UFES – DI – CT
Semestre 2020/2
Disciplina Tópicos em Programação I
Professor Saulo Bortolon

Trabalho Computacional
Parte 1: Racionais, Complexos e Vetores

Números Racionais -

1) Construa um TAD de números racionais. Números racionais podem ser definidos pela divisão de dois números inteiros (numerador e denominador). Por exemplo, os números 1/3 e 13/7. O sinal do número "racional" será determinando pelo produto do sinal do numerador e do denominador, conjuntamente.


```
typedef struct Racional_st
        {       long int num;
                long int den;
        }  Racional_t;

typedef struct Racional_st Racional_t;
```

apresentando como interface o ponteiro para um número racional

```
typedef struct Racional_st * Racional_pt;
```

A) Utilize a ferramenta make para gerenciar o processo de construção (build), isto é: a compilação, link-edição usando a ferramenta GNU Make tool. Um conjunto de testes será passado posteriormente.

B) Utilize o Algoritmo recursivo de Euclides para encontrar o maior divisor comum dos números. Isso é necessário para garantir que os números sejam sempre os menores possíveis (200/600 é "equivalente" a 1/3). Você pode encontrar o algoritmo procurando na Web.. Comece, por exemplo, nos seguintes sites:
   - wikipedia - "Euclidean Algorithm"
   - The Prime Glossary on the Euclidean Algorithm

C) Implemente os seguintes métodos como parte deste TAD:
   - cria um número racional
   - destrói um número racional
   - copia um número racional em outro número racional
   - compara dois números racionais (se a=b, resultado = 0; se a<b, resultado <0; se a>b, resultado >0)
   - verifica se dois números racionais são equivalentes
   - verifica se o denominador é zero (x/0)
   - verifica se o número é zero (0/x)
   - verifica se ambos (numerador e denominador) são zero (0/0)
   - soma dois racionais, gerando um terceiro (c = a+b)

- acumula um racional “b” a um primeiro racional “a” (a += b)
- subtrai dois racionais, gerando um terceiro
- multiplica dois racionais, gerando um terceiro
- multiplica um racional “b” a um primeiro racional “a” (a *= b)
- divide dois racionais, gerando um terceiro
- obtém o quadrado de um número racional
- obtém a raiz de um número racional [1]
- converte um número racional em um número real “*double*”
- converte um número *double* em um número racional[2]
- verifica se um número racional pode ser convertido em um número inteiro (considerando uma tolerância eps = 0.00001)
- escreve um número racional em um arquivo de texto no formato CSV (termos separados por vírgula).
- lê um arquivo racional de um arquivo de texto no formato CSV (termos separados por vírgula). Atente que um número racional não pode ter denominador nulo. Caso o denominador não exista no arquivo, assume-se valor do denominador é igual a “1”. Caso nem o denominador nem o numerador existam no arquivo, é porque o arquivo “terminou” ou “estava vazio”. Não é “*zero sobre zero*”.

D) Teste mínimo:
   - obtenha pelo menos um número racional no intervalo 3.1414 - 3.1416
     (dica: veja séries que geram o “*pi*” (veja exemplos de séries que geram o valor pi em www.davidhbailey.com/dhbpapers/pi-quest.pdf)
   - obtenha o seguinte resultado como um número racional:
     $3 + (8/60) + (29 / 60^2) + (44/60^3)$
   - converta os resultados acima em *double* e em *float*
   - *um arquivo de leitura com operações de teste será passado para vocês posteriormente*

---

1 Esta é mais complicada do que parece. Você pode ver um artigo a respeito em https://www.reidatcheson.com/coq/formalized%20mathematics/real%20analysis/2017/12/14/square-root-of-rational-numbers.html). Basicamente, temos uma aplicação do Algoritmo de Newton para resolver equações não-lineares. Assusta?

2 Esta também é mais complicada do que parece… você pode encontrar o código pronto em https://rosettacode.org/wiki/Convert_decimal_number_to_rational#C …. mas vai ter que entender o que ele está fazendo...

