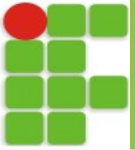


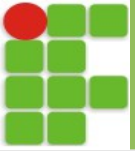
# Introdução à Programação

Estruturas, Uniões e Enumerações



# Estruturas

- Suponha que exista uma folha de pagamento em uma empresa, onde cada funcionário é descrito por um conjunto de atributos
  - ♦ Nome (*um vetor de caracteres*)
  - ♦ Número de seu departamento (*inteiro*)
  - ♦ Salário (*float*)
- Como descrever este tipo de registro em C?



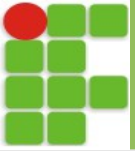
# Estruturas

## ■ Utilizando estruturas

- ♦ São tipos de dados compostos, cujos elementos individuais podem ser de tipos diferentes

## ■ Estruturas podem ser composta de variáveis do tipo

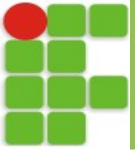
- ♦ `int`, `double`, `float`, `char`, `long`, `ponteiros`, `vetores`, *etc.*



# Estruturas

- Por exemplo, podemos criar um tipo funcionário
  - ♦ Que possua um primeiro nome, um último nome e um salário

```
struct funcionario {  
    char primeiroNome[10];  
    char ultimoNome[10];  
    double salario;  
};
```



# Inicializando Estruturas

- Estruturas podem ser inicializadas de forma semelhante a um array

```
#include <stdio.h>
struct funcionario {
    char primeiroNome[10];
    char ultimoNome[10];
    double salario;
};

int main(void) {
    struct funcionario func = {"Maria", "Antunes", 3459.99};
    printf("Funcionário: %s %s R$ %.2lf\n",
           func.primeiroNome, func.ultimoNome, func.salario);
    return 0;
}
```

The screenshot shows a terminal window titled "C:\cygwin\bin\sh.exe". The output of the program is displayed in two lines: "Funcionária: Maria Antunes R\$ 3459.99" and "[Pressione Enter para fechar a janela]". The text is in a monospaced font, with the first line in black and the second line in red. The terminal has a blue title bar and standard window controls.

# Estruturas

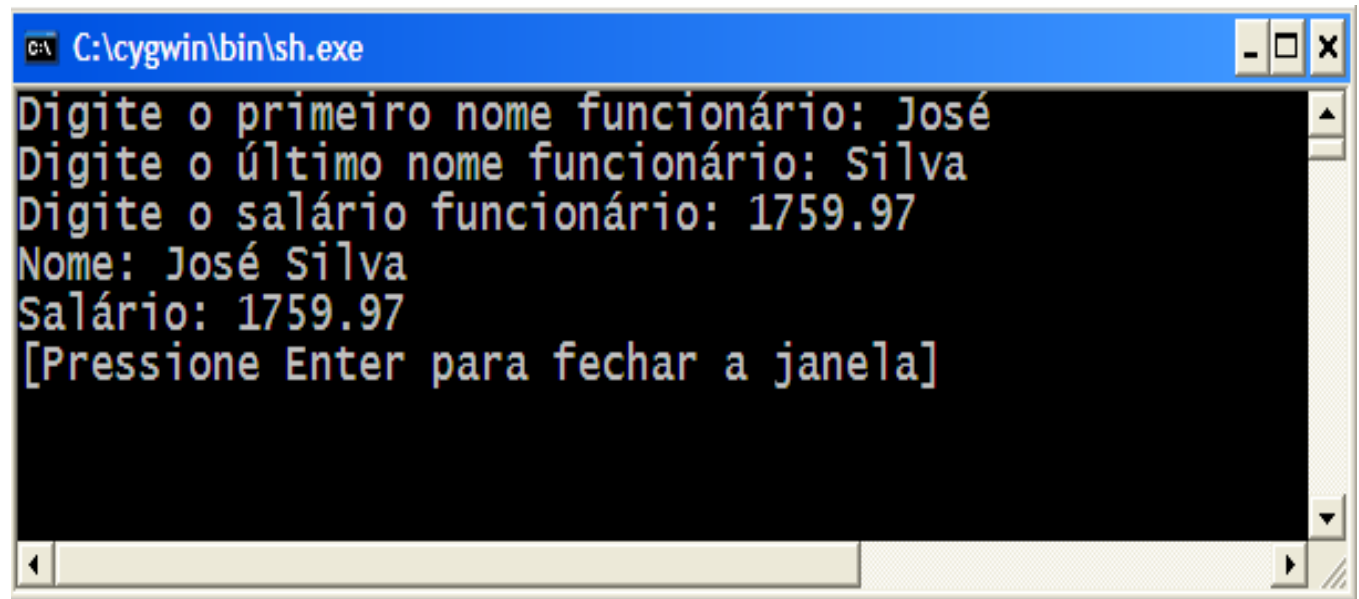
```
#include <stdio.h>

struct funcionario {
    char primeiroNome[10];
    char ultimoNome[10];
    double salario;
};

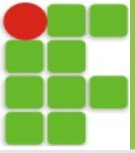
int main(void) {
    struct funcionario func1;
    printf("Digite o primeiro nome funcionário: "); scanf("%9s", func1.primeiroNome);
    printf("Digite o último nome funcionário: "); scanf("%9s", func1.ultimoNome);

    do {
        printf("Digite o salário funcionário: "); scanf("%lf", &func1.salario);
    } while (func1.salario <= 0);

    printf("Nome: %s %s\n", func1.primeiroNome, func1.ultimoNome);
    printf("Salário: %.2lf\n", func1.salario);
    return 0;
}
```



```
C:\cygwin\bin\sh.exe
Digite o primeiro nome funcionário: José
Digite o último nome funcionário: Silva
Digite o salário funcionário: 1759.97
Nome: José Silva
Salário: 1759.97
[Pressione Enter para fechar a janela]
```



