UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Faculdade da Computação
1º Trabalho de Algoritmos e 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/TAD), sendo a integridade desse arquivo de responsabilidade dos alunos.
- ✓ A apresentação individual deve ser agendada com o professor para a semana especificada no plano de ensino. É desejável que todos os integrantes de um grupo agendem horários consecutivos para suas apresentações (15 minutos por aluno).
- ✓ Os códigos deverão ser implementados somente em Linguagem C, sendo necessária a utilização das estruturas de dados conforme discutidas em sala.
- ✓ 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.

1) Implementar o TAD **lista não ordenada** de números reais (*float*) usando alocação **estática/sequencial**. Além das operações vistas em sala, o TAD também deve contemplar:
   - Remover ímpares: remove todos os elementos ímpares da lista.
   - Maior: retorna o maior elemento da lista
   - Tamanho: retorna o número de elementos da lista
   - Esvaziar: apaga todos os elementos da lista, tornando-a vazia.
   - Inverter: recebe uma lista L e retorna uma nova lista L2, formada pelos elementos de L na ordem inversa.
   - Concatenar: recebe duas listas (L1 e L2) e retorna uma nova lista L3 com os elementos de L1 seguidos dos elementos de L2. As listas originais não devem ser alteradas.

2) Implementar o TAD **lista ordenada** decrescente de caracteres usando alocação **estática/sequencial**. Além das operações vistas em sala, o TAD também deve contemplar:
   - Remover pares: remove todos os elementos pares da lista.
   - Menor: retorna o menor elemento da lista.
   - Tamanho: retorna o número de elementos da lista.
   - Esvaziar: apaga todos os elementos da lista, tornando-a vazia.
   - Intercalar: recebe duas listas ordenadas (L1 e L2) e retorna uma nova lista L3 com os elementos de L1 e L2 intercalados conforme o critério de ordenação. As listas originais não devem ser alteradas.

3) Implementar o TAD **lista ordenada** crescente de inteiros usando alocação **dinâmica/encadeada SEM cabeçalho**. Além das operações vistas em sala, o TAD também deve contemplar:
   - Tamanho: retorna o número de elementos da lista
   - Iguais: recebe duas listas ordenadas e verifica se elas são iguais
   - Maior: retorna o maior elemento da lista.
   - Tamanho: retorna o número de elementos da lista.
   - Esvaziar: apaga todos os elementos da lista, tornando-a vazia.
   - Intercalar: recebe duas listas ordenadas e retorna uma nova lista com os elementos das duas listas de entrada intercalados conforme o critério de ordenação. As listas originais não devem ser alteradas.

4) Implementar o TAD **lista ordenada** de números reais (*double*) usando alocação **dinâmica/ encadeada COM cabeçalho**. O TAD deve contemplar as operações definidas no item 3.

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:
   - Inserir no início: inserir o elemento no início da lista
   - Inserir na posição: insira o elemento em uma posição definida na chamada da função. A operação deve verificar se a posição desejada é válida.
   - Remove elemento da posição: remover o elemento que se encontra na posição definida na chamada da função, retornando seu valor para a aplicação. Se a posição não existir na lista, a operação também deve indicar falha.
   - Remove no fim: remover o último elemento da lista, retornando seu valor para a aplicação.
   - Tamanho: retorna o número de elementos da lista.

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:
   - Tamanho: retorna o número de elementos da lista
   - Remover todos: remove todas as ocorrências de um elemento da lista
   - Remover maior: remove o maior elemento encontrado na lista, retornando seu valor.
   - Esvaziar: apaga todos os elementos da lista, tornando-a vazia.
   - Inverter: recebe uma lista L e retorna uma nova lista L2, formada pelos elementos de L na ordem inversa.
   - Primos: retornar uma lista L2 formada pelos elementos da lista de entrada L que são números primos.

7) Implementar o problema de Josephus utilizando o TAD lista.

   Problema: um grupo de soldados está cercado pelo inimigo e existe apenas um cavalo para a fuga. Decidiu-se que o soldado que se salvará será definido na sorte, independente da patente. O processo de escolha seria por eliminação, sendo que o último soldado a ser selecionado se salvaria. O processo de eliminação consiste em: organizar os soldados em um volta da fogueira; escolher um soldado para iniciar a contagem e sortear um único número. Ao final da contagem, o soldado escolhido seria eliminado e o processo seria reiniciado a partir do próximo soldado, até só restar o soldado ganhador.

