

Enumerações

David Deharbe



Conceitos

- Uma enumeração permite definir um novo tipo que pode ter um número finito, e geralmente pequeno, de valores diferentes:
 - sexo de uma pessoa: homem, mulher.
 - naipe de uma carta de baralho tradicional: paus, copas, ouros, espadas.
 - cor de uma peça de xadrez: branco, preto.
 - tipo de uma peça de xadrez: peão, cavalo, bispo, torre, rei, dama.
 - cor no baralho MtG: branco, azul, vermelho, verde, preto.

Declaração de uma enumeração na linguagem C.

• Sintaxe

`<enumeração> ::= enum [<nome>] { <nome>+ } <nome>+`

```
▸ typedef enum { /* tipos de peça de xadrez */
    PAWN, TOWER, KNIGHT, BISHOP, QUEEN, KING
} chess_kind;
typedef enum { /* tipos de peça de xadrez */
    BLACK, WHITE
} chess_color;
typedef struct TSchess_piece {
    chess_kind kind;
    chess_color color;
} Tchess_piece;
```

Exemplo

```
#include <stdio.h>

typedef enum TEsignt {
    NEGATIVE,
    POSITIVE
} Tsign;

typedef struct TSrational {
    Tsign sign;
    unsigned num;
    unsigned den;
} Trational;
```

Exemplo

```
#include <stdio.h>

typedef enum TEsign {
    NEGATIVE,
    POSITIVE
} Tsign;

typedef struct TSrational {
    Tsign sign;
    unsigned num;
    unsigned den;
} Trational;

void rat_print(Trational r)
{
    if (r.sign == POSITIVE)
        printf("+");
    else
        printf("-");
    printf("%u/%u", r.num, r.den);
}
```

Exemplo

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

```
typedef enum TEsign {
    NEGATIVE,
    POSITIVE
} Tsign;
```

```
typedef struct TSrational {
    Tsign sign;
    unsigned num;
    unsigned den;
} Trational;
```

```
void rat_print(Trational r)
{
    if (r.sign == POSITIVE)
        printf("+");
    else
        printf("-");
    printf("%u/%u", r.num, r.den);
}
```

```
Trational rat_mult(Trational r1, Trational r2)
{
    Trational result;
    if (r1.sign == r2.sign)
        result.sign = POSITIVE;
    else
        result.sign = NEGATIVE;
    result.num = r1.num * r2.num;
    result.den = r1.den * r2.den;
    return result;
}
```

Exemplo

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

```
typedef enum TEsign {
    NEGATIVE,
    POSITIVE
} Tsign;
```

```
typedef struct TSrational {
    Tsign sign;
    unsigned num;
    unsigned den;
} Trational;
```

```
void rat_print(Trational r)
{
    if (r.sign == POSITIVE)
        printf("+");
    else
        printf("-");
    printf("%u/%u", r.num, r.den);
}
```

```
Trational rat_mult(Trational r1, Trational r2)
{
    Trational result;
    if (r1.sign == r2.sign)
        result.sign = POSITIVE;
    else
        result.sign = NEGATIVE;
    result.num = r1.num * r2.num;
    result.den = r1.den * r2.den;
    return result;
}
```

```
int main (void)
{
    Trational x = { POSITIVE, 2, 5 };
    Trational y = { NEGATIVE, 1, 2 };
    Trational z = rat_mult(x, y);
    rat_print(x); printf(" * "); rat_print(y);
    printf(" = "); rat_print(z);
    printf("\n");
    return 0;
}
```

Exemplo

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

```
typedef enum TEsign {
    NEGATIVE,
    POSITIVE
} Tsign;
```

```
typedef struct TRational {
    Tsign sign;
    unsigned num;
    unsigned den;
} TRational;
```

```
void rat_print(TRational r)
{
    if (r.sign == POSITIVE)
        printf("+");
    else
        printf("-");
    printf("%u/%u", r.num, r.den);
}
```

```
localhost:aula 26 david$ ./ex1
+2/5 * -1/2 = -2/10
```

```
{
    TRational result;
    if (r1.sign == r2.sign)
        result.sign = POSITIVE;
    else
        result.sign = NEGATIVE;
    result.num = r1.num * r2.num;
    result.den = r1.den * r2.den;
    return result;
}
```

```
int main (void)
{
    TRational x = { POSITIVE, 2, 5 };
    TRational y = { NEGATIVE, 1, 2 };
    TRational z = rat_mult(x, y);
    rat_print(x); printf(" * "); rat_print(y);
    printf(" = "); rat_print(z);
    printf("\n");
    return 0;
}
```

Uniões

David Deharbe



Conceitos

- Em C, uma união é um valor que pode ter valores de diferentes tipos.
 - Uma união de N tipos t_1, t_2, \dots, t_N ocupa o espaço necessário para armazenar um valor do tipo t_i que ocupa a maior quantidade de espaço.
- A sintaxe é similar à de registros.
 - a única diferença é a palavra chave **union** que substitui a palavra chave **struct**.

Declaração de união na linguagem C.

