



UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS - CSHNB
CURSO DE SISTEMAS DE INFORMAÇÃO
PICOS - PI

TIPO ESTRUTURADO:
STRUCT

Prof. Ma. Luana Batista da Cruz
luana.b.cruz@nca.ufma.br

Roteiro

2

- Tipos primitivos x Tipos estruturados
- Registro/struct
- Atribuição de estruturas
- Operações com estruturas
- Estruturas como retorno de função
- Tipos estruturados mais complexos
- Vetor de estruturas
- Comando typedef
- Ponteiros para estruturas
- Passagem de estruturas para funções
- Tipo union e enum

Tipos primitivos x Tipos estruturados

3

- ❑ C oferece **tipos primitivos** que servem para representar valores simples
 - Reais (float, double), inteiros (int), caracter (char)
- ❑ C oferece também mecanismos para estruturar dados complexos nos quais as informações são compostas por diversos campos

Tipos primitivos x Tipos estruturados

4

- ❑ C oferece **tipos primitivos** que servem para representar valores simples
 - Reais (float, double), inteiros (int), caracter (char)
- ❑ C oferece também mecanismos para estruturar dados complexos nos quais as informações são compostas por diversos campos

TIPOS ESTRUTURADOS!

Tipos estruturados

5

- ❑ Agrupa conjunto de tipos de dados distintos sob um único nome
- ❑ Podemos criar vários objetos na memória de um determinado tipo estruturado
 - **Registros ou struct**

Registro/struct

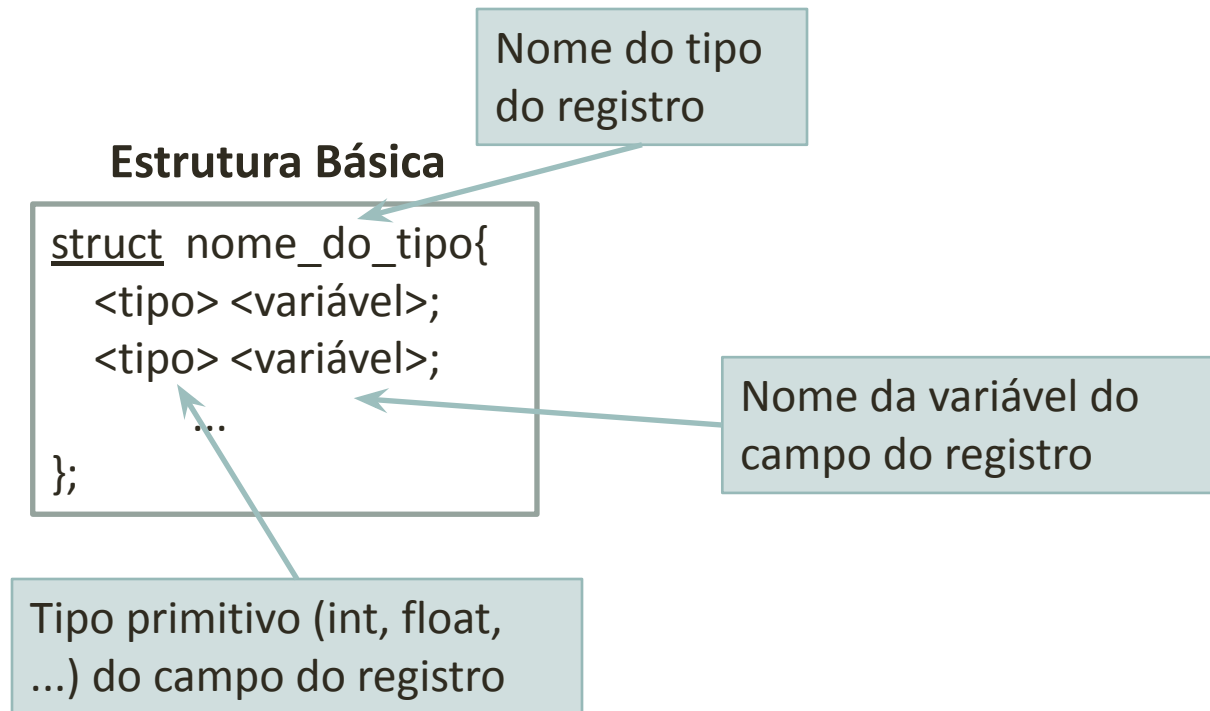
6

- ❑ Um **registro** é uma coleção de várias variáveis, possivelmente de tipos diferentes e logicamente relacionadas
- ❑ **Registros** são coleções de dados heterogêneos agrupados em uma mesma estrutura de dados
- ❑ Os **elementos** de um registro são **chamados de campos**

