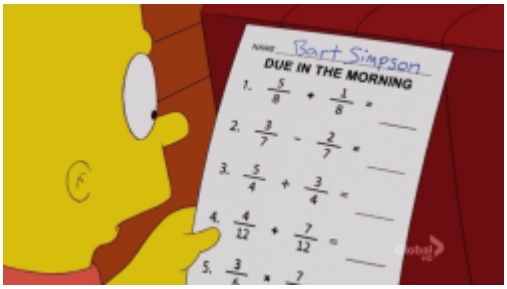
Números racionais

Este projeto foi inicialmente proposto pelo prof. Marcos Castilho para a disciplina de Programação 1 (CI1001).



Este trabalho tem como objetivo a implementação de um **Tipo Abstrato de Dados** (TAD) para números racionais, além de praticar o desenvolvimento de programas na linguagem C. A partir deste trabalho passaremos a entender como escrever programas que usam mais de um arquivo fonte.

Tipos Abstratos de Dados

Conceitualmente, um Tipo Abstrato de Dados (TAD) é uma abstração do dado, isto é, das informações armazenadas em memória. No caso deste trabalho, o TAD em questão são **números racionais** (também conhecidos como *frações*).

A principal característica de um TAD é que a maneira como se armazena o dado na memória não é relevante para que se possa manipular os dados. Em outras palavras, basta conhecer a abstração e ter disponível um conjunto de funções que manipulam os dados abstratamente. No caso de números racionais, seriam funções para:

- construir um número racional $^{a}/_{b}$, informando numerador e denominador
- simplificar um racional
- fazer operações aritméticas com racionais (+ - × ÷)
- comparar dois racionais (= < >)
- imprimir um racional
- ...

Na linguagem C é possível implementar este conceito de maneira elegante pela construção de um módulo escrito em arquivos separados que definem concretamente como um racional é implementado e que também contém as funções que manipulam efetivamente estes dados.

Uma vez que o módulo está pronto, é possível construir programas que usem este módulo e nos quais o dado está abstraído. Isto é, quem constrói o programa principal (*main*) não precisa (nem deve) conhecer a implementação concreta. Em outras palavras, se o módulo define que um número racional é uma *struct* com este ou aquele campo, quem implementa o programa principal não deve acessar os campos da estrutura, mas apenas usar as funções que a manipulam.

Consulte a página sobre organização de código para uma explicação com exemplos.

Atividade

Você deve implementar um programa que manipule números racionais, que são números da forma $^{a}/_{b}$, onde *a* e *b* são inteiros.

Você deve baixar este arquivo e abri-lo para poder fazer o trabalho, pois irá precisar de todos os arquivos ali contidos:

- `racional.h`: arquivo de cabeçalho com os protótipos das funções (**não deve ser alterado**).
- `racional.c`: arquivo que implementa as operações sobre números racionais ("esqueleto" a completar)
- `tp1.c`: código que usa a biblioteca de racionais ("esqueleto" a completar).
- `makefile`: arquivo do utilitário "make" para compilar seu código.

No arquivo `racional.h` foi definida uma estrutura (*struct*) para o tipo abstrato de dados *racional* e os protótipos das funções que permitem manipular essa estrutura. Você deve implementar essas funções no arquivo `racional.c`.

Este enunciado foi produzido pelo prof. Castilho para este projeto em sua versão inicial.

Números aleatórios

Em C, a geração de números aleatórios é usualmente feita usando as funções `srand` e `rand`, conforme o exemplo abaixo:

```
#include <stdlib.h>

int main ()
{
   int x ;

   srand (0) ; // inicia a semente do gerador de aleatórios com 0

   ...
   x = rand () % 1000 ; // gera um aleatório entre 0 e 999
   ...
}
```

Neste projeto, a função `rand()` será usada para sortear números dentro da biblioteca de racionais; a função `srand()` será usada no programa principal, para iniciar a sequência do gerador de aleatórios.

