendem uma estrutura são comumente chamadas de **elementos** ou **campos**.

2. Estruturas

- Cada campo deve ter um nome e deve ser referenciado por ele.
  - Não confundir com vetor: todos os elementos são do mesmo tipo e são referenciados por um índice.



2.1 Declaração

- Definição x Declaração

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

  - Pode ser vista como um modelo.
  - Cria o tipo struct pessoa, mas não aloca espaço para armazenamento.

- Exemplo

```
#include <stdio.h>
struct pessoa {
      char nome[30];
      int idade;
};
int main (void) {
    ...
   return(0);
}
```

2.1 Declaração

- Definição x Declaração

  struct pessoa pai, mae, tio, irmao**;**

  - Declaração de variáveis do tipo **pessoa**.
  - Aloca-se espaço para os identificadores pai, mae, tio e irmao.

- Exemplo

```
#include <stdio.h>
struct pessoa {
        char nome[30];
        int idade;
};
int main (void) {
   struct pessoa pai, mae,
                        tio, irmao;
   ...
   return(0);
}
```

2.1 Declaração

- Definição x Declaração

```
struct pessoa {
        char nome[30];
        int idade;
} pai, mae, tio, irmao;
```

  - Ao mesmo tempo define a estrutura pessoa e declara pai, mae, tio e irmao como variáveis desse tipo.

- Exemplo

```
#include <stdio.h>
int main (void) {
   struct pessoa {
      char nome[30];
      int idade;
   }pai, mae, tio, irmao;
    struct pessoa tia, avo;
   ...
   return(0);
}
```

2.1 Declaração

- Definição x Declaração

```
struct {
        char nome[30];
        int idade;
} pai, mae, tio, irmao;
```

  - Declara as variáveis pai, mae, tio e irmao, mas como a estrutura não possui um nome associado, não se pode declarar mais variáveis desse tipo posteriormente.

2.1 Declaração

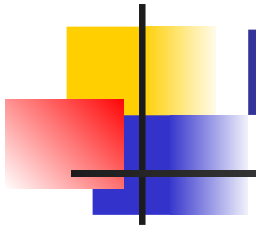
- Usando typedef

```
#include <stdio.h>
struct pessoa {
        char nome[30];
        int idade;
};
typedef struct pessoa familia;

int main (void) {
 familia pai, mae, tio, irmao;
     ...
   return(0);
}
```

2.1 Declaração

- Usando typedef

```
#include <stdio.h>
typedef struct pessoa {
        char nome[30];
        int idade;
}familia;

int main (void) {
 familia pai, mae, tio, irmao;
     ...
   return(0);
}
```



Exercício

- Defina um tipo de dado que contenha o nome, endereço e telefone de uma pessoa. Não utilize typedef. Crie uma variável do tipo de dado especificado.

- Defina um tipo de dado que contenha o nome, endereço e telefone de uma pessoa. Utilize typedef. Crie uma variável do tipo de dado especificado.

2.2 Acesso aos Dados

- Acesso feito via o operador ponto (.).
- Exemplo

```
struct estudante {
        int id;
        char turma;
} alunoicc;
```

```
int main () {
        int num;
        alunoicc.id = 10;
        alunoicc.turma = 'A';
        num = alunoicc.id;
}
```

2.3 Atribuição

- Uma estrutura pode ser atribuída a outra de mesmo tipo.
- Exemplo

```
struct {
        int x;
        char y;
} a, b;
```

```
int main () {
        a.x = 20;
        b = a;
        printf("%d", b.x);
}
```

2.3 Atribuição

- Inicialização

```
struct S {
        int x;
        char y;
};

struct S  b, a = {20,'t'};
```

```
int main () {
        b = a;
        printf("%d %c", b.x, b.y);
}
```

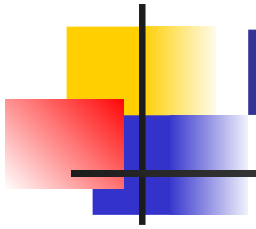
2.4 Operações

- Operações com campos de estruturas devem ser feitas membro a membro.

```
struct S {
        int x;
        char y;
};

struct S  b, a = {20,'t'};
```

```
int main () {
        b.x = a.x + 3;
        b.y = a.y + 3;
}
```

- b = a + 3 = ERRO!



