Aviso

Todas as questões deste conjunto devem utilizar alocação dinâmica para vetores e matrizes. Em geral, vocês deverão escolher entre ***malloc*** e ***calloc*** conforme a necessidade de inicialização da memória. Quando a questão exigir uma realocação, o uso de ***realloc*** será explicitamente informado. O preenchimento dos vetores, matrizes e variáveis pode ser feito livremente, seja de forma aleatória ou por entrada do usuário.

Sigam os protótipos de função fornecidos, podendo aumentar ou reduzir a quantidade de parâmetros e criar outras funções conforme a necessidade do problema. Lembrem-se de seguir as boas práticas de alocação e liberação de memória.

---

1. Dado um vetor nums contendo n números distintos no intervalo [0, n], retorne o único número no intervalo que está faltando no vetor.

   int numeroAusente(int *nums, int numsSize);

   Exemplo:

   Entrada: nums = [3, 0, 1]

   Saida: 2

2. Dado um array de números inteiros arr, retorne true se a quantidade de vezes que cada número aparece for única — ou seja, sem repetições entre as contagens. Caso contrário, retorne false.

   bool ocorrenciasUnicas(int * arr, int arrSize);

   Exemplo:

   Entrada: arr = [1,2,2,1,1,3]

   Saida: true

   Explicação:

   O valor 1 tem 3 ocorrências, 2 tem 2 e 3 tem 1. Não há dois valores o mesmo número de ocorrências.

3. Inicialmente, o vetor deverá ser preenchido. Em seguida, o programa deverá solicitar um valor alvo (target) e determinar a posição deste valor no vetor, obedecendo as seguintes propriedades:
   a. Se o valor alvo estiver presente no vetor, retorne o índice correspondente.
   b. Caso o valor não esteja presente, retorne o índice em que o número deveria ser inserido para manter a ordem crescente do vetor.

int searchInsert(int* nums, int numsSize, int target);

Exemplo 1: **Entrada**: nums = [1,3,5,6], alvo = 5 **Saida**: 2

Exemplo 2: **Entrada**: nums = [1,3,5,6], alvo = 7 **Saida**: 4

(**A solução deve tratar array não ordenados**)

4. Dadas dois vetores inteiros, nums1 e nums2, retorne o **menor número inteiro que seja comum** às duas. Ou seja, o menor valor que apareça **pelo menos uma vez em cada uma das duas matrizes**.
   Se não houver nenhum número inteiro comum entre nums1 e nums2, retorne -1.

   int elementMin(int *nums, int nums1Size, int *nums2, int nums2Size);

   (O vetor deve ser construído dinamicamente utilizando realloc, pois o número total de elementos não é conhecido previamente. Isso simula alocação sob demanda. Use sua criatividade para decidir como e quando encerrar a leitura dos dados.)

5. Dada uma matriz grid de dimensões MxN, retorne à quantidade de números negativos presentes na matriz.

   Int countNegatives(int **grid, int m, int n);

6. Dada um vetor arr de números inteiros, verifique se existem dois índices i e j tais que:
   a. i != j
   b. 0 <= i, j < arrSize
   c. arr[i] == 2 * arr[j]
   d. Se tal propriedade não existir retorne: Não há I e J que satisfaçam as condições

   bool checkifExist(int *arr, int arrSize);

   Exemplo:

   Entrada: arr = [10,2,5,3]
   Saída: true
   Explicação: para i = 0 e j = 2, arr [i] == 10 == 2 * 5 == 2 * arr [j]

(O vetor deve ser construído dinamicamente utilizando realloc, pois o número total de elementos não é conhecido previamente. Isso simula alocação sob demanda. Use sua criatividade para decidir como e quando encerrar a leitura dos dados.)

