Prova – P1

Atividade avaliativa
Estruturas de Dados 1 (1001502) - ENPE – Bloco B – 05/05 a 29/06 2021

Descrição
- Proponha solução para a prova abaixo, individualmente.

O que entregar
- **Documento único, formato DOC ou PDF**, contendo nome, RA, enunciado da prova e as soluções propostas.

Quando entregar
- A prova tem duração de 2 (duas) horas. Entregue até, no máximo, 2 horas após o início da prova.

Onde entregar
- No ambiente de interação da disciplina no Google Classroom, no link indicado.

Nome: Leticia Bossatto Marchezi RA: 791003 Data 26/05/2021

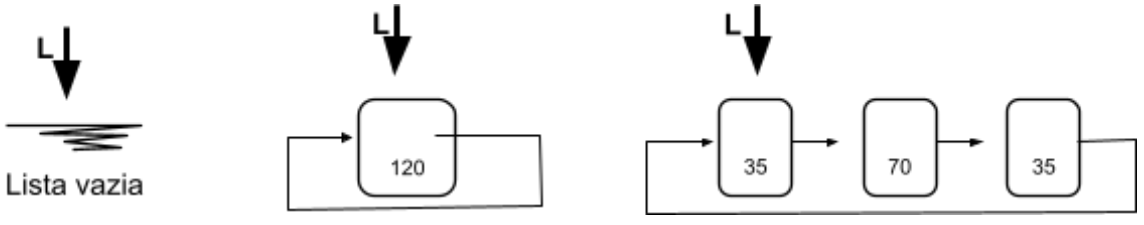
Orientações Gerais
- **Tempo para elaboração da prova: 2 horas**.

Orientações Quanto a Notação, Nomes das Variáveis, e Estruturas
- Use os **mesmos nomes fornecidos no enunciado** (L, F, X, etc.). Utilize variáveis auxiliares temporárias, o tanto quanto for necessário. É só declarar e usar. Mas **não considere a existência de nenhuma outra variável permanente**, além das definidas no enunciado. Não considere prontas para uso nenhuma operação, salvo se explicitamente indicado no enunciado da questão.
- Considere as **estruturas exatamente conforme definido no enunciado**, seja no texto da questão, seja nos diagramas.
- Para o desenvolvimento de algoritmos, use preferencialmente a notação adotada no Livro Texto: **p = NewNode; Deletenode(P), P->Info e P->Next,** sendo P uma variável do tipo **NodePtr** (ponteiro para nó). Quando a estrutura for duplamente encadeada, ao invés de P->Next considere que a notação contenha **P->Dir e P->Esq.** Também é possível implementar em C ou **C++**.

Questão 1 (4 pontos) Considere o Tipo Abstrato de Dado **Lista Cadastral** implementado através de uma **lista encadeada, circular, não ordenada, com elementos repetidos**, conforme os diagramas abaixo. **L** é um ponteiro para o início da lista. A lista vazia é composta pelo ponteiro L apontando para NULL. Implemente, **da forma mais apropriada para proporcionar portabilidade e reusabilidade**, as operações:

- **Remove-1** (variável por referência **L** do tipo NodePtr; variável **X** do tipo Inteiro, variável por referência Ok do tipo Boolean);
  /* remove uma única ocorrência do elemento de valor X da lista L. Qualquer uma das ocorrências de X. Caso nenhuma ocorrência de X for encontrada na lista L, o parâmetro Ok deve retornar FALSO. Ok deve retornar VERDADEIRO se uma ocorrência de X for encontrada e removida. Tipo NodePtr = ponteiro para Node */

- **Remove-Todos** (variável por referência **L** do tipo NodePtr; variável **X** do tipo Inteiro, variável por referência Ok do tipo Boolean);
  /* remove todas as ocorrências de valor X da lista L. Caso nenhuma ocorrência de X for encontrada na lista L, o parâmetro Ok deve retornar FALSO. Ok deve retornar VERDADEIRO se pelo menos uma ocorrência de X for encontrada e removida. Tipo NodePtr = ponteiro para Node */



```
void Remove-1(NodePtr *L, int X, bool *Ok){

    NodePtr anterior = L;

    NodePtr auxiliar;


    if(L==NULL){ // se a lista estiver vazia

        *Ok = false;

    }else if(L->Next==L){ // se a lista tiver 1 item

        if(L->Info==X){ // achou

            delete anterior;

              L = NULL; // o ponteiro de L aponta para o endereco NULL pois não
há mais elementos

            *Ok = true;

        }else{

            *Ok = false;

        }

    }else{ // 2 ou mais itens

        auxiliar=L->Next;

        if(anterior->Info==X){ // Caso o primeiro tenha a informação de X

            L=L->Next;

            while(auxiliar->Next!=anterior){ // busca o ultimo elemento

                auxiliar=auxiliar->Next;

            }

               auxiliar->Next=L; // troca o proximo do ultimo elemento para o
novo primeiro

            delete anterior;

            *Ok = true;

        }else{

                 while(auxiliar->Info!=X & & auxiliar->Next!=L){ // procura o
elemento com informação x na lista

                anterior=auxiliar;

                auxiliar=auxiliar->Next;

            }

            if(auxiliar->Info==X){ // deleta o elemento e troca o

                anterior->Next=auxiliar->Next;

                delete auxiliar;

                *Ok = true;

            }else{
```

