# IMD0029 - Estrutura de Dados Básicas 1 – 2017.1 – Prova de Reposição
## Prof. Eiji Adachi M. Barbosa

Nome:_____

Matrícula: _____

**ANTES DE COMEÇAR A PROVA,** leia atentamente as seguintes instruções:

- Esta é uma prova escrita de caráter individual e sem consultas a pessoas ou material (impresso ou eletrônico).
- A prova vale 10,0 pontos e o valor de cada questão é informado no seu enunciado.
- **Preze por respostas legíveis, bem organizadas e simples.**
- As respostas devem ser fornecidas preferencialmente em caneta. Respostas fornecidas a lápis serão aceitas, mas eventuais questionamentos sobre a correção não serão aceitos.
- Celulares e outros dispositivos eletrônicos devem permanecer desligados durante toda a prova.
- Desvios éticos ou de honestidade levarão à anulação da prova do candidato (nota igual a zero).

**Questão 1 (2,5 pontos):** Considere que um array inicialmente ordenado foi deslocado à direita sem querer e não se sabe quantas vezes ele foi deslocado. Um deslocamento à direita de um array faz com que o último elemento do array torne-se o primeiro. Por exemplo: dado o array de entrada {1,2,3,4,5,6,7,8,9,10}, após 3 deslocamentos à direita sucessivos, o array estará da seguinte forma: {8,9,10,1,2,3,4,5,6,7}. Neste contexto, implemente uma solução **iterativa** que deverá obrigatoriamente ter complexidade **Θ (lg(n))**. Sua função deverá seguir a seguinte assinatura:

```
int countShift(int a[], int arraySize) – Retorna a quantidade de vezes que o array de entrada foi deslocado à direita.
```

**Questão 2 (2,5 pontos):** Implemente uma versão do algoritmo de ordenação por intercalação (*Merge Sort*) que divide o vetor de entrada em 3 partes. Sua solução deve contemplar também o algoritmo de intercalação (Merge). Suas funções deverão seguir as seguintes assinaturas:

```
void merge(int v[], int init1, int init2, int init3, int end3);

void mergesort(int v[], int left, int right);
```

Na função merge, os parâmetros `init1`, `init2` e `init3` indicam os índices em que se iniciam os três sub-arrays do array `v` a serem ordenados, e o parâmetro `end3` indica o índice em que termina o último sub-array de v. Assuma também que o fim dos outros dois sub-arrays estão localizados imediatamente antes do início do sub-array seguinte. Isto é, assuma que `end1 = init2-1`, e `end2 = init3-1`. Por fim, a função mergesort será invocada da seguinte maneira:

```
int main(void) {
      int v[N] = {/* Valores quaisquer */};
      mergesort(v, 0, N-1); // mergesort irá ordenar os elementos entre 0 e N-1
      return 0;
}
```

**Questão 3 (3,0 pontos):** Muitas aplicações acessam um mesmo dado em curtos períodos de tempo. Para este tipo de aplicação, uma estratégia para otimizar o acesso aos dados comumente implementada em listas encadeadas é a estratégia da contagem de acessos. Nesta estratégia, após cada busca realizada com sucesso na lista, os nós devem ser rearranjados de modo que fiquem ordenados em ordem decrescente em relação ao número de acesso dos nós. Neste contexto, dada uma lista duplamente encadeada com sentinelas cabeça (Head) e cauda (Tail), implemente uma solução **iterativa** para a seguinte função, considerando a estrutura abaixo:

```
Node List::SearchCount(int K) – Retorna o nó da lista cujo conteúdo ('content') tem valor igual a 'K'. Caso a busca tenha insucesso, retorna-se 'null'. Caso a busca tenha sucesso, a lista deverá ser reajustada de modo a manter os nós ordenados em ordem decrescente em relação ao contador.
```

```
List{
   Node* head;
   Node* tail;
}
```

```
Node {
   Node* next;
   Node* previous;
   int content;
   int accessCount;
}
```

Obs.: Nesta questão, assuma que não há qualquer operação sobre a lista encadeada a ser reusada. Você deverá implementar toda a solução.

**Questão 4 (2,0 pontos):** Na notação usual de expressões aritméticas, os operadores são escritos entre os operandos; por isso, a notação é chamada infixa. Na notação pós-fixa, os operadores são escritos depois dos operandos. Eis alguns exemplos de expressões infixas e correspondentes expressões pós-fixas:

| Infixa | Pós-fixa |
|---|---|
| (A+B*C) | ABC*+ |
| (A*(B+C)/D-E) | ABC+*D/E- |
| (A+B*(C-D*(E-F)-G*H)-I*3) | ABCDEF-*-GH*-*+I3*- |
| (A+B*C/D*E-F) | ABC*D/E*+F- |
| (A+B+C*D-E*F*G) | AB+CD*+EF*G*- |
| (A+(B-(C+(D-(E+F))))) | ABCDEF+-+-+ |
| (A*(B+(C*(D+(E*(F+G)))))) | ABCDEFG+*+*+* |

Note que os operandos (A, B, C, etc.) aparecem na mesma ordem na expressão infixa e na correspondente expressão pós-fixa. Note também que a notação pós-fixa dispensa parênteses e regras de precedência entre operadores (como a precedência de * sobre + por exemplo), que são indispensáveis na notação infixa. Nesta questão, implemente uma função que recebe como entrada um array de strings representando uma expressão aritmética em notação pós-fixa e retorna o seu valor numérico correspondente, usando obrigatoriamente uma estrutura **PILHA** já implementada. Considere que a expressão representada no array de entrada é uma expressão em notação pós-fixa válida e que cada string do array de entrada representa ou um inteiro, ou um dos símbolos para as quatro operações aritméticas (+ - * /). Sua função deverá seguir a assinatura abaixo:

```
/* exp – expressão aritmética em notação pós-fixa.
tamExp – quantidade de caracteres da string exp*/
int AvaliarExpressao( String exp, int tamExp );
```

Obs.: Assuma que a estrutura Pilha já está implementada e você pode reusar todas suas operações básicas.