IMD0029 - Estrutura de Dados Básicas 1 – 2019.2
Prof. Eiji Adachi M. Barbosa
Atividade Avaliativa Prática em Laboratório – Unidade 1

ANTES DE COMEÇAR, leia atentamente as seguintes instruções:

- Esta é uma atividade de caráter individual e sem consultas a pessoas ou material (impresso ou eletrônico).
- A atividade vale 5,0 pontos na 1ª unidade. O valor de cada questão é informado no seu enunciado.
- Celulares e outros dispositivos eletrônicos devem permanecer desligados durante toda a prova.
- Desvios éticos ou de honestidade levarão à anulação da atividade do candidato (nota igual a zero).
- Junto a este enunciado, você também recebeu uma estrutura de diretórios contendo um diretório para cada questão. Em cada um destes diretórios, já existe uma assinatura de função e uma função main com um pequeno teste executável. A solução da sua questão deverá seguir a assinatura da função já estabelecida. Ou seja, não mude esta assinatura. Se necessário, crie funções auxiliares com outras assinaturas, mas **não mude a assinatura da função original!**

Questão 1 (1,0 ponto): Nesta questão, implemente uma função que calcula de forma **recursiva** a divisão entre dois números inteiros e positivos. Sua função deverá seguir a assinatura:

```
int divide(int a, int b) // significa: a dividido por b
```

Obs.: Não precisa tratar os casos de divisão por zero.

Questão 2 (1,0 ponto): Sequências bitônicas inversas são aquelas que possuem duas sequências, sendo uma sequência inicial decrescente, seguida de uma sequência crescente. Ou seja, os elementos de uma sequência bitônica inversa obedecem a seguinte relação:

$$A_0 > A_1 > ... > A_{i-1} > A_i < A_{i+1} < ... < A_n$$

Considere que um vetor bitônico-inverso é um vetor de inteiros sem repetições cujos elementos representam uma sequência bitônica inversa. Nesta questão, implemente a função findBase que recebe como entrada um vetor bitônico-inverso e retorna o índice da base, isto é, do elemento Ai, que representa a transição da sequencia decrescente para a sequencia crescente. Sua solução deverá ser **recursiva** ter complexidade Θ **(lg(n))**.

Questão 3 (1,5 ponto): Um anagrama é um jogo de palavras em que uma palavra é construída a partir do rearranjo das letras de uma outra palavra utilizando cada letra da palavra original exatamente uma vez, isto é, não mais nem menos do que isso. Por exemplo: Iracema é um anagrama da palavra America. Implemente a função isAnagram que recebe duas strings e retorna verdadeiro se estas palavras são anagramas e falso caso contrário.

Obs.: Nesta questão, pode assumir que as strings de entrada terão todos os caracteres em caixa alta (letras maiúsculas).
Dicas: Para acessar o i-ésimo caractere de uma string, basta usar o operador [ ] (ex.: char ci = str[i]). Para saber o tamanho de uma string, basta invocar o método size (ex.: size_t stringSize = str.size()).

Questão 4 (1,5 ponto): Implemente o algoritmo de ordenação QuickSort usando a estratégia de seleção de pivô "mediana de 3". Esta questão requer o arquivo de entrada input.txt que está disponível no diretório "q4" .

ENTREGÁVEL

O entregável desta atividade deverá seguir a mesma estrutura de diretórios do código fonte que você recebeu com este enunciado, obviamente, contendo os arquivos fonte utilizados para construir sua solução nos diretórios de cada questão. Além disso, o diretório pai deverá ter o seu nome e matrícula, seguindo o padrão `<PRIMEIRO_NOME>_<SOBRENOME>-<MATRICULA>` (Dica: basta renomear o diretório /src com o padrão definido anteriormente). Por exemplo:

```
> JOAO_SILVA-200012345
   > q1
   > q2
   > q3
   > q4
```

Toda esta estrutura de diretórios, incluindo os arquivos fonte com sua solução, deverá ser compactada num arquivo .zip que também deverá seguir o padrão `<PRIMEIRO_NOME>_<SOBRENOME>-<MATRICULA>`. **O .zip não deve conter qualquer arquivo executável.** Este arquivo compactado deverá ser entregue via SIGAA até as **20:25**. **Este é um prazo fixo que não será estendido**, exceto em casos muito excepcionais (ex.: SIGAA fora do ar). Ou seja, entregas após este horário não serão aceitas. A atividade do SIGAA permite apenas um envio, portanto certifique-se de que está enviando a versão correta antes de anexar ao SIGAA.

CRITÉRIOS DE CORREÇÃO

Para a correção desta atividade, serão levados em consideração, dentre outros, os seguintes pontos:

- Obediência às regras definidas para as assinaturas de função e para o entregável (arquivo .zip), conforme especificado no enunciado desta atividade
- Existência de erros ou warnings de compilação do código fonte[1]
- Programas executam sem apresentar falhas e produzem os resultados esperados
- Soluções atendem critérios de complexidade, caso estabelecido no enunciado
- Apresentação e organização do código fonte entregue (identação, nome das variáveis, modularização do código em funções, etc)

Obs.: Para cada questão, já há uma função main com um pequeno teste executável. Este é um teste simples que **não garante** a corretude da sua implementação. Ou seja, se sua implementação passou no teste executável disponibilizado junto a este enunciado, isto não é garantia de que ela está totalmente correta. Para fins de correção, eu utilizarei outra bateria de testes mais completa, além de analisar manualmente o código produzido.

[1] Compile usando as flags `-Wall -pedantic -std=c++11`