IMD0029 - Estrutura de Dados Básicas 1 – 2018.2
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 a nota igual a zero na Unidade 1.**
- 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): Dada uma string S, implemente uma função **recursiva** que retorne uma outra string com os caracteres de S em ordem inversa. Para isto, implemente a função *revert* no arquivo /q1/main.cpp.

Questão 2 (1,5 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. Neste questão, já existe uma implementação da função *findBase* que recebe como entrada um vetor bitônico-inverso e retorna o índice do elemento da "base". O elemento "base" é o último elemento da sequência inicial decrescente e o primeiro elemento da sequência final crescente, ou seja, é o elemento Ai da relação acima. A função *findBase* já implementada está no arquivo q2/BitonicSequence.cpp. Essa implementação disponibilizada possui alguns defeitos em sua implementação. Corrija os defeitos que você encontrar, deixando em forma de comentários a explicação do que estava errado e o que você fez para corrigir os defeitos encontrados.

Questão 3 (1,0 ponto): Implemente o Merge Sort. Implemente-o na função sort do arquivo q3/MergeSort.cpp. O programa implementado na função main() depende do arquivo input.txt disponível no diretório q3. Passe este arquivo como argumento pela linha de comando ao invocar o programa (ex.: ./main input.txt).

Questão 4 (1,5 ponto): Considere a seguinte estrutura que representa os dados de uma pessoa:

```
struct Person {
    string firstName;    string lastName;    string country;
};
```

Implemente uma versão do Insertion Sort que ordene um array de ponteiros para estrutura Person de modo que o array fique ordenado em ordem crescente pelo atributo 'country' e em ordem decrescente do atributo 'lastName'. Sua função deverá seguir a assinatura:

```
void sort(Person** people, int size)
```

Implemente-o na função sort do arquivo main.cpp. O programa implementado na função main() depende do arquivo input.txt disponível no diretório q4 (obs.: este arquivo é diferente do usado na questão anterior). Passe este arquivo como argumento pela linha de comando ao invocar o programa (ex.: ./main input.txt).

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. Caso não tenha feito alguma questão, remova o diretório correspondente à questão. Ou seja: só envie os diretórios das questões que você realmente fez.

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>`. Evite enviar o arquivo executável no arquivo .zip.

O 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 algumas questões, 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`