



Disciplina:

Programação Computacional

Prof. Fernando Rodrigues

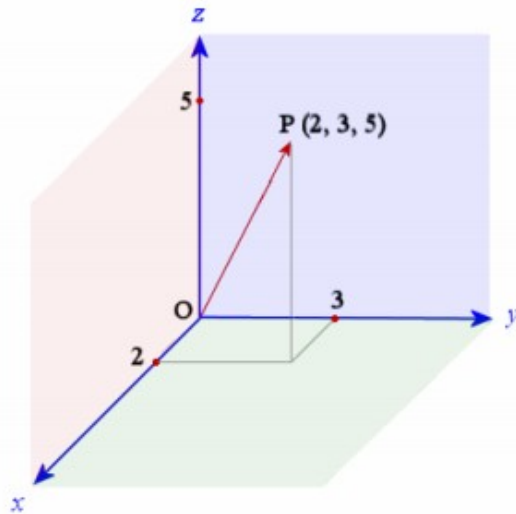
e-mail: fernandorodrigues@sobral.ufc.br



Aula 11: Programação em C

- ❖ Tipos de dados definidos pelo programador
- ❖ Estruturas de dados heterogêneas
 - Tipos estruturados (structs)
 - Uniões (unions)
 - Enumerações (enums)
 - Typedef

Estruturas de dados



$$A = \begin{bmatrix} -2.0 & 1.5 & 0.0 & 4.7 \\ 0.0 & 3.1 & 1.0 & 0.5 \\ -5.0 & -3.2 & 0.5 & -0.1 \\ 0.0 & 1.0 & -9.5 & 2.0 \end{bmatrix}$$

```
char name[80] = "Jean-Luc Picard";  
char rank[20] = "Captain";
```



Tipos estruturados

Tipos estruturados são entidades de programação que permitem o armazenamento de informação de diferentes tipos em um único local, denominado **estrutura**.

Uma estrutura é formada por **campos**. Cada campo armazena parte da informação.

Cada campo deve ser declarado individualmente. Um campo pode ser de um tipo primitivo (**char**, **int**, **float**, **double**) ou pode ser uma estrutura definida previamente.

Ao final de sua declaração, a estrutura deve receber um nome, definindo assim um novo tipo de dado.

Estruturas

- ▶ Uma estrutura é uma coleção de variáveis referenciadas por um nome, fornecendo uma maneira conveniente de ter informações relacionadas agrupadas.
- ▶ Uma *definição de estrutura* forma um modelo que pode ser usado para criar variáveis de estrutura.
- ▶ As variáveis que compreendem a estrutura são chamadas de membros da estrutura (também conhecidos como *elementos* ou *campos* da estrutura).

Estruturas

- ▶ Geralmente todos os elementos da estrutura são relacionados

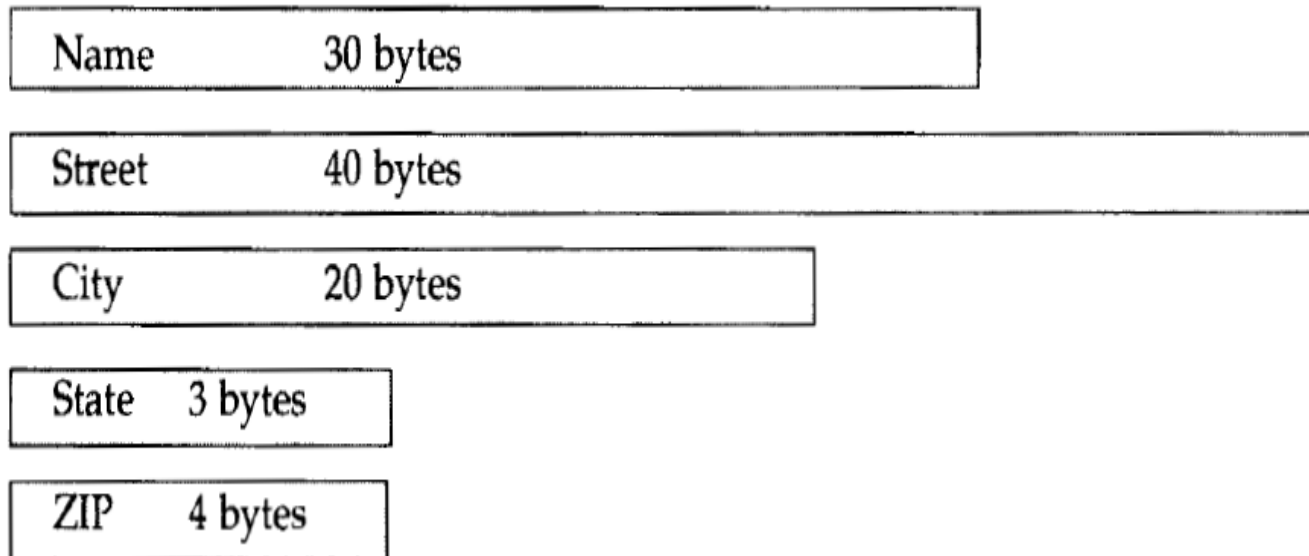
```
struct addr
{
    char name[30];
    char street[40];
    char city[20];
    char state[3];
    unsigned long int zip;
};
```

