UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Faculdade da Computação
1º Trabalho de Estrutura de Dados 1
Prof. Luiz Gustavo Almeida Martins

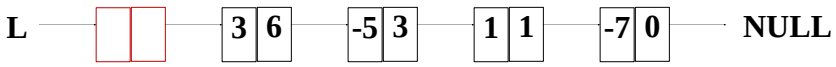
- ✓ Deve ser entregue um arquivo zip com os códigos das questões organizados em pastas (uma pasta por questão), sendo a integridade desse arquivo de responsabilidade dos alunos.
- ✓ Os códigos deverão ser implementados somente em Linguagem C, sendo necessária a implementação e utilização das estruturas de dados conforme discutido nas aulas.
- ✓ Deve-se aproveitar o conhecimento da estrutura e seu funcionamento para buscar a maior eficiência da lógica adotada na implementação das operações de um TAD.
- ✓ Em todas as questões, além das operações vistas nas aulas (ex: *cria_lista, lista_vazia, lista_cheia, insere, remove, esvazia_lista, apaga_lista, get_elem_pos*), deve-se implementar um programa aplicativo. Exceto pela questão 3, onde as funcionalidades do aplicativo já estão especificadas, nas demais deve-se implementar um programa aplicativo que disponibilize um menu que permita executar repetidamente todas as operações do TAD.
- ✓ **Antes de cada operação do TAD, deve-se colocar um comentário com a sua especificação.**

1) Implementar o TAD **lista ordenada CRESCENTE** de *strings* com no máximo 10 elementos, cada um com até 20 caracteres, usando alocação **estática/sequencial**. Além das operações básicas vistas em sala, o TAD também deve contemplar:
   - **Remove_todas:** remove todas as ocorrências da *string* informada como entrada.
   - **Remove_posicao:** se a posição for válida (existir na lista), remove o elemento que se encontra na posição indicada, retornando o seu valor.
   - **Tamanho_lista:** retorna a quantidade de elementos da lista.
   - **Intercala:** recebe duas listas ordenadas (L1 e L2) e retorna uma nova lista L3 formada pelos elementos de L1 e L2 intercalados. A intercalação dos elementos deve ser feita de modo a respeitar o critério de ordenação. As listas originais não devem ser alteradas.

2) Implementar o TAD **lista não ordenada** de no máximo 15 alunos, usando alocação **estática/sequencial**. Para cada aluno devem ser guardadas as seguintes informações: matrícula, nota final, número de faltas e seu conceito ('A' – aprovado ou 'R' - reprovado). Para simplificar a implementação, a estrutura aluno deve ser declarada no arquivo cabeçalho, permitindo sua utilização tanto no TAD, quanto pela aplicação. Além das operações vistas em sala, o TAD também deve contemplar:
   - **Insere_posicao:** insere o aluno na posição indicada, se ela for válida.
   - **Remove_menor:** remove o aluno da lista com a menor matrícula. Essa operação, quando bem-sucedida (não falhar), também deve retornar os dados do aluno removido.
   - **Remove_faltosos:** remove todos os alunos da lista cujo número de faltas seja superior ao valor especificado, retornando, ao final, a quantidade de alunos removidos
   - **Concatena:** recebe duas listas (L1 e L2) e retorna uma nova lista L3 formada pelos elementos de L1 e, na sequência, pelos elementos de L2. As listas originais não devem ser alteradas.

3) Implementar o TAD **lista ordenada DECRESCENTE** de termos de um polinômio, usando alocação **dinâmica/encadeada COM cabeçalho**, sendo que cada termo é formado pelo seu coeficiente e expoente (ex: $3x^2$). Nessa implementação a lista deve ser ordenada pelo expoente do termo e a estrutura do termo também deve estar encapsulada no TAD. As operações vistas em sala devem ser modificadas para atender as seguintes propriedades:
   - A operação de inserção não deve permitir a presença de dois termos com o mesmo expoente no polinômio, ou seja, se já existe um termo $3x^2$ no polinômio, ao ser inserido um novo termo $5x^2$, seu coeficiente será adicionado ao já existente resultando em $8x^2$.
   - A operação de remoção deve eliminar o termo do polinômio a partir do seu expoente, independentemente do coeficiente.
   - Também deve ser implementada uma operação *tamanho()* deve retornar a quantidade de termos do polinômio representando pela instância da lista (OBS: deve aproveitar a forma de implementação para realizar essa operação o mais eficiente possível).

