
Fundamentos de Programação

Aula 12 - Tipos Estruturados (Structs)

Arnaldo Barreto Vila Nova

Sumário

- Conceito
 - Declaração
 - Definição de TIPO
 - Exercícios
 - Ponteiros para estruturas
 - Estruturas como parâmetros
-

Conceito - Tipos Estruturados

- Exemplo: Os dados de cadastro de uma pessoa em determinado sistema são: Nome, Idade, Altura e Peso.
-

Conceito - Tipos Estruturados

- Exemplo: Os dados de cadastro de uma pessoa em determinado sistema são: Nome, Idade, Altura e Peso.

```
char nome[50];  
int idade;  
float altura, peso;
```

Até o momento, colocaríamos
as variáveis para este caso
dessa forma

Conceito - Tipos Estruturados

- Exemplo: Os dados de cadastro de uma pessoa em determinado sistema são: Nome, Idade, Altura e Peso.

```
char nome[50];  
int idade;  
float altura, peso;
```

- Para 10 pessoas

```
char nome[10][50];  
int idade[10];  
float altura[10], peso[10];
```

Ou algo assim para
várias pessoas

716	peso[9]
	...
680	peso[0]
676	altura[9]
	...
640	altura[0]
636	idade[9]
	...
600	idade[0]
599	nome[9]
	...
100	nome[0]

Conceito - Tipos Estruturados

- Mas dessa forma, os dados de uma pessoa estão espalhados na memória
 - matriz nome está no endereço 100
 - vetor idade está no endereço 600
 - vetor altura está no endereço 640
 - vetor peso está no endereço 680
- O programador precisa ter mais atenção na manipulação, correndo o risco de misturar dados de pessoas diferentes
- Além disso, para passar os dados de uma pessoa para uma função, cada dado tem que ser passado separadamente

716	peso[9]
	...
680	peso[0]
676	altura[9]
	...
640	altura[0]
636	idade[9]
	...
600	idade[0]
599	nome[9]
	...
100	nome[0]

Conceito - Tipos Estruturados

- Além disso, para passar os dados de uma pessoa para uma função, cada dado tem que ser passado separadamente
 - No nosso exemplo inicial, uma função para alterar os dados de uma pessoa, precisaria de pelo menos 4 parâmetros.
 - Se um determinado cadastro de uma pessoa ou objeto precisar de 30 informações diferentes, fica inviável fazer uma função com 30 parâmetros.
-

Conceito - Tipos Estruturados

658	pessoa[9]
	...
224	pessoa[2]
162	pessoa[1]
100	pessoa[0]

- E se os dados de cada pessoa estivessem agrupados?
 - Cada pessoa ocupando um espaço de 62 bytes na memória
 - char nome[50] → 50 bytes
 - int idade → 4 bytes
 - float altura → 4 bytes
 - float peso → 4 bytes
-

658	pessoa[9]
	...
224	pessoa[2]
162	pessoa[1]
100	pessoa[0]

Conceito - Tipos Estruturados

- E se os dados de cada pessoa estivessem agrupados?
 - Cada pessoa ocupando um espaço de 62 bytes na memória
 - char nome[50] → 50 bytes
 - int idade → 4 bytes
 - float altura → 4 bytes
 - float peso → 4 bytes
 - Dessa forma, poderemos:
 - trabalhar com ponteiros para as pessoas
 - passar as informações agrupadas para funções
-

Declaração - Struct

- Os Tipos Estruturados (ou structs) servem para agrupar diversos dados em uma única estrutura
 - Ou seja, vamos criar um tipo de variável capaz de armazenar diversos valores e informações diferentes
 - Essa é a base para o paradigma de Programação Orientada a Objetos, que terá sua própria disciplina mais à frente
-

Declaração - Struct

- Exemplo: fazer um programa para ler e imprimir dois pontos no plano (x, y)

Cada ponto precisa de 2 valores float:
a coordenada x, e a coordenada y

Declaração - Struct

- Exemplo: fazer um programa para ler e imprimir dois pontos no plano (x, y)

