30. Desenvolver um Tipo de Dado Abstrato (TDA) denominado `Rational` que represente frações matemáticas e permita a realização de operações aritméticas básicas com precisão exata.

Especificações técnicas:

1) Atributos:
   a) `numerator` – o numerador, do tipo inteiro;
   b) `denominator` – o denominador, do tipo inteiro.
2) Função de inicialização:
   a) Recebe dois inteiros (numerador, denominador);
   b) Normalização de sinal: o denominador deve ser sempre positivo. Ex.: 3/-4 → -3/4; -2/-5 → 2/5;
   c) Redução imediata: a fração deve ser armazenada na forma irredutível usando o Máximo Divisor Comum (MDC). Ex.: 2/4 → 1/2; 0/7 → 0/1;
   d) Caso o denominador seja zero, exibir mensagem em `stderr` e encerrar com `exit(1)`.
3) Implemente métodos que recebam dois objetos `Rational` e retornem um novo objeto `Rational` com o resultado:
   a) `add(Rational a, Rational b)` → adição
   b) `subtract(Rational a, Rational b)` → subtração
   c) `multiply(Rational a, Rational b)` → multiplicação
   d) `divide(Rational a, Rational b)` → divisão
4) Conversão e exibição:
   a) `toString(Rational number)` → retorna a string no formato "a/b", como "1/2";
   b) `toDouble(Rational number)` → retorna o valor decimal de ponto flutuante correspondente, como 0.5.

Requisitos de implementação:

- Mantenha a imutabilidade: os operandos não devem ser alterados; cada operação retorna uma nova TDA;
- Implemente uma função auxiliar para cálculo do MDC.

Exemplo de comportamento esperado:

```
Rational r1 = create(2, 4)  // armazena internamente como 1/2
Rational r2 = create(1, 3)  // armazena internamente como 1/3
Rational r3 = add(r1, r2)   // resulta em 5/6
char *text = toString(r3)   // "5/6"
double d = toDouble(r3)     // 0.833333...
Rational r4 = create(6, -8) // resulta em -3/4
char *text = toString(r4)   // "-3/4"
```

Observações:

- A linguagem de programação é de livre escolha, desde que suporte TDA, como as linguagens de programação C, Pascal e Ada.

Resposta codificada em C, disponível em https://onlinegdb.com/sTyl4XJDs

```
/**
 * Encapsulamento dos atributos do número racional
 * Arquivo Rational.h
 */
typedef struct rational Rational;

/**
 * Construtor do número racional
 *
 * @param numerator o numerador do número racional
 * @param denominator o denominador do número racional
 * @return a instância do número racional
 */
Rational* create (int, int);

/**
 * Destrutor do número racional
 *
 * @param instance a instância do número racional
 */
void destroy (Rational*);

/**
 * Realizar a adição de dois números racionais
 *
 * @param first o primeiro número racional
 * @param second o segundo número racional
 * @return a adição de dois números racionais
 */
Rational* add (Rational*, Rational*);

/**
 * Realizar a subtração de dois números racionais
 *
 * @param first o primeiro número racional
 * @param second o segundo número racional
 * @return a subtração de dois números racionais
 */
Rational* subtract (Rational*, Rational*);

/**
 * Realizar a multiplicação de dois números racionais
 *
 * @param first o primeiro número racional
 * @param second o segundo número racional
 * @return a multiplicação de dois números racionais
 */
Rational* multiply (Rational*, Rational*);

/**
 * Realizar a divisão de dois números racionais
 *
 * @param first o primeiro número racional
 * @param second o segundo número racional
 * @return a divisão de dois números racionais
 */
Rational* divide (Rational*, Rational*);
```

```
/**
 * Apresentar o número racional no formato racional
 *
 * @param instance a instância do número racional
 * @return o número racional no formato racional
 */
char* toString(Rational*);

/**
 * Apresentar o número racional no formato decimal
 *
 * @param instance a instância do número racional
 * @return o número racional no formato decimal
 */
double toDouble(Rational*);
```

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Rational.h"

/**
 * Encapsulamento dos atributos do número racional
 * Arquivo Rational.c
 */
struct rational
{
  /**
   * O numerador do número racional
   */
  int numerator;

  /**
   * O denominador do número racional
   */
  int denominator;
};

/**
 * Armazenar o numero racional na forma reduzida
 *
 * @param numerator o numerador do número racional
 * @param denominator o denominador do número racional
 * @return a instância do número racional
 */
Rational* reduction(int numerator, int denominator)
{
  if (denominator == 0)
  {
    fprintf(stderr, "Denominador inválido: 0");

    exit (EXIT_FAILURE);
  }

  if (denominator < 0)
  {
    numerator = numerator * (-1);

    denominator = denominator * (-1);
  }

  int largest = 0;

  int mdc = 1;

  if (numerator > denominator)
  {
    largest = numerator;
  }
  else
  {
    largest = denominator;
  }
```

