

# DIM0108.0 - Aula 27

David Deharbe [DIMAp/UFRN]

1 de dezembro de 2011

**Instruções** Segue uma lista de tarefas a realizar. Entregue o resultado do seu trabalho ao docente quando terminar a aula, ou a lista (o que vier primeiro). Você tem direito à consulta de material didático e a trabalhar em colaboração com outros alunos, sendo que cada resposta deve ser entregue individualmente.

- Leia a folha inteira antes de começar a elaborar as suas soluções.
- Foram incluídas, na margem, informações acerca da relação de dependência entre as diversas questões.
- Você pode usar as soluções anteriores na elaboração de uma nova solução.

**Tarefas** Considere os seguintes tipos:

```
/* a type for the sign of numbers */
typedef enum Tesign { NEGATIVE, POSITIVE } Tsign;

/* a type for rational numbers */
typedef struct TRational {
    Tsign sign;
    unsigned num;
    unsigned den;
} TRational;

/* a type for numbers: integers and rationals */
typedef struct TNumber {
    union {
        int integer;
        TRational rational;
    } value;
    enum {
        INTEGER,
        RATIONAL
    } kind;
} Tnumber;
```

Primeiro, definimos algumas sub-rotinas úteis para manipular números e a representação deles usando os tipos definidos acima.

A sub-rotina `ugcd` calcula o maior divisor comum de dois números positivos.

```
/* auxiliary routine: greatest common divisor */
unsigned ugcd (unsigned n1, unsigned n2)
{
    unsigned rest;
    if (n1 < n2) return ugcd(n2, n1);
    rest = n1 % n2;
    if (rest == 0)
        return n2;
    else
        return ugcd(n2, rest);
}
```

A sub-rotina `integer_first` inverte o valor de dois registros, sob a condição do primeiro ser a representação de um racional.

```

/* auxiliary routine: conditional swap of numbers pointed to by n1 and n2 */
void integer_first (Tnumber * n1, Tnumber * n2)
{
    if (n1->kind == RATIONAL) {
        Tnumber tmp = *n1;
        *n1 = *n2;
        *n2 = tmp;
    }
}

```

A sub-rotina `sign_coef` tem como argumento um valor `Tsign` e retorna 1 se o argumento for `POSITIVE`, e -1 caso contrário.

```

/* auxiliary routine: multiplicative coefficient corresponding to sign */
int sign_coef(Tsign sign)
{
    if (sign == POSITIVE) return 1;
    else return -1;
}

```

Escreva uma subrotina que normaliza um valor do tipo `Tnumber`. Se o campo `value` for seguir a alternativa `rational`, o denominador tem que ser diferente de um (1), e o numerador e o denominador deverão ser primos entre si.

Esta sub-rotina visa alterar uma estrutura. Como a passagem de estruturas em parâmetro é realizada por valor, precisamos lançar mão de um ponteiro para poder efetuar a alteração dentro da sub-rotina.

```
void number_normalize(Tnumber * Pnumber)
{
    if (Pnumber->kind == INTEGER)
        return;
    else {
        unsigned * num = &Pnumber->value.rational.num;
        if (*num == 0) {
            Pnumber->kind = INTEGER;
            Pnumber->value.integer = 0;
        } else {
            unsigned * den = &Pnumber->value.rational.den;
            Tsign sign = Pnumber->value.rational.sign;
            int d = ugcd(*num, *den);
            if (d != 1) {
                *num /= d;
                *den /= d;
            }
            if (*den == 1) {
                Pnumber->kind = INTEGER;
                Pnumber->value.integer = *num;
                if (sign == NEGATIVE)
                    Pnumber->value.integer *= -1;
            }
        }
    }
}
```