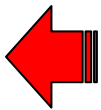
```
void main()
{
    float x1, x2, y1, y2;
    printf("ponto 1 (x y): ");
    scanf("%f %f", &x1, &y1);
    printf("ponto 2 (x y): ");
    scanf("%f %f", &x2, &y2);
    printf("ponto 1: (%.2f, %.2f)", x1, y1);
    printf("ponto 2: (%.2f, %.2f)", x2, y2);
}
```

Sem struct poderíamos
fazer algo assim: x1 e y1
para o primeiro ponto, x2 e
y2 para o segundo ponto
(ou com vetores)

Declaração - Struct

- Exemplo: fazer um programa para ler e imprimir dois pontos no plano (x, y)

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



Usando struct, primeiro definimos
como será nossa estrutura.

Neste caso, nossa estrutura chamada
ponto será formada por 2 float

Declaração - Struct

- Exemplo: fazer um programa para ler e imprimir dois pontos no plano (x, y)

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

```
void main()  
{
```



```
    struct ponto p1, p2;  
    printf("ponto 1 (x y): ");  
    scanf("%f %f", &p1.x, &p1.y);  
    printf("ponto 2 (x y): ");  
    scanf("%f %f", &p2.x, &p2.y);  
    printf("ponto 1: (%.2f, %.2f)", p1.x, p1.y);  
    printf("ponto 2: (%.2f, %.2f)", p2.x, p2.y);  
}
```

Com isso criamos o tipo
chamado **struct ponto**, e
podemos criar variáveis
deste tipo

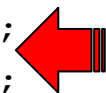
Declaração - Struct

- Exemplo: fazer um programa para ler e imprimir dois pontos no plano (x, y)

```
void main()
{
    struct ponto {
        float x;
        float y;
    };

    struct ponto p1, p2;
    printf("ponto 1 (x y): ");
    scanf("%f %f", &p1.x, &p1.y);
    printf("ponto 2 (x y): ");
    scanf("%f %f", &p2.x, &p2.y);
    printf("ponto 1: (%.2f, %.2f)", p1.x, p1.y);
    printf("ponto 2: (%.2f, %.2f)", p2.x, p2.y);
}
```

Daí podemos utilizar cada campo da nossa estrutura com um ponto (.) entre o nome da variável e do campo da estrutura



Declaração - Struct

- Definição do struct

Vamos ver a definição de um
struct mais de perto

```
struct NOME_DA_ESTRUTURA {  
    TIPO CAMPO1;  
    TIPO CAMPO2;  
    ...  
};
```

Declaração - Struct

- Definição do struct



Primeiro definimos o nome
da o struct, seguindo as
mesmas regras de
identificação de variáveis


```
struct NOME_DA_ESTRUTURA {  
    TIPO CAMPO1;  
    TIPO CAMPO2;  
    ...  
};
```

Declaração - Struct

- Definição do struct

Entre as chaves, vamos
colocar todos os atributos
ou campos que aquela
estrutura terá

```
struct NOME_DA_ESTRUTURA {  
    TIPO CAMPO1;  
    TIPO CAMPO2;  
    ...  
};
```




Declaração - Struct

- Definição do struct

Note que podemos ter aqui
tipos diferentes, vetores,
matrizes, ponteiros... até
mesmo outros tipos
estruturados

```
struct NOME_DA_ESTRUTURA {  
    TIPO CAMPO1;  
    TIPO CAMPO2;  
    ...  
};
```



Declaração - Struct

- Definição do struct

```
struct NOME_DA_ESTRUTURA {  
    TIPO CAMPO1;  
    TIPO CAMPO2;  
    ...  
};
```



E lembre-se de utilizar um
ponto-e-vírgula depois das
chaves da estrutura.

