# 29. Enumerações e uniões

DIM0321

2015.1



## Outline









## Contexto

Muitas vezes o conjunto de valores aceitáveis para uma variável é restrito

### Exemplo

- lógico: verdadeiro/falso
- sexo: homem/mulher
- tipo de peça de xadrez: peão, cavalo, bispo, torre, rei, dama

# Codificação

## Codificação com tipos de base

- Por exemplo, inteiros (peão = 1, cavalo = 2, bispo = 3, ...)
- Abordagem difícil de compreensão.
- Não é possível saber que vai ter apenas $n$ valores.
- O significado dos valores não é transparente.

## Enumeração

- Tipo especial de variável que deve conter um número restrito de valores distintos
- A representação **concreta** usa uma sequência automática de números naturais.



# Sintaxe / declaração

```c
enum peca { // O tipo enum é tratado como um inteiro
  PEAO,   // 0
  CAVALO, // 1
  BISPO,  // 2
  TORRE,  // 3
  REI,    // 4
  DAMA    // 5
};
enum peca p;

enum logico {  // Declaração no momento da declaração do enum
  V,
  F
} booleano, primos[100];

typedef enum peca peca_t;  // Podemos usar typedef

typedef enum sexo {
  HOMEM = 4,
  MULHER = 3
} sexo_t;

typedef enum cor {  // É possível atribuir valores diferentes do padrao 0,1,2,3,...
  PRETO , // = 0
  BRANCO = 2,
  AMARELO , // = 3
  VERMELHO = 9
} cor_t;
```



# Exemplo

```c
#include <stdio.h>
int main(void)
{
    enum dia
    {
        DOMINGO,
        SEGUNDA,
        TERCA,
        QUARTA,
        QUINTA,
        SEXTA,
        SABADO
    } dia_semana = QUARTA;

    enum dia primeiro_dia = DOMINGO;
    enum dia ultimo_dia = SABADO;

    printf( "Primeiro dia da semana: %i\n", primeiro_dia );
    printf( "Ultimo dia da semana: %i\n", ultimo_dia );
    printf( "Hoje = %i\n", dia_semana );
    printf( "Amanha = %i\n", ++dia_semana );
    dia_semana = SABADO;
    printf( "Outro dia da semana = %i\n", dia_semana );
    return 0;
}
```



# Exemplo

```c
#include <stdio.h>

typedef enum {
    NEGATIVO,
    POSITIVO
} sinal_t;

typedef struct {
    sinal_t sinal;
    unsigned int numerador;
    unsigned int denominador;
} racional_t;

void imprime_racional(racional_t r)
{
    printf("%c", r.sinal == POSITIVO ? '+' : '-');
    printf("%u/%u", r.numerador, r.denominador);
}

racional_t mult_racional(racional_t r1, racional_t r2)
{
    racional_t resultado;
    resultado.sinal = (r1.sinal == r2.sinal ? POSITIVO : NEGATIVO);
    resultado.numerador = r1.numerador * r2.numerador;
    resultado.denominador = r1.denominador * r2.denominador;
    return resultado;
}
```

```c
int main(void)
{
    racional_t x = { POSITIVO, 2, 5 };
    racional_t y = { NEGATIVO, 1, 2 };
    racional_t z = mult_racional(x, y);
    imprime_racional(x);
    printf(" * ");
    imprime_racional(y);
    printf(" = ");
    imprime_racional(z);
    return 0;
}
```

# Contexto

- Sistemas com pouco espaço de memória disponível: em certos casos precisamos economizar espaço de memória alocado desnecessariamente em registros.
- Em outros casos precisamos criar estruturas contendo uma mistura de diferentes tipos em um único campo. Ex: Conjunto de números inteiros e de números de ponto flutuante armazenados em um único vetor.