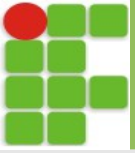
# Boa Prática de Programação

## ■ Importante

- ♦ Sempre limite o tamanho da string lido da entrada
- ♦ Ex. `scanf("%9s", func1.primeiroNome)`

## ■ *Nunca utilize a função gets!!!*

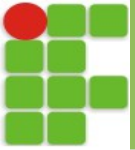
- ♦ Ex. `gets(func1.primeiroNome)`
- ♦ Ela é insegura por não verificar o tamanho da entrada



# Boa Prática de Programação

- Mais informações em
  - ♦ [http://www.owasp.org/index.php/Buffer\\_Overflow](http://www.owasp.org/index.php/Buffer_Overflow)





# Estruturas

- Uma estrutura pode ser composta por outras estruturas

```
struct nomeFuncionario {  
    char primeiro[10];  
    char ultimo[10];  
};  
  
struct funcionario {  
    struct nomeFuncionario nome;  
    double salario;  
};
```

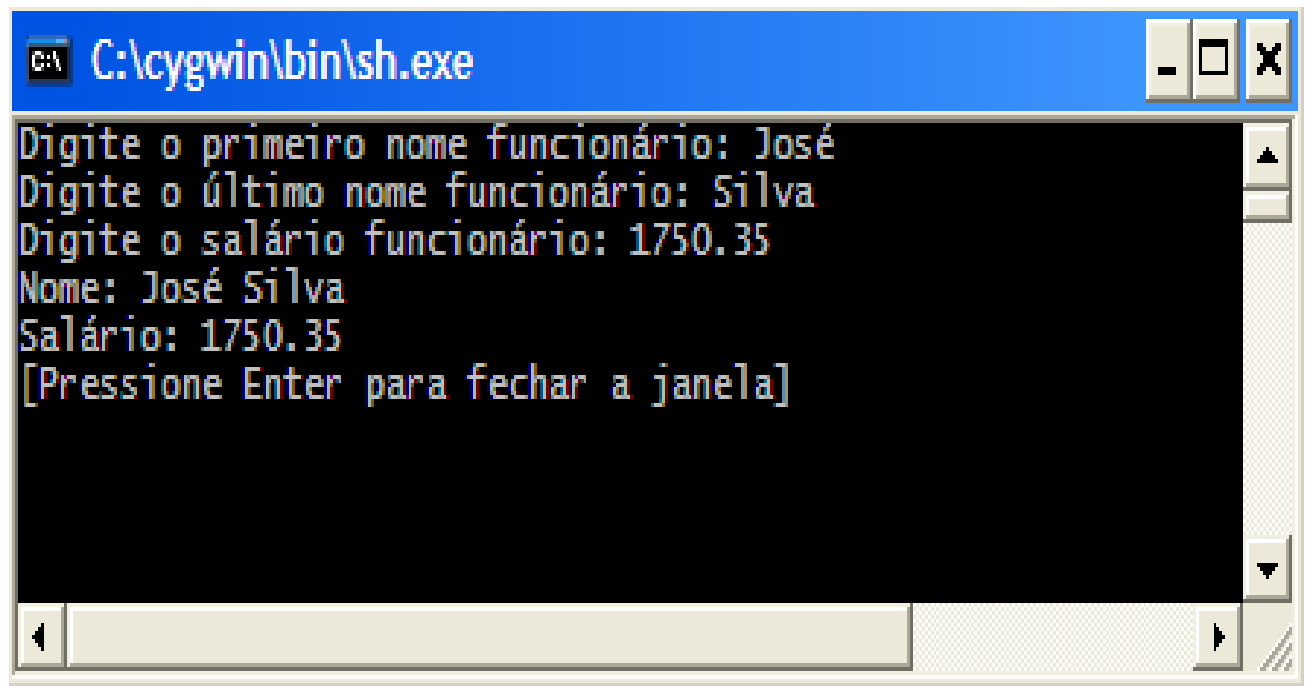
# Estruturas

```
#include <stdio.h>

struct nomeFuncionario {
    char primeiro[10];
    char ultimo[10];
};

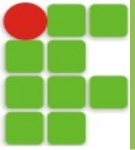
struct funcionario {
    struct nomeFuncionario nome;
    double salario;
};

int main(void) {
    struct funcionario func1;
    printf("Digite o primeiro nome funcionário: "); scanf("%9s", func1.nome.primeiro);
    printf("Digite o último nome funcionário: "); scanf("%9s", func1.nome.ultimo);
    do {
        printf("Digite o salário funcionário: "); scanf("%lf", &func1.salario);
    } while (func1.salario <= 0);
    printf("Nome: %s %s\n", func1.nome.primeiro, func1.nome.ultimo);
    printf("Salário: %.2lf\n", func1.salario);
    return 0;
}
```



The screenshot shows a terminal window titled "C:\cygwin\bin\sh.exe". The output of the program is as follows:


```
Digite o primeiro nome funcionário: José
Digite o último nome funcionário: Silva
Digite o salário funcionário: 1750.35
Nome: José Silva
Salário: 1750.35
[Pressione Enter para fechar a janela]
```

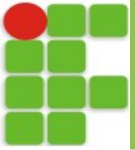


# Estruturas

- Uma estrutura não pode ser composta por ela mesma
  - ♦ O código abaixo resulta em erro de compilação

```
struct nomeFuncionario {  
    char primeiro[10];  
    char ultimo[10];  
};  
  
struct funcionario {  
    struct nomeFuncionario nome;  
    struct funcionario chefe;  
    double salario;  
};
```

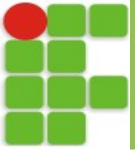




# Estruturas Auto-Referenciadas

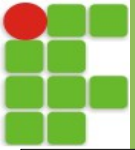
- São estruturas que contêm um ponteiro para uma estrutura do mesmo tipo

```
struct nomeFuncionario {  
    char primeiro[10];  
    char ultimo[10];  
};  
  
struct funcionario {  
    struct nomeFuncionario nome;  
    struct funcionario *chefe;  
    double salario;  
};
```



# Estruturas Auto-Referenciadas

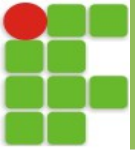
```
/*  
 * A função recebe dois ponteiros para a estrutura funcionario  
 * Observe a palavra struct para denotar que é um ponteiro para uma estrutura  
 */  
void lerFuncionario(struct funcionario* func, struct funcionario* chefe) {  
    printf("Digite o primeiro nome: ");  
    scanf("%9s", (*func).nome.primeiro);  
    printf("Digite o último nome: ");  
    scanf("%9s", (*func).nome.ultimo);  
  
    do {  
        printf("Digite o salário: ");  
        scanf("%lf", &(*func).salario);  
    } while ((*func).salario <= 0);  
  
    (*func).chefe = chefe;  
}
```



