

F3 – Simulação de Prova – P1

Atividade avaliativa para cômputo de Frequência
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 contendo nome, RA, enunciado da prova e as soluções propostas.

Quando entregar

- Até 3 (três) dias antes da prova.

Onde entregar

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

Nome: Leticia Bossatto Marchezi

RA: 791003

Data: 23/05/2021

Orientações Gerais

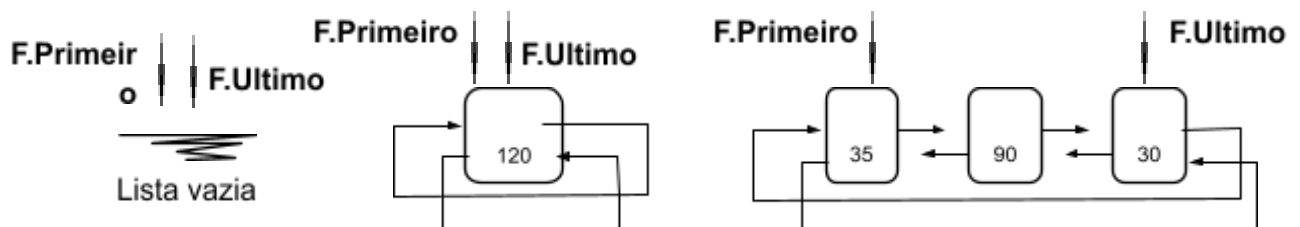
- Tempo para elaboração da prova: 2:30h (2 horas e 30 minutos).

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 (3 pontos) Considere o Tipo Abstrato de Dado **FILA**, implementado através de uma **lista duplamente encadeada, circular**, segundo os diagramas abaixo. Implemente, **da forma mais apropriada para proporcionar portabilidade e reusabilidade**, as operações Retira, Vazia e Destrói:

- **Boolean Vazia**(variável por referência F do tipo Fila)
// deve retornar verdadeiro se a fila não tiver nenhum elemento; falso caso contrário.
- **Retira**(variável por referência F do tipo Fila, variável por referência X do tipo Elemento, variável por referência Erro tipo boolean)
// retira 1 elemento da fila F. Erro deve retornar verdadeiro se a fila não tiver nenhum elemento para ser retirado; falso caso contrário.
- **Destrói**(variável por referência F do tipo Fila)
// desaloca (remove) todos os elementos da fila.



{o tipo Fila é um registro com 2 campos: F.Primeiro e F.Ultimo, ambos do tipo NodePtr (Ponteiro para nó) }

```

bool Vazia(Fila *lista){
    bool resultado;

    printf("prim:  %x      ult:  %x\n\n",  lista->FPrimeiro,
lista->FUltimo);

    if(lista->FPrimeiro==NULL && lista->FUltimo==NULL){

        resultado = true;
    }else{
        resultado = false;
    }
    return resultado;
}

```

```

void Retirar(Fila *lista, int *x, bool *Erro){
    Node *noAntigo;
    Node *trocaNovo;

    noAntigo = lista->FPrimeiro;
    printf("\n\nESQ:      %x      ELE MESMO:      %x\n\n",
(noAntigo->Esq), noAntigo);
    // Se a fila estiver vazia nao eh possivel retirar elemento
    // 0 ELEMENTOS
    if(noAntigo==NULL){
        cout<<"Nao ha elementos"<<endl <<endl;
        *Erro = true;

        // Fila com 1 elemento
        // Se o endereco do no a esquerda eh o proprio no, entao so ha
1 elemento
    }else if ((noAntigo->Esq)==(noAntigo)) {
        // Armazena o valor a ser excluido em x
        *x = noAntigo->info;

        // Retira as referências dos ponteiros pois nao ha mais
elementos

        lista->FPrimeiro=NULL;
        lista->FUltimo=NULL;

        delete noAntigo;
        noAntigo=NULL;
    }
}