Números Complexos

2) Construa um TAD número complexo - em 3 versões - que utilizem para representar seus componentes real e imaginário, respectivamente:

a) números “long *int*”
b) números “double”
c) números racionais (cujo TAD foi desenvolvido na questão anterior) .

Os seus TADs “Complexo” serão, portanto:

```
==========================================
        typedef struct I_Complexo_st
                {       long int  * real;
                        long int  * imag
                } I_Complexo_t;
        typedef struct I_Complexo_st * F_Complexo_pt;
==========================================
        typedef struct D_Complexo_st
                {       double  * real;
                        double  * imag
                } D_Complexo_t;
        typedef struct D_Complexo_st * D_Complexo_pt;
==========================================
        typedef struct R_Complexo_st
                {       Racional_pt  * real;
                        Racional_pt * imag
                } R_Complexo_t;
        typedef struct R_Complexo_st * R_Complexo_pt;
==========================================
```

Note que s componentes real e imag não são números. Elas são ponteiros para números.

Desenvolva os TADs de forma que você não precise reescrever todos os três códigos a cada vez que fizer uma “manutenção” no código. Para isso, você pode usar os métodos aprendidos na sala de aula, usando “templates-macro” do pré-processador :

```
#define TIPO_   int  ;
#define TIPADO_(Coisa)  I_Coisa
#define  FORMATO_  “%d”
```

DICA: Faça todos os métodos para um dos “tipos” e teste. Mas já faça isso definindo o “TIPO_” genérico. Depois, generalize os métodos para os outros dois usando “TIPADO_ (Coisa)”. Acredite: vai te dar menos trabalho!!!

Estes 3 TADs deverão ter pelo menos os seguintes métodos:

Criação, destruição e cópia

- criação de um número complexo
- destruição de um número complexo
- atribuir um número complexo a outro que já "*existe*" (isto é, que já foi "*criado*" antes)
- copiar um número complexo em outro que não foi "*criado*" ainda
- converter um número complexo de um "tipo" em outro:
    - D_Complexo_pt em I_Complexo_pt (*perigos*: arredonde para o mais próximo; cuidado com overflow e underflow)
    - I_Complexo_pt em D_Complexo_pt (aqui o "perigo" é menor)
    - D_Complexo_pt em R_Complexo_pt (também perigoso…)
    - R_Complexo_pt em D_Complexo_pt (também perigoso…)
    - R_Complexo_pt em I_Complexo_pt (também perigoso…conversões, overflow e underflow)
    - I_Complexo_pt em R_Complexo_pt (não há tantos perigos…)

Comparações complexas

- verificação se o módulo de um número complexo é zero (admitindo uma tolerância eps = 0.00001 para o valor do módulo)
- verificação se um número complexo é apenas real (admitindo uma tolerância eps = 0.00001 para o valor da parte imaginária)
- verificação se um número complexo é apenas imaginário (admitindo uma tolerância eps = 0.00001 para a parte real)
- comparar se um número complexo é igual, menor ou maior que outro (com relação ao seu módulo), considerando uma tolerância eps = 0.0001
- comparar se um número complexo é igual, menor ou maior que outro (com relação ao seu ângulo), considerando uma tolerância eps = 0.0001

Acessando/atribuindo valor às componentes de um número complexo

- retornar a parte real de um número complexo
- retornar a parte imaginária de um número complexo
- retornar a magnitude de um número complexo (módulo)
- retornar a fase de um número complexo (ângulo)
- atribuir novo valor à parte real de um número complexo representado por suas coordenadas
- atribuir novo valor à parte imaginária de um número complexo representado por suas coordenadas
- atribuir novo valor à magnitude de um número complexo (módulo) mantendo seu ângulo
- atribuir novo valor à fase de um número complexo (ângulo) mantendo o seu módulo

