Questões MALLOC:

1. Escreva uma função que receba um byte `c` (que pode representar um caractere ASCII, por exemplo) e transforme-o em uma string, ou seja, devolva uma string de comprimento 1 tendo `c` como único elemento.

```c
#include <stdlib.h>

char* converter_byte_para_string(char c) {
    char *string_resultante = malloc(2 * sizeof(char));
    if (string_resultante == NULL) {
        fprintf(stderr, "Erro: Falha na alocação de memória.\n");
        return NULL;
    }
    string_resultante[0] = c;
    string_resultante[1] = '\0';
    return string_resultante;
}
```

2. Discuta, passo a passo, o efeito do seguinte fragmento de código:

```
int *v;
v = malloc (10 * sizeof (int));
```

R: Ocorre a partir do fato de que de acordo com a variável criada, a cabem 10 bytes de tamanho da estrutura da int enviada. **Parcialmente correta!.** Cada int , com 32 bits ou 64 bits possuem 4 bytes... quando você coloca 10*sizeof, você multiplica por 10 a quantidade de bytes que você tem, levando ao máximo de 40 bytes. "A partir do fato de que, de acordo com a variável v (um ponteiro para inteiro), a chamada a `malloc` aloca um bloco de memória na heap contíguo, grande o suficiente para armazenar **10 valores do tipo `int`**. O tamanho total em bytes é `10 * sizeof(int)` (por exemplo, 40 bytes se `sizeof(int)` for 4)."

3. Discuta o efeito do seguinte fragmento de código:

```
x = malloc (10 * sizeof *x);
```

R: já está sendo inserido o ponteiro de X no local do (type) para que não seja necessário criar uma int/char/... *x, favorecendo a diminuição da linha de códigos... Em resumo, a linha `x = malloc(10 * sizeof *x);` **aloca dinamicamente na heap um bloco de**

memória contíguo grande o suficiente para armazenar 10 elementos do tipo para o qual x aponta. O ponteiro x passa a conter o endereço de início desse bloco de memória. A sintaxe `sizeof *x` é uma prática recomendada por tornar o código mais flexível e robusto em relação a mudanças no tipo de dados.

Sobre a função FREE:

1. O que há de errado com o seguinte fragmento de código?

```
int *v;
v = malloc (100 * sizeof (int));
v[0] = 999;
free (v+1);
```

R: A ideia de vetor e arg int? E a somatória no free? ...

Resposta correta: como corrigir o código.

Para liberar corretamente a memória, você deve passar o ponteiro v (que guarda o endereço inicial da alocação) para `free`:

C

```
#include <stdio.h>
#include <stdlib.h> // Para usar malloc e free

int main() {
    int *v;
    v = malloc(100 * sizeof(int));

    if (v == NULL) { // Sempre verifique se a alocação foi bem-
sucedida
        perror("Erro ao alocar memória");
        return 1;
    }

    v[0] = 999;

    // ... (uso do array v) ...
```

```
    free(v); // CORRETO: Libera o bloco de memória completo a partir
do endereço inicial
    v = NULL; // Boa prática: zera o ponteiro após liberar a memória
para evitar "dangling pointers"

    return 0;
}
```

OBS: **Em Resumo**

O C trata v como um **ponteiro** que aponta para o início de uma sequência de inteiros. A conveniência de usar a notação de colchetes ([ ]) com esse ponteiro é o que nos permite manipulá-lo como um array, mesmo que o compilador não "saiba" o "tamanho" do array em si uma vez que ele foi alocado. Quando você aloca memória dinamicamente usando `malloc` (ou `calloc`) e especifica um tamanho que é um múltiplo do tamanho de um tipo de dado (como `100 * sizeof(int)`), você está, na prática, criando um **bloco contíguo de memória** que pode ser tratado como um **vetor (array)**.

3. Discuta, passo a passo, o efeito do seguinte fragmento de código:

```
int *p, *q;
p = malloc (sizeof (int));
*p = 123;
q = malloc (sizeof (int));
*q = *p;
q = p;
free (p);
free (q); // má ideia...
q = NULL; // boa ideia
```

Problema: O bloco de memória em `0x2000` não pode mais ser acessado e se tornou um **vazamento de memória (memory leak)**. A memória foi alocada, mas a referência a ela foi perdida antes que pudesse ser liberada.

p e q estão relacionados a “espaço de memória”... ou seja, quem é quem em cada espaço. O que conta quem é na verdade é o *p e *q... ou seja, eles são os valores de fato e não o alocamento de memória deles...