• Sintaxe

`<união> ::= union { <campo>+ } <nome>+`

`<campo> ::= <tipo> <nome> ;`

```
▶ union { /* uma união com 4 alternativas */
    unsigned natural;
    int integer;
    double real;
    struct {
        enum { NEGATIVE, POSITIVE } sign;
        unsigned num;
        unsigned den;
    } rational;
} num;
```

Exemplo

```
#include <stdio.h>
typedef enum TSign {
    NEGATIVE,
    POSITIVE
} Tsign;
typedef struct TSrational {
    Tsign sign;
    unsigned num;
    unsigned den;
} Trational;
typedef union TUnum {
    unsigned natural;
    int integer;
    double real;
    Trational rational;
} Tnumber;
```

Exemplo

```
#include <stdio.h>
typedef enum TSign {
    NEGATIVE,
    POSITIVE
} Tsign;
typedef struct TRational {
    Tsign sign;
    unsigned num;
    unsigned den;
} TRational;
typedef union TUnum {
    unsigned natural;
    int integer;
    double real;
    TRational rational;
} Tnumber;
```

```
int main (void)
{
    Tnumber num1, num2, num3, num4;

    printf("sizeof(unsigned) = %lu\n"
           "sizeof(int) = %lu\n"
           "sizeof(double) = %lu\n"
           "sizeof(Tsign) = %lu\n"
           "sizeof(TRational) = %lu\n"
           "sizeof(Tnumber) = %lu\n",
           sizeof(unsigned), sizeof(int), sizeof(double),
           sizeof(Tsign), sizeof(TRational),
           sizeof(Tnumber));
    num1.natural = 1;
    num2.integer = 1;
    num3.real = 1.0;
    num4.rational.sign = POSITIVE;
    num4.rational.num = 1;
    num4.rational.den = 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;
}
```

Exemplo

```
int main (void)
{
    Tnumber num1, num2, num3, num4;
```

```
localhost:aula 26 david$ ./ex2
```

```
#include <stdio.h>
typedef enum {
    NEGATIVE,
    POSITIVE
} Tsign;
typedef struct {
    Tsign sign;
    unsigned int natural;
} Trational;
typedef union TUnum {
    unsigned int natural;
    int integer;
    double real;
    Trational rational;
} Tnumber;
```

```
sizeof(unsigned) = 4
sizeof(int) = 4
sizeof(double) = 8
sizeof(Tsign) = 4
sizeof(Trational) = 12
sizeof(Tnumber) = 16
sizeof(num1) = 16
sizeof(num2) = 16
sizeof(num3) = 16
sizeof(num4) = 16
```

```
num3.real = 1.0;
num4.rational.sign = POSITIVE;
num4.rational.num = 1;
num4.rational.den = 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;
```

Alerta sobre valores de tipo união

- Não existe comando C que permita determinar qual alternativa segue um valor de um tipo união.
- É necessário o programador incluir comandos e dados para manter e determinar esta informação.
- Uma solução é definir uma estrutura com:
 - um campo que guarda o valor do tipo união;
 - um campo que guarda qual alternativa este valor segue
 - ✓ esse segundo campo pode ser uma enumeração.

Exemplo

```
#include <stdio.h>
typedef enum TEsign {
    NEGATIVE,
    POSITIVE
} Tsign;
typedef struct TSrational {
    Tsign sign;
    unsigned num;
    unsigned den;
} Trational;

typedef struct TNumber {
    union {
        int integer;
        Trational rational;
    } value;
    enum {
        INTEGER,
        RATIONAL
    } kind;
} Tnumber;
```

```
void rat_print(Trational r)
{
    if (r.sign == POSITIVE) printf("+");
    else printf("-");
    printf("%u/%u", r.num, r.den);
}

void num_print(Tnumber n) {
    if (n.kind == INTEGER)
        printf("%i", n.value.integer);
    else
        rat_print(n.value.rational);
}

int main (void) {
    Tnumber num1, num2;
    num1.kind = INTEGER;
    num1.value.integer = 1;
    num2.kind = RATIONAL;
    num2.value.rational.sign = NEGATIVE;
    num2.value.rational.num = 1;
    num2.value.rational.den = 2;
    num_print(num1); printf("\n");
    num_print(num2); printf("\n");
    return 0;
}
```

Exemplo

```
#include <stdio.h>
typedef enum TSign {
    NEGATIVE,
    POSITIVE
} Tsign;
typedef struct TSrational {
    Tsign sign;
    unsigned num;
    unsigned den;
} Trational;

typedef struct TNumber {
    union {
        int integer;
        Trational rational;
    } value;
    enum {
        INTEGER,
        RATIONAL
    } kind;
} Tnumber;
```

```
localhost:aula 26 david$ ./ex3
```

```
1
-1/2
```

```
void rat_print(Trational r)
{
    printf("%d/%d", r.num, r.den);
}

void num_print(Tnumber n) {
    if (n.kind == INTEGER)
        printf("%i", n.value.integer);
    else
        rat_print(n.value.rational);
}

int main (void) {
    Tnumber num1, num2;
    num1.kind = INTEGER;
    num1.value.integer = 1;
    num2.kind = RATIONAL;
    num2.value.rational.sign = NEGATIVE;
    num2.value.rational.num = 1;
    num2.value.rational.den = 2;
    num_print(num1); printf("\n");
    num_print(num2); printf("\n");
    return 0;
}
```

Exercício

- Considerando o tipo **Tnumber**, definido anteriormente, escreva uma sub-rotina que calcula e retorna a soma de dois valores do tipo **Tnumber**.
 - Sempre que possível, o campo **value** deve ser um valor **integer**.
- Defina uma sub-rotina que calcula e retorna o produto de dois valores do tipo **Tnumber**.
- Defina uma sub-rotina que testa se um valor do tipo **Tnumber** é menor ou igual a um segundo valor do mesmo tipo.