



## USP - ICMC - SSC SSC 0301 - 2o. Semestre 2013

### Disciplina de Introdução à Computação para Engenharia Ambiental

**Prof. Dr. Fernando Santos Osório / Prof. Artur Lovato Cunha**

**LRM - Laboratório de Robótica Móvel do ICMC / CROB-SC**

**Email: [fosorio@icmc.usp.br](mailto:fosorio@icmc.usp.br) ou [fosorio@gmail.com](mailto:fosorio@gmail.com)**

**Email: [arturlc@icmc.usp.br](mailto:arturlc@icmc.usp.br)**

**Página Pessoal: <http://www.icmc.usp.br/~fosorio/>**

**Material on-line:**

**Wiki ICMC - <http://wiki.icmc.usp.br/index.php>**

**Wiki SSC0301 - [http://wiki.icmc.usp.br/index.php/SSC-301-2013\(fosorio\)](http://wiki.icmc.usp.br/index.php/SSC-301-2013(fosorio))**

## Agenda:

- **Tipos de dados Homogêneos: Vetores e Arrays**
- **Tipos de dados Heterogêneos: Estruturas (Structs)**  
**Dados compostos: estruturas de registros de dados**
- **Estruturas de dados compostas usando os comandos typedef e struct**
- **Exercícios**

**Informações Complementares e Atualizadas:  
Consulte REGULARMENTE  
O material disponível na COTEIA**

## Agenda:

- \* **Tipos de dados compostos:**
  - Estruturas compostas tipo VETOR (dados homogêneos)**
  - Estruturas compostas tipo STRUCT (dados heterogêneos)**
- \* **Definição de novos tipos de Dados: Typedef**
- \* **Vetor de Struct!**
- \* **Arquivos com Structs!**

## Exemplos Práticos

## Vetores: Estruturas compostas por dados homogêneos

### Tipos de Dados em “C” : Vetores

#### - Vetores numéricos:

```
int    Hora[24];      => Hora[0] .. Hora[23] com valores do tipo “int”  
double Notas[10];   => Notas[0] .. Notas[9] com valores do tipo “double”  
Notas[0] = 10.0;
```

N[0]	N[1]	N[2]	N[3]	N[4]	N[5]	N[6]	N[7]	N[8]	N[9]
------	------	------	------	------	------	------	------	------	------

#### - Vetores de caracteres:

```
char  Letras[26];    => Letras[0] .. Letras[25] com valores do tip “char”  
Letras[0] = ‘a’; Letras[25] = ‘z’;
```

```
char  Nome[10];      => Nome[0] .. Nome[9] onde uma posição é reservada para a  
                        marca de fim da string de nome! (Marca = ‘\0’)
```

```
strcpy(Nome, “123456789”); => O Nome não deve ter mais de 9 caracteres, pois o décimo é o ‘\0’  
Strings são manipuladas através de rotinas especiais:  
strcpy, strlen, strcmp, sprintf, sscanf, ... #include <string.h>
```

N[0]	N[1]	N[2]	N[3]	N[4]	N[5]	N[6]	N[7]	N[8]	N[9]
F	U	L	A	N	O	\0	?	?	?

## Vetores e Matrizes: Estruturas compostas por dados homogêneos

### Tipos de Dados em “C” : Vetores bi-dimensionais

- Vetores numéricos bi-dimensionais:

```
int Matriz [3][10];
```

```
Matriz[0][0] = 1; ... Matriz [2][9] = 30;
```

M[0][0]	M[0][1]	M[0][2]	M[0][3]	M[0][4]	M[0][5]	M[0][6]	M[0][7]	M[0][8]	M[0][9]
M[1][0]	M[1][1]	M[1][2]	M[1][3]	M[1][4]	M[1][5]	M[1][6]	M[1][7]	M[1][8]	M[1][9]
M[2][0]	M[2][1]	M[2][2]	M[2][3]	M[2][4]	M[2][5]	M[2][6]	M[2][7]	M[2][8]	M[2][9]

