Departamento de Sistemas de Computação
Universidade de São Paulo
SSC 0601 – Laboratório de Introdução à Ciência de Computação I

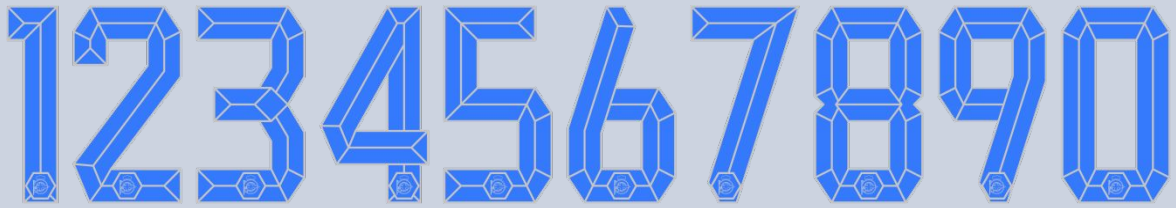
Aula 6
Estruturas

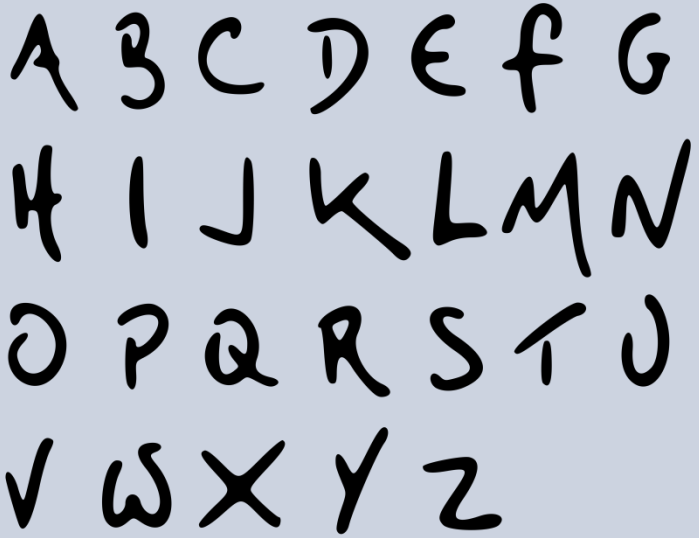
Responsável
Prof. Armando Toda (armando.toda@usp.br)

Estruturas

- **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





Vetores:

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	?	?	?

Matrizes:

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:

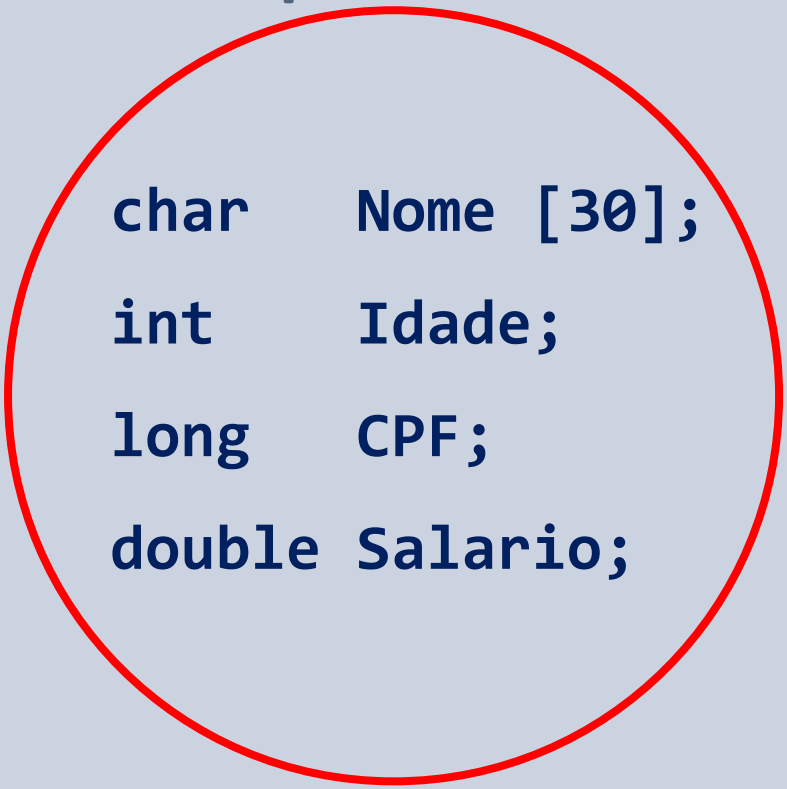
```
int num [5] =  { 1, 2, 3, 4, 5 };

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

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

Vetores e Matrizes: Estruturas compostas por dados homogêneos

O que fazer quando precisamos armazenar na memória

Informações de diferentes tipo ?!? Nome, Idade, CPF, Salário, etc.

pessoa



Estruturas

- ***Structs*** são coleções de dados heterogêneos agrupados em um mesmo elemento de dados

- Ex: armazenar os dados de uma pessoa Nome, Idade, CPF, Salário.

Estruturas: Struct

- Declaração:

```
struct pessoa{
    char   Nome [30];
    int    Idade;
    long   CPF;
    double Salario;
} pessoa1, pessoa2;
```

  - Neste caso, a estrutura foi definida e com ela duas variáveis, *pessoa1* e *pessoa2,* foram declaradas (cada uma contendo uma cadeia de char, um int, um long e um double).

Declaração: Struct

- Formato da declaração:

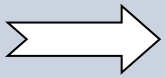
```
struct nome_da_estrutura {
        tipo_1 dado_1;
        tipo_2 dado_2;
        ...
        tipo_n dado_n;
} lista_de_variaveis;
```

- 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 **struct 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*.campo**

- Ex: **p1.x** = 10; */*atribuição */*

  p2.y = 15;

  if ((**p1.x** >= **p2.x**) && (**p1.y** >= **p2.y**) ...)

Atribuição de Estruturas

- É possível inicializar uma estrutura no momento de sua declaração:

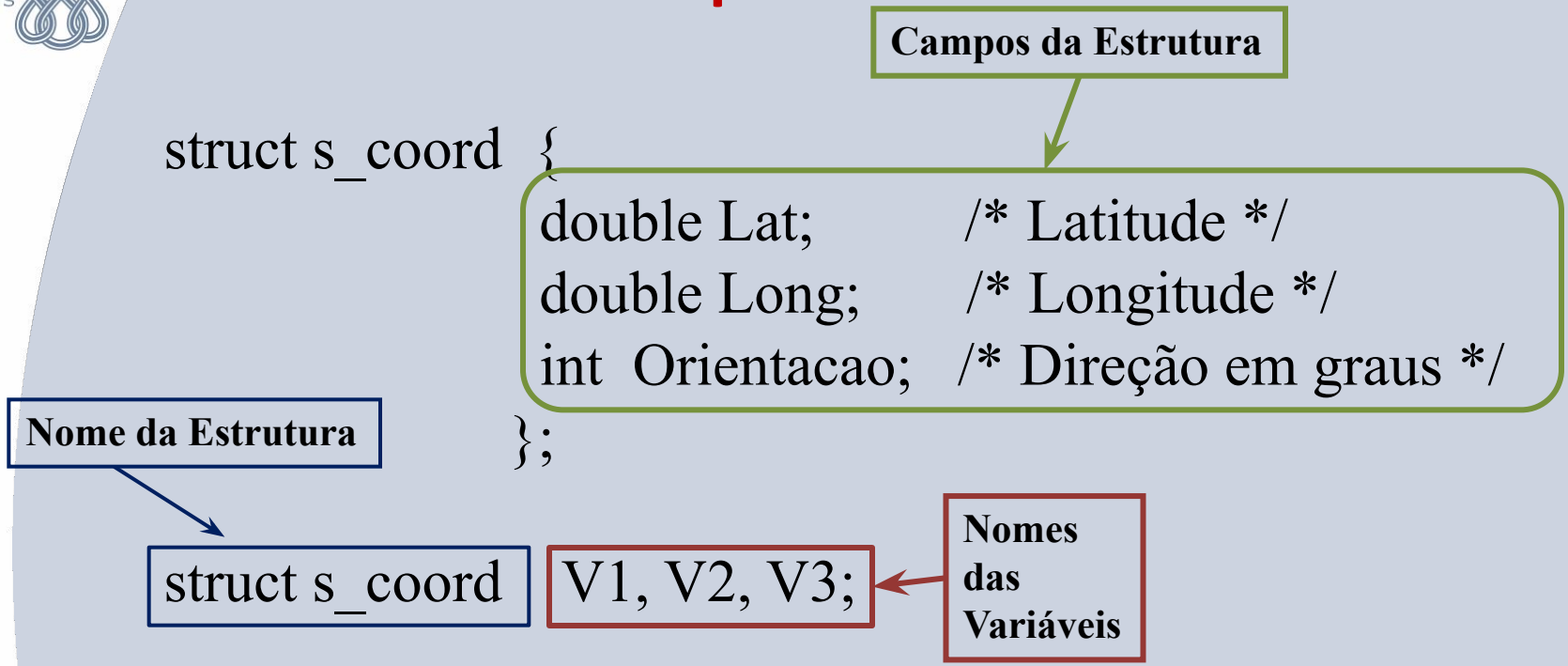
```
struct ponto p1 = { 220, 110 };
```

- 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




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