7. Dado um vetor de inteiros nums, remova os elementos duplicados **no próprio vetor**, mantendo a ordem original. A função deve retornar o **ponteiro para o vetor modificado**, contendo os elementos únicos nas primeiras posições, e atualizar o valor apontado por numsSize com a nova quantidade de elementos únicos.
   int* removeDuplicates(int* nums, int* numsSize);

   Exemplo:
   Entrada: nums = [0,0,1,1,1,2,2,3,3,4]
   Saida: nums = [0,1,2,3,4,_,_,_,_,_]
8. **Lucas**, está enfrentando o chefe final no seu videogame favorito. O chefe possui **h** de vida. Seu personagem possui **n** ataques. O **i-ésimo** ataque causa $\mathbf{a_i}$ de dano ao chefe, mas possui um tempo de recarga de $\mathbf{c_i}$ turnos, o que significa que a próxima vez que você poderá usá-lo será no turno $\mathbf{x + c_i}$, se o turno atual for **x**.
   A cada turno, você pode usar **todos os ataques que não estão em recarga, todos de uma vez**. Se todos os ataques estiverem em recarga, você **não faz nada no turno** e apenas avança para o próximo. Inicialmente, **todos os ataques estão disponíveis** (sem cooldown). **Quantos turnos você, Lucas, levará para derrotar o chefe?** O chefe é derrotado quando sua vida for **0 ou menos**.
   (***Desenvolva uma struct que se encaixe no problema, e utilize ela para tal***)

   Exemplo:
   H = 5, N = 2
   Ai = [2, 1] => dano de seus ataques.
   Ci = [2, 1] => o cooldown de seus ataques.
   Saida: 3
   Explicação:

   Turno 1: Use os ataques **1** e **2**, causando **3 de dano** ao chefe. O chefe agora tem **2 de vida** restante.

   Turno 2: Use o **ataque 2**, causando **1 de dano** ao chefe. O chefe agora tem **1 de vida** restante.

Turno 3: Use o **ataque 1**, causando **2 de dano** ao chefe. O chefe agora tem **−1 de vida**. Como a vida dele é menor ou igual a 0, **você derrotou o chefe**.

9. Emanuel tem um número *A*, que ele deseja se transformar em um número B. Para esse fim, ele pode fazer dois tipos de operações:
    a. Multiplique o número atual por 2 (ou seja, substitua o número *x* por 2 * x);
    b. Anexe o dígito 1 à direita do número atual (ou seja, substitua o número *x* por 10 * *x* + 1).

    Você precisa ajudar Emanuel para transformar o número *A* no número *B* usando apenas as operações descritas acima ou descobrir que é impossível.

    (***Utilize o realloc, para o vetor crescer dinamicamente, sem desperdícios de memória***)

    Exemplo 1:

    Entrada: A = 2 B = 162

    Saida:

    Sim
    2 4 8 81 162

    Exemplo 2:

    Entrada: A = 100 B = 40021

    Saida:

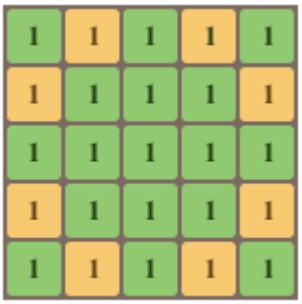
    Sim
    100 200 2001 4002 40021

10. Lucas Emanuel é viciado em matrizes quadradas. Agora ele está ocupado estudando uma matriz *n x n, onde n* é ímpar. Lucas considera os seguintes elementos da matriz bom:
    a. Elementos da diagonal principal.
    b. Elementos da diagonal secundária.
    c. Elementos da linha central
    d. Elementos da coluna central

    Vale – se a seguinte propriedade algo é central se possui (n - 1) / 2 elementos na primeira metade e (n – 1) / 2 na segunda metade. ***Ajude Lucas a descobrir a soma dos elementos considerados bons.***