Declaração de tipos de registros

7

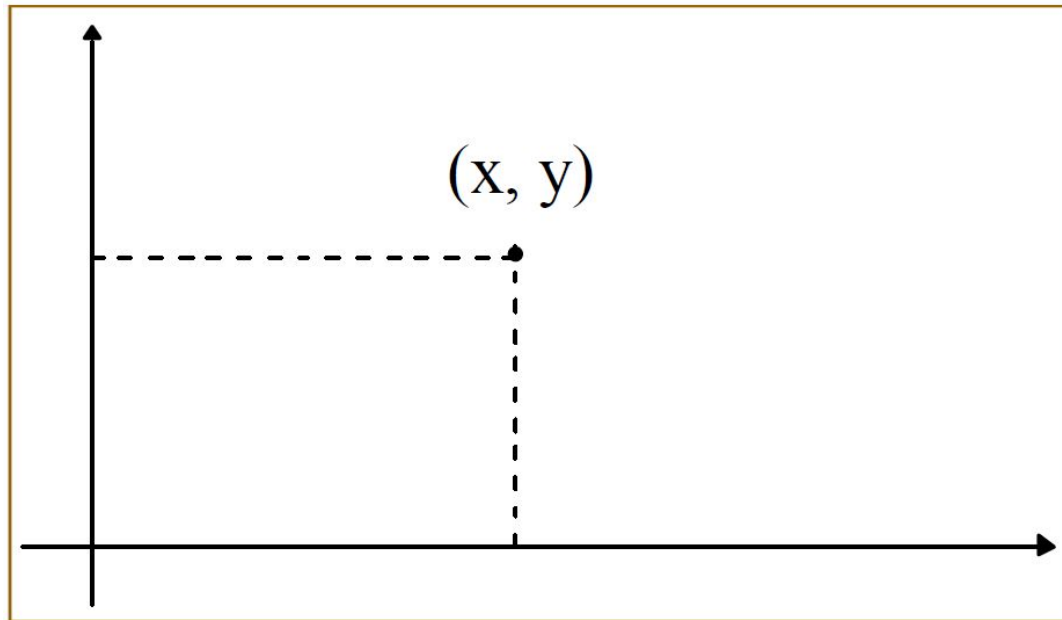
- ❑ A declaração de um registro é a declaração de um tipo personalizado que será utilizado por uma variável



Importância de tipos estruturados

8

- ❑ Considere um ponto representado por duas coordenadas: x e y



Importância de tipos estruturados

9

- ❑ Considere um ponto representado por duas coordenadas: x e y
- ❑ **Sem** mecanismos para agrupar as duas coordenadas:

Importância de tipos estruturados

10

- ❑ Considere um ponto representado por duas coordenadas: x e y
- ❑ **Sem** mecanismos para agrupar as duas coordenadas:

```
int main() {  
    float x;  
    float y;  
    ...  
}
```

Importância de tipos estruturados

11

- ❑ Considere um ponto representado por duas coordenadas: x e y
- ❑ **Sem** mecanismos para agrupar as duas coordenadas:

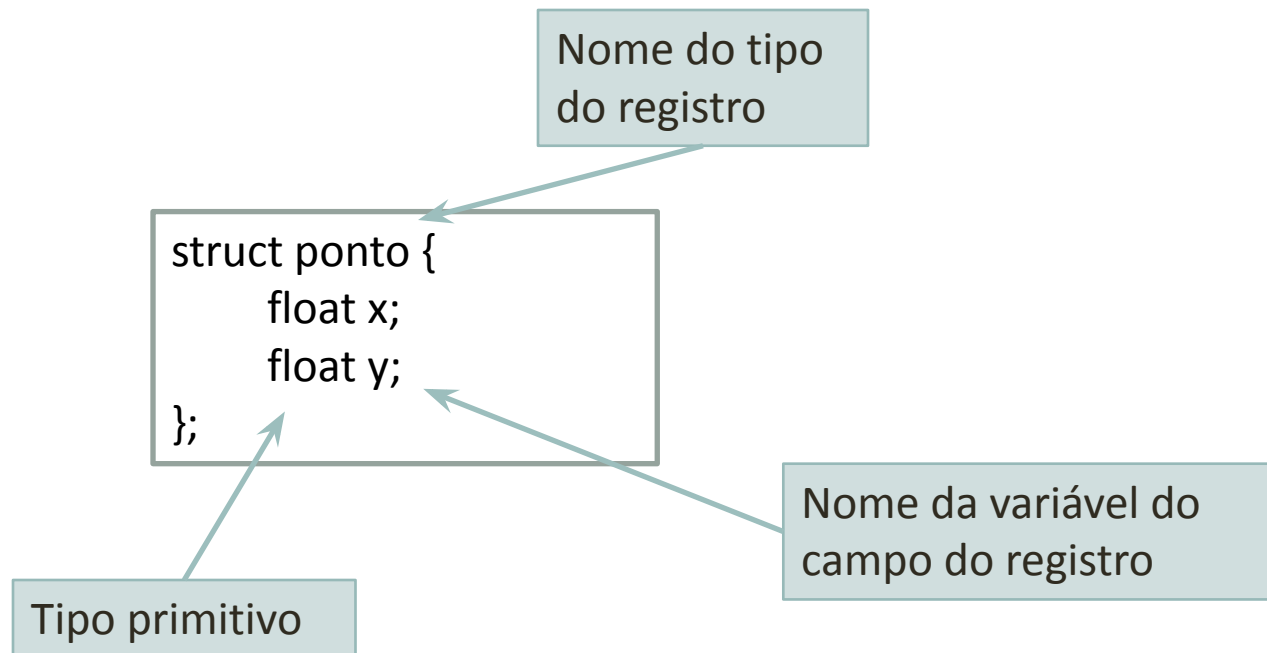
```
int main() {  
    float x;  
    float y;  
    ...  
}
```

Não é possível saber que estas variáveis representam as coordenadas de um ponto

Importância de tipos estruturados

12

- ❑ Uma estrutura em C serve para agrupar diversas variáveis dentro de um único contexto



Declarando variáveis do tipo ponto

13

- ❑ A estrutura ponto passa a ser um tipo estruturado
- ❑ Então, podemos declarar uma variável deste tipo da seguinte forma:

```
struct ponto {  
    float x;  
    float y;  
};  
  
int main() {  
    struct ponto p1;  
    ...  
}
```

A variável é do tipo struct ponto

Declarando variáveis do tipo ponto

14

- ❑ A estrutura ponto passa a ser um tipo estruturado
- ❑ Então, podemos declarar uma variável deste tipo da seguinte forma:

```
struct ponto {  
    float x;  
    float y;  
};  
  
int main() {  
    struct ponto p1;  
    ...  
}
```



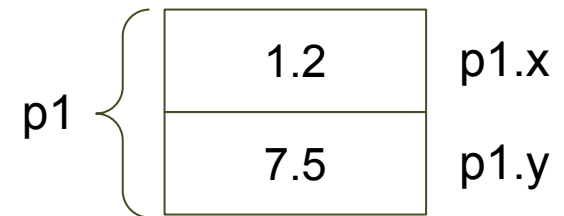
A variável é do tipo struct ponto

Atribuição de estruturas

15

- ❑ Membros de uma estrutura são acessados via o operador de acesso (“.”)

```
struct ponto {  
    float x;  
    float y;  
};  
  
int main() {  
    struct ponto p1;  
    p1.x = 1.2;  
    p1.y = 7.5;  
    ...  
}
```



O nome da variável do tipo struct ponto deve vir antes do “.”