```

```

        *Erro = false;
        cout<<noAntigo<<endl;
        cout<<"Excluido "<<*x<<". Sobrou 0 elementos"<<endl;
    }
    // 2 OU MAIS ELEMENTOS
    else{
        // Armazena o valor a ser excluido em x
        *x = noAntigo->info;
        printf("\n\n\n%d", *x);
        // Troca o primeiro elemento para o próximo a direita
        lista->FPrimeiro=noAntigo->Dir;
        // Troca o no a esquerda do novo 1° para o ultimo elemento
da fila
        trocaNovo=lista->FPrimeiro;
        trocaNovo->Esq=lista->FUltimo;

        // Troca o no da direita do último elemento para o novo 1°
        trocaNovo=lista->FUltimo;
        trocaNovo->Dir=lista->FPrimeiro;
        cout<<"Excluido "<<*x<<". Sobrou 1 elementos"<<endl;
        delete noAntigo;

        noAntigo=NULL;
        *Erro = false;
        cout<<noAntigo<<endl;
    }
}

```

```

void Destroi(Fila *lista){
    int ptr;
    bool erro;
    while(Vazia(lista)==0){
        Retirar(lista, &ptr, &erro);
    }

    delete lista;
    lista=NULL;
}

```

Questão 2 (5 pontos) Fila de Vacinação (FV). Um município está implementando uma **Fila de Vacinação**. O primeiro critério para entrar na Fila de Vacinação é a **idade**, ou seja: pessoas mais velhas entrarão na fila à frente de pessoas mais jovens que já estiverem na fila. O segundo critério da Fila de Vacinação é a ordem de chegada, ou seja: se já existem pessoas na fila com determinada idade, em anos, novas pessoas com essa mesma idade em anos devem ser inseridas na fila após as pessoas de mesma idade que entraram na fila primeiro. Ou seja, a Fila de Vacinação será uma "Fila de Prioridades", na qual deve ser priorizado o atendimento às pessoas de maior idade em anos. Uma possível solução a essa situação é implementar a Fila de Vacinação através de uma lista encadeada, circular, ordenada em ordem decrescente pela idade em anos, com elementos repetidos, conforme os diagramas abaixo.



Nos diagramas, **FV** é um ponteiro para o início da estrutura de dados. A estrutura vazia é composta pelo ponteiro FV apontando para NULL. Implemente a operação:

- **Inserir** (variável por referência **FV** do tipo FilaDeVacinação; variável **Idade** do tipo inteiro);

/* insere 1 pessoa com a idade fornecida como parâmetro na fila FV. Desconsidere a possibilidade de a estrutura estar cheia. Tipo FilaDeVacinação = um ponteiro do tipo NodePtr (ponteiro para nó). Deve-se inserir as pessoas mais velhas antes das mais jovens. Pessoas que tenham a mesma idade em anos, devem ser inseridas após as demais que já estão na fila */

Erros comuns:

- deixar de tratar um caso (inserir no meio, inserir no início, etc.) – descontar 1,5 pontos a cada caso não tratado;
- condição de repetição entrar em loop ou outro erro - descontar 1,5 pontos;
- problema com Nul (ex: acessar o valor apontado por um ponteiro cujo valor pode ser null) - descontar 1,5 pontos.

Roteiro para a solução:

- Identifique os casos a serem tratados;
- Desenhe cada caso a ser tratado;
- Faça um algoritmo para cada caso a ser tratado, e desenhe passo a passo, até a situação final;
- Faça um algoritmo geral o mais simples possível, abrangendo todos os casos.

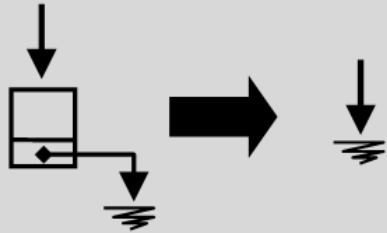
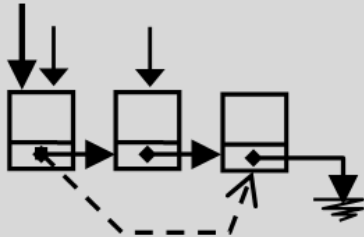
Passos para Construir uma Boa Solução. Livro Estruturas de Dados com Jogos, Capítulo 6:

Passos Para Construir uma Boa Solução

Passo 1: Identificar Casos e Desenhar a Situação Inicial. Identifique todos os casos, enumere como Caso 1, Caso 2, Caso 3, etc. Faça um desenho da situação inicial de cada caso; Pense sempre nas situações de lista vazia, lista com 1 único elemento, inserir ou eliminar no começo, no meio e no final da lista; encontrar ou não encontrar o elemento na lista, inserir o primeiro, retirar o único, e assim por diante.

