

Lista de Exercícios – Tipo Abstrato de Dados (TAD)<sup>1</sup>  
Prof. André Luiz Moura

## Sumário

<b>1 Conjunto de Números Naturais</b>	<b>1</b>
<b>2 Manipulando Datas</b>	<b>3</b>
<b>3 Processamento de Textos</b>	<b>5</b>
<b>4 Mundo das Bactérias</b>	<b>7</b>

### Observações:

- A resolução de todos os exercícios desta lista pressupõe a utilização do conceito de *Tipo Abstrato de Dados* (TAD) durante a implementação;
- Resolva a quantidade de questões cujos temas que perfaçam, no mínimo, quatro sinais de adição (+++);
- **Um único arquivo** pode ser enviado como proposta de solução para um problema (ou seja, um único arquivo com a extensão .c), tal arquivo deverá ter a estrutura apresentada na figura a seguir;
- Assim, o uso `tad.h` significa que a função `main()` elaborada como proposta de solução para o problema deve somente utilizar operações presentes neste arquivo.

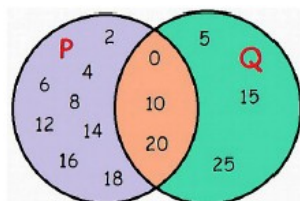
```
#include <stdio.h>
#include <stdlib.h>

<TADs: conteúdo do(s) arquivo(s) .h>

<TADs: conteúdo do(s) arquivo(s) .c , sem #include "TAD.h">

int main () {
    <seu código>
}
```

- O arquivos contendo a implementação do TAD.h, TAD.c e TestaTAD.c devem obedecer ao seguinte formato: **q<número>TAD.h**, **q<número>TestaTAD.c**, **q<número>TAD.c**. Exemplos: **q09TAD.h**, **q09TAD.c**, **q09TestaTAD.c**.
- Compactar todos os arquivos com como segue: **ResListaTAD.zip**



## 1 Conjuntos de Números Naturais

(++) Escreva, em C, um programa que seja capaz de implementar um conjunto de números naturais

<sup>1</sup> Extraída de listas elaboradas pelos Profs. Wanderley e Jacson

empregando o conceito de Tipo Abstrato de Dado (TAD).

**O programa implementar, no mínimo, as seguintes operações:**

**1. criar um conjunto C, inicialmente vazio:**

`int criaConjunto(C);`

retornando SUCESSO ou FALHA. A falha ocorre se não for possível, por alguma ocorrência, criar o conjunto C.

**2. verificar se o conjunto C é vazio:**

`int conjuntoVazio(C);`

retornando TRUE ou FALSE.

**3. incluir o elemento x no conjunto C:**

`int insereElementoConjunto(x, C);`

retornando SUCESSO ou FALHA. A falha acontece quando o elemento x já está presente no conjunto C.

**4. excluir o elemento x do conjunto C:**

`int excluirElementoConjunto(x, C);`

retornando SUCESSO ou FALHA. A falha acontece quando o elemento x não está presente no conjunto C.

**5. calcular a cardinalidade do conjunto C:**

`int tamanhoConjunto(C);`

retornando a quantidade de elementos em C. O valor 0 (zero) indica que o conjunto está vazio.

**6. determinar a quantidade de elementos do conjunto C que são maiores que x:**

`int maior(x, C);`

**7. determinar a quantidade de elementos do conjunto C que são menores que x:**

`int menor(x, C);`

**8. verificar se o elemento x pertence ao conjunto C:**

`int pertenceConjunto(x, C);`

retornando TRUE ou FALSE.

**9. comparar se dois conjuntos, C1 e C2 são idênticos:**

`int conjuntosIdenticos(C1, C2);`

retornando TRUE ou FALSE.

**10. identificar se o conjunto C1 é subconjunto do conjunto C2:**

`int subconjunto(C1, C2);`

retornando TRUE ou FALSE.

**11. gerar o complemento do conjunto C1 em relação ao conjunto C2:**

`Conjunto complemento(C1, C2);`

retornando um conjunto que contém elementos de C2 que não pertencem a C1.