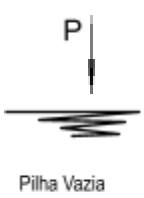
```
                *Ok = false;
            }
        }
    }
}
```

```
void RemoveTodos(NodePtr *L, int X, bool *Ok){
    bool *resultadoUm;
    NodePtr auxiliar = L;

    if(L==NULL){ // não é possível retirar elementos de uma lista
        *Ok = false;
    else if(L->Next == L){ // caso tenha um elemento
        Remove1(L, X, resultadoUm);
            if(*resultadoUm){ // seta Ok como true se encontrou o elemento
            *Ok=true;
        }
    }else{
        *Ok = false; // seta como falso por padrao
              if(L->Info==X){  // se o primeiro elemento tiver a informação igual a x
            Remove1(L, X, resultadoUm);
             if(*resultadoUm){ // caso a exclusao tenha ocorrido com sucesso, seta Ok como true
                *Ok=true;
            }
        }
        auxiliar=L->Next
        while(auxiliar!=L){ // varre a lista em busca de elementos
            if(auxiliar->Info==X){
                Remove1(L, X, resultadoUm);
                  if(*resultadoUm){ // caso pelo menos uma exclusao tenha ocorrido com sucesso, seta Ok como true
```

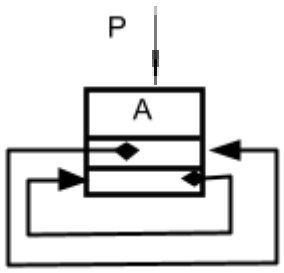
```
                    *Ok=true;

                }

            }

        }

    }

}
```

Questão 2 (3 pontos) Considere o Tipo Abstrato de Dados **Pilha** implementado através de uma **lista duplamente encadeada, circular,** segundo os diagramas abaixo. No diagrama à esquerda, a Pilha P está vazia. No diagrama ao centro a Pilha P contém um único elemento, e no diagrama mais à direita, o elemento de valor **D está no topo** (ordem de entrada A, B, C e D). Implemente, da forma mais apropriada para proporcionar portabilidade e reusabilidade, as operações:

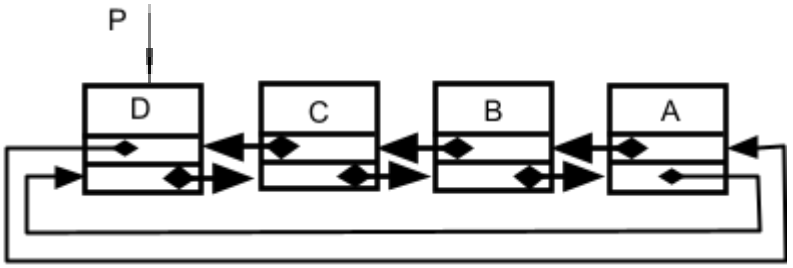
- **Empilha** (variável por referência P do tipo NodePtr, variável X do tipo Char);
  /* empilha o elemento X na Pilha P. Desconsidere a possibilidade de a pilha estar cheia. Tipo NodePtr = ponteiro para nó */



Pilha Vazia



Pilha com 1 elemento



Pilha com 4 elementos, e D está no topo (ordem de entrada: A, B, C e D)

