IMD0029 - Estruturas de Dados 1 - 2024.1 - Unidade 2

Prof. Eiji Adachi

LEIA ANTES DE COMEÇAR

- Esta é uma atividade de caráter individual.
- O valor de cada questão é informado no seu enunciado.
- Desvios éticos ou de honestidade levarão à nota igual a zero na respectiva unidade.
- É terminantemente proibido o acesso a qualquer tipo de material impresso ou eletrônico. Qualquer acesso (ex.: sites na Internet) resultará na anulação da prova e nota igual a zero na respectiva unidade.
- Celulares e outros dispositivos eletrônicos devem permanecer desligados e à vista durante toda a prova. Qualque consulta a dispositivos eletrônicos resultará na anulação da prova e nota igual a zero na respectiva unidade.
- 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
>   lib
>   q1
>   q2
>   q3
```

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` .

Caso você não tenha resolvido alguma questão, não coloque o seu respectivo diretório no entregável. Ou seja, remova o diretório da questão que você não fez antes de compactar o arquivo entregável.

O .zip não deve conter qualquer arquivo executável ou código objeto (arquivo com extensão `.o`). Execute o comando `make clean` antes de compactar o seu entregável para remover esses arquivos.

O 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: 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 **DECRESCENTE**. Considere que a lista ou está vazia ou já está ordenada. Caso o elemento já exista na lista, a função não cria um novo nó e retorna `false`; caso o elemento ainda não exista na lista, deve-se criar um novo nó e encadeá-lo na lista, mantendo-a ordenada, incrementando a quantidade de elementos da lista e retornando `true`.

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

Questão 2 - Valor: 2.0

Dada uma lista simplesmente encadeada com ponteiro para o início, implemente o seguinte método que remove elementos repetidos da lista:

```
int ListaEncadeada::removerTodos(int val)
```

Esse método remove todos os elementos da lista que possuem valor igual a `val`, retornando o número de elementos que foram removidos da lista.

Por exemplo: considere uma lista no seguinte estado `{1 -> 7 -> 3 -> 7 -> 5 -> 7 -> 14 -> null}`. Após a execução da chamada `removerTodos(7)`, a lista estará no estado `{1 -> 3 -> 5 -> 14 -> null}` e o método retornará 3, pois foram removidos três nós cujos valores eram `7`.

Sua solução deverá ter complexidade de tempo da ordem de `O(n)`.

Obs.: Não é permitido usar estruturas auxiliares, como stack, array ou vector, por exemplo.

IMPORTANTE: Não alterar o método `ListaEncadeada::imprimir`, pois ele é usado pelos testes.

Questão 3 - Valor: 1.5

Nesta questão, implemente o seguinte método:

```
bool Analisador::eBemFormada(std::string entrada)
```

Este método recebe como entrada uma string que representa uma sequência de chaves `{}`, parênteses `()` e colchetes `[]` aninhados e verifica se o aninhamento dessa sequência é bem formada ou não.

Uma sequência é considerada bem formada se:

- Cada abertura de parêntese, colchete ou chave tem um fechamento correspondente.
- Os fechamentos estão na ordem correta.

Por exemplo:

- A sequência `{[()]()}` é bem formada.
- A sequência `[()` não é bem formada, porque o primeiro colchete não tem um fechamento correspondente.
- A sequência `[())` não é bem formada, porque os fechamentos não estão na ordem correta.

Obs.: Sua solução deverá, obrigatoriamente, usar uma pilha. Para isso, use uma `stack` da biblioteca padrão de C++. No arquivo Analisador.cpp há um exemplo de como iterar sobre os caracteres da string de entrada e como usar uma pilha do tipo `stack`.

Obs.: Considere que o aninhamento entre os parênteses, as chaves e os colchetes podem ocorrer em qualquer ordem na sequência de entrada.