- Inicialização de vetores:

```
int num [5] = { 1, 2, 3, 4, 5 };
```

```
char vogais[5] = { 'a', 'e', 'i', 'o', 'u' };
```

```
double matriz [3][2] = { { 0,0 }, { 0,1 },  
                          { 1,0 }, { 1,1 },  
                          { 2,0 }, { 2,1 } };
```

## Criando novos tipos de dados: TYPEDEF

- Comando TYPEDEF : Criando novos tipos

```
typedef enum { seg, ter, qua, qui, sex } dias_semana;  
typedef enum { sab, dom } fim_de_semana;  
typedef short int tipo_ano;
```

```
tipo_ano ano_nascimento; /* A variável ano_nascimento é do tipo “short int” */  
dias_semana compromisso; /* compromisso é uma variável do tipo dias_semana */
```

**Exemplo:**

```
typedef double nota;
```

```
main()
```

```
{
```

```
    nota p1,p2;  
    nota media;
```

```
    printf("Nota da Prova 1: "); scanf ("%lf",&p1);
```

```
    printf("Nota da Prova 2: "); scanf ("%lf",&p2);
```

```
    media=(p1+p2)/2.0;
```

```
    printf("Media: %.2lf",media);
```

```
    getch();
```

```
}
```

## Criando novos tipos de dados: **TYPEDEF**

Cria um tipo de dados chamado “nota” do tipo “double”

**Exemplo:**

```
typedef double nota;
```

```
main()
```

```
{
```

```
    nota p1,p2;
```

```
    nota media;
```

```
    printf("Nota da Prova 1: "); scanf ("%lf",&p1);
```

```
    printf("Nota da Prova 2: "); scanf ("%lf",&p2);
```

```
    media=(p1+p2)/2.0;
```

```
    printf("Media: %.2lf",media);
```

```
    getch();
```

```
}
```

## Criando novos tipos de dados: **TYPEDEF**

Cria um tipo de dados chamado “t\_conceito” do tipo “char”

```
typedef char t_conceito;
```

```
Nota = 'A'; ... ; Nota = 'B';
```

