Trabalho prático de CI1001
TP3
Departamento de Informática/UFPR

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 tp3, deverá estar no subdiretório tp3;
- 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 tp3;
  - Então seu tgz deve ser criado assim:
    - Estando no diretório tp3, faça:
    - `cd ..`
    - tar zcvf tp3.tgz tp3
  - Desta maneira, quando os professores abrirem o tgz (com o comando tar zxvf tp3.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 mais uma vez o Tipo Abstrato de Dados (TAD) para números racionais feito no TP1 e TP2 para exercitarmos alocação dinâmica.

O programa principal é mais parecido com o do exercício com vetores, mas desta vez é necessário cuidar dos processos de alocação dinâmica, basicamente, alocar e liberar espaços, além de termos alguns desafios a mais para não esquecermos que esta disciplina também é de algoritmos!

Nesta fase do aprendizado, a ferramenta `valgrind` ajuda a detectar vazamentos de memória (leaks), além de outros erros cometidos, como variáveis não inicializadas, etc.

Assim, são objetivos deste trabalho a prática dos seguintes conceitos:

- Alocação dinâmica de structs e de vetores;
- Manipulação de ponteiros;
- Uso da ferramenta `valgrind`.

3 O trabalho

Você deve reescrever a sua implementação do arquivo `racionais.c` conforme o novo arquivo `racionais.h` fornecido. A diferença básica está no fato das funções retornarem ponteiros para racionais (ponteiros para as structs) e não as structs propriamente ditas.

Você deve baixar o tp3.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.

tp3.c: um esqueleto de arquivo tp3.c.

casos_de_teste: um diretório com um conjunto de entradas e saídas para fins de testes.

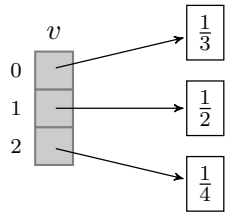
testa.sh: um script shell para testar o seu programa.

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

Você deve implementar um programa que manipule ponteiros para números racionais, que são números da forma $\frac{num}{den}$, onde *num* e *den* são números inteiros.

Inicialmente, você vai alocar dinamicamente um vetor de ponteiros para números racionais. Em seguida, você vai inicializar o vetor com ponteiros para números racionais lidos a partir do teclado e vai inserir estes ponteiros, na mesma ordem da leitura, no vetor.

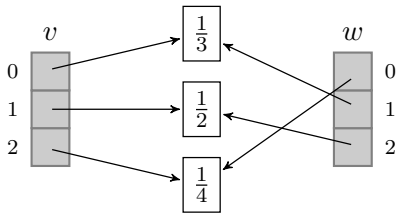
A título de exemplo, considere a figura abaixo. Pode-se ver um vetor v contendo três elementos (índices de 0 a 2). O exemplo mostra que foram lidos, nesta ordem, os números racionais $\frac{1}{3}$, $\frac{1}{2}$ e $\frac{1}{4}$. O vetor então contém ponteiros que apontam para estes racionais nas posições 0, 1 e 2 do vetor.



Agora seu programa deve manipular este vetor para eliminar os inválidos e em seguida ordená-lo em ordem crescente.

A ideia é que a *struct* pode ser grande e não queremos ficar trocando estas de lugar, só queremos movimentar ponteiros, que custa bem menos.

Considere na figura abaixo que o vetor w é o vetor v ordenado, a ilustração é para fins didáticos apenas. Na figura, o vetor w, quando percorrido do índice 0 até o índice 2, permite ver os racionais ordenados, isto é, as posições 0, 1 e 2 do vetor w apontam respectivamente para os racionais $\frac{1}{4}$, $\frac{1}{3}$ e $\frac{1}{2}$, isto é, estão ordenados.



Como último desafio, você deve imprimir a soma de todos os racionais apontados a partir do vetor. Este algoritmo é simples, o complicado aqui é cuidar da liberação de memória.

- 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!
- Não mande mensagens pelo moodle, os professores nem sempre estão logados na plataforma. As responstas serão mais rápidas se as mensagens vierem no email acima.
- Dúvidas podem e devem ser resolvidas durante as aulas.