- ▶ Até o ponto acima, nenhuma variável foi declarada.

```
struct addr addr_info;
```

Estruturas

- ▶ Quando uma variável de estrutura é declarada, o compilador C automaticamente aloca memória suficiente para acomodar todos os seus membros.



Estruturas (structs)

► Forma geral de uma estrutura

```
struct identificador {  
    tipo nome_da_variável;  
    tipo nome_da_variável;  
    tipo nome_da_variável;  
    .  
    .  
    .  
};
```

- ▶ **typedef**: define um “apelido” ou sinônimo para um tipo de dados

Tipos estruturados

Estruturas permitem modelar entidades constituídas por mais de uma informação, como por exemplo:

- Ponto no R^3 : $P = (x, y, z)$.

```
typedef struct {  
    float x;  
    float y;  
    float z;  
} ponto;
```

- Dados de aluno.

```
typedef struct {  
    char nome[80];  
    int matricula;  
    char curso[30];  
    int credits;  
    float CR;  
} aluno;
```

- Número complexo: $z = (Re, Im)$.

```
typedef struct {  
    float Re;  
    float Im;  
} complexo;
```


Trabalhando com estruturas

```
typedef struct {  
    char  marca[20];  
    char  modelo[20];  
    int   ano;  
    float km;  
    int   portas;  
    char  cor[20];  
} automovel;
```

Uma vez definida uma estrutura, ela pode ser usada como um novo tipo para a declaração de variáveis.

```
automovel A;           // A é uma variável do tipo automóvel
```

O acesso aos campos de uma estrutura é feita utilizando o operador ".".

```
A.ano = 2009;           // definir o ano do carro A  
A.km = 17890.3;         // definir a quilometragem do carro A  
scanf("%d", &A.portas); // ler o número de portas do carro A  
gets(A.cor);            // ler a cor do carro A
```

Vetor de estruturas

Armazenar em um vetor os dados dos alunos de uma classe.

```
#include <math.h>
#define N 50

typedef struct {
    char nome[80];
    int matricula;
    char curso[30];
} aluno;

int main()
{
    aluno classe[N];
    int i;

    for (i = 0; i < N; i++)
    {
        printf("Nome do aluno %d: ", i); gets(classe[i].nome);
        printf("Numero do aluno %d: ", i); scanf("%d", &classe[i].matricula);
        printf("Curso do aluno %d: ", i); gets(classe[i].curso);
    }
    ...
}
```

Atribuição de estruturas

- ▶ Se o compilador C é compatível com o padrão ANSI, a informação contida em uma estrutura pode ser atribuída a outra estrutura do mesmo tipo.
- ▶ Ou seja, em vez de atribuir os valores de todos os elementos separadamente, pode-se empregar um único comando de atribuição.