Cria uma variável “Nota” do tipo “t\_conceito”

```
t_conceito Nota;
```

---

```
typedef unsigned int t_idade;
```

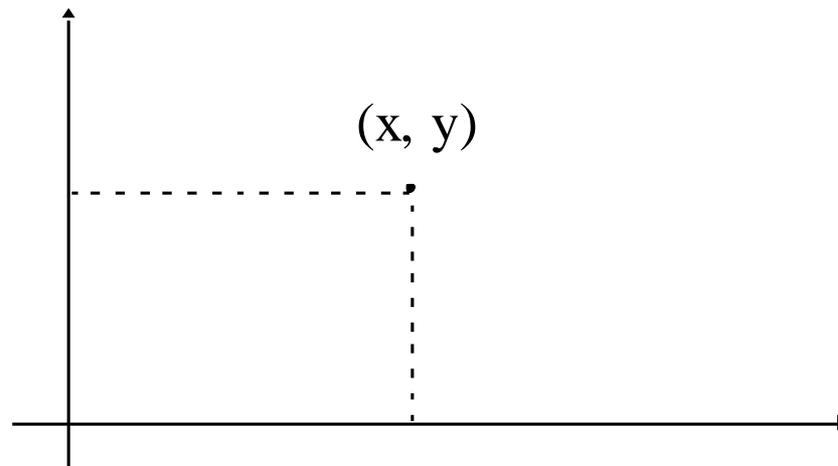
```
t_idade Minha_Idade;
```

**Tipo da Variável**

**Nome da Variável**

## Estruturas

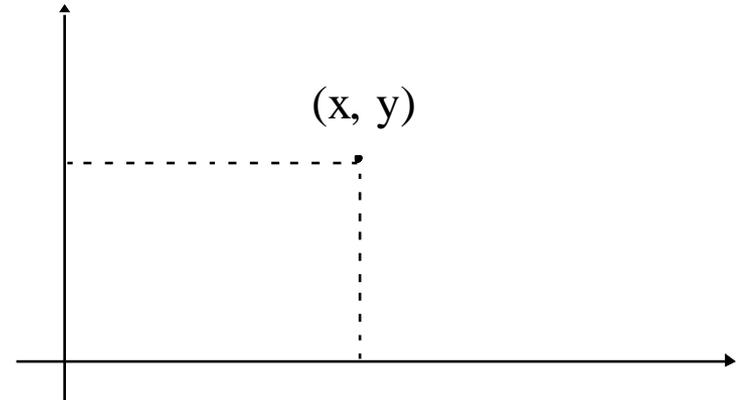
- **Structs** são coleções de dados heterogêneos agrupados em um mesmo elemento de dados
- Ex: armazenar as coordenadas (x,y) de um ponto:



## Estruturas: Struct

- Declaração:

```
struct {  
    int x;  
    int y;  
} p1, p2;
```



- A estrutura contém dois inteiros,  $x$  e  $y$
- Neste caso, a estrutura foi definida e com ela duas variáveis,  $p1$  e  $p2$ , foram declaradas (cada um contendo duas coordenadas).

## Declaração: Struct

- Formato da declaração:

```
struct nome_da_estrutura {  
    tipo_1 dado_1;  
    tipo_2 dado_2;  
    ...  
    tipo_n dado_n;  
} lista_de_variaveis;
```

- A estrutura pode agrupar um número arbitrário de dados de tipos diferentes
- Pode-se nomear a estrutura para aumentar a facilidade em referenciá-la

## Nomeando uma Estrutura

```
struct {  
    int x;  
    int y;  
} p1;  
  
struct {  
    int x;  
    int y;  
} p2;
```

Para evitar  
a repetição 

```
struct ponto {  
    int x;  
    int y;  
};  
  
struct ponto p1, p2;
```

**struct ponto** define um *novo tipo de dado*

Pode-se definir novas variáveis do tipo **ponto**

## Estruturas: Struct

- Assim como as demais variáveis compostas, temos de ter a capacidade de manipular seus elementos (os **campos**) individualmente.
- Acessando os dados:

*nome\_variavel\_struct.campos*

- Ex: `p1.x = 10;`      */\*atribuição\*/*  
    `p2.y = 15;`  
    `if ( (p1.x >= p2.x) && (p1.y >= p2.y) ... )`

## Atribuição de Estruturas

- Tal qual a demais variáveis, é possível inicializar uma estrutura no momento de sua declaração:

```
struct ponto p1 = { 220, 110 };
```

- A operação de atribuição entre estruturas do mesmo tipo pode acontecer de maneira direta:

```
struct ponto p1 = { 220, 110 };
```

```
struct ponto p2;
```

```
p2 = p1;
```

```
/* p2.x = p1.x e p2.y = p1.y */
```

- Note que os campos correspondentes das estruturas são automaticamente copiados do destino para a fonte

## Estruturas: exemplo

```
struct coord {
```

```
double Lat;      /* Latitude */  
double Long;    /* Longitude */  
int Orientacao; /* Direção em graus */  
};
```

Campos da Estrutura

Nome da Estrutura

```
struct coord
```

```
V1, V2, V3;
```

```
V1.Lat = 3.25;  
V1.Long = 27.65;  
V1.Orientacao = 35;
```

Nomes  
das  
Variáveis

```
V2 = V1;
```

## Espaço alocado para uma Estrutura

```
struct aluno {  
    char nome[20];      /* array 20 bytes */  
    int idade;         /* 4 bytes */  
    char matricula[8]; /* array 8 bytes */  
};
```

```
struct aluno al;  
strcpy( al.nome, "Fulano");  
al.idade = 21;  
strcpy( al.matricula, "1234567");
```

struct aluno al

"Assis"
21
"1234567"

## Composição de Estruturas

- De fato, as *structs* definem novos tipos de dados (tipos do usuário) e portanto podem conter campos de qualquer tipo, quer sejam tipos básicos ou outros tipos definidos pelo usuário.
- Inclusive, suportam a definição de estruturas compostas de outras estruturas!
  - Um retângulo poderia ser definido por dois pontos: o superior esquerdo e o inferior direito.