   Utilizando o TAD acima, também implemente um programa aplicativo para manipular polinômios. Para tal, o polinômio deve ser armazenado através de uma lista ordenada, sendo que cada elemento *k* da lista deve armazenar o *k*-ésimo termo do polinômio, armazenando o valor *k* da potência de *x* (inteiro) e o coeficiente a_k correspondente (inteiro). Por exemplo, o polinômio $P(x) = 3x^6 - 5x^3 + x - 7$ deve ser representado pela lista (nó cabeçalho em vermelho):



   O aplicativo deve permitir ao usuário selecionar repetidamente (exceto a 1a opção) qualquer uma das operações a seguir:
   - **Criar um polinômio**, que consiste em criar uma instância de lista vazia.
   - **Inserir um novo termo em um polinômio** já criado.
   - **Informar o tamanho atual do polinômio**
   - **Eliminar o termo associado à *k*-ésima potência**, sendo *k* um valor informado pelo usuário.
   - **Reinicializar um polinômio**, que consiste em retornar a lista para o estado de vazia.
   - **Imprimir um polinômio**. Por exemplo, para o polinômio P(x) apresentado acima, o programa deve mostrar:

     P(x) = 3x^6 - 5x^3 + x^1 - 7x^0

   - **Calcular o polinômio** para um valor de *x* fornecido pelo usuário. Observe que a lista não será alterada, apenas seus elementos serão consultados durante o cálculo do polinômio. Por exemplo, se o polinômio P(x) acima for calculado com $x = 2$, será mostrado na tela:
     P(2) = 3(2)^6 - 5(2)^3 + (2)^1 – 7(2)^0 = 192 – 40 + 2 – 7 = 147
   - **Sair do programa**.

4) Implementar o TAD **lista não ordenada** de números reais (*float*), usando alocação **dinâmica/ encadeada SIMPLES (sem qualquer técnica de encadeamento)**. Além das operações vistas em sala, o TAD também deve contemplar:
   - **Remove_maior:** remove e retorna o maior elemento da lista. No caso de empate, deve-se remover a **primeira ocorrência** encontrada.
   - **Tamanho_lista:** retorna a quantidade de elementos da lista
   - **Intercala:** recebe duas listas de entrada (L1 e L2) e retorna uma nova lista (L3) com os elementos das duas listas de entrada intercalados. As listas originais não devem ser alteradas.

5) Implementar o TAD **lista não ordenada** de *caracteres*, usando alocação **dinâmica com encadeamento CÍCLICO**. Além das operações vistas em sala, o TAD também deve contemplar:
   - **Insere_inicio:** insere o elemento no início da lista.
   - **Remove_fim:** remover o último elemento da lista, retornando seu valor para a aplicação.
   - **Tamanho_lista:** retorna a quantidade de elementos da lista.
   - **Iguais:** verifica se as duas listas dadas como entrada (L1 e L2) são idênticas ou não.

6) Implementar o TAD **lista não ordenada** de números inteiros, usando alocação **dinâmica com encadeamento DUPLO**. Além das operações vistas em sala, o TAD deve contemplar:
   - **Remove_pares:** remove todos os elementos pares da lista.
   - **Remove_maior:** remove o maior elemento da lista, retornando seu valor. No caso de empate, deve-se remover a **última ocorrência** encontrada.
   - **Insere_posicao:** insere o elemento dado na posição indicada. Se ela não existir na lista, a operação deve indicar falha.
   - **Inverte_lista:** recebe uma lista L e retorna uma nova lista L2, formada pelos elementos de L na ordem inversa. A lista original não deve ser alterada.