Caso 1

Caso 2

<p>Passo 2: Desenho da Situação Final. Para cada caso identificado, faça um desenho também da situação final desejada, ou seja, de como o desenho deverá ficar após a execução da operação que você está projetando.</p>	
<p>Passo 3: Algoritmo para Tratar Cada Caso, Separadamente. Identifique para cada caso, o trecho de algoritmo necessário para levar o desenho da situação inicial para a situação final.</p>	<p>Para Tratar Caso 2:</p> <ul style="list-style-type: none"> • DeleteNode(P); • L=<u>Null</u>;
<p>Passo 4: Faça um Algoritmo Geral do Modo Mais Simples. Só faça o algoritmo geral após identificar todos os casos, desenhar a situação inicial e <u>final</u>, e após identificar os trechos de algoritmo necessários para tratar cada caso, individualmente. Não pense em fazer o algoritmo do modo "mais curto"; pense em fazer do modo "mais simples", tratando separadamente cada caso.</p>	<p>Algoritmo Geral</p> <p>Se Caso 1 Então [tratar Caso 1] Senão Se Caso 2 Então [tratar Caso 2] Senão Se Caso 3 Então [tratar Caso 3] Senão...</p>
<p>Passo 5: Testar Cada Caso, Alterando o Desenho Passo a Passo. Após elaborar o algoritmo completo, teste passo a passo, fazendo desenhos. Teste para todos os casos identificados. Para cada caso, parta da situação inicial, e execute o algoritmo. A cada comando, altere o desenho. Ao final da execução, verifique se o desenho chegou à situação final pretendida.</p>	

Resposta:

```
void Insere(FilaDeVacinao *FV, int Idade){
    NodePtr elemInserir = new Node;
    NodePtr auxiliar, antigo;
    int continuar=1;

    elemInserir->info=Idade;

    // Fila vazia
    if (FV==NULL) {
        // o novo no vai ser o primeiro elemento e o proximo sera
        ele mesmo
    }
}
```

```

        elemInserir->next=elemInserir;
        FV = elemInserir;

    }
    // Fila com um elemento
    else if (FV->next==FV) {
        // Insere depois do elemento existente se a idade for igual
ou menor
        if (Idade<=FV) {
            FV->next=elemInserir;
            elemInserir->next=FV;
        }
        // Insere como primeiro elemento se a idade for maior
    else{
        auxiliar = FV;
        FV=elemInserir;
        FV->next=auxiliar;
        auxiliar->next=FV;
    }
    // caso tenha 2 ou mais elementos
    }else{
        auxiliar=FV->next;
        antigo=FV;
        // tratando a mudanca caso a idade seja maior do que o
primeiro elemento
        // repete enquanto o campo atual for menor do que a idade a
ser inserida
        // e ate atingir o ultimo elemento
        while(auxiliar->info>=Idade && auxiliar!=FV && continuar){

            if(auxiliar==FV){ // tratar caso seja o ultimo elemento
da fila
                elemInserir->next=FV;
                antigo->next=elemInserir;
                continuar = 0;
            }
            else if (auxiliar->info<Idade){ // Se o proximo elemento
for menor do que a ser inserido
                elemInserir->next=aux;
                antigo->next=elemInserir;
                continuar = 0;
            }
        }
    }
}

```

```

    }

    // passa para o proximo no
    auxiliar=auxiliar->next;
    antigo=antigo->next;

}

if (FV->info<Idade){ // tratar caso seja maior do que o
elemento 1

    FV=elemInserir;
    FV->next=primAntigo;
    // busca ate o ultimo elemento para alterar o prox para
o novo FV

    while(auxiliar->next != primAntigo){
        auxiliar=auxiliar->next;
    }

    auxiliar->next=FV; // Atualiza o proximo do ultimo
elemento
}

}

}

```

Questão 3 (2 pontos) Considere uma **Lista Cadastral Sem Elementos Repetidos Implementada como uma Lista Encadeada Ordenada Circular**. Alguém implementou a operação **Retira** da forma apresentada a seguir.

