



# Prática Profissional II – Linguagem de Programação Estruturada

Curso: Análise e Desenvolvimento de Sistemas

Modalidade: Presencial

Professor Esp. Wesley Tschiedel

Email: [wesley.tschiedel@ucb.br](mailto:wesley.tschiedel@ucb.br)

# **Registros (estruturas). Exercícios de fixação.**

**Registro (struct)** é uma coleção de uma ou mais variáveis, possivelmente de tipos diferentes, colocadas juntas sob um único nome.

O exemplo de um registro é a folha de pagamento de uma empresa:

Um funcionário é descrito por um conjunto de atributos tais como nome **("string")**, o número do seu departamento **(inteiro)**, salário **(float)**, dentre outros elementos.



Uma **struct** consiste de um certo número de itens de dados, chamados **membros** do registro, que **não** necessitam **ser** do **mesmo tipo**, agrupados juntos.

## EXEMPLO

```
1. struct facil {      //define tipo de dado
2.     int num;        //inteiro no registro
3.     char ch;        //caractere no registro
4. };
5. main(){
6.     struct facil reg1; //declara reg1 como tipo do registro
7.     reg1.num=2;
8.     reg1.ch='Z';
9.     printf("reg1.num = %d, reg1.ch = %c \n", reg1.num,
10.    reg1.ch);
11. }
```

O programa mostra os três principais aspectos de registros:

- 1. definição do tipo struct;**
- 2. declaração de variáveis;**
- 3. acesso a seus membros.**

ESPECIFICAÇÃO DO TIPO DE DADO

“ETIQUETA” DO REGISTRO

```
struct facil {  
    int num;  
    char ch;  
};
```

MEMBROS DA ESTRUTURA



## DECLARANDO UM TIPO REGISTRO

Primeiro você deve definir o tipo da estrutura que você quer criar.

Um registro pode conter qualquer número de membros de diferentes tipos, o programa deve avisar o compilador de como é formado o registro particular antes de seu uso.

No exemplo anterior as seguintes instruções definem o tipo do registro:

```
struct facil {  
    int num;  
    char ch;  
};
```

Estas instruções definem um novo tipo de dado chamado **struct facil**.

Cada variável deste tipo será composta por dois elementos:

- uma variável inteira chamada **num;**
- uma variável caractere chamada **ch.**



A instrução anterior não declara qualquer variável, por isso não é reservado nenhum espaço de memória.

Somente é mostrado ao compilador como é formado o tipo **struct facil**.



A palavra **struct** informa ao compilador que um tipo de dado está sendo declarado e o nome **facil** é chamado “**etiqueta**” e nomeia o registro particular que está sendo definido.

**A “etiqueta” não é o nome de uma variável, e  
sim o nome de um tipo.**

Os membros do registro devem estar entre chaves, e a instrução completa termina por ponto e vírgula.