Operações

- calcular o conjugado de um número complexo
- somar, subtrair, dividir e multiplicar dois números complexos gerando um terceiro número complexo (são quatro métodos diferentes, sendo que os pares de números complexos são do mesmo "tipo" e geram um valor também o mesmo "tipo". Atenção para overflow e underflow. Ambos devem ser tratados como "exceção" e devolver números NAN – *Not A Number*.) ;
- "acumular" somando (a += b) e multiplicando (a *= b) dois números complexos (duas operações diferentes, mas a e b são do mesmo "tipo" de número complexo)
- fazer um método que executa uma das seis operações acima, sempre considerando que a-b-c são números complexos do mesmo "tipo", dependendo da string "operação" recebida como argumento. Operação pode ser "+", "-", "*", "/","+=", "*=". Por exemplo:

```
D_Complexo_pt  OperacaoComplexo (D_Complexo_pt      valor1,
                                 D_Complexo_pt      valor2,
                                 char *             oper);
```

Por exemplo, :

```
D_Complexo_pt a, b, c;
D_Criar_Complexo (1,2);
D_Criar_Complexo (3,4);
c = OperacaoComplexo (1,b,"+");
D_Exibir_Complexo (c);
```

o resultado será :

```
4 + 6i
```

Persistência (leitura e gravação em arquivos)

- escreve um número complexo em um arquivo CSV (veja formatos típicos abaixo)
- lê um número complexo de um arquivo CSV (veja formatos típicos abaixo):

Formatos típicos de números complexos em um arquivo CSV:

11+12i, 13.11, 14.22i, 15-16i, -17+18i, - 19 i,

Formatos atípicos de números complexos usando números racionais[3]

1/2 + 3/4 i, 1 / 2 + 3 / 4 i, 1/2 + 3/4 i, 1/2 + 3/4 i, 1/2 + 3/4 i,

Note que podem haver espaços em branco entre os dígitos, os sinais e as vírgulas. Podem haver espaços em branco até mesmo entre os dígitos de um número (123 = 12 3). **Os números são separados por vírgulas. Espaços em branco são desprezados.**

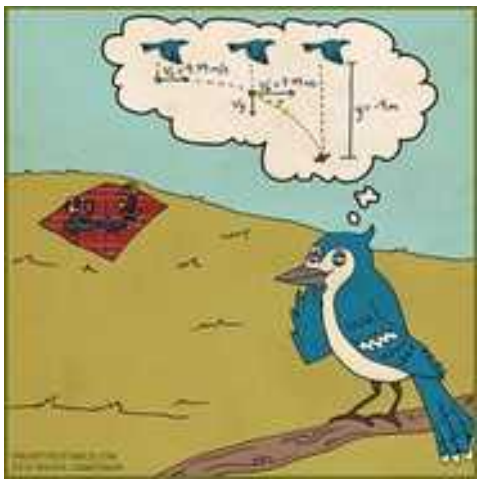
Para simplificar inicialmente nosso exercício, você pode assumir que um número complexo tem ambos os termos do mesmo tipo. Não haverá mistura de um real inteiro com um imaginário *double*, ou de um real *double* com um imaginário racional. Também para simplificar, assuma que não ocorrerão duas "vírgulas seguidas" (isto é: um número nulo tem que ser escrito como 0.000).

Se, preferir fazer uma versão mais completa em que estas misturas sejam permitidas, na leitura de número em que os dois termos não forem do mesmo tipo, converta um deles para o tipo de maior hierarquia:

inteiro → racional → double

3 **(*já que nós somos irracionais*….)**

Vetores



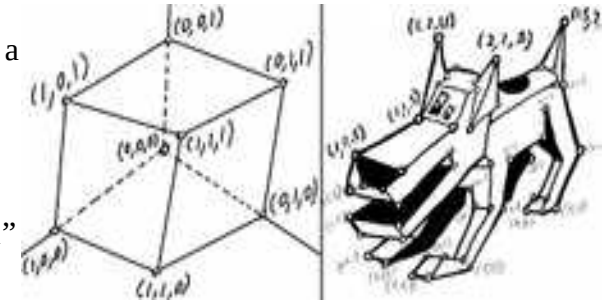
Crie um TAD vetores que possa operar sobre números **inteiros, double ou complexos.**

Neste caso aqui, podemos esquecer os “racionais”. Eles já cumpriram o papel deles.

