

1 Sobre a entrega do trabalho

São requisitos para atribuição de notas a este trabalho:

- Uso de um arquivo makefile para facilitar a compilação. Os professores rodarão “make” e deverão obter o arquivo executável funcional com a sua solução. Este executável, cujo nome deverá ser `tp2`, deverá estar no subdiretório `tp2`;
- Ao compilar, incluir pelo menos `-Wall -g`. Se não compilar, o trabalho vale zero. Haverá desconto por cada *warning*;
- Arquivo de entrega:
 - Deve estar no formato tar comprimido (`.tgz`);
 - O `tgz` deve ser criado considerando-se que existe um diretório com o nome do trabalho. Por exemplo, este trabalho é o `tp2`;
 - Então seu `tgz` deve ser criado assim:
 - * Estando no diretório `tp2`, faça:
 - * `cd ..`
 - * `tar zcvf tp2.tgz tp2`
 - Desta maneira, quando os professores abrirem o `tgz` (com o comando `tar zxvf tp2.tgz`) terão garantidamente o diretório correto da entrega para poderem fazer a correção semi-automática.
 - O que colocar no `tgz`? Todos os arquivos que são necessários para a compilação, por isso se você usa arquivos além dos especificados, coloque-os também. Mas minimamente deve conter todos os arquivos `.c`, `.h` e o `makefile`;
 - Os professores testarão seus programas em uma máquina do departamento de informática (por exemplo, `cpu1`), por isso, antes de entregar seu trabalho faça um teste em máquinas do `dinf` para garantir que tudo funcione bem.

2 Objetivos

Este trabalho tem como objetivo modificar o Tipo Abstrato de Dados (TAD) para números racionais feito no TP1 para que algumas funções recebam parâmetros por endereços. O motivo da alteração é para que os protótipos das funções fiquem mais elegantes, pois tendo a opção de modificar o valor de uma variável recebida como parâmetro por endereço, podemos utilizar o retorno da função para retornar um código de erro.

Observem que nesta versão do `racionais.h` também foram adicionadas estas linhas:

```
#ifndef _LIBracionais_t_H
#define _LIBracionais_t_H

#endif
```

São instruções para o pré-processador e tem o efeito de evitar que um mesmo arquivo seja incluído mais de uma vez.

3 O trabalho

Você deve adaptar a sua implementação do arquivo `racionais.c` conforme o novo arquivo `racionais.h` fornecido. Observem que algumas poucas funções (as que fazem as operações e a simplificação) são diferentes da versão anterior do TP1 pois contêm parâmetros por endereços.

Você deve baixar o `tp2.tgz` anexo a este enunciado e abrí-lo para poder fazer o trabalho, pois irá precisar de todos os arquivos ali contidos:

racionais.h: arquivo (read only) de *header* com todos os protótipos das funções para manipular números racionais;

racionais.c: um esqueleto de arquivo `racionais.c`;

makefile: sugestão de um makefile que você pode usar.

É sua responsabilidade fazer as adaptações necessárias neste arquivo sugerido.

tp2.c: um esqueleto de arquivo `tp2.c`.

O arquivo `.h` não pode ser alterado. Na correção, os professores usarão os arquivos `.h` originais.

- Use boas práticas de programação, como indentação, bons nomes para variáveis, comentários no código, bibliotecas, defines... Um trabalho que não tenha sido implementado com boas práticas vale zero.
- Quaisquer dúvidas com relação a este enunciado devem ser solucionadas via email para `prog1prof@inf.ufpr.br` pois assim todos os professores receberão os questionamentos. Na dúvida, não tome decisões sobre a especificação, pergunte!
- Dúvidas podem e devem ser resolvidas durante as aulas.