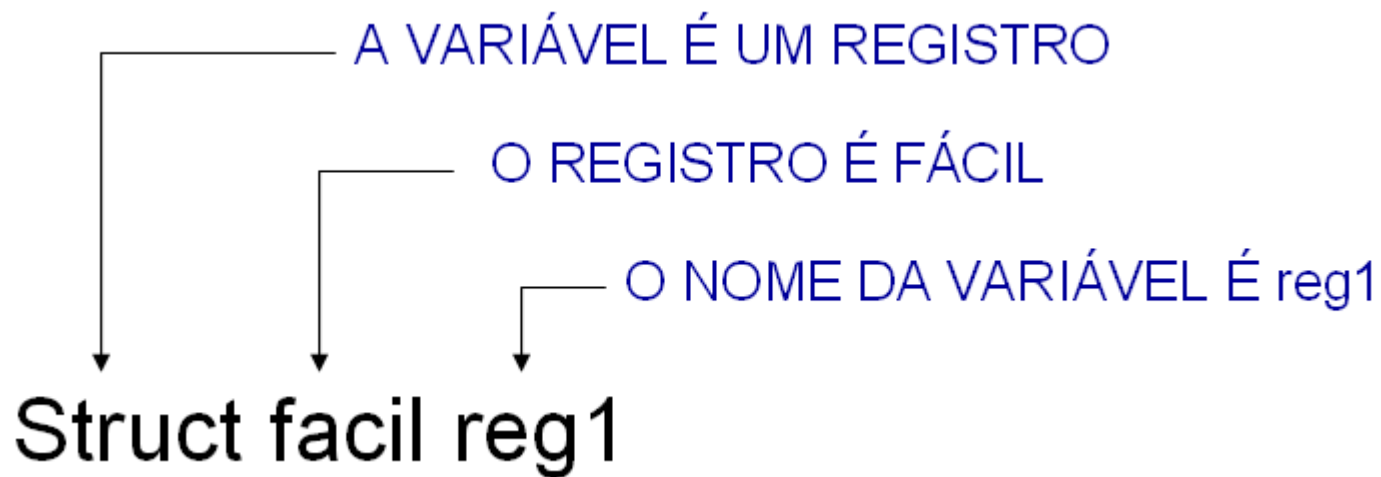
**UM REGISTRO É UM TIPO DE DADO CUJO  
FORMATO É DEFINIDO PELO PROGRAMADOR.**



# DECLARANDO AS VARIÁVEIS DO REGISTRO

A VARIÁVEL É UM REGISTRO  
O REGISTRO É FÁCIL  
O NOME DA VARIÁVEL É reg1

Struct facil reg1



A instrução anterior solicita ao compilador a alocação de espaço de memória suficiente para armazenar a variável **reg1** que é do tipo **struct facil**, neste caso **3 bytes** (dois para o inteiro e um para o caractere).

## ACESSANDO MEMBROS DA ESTRUTURA

Os registros usam o **“operador ponto” (.)**, que é também chamado **“operador de associação”** para acesso aos elementos do registro.



A sintaxe apropriada para referenciar “num” que é parte do registro **reg1**, é:

**reg1.num**

O nome da variável que precede o ponto é o nome do registro e o nome que o segue é o de um membro específico da estrutura.



**O OPERADOR (.) CONECTA O NOME DA  
VARIÁVEL ESTRUTURA A UM MEMBRO DO  
REGISTRO.**

## MÚLTIPLOS REGISTROS DE MESMO TIPO

Da mesma forma que podemos ter várias variáveis do tipo **int** em um programa, podemos também ter qualquer número de variáveis do tipo de um registro pré-definido.

## EXEMPLO

```
1. main(){
2.     struct facil {
3.         int num;
4.         char ch;
5.     };
6.     struct facil xx1;
7.     struct facil xx2;
8.     xx1.num=2;
9.     xx1.ch='Z';
10.    xx2.num=3;
11.    xx2.ch='Y';
12.    printf("xx1.num = %d, xx1.ch = %c \n", xx1.num, xx1.ch);
13.    printf("xx2.num = %d, xx2.ch = %c \n", xx2.num, xx2.ch);
14.}
```



## COMBINANDO DECLARAÇÕES

Você pode combinar em uma instrução a definição do tipo registro e a declaração das variáveis tipo registro.



## EXEMPLO

```
1. main()
2. {
3.     struct facil {
4.         int num;
5.         char ch;
6.     } xx1, xx2;
7.     xx1.num=2;
8.     xx1.ch='Z';
9.     xx2.num=3;
10.    xx2.ch='Y';
11.    printf("xx1.num = %d, xx1.ch = %c \n", xx1.num, xx1.ch);
12.    printf("xx2.num = %d, xx2.ch = %c \n", xx2.num, xx2.ch);
13.}
```

## DEFINIÇÃO DE REGISTROS SEM “ETIQUETA”

A convenção normal é a de usar etiqueta do registro quando a expectativa é criar várias variáveis do mesmo tipo estrutura.

Se você espera usar uma única declaração de variável do tipo estrutura, você pode combinar a declaração com a definição da estrutura e omitir a etiqueta:

```
struct {  
    int num;  
    char ch;  
} xx1, xx2;
```



## INICIALIZANDO REGISTRO

Podemos inicializar uma variável do tipo registro com elemento pré-definidos.