O TAD que você vai criar deve ter os seguintes “campos” / “atributos”:

- n = número atual de elementos;
- N = número máximo de elementos ;
- i = índice do elemento “atual”;
- elementos, propriamente ditos, acessíveis em sequência.

O número atual de elementos é a quantidade de elementos contidos (no momento) dentro do vetor. Necessariamente é menor do que o “número máximo de elementos”.



O número máximo de elementos deve ser aumentado (multiplicado por 2) se o número atual de elementos chegar a igualá-lo. Ele começa com um valor igual a 100. Deve também ser reduzido pela metade se o número atual de elementos “n” baixar para menos do que N/4. Por exemplo, se n baixar de 4000, reduz-se N de 16000 para 8000.
O índice do elemento atual “i” é “incrementado de 0 até n-1” ao se percorrer os elementos do vetor do “início” para “fim” , e é “decrementado de n-1 até 0” ao se percorrer o vetor da posição “fim” até a posição “início”.

O TAD que você vai criar deve ter os seguintes métodos:

- criar o vetor
- destruir o vetor
- atribuir uma cópia do vetor a outro vetor já existente;
- criar uma cópia do vetor, criando um novo vetor

- devolver o número atual de elementos;
- devolver o número máximo de elementos;
- devolver o índice do elemento atual;
- devolver o primeiro elemento e posicionar “i” na primeira posição);
- devolver o próximo elemento (posicionando “i” nesta posição);
- devolver o elemento anterior (posicionando “i” nesta posição);
- devolver o último elemento e posicionar “i” na última posição;
- devolver o elemento da i-ésima posição (lá posicionando o índice “i”);
- atribuir o valor “v” na i-ésima posição do vetor, posicionando ali o índice “i”;
- atribuir o valor “v” depois da última posição do vetor, incrementando o “número atual de elementos”;
- eliminar a i-ésima posição do vetor, devolvendo o seu valor atual e decrementando o “número atual de elementos”
- eliminar todos os elementos do vetor;

- obter o valor do elemento de maior módulo existente no vetor (e posicionar o índice atual naquela posição). Em caso de empate, qualquer uma das posições serve;
- obter o valor do elemento de menor módulo existente no vetor (e posicionar o índice atual naquela posição). Em caso de empate, qualquer uma das posições serve;
- determinar quantos dos elementos do vetor têm valor igual a “v” (considerar uma tolerância eps = 0.0001)
- devolver um vetor com as posições dos elementos que têm valor igual a “v” (considerar uma tolerância eps = 0.0001)
- ordenar os elementos do vetor de acordo com uma função “critério de comparação”.
    - A função “critério” recebe dois elementos do vetor e os compara: se a=b, retorna “0”; se a<b, retorna -1; se a>b, retorna +1. A função “critério” é um dos argumentos deste método de ordenação;
- intercalar dois vetores ordenados, criando um novo vetor igualmente ordenado, usando uma função “critério” que igualmente um argumento deste método de intercalação ;

- somar dois vetores de mesmo tamanho criando um novo vetor (C = A + B);
- multiplicar dois vetores (“produto interno”) gerando um valor “produto” (c = A . B)
- subtrair um vetor de outro vetor, criando um novo vetor; (C = A – B);
- multiplicar um vetor por um escalar, alterando os valores do vetor original (A = k * A)
- acumular um vetor B sobre um outro vetor A do mesmo tamanho, somando os elementos um-a-um (A = A+ B)

- determinar a média aritmética dos valores de um vetor[4] $\bar{x} = \frac{\sum x_i}{n}$
- determinar a variância dos valores de um vetor $s^2 = \frac{\sum_{i=1}^{n} (X_i - \bar{X})^2}{n-1}$
- determinar o desvio padrão dos valores de um vetor (raiz quadrada da variância)

- determinar a mediana dos valores de um vetor (o valor que separa os 50% maiores dos 50% menores. (referência https://pt.wikipedia.org/wiki/M%C3%A9dia) . Este algoritmo aparenta ser, enganosamente, fácil. (OBS: para números complexos há outros algoritmos e medidas mais importantes, principalmente na área de processamento de sinais)
    - Uma implementação “burra”: ordena o vetor e pega o elemento central ou a média dos dois elementos centrais;
    - uma implementação mais esperta é oferecida por Niklaus Wirth[5] e uma implementação está disponível em http://ndevilla.free.fr/median/median/index.html . Lá há outras implementações igualmente disponíveis.