# Estruturas Auto-Referenciadas

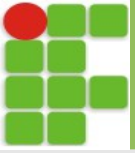
```
#include <stdio.h>

int main(void) {
    struct funcionario *empregado, *patrao;
    //Alocando espaço para a estrutura dinamicamente
    empregado = (struct funcionario*) malloc(sizeof(struct funcionario));
    patrao = (struct funcionario*) malloc(sizeof(struct funcionario));
    if (empregado != NULL && patrao != NULL) {
        printf("Digite os dados do chefe\n");
        lerFuncionario(patrao, NULL);
        printf("Digite os dados do subordinado\n");
        lerFuncionario(empregado, patrao);
        printf("Empregado: %s %s R$ %.2lf\n", (*empregado).nome.primeiro,
            (*empregado).nome.ultimo, (*empregado).salario);
        printf("Chefe: %s %s R$ %.2lf\n", empregado->chefe->nome.primeiro,
            empregado->chefe->nome.ultimo, empregado->chefe->salario);
        free(empregado);
        free(patrao);
    }
    return 0;
}
```



# Estruturas Auto-Referenciadas

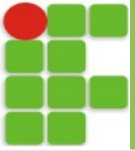
```
C:\cygwin\bin\sh.exe
Digite os dados do chefe
Digite o primeiro nome: Luís
Digite o último nome: Silva
Digite o salário: 12769.45
Digite os dados do subordinado
Digite o primeiro nome: José
Digite o último nome: Silva
Digite o salário: 1234.56
Empregado: José Silva R$ 1234.56
Chefe: Luís Silva R$ 12769.45
[Pressione Enter para fechar a janela]
```



# Estruturas Auto-Referenciadas

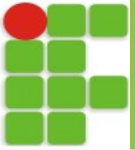
- Se declaramos um ponteiro para uma estruturas
  - ♦ Digamos funcPtr
  - ♦ funcPtr->salario equivale a (\*funcPtr).salario





# Typedef

- A palavra chave *typedef* provém um mecanismo para criar sinônimos (*alias*)
  - ♦ Para um tipo de dado previamente definido



# Typedef

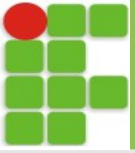
```
#include <stdio.h>
```

```
struct contaStruct {  
    char correntista[30];  
    double saldo;  
};
```

```
typedef struct contaStruct conta;
```

```
int main(void) {  
    conta conta1 = {"Carlota Joaquina", 11786.78};  
    printf("Funcionária: %s R$ %.2lf\n", conta1.correntista, conta1.saldo);  
    return 0;  
}
```

A screenshot of a terminal window titled "C:\cygwin\bin\sh.exe". The window displays the output of the C program: "Funcionária: Carlota Joaquina R\$ 11786.78" followed by a prompt "[Pressione Enter para fechar a janela]". The text is displayed in a monospaced font with some color coding (blue for 'Funcionária', red for 'R\$', green for '11786.78'). The terminal has a standard Windows-style title bar and scrollbars.