Declaração - Struct

- Declaração de uma variável de estrutura

```
struct NOME_DA_ESTRUTURA {  
    TIPO CAMPO1;  
    TIPO CAMPO2;  
    ...  
};
```



```
void main()  
{  
    struct NOME_DA_ESTRUTURA VAR;  
}
```

Normalmente definimos as
estruturas de forma global,
ou seja, antes da main()

Declaração - Struct

- Declaração de uma variável de estrutura

```
struct NOME_DA_ESTRUTURA {  
    TIPO CAMPO1;  
    TIPO CAMPO2;  
    ...  
};
```

```
void main()  
{
```

```
    struct NOME_DA_ESTRUTURA VAR;
```



E quando vamos utilizar uma variável dessa nossa estrutura, colocamos **struct NOME** como tipo da nossa variável

Declaração - Struct

- Declaração de uma variável de estrutura

```
struct NOME_DA_ESTRUTURA {  
    TIPO CAMPO1;  
    TIPO CAMPO2;  
    ...  
};
```

```
void main()  
{
```

```
    struct NOME_DA_ESTRUTURA VAR;
```



Mais adiante, neste slide, será mostrado como utilizar o **typedef** para definir melhor o nome de suas estruturas

E quando vamos utilizar uma variável dessa nossa estrutura, colocamos **struct NOME** como tipo da nossa variável

Declaração - Struct

- Manipular valores da estrutura

```
struct NOME_DA_ESTRUTURA {  
    TIPO CAMPO1;  
    TIPO CAMPO2;  
    ...  
};
```

```
void main()  
{  
    struct NOME_DA_ESTRUTURA VAR;  
    VAR.CAMPO1 = VALOR1;  
    VAR.CAMPO2 = VALOR2;  
}
```



Podemos manipular ou acessar cada campo da nossa variável de tipo estruturado utilizando um ponto (.) entre o nome da variável e do campo desejado

Declaração - Struct

- Exemplo: cadastro de uma pessoa

```
struct dados {  
    char nome[50];  
    int idade;  
    float altura;  
    float peso;  
};
```



Por exemplo, para um cadastro simples de pessoas, podemos ter uma estrutura com os seguintes campos

Declaração - Struct

- Exemplo: cadastro de uma pessoa

```
struct dados {  
    char nome[50];  
    int idade;  
    float altura;  
    float peso;  
};
```

```
void main()
```

```
{
```

```
    struct dados cad;
```

```
    scanf("%s", &cad.nome);
```

```
    scanf("%d", &cad.idade);
```

```
    scanf("%f", &cad.altura);
```

```
    scanf("%f", &cad.peso);
```

```
    printf("\n%s tem %d anos, mede %.2f metros e pesa  
%.2f kg.", cad.nome, cad.idade, cad.altura, cad.peso);  
}
```



Para cada variável deste tipo
estruturado podemos
manipular cada campo
referente às informações de
uma pessoa

Declaração - Struct

- Exemplo: cadastro de 10 pessoas?

```
struct dados {  
    char nome[50];  
    int idade;  
    float altura;  
    float peso;  
};
```

```
void main()  
{  
    struct dados cad[10];  
    int i;  
    for (i=0;i<10;i++)  
    {  
        scanf("%s", &cad[i].nome);  
        scanf("%d", &cad[i].idade);  
        scanf("%f", &cad[i].altura);  
        scanf("%f", &cad[i].peso);  
    }  
}
```



Também é possível criarmos
vetores e matrizes dos nossos
tipos estruturados

Declaração - Struct

- Exemplo: cadastro de 10 pessoas?

```
void main()
{
    struct dados cad[10];
    int i;
    for (i=0;i<10;i++)
    {
        scanf("%s", &cad[i].nome);
        scanf("%d", &cad[i].idade);
        scanf("%f", &cad[i].altura);
        scanf("%f", &cad[i].peso);
    }
}
```

Neste exemplo, cada posição **i**
do vetor **cad** vai ter os dados
de uma pessoa



Definição de TIPO

- Podemos declarar nomes diferentes para tipos usando o comando **typedef**

Pode ser chato ficar declarando variáveis do tipo **struct dados_pessoa** muitas vezes no código... Então vamos chamar o tipo de outra coisa.