4 A média aritmética de um conjunto de números complexos define o “centróide” de n partículas idênticas espalhadas sobre o plano

5 Niklaus Wirth, Algorithms + Data structures = Programs (p 84).

Trabalho Computacional
Parte 2: Polinômios, Quaternions e Matrizes

Polinômios

3) Criar um conjunto de TADs de polinômios:

$$P(x) = a0 + a_1.x + a_2.x^2 + \ldots + a_n.x^n$$

Os coeficientes e o valor de x podem ser de seis tipos diferentes:

- inteiros long int,
- double,
- racional de long int,
- complexo de inteiros,
- complexo de double,
- complexo de racionais long int).

Para cada um dos 6 tipos, tem-se um TAD diferente.

Um exemplo de TAD para polinômios do tipo "double" pode ser encontrado em https://github.com/fabiomaia/libpolynomial.

Você deve desenvolver os seguintes métodos para cada um destes 6 TADs:

- criar um novo polinômio
- copiar um polinômio a outro que ainda não foi criado
- atribuir um polinômio a outro que já foi criado antes
- destruir um polinômio
- "devolver" o grau de um polinômio
- aumentar o grau de um polinômio (atribuindo zeros aos coeficientes da alto grau acrescentados)
- diminuir o grau de um polinômio (retirando os membros de alto grau que tenham coeficiente igual a zero)
- "devolver" o n-ésimo coeficiente de um polinômio
- atribuir um valor ao n-ésimo coeficiente de um polinômio
- determinar se um polinômio é "zero" (todos os coeficientes nulos), considerando uma tolerância fixa eps = 0.0001
- determinar se um polinômio é igual a outro (tolerância fixa eps = 0.0001)
- avaliar um polinômio (dica: procure na wikipedia por "*Horner's rule*")
- adicionar dois polinômios gerando um terceiro (c = a + b)
- subtrair dois polinomios um do outro, gerando um terceiro (c = a - b)
- multiplicar dois polinômios gerando um terceiro (c = a * b)
- multiplicar um polinômio por uma constante (b = k * a). A constante é do mesmo tipo dos coeficientes do polinômio.
- Determine o polinômio-derivada P'(x) de um polinômio P(x)
- Determine o polinômio-integral P(x) de um polinômio P´(x)

- gravar um polinômio em um arquivo no formato CSV (ver descrição abaixo)
- salvar um polinômio em um arquivo no formato CSV (ver descrição abaixo)

• Desafios para os fortes de coração [6]:

Os seguintes programas devem ser resolvidos apenas para polinômios do tipo double e complexo de doubles. Não resolver para coeficientes inteiros ou "racionais". Sua resolução não é obrigatória, mas conta pontos extras (0.5 ponto extra por problema). Não tente descobrir a roda. Use algoritmos que outros já desenvolveram. E não se iluda: alguns dos problemas são estudados por pesquisadores até hoje (porque são importantes…[7]).

- Fatorar um polinômio por outro da forma (x-P), obtendo o resto R, onde x, P e R são variável e coeficientes (respectivamente) do mesmo tipo do polinômio P(x);
- fatorar um polinômio por outro de grau 2 (x2 – Qx + P), obtendo o polinômio resto da divisão (procurar por *Euclidean division*")
- dividir um polinômio p(x) por outro g(x), de grau menor, com um resto r(x)
- ajustar um polinômio p(x), de grau N a um conjunto de (N+1) pontos (você terá que usar matrizes e vetores)
- ajustar um polinômio p(x), de grau n < N, a um conjunto de (N+1) pontos pelo critério dos mínimos quadrados (você gerá que usar matrizes e vetores)
- calcular todas as raízes de um polinômio[8]

6 Pessoal, estas coisas podem ficar extremamente complicadas. Não se iludam. Tanto o código quanto a matemática em si. As coisas complicam principalmente com os números complexos e racionais. Vejam, por exemplo, a biblioteca FLINT. Ela está disponível em http://www.flintlib.org/links.html. Por exemplo, este é o link para o algorimo de adição de dois polinômios. https://github.com/BrianGladman/flint/blob/trunk/fq_poly_templates/add.c Vejam que vocês agora já sabem o que é um "template".