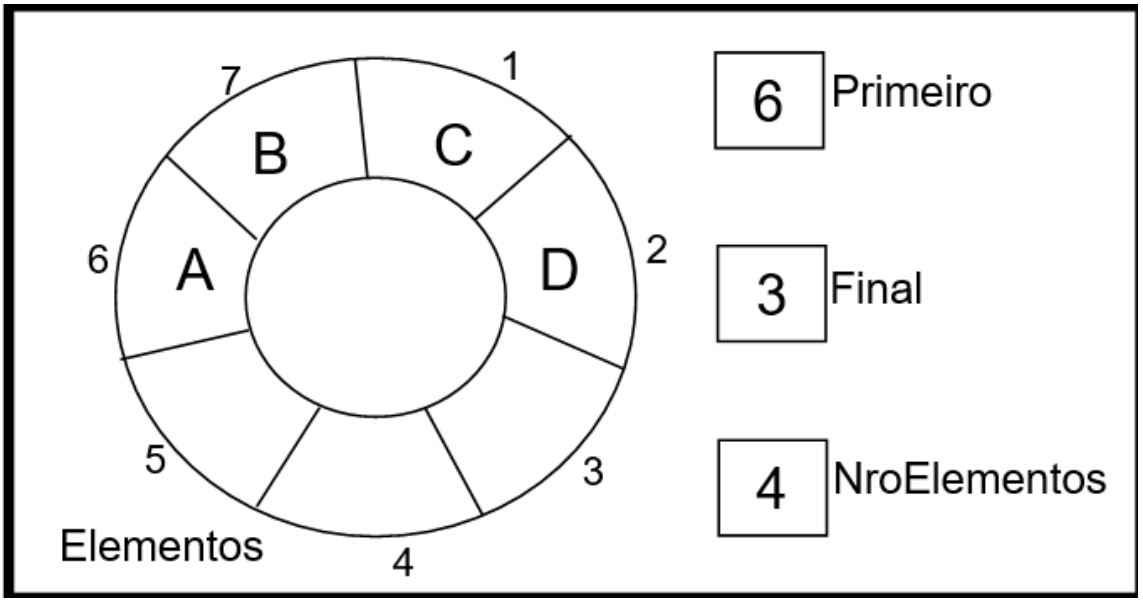
```
void Insere(NodePtr P, char X){
    NodePtr elemInserir = new Node;
    NodePtr auxiliar, antigo;

    elemInserir->Info=X;

    // Pilha vazia
    if(P==NULL){
        // o novo no vai ser o primeiro elemento e o proximo sera
ele mesmo
        elemInserir->Dir=elemInserir;
        elemInserir->Esq=elemInserir;
        P = elemInserir;

    }
    // Pilha com um elemento
    else if(P->Dir==P){
```

```
        // Insere em P
        elemInserir->Dir=P;
        elemInserir->Esq=P;

        P->Dir=elemInserir;
        P->Esq=elemInserir;

        P=elemInserir;

    // caso tenha 2 ou mais elementos
    }else{
        antigo = P;
        auxiliar = P->Dir;
        // adiciona o novo elemento
        P = elemInserir;
        // troca as referencias dos dois lados
        P->Dir=antigo;
        antigo->Esq=P;

        P=elemInserir;
        // varre ate achar o primeiro elemento
        while(auxiliar->Dir!=antigo){
            auxiliar=auxiliar->Dir;
        }
        // muda a referencia dos elementos
        auxiliar->Dir=P;
        P->Esq=auxiliar;
    }

}
```

Questão 3 (3 pontos) Considere o Tipo Abstrato de Dados **Fila** implementado com **alocação sequencial e estática**, com um **vetor circular,** sem realocação de elementos, segundo o diagrama abaixo. Implemente, da forma mais apropriada para proporcionar portabilidade e reusabilidade, a operação:

Insere (parâmetro por referência F do tipo Fila, parâmetro X do tipo Char, parâmetro por referência DeuCerto do tipo Boolean) {

/* Insere o elemento X na Fila F. O parâmetro DeuCerto deve indicar se a operação foi bem sucedida ou não. A operação só não será bem sucedida se tentarmos inserir um elemento em uma Fila cheia. Uma Fila F do tipo Fila é composta por 4 campos: o vetor F.Elementos, e os inteiros F.Primeiro, F.Final e F.NroDeElementos. F.Primeiro indica a posição do primeiro da Fila, F.Final indica a primeira posição vaga do vetor, e F.NroDeElementos indica a quantidade de elementos na fila F, em determinado momento. Considere que a constante TamanhoDoVetor, que indica o número de elementos que podem ser armazenados da Fila. Os elementos da Fila são do tipo Char. Considere ainda pronta para uso a operação Cheia(F), que verifica se uma fia F está cheia ou não. Ou seja, é possível utilizar, e não é necessário desenvolver a operação Cheia(F). */




```c
void Insere(Fila *F, char X, bool *DeuCerto){
    if(Cheia(F)){ // verificando se esta vazia
        *DeuCerto=false;
    }else{ // inserindo
        F->NroDeElementos=F->NroDeElementos+1;
        F->Elementos[F->Final]; // adicionando o elemento no final
        *DeuCerto=true;
        if(F->Final==F->NroDeElementos) {
            F->Final=1;
        }else{ // aumenta o contador do ultimo elemento
            F->Final=F->Final+1;
        }

    }
}
```