# União

## Definição

- Tipo especial de registro
- Ele armazena apenas um dos valores de seus campos na memória.
- O compilador aloca espaço suficiente apenas para **o maior** dos campos, em termos de bytes alocados. Todos os campos compartilham o mesmo espaço na memória. A atribuição de um valor a um dos campos também altera os valores dos outros campos.
  - `struct s { char c; int i; }` = 5 bytes
  - `union u { char c; int i; }` = 4 bytes



# Sintaxe / declaração

De um novo tipo / muito parecido a um registro. Só o primeiro campo pode ser inicializado na declaração.

```
1  union num {
2    int i;
3    float f;
4  };
5
6  union num v[5];
7
8  typedef union num num_t;
9
10 num_t n;
```

# Exemplo

```c
#include <stdio.h>
#define MAX 100

typedef enum {
    TIPO_INT,
    TIPO_DOUBLE
} tipo;

typedef struct {
    tipo tipo;
    union {
        int i;
        double d;
    } uniao;
} mixnum;
```

```c
int main(void)
{
    int i, soma = 0, inteiro = 0, real = 0;
    double produto = 1.;
    mixnum numero[MAX];
    for (i = 0; i < MAX; i++) {
        scanf("%i", &numero[i].tipo);
        if (numero[i].tipo == TIPO_INT)
            scanf("%i", &numero[i].uniao.i);
        else
            scanf("%lf", &numero[i].uniao.d);
    }
    for (i = 0; i < MAX; i++) {
        if (numero[i].tipo == TIPO_INT) {
            soma += numero[i].uniao.i;
            inteiro = 1;
        } else {
            produto *= numero[i].uniao.d;
            real = 1;
        }
    }
    if (inteiro) printf("Soma dos inteiros: %i\n", soma);
    if (real) printf ("Produto dos reais: %g\n", produto);
    return 0;
}
```



# Exemplo

```c
typedef enum {
    NEGATIVO,
    POSITIVO
} sinal_t;

typedef struct {
    sinal_t sinal;
    unsigned int numerador;
    unsigned int denominador;
} racional_t;

typedef union {
    unsigned int natural;
    int inteiro;
    racional_t racional;
} num_t;
```

```c
#include <stdio.h>
int main(void)
{
    num_t num1, num2, num3, num4;
    printf("sizeof (unsigned int) = %lu\n"
           "sizeof (int) = %lu\n"
           "sizeof (double) = %lu\n"
           "sizeof (sinal_t) = %lu\n"
           "sizeof (racional_t) = %lu\n"
           "sizeof (num_t) = %lu\n",
           sizeof(unsigned int), sizeof(int),
           sizeof(double), sizeof(sinal_t),
           sizeof(racional_t), sizeof(num_t));
    num1.natural = 1;
    num2.inteiro = 1;
    num4.racional.sinal = POSITIVO;
    num4.racional.numerador = 1;
    num4.racional.denominador = 1;
    printf("sizeof (num1) = %lu\n"
           "sizeof (num2) = %lu\n"
           "sizeof (num3) = %lu\n"
           "sizeof (num4) = %lu\n",
           sizeof(num1), sizeof(num2),
           sizeof(num3), sizeof(num4));
    return 0;
}
```



# União / bit-field

```c
#include <stdio.h>

typedef union {
     float f;
     struct {
          unsigned mantissa : 23;
          unsigned exponent : 8;
          unsigned sign : 1;
     } parts;
} myfloat;


int main(void)
{
     myfloat f;
     scanf("%f", &f.f);
     printf("correct mantissa : %u\n", f.parts.mantissa);
     printf("correct exponent : %u\n", f.parts.exponent);
     return 0;
}
```



# Exercício

O que faz esse trecho de código ?

```c
#include <stdio.h>

typedef union {
    struct {  /* registro anônimo que não declara campo */
        unsigned char blue;
        unsigned char green;
        unsigned char red;
        unsigned char alpha;
    };
    unsigned int value;
} color_t;

int main( void )
{
    color_t color;
    scanf( "%u", &color.value );
    printf( "%hhu\n", color.alpha );
    printf( "%hhu\n", color.red );
    printf( "%hhu\n", color.green );
    printf( "%hhu\n", color.blue );
    return 0;
}
```

# Referências

## Precisões

[KR88] 6 (6.8, 6.9)

[Bac13] 8.2, 8.3


André Backes, *Linguagem C completa e descomplicada*, Elsevier, 2013.

Brian W. Kernighan and Dennis M. Ritchie, *The (ANSI) C Programming Language*, 2nd ed., Prentice Hall Professional Technical Reference, 1988.




# Perguntas ?


`http://dimap.ufrn.br/~richard/dim0321`