Escreva uma subrotina que calcula e retorna a soma de dois valores do tipo Tnumber. Sempre que possível, ou seja quando o numerador do resultado é um múltiplo do seu denominador, o campo value deve seguir a alternativa integer.

```
Tnumber number_add(Tnumber n1, Tnumber n2)
{
    Tnumber result;
    if (n1.kind == INTEGER && n2.kind == INTEGER) {
        result.kind = INTEGER;
        result.value.integer = n1.value.integer + n2.value.integer;
    } else if (n1.kind == RATIONAL && n2.kind == RATIONAL) {
        Trational r1 = n1.value.rational, r2 = n2.value.rational;
        unsigned f1, f2;
        result.kind = RATIONAL;
        result.value.rational.den = r1.den * r2.den;
        result.value.rational.num = 0;
        f1 = r1.num*r2.den;
        f2 = r1.den*r2.num;
        if (r1.sign == r2.sign) {
            result.value.rational.sign = r1.sign;
            result.value.rational.num = f1 + f2;
        } else if (f1 == f2) {
            result.value.rational.sign = POSITIVE;
            result.value.rational.num = 0;
        } else if (f1 > f2) {
            result.value.rational.sign = r1.sign;
            result.value.rational.num = f1 - f2;
        } else {
            result.value.rational.sign = r2.sign;
            result.value.rational.num = f2 - f1;
        }
        number_normalize(&result);
    } else {
        int i1, num;
        unsigned num2, den2;
        Tsign sign2;
        integer_first(&n1, &n2);
        i1 = n1.value.integer;
        num2 = n2.value.rational.num;
        den2 = n2.value.rational.den;
        sign2 = n2.value.rational.sign;
        result.kind = RATIONAL;
        result.value.rational.den = den2;
        if (sign2 == NEGATIVE)
            num = - num2 + den2 * i1;
        else
            num = num2 + den2 * i1;
        if (num < 0) {
            result.value.rational.sign = NEGATIVE;
            result.value.rational.num = -num;
        } else {
            result.value.rational.sign = POSITIVE;
            result.value.rational.num = num;
        }
    }
    return result;
}
```

Defina uma sub-rotina que calcula e retorna o produto de dois valores do tipo Tnumber.

```
Tnumber number_mult(Tnumber n1, Tnumber n2)
{
    Tnumber result;
    if (n1.kind == INTEGER && n2.kind == INTEGER) {
        result.kind = INTEGER;
        result.value.integer = n1.value.integer * n2.value.integer;
    } else if (n1.kind == RATIONAL && n2.kind == RATIONAL) {
        unsigned f, g;
        if (n1.value.rational.sign == n2.value.rational.sign)
            result.value.rational.sign = POSITIVE;
        else
            result.value.rational.sign = NEGATIVE;
        f = ugcd(n1.value.rational.num, n2.value.rational.den);
        g = ugcd(n1.value.rational.den, n2.value.rational.num);
        result.value.rational.num = (n1.value.rational.num / f) *
            (n2.value.rational.num / g);
        result.value.rational.den = (n1.value.rational.den / g) *
            (n2.value.rational.den / f);
    } else {
        int coef;
        unsigned abs, f;
        integer_first(&n1, &n2);
        if (n1.value.integer < 0) {
            coef = -1;
            abs = - n1.value.integer;
        } else {
            coef = 1;
            abs = n1.value.integer;
        }
        f = ugcd(abs, n2.value.rational.den);
        if (f == n2.value.rational.den) {
            result.kind = INTEGER;
            result.value.integer = coef * (abs / f) * n1.value.rational.num;
        } else {
            result.kind = RATIONAL;
            result.value.rational.den = n2.value.rational.den / f;
            result.value.rational.num = n1.value.rational.num * (abs / f);
            if ((coef == -1 && n1.value.rational.sign == NEGATIVE) ||
                (coef == 1 && n1.value.rational.sign == POSITIVE))
                result.value.rational.sign = POSITIVE;
            else
                result.value.rational.sign = NEGATIVE;
        }
    }
    return result;
}
```

Defina uma sub-rotina que determina se um valor do tipo Tnumber é menor ou igual a outro valor do tipo Tnumber.

```
int number_le (Tnumber n1, Tnumber n2)
{
  if (n1.kind == INTEGER && n2.kind == INTEGER)
    return n1.value.integer <= n2.value.integer;
  else if (n1.kind == RATIONAL && n2.kind == RATIONAL) {
    return
      (sign_coef(n1.value.rational.sign) * n1.value.rational.num *
       n2.value.rational.den) <=
      (sign_coef(n2.value.rational.sign) * n2.value.rational.num *
       n1.value.rational.den);
  } else {
    integer_first(&n1, &n2);
    return
      sign_coef(n2.value.rational.sign) * n1.value.integer * n2.value.rational.den <=
      n2.value.rational.num;
  }
}
```