**12. gerar a união do conjunto C1 com o conjunto C2:**

`Conjunto uniao(C1, C2);`

retornando um conjunto que contém elementos que estão em C1 ou em C2.

**13. gerar a intersecção do conjunto C1 com o conjunto C2:**

`Conjunto interseccao(C1, C2);`

retornando um conjunto que contém elementos que estão em C1 e, simultaneamente, em C2.

**14. gerar a diferença entre o conjunto C1 e o conjunto C2:**

`Conjunto diferenca(C1, C2);`

retornando um conjunto que contém elementos de C1 que não pertencem a C2.

**15. gerar o conjunto das partes do conjunto C:**

`Conjunto conjuntoPartes(C);`

**16. mostrar os elementos presentes no conjunto C:**

`void mostraConjunto(C, ordem);`

se ordem for igual a CRESCENTE, os elementos de C devem ser mostrados em ordem crescente. Se ordem for igual a DECRESCENTE, os elementos de C devem ser mostrados em ordem decrescente.

**17. copiar o conjunto C1 para o conjunto C2:**

`int copiarConjunto(C1, C2);`

retornando SUCESSO ou FALHA. A falha acontece quando, por algum motivo, não é possível copiar o conjunto C1 para o conjunto C2.

**18. destruir o conjunto C:**

`int destroiConjunto(C);`

retornando SUCESSO ou FALHA. A falha acontece quando, por algum motivo, não é possível eliminar o conjunto C da memória.

**Observação:** Considere que:

- SUCESSO = 1; FALHA = 0;
- TRUE = 1; FALSE = 0.

**Entradas e Saídas**

O programa elaborado deverá ter um menu que permita ao usuário selecionar cada uma das operações supramencionadas, executá-la e, em seguida, retornar ao menu para escolher uma nova opção.

Para *finalizar o programa* o usuário deverá fornecer um entrada especial. Por exemplo, o número 0 (zero) como opção no menu.

O(A) estudante tem liberdade para escolher como implementar a funcionalidade de menu.

**2 Manipulando Datas**

(++) A capacidade de manipular datas é de extrema importância em muitas aplicações práticas na área de processamento de dados. Infelizmente nem sempre há, numa determinada linguagem de programação que se está utilizando, uma biblioteca com variadas funções para isto.

Você está participando do desenvolvimento de uma biblioteca para esta finalidade, sendo que ela deverá ser escrita em C e conter pelo menos as seguintes funções, expressas por seus cabeçalhos:

**data.h**

1. `Data * criaData (unsigned int dia, unsigned int mes, unsigned int ano);`  
Cria, de maneira dinâmica, uma data a partir dos valores para dia, mês e ano fornecidos.
2. `Data * copiaData (Data d);`  
Cria uma cópia da data d, retornando-a.
3. `void liberaData (Data * d);`  
Destroi a data indicada por d.
4. `Data * somaDiasData (Data d, unsigned int dias);`  
Retorna uma data que é dias dias posteriores à data d. Por exemplo, fornecendo a data d = 12/11/2019 e dias = 5, retornará a data 17/11/2019.
5. `Data * subtrairDiasData (Data d, unsigned int dias);`  
Retorna uma data que é dias dias anteriores à data d. Por exemplo, fornecendo a data d = 12/11/2019 e dias = 15, retornará a data 28/10/2019.
6. `void atribuirData (Data * d, unsigned int dia, unsigned int mes, unsigned int ano);`  
Atribui, à data d, a data dia/mes/ano especificada. Se não for possível, então faz com que d seja alterada para NULL.
7. `unsigned int obtemDiaData (Data d);`  
Retorna a componente dia da data d.
8. `unsigned int obtemMesData (Data d);`  
Retorna a componente mes da data d.
9. `unsigned int obtemAnoData (Data d);`  
Retorna a componente ano da data d.
10. `unsigned int bissextoData (Data d);`  
Retorna TRUE se a data pertence a um ano bissexto. Do contrário, retorna FALSE.
11. `int comparaData (Data d1, Data d2);`  
Retorna MENOR se  $d1 < d2$ , retorna IGUAL se  $d1 = d2$  ou retorna MAIOR, se  $d1 > d2$ .
12. `unsigned int numeroDiasDatas (Data d1, Data d2);`  
Retorna o número de dias que existe entre as datas d1 e d2.  
Se  $d1 = d2$ , então o número de dias é igual a 0 (zero). Do contrário, será um número estritamente positivo.
13. `unsigned int numeroMesesDatas (Data d1, Data d2);`  
Se d1 e d2 estão no mesmo mês/ano, então o número de meses é igual a 0 (zero). Do contrário, será um número estritamente positivo.
14. `unsigned int numeroAnosDatas (Data d1, Data d2);`  
Se d1 e d2 estão no mesmo ano, então o número de anos é igual a 0 (zero). Do contrário, será um número estritamente positivo.
15. `unsigned int obtemDiaSemanaData (Data d);`  
Retorna o dia da semana correspondente à data d. Considerando que DOMINGO = 1; SEGUNDA-FEIRA = 2; ...; SÁBADO = 7.

```
16.char * imprimeData (Data d, char * formato);
```

