Atenção:

OBS.1: Todas as implementações devem ser feitas em **linguagem C**.

OBS.2: Leia atentamente o enunciado de cada questão. Caso a estrutura e/ou função implementada não esteja de acordo com a solicitada no enunciado, a questão receberá nota **ZERO**.

OBS.3: A eficiência de cada função implementada será avaliada, ou seja, caso não seja a mais eficiente possível será descontado ponto da questão.

OBS.4: Caso seja identificado cola, todos os envolvidos receberão nota **ZERO**.

Questão 1: Sobre listas simplesmente encadeadas com as seguintes características: (i) são circulares, (ii) não possuem nó cabeça, (iii) não possuem nó sentinela, (iv) podem conter números repetidos, e (v) as chaves (do tipo inteiro) da lista estão **sempre em ordem crescente**. Implemente o que se pede.

0,5 a) *(0,5 ponto)* Implemente a estrutura de um nó da lista.

0,5 b) *(0,5 ponto)* Implemente a função para alocar e inicializar um nó da lista com um número fornecido como parâmetro da função.

1,2 c) (2,0 pontos) Implemente a função que insere um número na lista. Todas as funções chamadas pela sua função (que não pertencem a nenhuma biblioteca padrão da linguagem C) devem ser implementadas.

Questão 2: Podemos representar polinômios utilizando listas duplamente encadeadas com nó cabeça, cujos nós armazenam 2 dados: coeficiente e expoente. Tais listas são ordenadas decrescentemente de acordo com o expoente. Posto isso, implemente o que se pede.

0,5 a) (0,5 ponto) Implemente todas as estruturas necessárias para armazenar um polinômio utilizando uma lista duplamente encadeada com nó cabeça.

0,5 b) (0,5 ponto) Implemente uma função que cria e inicializa um nó da lista (estrutura implementada no item anterior).

2,7 c) (3,0 pontos) Implemente uma função que recebe dois polinômios A e B (cada um em sua respectiva lista) e cria uma terceira lista C resultante da operação $A+B$. Todas as funções chamadas pela sua função (que não pertencem a nenhuma biblioteca padrão da linguagem C) devem ser implementadas.

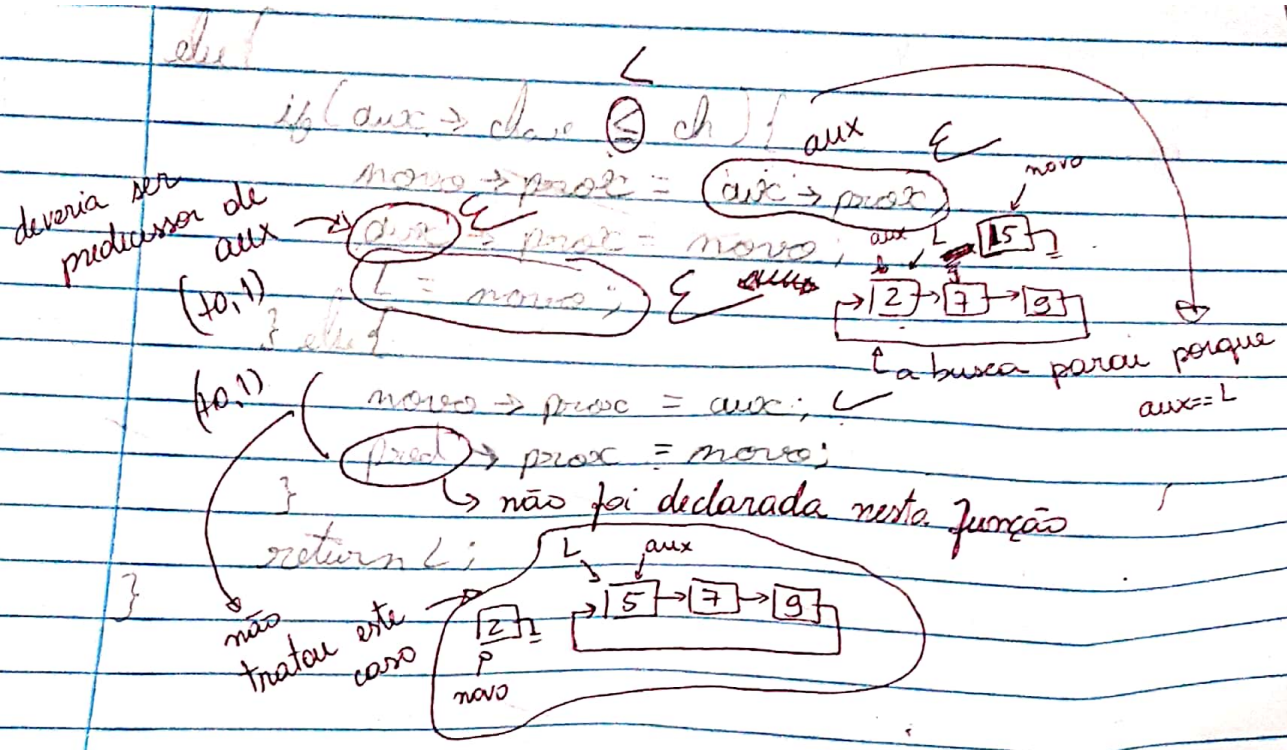
2,7 d) (3,0 pontos) Implemente uma função que recebe dois polinômios A e B (cada um em sua respectiva lista) e cria uma terceira lista D resultante da operação $A-B$. Todas as funções chamadas pela sua função (que não pertencem a nenhuma biblioteca padrão da linguagem C) devem ser implementadas. **Observação:** monômios com coeficiente igual a zero não devem ser armazenados na lista.