```
V3.Lat = 3.25;
V3.Long = 27.65;
V3.Orientacao =35;
```

```
V2 = V1;
```

Estruturas: exemplo

```
struct s_aluno {
    char nome[20];
    int    idade;
    char matricula[8];
};

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

“Fulano“
21
“1234567"

Estruturas: exemplo

```
#define LEN 50
struct endereco {
      char rua[LEN];
      char cidade_estado_cep[LEN];
} ender ;
printf("\n Entre rua: ");
gets(ender.rua);
printf("\n Entre cidade/estado/cep: ");
gets(ender.cidade_estado_cep);

printf("\t %s\n", ender.rua);
printf("\t %s\n", ender.cidade_estado_cep);
```

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 } };
```

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

Criando novos tipos de dados: TYPEDEF e STRUCT

```
struct data{
      int dia;
      int mês;
      int ano;

};

struct data dataNascim;
```

```
typedef struct {
      int dia;
      int mês;
      int ano;

} data ;

data dataNascim;
```

Criando novos tipos de dados: TYPEDEF e STRUCT

```
struct {
      int dia;
      int mês;
      int ano;

}dataNascim;

typedef struct dataNascim data;
```

Sintaxe da Declaração:

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

Criando novos tipos de dados: TYPEDEF

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

Exemplo:

```
typedef   double  t_nota;

main()
{
   t_nota  p1,p2;
   t_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();
}
```

Vetores de Registros de Dados - EXEMPLO

- Crie um tipo de dado t_vetor_dados que armazene o dia, mês, ano, tempo mínimo e tempo máximo.
- Declare um vetor do tipo t_vetor_dados com 365 posições.
- Escreva um programa para preencher todos os campos das 365 posições do vetor.

Vetores de Registros de Dados - EXEMPLO

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

t_vetor_dados   Medidas[365];

main( )
{   int cont;

   for (cont = 0; cont < 365;  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 );
   }
}
```

Exercício

1. Escreva um trecho de código em "C" para fazer a criação dos novos tipos de dados conforme solicitado abaixo:

A) **Horário**: composto de hora, minutos e segundos

B) **Data**: composto de dia, mês e ano

C) **Compromisso**: composto de uma data, horário e local (palavra de no máximo 30 caracteres).

Faça a leitura (digitado pelo usuário) e impressão dos dados.

2. Crie uma estrutura que armazena as coordenadas de um ponto cartesiano (x, y). Utilize essa estrutura para fazer um programa que calcula a equação da reta y = ax+b através de dois pontos cartesianos p1 e p2 distintos (Dica: resolva a equação antes de pensar na implementação).