Definição de TIPO

- Podemos declarar nomes diferentes para tipos usando o comando **typedef**
- Exemplo: ao invés de usar **unsigned int** usar somente **uint**

➡ **typedef** unsigned int **uint**;


```
void main()  
{  
    uint x, y;  
    ...  
}
```

Para isso usamos
typedef NOME_ANTIGO nome_novo;

Definição de TIPO

- Podemos declarar nomes diferentes para tipos usando o comando **typedef**
- Exemplo: ao invés de usar **unsigned int** usar somente **uint**

```
typedef unsigned int uint;
```

```
void main()  
{  
     uint x, y;  
    ...  
}
```

A partir de então podemos
utilizar somente o **nome_novo**
para nos referirmos àquele tipo

Definição de TIPO

- Exemplos:

```
typedef unsigned int UInt;  
typedef int* pInt;  
typedef char Str50[50];
```

```
void main()  
{  
    UInt x;  
    pInt p;  
    Str50 nome;  
    ...  
}
```



Podemos fazer isso também
para ponteiros e até para
vetores e matrizes

Definição de TIPO

- Exemplos:

```
typedef unsigned int UInt;  
typedef int* pInt;  
typedef char Str50[50];
```

```
void main()  
{  
    UInt x;  
    pInt p;  
    Str50 nome;  
    ...  
}
```




Neste último exemplo, toda
variável do tipo **Str50** será um
vetor char de 50 posições

Definição de TIPO

- Exemplo: cadastro de 10 pessoas

```
struct dados {  
    char nome[50];  
    int idade;  
    float altura;  
    float peso;  
};
```

```
void main()  
{  
      
    struct dados cad[10];  
    int i;  
    for (i=0;i<10;i++)  
    {  
        scanf("%s", &cad[i].nome);  
        scanf("%d", &cad[i].idade);  
        scanf("%f", &cad[i].altura);  
        scanf("%f", &cad[i].peso);  
    }  
}
```

Lembra da nossa estrutura com
os dados das pessoas?

Também podemos mudar o
nome do nosso tipo estruturado

Definição de TIPO

- Exemplo: cadastro de 10 pessoas

```
struct dados {  
    char nome[50];  
    int idade;  
    float altura;  
    float peso;  
};
```

```
typedef struct dados Pessoa;
```



```
void main()  
{
```



```
    Pessoa cad[10];
```

```
    int i;
```

```
    for (i=0;i<10;i++)
```

```
    {
```

```
        scanf("%s", &cad[i].nome);
```

```
        scanf("%d", &cad[i].idade);
```

```
        scanf("%f", &cad[i].altura);
```

```
        scanf("%f", &cad[i].peso);
```

```
    }
```

```
}
```

Com o typedef podemos chamar apenas de **Pessoa**, por exemplo

Definição de TIPO

- Exemplo: cadastro de 10 pessoas

```
typedef struct dados {  
    char nome[50];  
    int idade;  
    float altura;  
    float peso;  
} Pessoa;
```



Podemos também mesclar o
typedef à definição da estrutura

```
void main()  
{  
    Pessoa cad[10];  
    int i;  
    for (i=0;i<10;i++)  
    {  
        scanf("%s", &cad[i].nome);  
        scanf("%d", &cad[i].idade);  
        scanf("%f", &cad[i].altura);  
        scanf("%f", &cad[i].peso);  
    }  
}
```

Exercício

- Crie uma estrutura chamada **data** contendo variáveis para o dia, o mês e o ano. Depois crie uma estrutura **pessoa** contendo o nome e data de nascimento. Escreva um programa que receba os dados de 5 pessoas e calcule quantos anos cada uma terá no dia 1º de Janeiro de 2017.
-

Ponteiros de Structs

- As informações estão agrupadas em uma estrutura, então podemos utilizar ponteiros:

```
Pessoa aluno;  
Pessoa *p = &aluno;  
(*p).idade = 20;  
p->idade = 10;  
printf("idade: %d\n", aluno.idade);  
printf("idade: %d\n", p->idade);  
printf("endereço da idade: %p\n", &p->idade);  
printf("endereço de aluno: %p\n", p);  
printf("endereço do ponteiro: %p\n", &p);
```

Um ponteiro de um tipo estruturado
é basicamente igual a um ponteiro de
tipo comum

Ponteiros de Structs

- As informações estão agrupadas em uma estrutura, então podemos utilizar ponteiros:

Para acessar os campos através um ponteiro podemos utilizar essas duas formas.

