IMD0029 - Estruturas de Dados 1 - 2023.1 - Unidade 2

Prof. Eiji Adachi

LEIA ANTES DE COMEÇAR

- Esta é uma atividade de caráter individual e sem consultas a pessoas ou material (impresso ou eletrônico).
- 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, um arquivo `Makefile` e um teste executável, que pode ser executado usando o comando `make run-test`.
- A solução da sua questão deverá seguir a assinatura da função já estabelecida, conforme o enunciado. Ou seja, não mude esta assinatura. Se necessário, crie funções auxiliares com outras assinaturas, mas não mude a assinatura da função original.

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 e estrutura de diretórios), conforme especificado no enunciado.
- Existência de erros ou warnings de compilação do código fonte. Compile usando o arquivo `Makefile` disponibilizado.
- 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 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.

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 raiz deverá ter o seu nome, seguindo o padrão `<PRIMEIRO_NOME>_<SOBRENOME>`

Por exemplo:

```
> JOAO_SILVA
>   q1
>   q2
>   q3
>   q4
```

Toda esta estrutura de diretórios, incluindo os arquivos fonte com sua solução, deverá ser compactada num arquivo .zip (não é .rar, nem .tar.gz, nem qualquer outra extensão) que também deverá seguir o padrão `<PRIMEIRO_NOME>_<SOBRENOME>.zip`

O .zip não deve conter qualquer arquivo executável. Execute o comando `make clean` antes de compactar o seu entregável.

Este arquivo compactado deverá ser entregue via SIGAA até o horário estabelecido na tarefa. 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. Certifique-se de que está enviando a versão correta antes de anexar ao SIGAA.

Questão 1 - Valor: 2.0

Na notação infixa de expressões aritméticas, os operadores são escritos entre os operandos. Por exemplo: 1 + 2, ou 2 - 3 * 2. Já na notação pós-fixa, ou notação polonesa inversa, os operadores são escritos depois dos operandos. Por exemplo: 1 2 +, ou 2 3 2 * -. Eis alguns outros exemplos de expressões infixas e correspondentes expressões pós-fixas:

Expressão em notação Infixa	Expressão equivalente em notação Pós-fixa
1 + 2	1 2 +
2 / 2	2 2 /
1 + 2 * 3	1 2 3 * +
1 * 2 + 3 - 4	1 2 * 3 + 4 -

Nesta questão, implemente a seguinte função:

```
int resolverExpressao(std::vector<std::string> partesDaExpressao)
```

Essa função recebe como parâmetro de entrada um `vector` de `strings` representando uma expressão aritmética em notação pós-fixa e que retorna o seu valor numérico correspondente. Por exemplo: a expressão `1 2 3 * +` será representada no código da seguinte maneira: `std::vector<std::string> entrada = {"1", "2", "3", "*", "+"}`.

A função está definida no arquivo `include/Expressao.h` e sua implementação deve ser feita no arquivo `src/Expressao.cpp`.

Sua solução deverá, obrigatoriamente, usar uma pilha; para isso, use a estrutura `stack` da biblioteca padrão de C++. No arquivo `Expressao.cpp`, existem exemplos de código de como declarar e manipular uma `stack` de C++ e como converter uma `string` em um valor inteiro. Quaisquer dúvidas quanto ao uso dessa estrutura, pode me chamar.

Questão 2 - Valor: 1.5

Dada uma lista duplamente encadeada com sentinelas cabeça e cauda, implemente o seguinte método:

```
bool ListaDuplamenteEncadeada::inserirOrdenado(std::string s)
```

Esse método insere um elemento na lista, mantendo-a ordenada em ordem crescente. Considere que a lista ou está vazia ou já está ordenada em ordem crescente. Caso o elemento já exista na lista, a função não cria um novo nó e retorna falso; caso o elemento ainda não exista na lista, deve-se criar um novo nó e encadeá-lo na lista, incrementando a quantidade de elementos da lista e mantendo-a ordenada.

Para esta questão, implemente o método `inserirOrdenado` já declarado no arquivo `src/ListaDuplamenteEncadeada.cpp`. Sua solução deverá ter complexidade $\theta(n)$.

Questão 3 - Valor: 1.5

Dada uma lista simplesmente encadeada com ponteiro para o início, implemente o seguinte método:

```
void  ListaEncadeada::ordenar()
```

Esse método implementa algum algoritmo de ordenação iterativo para ordenar a lista em ordem crescente. Você pode usar qualquer algoritmo de ordenação, desde que seja iterativo.

Para esta questão, implemente o método `ordenar` já declarado no arquivo `src/ListaEncadeada.cpp`.

Obs.: A ordenação deve ser implementada manipulando apenas os nós da lista; não é permitido usar um array auxiliar, ou outro tipo de estrutura, para ordenar a lista.