Retorna uma string com a data “formatada” de acordo com o especificado em formato.

Se formato = “ddmmaaaa”, então a string retornada deverá apresentar os dois dígitos do dia, os dois dígitos do mês e os quatro dígitos do ano, nesta ordem, e separados por uma (/ – barra). Por exemplo: “12/11/2019”.

Se formato = “aaaammdd”, então a string retornada deverá apresentar os quatro dígitos do ano, os dois dígitos do mês e os dois dígitos do dia, nesta ordem, e separados por uma (/ – barra). Por exemplo: “2019/11/12”.

De maneira análoga, são válidas as seguintes strings de formatação:

- “aaaa”;
- “mm”;
- “dd”;
- “ddmm”.

### Entrada e Saídas

O programa elaborado deverá ter um menu que permita ao usuário selecionar cada uma das operações supramencionadas, executá-la e, em seguida, retornar ao menu para escolher uma nova opção.

Para finalizar o programa o usuário deverá fornecer um entrada especial. Por exemplo, o número 0 (zero) como opção no menu.

O(A) estudante tem liberdade para escolher como implementar a funcionalidade de menu.

### Observações

1. Uma data é formada por seu dia, mes e ano;
2. TRUE = 1; FALSE = 0;
3. A função comparaData (Data d1, Data d2) deve retornar:  
MENOR quando  $d1 < d2$ ;  
IGUAL quando  $d1 = d2$ ;  
MAIOR quando  $d1 > d2$ .  
com MENOR = -1; IGUAL = 0 e MAIOR = 1.
4. Fique atento a um importante evento que ocorreu no mês de outubro de 1582 envolvendo o calendário Gregoriano – pesquise sobre isto antes de implementar a função obtemDiaSemanaData (Data d).



### 3 Processamento de Textos

(+++ ) Vamos, agora, elaborar uma TAD que seja capaz, de maneira simples, representar um texto.

Nesse contexto, um texto é concebido como sendo uma sequência de caracteres sem nenhuma formatação especial, ou seja, não existe negrito, itálico ou qualquer outro atributo especial aplicado sobre os caracteres que formam o texto: há simplesmente o caractere. Entretanto, ele pode ser uma letra maiúscula ou minúscula, um dígito ou um símbolo especial.

Considere que as seguintes operações estão previstas para estarem presentes no TAD texto.h:

```
1.Texto * criaTexto (char * t);
```

```
2.void liberaTexto (Texto * t);
```

```

3. unsigned int obtemTamanhoTexto (Texto * t);

4. char * obtemTexto (Texto * t;

5. void mostraTexto (Texto *t, unsigned int colunas);

6. Texto * copiaTexto (Texto * t);

7. void alteraTexto (Texto * t, char * alteracao);

8. Texto * concatenaTextos (Texto * t1, Texto * t2);

9. char * obtemSubtexto (Texto * t, unsigned int inicio, unsigned int
tamanho);

10. unsigned int encontraSubtexto (Texto * t, char * subtexto, unsigned
int ocorrencia);

11. int comparaTextos (Texto * t1, Texto * t2);

```

### Entrada e Saídas

O programa elaborado deverá ter um menu que permita ao usuário selecionar cada uma das operações supramencionadas, executá-la e, em seguida, retornar ao menu para escolher uma nova opção.