Programa principal

O programa principal (`tp1.c`) contém a função `main`. Ele deve incluir o *header* `racional.h` e ter um laço principal que implemente corretamente em C o seguinte pseudocódigo:

```
inicialize a semente aleatória (uma única vez em todo o código)
- sugestão: use "srand (0)" para facilitar os testes

leia um valor n   tal que 0 < n < 100
leia um valor max tal que 0 < max < 30

para todo i de 1 até n faça
  /* use um único espaço em branco separando números na mesma linha */
  imprima o valor de i seguido de um ":" e um espaço em branco

  sortear dois racionais r1 e r2
  - os numeradores e denominadores devem estar entre -max e +max

  imprima r1 e r2, na mesma linha e não mude de linha

  se r1 ou r2 for inválido, então
    imprima "NUMERO INVALIDO" e mude de linha
```

```
    retorne 1
  fim se

  calcule r1 + r2
  calcule r1 - r2
  calcule r1 * r2
  calcule r1 / r2

  se a divisão for invalida, então:
    imprima "DIVISAO INVALIDA" e mude de linha
    retorne 1
  fim se
  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
```

Para usar corretamente o conceito de TAD, o programa principal **NÃO DEVE ACESSAR** os campos internos do `struct racional`. Os acessos aos racionais devem ser **sempre** feitos usando as funções definidas em `racional.h` e implementadas em `racional.c`.

Exemplos de funcionamento

O fluxo de execução e a saída do programa variam em função dos dados de entrada (n e max) e do valor inicial do gerador de números aleatórios (semente). Para este projeto, a semente deve ser mantida sempre em zero (0).

a) Todos os passos do algoritmo

Considerando `n = 10` e `max = 6`, o programa faz todas as suas iterações:

```
10 6
1: -2 -5/2 -9/2 1/2 5 4/5
2: 1 -2 -1 3 -2 -1/2
3: -5 4/3 -11/3 -19/3 -20/3 -15/4
4: 3 -6/5 9/5 21/5 -18/5 -5/2
5: 3/4 4/5 31/20 -1/20 3/5 15/16
6: 1/2 1 3/2 -1/2 1/2 1/2
7: 3 4/3 13/3 5/3 4 9/4
8: -2 1/2 -3/2 -5/2 -1 -4
9: -5/3 1/2 -7/6 -13/6 -5/6 -10/3
10: -4 3/5 -17/5 -23/5 -12/5 -20/3
```

Observações:

1. observe, por exemplo, as iterações 2 e 8, onde há números racionais que foram simplificados no formato `VALOR/1`, deixando de exibir o denominador;

2. consulte o arquivo `racional.h` para mais detalhes e regras sobre as simplificações.

b) Fim antecipado 1

Nesta execução, com `n = 10` e `max = 11`, o programa executa até o primeiro `retorne 1`, com `r1` ou `r2` inválidos:

```
10 11
1: 0 5/9 5/9 -5/9 0 0
2: 5/11 -3/10 17/110 83/110 -3/22 -50/33
3: 0 6/5 6/5 -6/5 0 0
4: -5/6 -1/3 -7/6 -1/2 5/18 5/2
5: -4 -3 -7 -1 12 4/3
6: INVALIDO 9/8 NUMERO INVALIDO
```

Observações:

1. na sexta iteração, `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 pseudocódigo (a mesma que está à direita de `9/8` no exemplo de execução;
3. consulte o arquivo `racional.h` para mais detalhes e regras sobre a mensagem `INVALIDO`.

c) Fim antecipado 2

Nesta execução, com `n = 10` e `max = 5`, o programa termina após encontrar o segundo `retorne 1` (da divisão inválida):

```
10 5
1: 1/5 -1/3 -2/15 8/15 -1/15 -3/5
2: 4 -5 -1 9 -20 -4/5
3: 1/2 3/2 2 -1 3/4 1/3
4: 0 -2 -2 2 0 0
5: 4/5 3/5 7/5 1/5 12/25 4/3
6: 5/3 0 DIVISAO INVALIDA
```

Entregáveis

Entregue um único arquivo `tp1.tgz` que contenha por sua vez os seguintes arquivos:

- `racional.h`: o mesmo arquivo fornecido, **não o modifique**
- `racional.c`: sua implementação das funções definidas em `racional.h`
- `tp1.c`: contém a função `main` que usa os números racionais
- `makefile`

Importante:

- Use boas práticas de programação, como endentação, bons nomes para variáveis, comentários no código, bibliotecas, defines, etc. Um trabalho que não tenha sido implementado com boas práticas vale zero.
- Na dúvida, não tome decisões sobre a especificação, pergunte!
- Dúvidas podem e devem ser resolvidas durante as aulas.

- Dúvidas com relação a este enunciado devem ser solucionadas via e-mail para `prog1prof@inf.ufpr.br`, assim todos os professores da disciplina receberão os questionamentos.