Exercício

- Crie um tipo de dado chamado “fracao” que conterá o numerador e o denominador de uma fração. No seu programa, peça que o usuário digite os valores (numerador e denomidador) de duas frações, e em seguida, retorne a soma dessas duas frações, no formato a/b.

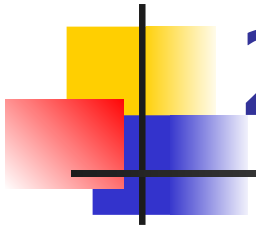
2.5 Composição

- Estruturas podem ser campos de estruturas.

```
struct ponto {
        int x;
        int y;
};

struct linha {
        struct ponto p1;
        struct ponto p2;
} lin;
```

Acesso aos dados:

```
lin.p1.x = 10;
lin.p1.y = 10;
lin.p2.x = 50;
lin.p2.y = 20;
```



2.5 Composição

- Estruturas podem ser autoreferenciadas:

```
typedef struct ponto {
        int x;
        int y;
        struct ponto *p;
};
```

2.6 Vetor de Estruturas

- Exemplo

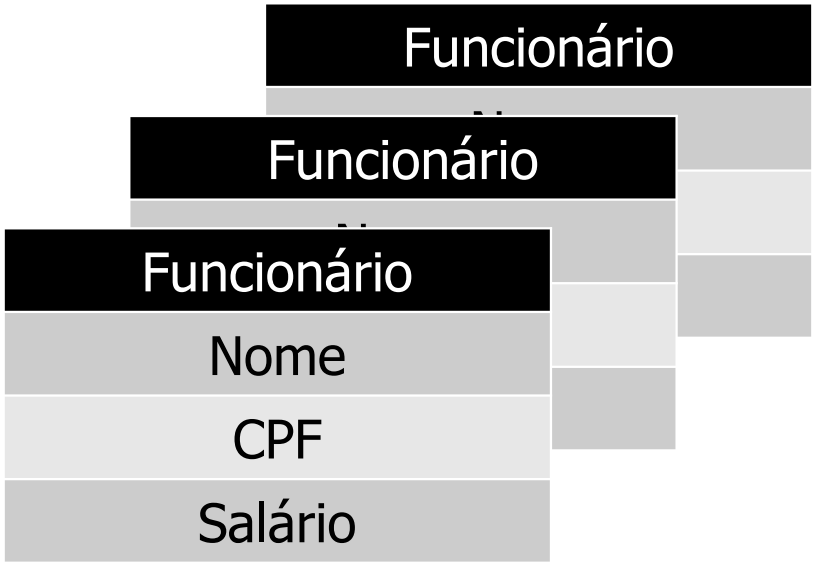
Ficha de funcionário

Funcionário
Nome
CPF
Salário

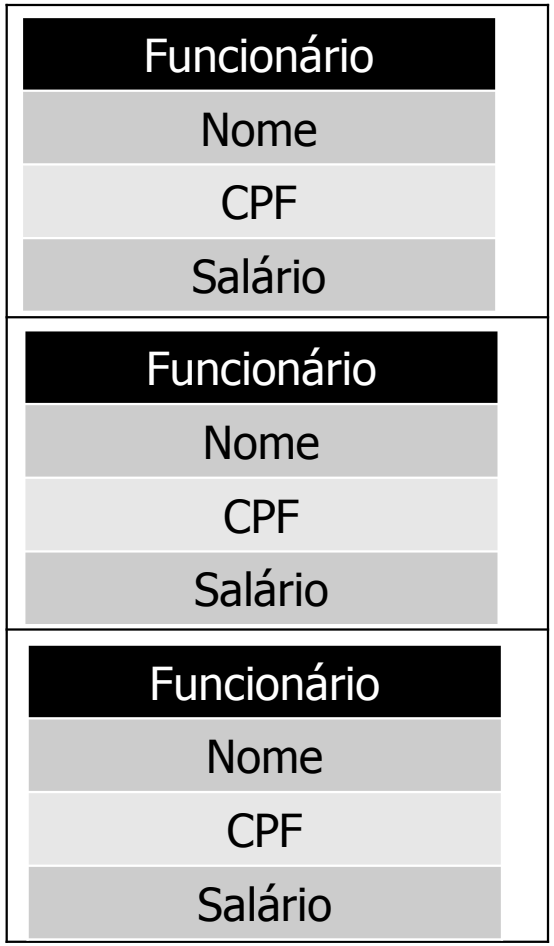
```
struct funcionario {
   char nome[50];
   int CPF;
   float salario;
};
```

2.6 Vetor de Estruturas

- Se, ao invés de uma ficha de funcionário, quisermos cadastrar várias fichas?
  - Vetor de estruturas.