Para finalizar o programa o usuário deverá fornecer um entrada especial. Por exemplo, o número 0 (zero) como opção no menu.

O(A) estudante tem liberdade para escolher como implementar a funcionalidade de menu.

### Observações

1. O procedimento alteraTexto substitui o texto presente em t pelo texto recebido em alteracao, mesmo que eles tenham tamanhos diferentes;
2. A função obtemTexto deverá retornar uma cadeia de caracteres com o texto armazenado;
3. A função comparaTextos(Texto \* t1, Texto \* t2) deve retornar:  
 MENOR quando  $t1 < t2$ ;  
 IGUAL quando  $t1 = t2$ ;  
 MAIOR quando  $t1 > t2$ .  
 com MENOR = -1; IGUAL = 0 e MAIOR = 1.
4. A função obtemSubtexto deverá uma cadeia de caracteres que se inicia na inicio-ésima posição de textttt e conter tamanho caracteres de extensão.  
 A primeira posição de t é a de número 1 (um). Se, a partir da posição inicio não for possível obter os tamanho caracteres solicitados, então a função deverá retornar uma cadeia que se inicia na posição inicio de t e contém até seu último caractere;
5. A função encontraSubtexto deve procurar pela ocorrência de número ocorrência ( $1^a$ ,  $2^a$ ,  $3^a$ , ...,  $n^a$ ) de subtexto em t.  
 Se encontrar esta ocorrência, deverá retornar a posição do primeiro caractere dela em t. Do contrário, deverá retornar 0 (zero), pois o primeiro caractere de t é considerado como sendo o de número 1 (um);
6. A função mostraTexto deve apresentar t, no dispositivo de saída do computador (normalmente o monitor de vídeo), de tal maneira que a cada linha do dispositivo de saída sejam apresentados colunas caracteres. Por consequência, o texto t poderá ocupar uma ou mais linhas.



#### 4 Mundo das Bactérias

(+++++)

Um estudante de Ciências Biológicas precisa fazer uma sequência de experimentos com uma cultura de bactérias para comprovar hipóteses a respeito de uma pesquisa da qual está participando envolvendo a movimentação de bactérias nesta cultura.

O problema é que para fazer experimentos no mundo real ele consome muito tempo e, sabendo que você é estudante do INF/UFG, pediu que você elaborasse um programa de computador que tornasse possível fazer uma simulação de culturas de bactérias.

Você gostou do desafio e passou algumas horas conversando com o estudante e, ao final, imaginou que uma cultura de bactérias pode ser representada por meio de uma matriz bidimensional  $W$ , que contém um par de números naturais  $(x_i; y_i)$ , com  $n$  linhas e  $m$  colunas. Você considerou que  $n, m \geq 1$  e  $1 \leq n, m \leq 1000$ .

Cada bactéria é representada por um par  $(x_i; y_i)$ , onde:

$x_i$  é a identificação daquela bactéria, ou seja, o número associado a ela – que é chamada de ordem da bactéria. Sabe-se que  $1 \leq x_i \leq (n \cdot m)$ ;

$y_i$  é a força da bactéria em relação às demais bactérias de sua cultura ou de outras culturas.

A força é expressa por um número que varia de 1 a 100, inclusive extremos, sendo que a força máxima é 100 e a mínima 1.

Como está você, neste momento, estudando os TADs, também se propôs a elaborar o programa solicitado utilizando este conceito e a linguagem de programação C, para tornar a aventura mais “hard”.

