UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

# Fundamentos de Programação

CP41F

Declaração de tipos (typedef). Tipos enumerados (enum). Registros (struct). Uniões (union).

**Aula 18**
**Prof. Daniel Cavalcanti Jeronymo**

**Universidade Tecnológica Federal do Paraná (UTFPR)**
Engenharia de Computação – 1º Período
2016.1

# Plano de Aula

- Declaração de tipos (typedef).
- Tipos enumerados (enum).
- Registros (struct).
- Uniões (union).

# Declaração de tipos

- Definição de tipos pelo usuário
  - Palavra chave **typedef**
  - Sintaxe:

    **typedef** velho_tipo novo_tipo**;**

# Declaração de tipos

- Exemplo

```
typedef unsigned int NotaAluno;
typedef char NomeAluno[20];

NotaAluno A1 = 0;
NomeAluno fulano = "Michael Jah Elvis";

printf("%s: %d", fulano, A1);
```

# Declaração de tipos

- Vantagens
  - Torna o código mais portável
  - Reduz a **verbosidade** do código
- Desvantagem
  - Simplificação excessiva leva a **obfuscação**

# Tipos enumerados

- Define uma lista de nomes valorados
  - Sintaxe:

```
enum tag
{
        nome1,
        nome2,
        …
};

enum tag variavel = nome1;
```

Em C, tags são diferentes de nomes.

Neste exemplo a seguinte declaração é válida:

```
enum tag tag = nome1;
```

Tags são utilizadas em: enum, struct e union.

# Tipos enumerados

- Valores podem ser definidos manualmente
  - Sintaxe:

```
enum tag
{
        nome1=10,
        nome2,
        ...
};

enum tag variavel = nome1;
```

# Tipos enumerados

- Valores podem ser definidos manualmente
  - Sintaxe:

```
enum tag
{
        nome1=10,
        nome2=20,
        ...
};

enum tag variavel = nome1;
```

# Tipos enumerados

- Exemplo – definir um conjunto de cores
  - Utilizando definições:

```
#define BRANCO 0
#define AZUL 1
int cor = AZUL;
```

  - Utilizando enumeração:

```
enum Cores
{
   BRANCO,
   AZUL
};
enum Cores cor = AZUL;
```

# Registros

- Estrutura fundamental de dado
  - Coleção heterogênea de valores
  - Valores armazenados em **campos**
  - **Nomes** são associados aos campos

# Registros

- Definição de um registro
  - Sintaxe

```
struct tag
{
        declaração de campo 1;
        declaração de campo 2;
        ...
};

struct tag variavel;
```

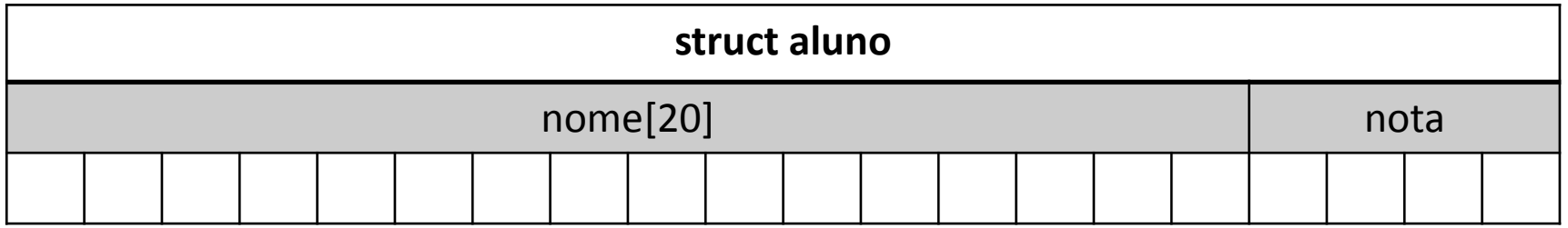
# Registros

- Exemplo – Definindo uma estrutura para registrar um aluno

```
struct aluno
{
   char nome[20];
   int nota;
};

struct aluno michael = {"Michael Jah Elvis", 0};

printf("%s: %d\n", michael.nome, michael.nota);
```