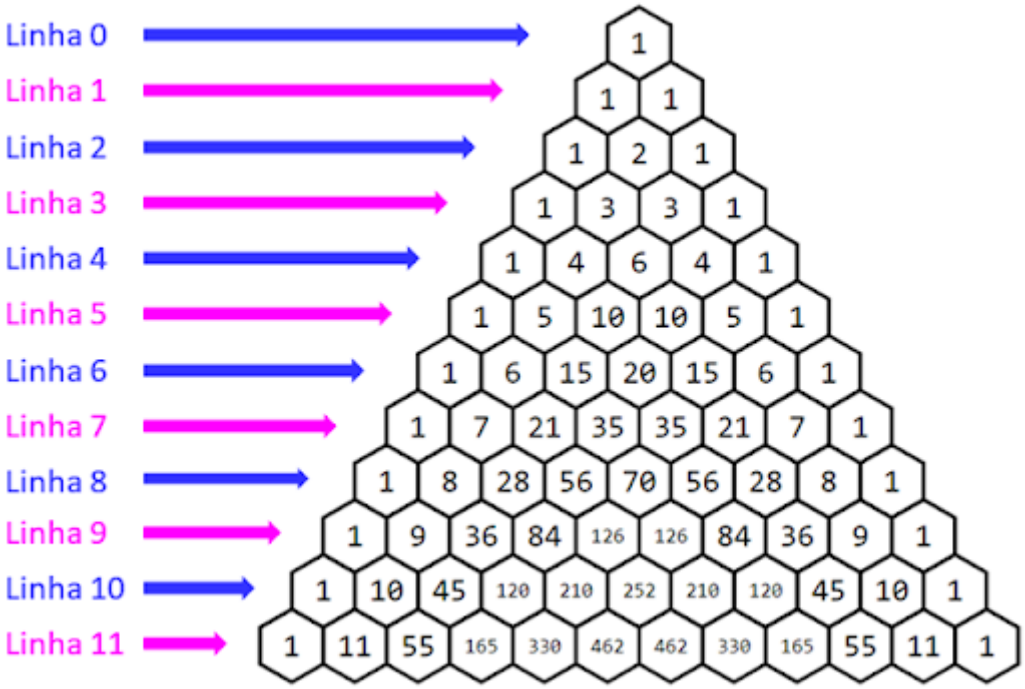
    Exemplo:

    Entrada: n = 5



Saida: 17

11. Dado um número inteiro N retorne a linha de índice (começando do zero) do Triângulo de Pascal. No Triângulo de Pascal, cada número é a soma dos dois números acima dele.



Dica: (Utilize Matriz)

Exemplo

Entrada: N = 3

Saida: [1, 3, 3, 1]

12. Gere strings binários sem zeros adjacentes, de tamanho N, ou seja, essa string não deve possuir dois ‘00’ consecutivos como substrings.

char ** validstrings(int n, int *returnSize);

Exemplo

Entrada: N = 3

Saida:

["010", "011", "101", "110", "111"]

Problema Final – Implementação de Vetor Dinâmico

Embora essa seja uma questão com cara de "TAD" (*Tipo Abstrato de Dados*), o grande objetivo aqui é treinar **alocação dinâmica** na prática, trabalhando com struct, malloc, realloc, e funções bem-organizadas. O problema é implementar um vetor dinâmico de inteiros com operações básicas, simulando uma estrutura parecida com o vector da C++ STL. (***Os protótipos podem ser adaptados conforme necessário durante a implementação.***)

```
Typedef struct {
        int *dados;
        int tamanho;
        int capacidade;
} Vector;
```

Implemente as seguintes funções:

Parte 1 -

```
Vector* criarVector(int capacidadeInicial);
int push_back(Vector* v, int elemento);
int push(Vector *v, int elemento, int index);
```

Parte 2 -

```
int pop(Vector *v, int index);
int pop_Back(Vector* v);
```

Parte 3 -

```
Int size(Vector *v);
void free_vector(Vector *v);
void clear(Vector *v);
```