Questão 1

```
#include <stdlib.h>
#include <stdio.h>
typedef struct no{
    int chave;
    struct no *prox;
} No;

No *criaNo(int ch){
    No *novo = (No*) malloc(sizeof(No));
    if(novo == NULL){
        printf("Falta de alocação");
        exit(1);
    }
    novo->chave = ch;
    novo->prox = NULL;
    return novo;
}

No *Busca(No *L, int ch){
    No *aux = L, *pred = NULL;
    if(L == NULL) return NULL;
    do{ pred = aux;
        aux = aux->prox;
    }while(aux != L && aux->chave < ch);
    return aux;
}

No *insOrdenada(No *L, int ch){
    No *novo = criaNo(ch);
    No *aux = Busca(L, ch);
    if(L == NULL){ L = novo;
        novo->prox = L; }
```

(+0,4) (+0,2) (+0,1) (+0,3)

coeficiente igual a zero não devem ser armazenados na lista.



Questão 2

```
#include <STDIO.h>
#include <STDLIB.h>
typedef struct no{
    int coe, expo;
    struct no *prox, *ant;
}No;

No *criaNo(int coeficiente, int expoente){
    No *novo = (No*) malloc(sizeof(No));
    if(novo == NULL){
        printf("Erro de alocação");
        exit(1);
    }
    novo → coe = coeficiente;
    novo → expo = expoente;
    novo → prox = NULL;
```

```
novo->ant = NULL;
return novo;
}
No* Busca(No* L, int cl){
    No* aux = L->prox;
    while(aux != L && aux->expo >= cl){
        if(aux->prox == L) break;
        aux = aux->prox;
    }
    return aux;
}

No* insereBalanceada(No* L, int coeficiente, int expoente){
    No* aux1 = L;   -> não usou.
    No* novo = criaNo(coeficiente, expoente);
    No* aux = Busca(L, expoente);
(0,1) if(L->prox == NULL){   <- somente se a lista não for circular.
        L->prox = novo;
        novo->ant = L;   <- somente se a lista for circular
    }else{
        if(aux->expo > expoente){
            novo->ant = aux;
            aux->prox = novo;
        }
(0,1) else if(aux->ant == L){   <- it?
            aux->ant->prox = novo;
            novo->ant = aux->ant;
            novo->prox = aux;
            aux->ant = novo;
        }
    return L;
}
```

```
No *Soma (No *L1, No *L2){
    No * aux1 = L1->prox;
    No * aux2 = L2->prox;
    No * aux3 = NULL;

    While (aux1 != L1){          // lista circular
        int q = 0;
        While(aux2 != L2){       // lista circular.
            if (aux1->expo == aux2->expo){
            aux3 = insereOrdenada(aux3, (aux1->coe + aux2->coe),
            aux1->expo);
            q++;
            }
            aux2 = aux2->prox;
        }
        if (q == 0){
            aux3 = insereOrdenada(aux3, aux1->coe,
            aux1->expo);
            }
        aux1 = aux1->prox;
        aux2 = L2->prox;}
    }
    aux1 = L1->prox;
    aux2 = L2->prox;

    While(aux2 != L2){
        int q = 0;
        While (aux1 != L1){
            if(aux2->expo == aux1->expo){
            q++;}
(-0,1)      aux1 = aux1->prox;}
```

Continuação: Q2.

```
if(q == 0){
    aux3 = intercalada(aux3, aux2->coe, aux2->expo);
}
aux2 = L2->prox;
aux1 = aux1->prox;
}
return aux3;}

No* Subtracao(No* L1, No* L2){
(-0,2) No* aux1 = L1->prox?
       No* aux2 = L2->prox?
       No* aux3 = NULL;

while(aux1 != L1){
    while(aux2 != L2){
        if(aux1->expo == aux2->expo){
        aux3 = intercalada(aux3, (aux1->coe - aux2->coe), aux1->expo);
        aux2 = aux2->prox;
        }
    aux2 = L2->prox;
    aux1 = aux1->prox;}
    aux1 = L1->prox;

while(aux2 != L2){
    int q = 0;
    while(aux1 != L1){
    if(aux2->expo == aux1->expo){
```

q++; } —

auc2 = auc2 → proxc; } —

if(q==0){

auc3 = anseraOrdenada(auc3, -(auc1 → coe), auc2 →

expo); } —

auc①^{1} = ②^{1} → proxc; —

auc①$_{2}$ = auc①$_{2}$ → proxc; }

return auc3; }

(0,1)