Universidade de São Paulo – São Carlos
Instituto de Ciências Matemáticas e de Computação

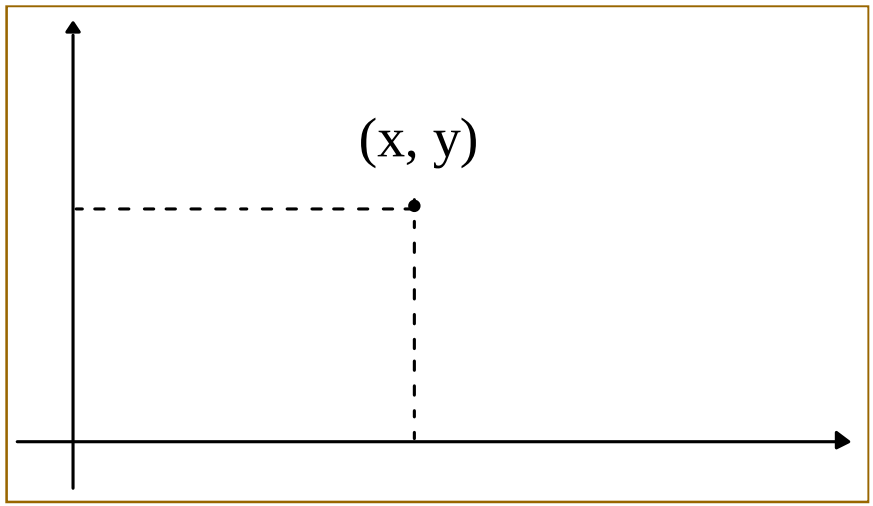
Estruturas

Profa Rosana Braga

Estruturas

- *`struct`* são coleções de dados heterogêneos agrupados em uma mesma estrutura de dados

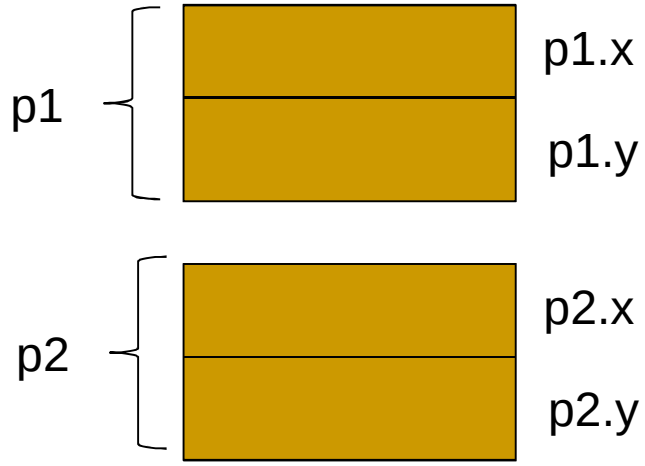
- Ex: armazenar as coordenadas (x,y) de um ponto:



Estruturas

- Declaração:

```
struct {
        int x;
        int y;
      } p1, p2;
```



- a estrutura contém dois inteiros, *x* e *y*
- *p1* e *p2* são duas variáveis tipo *struct* contendo duas coordenadas cada.

Declaração

- Formato da declaração:

```
struct nome_da_estrutura {
                        tipo_1 dado_1;
                        tipo_2 dado_2;
                        ...
                        tipo_n dado_n;
                } lista_de_variaveis;
```

- A estrutura pode agrupar um número arbitrário de dados de tipos diferentes
- Pode-se nomear a estrutura para referenciá-la

Nomeando uma Estrutura

```
struct {
  int x;
  int y;
} p1;

struct {
  int x;
  int y;
} p2;
```

Repetição

struct ponto define um *novo tipo de dado*

```
struct ponto {
        int x;
        int y;
};
struct ponto p1, p2;
```