7 Faça, por exemplo, esta pesquisa no scholar google: https://scholar.google.com/scholar?as_ylo=2017&q=polynomial+roots+finding&hl=pt-BR&as_sdt=0,5

8 Há várias formas de fazê-lo. Veja, por exemplo, o algoritmo de Durand-Kerner. https://en.wikipedia.org/wiki/Durand%E2%80%93Kerner_method

- # Formato CSV para para polinômios

Neste exercício, assuma que o arquivo é “bem comportado”, isto é:

- que não haverão linhas em branco no meio do arquivo
- que não faltarão coeficientes (números nulos serão escritos como 0, 0.000 ou 0/1)
- que não haverão duas “,” seguidas,
- que cada um dos polinômios será escrito em uma única linha.
- que cada polinômio tem todos seus coeficientes do mesmo tipo
- o sinal “decimal” é o ponto e a vírgula separa dois números diferentes
- o primeiro termo de cada linha é um número inteiro que descreve o grau do polinômio
- os demais termos na mesma linha podem ser inteiros (long int), reais (double) ou racionais (long int / long int)
- uma mesma linha não mistura coeficientes de diferentes tipos. Isto é:
  - ninguém nunca vai escrever um número racional dentre os coeficientes de um polinômio em que os outros coeficientes são double
  - ninguém nunca vai escrever um número inteiro dentre os coeficientes de um polinômio em que os outros coeficientes são double
  - ninguém nunca vai escrever um número inteiro dentre os coeficientes de um polinômio em que os outros coeficientes são racionais

Contudo, podem ocorrer as seguintes anomalias:

- poderão ou não haver espaços ou tabs entre os dígitos de um número;
- poderão ou não haver espaços ou tabs entre os dígitos de um número e os sinais “.” (ponto), “+”, “-”, “/” e “,” (vírgula)

linha 0 - cabeçalho:
Grau, Coeficientes
linhas típicas:

4 ,	1.111,	2.222,	3.333,	4.444,	5.555
3,	1.111,	2.222,	3.333,	4.444	
2,	111 / 123,	222 / 456,	333 / 789,		
2,	1.111 + 1.111 i,	2.222 - 2.222 i,	3.333 + 3.333 i		
2,	1.111, - 1.111 i,	2.222 + 2.222 i,	3.333 - 3.333 i		
2,	11/12 + 22/34 i,	33/56 - 44/78 i,	55/910 + 66/1112 i		

Representam os seguintes polinômios, respectivamente:

$1.111 + 2.222\ x^1 + 3.333\ x^2 + 4.444\ x^3 + 5.555\ x^4$ onde x é do tipo float

$1.111 + 2.222\ x^1 + 3.333\ x^2 + 4.444\ x^3 + 5.555\ x^4$ onde x é do tipo double

$(111/123) + (222/456)\ x^1 + (333/789)\ x^2$ onde x é do tipo “racional”

$(1.111 + 1.111\ i) + (2.222 + 2.222\ i)\ x^1 + (3.333 + 3.333\ i)\ x^2$
onde x é do tipo complex float

$(1.111 + 1.111\ i) + (2.222 + 2.222\ i)\ x^1 + (3.333 + 3.333\ i)\ x^2$
onde x é do tipo complexo double

$((11/12) + (22/34)\ i) + ((33/56) + (44/78)\ i)\ x^1 + ((55/910) + (66/1112)\ i)\ x^2$
onde x é do tipo complexo racional

Matrizes:

(em redação)
Nesta parte, faremos uso de matrizes uni, bi e tri-dimensionais. Não trabalharemos mais com os números complexos e racionais, contudo.
Será um uso mais “comercial”.