## Composição de Estruturas

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

```
struct retangulo {  
    struct ponto cantoSupEsq;  
    struct ponto cantoInfDir;  
};  
struct retangulo r = { { 10, 20 }, { 30 , 40 } };
```

Acesso aos dados:

```
r.cantoInfDir.x    = 0;  
r.cantoSupEsq.x   += 10;  
r.cantoSupEsq.y   -= 10;
```

## Criando novos tipos de dados: TYPEDEF e STRUCT

- Comando STRUCT: Criando tipos compostos (registros)

```
struct data {  
    int dia;  
    int mes;  
    int ano;  
};
```

```
struct data data_nasc;
```

```
data_nasc.dia = 1;  
data_nasc.mes = 1;  
data_nasc.ano = 2000;
```

```
typedef struct {  
    long int nro_funcionario;  
    double salario;  
    data data_contratacao;  
} reg_funcionario;  
  
reg_funcionario diretor_dept_pessoal;  
  
diretor_dept_pessoal.nro_funcionario = 1234567;  
diretor_dept_pessoal.salario = 9999.99;  
diretor_dept_pessoal.data_contratacao.ano = 1999;
```

### Sintaxe da Declaração:

```
struct nome_reg { ... };
```

ou

```
struct { ... } nome_reg;
```

```
typedef struct { ... } nome_novo_tipo;
```

ou

```
typedef struct nome_reg nome_novo_tipo;
```

## Criando novos tipos de dados: TYPEDEF e STRUCT

- Comando STRUCT: Criando tipos compostos (registros)

```
struct data {  
    int dia;  
    int mes;  
    int ano;  
};
```

```
struct data data_nasc;
```

```
data_nasc.dia = 1;  
data_nasc.mes = 1;  
data_nasc.ano = 2000;
```

```
typedef  
struct {  
    int dia;  
    int mes;  
    int ano;  
} data;
```

```
data data_nasc;
```

```
data_nasc.dia = 1;  
data_nasc.mes = 1;  
data_nasc.ano = 2000;
```

### Sintaxe da Declaração:

```
struct nome_reg { ... };
```

ou

```
struct { ... } nome_reg;
```

```
typedef struct { ... } nome_novo_tipo;
```

ou

```
typedef struct nome_reg nome_novo_tipo;
```

## Criando novos tipos de dados: TYPEDEF e STRUCT

- Comando STRUCT: Criando tipos compostos (registros)

```
typedef
struct {
    int dia;
    int mes;
    int ano;
} data;
```

```
data data_nasc;
```

```
data_nasc.dia = 1;
data_nasc.mes = 1;
data_nasc.ano = 2000;
```

```
typedef struct {
    long int nro_funcionario;
    double salario;
    data data_contratacao;
} reg_funcionario;

reg_funcionario diretor_dept_pessoal;

diretor_dept_pessoal.nro_funcionario = 1234567;
diretor_dept_pessoal.salario = 9999.99;
diretor_dept_pessoal.data_contratacao.ano = 1999;
```

### Sintaxe da Declaração:

```
struct nome_reg { ... }; typedef struct { ... } nome_novo_tipo;
ou
struct { ... } nome_reg; typedef struct nome_reg nome_novo_tipo;
```

