UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA
INF09416 - Tópicos Especiais em Programação I - Turma: 01 Earte 2020/1

EXERCÍCIOS DE AVALIAÇÃO

1) [VALOR 3 PONTOS] Implemente um tipo abstrato de dados para operações em números complexos da forma
p = a + bi , onde a e b são números reais (*double*).

Construa os arquivos ".h" (interface) e ".c" implementação que permitam a execução do seguinte teste:

```
include <stdio.h>
include "complexo.h"
int main ( )
{
        complexo_t      *p, *q,
                        vet1[10], vet2[],
                        soma, produto, divisao, subtracao,
                        produtorio, somatorio;
        unsigned int i;
        long double modulo;

        CrieNumeroComplexo (p);
        CrieNumeroComplexo (q);
        CrieVetorComplexo (vet2, 10);

        PreencheComplexoDeReais (p, 1.0 , 2.0);
        PreencheComplexoDeReais (q, 3.0 , 4.0);

        soma            = SomaComplexos (p, q);
        produto         = ProdutoComplexos (p,q);
        subtracao       = SubtracaoComplexos (p,q);
        divisao  = DivisaoComplexos (p,q);
        modulo = ModuloComplexo (p)

        for (i = 0; i < 10; i++)
        {
                PreencheComplexoDeReais (&(vet1[i]), double(i) , double(i));
                PreencheComplexoDeReais (&(vet2[i]), double(i) , double(i));
        }

        produtorio = ProdutoInternoVetComplexo (vet1, vet2, 10);
        somatorio  = SomaElementosVetComplexo (vet1, 10)

        ApresentaComplexo (soma, "p + q = ");
        ApresentaComplexo (produto, "p * q = ");
        ApresentaComplexo (divisao, "p / q = ");
        ApresentaComplexo (subtracao, "p + q = ");
        ApresentaComplexo (produtorio, "vet1 . vet2 = ");
        ApresentaComplexo (somatorio, "soma elementos de vet1 = ");
        printf ("O modulo de p eh %e", modulo);
}
```

Atenção para condições de erro: falta de memória para alocação, divisão por zero

2) [VALOR 4 PONTOS] Altere a solução da questão 1 de forma que você possa criar (NO MESMO PROGRAMA) tanto números complexos de *long double* quanto números complexos de *double*, usando as facilidades do pré-processador. Para isso, utilize as macros genéricas como definidas em https://rebelsky.cs.grinnell.edu/musings/cnix-macros-generics e que foram estudadas em aula.

Os nomes dos tipos gerados serão:

para complexos de double:	Complexo_t_D
para complexos de long double:	Complexo_t_Ld

E o teu programa principal teria algo parecido com o seguinte, no cabeçalho (“includes...”):

```
#undef TYPE
#undef TYPED

#define TYPE double

#define TYPED(THING) THING ## _D

#include "generic-complexo.h"

// --------------------------------

#undef TYPE

#undef TYPED

#define TYPE long double

#define TYPED(THING) THING ## _Ld

#include "generic-complexo.h"
```

O main( ) terá execução parecida com:

```
include <stdio.h>
include “complexo.h”
int main ( )
{
        complexo_t_D        *p, *q,
                            soma1, produto1, divisao1, subtracao1;

        complexo_t_Ld       *r, *s,
                            soma2, produto2, divisao2, subtracao2;

        CrieNumeroComplexo_D (p);
        CrieNumeroComplexo_D (q);

        PreencheComplexoDeReais_D (p, 1.0 , 2.0);
        PreencheComplexoDeReais_D (q, 3.0 , 4.0);
        soma1               = SomaComplexos_D (p, q);
        produto1    = ProdutoComplexos_D (p,q);
        subtracao1  = SubtracaoComplexos_D (p,q);
        divisao1    = DivisaoComplexos_D (p,q);

        CrieNumeroComplexo_Ld (p);
        CrieNumeroComplexo_Ld (q);

        PreencheComplexoDeReais_Ld (p, 1.0 , 2.0);
        PreencheComplexoDeReais_Ld (q, 3.0 , 4.0);
        soma2       = SomaComplexos_Ld (p, q);
        produto2    = ProdutoComplexos_Ld (p,q);
        subtracao2  = SubtracaoComplexos_Ld (p,q);
        divisao2    = DivisaoComplexos_Ld (p,q);

        ….
}
```

Pergunta extra: se você tiver que adicionar funções que convertem números complexos de um tipo em outro, como você faria ? Por exemplo:

```
        Complexo_t_D        valor1;
        Complexo_t_Ld       valor2;

        valor2 = ConverteEmComplexoLong_D (valor1);

        valor1 = ConverteEmComplexoDouble_Ld (valor2);
```

Para que estas funções seriam úteis? Como você trataria casos de underflow e overflow, nestas funções? Não precisa implementar. Pense a respeito.