Atribuição de estruturas

16

```
#include <stdio.h>
struct ponto {
    float x;
    float y;
};

int main() {
    struct ponto p1;
    printf("\nDigite as coordenadas do ponto (x,y): ");
    scanf ("%f %f", &p1.x, &p1.y);
    printf("O ponto fornecido foi: (%f,%f)\n", p.x, p.y);
    return 0;
}
```


Atribuição de estruturas

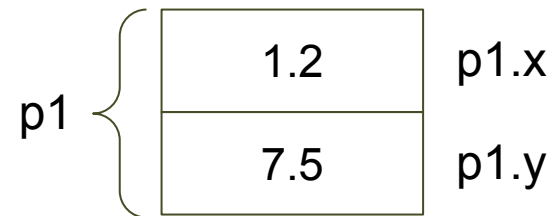
17

❏ Inicialização de uma estrutura:

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

```
int main() {  
    struct ponto p1 = {1.2, 7.5};  
    ...  
}
```

Deve-se inicializar os membros na ordem correta



Atribuição de estruturas

18

- ❑ A inicialização de uma estrutura **deve ser feita no ato de sua declaração**

```
struct ponto {  
    float x;  
    float y;  
};  
  
int main() {  
    struct ponto p1;  
    p1 = {1.2, 7.5};  
    ...  
}
```

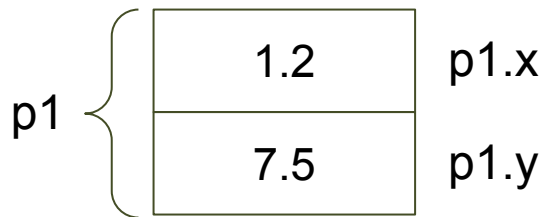
Atribuição de uma
estrutura para a
variável p1
(Errado)

Atribuição de estruturas

19

❏ Atribuição entre **estruturas do mesmo tipo**:

```
struct ponto p1 = {1.2, 7.5};  
struct ponto p2;  
//p2 = p1;
```



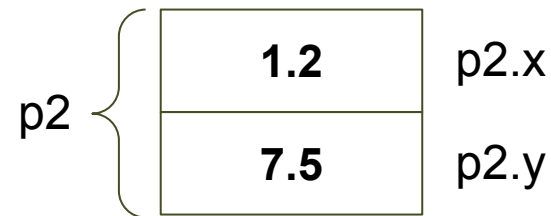
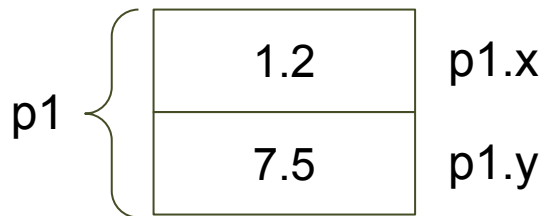
Atribuição de estruturas

20

❏ Atribuição entre **estruturas do mesmo tipo**:

```
struct ponto p1 = {1.2, 7.5};  
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



Outras operações com estruturas

21

- ❑ Como escrever um programa que imprime a **soma das coordenadas de dois pontos?**

```
#include <stdio.h>
struct ponto {
    float x;
    float y;
};

int main() {
    struct ponto p3;
    struct ponto p1 = {0.0, 4.5};
    struct ponto p2 = {1.0, 2.5};
    p3 = p1 + p2;
    printf("O x e y do novo ponto eh: (%f, %f)", p3.x, p3.y);
    return 0;
}
```

Outras operações com estruturas

22

- ❑ Como escrever um programa que imprime a **soma das coordenadas de dois pontos?**

```
#include <stdio.h>
struct ponto {
    float x;
    float y;
};

int main() {
    struct ponto p3;
    struct ponto p1 = {0.0, 4.5};
    struct ponto p2 = {1.0, 2.5};
    p3 = p1 + p2;
    printf("O x e y do novo ponto eh: (%f, %f)", p3.x, p3.y);
    return 0;
}
```

Errado! Não podemos somar estruturas inteiras

Outras operações com estruturas

23

- ❑ Como escrever um programa que imprime a **soma das coordenadas de dois pontos?**

```
#include <stdio.h>
struct ponto {
    float x;
    float y;
};

int main() {
    struct ponto p3;
    struct ponto p1 = {0.0, 4.5};
    struct ponto p2 = {1.0, 2.5};
    p3.x = p1.x + p2.x;
    p3.y = p1.y + p2.y;
    printf("O x e y do novo ponto eh: (%f, %f)", p3.x, p3.y);
    return 0;
}
```

Certo! Temos que somar campo a campo das variáveis do tipo struct ponto

Estruturas como retorno de função

24

- ❑ Função para criar um ponto e retornar a struct ponto pela função