```
  for (int loop = 2; loop <= largest; loop++)
  {
    if ((numerator % loop == 0) && (denominator % loop == 0))
    {
      mdc = loop;
    }
  }

  Rational* instance = (Rational*) malloc(sizeof(Rational));

  instance->numerator = numerator / mdc;

  instance->denominator = denominator / mdc;

  return instance;
}

/**
 * Retornar a quantidade de dígitos do número
 *
 * @param number o número
 * @return a quantidade de dígitos do número
 */
int digits(int number)
{
  if (number == 0) return 1;

  int counter = 0;

  if (number < 0)
  {
    number = abs(number);

    counter = 1;
  }

  while (number > 0)
  {
    number /= 10;

    counter++;
  }

  return counter;
}

/**
 * Construtor do número racional
 *
 * @param numerator o numerador do número racional
 * @param denominator o denominador do número racional
 * @return a instância do número racional
 */
Rational* create (int numerator, int denominator)
{
  return reduction(numerator, denominator);
}
```

```
/**
 * Destrutor do número racional
 *
 * @param instance a instância do número racional
 */
void destroy (Rational* instance)
{
  free(instance);
}

/**
 * Realizar a adição de dois números racionais
 *
 * @param first o primeiro número racional
 * @param second o segundo número racional
 * @return a adição de dois números racionais
 */
Rational* add (Rational* first, Rational* second)
{
  int numerator = first->numerator * second->denominator +
                  first->denominator * second->numerator;

  int denominator = first->denominator * second->denominator;

  return reduction(numerator, denominator);
}

/**
 * Realizar a subtração de dois números racionais
 *
 * @param first o primeiro número racional
 * @param second o segundo número racional
 * @return a subtração de dois números racionais
 */
Rational* subtract (Rational* first, Rational* second)
{
  int numerator = first->numerator * second->denominator -
                  first->denominator * second->numerator;

  int denominator = first->denominator * second->denominator;

  return reduction(numerator, denominator);
}

/**
 * Realizar a multiplicação de dois números racionais
 *
 * @param first o primeiro número racional
 * @param second o segundo número racional
 * @return a multiplicação de dois números racionais
 */
Rational* multiply (Rational* first, Rational* second)
{
  int numerator = first->numerator * second->numerator;

  int denominator = first->denominator * second->denominator;

  return reduction(numerator, denominator);
}
```

```
/**
 * Realizar a divisão de dois números racionais
 *
 * @param first o primeiro número racional
 * @param second o segundo número racional
 * @return a divisão de dois números racionais
 */
Rational* divide (Rational* first, Rational* second)
{
  int numerator = first->numerator * second->denominator;

  int denominator = first->denominator * second->numerator;

  return reduction(numerator, denominator);
}

/**
 * Apresentar o número racional no formato racional
 *
 * @param instance a instância do número racional
 * @return o número racional no formato racional
 */
char* toString(Rational* instance)
{
  int length = digits(instance->numerator) + 2 + digits(instance->denominator);

  char text[length];

      snprintf(text,   sizeof(text),   "%d/%d",   instance->numerator,   instance-
>denominator);

  char *str = (char*)malloc(length * sizeof(char));

  strcpy(str, text);

  return str;
}

/**
 * Apresentar o número racional no formato decimal
 *
 * @param instance a instância do número racional
 * @return o número racional no formato decimal
 */
double toDouble(Rational* instance)
{
  return 1.0 * instance->numerator / instance->denominator;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include "Rational.h"

int main()
{
  Rational* first = create(-6, -4);
  char* textFirst = toString(first);

  Rational* second = create(5, -15);
  char* textSecond = toString(second);


  Rational* result = add(first, second);
  char* text = toString(result);
     printf("Add:  %s  +  %s  =  %s  (%f)\n",  textFirst,  textSecond,  text,
toDouble(result));
  destroy(result);
  free(text);


  result = subtract(first, second);
  text = toString(result);
    printf("Subtract:  %s  -  %s  =  %s  (%f)\n",  textFirst,  textSecond,  text,
toDouble(result));
  destroy(result);
  free(text);


  result = multiply(first, second);
  text = toString(result);
    printf("Multiply:  %s  *  %s  =  %s  (%f)\n",  textFirst,  textSecond,  text,
toDouble(result));
  destroy(result);
  free(text);


  result = divide(first, second);
  text = toString(result);
     printf("Divide:  %s  /  %s  =  %s  (%f)\n",  textFirst,  textSecond,  text,
toDouble(result));
  destroy(result);
  free(text);


  destroy(first);
  free(textFirst);

  destroy(second);
  free(textSecond);

  return 0;
}
```