Atribuição de estruturas

```
#include <stdio.h>

int main(void)
{
    struct {
        int a;
        int b;
    } x, y;

    x.a = 10;

    y = x;  /* assign one structure to another */

    printf("%d", y.a);

    return 0;
}
```

Declaração de estruturas – 4 Formas

```
struct <identificador>{  
    tipo nome_campo;  
    ...  
};
```

```
struct <identificador>{  
    tipo nome_campo;  
    ...  
}<nome_variável>;
```

```
struct {  
    tipo nome_campo;  
    ...  
}<nome_variável>;  
  
typedef struct {  
    tipo nome_campo;  
    ...  
}<identificador>;
```

Exercícios I

- 1) Implemente um programa que leia o nome, a idade e o endereço de uma pessoa e armazene esses dados em uma estrutura. Em seguida, imprima na tela os dados da estrutura lida.
 - 2) Crie uma estrutura para representar as coordenadas de um ponto no plano (posições X e Y). Em seguida, declare e leia do teclado um ponto e exiba a distância dele até a origem das coordenadas, isto é, a posição (0,0).
 - 3) Crie uma estrutura representando um aluno de uma disciplina. Essa estrutura deve conter o número de matrícula do aluno, seu nome e as notas de três provas. Agora, escreva um programa que leia os dados de cinco alunos e os armazena nessa estrutura. Em seguida, exiba o nome e as notas do aluno que possui a maior média geral dentre os cinco.
-



Unões: union

- ▶ Uma união pode ser vista como uma lista de variáveis, e cada uma delas pode ter qualquer tipo.
- ▶ A ideia básica por trás da união é similar à da estrutura: criar apenas um tipo de dado que contenha vários membros, que nada mais são do que outras variáveis.
- ▶ Declarando uma união:
A forma geral da definição de união utiliza o comando “*union*”:

```
union nome_union  
{  
    tipo1 campo1;  
    tipo2 campo2;  
    ...  
    tipoN campoN;  
};
```

Unões x Estruturas

- ▶ Diferentemente das estruturas, todos os elementos contidos na união ocupam o mesmo espaço físico na memória.
- ▶ Uma estrutura reserva espaço de memória para todos os seus elementos, enquanto uma *union* reserva espaço de memória para o seu maior elemento e compartilha essa memória com os demais elementos.
- ▶ Numa *struct* é alocado espaço suficiente para armazenar todos os seus elementos, enquanto numa *union* é alocado espaço para armazenar o maior dos elementos que a compõem.

Union

- ▶ Tome como exemplo a seguinte declaração de união:

union *exemplo*

```
{  
    short int x;  
    unsigned char c;  
};
```

- ▶ Essa união possui o nome *exemplo* e duas variáveis: *x*, do tipo `short int` (dois bytes), e *c*, do tipo `unsigned char` (um byte). Assim, uma variável declarada desse tipo (**union** *exemplo* *t*;) ocupará dois bytes na memória, que é o tamanho do maior dos elementos da união (**short int**).
- ▶ Isso acontece porque o espaço de memória é compartilhado. Portanto, é de total responsabilidade do programador saber qual dado foi mais recentemente armazenado em uma união.



Em uma união, apenas um membro pode ser armazenado de cada vez.

Union: Exemplo



Como todos os elementos de uma união se referem a um mesmo local na memória, a modificação de um dos elementos afetará o valor de todos os demais. Numa união, é impossível armazenar valores independentes.

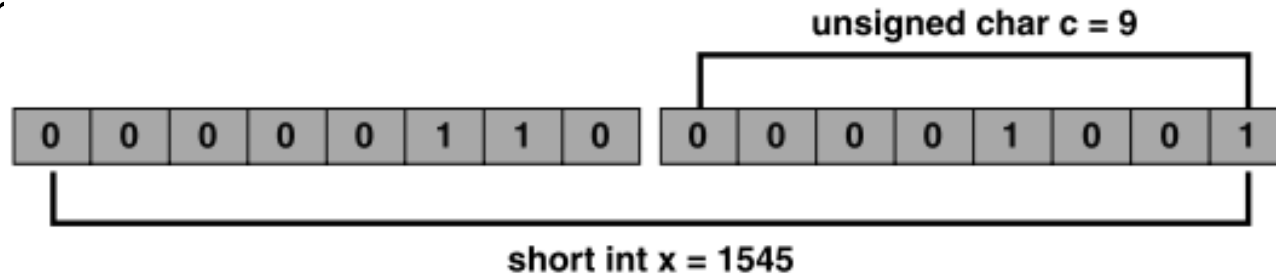
```
01  #include <stdio.h>
02  #include <stdlib.h>
03  union tipo{
04      short int x;
05      unsigned char c;
06  };
07  int main(){
08      union tipo t;
09      t.x = 1545;
10      printf("x = %d\n",t.x);
11      printf("c = %d\n",t.c);
12      t.c = 69;
13      printf("x = %d\n",t.x);
14      printf("c = %d\n",t.c);
15      system("pause");
16      return 0;
17  }
```