   Entradas:
   - Nomes dos soldados que estão cercados
   - Uma das opções para a posição inicial da contagem:
     1. Iniciar contagem a partir do primeiro soldado da lista.
     2. Iniciar contagem a partir de uma posição sorteada aleatoriamente da lista.
     3. Iniciar contagem a partir de um soldado específico, cujo o nome deve ser dado pelo usuário.

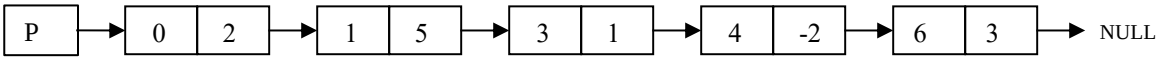
   Saídas:
   - No caso da opção de contagem (2), imprimir o nome do soldado sorteado.
   - Imprimir o número sorteado.
   - Imprimir os nomes dos soldados eliminados, na ordem de eliminação.
   - Imprimir o nome do Sobrevivente.

OBS: cabe ao aluno escolher a **MELHOR** forma de implementação para o problema. Tal escolha deve ser justificada através de linhas de comentário no início do arquivo do TAD.

8) Implementar um programa para manipulação de polinômios do tipo

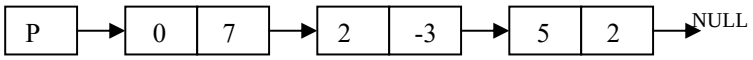
$$P(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x^1 + a_0$$

Para tal, o polinômio deve ser armazenado através de uma lista ordenada, sendo que cada elemento *i* da lista deve armazenar o *k*-ésimo termo do polinômio (diferente de 0), e deve conter o valor *k* da potência de x (inteiro) e o coeficiente a_k correspondente (inteiro). Por exemplo, o polinômio $P(x) = 3x^6 - 2x^4 + x^3 + 5x + 2$ deve ser representado pela lista:



Fica a critério do aluno a escolha da melhor forma de implementação, sendo que uma justificativa deve ser colocada na forma de comentário no início do arquivo do TAD. Deve ser criada um programa aplicativo que permita ao usuário selecionar repetidamente (exceto a 1ª opção) qualquer uma das operações a seguir:

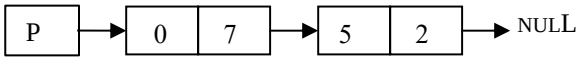
- **Inicializar um polinômio.** Fazer $P(x)= 0x^0$.
- **Inserir um novo termo $a_k x^k$ no polinômio existente.** Se já existe um termo $a_k' x^k$ no polinômio o valor do coeficiente do novo termo a_k deve ser adicionado ao já existente a_k', assim: $P(x) = a_n x^n + ...(a_k + a_k')x^k ... + a_1 x^1 + a_0$
- **Imprimir P(x).** Se o polinômio for $P(x) = 2x^5 - 3x^2 + 7$, a representação interna será:



A seguinte expressão deverá ser visualizada na tela: **+7 -3x^2 +2x^5**

- **Eliminar o termo associado à *k*-ésima potência**. Se o polinômio atual for $P(x) = 2x^5 - 3x^2 + 7$ (representação interna no exemplo acima) e o usuário solicitar a remoção do termo associado à potência 2 de x, o polinômio resultante será $P(x) = 2x^5 + 7$ e o nó referente à potência 2 de x deve ser liberado resultando na estrutura:

10)
11)



- **Reinicializar um polinômio.** Fazer $P(x)= 0x^0$ e liberar os nós do P(x) anterior.
- **Calcular o valor de P(x) para um valor de x solicitado**. Por exemplo, se o polinômio atual for $P(x)=3x^6 - 2x^4 + x^3 + 5x + 2$ e o usuário solicitar o cálculo de P(x) para x = 2, o valor de P(2) deve ser calculado: $P(2) = 3(2)^6 - 2(2)^4 + (2)^3 + 5(2) + 2 = 3\times64 - 2\times16 + 1\times8 + 5\times2 + 2 = 180$ e o valor 180 deve ser apresentado na tela.
- **Sair do sistema**.