```
struct ponto cria_ponto(float x, float y) {  
    struct ponto tmp;  
    tmp.x = x;  
    tmp.y = y;  
    return tmp;  
}  
  
int main () {  
    struct ponto p = cria_ponto(10.0, 20.0);  
    ...  
}
```


Estruturas como retorno de função

25

- ❑ Função para somar as coordenadas de dois pontos

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

int main () {
    struct ponto p1 = cria_ponto(10.0, 20.0);
    struct ponto p2 = cria_ponto(30.0, 70.0);
    struct ponto p3 = soma_pts(p1, p2);
    printf("A soma dos pontos de p1 e p2: (%f, %f)", p3.x,p3.y);
    ...
}
```

Estruturas como retorno de função

26

- ❑ Função para somar as coordenadas de dois pontos

```
struct ponto altera_ponto(float x, float y){
    struct ponto q;
    q.x = x;
    q.y = y;
    return q;
}

int main () {
    struct ponto p1 = cria_ponto(10.0, 20.0);
    p1 = altera_ponto(3.9, 1.9);
    printf("As novas coordenadas de p1: (%f, %f)", p1.x,p1.y);
    ...
}
```

Tipos estruturados mais complexos

27

- Podemos ter também um registro dentro de outro registro. Isso é útil para modularizar melhor alguns campos dentro de um registro

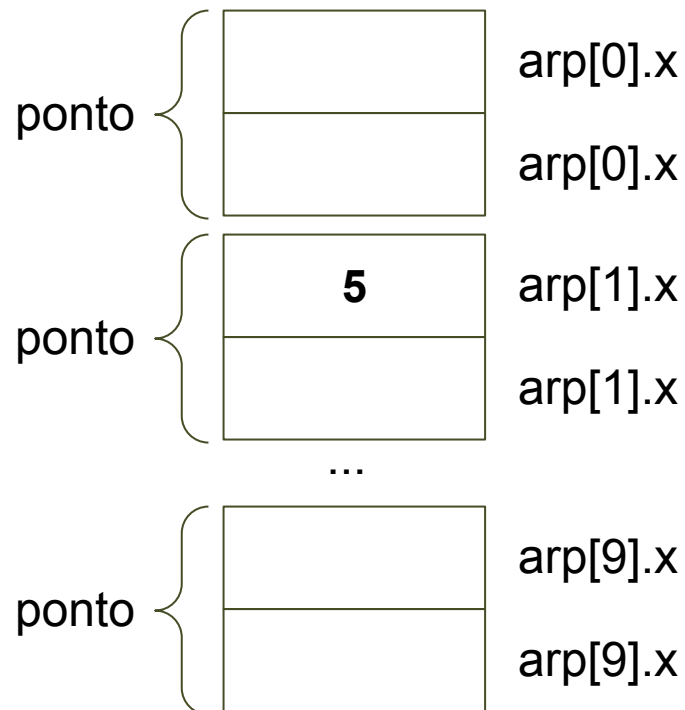
```
struct ponto {  
    float x;  
    float y;  
};  
  
struct reta {  
    struct ponto primeiro_ponto;  
    struct ponto segundo_ponto;  
};  
  
int main () {  
    struct ponto p1 = cria_ponto(3.0, 3.0);  
    struct ponto p2 = cria_ponto(2.0, 6.0);  
    struct reta r1 = {p1, p2};  
    ...  
}
```