4 Seu programa

No arquivo `racionais.h` foi definida uma interface para o tipo abstrato de dados *racional* que usa a mesma *struct* para os números racionais usada no TP1. Você deve implementar o arquivo `racionais.c` conforme especificado no `racionais.h` fornecido. A sua função `main` deve incluir o *header* `racionais.h` e deve implementar corretamente em *C* o seguinte pseudo-código:

```

inicialize a semente randomica, uma unica vez em todo o codigo
    - sugestao: use "srand (0)" para facilitar os testes
leia um n tal que 0 < n < 100
leia um max tal que 0 < max < 30
para todo i de 1 ate n faca
    /* use um unico espaco em branco separando numeros na mesma linha */
    imprima o valor de i seguido de um : e um espaco em branco
    sortear dois racionais r1 e r2
        - os numeradores e denominadores devem estar entre -n e n
    imprima r1 e r2, na mesma linha e nao mude de linha
    se r1 ou r2 forem um dos dois invalidos, entao:
        imprima "NUMERO INVALIDO" e retorne 1
    calcule r1 + r2
    calcule r1 - r2
    calcule r1 * r2
    calcule r1 / r2
    se a divisao for invalida, entao:
        imprima "DIVISAO INVALIDA" e retorne 1
    imprima na mesma linha r1 + r2
    imprima na mesma linha r1 - r2

```

```

        imprima na mesma linha r1 * r2
        imprima na mesma linha r1 / r2
        mude de linha
    fim_para

    retorne 0

```

Imprima os elementos do vetor em uma única linha usando um único espaço em branco para separar os elementos. Ao final do vetor mude de linha.

5 Exemplos de entrada e saída

Nos exemplos abaixo, se considera que o usuário digitou `n = 10` e `max = 17` como entrada. A saída será diferente, dependendo do gerador de números aleatórios (semente/*seed*).

5.1 Todos os passos do algoritmo

Com semente randômica `srand (10)`, o programa faz todas suas iterações:

```

10 17
1: 13/4 2/17 229/68 213/68 13/34 221/8
2: 12/17 5/13 241/221 71/221 60/221 156/85
3: 17/5 1 22/5 12/5 17/5 17/5
4: 0 2 2 -2 0 0
5: 3/8 17/12 43/24 -25/24 17/32 9/34
6: 6/5 7/13 113/65 43/65 42/65 78/35
7: 8 16 24 -8 128 1/2
8: 1/3 9/17 44/51 -10/51 3/17 17/27
9: 3/2 14/9 55/18 -1/18 7/3 27/28
10: 9 3/8 75/8 69/8 27/8 24

```

OBS.:

1. repare, por exemplo, nas iterações 4, 7 ou 10. Nelas, há números racionais que foram simplificados no formato VALOR/1, deixando de exibir o denominador;

5.2 Fim antecipado 1

Nesta outra execução, com semente randômica `srand (0)`, o programa executa até o primeiro `retorne 1` (de `r1` ou `r2` inválidos):

```
10 17
1: 1/16 9/7 151/112 -137/112 9/112 7/144
2: 5/7 5/6 65/42 -5/42 25/42 6/7
3: 15 14/13 209/13 181/13 210/13 195/14
4: 2 5/4 13/4 3/4 5/2 8/5
5: INVALIDO 8/5 NUMERO INVALIDO
```

OBS.:

1. na quinta iteração, o `r1` seria `6/0`; ou seja, um número racional inválido. Então, em vez de se exibir o número inválido, a mensagem “INVALIDO” é exibida em seu lugar;
2. não confundir a mensagem “INVALIDO” acima com a mensagem “NUMERO INVALIDO” do pseudo-código (a mesma que está à direita de `8/5` no exemplo de execução);
3. consultar o arquivo `racionais.h` para mais detalhes e regras sobre a mensagem “INVALIDO”.

5.3 Fim antecipado 2

Na execução abaixo, com semente randômica `srand (4)`, o programa termina após encontrar o segundo `retorne 1` (da divisão inválida):

```
10 17
1: 7/5 5/4 53/20 3/20 7/4 28/25
2: 7/13 15/13 22/13 -8/13 105/169 7/15
3: 14/11 2/3 64/33 20/33 28/33 21/11
4: 1/2 13/6 8/3 -5/3 13/12 3/13
5: 10/7 15/11 215/77 5/77 150/77 22/21
6: 5/2 2/3 19/6 11/6 5/3 15/4
7: 9/16 0 DIVISAO INVALIDA
```

6 O que entregar

Entregue um único arquivo `tp2.tgz` que contenha por sua vez os seguintes arquivos:

- `racionais.h`: o mesmo arquivo fornecido, não o modifique;
- `racionais.c`: sua implementação do `racionais.h`;
- `tp2.c`: contém a função `main` que usa os `racionais`;
- `makefile`

Atenção: Não modifique em nenhuma hipótese o arquivo `racionais.h`. Na correção, os professores usarão o arquivo originalmente fornecido.

Bom trabalho!