# Registros

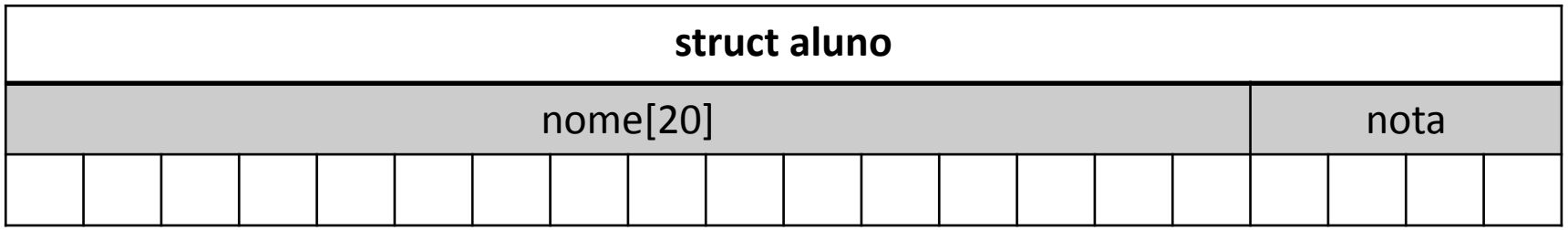
- Exemplo – Acesso por ponteiros

```
struct aluno
{
   char nome[20];
   int nota;
};

struct aluno * michael = malloc(sizeof(struct aluno));

strcpy(michael->nome, "Michael Jah Elvis");
michael->nota = 0;

printf("%s: %d\n", michael->nome, michael->nota);
```



# Registros

- Cuidado!


- O tamanho de uma estrutura é alinhado na memória

- Verifique o sizeof da estrutura anterior, altere o valor de nome para 19 caracteres e verifique novamente

# Registros

- Exemplo – Criar e acessar uma estrutura para carros

```
struct Carro
{
   unsigned int ano;
   const char * modelo;
   void (*imprime)(struct Carro);
};
```

# Registros

- Exemplo – Criar e acessar uma estrutura para carros (funções para exibir dados)

```
void imprimeTotal(struct Carro carro)
{
   printf("Ano: %u\nModelo: %s\n", carro.ano, carro.modelo);
}

void imprimeModelo(struct Carro carro)
{
   printf("Modelo: %s\n", carro.modelo);
}
```

# Registros

- Exemplo – Criar e acessar uma estrutura para carros (criando um carro)

```
struct Carro novoCarro(unsigned int ano, const char *modelo)
{
   struct Carro carro;
   carro.ano = ano;
   carro.modelo = modelo;
   carro.imprime = imprimeTotal;
   return carro;
}
```

# Registros

- Exemplo – Criar e acessar uma estrutura para carros (juntando tudo!)

```
int main()
{
   struct Carro meuCarro = novoCarro(1990, "Opalao");

   meuCarro.imprime(meuCarro);

   return 0;
}
```

# Registros

- Passagem de estruturas como parâmetros para funções
  - Por valor
    - Copia a estrutura **inteira**, consome mais espaço na pilha, mais lento
  - Por referência
    - Copia o **ponteiro** para a estrutura, consome menos espaço na pilha e mais rápido

# Uniões

- Permitem tratar um bloco de memória como diferentes tipos de dados

- Sintaxe

```
union tag
{
        declaração de campo 1;
        declaração de campo 2;
        ...
};

union tag uma_uniao;
```

# Uniões

- Exemplo – Acessar os bits individuais de um byte

```
union byte
{
   unsigned char valor;
   struct
   {
      unsigned int b0 : 1:
      unsigned int b1 : 1:
      unsigned int b2 : 1:
      unsigned int b3 : 1:
      unsigned int b4 : 1:
      unsigned int b5 : 1:
      unsigned int b6 : 1:
      unsigned int b7 : 1;
   } bits;
};

union byte exemplo;

exemplo.valor = 0xF0;

printf("%d %d\n", exemplo.bits.b0, exemplo.bits.b7);
```

Campo de bits (bit field)

Cuidado ao usar e assumir que é mais rápido ou consome menos memória.

Não é padronizado, depende da implementação do compilador.

# Prática

- Como utilizar typedefs para reduzir declarações de estruturas, enumerações ou uniões?

# Prática

- Crie uma biblioteca consistindo de ponto.h e ponto.c

- Defina uma estrutura Ponto que represente pontos no plano Cartesiano.

- Escreva funções para medir a distância entre dois pontos e comparar se dois pontos são iguais.

# Prática

- Crie uma biblioteca consistindo de retangulo.h e retangulo.c

- Defina uma estrutura **Retangulo** utilizando a definição anterior de **Ponto**

- Assuma que o retângulo definido nessa biblioteca seja paralelo aos eixos.

- Escreva funções para: computar a área do retângulo, determinar se um ponto está localizado dentro de um retângulo e determinar se dois retângulos estão separados ou sobrepostos.