Vetor de estruturas

28

❏ Criar 10 pontos usando vetor

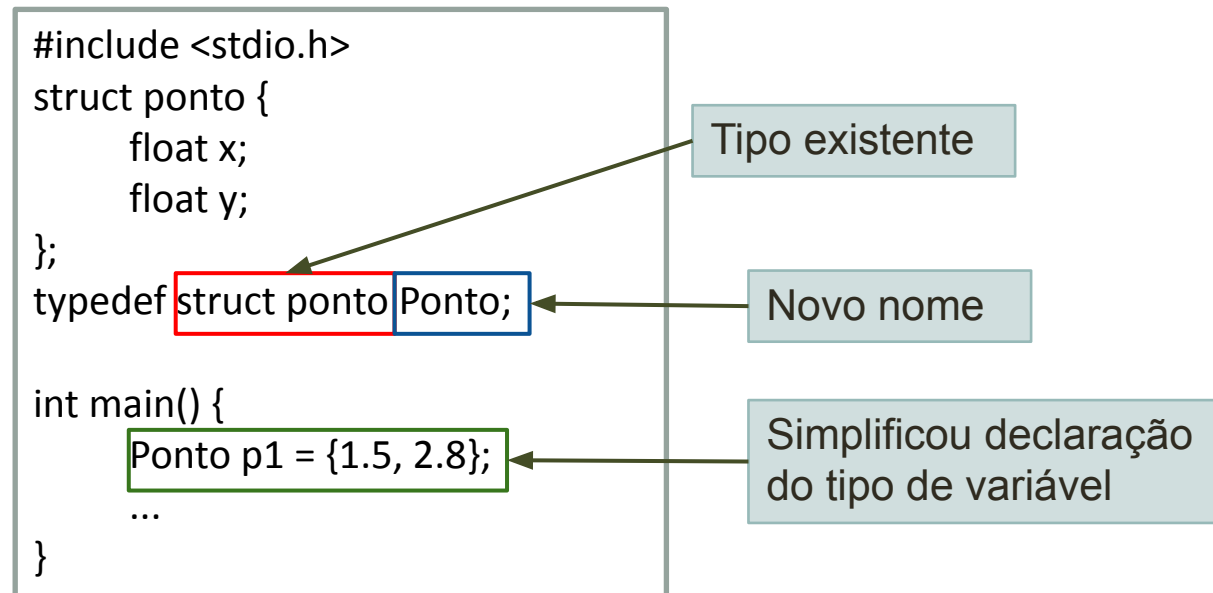
```
#define TAM 10  
struct ponto arp[TAM]; /* cria um array de 10 pontos */  
arp[1].x = 5; /* atribui 5 a coordenada x do 2º ponto */
```



Definição de “novos” tipos

29

- ❑ A linguagem C permite criar nomes de tipos:
 - **Estrutura:** typedef tipo_existente sinonimo;
 - **Ex:** typedef float Real
- ❑ Em geral, definimos nomes de tipos para as estruturas com as quais nosso programa trabalham. Por exemplo:



Ponteiros para estruturas

30

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

```
struct ponto *pp;
```

- ❑ Para acessar os campos

```
(*pp).x = 12.0;
```

- ❑ De maneira simplificada

```
pp->x = 12.0;
```

- ❑ Para acessar o endereço de um campo

```
&pp->x
```

Ponteiros para estruturas

31

❏ Exemplo

```
struct ponto *pp;  
struct ponto p1 = {10.0, 20.0};  
pp = &p1;  
printf("Ponto P1: (%f %f)\n", (*pp).x, (*pp).y);  
printf("Ponto P1: (%f %f)\n", pp->x, pp->y);
```

Passagem de estruturas para funções

32

❑ Passagem por **valor**:

- Análoga à passagem de variáveis simples
- Função recebe toda a estrutura como parâmetro
- Função acessa a cópia da estrutura na pilha
- Função não altera os valores dos campos da estrutura original
- Operação pode ser custosa se a estrutura for muito grande

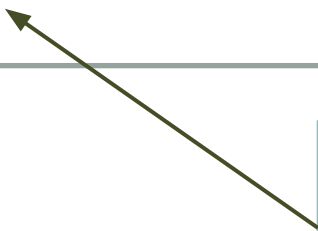
```
/* função que imprime as coordenadas do ponto */  
void imprime (struct ponto p){  
    printf("O ponto fornecido foi: (%.2f, %.2f)\n", p.x, p.y);  
}  
  
int main(){  
    struct ponto p1 = {10.0, 20.0};  
    imprime(p1); // a estrutura inteira é copiada para pilha  
}
```


Passagem de estruturas para funções

33

- Passagem por **valor**: a estrutura inteira é copiada para pilha

```
void imprime (struct ponto p){  
    printf("O ponto fornecido foi: (%.2f, %.2f)\n", p.x, p.y);  
}  
  
int main(){  
    struct ponto *pp;  
    struct ponto p1 = {10.0, 20.0};  
    pp = &p1;  
    imprime(*pp);  
}
```



Passamos por parâmetro
o conteúdo do endereço
(estrutura inteira) que pp
aponta

Passagem de estruturas para funções

34

- ❑ Passagem por **referência**:
 - Apenas o ponteiro da estrutura é passado para pilha, mesmo que não seja necessário alterar os valores dos campos dentro da função