Qual(is) dos seguintes casos indicados está(ão) sendo tratado(s) incorretamente na solução apresentada? Por que?

- Caso 1: Remover do meio da lista;
- Caso 1': Remover do final da lista;
- Caso 2: Remover o elemento que está no primeiro nó da lista, sendo que a lista possui vários elementos;
- Caso 3: Remover o elemento que está no primeiro nó da lista sendo que a lista contém um único elemento;
- Caso 4: Não achar X na lista;
- Caso 5: Lista vazia.

Solução a analisar:

Retira (parâmetro por referência L do tipo Lista, parâmetro X do tipo Char, parâmetro por referência Ok do tipo Boolean) {
 /* Caso X for encontrado na Lista, retira X da Lista e Ok retorna Verdadeiro. Se X não estiver na Lista, não retira, e Ok retorna Falso
 */

Variáveis P, Anterior do tipo NodePtr; // Tipo NodePtr = ponteiro para Nó
 Variável AchouX do tipo Boolean;

```
{
/* Passo 1 – Procura X */
Se (L != Null)
Então Se (L → Info == X) // achou X logo no primeiro Nó...
Então { // posiciona P no primeiro e Anterior no último nó da Lista
P = L;
Anterior = P → Next;
Enquanto (Anterior → Next != P) Faça {Anterior = Anterior → Next; }
} // fim então
Senão { P = L → Next; // começa P no próximo de L
Anterior = L; // Anterior começa em L

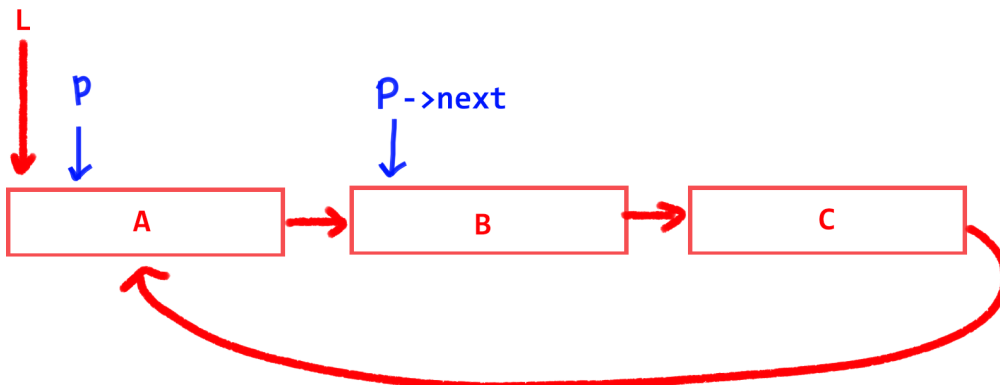
Enquanto ( P != L) e (P → Info < X) Faça {
Anterior = P;
P = P → Next; }
Se (P → Info != X)
Então AchouX = Falso;
Senão AchouX = Verdadeiro;
} // senão

Senão AchouX = Falso;
} // fim ProcuraX

/* Passo 2: se encontrou X, remover da Lista o Nó que contém X */
Se (AchouX == Verdadeiro)

Então { Se (P == L)
Então L = L → Next;
Anterior → Next = P → Next;
DeleteNode( P ); }
Ok = Verdadeiro;
P = Null;
Anterior = Null;
}
Senão Ok = Falso;
} // fim Retira
```

O caso 1 está sendo tratado inadequadamente pois o ponteiro Anterior está na posição errada.



P->next não aponta para o último elemento da lista, mas para o próximo. Já no caso 1', o teste irá falhar em Se (P == L) , pois P estará numa posição posterior a L e não na mesma, logo, o bloco conseguinte não será executado e o nó não será excluído.