- Pode-se definir novas variáveis do tipo **ponto**

Estruturas

- acesso aos dados:

struct-var.campo

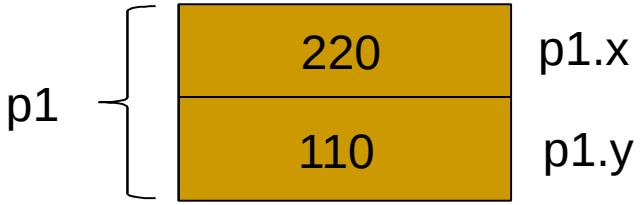
Ex:

```
p1.x = 10;     /*atribuição */
p2.y = 15;
if (p1.x >= p2.x) &&
   (p1.y >= p2.y) ...
```

Atribuição de Estruturas

- Inicialização de uma estrutura:

  struct ponto p1 = { 220, 110 };



- Atribuição entre estruturas *do mesmo tipo*:

```
struct ponto p1 = { 220, 110 };
struct ponto p2;

        p2 = p1;        /* p2.x = p1.x e p2.y = p1.y */
```

- Os campos correspondentes das estruturas são automaticamente copiados da fonte para o destino

Atribuição de Estruturas

- Atenção para estruturas que contenham ponteiros:

```
struct aluno {
        char *nome; int idade;
} a1, a2;

a1.nome = “Alfredo”;
a1.idade = 22;
a2 = a1;
```

Agora a1 e a2 apontam para o mesmo *string* nome:

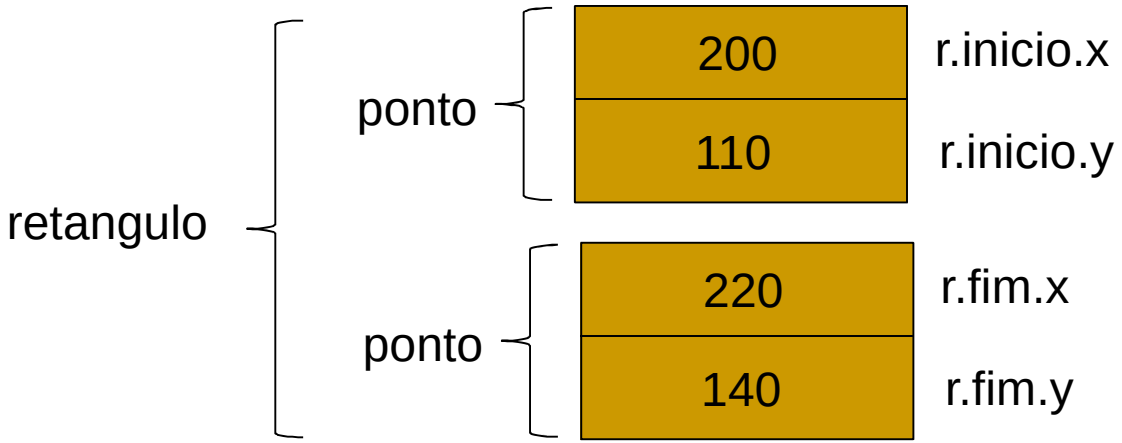
a1.nome == a2.nome == “Alfredo”

Composição de Estruturas

```
struct retangulo {
        struct ponto inicio;
        struct ponto fim;
      };
      struct retangulo r = { { 10, 20 }, { 30 , 40 } };
```

Acesso aos dados:

```
r.inicio.x += 10;
r.inicio.y -= 10;
```



Estruturas como retorno de função

```
struct ponto cria_ponto (int x, int y) {
        struct ponto tmp;

        tmp.x = x;
        tmp.y = y;
        return tmp;
}

main () {
        struct ponto p = cria_ponto(10, 20);
}
```

Operações

- operações entre membros das estruturas devem ser feitas membro a membro:

```
/* retorna uma cópia de p1 = p1 + p2 */
        struct ponto soma_pts (struct ponto p1, struct
ponto p2)
        {
         p1.x += p2.x;
         p1.y += p2.y;

         return p1;   /* retorna uma copia de p1 */
        }
```