Depois de alguns rascunhos, você chegou à conclusão de que as seguintes operações devem ser disponibilizadas para o estudante de Ciências Biológicas por seu programa C:

```
[01] World * newWorld (unsigned int n, unsigned int m);
```

Cria uma nova cultura de bactérias, com  $n$  linhas e  $m$  colunas de tamanho. A cultura deve, após a operação, ficar vazia, ou seja, não conterá nenhuma bactéria.

```
[02] World * cloneWorld (World * w);
```

Faz a cópia da cultura de bactérias  $w$ , gerando um clone dela, mesmo que esta esteja vazia.

```
[03] void freeWorld (World * w);
```

Destrói a cultura de bactérias  $w$ .

```
[04] unsigned int randomWorld (World * w, unsigned int n);
```

Insere, em posições aleatórias, livres e distintas da cultura  $w$ ,  $n$  bactérias.

A ordem a ser utilizada para as bactérias a serem adicionadas se inicia no número natural seguinte ao associado à bactéria que possuir o maior ordem na cultura  $w$ . A força de cada bactéria deverá ser gerada aleatoriamente.

Se não for possível inserir as  $n$  bactérias em  $w$  (Por exemplo, por não haver espaço livre suficiente na cultura  $w$ ), a função deve retornar 0 (zero). Do contrário deve retornar 1 (um).

**Observação:** Se  $w$  estiver inicialmente vazia, então a ordem das bactérias deve ser iniciada em 1 (um) e, por consequência, terminar em  $n$ .

```
[05] unsigned int addBacterium (World * w, unsigned int n, unsigned int f);
```

Insere, numa posição livre aleatoriamente escolhida, na cultura w, a bactéria cuja ordem é dada pelo número n e cuja força é dada por f.

Se a bactéria de ordem n já está na cultura ou se a cultura não possui espaço para nenhuma bactéria adicional, a função deve retornar 0 (zero). Na hipótese de sucesso, a função deve retornar 1 (um).

```
[06] unsigned int addBacteriumXY (World * w, unsigned int n, unsigned int x, unsigned int y, unsigned int f);
```

Insere, na linha x e coluna y da cultura w, a bactéria cuja ordem é dada pelo número n e cuja força é f.

Se a bactéria de ordem n já está na cultura ou se a cultura não possui espaço para nenhuma bactéria adicional ou, ainda, se a posição (x,y) estiver ocupada, a função deve retornar 0 (zero). Na hipótese de sucesso, a função deve retornar 1 (um).

```
[07] unsigned int killBacterium (World * w, unsigned int n);
```

Mata (destrói) a bactéria cuja ordem é dada pelo número n, independentemente de sua localização na cultura.

Se a bactéria de ordem n não estiver na cultura, a função deve retornar 0 (zero). Na hipótese de sucesso, a função deve retornar 1 (um).

```
[08] unsigned int killBacteriumXY (World * w, unsigned int x, unsigned int y);
```

Mata (destrói) a bactéria que está na linha x e coluna y da cultura w.

Se não há bactéria na posição (x,y), a função deve retornar 0 (zero). Na hipótese de sucesso, a função deve retornar 1 (um).

```
[09] World * jointWorlds (World * w1, World *w2);
```

Realiza a união das culturas de bactérias w1 e w2, gerando uma nova cultura.

Não pode haver colisão entre as posições ocupadas por nenhuma das bactérias proveniente das culturas originais, pois esta união deve ser pacífica. Se houver colisão, a função deverá retornar NULL para indicar que a união não é possível. Do contrário, retorna a nova cultura gerada.

**Observação:** Lembre-se de que a cultura gerada deverá ter seus indivíduos renumerados, ou seja, a ordem dos elementos pertencentes à cultura gerada deverá ser alterada. Faça isto renumerando as bactérias provenientes da cultura w2 de maneira a dar sequência à máxima ordem existente na cultura w1.

```
[10] World * warWorlds (World * w1, World *w2);
```

Coloca as culturas de bactérias w1 e w2 em guerra, gerando uma nova cultura com as bactérias sobreviventes de acordo com as seguintes regras:

1. se houver duas bactérias que deveriam, na nova cultura, ocupar a mesma posição, a bactéria mais forte fagocita a bactéria mais fraca e ocupa aquele lugar na nova cultura. Além disso, sua nova força é adicionada à força da bactéria que acabou de fagocitar. Se houver empate, escolha a originária da cultura w1 para ir para a nova cultura, com força dobrada.