4 Seu programa

No arquivo `racionais.h` foi definida uma nova interface para o tipo abstrato de dados *racional*. 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 ter um laço principal que implemente corretamente em *C* o seguinte pseudo-código:

```
use srand (0)

leia um n tal que 0 < n < 100
crie um vetor de n posicoes contendo ponteiros para numeros racionais
    - Os racionais deverao ser inicializados com valores lidos do teclado
    - Este vetor tambem deve ser alocado dinamicamente
imprima os racionais apontados pelos elementos do vetor
elimine deste vetor os racionais invalidos
imprima o vetor resultante
ordene este vetor
imprima o vetor ordenado
calcule e imprima a soma de todos os racionais apontados pelo vetor
ao final, mude de linha

retorne 0
```

Seu programa deve liberar toda a memória alocada:

- todos os racionais;
- o vetor;
- o espaço utilizado para fazer o cálculo da soma.

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 Exemplo de entrada e saída

Nos casos de teste disponibilizados no arquivo `tp3.tgz` usamos `srand (0)`. A saída será diferente se for utilizada outra semente randômica.

Existem 5 arquivos de entrada e as respectivas saídas esperadas para cada entrada. É evidente que os números nos nomes dos arquivos definem a relação correta entre o arquivo de entrada e o de saída. Por exemplo, a entrada_1.txt deve ser combinada com a saida_1.txt e assim por diante.

6 Arquivo de teste

Disponibilizamos um script shell que visa testar seu programa. Neste script fazemos uso pipes combinado com o comando "`diff`", o qual faz a comparação da entrada com a saída.

O uso do script é:

- `./testa 1`: teste inicial, não usa o valgrind, serve para você ver se a lógica do seu programa está correta. Se a saída do script for vazia é porque seu programa está correto para os 5 casos de entrada fornecidos.
- `./testa 2`: após você ter sucesso no teste 1, use este teste para que o valgrind aponte demais erros, não apenas os vazamentos de memória que seu programa tem como também outros erros que ele encontra. Aqui a saída não será vazia.

Observação importante: pode ser que você não consiga concluir o teste 1 por causa de problemas mais graves, como segmentation fault ou outras coisas ruins. Neste caso você talvez queira rodar o teste 2 até encontrar o problema.

Se o seu programa não apresentar vazamentos de memória, você deverá receber a mensagem:

`All heap blocks were freed -- no leaks are possible`.

Caso contrário, seu programa tem vazamentos e as mensagens de erro deverão indicar a causa.

Outro tipo de mensagens de erro que podem ocorrer são do tipo:

`Invalid read of size 8` ou outras coisas estranhas. Significa que seu programa tem erros mais graves, embora talvez possa ter passado eventualmente pelo teste 1.

Os professores podem ajudar a entender as mensagens do valgrind!

7 O que entregar

Entregue um único arquivo `tp3.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`;
- `tp3.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.

8 Recomendação

Faça o programa aos poucos, construa um `main` mínimo que apenas inicializa as estruturas e as imprima. Depois vá modificando aos poucos o seu `main`, sempre garantindo que até o ponto anterior estava tudo funcionando perfeitamente.

Faça em seguida a eliminação dos inválidos, já que você deve ter implementado esta função no exercício de aquecimento passado. Teste e garanta que funcione.

Em seguida pode ordenar e imprimir.

Deixe por último a soma dos elementos, ela não é trivial por causa da liberação de memória.

Ao desenvolver, faça uso de `printfs` para ajudar a depurar o código.

Uma dica importante: quando ocorre `segmentation fault`, nem sempre o `printf` funciona. Isso ocorre pois as impressões não ocorrem imediatamente, elas vão para um buffer que pode não ser esvaziado quando ocorre o `segmentation fault`.

Neste caso, faça uso da função `fprintf`, que faz com que a saída seja impressa em um arquivo. A letra "`f`" antes de `printf` vem de *file* e significa imprimir em arquivo. A recomendação é imprimir na saída padrão de erros, a `stderr`. Assim, você pode substituir um

```
printf ("blabla", lista de variaveis);
```

por:

```
fprintf (stderr, "blabla", lista de variaveis);
```

Bom trabalho!