Ponteiros para Estruturas

- estruturas grandes são passadas como parâmetro de forma mais eficiente através de ponteiros

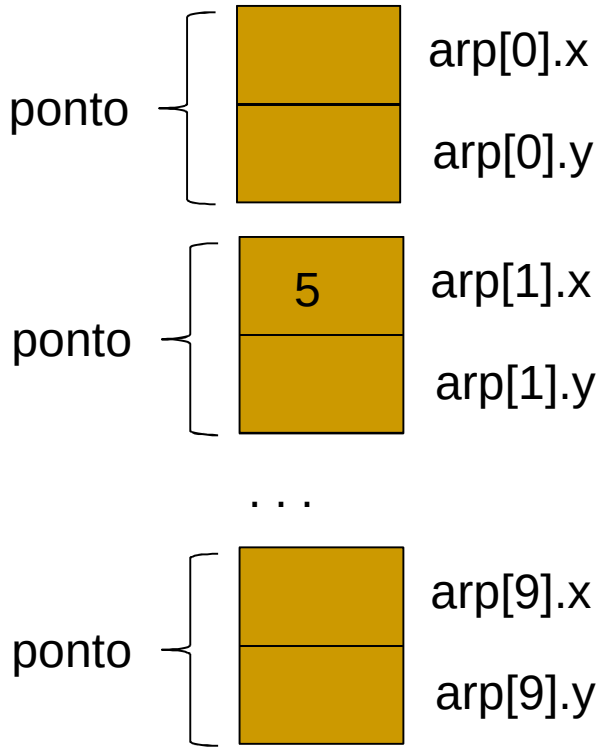
```
struct ponto *pp;
        struct ponto p1 = { 10, 20 };
        pp = &p1;
        printf("Ponto P1: (%d %d)\n", (*pp).x, (*pp).y};
```

- acesso via operador "**->**":

```
printf("Ponto P1: (%d %d)\n", pp->x, pp->y};
```

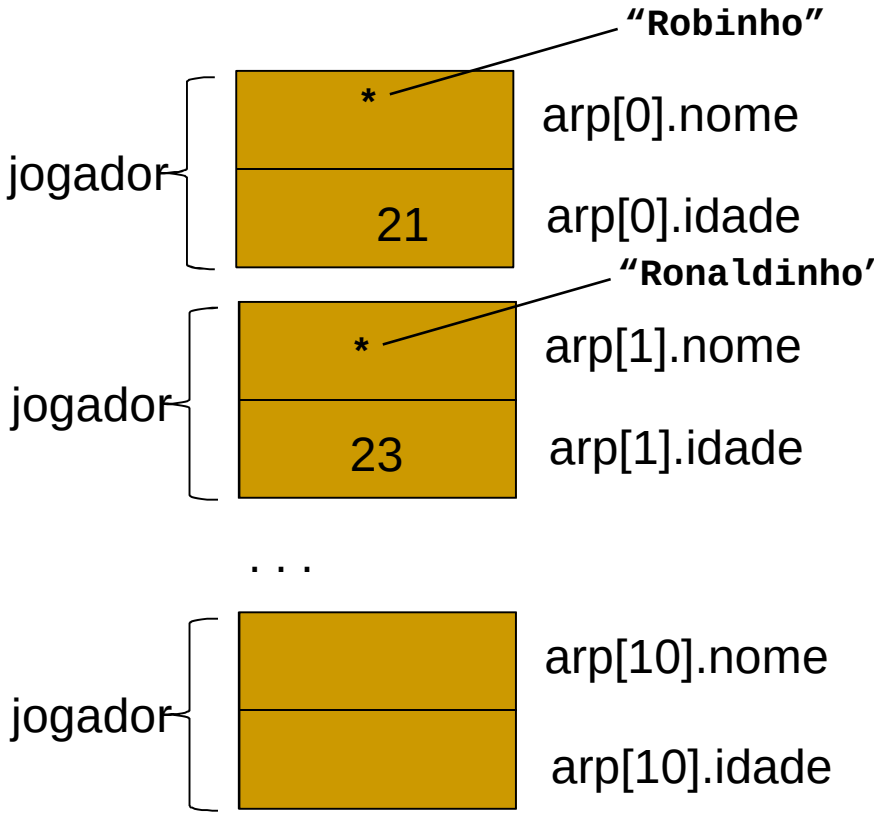
Arrays de Estruturas

```
struct ponto arp[10];
/* cria um array de 10 pontos */
arp[1].x = 5; /*atribui 5 a coordenada x do 2º ponto */
```



