## Fundamentos da Programação

### lista de exercícios 10

1. **Desordenação de listas**

   a) Escreva uma função que escolhe uma posição aleatória `p`, e troca o elemento da primeira posição com o elemento da posição `p`.

   A seguir, faça um programa que chama essa função muitas vezes, e veja o resultado.

   b) Escreva uma função que escolhe uma posição aleatória `p`, e troca a porção `L[0..p-1]` da lista de lugar com a porção `L[p..N-1]`.

   A seguir, faça um programa que chama essa função muitas vezes, e veja o resultado.

2. **Manipulação de listas ordenadas**

   Lembre que a implementação das operações com listas ordenadas (busca, inserção e remoção) envolviam várias etapas.

   Reorganize esse código utilizando a técnica das caixinhas, e verifique se você consegue implementar funções que podem ser utilizadas em mais de uma operação.

3. **Aritmética com racionais**

   Defina o seguinte struct para armazenar números racionais

```
struct racional
 {
    int   num;   // numerador
    int   den;   // denominador
 };
```

   E em seguida escreva as seguintes funções que implementam as operações aritméticas sobre números racionais

```
struct racional somaR (struct racional r1, struct racional r2)

struct racional subtraçãoR (struct racional r1, struct racional r2)

struct racional multiplicaçãoR (struct racional r1, struct racional r2)

struct racional divisaoR (struct racional r1, struct racional r2)
```

   E implemente também a função

```
struct racional simplificaR (struct racional r1, struct racional r2)
```

   que coloca a fração nos termos mais simples possíveis — (*i.e., numerador e denominador são primos entre si*)

4. **Listas ordenadas de racionais**

   Implemente a função

   ```
   int comparaR (struct racional r1, struct racional r2)
   ```

   que retorna:

   - `0`, se os números racionais $\texttt{r}_1$ e $\texttt{r}_2$ correspondem ao mesmo valor
   - `1`, se o racional $\texttt{r}_1$ é maior que o racional $\texttt{r}_2$
   - `-1`, se o racional $\texttt{r}_2$ é maior que o racional $\texttt{r}_1$

   A seguir, utilize essa função para implementar uma outra função que ordena uma lista de racionais.

5. **Lista de listas**

   Imagine que alguém decide armazenar dados em uma lista de listas, ao invés de uma única lista.

   Por exemplo, isso aqui

   ```
   int L[4][5]  =  { { 1,  4,  9, 21, 18 },
                     { 3,  8, 17, 33, 42 },
                     { 2,  6, 11, 13, 15 },
                     { 5, 10, 14, 22, 25 } };
   ```

   ao invés disso aqui

   ```
   int L[20]  =  {  1,  2,  3,  4,  5,  6,  8,  9, 10, 11,
                   13, 14, 15, 17, 18, 21, 22, 25, 33, 42 };
   ```

   A vantagem dessa ideia é que ela evita grandes deslocamentos, no momento da inserção e remoção de elementos — (*o que é necessário para manter a(s) lista(s) ordenadas*)

   Por outro lado, trabalhar com um lista de listas envolve toda uma série de pequenas complicações adicionais — (*que você vai descobrir quando for programar*)

   Mas, depois que a gente escreve boas funções para acessar a lista de listas, a gente não precisa mais se preocupar com isso.

   E daí, a gente pode usufruir do ganho de eficiência que elas oferecem.

   a) Escreva uma função que inicializa a lista de listas

      ```
      int L[M][N];
      ```

      de maneira aleatória — (*i.e., uma quantidade aleatória de elementos aleatórios em cada lista*)

   b) Escreva uma função que ordena todas as listas de `L`.

   c) Escreva uma função que busca um número `x` na lista de listas `L`.

   d) Escreva uma função que insere um novo elemento `x` na lista de listas `L` — (*se ele ainda não estiver lá*)

   e) Escreva uma função que remove um número `x` da lista de listas `L`.

   f) Realize um teste de velocidade, para comparar a lista de listas com a lista ordenada.