### Sintaxe do Uso:

```
registro.campo = dado;
variavel = registro.campo
```

## USO PRÁTICO

## **TYPEDEF e STRUCT**

## Criando novos tipos de dados: **TYPEDEF** e **STRUCT**

### Usando na prática...

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

```
typedef struct {
    int dia;
    int mes;
    int ano;
} t_data;
```

```
typedef struct {
    char nome[30];
    t_data dnasc;
    t_data ingresso;
    int cod_depto;
    double salario;
} t_reg_func;
```

```
t_reg_func Chefe;
t_reg_func Secretaria;
```

```
main()
{
    /* Dados do Chefe */
    printf("Nome: ");
    scanf ("%s",Chefe.nome);
    printf("Data de Nascimento: ");
    scanf ("%d",&(Chefe.dnasc.dia));
    scanf ("%d",&(Chefe.dnasc.mes));
    scanf ("%d",&(Chefe.dnasc.ano));
    printf("Codigo do Departamento: ");
    scanf ("%d",&(Chefe.cod_depto));
    printf("Salario: ");
    scanf ("%lf",&(Chefe.salario));
    printf("\n");

    getch();

    /* Exibe na tela os dados */
    printf("Nome do Chefe: %s\n",Chefe.nome);
    printf(" Data de Nascimento: %d/%d/%d\n",
    Chefe.dnasc.dia, Chefe.dnasc.mes, Chefe.dnasc.ano);
    printf(" Salario: %.2lf\n", Chefe.salario);
    getch();
}
```

## Criando novos tipos de dados: **TYPEDEF** e **STRUCT**

### Usando na prática...

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

```
typedef struct {
    int dia;
    int mes;
    int ano;
} t_data;
```

```
typedef struct {
    char nome[30];
    t_data dnasc;
    t_data ingresso;
    int cod_depto;
    double salario;
} t_reg_func;
```

```
t_reg_func Chefe;
t_reg_func Secretaria;
```

```
main()
{
    /* Dados da Secretaria */
    printf("Nome: ");
    scanf ("%s",Secretaria.nome);
    printf("Data de Nascimento: ");
    scanf ("%d",&(Secretaria.dnasc.dia));
    scanf ("%d",&(Secretaria.dnasc.mes));
    scanf ("%d",&(Secretaria.dnasc.ano));
    printf("Codigo do Departamento: ");
    scanf ("%d",&(Secretaria.cod_depto));
    printf("Salario: ");
    scanf ("%lf",&(Secretaria.salario));
    printf("\n");
    getch();

    /* Exibe na tela os dados */
    printf("Nome da Secretaria: %s\n",Secretaria.nome);
    printf("  Data de Nascimento: %d/%d/%d\n",
        Secretaria.dnasc.dia, Secretaria.dnasc.mes,
        Secretaria.dnasc.ano);
    printf("  Salario: %.2lf\n", Secretaria.salario);
    getch();
}
```

## Vetores de Registros de Dados - EXEMPLO

```
typedef struct {
    int dia, mes, ano;
    double temp_min, temp_max;
} t_vetor_dados;

t_vetor_dados Medidas[100];

main( )
{ int cont;

  for (cont = 0; cont < 100; cont++)
  {
    printf (“Dia : “); scanf (“%d”, & Medidas[cont].dia );
    printf (“Mes: “); scanf (“%d”, & Medidas[cont].mes );
    printf (“Ano: “); scanf (“%d”, & Medidas[cont].ano );
    printf (“Temp. Minima: “); scanf (“%lf”, & Medidas[cont].temp_min );
    printf (“Temp. Maxima: “); scanf (“%lf”, & Medidas[cont].temp_max );
  }
}
```



## INFORMAÇÕES SOBRE A DISCIPLINA

**USP - Universidade de São Paulo - São Carlos, SP**  
**ICMC - Instituto de Ciências Matemáticas e de Computação**  
**SSC - Departamento de Sistemas de Computação**

**Prof. Fernando Santos OSÓRIO**

**Web institucional: <http://www.icmc.usp.br/>**

**Página pessoal: <http://www.icmc.usp.br/~fosorio/>**

**Página do Grupo de Pesquisa: <http://www.lrm.icmc.usp.br/>**

**E-mail: fosorio [at] icmc. usp. br ou fosorio [at] gmail. com**

**Disciplina de Introdução a Computação – Eng. Ambiental**

**WIKI - [http://wiki.icmc.usp.br/index.php/SSC-301-2013\(fosorio\)](http://wiki.icmc.usp.br/index.php/SSC-301-2013(fosorio))**

- > Programa, Material de Aulas, Critérios de Avaliação,**
- > Trabalhos Práticos, Datas das Provas, Notas**