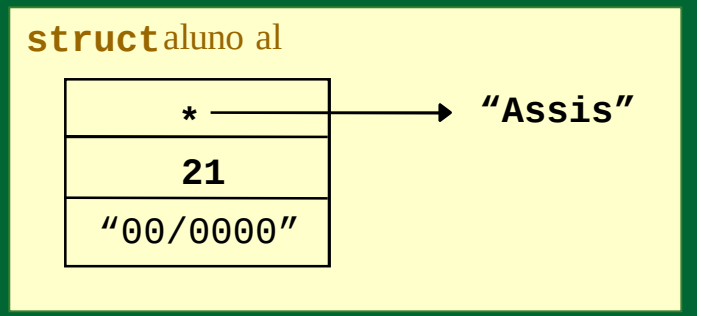
Arrays de Estruturas

```
struct jogador {
    char *nome;
    int idade;
};

struct jogador Brasil[11] = {
            “Robinho”,        21,
            “Ronaldinho”, 23, ...
};
```



Espaço Alocado para uma Estrutura

```
struct aluno {
    char *nome;          /* ponteiro 4 bytes */
    short idade;         /* 2 bytes */
    char matricula[8];   /* array 8 bytes */
};

struct aluno al;
al.nome = "Assis";
al.idade = 21;
strcpy(al.matricula, "00/0001");
```



Função *sizeof(tipo)*

- A função *sizeof(tipo)* retorna o tamanho em bytes ocupado em memória pelo tipo de dado passado como parâmetro

- Ex.:

```
sizeof(int)              => 4 bytes
sizeof(char)             => 1 byte
sizeof(struct ponto)     => 8 bytes
sizeof(struct ponto *)   => 4 bytes
```

Exercício Prático

- Considere um cadastro de produtos de um estoque, com as seguintes informações para cada produto:
  - Código de identificação do produto: representado por um valor inteiro
  - Nome do produto: com ate 50 caracteres
  - Quantidade disponível no estoque: representado por um número inteiro
  - Preço de venda: representado por um valor real

(a) Defina uma estrutura em C, denominada produto, que tenha os campos apropriados para guardar as informações de um produto, conforme descrito acima.

(b) Escreva um programa que leia um inteiro a, uma string b de 50 caracteres, um inteiro c, e um float d e atribua esses valores lidos aos componentes de uma variável p que é do tipo struct produto.

(c) Imprima os valores de p.

Exercício Prático

- Considere o mesmo produto do exercício anterior.
- (a) Declare um vetor de estruturas produto com 5 produtos.

- (b) Escreva um programa principal que, para i de 0 a 4, leia os campos da estrutura nas variáveis a, b, c e d (como no exercício anterior), chame uma função para armazenar essas variaveis na posição i do vetor de estrutura.

- (c) A função recebe os dados de um produto (código, nome, quantidade e preço) e armazena-os em um endereço de um struct produto recebido como parâmetro. Essa função pode ter o seguinte protótipo:

  void gravaProd (int cod, char* nome, int quant, float preco, struct produto *p);

- (d) Imprima o vetor de produtos, sendo um produto por linha, e no fim o total que o dono da loja receberia se vendesse todos os produtos.