Saída

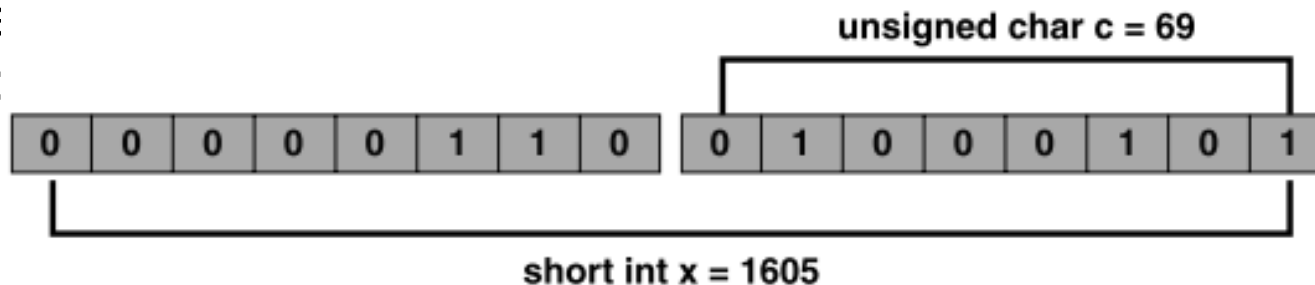
```
x = 1545
c = 9
x = 1605
c = 69
```

Union: Exemplo

- ▶ Nesse exemplo, a variável `x` é do tipo `short int` e ocupa 16 bits (dois bytes) de memória. Já a variável `c` é do tipo `unsigned char` e ocupa os oito primeiros bits (um byte) de `x`. Quando atribuímos o valor 1545 à variável `x`, a variável `c` recebe a porção de `x` equivalente ao número



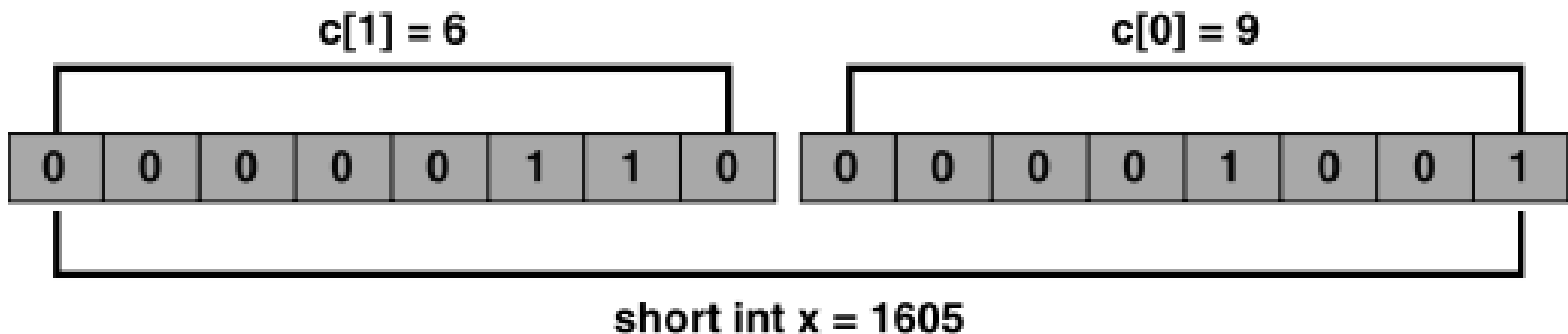
- ▶ Do mesmo modo, se modificarmos o valor da variável `c` para 69, estaremos modificando o valor de `x` para 1605:



Union

- ▶ Dada a seguinte declaração de união:

```
union tipo{  
    short int x;  
    unsigned char c[2];  
};
```
- ▶ Sabemos que a variável x ocupa dois bytes na memória. Como cada posição da variável c ocupa apenas um byte, podemos acessar facilmente cada uma das partes da variável x sem precisar recorrer a operações de manipulação de bits (operações lógicas e de



Union x Struct (Tamanho em memória)

```
main() {  
    union info {  
        char c;  
        int i;  
        float f;  
    };  
    struct dado {  
        char c;  
        int i;  
        float f;  
    };  
    printf( "Tamanho dos dados na union: %d\n", sizeof( union info ) );  
    printf( "Tamanho dos dados na struct: %d", sizeof( struct dado ) );  
}
```

Enumerações: enum

- ▶ Uma enumeração pode ser vista como uma lista de constantes, em que cada constante possui um nome significativo.
- ▶ A ideia básica por trás da enumeração é criar apenas um tipo de dado que contenha várias constantes, e uma variável desse tipo só poderá receber como valor uma dessas constantes.

- ▶ Declarando uma enumeração

A forma geral da definição de uma enumeração utiliza o comando `enum`:

```
enum nome_enum { lista_de_identificadores };
```

- ▶ Nessa declaração, `lista_de_identificadores` é uma lista de palavras separadas por vírgula e delimitadas pelo operador de chaves (`{ }`).

Enumerações: enum

- ▶ Por exemplo, o comando:

```
enum semana {Domingo, Segunda, Terca, Quarta, Quinta, Sexta, Sabado };
```

- ▶ cria uma enumeração de nome semana, cujos valores constantes são os nomes dos dias da semana.
- ▶ Na definição da enumeração é possível definir algumas variáveis desse tipo. Para isso, basta colocar os nomes das variáveis declaradas após o comando de fechar chaves (}) da enumeração e antes do ponto e vírgula (;):

```
enum semana {Domingo, Segunda, Terca, Quarta, Quinta, Sexta, Sabado}  
s1, s2;
```

- ▶ Uma vez definida a enumeração, uma variável pode ser declarada de modo similar aos tipos já existentes: `enum semana s;`
- ▶ e inicializada como qualquer outra variável, usando, para isso, uma das constantes da enumeração: `s = Segunda;`

Enum: Simulando o tipo boolean em C

- ▶ Para simular o tipo booleano vamos utilizar enumerações e a palavra reservada typedef. Primeiro, criaremos uma enumeração chamada boolean que assuma os valores true e false. Em seguida, utilizaremos o typedef para permitir a declaração de variáveis do tipo enum boolean como se fossem um tipo primitivo qualquer.

```
1 // Criando a enumeração:
2 enum boolean {
3     true = 1, false = 0
4 };
5 // Permitindo a sua declaração como um tipo qualquer:
6 typedef enum boolean bool;
7
8 // Agora podemos escrever e compilar os códigos como:
9 int main () {
10
11     bool b = true;
12     if (b) {
13         b = false;
14     }
15
16     return 0;
17 }
```


Exercícios II

1) Crie uma enumeração representando os dias da semana. Agora, escreva um programa que leia um valor inteiro do teclado e exiba o dia da semana correspondente.

2) Crie uma enumeração representando os meses do ano. Agora, escreva um programa que leia um valor inteiro do teclado e exiba o nome do mês correspondente e quantos dias ele possui.

3) Crie uma enumeração para representar o estado civil (solteiro, casado, separado, viúvo), usando typedef. Agora, escreva um programa que defina uma struct pessoa, que tenha os campos nome, idade e peso, além de um campo do tipo “estado civil”. Defina um array de 3 “pessoas”, povoe este array e exiba em tela o valor dos 3 elementos.



Exercícios III

1) Crie uma união contendo dois tipos básicos diferentes. Agora, escreva um programa que inicialize um dos tipos dessa união e exiba em tela o valor do outro tipo.



Fim