3) [VALOR 5 PONTOS] Altere a implementação da questão 1 de forma que você possa utilizar funções chamadas a partir de tabelas de funções virtuais. Assim, você precisará criar:

- **uma interface ".h" e uma implementação ".c" para números complexos genéricos**
- **uma interface ".h" e uma implementação ".c" para números complexos baseados em double**
- **uma interface ".h" e uma implementação ".c" para números complexos baseados em long double**

O esquema de criação de números complexos "genéricos" adotado nesta questão é o mesmo apresentado em sala a partir da apostila *"Application Note: Object-Oriented Programming in C"*[1],

Em cada uma destas três interfaces e implementações, devem ser criadas as funções:

- **bool construtor (me)** : aloca espaço de memória para a parte real e a parte imaginária do número complexo ("me" é um ponteiro para o número complexo, nesta e nas próximas funções). Retorna falso se não conseguiu criar.
- **bool destrutor (me)**: libera a área de memória alocada para o número complexo. Retorna falso se não conseguiu destruir.
- **Complexo_? *outro = copia_? (me)**: copia um número complexo em outro, do mesmo tipo "?", "criando" o "outro". Se "outro" já existia, deve liberar a área de memória anteriormente ocupada por ele, antes de criá-lo novamente.
- **Complexo_X *outro = converte_Y (me)**: converte o número complexo de *tipo Y* "me" no tipo de número complexo *do tipo X* "outro". Se "outro" já existia, deve liberar a área de memória anteriormente ocupada por ele, antes de criá-lo novamente. X e Y podem ser dos tipos double e long double.
- **Complexo_X resultado = soma_X (me, outro)**: soma dois números complexos de mesmo tipo "X" em um número complexo resultado também do tipo "X"
- **Complexo_X resultado = soma_X (me, outro)**: soma dois números complexos de mesmo tipo "X" em um número complexo resultado também do tipo "X"
- **long double modulo = ModuloComplexo_X (me):** calcula e retorna o módulo do número complexo, sempre retornando o valor para um número do tipo long double, independente se o tipo de "me" é um complexo "double" ou "long double"
- **Complexo_G maior = MaiorComplexoDoVetor (vetorzao)**: retorna o maior complexo dentro de um vetor de complexos genéricos chamado vetorzao, que mistura complexos do tipo "*double*" com complexos do tipo "*long double*".

- **Complexo_Ld somatudo = SomaComplexosDoVetor (vetorzao)**: retorna a soma dos números complexos dentro de um vetor de complexos genéricos chamado vetorzao, que mistura complexos do tipo "*double*" com complexos do tipo "*long double*". O resultado é sempre um complexo do tipo "*long double*", mesmo que todos os elementos dentro do vetorzao sejam somente do tipo *double.*
- … **outras funções que você achar necessárias**

Lembre-se que cada "estrutura número complexo", nesta implementação "orientada a objetos", será composta de:

- umponteiro para uma estrutura chamada "vtbl", que reúne os ponteiros para as funções que implementam métodos que são "substituídos" nas diferentes implementações dos números complexos
- o valor da componente real do número complexo
- o valor da componente imaginária do número complexo

1disponível em https://www.state-machine.com/doc/AN_OOP_in_C.pdf

Perguntas importantes:

- por que algumas funções são classificadas como static nesta implementação? O que isso significa?
- Por que algumas funções ficam “dentro da tabela de funções virtuais” e outras não ? Qual a diferença entre elas?
- Por que algumas das funções “abstratas”, isto é, funções da tabela virtual do número complexo “genérico” devem ser “*implementadas*” se elas nunca devem ser realmente “*executadas*” ?
- Como o compilador seleciona a função adequada a ser chamada (soma, produto, divisão,…) se o “*me*”, isto é, o número complexo “pai” da operação pode ser do tipo “double” em um momento e “long double” em outro? Isto é, como funciona o esquema de “tabela de funções virtuais” ?

4) ´[VALOR ATÉ 8 PONTOS] Implemente o método de Durand-Kerner para cálculo de raízes de polinômios complexos (veja em https://en.wikipedia.org/wiki/Durand%E2%80%93Kerner_method).

a) Se você implementar o método de Durand-Kerner empregando o arquivo <complex.h> que está disponível na biblioteca padrão Ansi C para um polinômio qualquer de grau sempre igual a 4 (como está explicado na página da Wikipedia), você ganhará 1 ponto;

b) Se você implementar o método de Durand-Kerner empregando o arquivo <complex.h> que está disponível na biblioteca padrão Ansi C para um polinômio qualquer de grau inferior a 10, você ganhará mais 1 pontos (se tiver feito o item (a)) ou 2 pontos (se **não** tiver feito o item (a));

c) Se você implementar o método de Durand-Kerner empregando o tipo abstrato de dados que foi implementado na questão 1, ganhará mais 2 pontos;

d) Se você implementar o método de Durand-Kerner empregando o tipo abstrato de dados implementado na questão 3 para um polinômio de tamanho sempre igual a 4, ganhará mais 2 pontos.

e) Se você implementar o método de Durand-Kerner empregando o tipo abstrato de dados implementado na questão 3 para um polinômio de **tamanho inferior a 10**, ganhará:

- 4 pontos (se você não implementou o item “d” ou o item “a”)
- 8 pontos (se você não implementou nada dos itens anteriores) .

Não se preocupe com casos de raízes repetidas. Somente apresente as raízes.