Se a força da bactéria vencedora superar a força máxima estabelecida, a ela deverá ser atribuída a força máxima.

2. se não houver disputa, a bactéria deverá permanecer, na nova cultura, na mesma posição em que está na cultura de origem, com sua mesma força original.

Observação: Lembre-se de que a cultura gerada deverá ter seus indivíduos renumerados, ou seja, a ordem dos elementos pertencentes à cultura w deverá ser alterada. Faça isto renumerando as bactérias provenientes da cultura w2 de maneira a dar sequência à máxima ordem existente na cultura w1.

```
[11] World * probabilisticWarWorlds (World * w1, World *w2, float p);
```

Coloca as culturas de bactérias w1 e w2 em guerra, gerando uma nova cultura com as bactérias sobreviventes de acordo com as seguintes regras:

1. se houver duas bactérias que deveriam, na nova cultura, ocupar a mesma posição, a bactéria mais forte fagocita a bactéria mais fraca de acordo com a probabilidade p, sabendo-se que  $0 < p \leq 1$ , e ocupa aquele lugar na nova cultura. Do contrário, a bactéria mais fraca é que fagocita a bactéria mais forte e ocupa



aquele lugar na nova cultura.

Além disso, a nova força da bactéria vencedora é adicionada à da bactéria derrotada. Se a força da bactéria vencedora superar a força máxima estabelecida, a ela deverá ser atribuída a força máxima.

2. se não houver disputa, a bactéria deverá permanecer, na nova cultura, na mesma posição em que está na cultura de origem, com sua mesma força original.

**Observação:** Lembre-se de que a cultura gerada deverá ter seus indivíduos renumerados, ou seja, a ordem dos elementos pertencentes à cultura  $w$  deverá ser alterada. Faça isto renumerando as bactérias provenientes da cultura  $w_2$  de maneira a dar sequência à máxima ordem existente na cultura  $w_1$ .

[12] `unsigned int sizeWorld (World * w);`

retorna o número de bactérias existentes na cultura  $w$ . O valor 0 (zero) indicará que a cultura está vazia.

[13] `unsigned int forceWorld (World * w);`

retorna a soma de todas as forças das bactérias existentes na cultura  $w$ . O valor 0 (zero) indicará que a cultura está vazia.

[14] `unsigned int showWorld (World * w);`

apresenta, no dispositivo de saída (normalmente o monitor de vídeo), uma “imagem” matricial da cultura  $w$  de acordo com seu tamanho (expressso por  $n \# m$ ).

**Observação:** A proposta aqui é que você elabore, livremente, uma maneira de mostrar o que está presente na cultura  $w$ . Por exemplo, mesmo no modo texto, é possível construir um mapa como o representado abaixo, para uma cultura de tamanho  $5 \# 5$ :

	1	2	3	4	5
1	6,1	2,100	5,70		
2				4,32	3,3
3	7,4		8,3		
4		1,80			10,80
5				9,45	

O conteúdo da linha 1, coluna 1, indica que nesta célula está a bactéria de ordem 6 e cuja força é igual a 1. Na posição (2,4), portanto, está a bactéria de ordem 4 e de 32.

Apesar do exemplo, você está livre para criar outras maneiras de mostrar a cultura: fique livre para concebê-la.

Observação: Se a cultura  $w$  for muito grande, ou seja, a dimensão  $n \# m$  não cabe inteiramente no dispositivo de saída, você poderá fazer com que o usuário escolha uma porção da cultura a ser visualizada. Isto pode ser feito fazendo com que ele selecione a porção das linhas e das colunas a serem mostradas.

Por exemplo, informando (1,10) e (6,14) significa que ele deseja visualizar as linhas de 1 a 10 e as colunas de 6 a 14. Evidentemente esta porção a ser visualizada não poderá superar a possibilidade de apresentação do dispositivo de saída. Se superar, avise-o da incorreção e peça para que ele selecione uma área menor.