A segunda é mais comum para diferenciar melhor de uma variável comum do tipo estruturado

```
Pessoa aluno;  
Pessoa *p = &aluno;  
➡ (*p).idade = 20;  
➡ p->idade = 10;  
printf("idade: %d\n", aluno.idade);  
printf("idade: %d\n", p->idade);  
printf("endereço da idade: %p\n", &p->idade);  
printf("endereço de aluno: %p\n", p);  
printf("endereço do ponteiro: %p\n", &p);
```

Estruturas como parâmetros

- É possível passar uma estrutura como parâmetro de uma função, mas não é o ideal.
-

Estruturas como parâmetros

- É possível passar uma estrutura como parâmetro de uma função, mas não é o ideal.
- Exemplo:

```
struct dados{  
    char nome[50];  
    int idade;  
    float altura;  
    float peso;  
};
```

```
typedef struct dados Pessoa;
```



```
void imprimir(Pessoa p)  
{  
    printf("%s tem:\n", p.nome);  
    printf("%d anos, %.2f metros e %.2f  
quilos\n", p.idade, p.altura, p.peso);  
}
```

Ao passar uma estrutura como parâmetro desta forma estamos duplicando as informações na memória, mesmo que temporariamente

Estruturas como parâmetros

- É possível passar uma estrutura como parâmetro de uma função, mas não é o ideal.
- Exemplo:

Neste exemplo, cada chamada da função **imprimir** irá requisitar 62 bytes extras só para exibir as informações de 1 pessoa

```
struct dados{  
    char nome[50];  
    int idade;  
    float altura;  
    float peso;  
};
```

```
typedef struct dados Pessoa;
```



```
void imprimir(Pessoa p)  
{  
    printf("%s tem:\n", p.nome);  
    printf("%d anos, %.2f metros e %.2f  
quilos\n", p.idade, p.altura, p.peso);  
}
```

Estruturas como parâmetros

- É possível passar uma estrutura como parâmetro de uma função, mas não é o ideal.
- Exemplo:

Isso pode ser um problema para um cadastro maior, envolvendo, por exemplo, endereço, cpf, rg, data de nascimento...



```
struct dados{
    char nome[50];
    int idade;
    float altura;
    float peso;
};
```

```
typedef struct dados Pessoa;
```

```
void imprimir(Pessoa p)
{
    printf("%s tem:\n", p.nome);
    printf("%d anos, %.2f metros e %.2f
quilos\n", p.idade, p.altura, p.peso);
}
```

Estruturas como parâmetros

- O ideal é utilizar ponteiros.
- Exemplo:

Dessa forma, independente do tamanho da estrutura, cada chamada só requisitará 4 bytes extras (ou 8 bytes, dependendo do Sistema Operacional)

```
struct dados{  
    char nome[50];  
    int idade;  
    float altura;  
    float peso;  
};
```

```
typedef struct dados Pessoa;
```



```
void imprimir(Pessoa * p)  
{  
    printf("%s tem:\n", p->nome);  
    printf("%d anos, %.2f metros e %.2f  
quilos\n", p->idade, p->altura, p->peso);  
}
```

Exercícios

- Escreva uma função para calcular e retornar a distância entre dois pontos no plano, onde a função deve ser definida como:

float distancia (struct ponto *p, struct ponto *q)

Obs.: a distância entre dois pontos é dada por:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Exercícios

- Crie uma estrutura chamada **data** contendo variáveis para o dia, o mês e o ano. Depois crie uma estrutura **pessoa** contendo o nome e data de nascimento. Escreva um programa que receba os dados de 5 pessoas e calcule quantos anos cada uma terá no dia 1º de Janeiro de 2017.
 - Faça a questão anterior com uma função que faça o cálculo da idade e uma função que imprima os dados de uma pessoa
-