```c
/*
 * Tipos Abstratos de Dados - TADs
 * Arquivo de implementação para TAD racional.
 * Feito em 16/09/2024 para a disciplina CI1001 - Programação 1.
 *
 * Este arquivo deve conter as implementações das funções cujos protótipos
 * foram definidos em racional.h. Neste arquivo também podem ser definidas
 * funções auxiliares para facilitar a implementação daquelas funções.
*/

#include <stdio.h>
#include <stdlib.h>
#include "racional.h"

/* Retorna um número aleatório entre min e max, inclusive
 * (max - min + 1) gera um número entre 0 e (max - min)
 * Somando min, temos o deslocamento do resultado para o intervalo [min, max]
*/
long aleat(long min, long max)
{
  return rand() % (max - min + 1) + min;
}

/* Máximo Divisor Comum entre a e b pelo método de Euclides */
long mdc(long a, long b)
{
  if (b == 0) {
    return a;
  }

  return mdc(b, a % b);
}

/* Mínimo Múltiplo Comum entre a e b */
long mmc(long a, long b)
{
  return (a * b) / mdc(a, b);
}

/* Recebe um número racional e o simplifica */
struct racional simplifica_r(struct racional r)
{
  long resultado_mdc;
  long numerador, denominador;

  /* Verifica e retorna o racional se ele for inválido */
  if (!valido_r(r)) {
    return r;
  }

  resultado_mdc = mdc(r.num, r.den);

  /* Simplifica o numerador e o denominador dividindo-os pelo MDC */
  numerador = r.num / resultado_mdc;
  denominador = r.den / resultado_mdc;

  /* Ajusta o sinal para garantir que o denominador seja positivo */
  if (denominador < 0) {
    numerador = -numerador;
    denominador = -denominador;
  }

  return cria_r(numerador, denominador);
}

/* Cria um número racional com o numerador e denominador indicados */
struct racional cria_r(long numerador, long denominador)
{
  struct racional novo = {numerador, denominador};
  return novo;
}

/* Retorna 1 se o racional r for válido ou 0 se for inválido.
 * Um racional é inválido se seu denominador for zero */
int valido_r(struct racional r)
{
  return r.den != 0; // Condição retorna diretamente os valores esperados (0 ou 1)
}

/* Retorna um número racional aleatório na forma simplificada */
struct racional sorteia_r(long min, long max)
{
```

```c
  long numerador = aleat(min, max);
  long denominador = aleat(min, max);

  return simplifica_r(cria_r(numerador, denominador));
}

/* Imprime um racional r */
void imprime_r(struct racional r)
{
  struct racional numero_simplificado = simplifica_r(r);

  if (!valido_r(numero_simplificado)) {
    printf("INVALIDO");
    /* O else-if a seguir resolve tais problemas:
    * 1. Numerador igual a 0
    * 2. Denominador igual a 1
    * 3. Numerador e denominador iguais
    * No caso do item 3, possivelmente ele cairá no item 2, já que anteriormente
    * aconteceu a simplificação do número racional, levando à situação 1/1
    */
  } else if (!numero_simplificado.num || numero_simplificado.den == 1) {
      printf("%ld", numero_simplificado.num);
  } else {
      printf("%ld/%ld", numero_simplificado.num, numero_simplificado.den);
  }
}

/* Retorna a soma dos racionais r1 e r2 */
struct racional soma_r(struct racional r1, struct racional r2)
{
  /* Se r1 ou r2 for inválido, o resultado deve ser inválido */
  if (!valido_r(r1) || !valido_r(r2)) {
    return cria_r(0, 0);
  }

  long resultado_mmc = mmc(r1.den, r2.den);
  long numerador = ((resultado_mmc * r1.num / r1.den) + resultado_mmc * r2.num / r2.den);

  return cria_r(numerador, resultado_mmc);
}

/* Retorna a subtração dos racionais r1 e r2 */
struct racional subtrai_r(struct racional r1, struct racional r2)
{
  /* Se r1 ou r2 for inválido, o resultado deve ser inválido */
  if (!valido_r(r1) || !valido_r(r2)) {
    return cria_r(0, 0);
  }

  long resultado_mmc = mmc(r1.den, r2.den);
  long numerador = ((resultado_mmc * r1.num / r1.den) - resultado_mmc * r2.num / r2.den);

  return cria_r(numerador, resultado_mmc);
}

/* Retorna a multiplicação dos racionais r1 e r2 */
struct racional multiplica_r(struct racional r1, struct racional r2)
{
  /* Se r1 ou r2 for inválido, o resultado deve ser inválido */
  if (!valido_r(r1) || !valido_r(r2)) {
    return cria_r(0, 0);
  }

  return cria_r(
    r1.num * r2.num,
    r1.den * r2.den
  );
}

/* Retorna a divisão dos racionais r1 e r2 */
struct racional divide_r(struct racional r1, struct racional r2)
{
  /* Se r1 ou r2 for inválido, o resultado deve ser inválido
   * O mesmo vale para o numerador de r2 ser igual a zero
  */
  if (!valido_r(r1) || !valido_r(r2) || !r2.num) {
    return cria_r(0, 0);
  }

  return cria_r(
    r1.num * r2.den,
    r1.den * r2.num
```

```
  );
}
```

```c
/*
 * Tipos Abstratos de Dados - TADs
 * Arquivo do programa principal, que usa o TAD racional.
 * Feito em 16/09/2024 para a disciplina CI1001 - Programação 1.
*/

#include <stdio.h>
#include <stdlib.h>
#include "racional.h"

/* programa principal */
int main()
{
  int n, max, min;
  struct racional r1, r2;

  srand(0);

  /* Certifica que o usuário informou os números no intervalo esperado:
   * 0 < n < 100
   * 0 < max < 30
  */
  do {
    scanf("%d", &n);
    scanf("%d", &max);
  } while ((n <= 0 || n >= 100) || (max <= 0 || max >= 30));

  /* min será o extremo negativo de max */
  min = -max;

  for (int i = 1; i <= n; i++) {
    printf("%d: ", i);

    r1 = sorteia_r(min, max);
    r2 = sorteia_r(min, max);

    imprime_r(r1);

    printf(" ");
    imprime_r(r2);

    /* Evita a realização das operações no caso de números inválidos */
    if (!valido_r(r1) || !valido_r(r2)) {
      printf(" ");
      printf("NUMERO INVALIDO\n");
      return 1;
    }

    struct racional soma, subtracao, multiplicacao, divisao;
    soma = soma_r(r1, r2);
    subtracao = subtrai_r(r1, r2);
    multiplicacao = multiplica_r(r1, r2);
    divisao = divide_r(r1, r2);

    /* Evita a impressão dos resultados no caso de divisão inválida */
    if (!valido_r(divisao)) {
      printf(" ");
      printf("DIVISAO INVALIDA\n");
      return 1;
    }

    printf(" ");
    imprime_r(soma);

    printf(" ");
    imprime_r(subtracao);

    printf(" ");
    imprime_r(multiplicacao);

    printf(" ");
    imprime_r(divisao);
    printf("\n");
  }

  return 0;
}
```

não espalhe as declarações de variáveis, mantenha-as no início da função (exceto para os contadores de "for" ou alguma variável auxiliar dentro de um bloco.