

Diversos Tópicos



Prof. Bruno Travençolo

typedef

- ▶ DICA: Utilizar typedef após estar bem familiarizado com a criação e manipulação de structs



typedef

- ▶ O typedef serve para apelidarmos (dar um pseudônimo) um tipo de dado
 - ▶ Apelidar – tradução da palavra inglesa *alias*
- ▶ Ou seja, podemos chamar um tipo por um outro nome
- ▶ Exemplo

// criando o tipo de dado Inteiro (em português!)

```
typedef int Inteiro
```

```
int main(){
```

```
    Inteiro num_alunos; // cria uma variável do tipo inteiro, que  
    na verdade é um tipo in
```

```
}
```



typedef

- ▶ O typedef é muito utilizado para definir nomes mais simples para estrutura, evitando carregar a palavra 'struct' sempre que referenciamos a estrutura

```
struct cadastro {  
    char nome[300];  
    int idade;  
};
```

```
// redefinindo o tipo struct cadastro  
typedef struct cadastro CadAlunos;
```



typedef

- ▶ O typedef é muito utilizado para definir nomes mais simples para estrutura, evitando carregar a palavra 'struct' sempre que referenciamos a estrutura

```
struct cadastro {  
    char nome[300];  
    int idade;  
};
```



Definição da struct

```
// redefinindo o tipo struct cadastro  
typedef struct cadastro CadAlunos;
```



typedef

- ▶ O typedef é muito utilizado para definir nomes mais simples para estrutura, evitando carregar a palavra 'struct' sempre que referenciamos a estrutura

```
struct cadastro {  
    char nome[300];  
    int idade;  
};
```

```
// redefinindo o tipo struct cadastro  
typedef struct cadastro CadAlunos;
```

}
Apelido:
CadAlunos



typedef

- ▶ O typedef é muito utilizado para definir nomes mais simples para estrutura, evitando carregar a palavra 'struct' sempre que referenciamos a estrutura

```
struct cadastro {  
    char nome[300];  
    int idade;  
};
```

```
// redefinindo o tipo struct cadastro
```

```
typedef struct cadastro CadAlunos;
```



Nome original
do tipo

Apelido

typedef

► Declarando a variável pelo apelido (*alias*)

```
struct cadastro {  
    char nome[300];  
    int idade;  
};
```

```
// redefinindo o tipo struct cadastro  
typedef struct cadastro CadAlunos;
```

```
int main()  
{
```

```
    struct cadastro aluno1;
```

```
    CadAlunos aluno2;
```

```
    strcpy(aluno1.nome, "Marcos");
```

```
    aluno1.idade = 5;
```

```
    aluno2 = aluno1;
```

A sintaxe é a mesma de
sempre:
<tipo> nome-variável

typedef

► Declarando a variável pelo apelido (*alias*)

```
struct cadastro {  
    char nome[300];  
    int idade;  
};
```

```
// redefinindo o tipo struct cadastro  
typedef struct cadastro CadAlunos;
```

```
int main()  
{
```

```
    struct cadastro aluno1;  
    CadAlunos aluno2;
```

```
    strcpy(aluno1.nome, "Marcos");  
    aluno1.idade = 5;
```



Note que ainda
podemos usar o tipo
original

```
    aluno2 = aluno1;
```

```
struct cadastro {
    char nome[300];
    int idade;
};

// redefinindo o tipo struct cadastro
typedef struct cadastro CadAlunos;

int main()
{

    struct cadastro aluno1;
    CadAlunos aluno2;

    strcpy(aluno1.nome, "Marcos");
    aluno1.idade = 5;

    aluno2 = aluno1;

    printf("Nome: %s\n", aluno2.nome);
    printf("Idade: %d\n", aluno2.idade);

    return 0;
}
```

Mesmo que os tipos tenham nomes diferentes (um é 'struct cadastro' e o outro é CadAlunos) temos que lembrar que eles são do mesmo tipo, e pode-se então fazer atribuições



```
struct cadastro {  
    char nome[300];  
    int idade;  
};
```

```
// redefinindo o tipo struct cadastro  
typedef struct cadastro cadastro;
```

```
int main()  
{  
  
    struct cadastro aluno1;  
    cadastro aluno2;  
  
    strcpy(aluno1.nome, "Marcos");  
    aluno1.idade = 5;  
  
    aluno2 = aluno1;  
  
    printf("Nome: %s\n", aluno2.nome);  
    printf("Idade: %d\n", aluno2.idade);  
  
    return 0;  
}
```