## EXEMPLO

```
1. struct livro {
2.     char titulo[30];
3.     int regnum;
4. };
5. main(){
6.     struct livro livro1 = {"Helena", 102};
7.     struct livro livro2 = {"Iracema", 321};
8.     printf("\nLista de livros:\n");
9.     printf(" Titulo: %s\n", livro1.titulo);
10.    printf(" Numero do registro: %d\n", livro1.regnum);
11.    printf(" Titulo: %s\n", livro2.titulo);
12.    printf(" Numero do registro: %d\n", livro2.regnum);
13. }
```

## ATRIBUIÇÕES ENTRE ESTRUTURAS

Se duas variáveis são estrutura do mesmo tipo:

**var2 = var1;**

## EXEMPLO

```
1. struct livro {
2.     char titulo[30];
3.     int regnum;
4. };
5. main(){
6.     struct livro livro1 = {"Helena", 102};
7.     struct livro livro2;
8.     livro2=livro1;
9.     printf("\nLista de livros:\n");
10.    printf(" Titulo: %s\n", livro1.titulo);
11.    printf(" Numero do registro: %d\n", livro1.regnum);
12.    printf(" Titulo: %s\n", livro2.titulo);
13.    printf(" Numero do registro: %d\n", livro2.regnum);
14. }
```



## REGISTROS ANINHADOS

Podemos ter registros que contêm outros registros. O que pode ser um poderoso caminho para a criação de tipos de dados complexos.



```
1. struct livro {
2.     char titulo[30];
3.     int codLivro;
4. };
5. struct grupo {
6.     struct livro dicionario;
7.     struct livro literatura;
8. };
9. struct grupo grupo1 = {"Aurelio", 134}, {"Iracema", 321}};
10. main() {
11.     printf("\nDicionario:\n");
12.     printf(" Titulo: %s\n", grupo1.dicionario.titulo);
13.     printf(" Numero do registro: %d\n", grupo1.dicionario.codLivro);
14.     printf("\nLiteratura:\n");
15.     printf(" Titulo: %s\n", grupo1.literatura.titulo);
16.     printf(" Numero do registro: %d\n", grupo1.literatura.codLivro);
17. }
```

# PASSANDO ESTRUTURAS PARA FUNÇÕES

```
#include<stdio.h>
#include <stdlib.h>//atoi
void list(struct livro liv);
struct livro novonome();
struct livro {
    char titulo[30];
    int regnum;
};
```

```
int main(){
    struct livro livro1;
    struct livro livro2;

    livro1=novonome();
    livro2=novonome();
    list(livro1);
    list(livro2);
}
```

```
struct livro novonome(){
    char numstr[81];
    struct livro livr;

    printf("\nNovo livro\nDigite titulo: ");
    gets(livr.titulo);
    printf("Digite o numero do registro (3 digitos): ");
    gets(numstr);
    livr.regnum=atoi(numstr);// a função atoi() transforma uma string em
inteiro.
    return livr;
}

void list(struct livro liv){
    printf("\nLivro: \n");
    printf(" Titulo: %s\n", liv.titulo);
    printf(" Numero do registro: %d\n", liv.regnum);
}
```



# **Atividade Prática**



# Referências Bibliográficas

## Básica:

- EVARISTO, J., **Aprendendo a programar programando em C**, Book Express, 2001, 205p.
- MIZRAHI, V. V., **Treinamento em Linguagem C**, Módulo 1 e 2, Makron Books do Brasil Editora Ltda, 1990, 273 p.
- SCHILDT, H., **C Completo e Total**, Editora Makron Books doBrasil Editora Ltda, 1997, 827p.

# Referências Bibliográficas

## Complementar:

- DEITEL, H. M. e Deitel, P. J., **C++ Como Programar**, 3. ed. Porto Alegre: Artmed Editora S.A, 2001. 1098 p.
- MANZANO, J. A. N. G. **Estudo Dirigido: Linguagem C**. 6. ed. São Paulo: Érica, 2002.
- SOFFNER, Renato. **Algoritmos e programação em linguagem C**. São Paulo Saraiva 2013.
- TENENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. **Estruturas de Dados Usando C**. São Paulo: Makron Books, 1995.
- ZIVIANI, Nivio. **Projeto de algoritmos: com implementações em Pascal e C**. 3. ed., rev. e ampl. São Paulo, SP: Cengage Learning, 2015. xx, 639 p.