Passagem de estruturas para funções

35

- ❑ Passagem por **referência**: exemplo para ler coordenadas do ponto

```
void imprime (struct ponto* pp){
    printf("O ponto fornecido foi: (%.2f, %.2f)\n", pp->x, pp->y);
}

void captura (struct ponto* pp){
    printf("Digite as coordenadas do ponto (x, y): ");
    scanf("%f %f", &pp->x, &pp->y);
}

int main(){
    struct ponto p;
    captura (&p);
    imprime (&p);
    return 0;
}
```

Tipo União

36

□ Union

- Localização de memória compartilhada por diferentes variáveis, que podem ser de tipos diferentes
- Uniões usadas para armazenar valores heterogêneos em um mesmo espaço de memória

```
union exemplo{  
    int i;  
    char c;  
};  
union exemplo v;
```

- Campos i e c compartilham o mesmo espaço de memória
- Variável v ocupa pelo menos o espaço necessário para armazenar o maior de seus campos (um inteiro, no caso)

Tipo União

37

❏ Armazenamento:

- Apenas um único elemento de uma união pode estar armazenado num determinado instante
- A atribuição a um campo da união sobrescreve o valor anteriormente atribuído a qualquer outro campo

```
union exemplo{  
    int i;  
    char c;  
};  
union exemplo v;  
  
v.i = 10;  /* alternativa 1 */  
v.c = 'x'  /* alternativa 2 */
```

Tipo União

38

❏ Armazenamento:

- Apenas um único elemento de uma união pode estar armazenado num determinado instante
- A atribuição a um campo da união sobrescreve o valor anteriormente atribuído a qualquer outro campo

```
union exemplo{  
    int i;  
    char c;  
};  
union exemplo v;  
  
v.i = 10;  /* alternativa 1 */  
v.c = 'x'  /* alternativa 2 */
```

Então o que vai ser
impresso?

Tipo União

39

- ❑ Quando devo utilizar?
 - Em contextos de memória muito limitada
 - Quando tiver muitas variáveis dentro do programa mas não são utilizadas ao mesmo tempo

Tipo Enumeração

40

□ Enum

- Declara uma enumeração, ou seja, um conjunto de constantes inteiras com nomes que especifica os valores legais que uma variável daquele tipo pode ter
- Oferece uma forma mais elegante de organizar valores constantes

Tipo Enumeração

41

❏ Exemplo – tipo Booleano:

```
enum bool {  
    TRUE = 1,  
    FALSE = 0  
};  
  
typedef enum bool Bool;  
  
Bool resultado;
```

bool	Declara as constantes FALSE e TRUE associa TRUE ao valor 1 e FALSE ao valor 0
Bool	Declara um tipo cujos valores só podem ser TRUE (1) ou FALSE (0)
resultado	Variável que pode receber apenas os valores TRUE ou FALSE

Tipo Enumeração

42

- ❑ Pra que devo utilizar?
 - Tornam o código mais explícito, mais legível, e menos vulnerável a erros de programação

Atividade

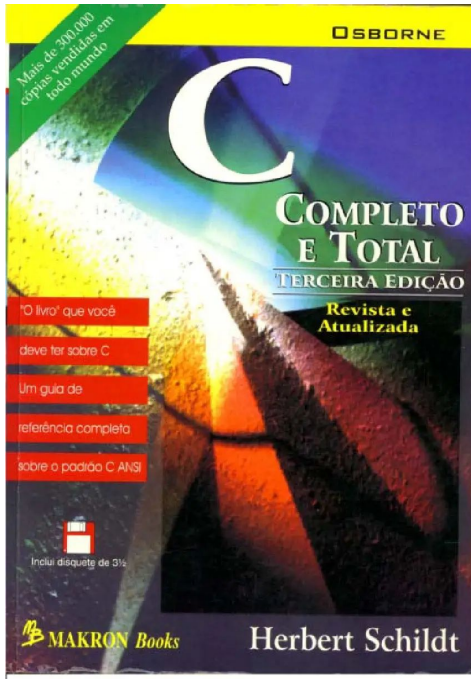
43

- ❑ **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 até 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

Referências



44



SCHILDT, Herbert. **C completo e total**. Makron, 3ª edição revista e atualizada, 1997.



SZWARCHFITER, J. **Estruturas de Dados e seus algoritmos**. 3 ed. Rio de Janeiro: LTC, 2010.