Pode-se usar o
mesmo nome da
struct



```
// redefinindo o tipo struct cadastro
typedef struct cadastro cadastro;
```

Pode-se fazer a definição
do novo nome antes
mesmo de declarar a struct

```
struct cadastro {
    char nome[300];
    int idade;
};
```

```
int main()
{
```

```
    struct cadastro aluno1;
    cadastro aluno2;
```

```
    strcpy(aluno1.nome, "Marcos");
    aluno1.idade = 5;
```

```
    aluno2 = aluno1;
```


```
    printf("Nome: %s\n", aluno2.nome);
    printf("Idade: %d\n", aluno2.idade);
```

```
    return 0;
```


```
}
```

```
typedef struct cadastro {  
    char nome[300];  
    int idade;  
} CadAluno;
```

Pode-se fazer a definição
do novo nome junto com a
definição da struct



```
int main()  
{  
  
    struct cadastro aluno1;  
    CadAluno aluno2;  
  
    strcpy(aluno1.nome, "Marcos");  
    aluno1.idade = 5;  
  
    aluno2 = aluno1;  
  
    printf("Nome: %s\n", aluno2.nome);  
    printf("Idade: %d\n", aluno2.idade);  
  
    return 0;  
}
```



```
typedef struct {  
    char nome[300];  
    int idade;  
} CadAluno;
```



Pode-se fazer a definição
do novo nome junto com a
definição da struct

(outra opção)

```
int main()  
{  
  
struct cadastro aluno1;  
CadAluno aluno2;  
  
strcpy(aluno1.nome, "Marcos");  
aluno1.idade = 5;  
  
aluno2 = aluno1;  
  
printf("Nome: %s\n", aluno2.nome);  
printf("Idade: %d\n", aluno2.idade);  
  
return 0;  
}
```



Argumentos de linha de comando

- ▶ Ao executar um programa é possível passar alguns parâmetros diretamente do sistema operacional para o programa aplicativo
- ▶ Para isso, na chamada do programa, em linha de comando, basta digitar os valores dos argumentos
- ▶ Exemplo
- ▶ Imagine um programa chamado **eco**, que mostra na tela tudo o que o usuário passa de parâmetro para o programa



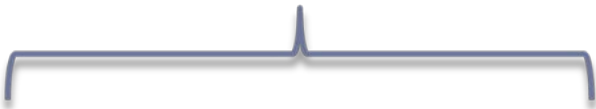
```
C:\Users\Bruno>eco laranja  
laranja
```

```
C:\Users\Bruno>eco laranja banana  
laranja banana
```

```
C:\Users\Bruno>eco laranja banana 22  
laranja banana 22
```



Local do programa



```
C:\Users\Bruno>eco laranja  
laranja
```

```
C:\Users\Bruno>eco laranja banana  
laranja banana
```

```
C:\Users\Bruno>eco laranja banana 22  
laranja banana 22
```



Nome do arquivo executável do programa (eco.exe)



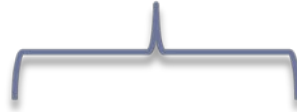
```
C:\Users\Bruno>eco laranja  
laranja
```

```
C:\Users\Bruno>eco laranja banana  
laranja banana
```

```
C:\Users\Bruno>eco laranja banana 22  
laranja banana 22
```



Chamada, com um só argumento



```
C:\Users\Bruno>eco laranja  
laranja
```


```
C:\Users\Bruno>eco laranja banana  
laranja banana
```

```
C:\Users\Bruno>eco laranja banana 22  
laranja banana 22
```



Chamada, com um só argumento

C:\Users\Bruno>eco laranja
laranja



C:\Users\Bruno>eco laranja banana
laranja banana

C:\Users\Bruno>eco laranja banana 22
laranja banana 22



C:\Users\Bruno>eco laranja

laranja

Chamada com dois argumentos

C:\Users\Bruno>eco laranja banana

laranja banana

Saída

C:\Users\Bruno>eco laranja banana 22

laranja banana 22

```
C:\Users\Bruno>eco laranja  
laranja
```

```
C:\Users\Bruno>eco laranja banana  
laranja banana
```

Chamada com três argumentos

```
C:\Users\Bruno>eco laranja banana 22  
laranja banana 22
```

Saída