# Unões

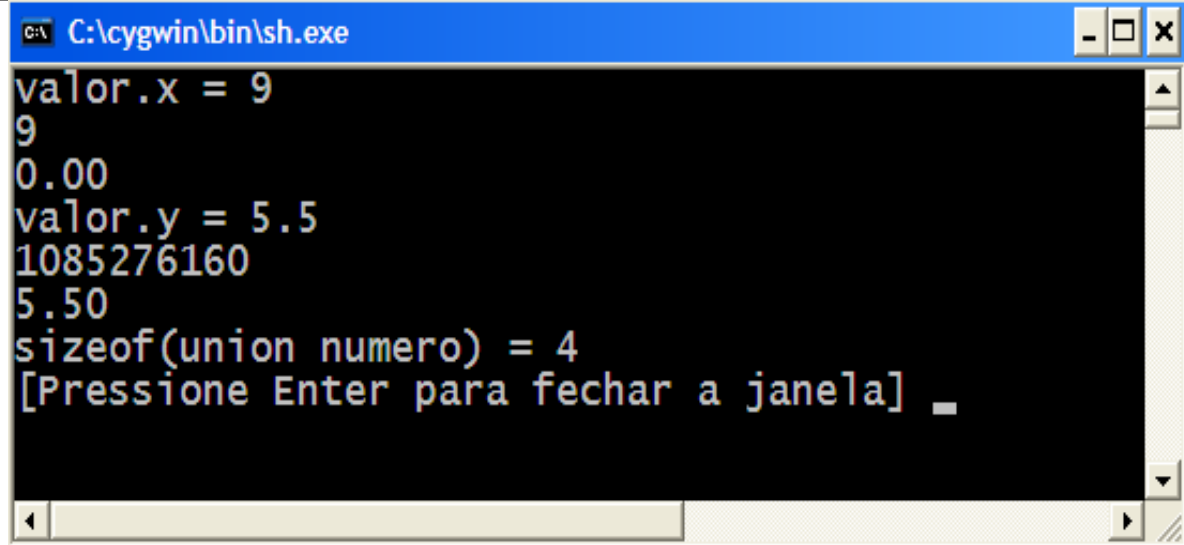
- Assim como estruturas, uniões são tipos de dados derivados
- Uniões são declaradas com a palavra chave union
  - ♦ Em uma union, as variáveis compartilham o mesmo espaço de armazenamento
  - ♦ Significa que apenas uma variável pode ser armazenada por vez

# Uniãoes

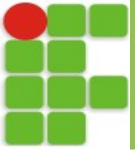
```
#include <stdio.h>
```

```
union numero {  
    int x;  
    float y;  
};
```

```
int main(void) {  
    union numero valor;  
    valor.x = 9;  
    printf("valor.x = 9\n%d\n", valor.x);  
    printf("%.2f\n", valor.y);  
  
    valor.y = 5.5;  
    printf("valor.y = 5.5\n%d\n", valor.x);  
    printf("%.2f\n", valor.y);  
    printf("sizeof(union numero) = %d\n", sizeof(union numero));  
    return 0;  
}
```



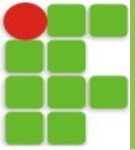
```
C:\cygwin\bin\sh.exe  
valor.x = 9  
9  
0.00  
valor.y = 5.5  
1085276160  
5.50  
sizeof(union numero) = 4  
[Pressione Enter para fechar a janela]
```



# Enumerações

- Uma enumeração é um conjunto de inteiros constantes representados por um identificador
  - ♦ Uma enumeração é introduzida pela palavra chave *enum*
  - ♦ O primeiro inteiro tem valor 0
  - ♦ Os demais tem o valor do anterior mais 1

```
enum meses {  
    JAN, FEV, MAR, ABR, MAI, JUN, JUL, AGO, SET, OUT, NOV, DEZ  
};
```



# Enumerações

- Se não quisermos que a enum comece com 0
  - ♦ Podemos especificar o valor da primeira constante

```
enum meses {  
    JAN = 1, FEV, MAR, ABR, MAI, JUN, JUL, AGO, SET, OUT, NOV, DEZ  
};
```

# Enumerações

```
#include <stdio.h>
```

```
enum meses {  
    JAN = 1, FEV, MAR, ABR, MAI, JUN, JUL, AGO, SET, OUT, NOV, DEZ  
};
```

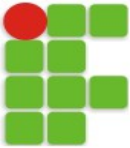
```
int main(void) {  
    enum meses mes;
```

```
    const char * const nomesMeses[] =  
        {"", "Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho",  
        "Julho", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro"};
```

```
    for (mes = JAN; mes <= DEZ; mes++) {  
        printf("%2d%10s\n", mes, nomesMeses[mes]);  
    }
```

```
    return 0;
```

```
}
```



# Enumerações

A screenshot of a Cygwin terminal window. The title bar is blue and contains the text "C:\cygwin\bin\sh.exe" and standard window control buttons (minimize, maximize, close). The terminal area has a black background with white text. It displays a list of months, each preceded by a number from 1 to 12. The text is as follows:

```
1 Janeiro
2 Fevereiro
3 Março
4 Abril
5 Maio
6 Junho
7 Julho
8 Agosto
9 Setembro
10 Outubro
11 Novembro
12 Dezembro
```

The terminal window includes a vertical scrollbar on the right side and a horizontal scrollbar at the bottom.