Nome: Matheus Goulart Ranzani
RA: 800278

Prova 1

Questão 1

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.

```
void remove_1(NodePtr *L, int x, int *ok) {
    // Verificação se a lista está vazia
    if (*L == NULL) {
        ok = 0;
        return;
    }

    NodePtr p = *L;
    NodePtr anterior = NULL;

    while (p != NULL) {
        if (p->info == x) {
            if (anterior == NULL) { // Começo da lista
                *L = (*L)->next;
                free(p);
            } else if (p->next == NULL) { // Fim da lista
                anterior->next = NULL;
                free(p);
            } else { // Meio da lista
                anterior->next = p->next;
                free(p);
            }

            *ok = 1;
            return;
        } else {
            anterior = p;
            p = p->next;
        }
    }

    *ok = 0;
}
```

Remove-Todos (variável por referência L do tipo NodePtr; variável X do tipo Inteiro, variável por referência Ok 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_todos(NodePtr *L, int X, int *ok) {
    *ok = 1;

    while (*ok) {
        remove_1(L, X, ok);
    }
}
```

Questão 2

Boolean Vazia(variável por referência P do tipo Pilha) // retorna True se a Pilha não tiver nenhum elemento; falso caso contrário.

```
int vazia(Pilha *P) {
    if (P->header->esq == P->header) {
        return 1;
    } else {
        return 0;
    }
}
```

Empilha (variável por referência P do tipo Pilha, variável X do tipo Elemento) /* insere 1 elemento na Pilha P. Desconsidere a possibilidade de a Pilha estar cheia */

```
void empilha(Pilha *P, Elemento X) {
    NodePtr p = malloc(sizeof(Node));

    p->info = X;
    p->dir = P->header->dir;
    p->esq = P->header;

    P->header->dir->esq = p;
    P->header->dir = p;
}
```

Questão 3

Int NroPessoasVacinadasOuInfectadas(Lista L1, Lista L2);
// Calcula e retorna o número de pessoas que foram vacinadas (L1), ou infectadas (L2) ou ambos.
// Considere que o elemento das listas seja do tipo “Pessoa”.

```
int NroPessoasVacinadasOuInfectadas(Lista L1, Lista L2) {
    int total_L1 = 0;
    int total_L2 = 0;
    int total_repetidos = 0;
    int tem_elemento = 1;
    int resultado;

    Pessoa p;

    PegaOPrimeiro(L1, X, tem_elemento);
    while (tem_elemento) {
        total_L1++;
        PegaOProximo(L1, X, tem_elemento);
    }

    PegaOPrimeiro(L2, X, tem_elemento);
    while (tem_elemento) {
        total_L2++;
        PegaOProximo(L2, X, tem_elemento);
    }

    PegaOPrimeiro(L1, X, tem_elemento);
    while (tem_elemento) {
        if (EstaNaLista(L2, X)) {
            total_repetidos++;
        }
        PegaOProximo(L1, X, tem_elemento);
    }

    resultado = (total_L1 + total_L2) - total_repetidos;

    return resultado;
}
```