## IMPLEMENTAÇÃO OPCIONAL

A implementação das funções a seguir é opcional, pois elas são um experimento contínuo com o que acontece na interação entre culturas de bactérias. Para aqueles(as) que gostam que “quebra-cabeças”,

[15] `World * continuumWarWorlds (World * w1, World * w2);`

coloca as culturas de bactérias  $w_1$  e  $w_2$  em guerra contínua, gerando uma sequência de novas culturas com as bactérias sobreviventes que, novamente, voltam à guerra. As regras para uma batalha desta guerra são as seguintes:

1. se houver duas bactérias que deveriam, na nova cultura, ocupar a mesma posição, a bactéria mais forte fagocita a bactéria mais fraca e ocupa aquele lugar na nova cultura. Além disso, sua nova força é

adicionada à força da bactéria que acabou de fagocitar. Se houver empate, escolha a originária da cultura w1 para ir para a nova cultura, com força dobrada.

Se a força da bactéria vencedora superar a força máxima estabelecida, a ela deverá ser atribuída a força máxima.

2. se não houver disputa, a bactéria deverá permanecer, na nova cultura, na mesma posição em que está na cultura de origem, com sua mesma força original.

Ao terminar uma batalha, a nova cultura (ou seja, a gerada pela batalha das culturas w1 e w2. Vamos chamá-la de w.) irá travar uma nova batalha com w1 ou w2 originais.

Quem irá para a batalha com w?

Aquela que tiver a maior soma das forças de suas bactérias! (Lembre-se, a função forceWorld (World \* w) lhe fornecerá esta informação).

Assim, a cultura w tomará o lugar w1 na nova batalha e a cultura mais forte entre w1 e w2 originais tomará o lugar de w2 nesta nova batalha.

Esta guerra terá fim?

Esta é uma ótima questão para experimentação: uma guerra terminará depois de uma sequência de DuracaoGuerra batalhas ou, se por algum motivo, houver uma estabilização do processo de guerrilha.

O valor de DuracaoGuerra será fixado por você durante a experimentação: 10, 20, 30, 50, ... 1000.

O que é uma estabilização da guerra?

Vamos considerar que a guerra ficou estável se após uma sequência de DuracaoEstabilizacao batalhas, não há mudança no valor retornado pela função forceWorld quando esta é aplicada na cultura vencedora da batalha.

Por exemplo: Depois de uma sequência de 10 batalhas (que seria o valor fixado para DuracaoEstabilizacao, a cultura vencedora está sempre com o mesmo valor de forceWorld obtido.

O valor DuracaoEstabilizacao terá que ser fixado para ser menor que o valor de DuracaoGuerra corrente.

Ao terminar a função deve retornar NULL se houve algum problema que a impediu de continuar até completar.

Do contrário, deve retornar a cultura vencedora da guerra contínua.

**Observação:** Lembre-se de que a cultura gerada em cada batalha deverá ter seus indivíduos renumerados, ou seja, a ordem dos elementos pertencentes à cultura w deverá ser alterada. Faça isto renumerando as bactérias provenientes da cultura w2 de maneira a dar sequência à máxima ordem existente na cultura w1.

[16] World \* continuumProbabilisticWarWorlds (World \* w1, World \*w2, float p) ;

é a versão probabilística da função World \* continuumWarWorlds (World \* w1, World \*w2), ou seja, ela utiliza as regras estabelecida por esta função para a guerra contínua, mas cada batalha entre w1 e w2 é realizada de acordo com as regras de probabilidade fixada em World \* probabilisticWarWorlds (World \* w1, World \*w2, float p).

### Entradas e Saídas

O programa elaborado deverá ter um menu que permita ao usuário selecionar cada uma das operações supramencionadas, executá-la e, em seguida, retornar ao menu para escolher uma nova opção.

Para finalizar o programa o usuário deverá fornecer um entrada especial. Por exemplo, o número 0 (zero) como opção no menu.

O(A) estudante tem liberdade para escolher como implementar a funcionalidade de menu.

### Observações:

Não há observações adicionais.