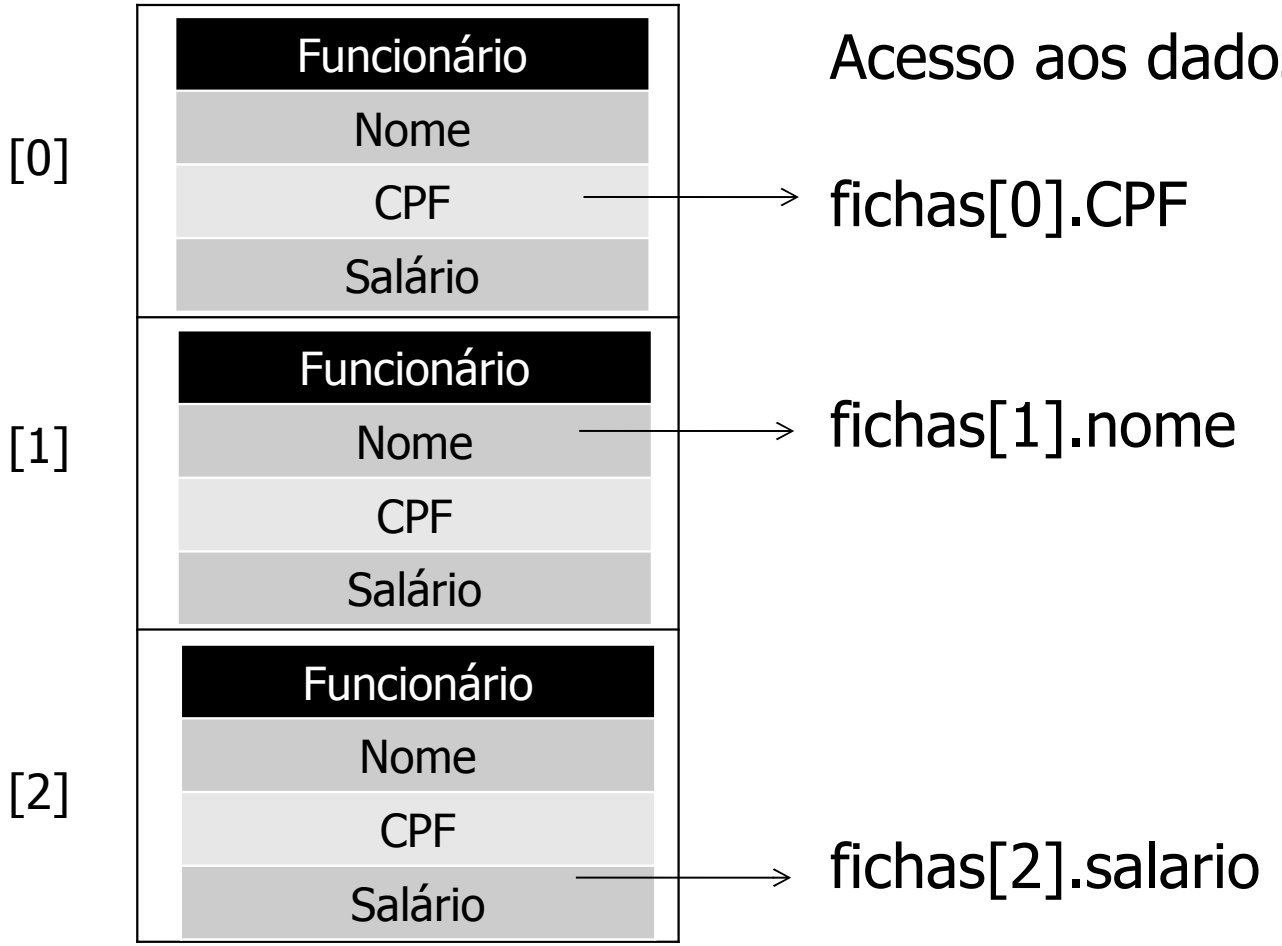


2.6 Vetor de Estruturas




```
struct funcionario {
        char nome[50];
        int CPF;
        float salario;
};

struct funcionario fichas[50];
```

2.6 Vetor de Estruturas

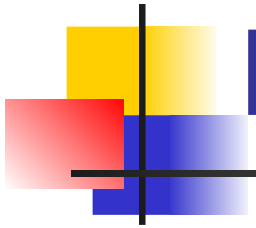


Acesso aos dados:

fichas[0].CPF

fichas[1].nome

fichas[2].salario



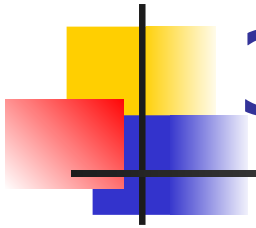
Exercício

- Faça as declarações necessárias para criar um tipo de dado para armazenar as seguintes informações: os registros dos funcionários de uma empresa. A empresa possui 150 empregados. Cada registro possui as seguintes informações: número funcional, nome, sexo, idade e salário.

3. Estruturas na Heap

- Uma struct pode ser alocada dinamicamente na memória heap.

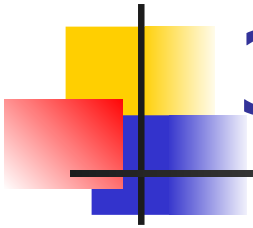
```
struct funcionario {
        char nome[50];
        int CPF;
        float salario;
};
int main (void){
    struct funcionario *p;
    p = (strcut funcionario *) malloc (sizeof(struct funcionario));
    ...
    return(0);
}
```



3. Estruturas na Heap

- Acesso aos campos via operador seta: ->

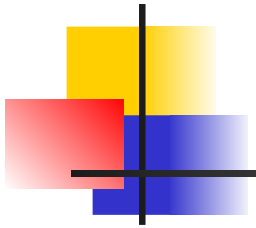
```
struct funcionario {
        char nome[50];
        int CPF;
        float salario;
};
int main (void){
    struct funcionario *p;
    p = (strcut funcionario *) malloc (sizeof(struct funcionario));
    p->salario = 123456.0;
    printf("%f", p->salario);
    return(0);
}
```



3. Estruturas na Heap

Vetor de estruturas na Heap

```
struct funcionario {
        char nome[50];
        int CPF;
        float salario;
};
int main (void){
    struct funcionario *p;
    p = (strcut funcionario *) malloc (sizeof(struct funcionario)*5);
    p[2].salario = 123456.0; /*aqui se usa o operador ponto */
    return(0);
}
```



- Fim.